Жава хэлний
сурах бичиг

Агуулга

Обьект хандалтат хөтөлбөрлөлийн ухагдахуунууд....5

ОБЬЕКТ ХАНДАЛТАТ ХӨТӨЛБӨРЛӨЛИЙН УХАГДАХУУНУУД....5

Тусагдахуун гэж юу вэ?....5
Мэдээ гэж юу вэ?....7
Анги гэж юу вэ?....8
Удамшил гэж юу вэ?....9
Үүд гэж юу вэ?....11
Эдгээр ухагдахуунууд нь хэрхэн код руу хөрвүүлэгдэх вэ?....11
Асуулт болон дасгал....19
Хэлний үндсүүд....20

ХЭЛНИЙ ҮНДСҮҮД....20

Хувьсагч....20
Өгөгдлийн төрлүүд....22
Хувьсагчийн нэр....23
Цар хүрээ....24
Хувьсагч цэнэглэх....25
Төгс хувьсагч....26
Дүгнэлт....26
Асуулт болон дасгал....27
Оператор....27
Арифметик оператор....28
Харьцуулах болон болзолт оператор....32
Шилжүүлэх болон логик оператор....35
Утга олгох оператор....39
Бусад оператор....41
Дүгнэлт....41
Асуулт болон дасгал....45
Илэрхийллүүд, хэллэгүүд, мөн блокууд....46
Асуулт болон дасгал....50
Хэллэгүүдийн удирдлагын урсгал....50
while болон do – while хэллэгүүд....51
for хэллэг....53
if / else хэллэгүүд....55
switch хэллэг....57
Гажуудал засах хэллэгүүд....60
Салаалуулах хэллэгүүд....61
Дүгнэлт....64
Асуулт болон дасгал....67
Объектын үндэс ба өгөгдлийн энгийн объектууд....69

ОБЪЕКТЫН ҮНДЭС БА ӨГӨГДЛИЙН ЭНГИЙН ОБЪЕКТУУД....69

Объект амьдралын мөчлөг....69
Объектыг үүсгэх нь....70
Объектуудыг хэрэглэх....74
Хэрэгцээгүй объектыг цэвэрлэх....78
Дүгнэлт....78
Асуулт болон дасгал....79
Тэмдэгтүүд ба Хэлхээнүүд....80
Тэмдэгтүүд....80
Хэлхээ болон хэлхээн буфер....82
Хэлхээ ба хэлхээн буферыг үүсгэх....83
Хэлхээн болон хэлхээн буферын уртыг авах....85
Хэлхээ болон хэлхээн буфераас тэмдэгтийг индексээр авах....86
Хэлхээн дотроос дэд хэлхээ юмуу тэмдэгт хайх....87
Хэлхээ болон хэлхээний хэсгүүдийг харьцуулах....89
Тэмдэгт хэлхээтэй ажиллах....90
Хэлхээн буферыг өөрчлөх....92
Хэлхээ болон эмхэтгүүр....93
Дүгнэлт....94

Асуулт болон дасгал.....*95*
Тоонууд.....96
Тооны ангиуд.....*96*
Хэлхээг тоон төрөл рүү хөрвүүлэх.....*98*
Тоон төрлийг хэлхээ рүү хөрвүүлэх.....*99*
Тоог форматлах.....*100*
Тоог өөрийнхөөрөө форматлах.....*102*
Үндсэн арифметикаас цааших.....*104*
Дүгнэлт.....*108*
Асуулт болон дасгал.....*109*
Бүл.....109
Бүлийг үүсгэх болон ашиглах.....*110*
Объектуудын бүлүүд.....*113*
Бүлүүдийн бүлүүд.....*114*
Бүлийг хуулбарлах.....*115*
Дүгнэлт.....*118*
Асуулт болон дасгал.....*118*
Ангиуд болон удамшил.....119

АНГИУД БОЛОН УДАМШИЛ.....119

Ангиудыг үүсгэх.....119
Анги зарлах.....*120*
Гишүүн хувьсагчийг зарлах.....*121*
Арга тодорхойлох.....*122*
Ангийн байгуулагч бичих.....*125*
Арга болон байгуулагч руу мэдээлэл дамжуулах.....*126*
Аргаас утга буцаах.....*130*
This түлхүүр үг ашиглах.....*131*
Ангийн гишүүд рүү хандах хандалтыг удирдах.....*132*
Төл болон анги гишүүдийг ойлгох.....*137*
Төл болон анги гишүүдийг цэнэглэх.....*139*
Нэмэлт тайлбар.....*141*
Дүгнэлт.....*143*
Асуулт болон дасгал.....*143*
Удамшлыг зохицуулах.....144
Аргыг дахин тодорхойлох ба далдлах.....*144*
Гишүүн хувьсагчийг далдлах.....*147*
Super түлхүүр үгийг ашиглах.....*147*
Обьектын ураг болох.....*148*
Төгс анги болон аргыг бичих.....*152*
Хийсвэр анги болон арга бичих нь.....*153*
Дүгнэлт.....*155*
Асуулт болон дасгал.....*156*
Багтсан ангиуд.....156
Дотоод ангиуд.....*158*
Дүгнэлт.....*161*
Асуулт болон дасгал.....*161*
Тоочих төрлүүд.....163
Асуулт болон дасгал.....165
Ерөнхий төрлүүд.....165
Ерөнхий төрлийг тодорхойлох болон ашиглах.....*165*
Ерөнхий төрлүүдийн хоорондох харилцан холбоо.....*166*
Wildcard төрлүүд.....*167*
Ерөнхий төрлийн аргуудыг тодорхойлох болон ашиглах.....*168*
Ерөнхий төрлүүдийг уламжлал кодтой ашиглах.....*168*
Интерфейсүүд болон багцууд.....171

ИНТЕРФЕЙСҮҮД БОЛОН БАГЦУУД.....171

Интерфейсүүдийг үүсгэх болон ашиглах.....171
Интерфейсийг тодорхойлох.....*172*
Интерфейсийг хэрэгжүүлэх.....*174*
Интерфейсийг төрлөөр ашиглах.....*175*

Анхаар! Интерфейсийг ихэсгэж болохгүй ... *175*
"Static Import" байгууламж ... *176*
Дүгнэлт ... *177*
Асуулт болон дасгал ... *177*
Багцуудыг үүсгэх болон ашиглах ... 177
Багц үүсгэх ... *179*
Багцыг нэрлэх ... *180*
Багц гишүүдийг ашиглах ... *180*
Source болон анги файлуудыг удирдах ... *182*
Дүгнэлт ... *184*
Асуулт болон дасгал ... *184*
Кодлолын ерөнхий асуудлуудыг шийдэх ... 185
КОДЛОЛЫН ЕРӨНХИЙ АСУУДЛУУДЫГ ШИЙДЭХ ... 185
Товъёог ... 188
ТОВЪЁОГ ... 188

Обьект хандалтат хөтөлбөрлөлийн ухагдахуунууд

Хэрэв та обьект хандалтат хэлийг өмнө нь хэрэглэж байгаагүй бол, кодоо бичиж эхлэхээсээ өмнө дараах сэдвүүдийг ойлгох хэрэгтэй. Обьект гэж юу болох, анги гэж юу болох, обьект болон анги нь яаж хоорондоо холбогддог, мөн мэдээ ашиглан обьектыг яаж дамжуулдаг талаар ойлгох хэрэгтэй. Уг бүлгийн эхний хэсгүүд нь обьект хандалтат хөтөлбөрлөлийн цаадах ухагдахуунуудыг тайлбарлана. Сүүлийн хэсэг нь эдгээр ухагдахуунуудыг кодтой хэрхэн уялдахыг харуулна.

- Мэдээ гэж юу вэ?
- Тусагдахуун гэж юу вэ?
- Анги гэж юу вэ?
- Удамшил гэж юу вэ?
- Үүд гэж юу вэ?
- Эдгээр ухагдахуунууд кодтой хэрхэн уялдах вэ?
- Асуулт болон дасгал

Тусагдахуун гэж юу вэ?

Тусагдахуун чиглэлийн сэтгэлгээ ойлгох тулгуур бол тусагдахуун юм. Та эргэн тойрноо ажиглаад таны нохой, таны ширээ, таны зурагт, таны унадаг дугуй зэрэг бодит ертөнцийн тусагдахууны олон жишээ харж болно.

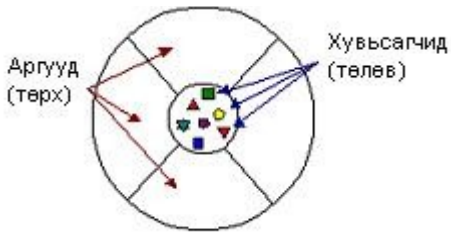
Эдгээр бодит ертөнцийн тусагдахууныг төлөв ба төрх гэсэн хоёр үзүүлэлт нэгтгэдэг. Жишээлбэл, нохой нь төлөв (нэр, өнгө, үүлдэр, өлсгөлөн) ба төрх (хуцах, дайрах, сүүлээ шарвах) агуулна. Унадаг дугуй мөн л төлөв (тохиосон араа, жийлтийн хэмнэл, хоёр дугуй, арааны тоо) ба төрх (зогсоох, хурдасгах, удаашруулах, араа солих) агуулна.

Туурвил тусагдахуунууд нь бодит ертөнцийн тусагдахуунуудыг загварчилдаг тул мөн төлөв болон төрх агуулна. Туурвил тусагдахуунууд нь өөрийн төлвөө нэг юмуу түүнээс дээш *хувьсагч* дотор тээнэ. Хувьсагч бол төлөөллөөр нэршсэн өгөгдөхүүний нэг нэгж юм. Туурвил тусагдахуун нь өөрийн төрхөө *аргууд* ашиглан хэрэгжуулнэ. Арга бол тусагдахуунтай холбоотой эрдэм (дэд ажил) юм.

Тодорхойлолт: Тусагдахуун бол хувьсагчид болон холбогдох аргуудын багц юм.

Бодит ертөнцийн тусагдахуунуудыг туурвил тусагдахуунуудаар орлуулж болно. Магадгуй та бодит ертөнцийн нохойг анимацийн хөтөлбөр дотор туурвил тусагдахуун болгон эсвэл бодит ертөнцийн унадаг дугуйг дасгалын электрон унадаг дугуй удирддаг хөтөлбөр дотор туурвил тусагдахуун болгон дүрсэлж болох юм. Туурвил тусагдахуунууд ашиглан хийсвэр ухагдахуунуудыг загварчилж бас болно. Жишээлбэл, GUI цонхны тогтолцоонд өргөн хэрэглэгддэг үзэгдэл бол хулганын даруул юмуу товчлуурын товч дээр дарсан хэрэглэгчийн үйлдлийг дүрсэлдэг хийсвэр тусагдахуун юм.

Дараах зургаар туурвил тусагдахууны нийтлэг дүрслэлийг харуулав.



Туурвил тусагдахуунуудын мэдэх (төлөв), чадах (төрх) бүхнийг тухайн тусагдахуун доторхи хувьсагчид болон аргуудаар илэрхийлдэг. Бодит ертөнцийн унадаг дугуй загварчилсан туурвил тусагдахуун нь унадаг дугуйны төлөв тусгасан нэлээд хувьсагч агуулж болно: хурд нь 10 миль/цаг, жийлтийн хэмнэл нь 90 миль/цаг, тохиосон араа нь 5 дугаар араа гэх мэт. Эдгээр хувьсагчид нь тухайн нэг унадаг дугуй тусагдахууны төлөв агуулж байгаа, түүнчлэн тусагдахуун чиглэлийн нэр томъёонд тухайлсан тусагдахууныг дүр хэмээн нэрлэдэг тул эдгээрийг *дүрийн хувьсагчид* гэнэ.

Дараах зурагт унадаг дугуйг туурвил тусагдахуунаар дүрсэлжээ.

Хувьсагчдаас гадна, туурвил унадаг дугуй нь зогсоох, жийлтийн хэмнэл өөрчлөх болон араа солих аргууд агуулж болно. (Хэр өгсүүр газар хэддүгээр араагаар хэр хурдан жийж байна, тоормоз атгаатай байна уу зэргээс унадаг дугуйн хурд дам хамаарах тул унадаг дугуйнд хурд өөрчлөх арга байхгүй байж болно). Тухайн нэг унадаг дугуйн дурын төлөвийг өөрчилж буй тул эдгээр аргыг **дүрийн аргууд гэнэ.**

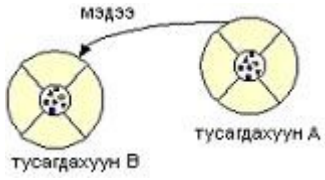
Тусагдахууны диаграм дээр тусагдахууны хувьсагчид нь тусагдахууны төв буюу цөмийг бүрдүүлнэ гэдгийг тодорхой харуулж байна. Аргууд нь тусагдахууны цөмийг хөтөлбөр доторхи бусад тусагдахуунаас далдлан хүрээлсэн байна. Тусагдахууны хувьсагчдыг аргуудын хамгаалалтын бүрхүүлээр хуяглахыг **битүүмжлэл** гэнэ. Аргуудын хамгаалалтын бүрхүүл дотор багцлагдсан хувьсагчдын цөм мэтээр зүйрлэсэн тусагдахууны дүрслэл нь тусагдахууны төгс дүрслэл төдийгүй тусагдахуун чиглэлийн тогтолцоо зохиогчдын амин шүтээн юм. Гэвч үүгээр бүх түүх дуусахгүй. Гол төлөв ажлын шаардлагаар тусагдахууны зарим хувьсагчийг ил гаргах, зарим аргыг далдлах хэрэгцээ гардаг. Java хөтөлбөрлөлийн хэлэнд тусагдахууны хувьсагч, арга нэг бүрийг нэвтрэлтийн дөрвөн түвшний нэгээр зүйлчилж болно. Тухайн нэг хувьсагч юмуу арга руу өөр ямар тусагдахуун буюу анги нэвтэрч болохыг нэвтрэлтийн түвшин зохицуулна. Холбоотой хувьсагч болон аргуудыг туурвилын нэг багц болгон битүүмжилсэн нь туурвил туурвигчдад үндсэн хоёр давуу тал авч ирсэн тун энгийн атлаа үлэмж хүчтэй санаа юм.

- Цогц чанар: нэг тусагдахууны эх кодыг нөгөө тусагдахууны эх кодоос хамааралгүй бичиж, ашиглаж болно. Түүнчлэн нэг тогтолцоо дотор тусагдахууныг чөлөөтэй дамжуулж болно. Таны унадаг дугуй өөр хүнд очоод ажилласаар байх болно.

- Мэдээлэл далдлалт: Тусагдахуун нь бусад тусагдахуунтай харилцахдаа нийтийн үүд ашиглана. Тусагдахуун нь өөртэйгөө холбоотой бусад тусагдахуунуудад огт нөлөөлөхгүйгээр дурын үед өөрчилж болох хувийн мэдээлэл болон аргууд агуулж болно. Та дугуй унахдаа арааны механизмыг нь ойлгох албагүй.

Мэдээ гэж юу вэ?

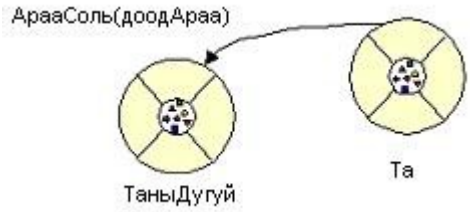
Тусагдахуун дангаараа тийм ч хэрэгтэй биш. Харин үүний оронд, тусагдахуун нь хэдэн зуун тусагдахуун агуулсан томоохон хөтөлбөр юмуу хэрэглүүрийн хэсэг хэлбэрээр тохиолддог. Эдгээр тусагдахууны харилцан үйлчлэлээр дамжуулан хөтөлбөрлөгчид нь өндөр эрэмбэлэгдсэн гүйцэтгэх чадвар болон илүү түвэгтэй төрх бүтээдэг. Таны гараашид хэвтэх унадаг дугуй бол ердөө л нэг металийн хайлш, резин төдий цутгамал бөгөөд яг өөрөө ямар нэгэн үйл ажиллагаа явуулж үл чадна. Зөвхөн өөр тусагдахуунтай (та) харьцах (жийх) тохиолдолд унадаг дугуйны хэрэг гарна.



Туурвил тусагдахуунууд нь бие биедээ *мэдээнүүд* илгээх замаар бие биетэйгээ харьцаж, харилцан үйлчилнэ. Тусагдахуун А нь тусагдахуун В-ээс аль нэг арга гүйцэтгэхийг хүсэхдээ тусагдахуун В рүү мэдээ илгээнэ.

Зарим үед хүлээн авагч тусагдахуун нь юу хийхээ тодорхой мэдэхийн тулд нэмэлт мэдээлэл шаарддаг; жишээ нь араа солихдоо ямар араа гэдгээ тусгах шаардлага гарна. Ийм мэдээллийг *хэмжигдэхүүнүүд* маягаар мэдээний хамт илгээнэ.

Дараах зураг нь мэдээ бүрдүүлэгч гурван хэсгийг харуулжээ



1. Мэдээний зорьсон тусагдахуун (таныДугуй)
2. Гүйцэтгэх аргын нэр (арааСоль)
3. Аргад шаардагдах ямарваа хэмжигдэхүүн (доодАраа)

Эдгээр гурван хэсэг нь захиалсан арга гүйцэтгэхэд нь хулээн авагч тусагдахуунд бүрэн хүрэлцэнэ.
Мэдээ нь чухал хоёр давуу тал өгнө:

1. Тусагдахууны төрх нь аргаар илэрхийлэгдэнэ, тиймээс мэдээ илгээлт нь (хувьсагчийн шууд нэвтрэлтээс гадна) тусагдахуун хоорондын боломжит бүх харилцан үйлчлэлийг дэмжинэ.

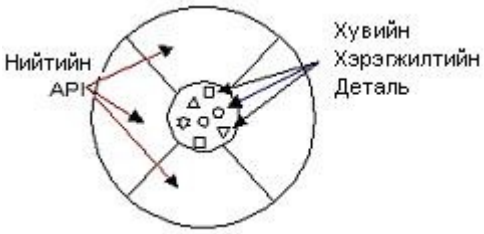
2. Бие биедээ мэдээ илгээхийн тулд тусагдахуунууд нь заавал нэг үйл юмуу нэг тооцоолуур дотор байрлах албагүй.

Анги гэж юу вэ?

Бодит ертөнцөд нэг төрлийн олон тусагдахуун элбэг тохиолдоно. Жишээ нь, таны унадаг дугуй бол энэ дэлхий дээр байгаа олон унадаг дугуйны Зөвхөн нэг нь юм. Тусагдахуун чиглэлийн хэлээр яривал, таны дугуй тусагдахуун нь дугуй гэдэг тусагдахууны ангийн нэг **дүр** юм. Унадаг дугуйнууд нь нийтлэг төлөв (тохиосон араа, жийлтийн хэмнэл, хоёр дугуй, арааны тоо) болон төрх (зогсоох, хурдасгах, удаашруулах, араа солих) агуулна. Гэхдээ унадаг дугуй бүрийн төлөв нь бусдынхаас ялгаатай, бие даасан байдалтай байна.

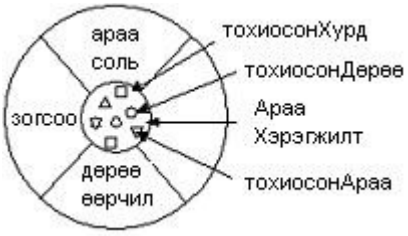
Унадаг дугуй үйлдвэрлэгчид унадаг дугуйны нийтлэг үзүүлэлтийг ашиглан нэг хэвээр олон унадаг дугуй үйлдвэрлэдэг. Үйлдвэрлэж буй унадаг дугуй болгонд нэг хэв бүтээдэг бол тун үр ашиггүй байхсан.

Тусагдахуун чиглэлийн туурвил дотор нийтлэг үзүүлэлттэй нэг төрлийн олон тусагдахуун элбэг тохиолдоно: тэгш өнцөгтүүд, ажилчдын бичлэгүүд, видео клипүүд гэх мэт. Унадаг дугуй үйлдвэрлэгчдийн жишээгээр нэг төрлийн тусагдахуунууд адилхан гэдэг дээр ундэслэн тэдгээр тусагдахуунуудад зориулсан хэв бүтээж болно. Тусагдахуунд зориулсан туурвил хэвийг **анги** гэнэ.

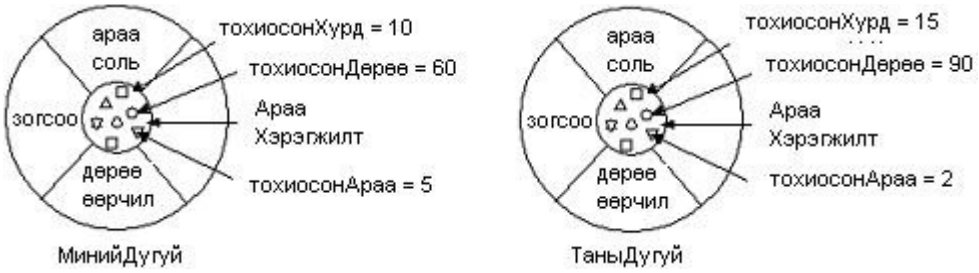


Тодорхойлолт: Анги бол тодорхой төрлийн бух тусагдахууны нийтлэг хувьсагчид болон аргуудыг тодорхойлсон хэв буюу үлгэр юм.

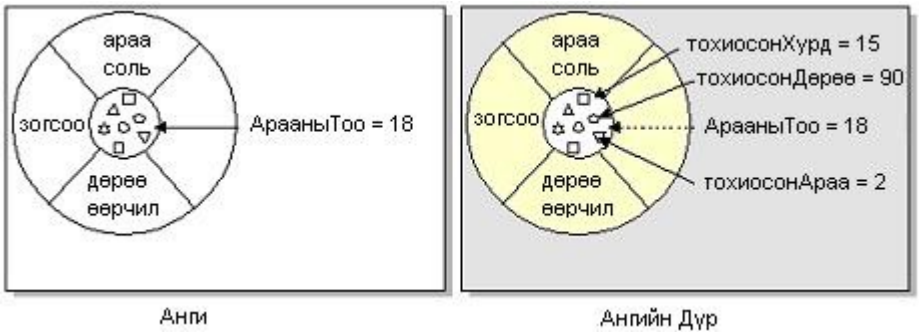
Манай унадаг дугуйны жишээнд дурдагдсан анги нь унадаг дугуй тусагдахуун бүрийн тохиосон араа, жийлтийн хэмнэл зэргийг агуулахад шаардагдах дүрийн хувьсагчдыг зарлаж болно. Үүний дээр дараах зурагт үзүүлсэнчлэн, уг анги нь дугуйчинд араа солих, зогсоох, дорооний хэмнэл өөрчлөх бололцоо олгосон дурын аргууд зарлан хэрэгжүүлж болно.



Унадаг дугуй анги бүтээсний дараа уг ангиас хэдэн ч тусагдахуун байгуулж болно. Ангийн дүр үүсгэх үед тогтолцоо нь тусагдахуун болон түүний бүх дүрийн хувьсагчдад хүрэлцэхуйц хэмжээний ой хуваарилна. Тэдгээр дүр нь тухайн анги дотор тодорхойлогдсон бүх дүрийн хувьсагчийн өөрийн гэсэн оногдлыг тус бүрдээ эзэмшинэ.



Дурын хувьсагчдаас гадна, ангиуд нь **ангийн хувьсагчид** зарлаж болно. Ангийн хувьсагч нь тухайн ангийн бүх тусагдахууны дундын эзэмшлийн мэдээлэл агуулна. Жишээ нь, бүх унадаг дугуйны арааны тоо ижил гэж тооцъё. Энэ тохиолдолд арааны тоо хадгалах дүрийн хувьсагч тодорхойлох нь үр ашиг муутай, дүр болгон өөрийн гэсэн дүрийн хувьсагчтай байх боловч бүх дүрийн хувьд утга нь ижил байна. Иймэрхүү нөхцөлд арааны тоо агуулсан ангийн хувьсагч тодорхойлж болно. Анги нь **ангийн арга** тодорхойлж бас болно. Дүрийн аргыг тухайн нэг тусагдахуун дээр дууддаг бол ангийн аргыг ангиас нь шууд дуудаж болно.



Тусагдахуун ба Анги

Тусагдахуун болон ангийн зургууд хоорондоо их адилхан байгааг та лав анзаарсан байх. Анги, тусагдахуун хоёрын ялгаа нь будилааны эх үүсвэр болох нь олонтаа. Бодит ертөнцөд анги бол дүрсэлж буй тусагдахуун өөрөө биш гэдэг нь илэрхий, унадаг дугуйны хэв нь унадаг дугуйнаас ялгаатай. Гэвч туурвил дотор анги, тусагдахуун хоёрыг ялгахад бага зэрэг бэрхшээл учирдаг. Энэ нь нэг талаар туурвил тусагдахуун бол хийсвэр ухагдахуун юмуу бодит ертөнцийн тусагдахууны электрон загвар гэдэгтэй юуны өмнө холбоотой. Нөгөө талаар анги, дүр хоёрын талаар дурдахдаа “тусагдахуун” гэдэг нэр томъёо адилхан хэрэглэдэгтэй бас холбоотой.

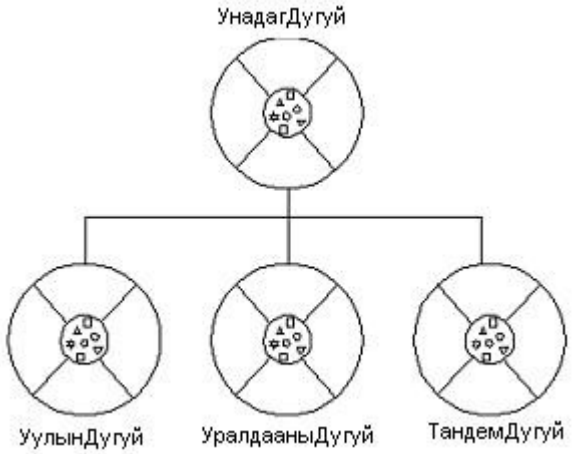
Зураг дээр анги нь тусагдахуун биш, хэв гэдгийг харуулахын тулд дэвсгэр зураагүй байна. Нөгөөтэйгүүр, тусагдахуун оршин байна, түүнийг хэрэглэж болно гэдэг санааны үүднээс тусагдахууныг дэвсгэртэй зурсан байна.

Удамшил гэж юу вэ?

Ерөнхийдөө, тусагдахуун нь анги дотор тодорхойлогдоно. Ангийг нь мэдсэнээр тусагдахууны талаар ихийг мэдэж болно. Хэдийгээр та penny-farthing гэж мэдэхгүй боловч тэр бол дугуй гэдгийг хэлбэл энэ нь хоёр дугуйтай, жолооны бариултай, дөрөөтэй гэдгийг нь та лав гадарлах байх.

Тусагдахуун чиглэлийн тогтолцоо нь энэ алхмыг улам гүнзгийрүүлэн нэг ангийг нөгөө анги дээр тулгуурлан тодорхойлох бололцоо олгодог. Жишээлбэл, уулын дугуй,

уралдааны дугуй, тандем нь бугд унадаг дугуй юм. Тусагдахуун чиглэлийн хэлээр, уулын дугуй, уралдааны дугуй, тандем зэрэг нь унадаг дугуй ангийн **дэд ангиуд** юм. Уунтэй тостэйгөөр, унадаг дугуй анги нь уулын дугуй, уралдааны дугуй болон тандемын **дээд анги** юм. Энэ шутэлцээг доорхи зургаар харуулав.



Дэд анги бүр нь дээд ангиас (хувьсагчийн мэдэгдэл хэлбэртэй) төлөв өвлөнө. Уулын дугуй, уралдааны дугуй, тандем зэрэг нь зарим ижил төлөв агуулна: дөрөө, хурд гэх мэт. Түүнчлэн дэд анги бүр нь дээд ангиас аргууд өвлөнө. Уулын дугуй, уралдааны дугуй, тандем зэрэг нь зарим ижил арга агуулна: зогсоох, жийлтийн хурд өөрчлөх гэх мэт.

Гэхдээ, дэд ангиуд нь дээд ангиас удамшсан төлөв болон төрхөөр хязгаарлагдахгүй. Дэд ангиуд нь дээд ангиас удамшсан хувьсагчид болон аргууд дээр шинийг нэмж болно. Тандем дугуй нь хоёр суудал, хоёр хос бариултай байхад зарим уулын дугуй нь арааны харьцаа багатай нэмэлт араагаар тоноглогдсон байдаг.

Түүнчлэн, дэд ангиуд нь удамшсан аргуудыг шинэчлэн эдгээр аргын өвөрмөц хэрэгжилт бүрдүүлнэ. Жишээлбэл, таны уулын дугуй нэмэлт араануудаар тоноглогдсон бол тэдгээр арааг ашигласан “арааСоль” арга шинээр тодорхойлж болно.

Удамшил нь Зөвхөн нэг давхаргаар хязгаарлагдахгүй. Удамшлын мод буюу ангийн **шатлал** нь шаардлагатай бол хичнээн ч гүн байж болно. Аргууд болон хувьсагчид нь түвшний дагуу доош өвлөгдөнө. Ерөнхийдөө, шатлал доошлохын хэрээр ангийн төрх илүү нарийсна.

Ангийн шатлалын оргилд Object анги байрлах бөгөөд аливаа анги нь (шууд ба дам хэлбэрээр) түүний удам байдаг. Object төрлийн хувьсагч нь ангийн дүр юмуу бүлгэм зэрэг дурын тусагдахууны заалтуур агуулж чадна. Object нь Java Виртуал Машин дотор ажиллах бух тусагдахууны нийтлэг төрхийг хангана. Жишээлбэл, бүх ангиуд нь тухайн тусагдахууны хэлхээ дүрслэл буцаадаг Object-ийн toString арга өвлөнө.

Удамшил нь дараах давуу тал үзүүлнэ:

- Дэд ангиуд нь дээд ангиас нийлүүлэгдэх ерөнхий элементүүдийн суурь дээр тулгуурласан өвөрмөц төрхийг хангана. Удамшлын хэрэглээгээр дамжуулан хөтөлбөрлөгчид нь дээд анги дахь кодыг олон дахин хэрэглэж болно.

- Хөтөлбөрлөгчид нь “нийтлэг” төрх тодорхойлдог хийсвэр анги гэж нэрлэгдэх дээд ангийг хэрэгжуулж болно. Хийсвэр дээд анги нь төрх тодорхойлохдоо

хэсэгчлэн хэрэгжүүлж болно, гэхдээ ийм ангийн ихэнх хэсгийг тодорхойлоогүй буюу хэрэгжүүлээгүй байдаг. Бусад хөтөлбөрлөгчид тусгайлсан дэд ангиуд бүтээхдээ деталиудыг нь гүйцээн бичдэг.

Үүд гэж юу вэ?

Англи хэлэнд, интерфейс (үүд) бол хоорондоо холбоогүй зүйлс харилцан үйлчлэлцэхдээ хэрэглэдэг төхөөрөмж буюу тогтолцоо юм. Энэ тодорхойлолт ёсоор, алсын удирдлага бол зурагт, та хоёрын хоорондын үүд, англи хэл бол хоёр хүний хоорондын үүд, түүнчлэн цэргийн ангид мөрдөгддөг төрхийн протокол нь өөр өөр цолтой хүмүүсийн хоорондын үүд юм. Java хөтөлбөрлөлийн хэлээр, үүд бол хоорондоо холбоогүй тусагдахуунууд бие биетэйгээ харьцахад хэрэглэдэг хэрэгслэл юм. Үүд нь магадгүй протоколтой (тохиролцсон төрх) илүү төстэй. Үнэндээ, бусад тусагдахуун чиглэлийн хэлүүд үүдийн чадвартай байдаг, гэхдээ тэд өөрсдийн үүдийг протокол гэж нэрлэдэг.

Унадаг дугуй анги болон түүний ангийн шатлал нь “унадаг дугуйлаг” үүднээс унадаг дугуй юу хийж чадах, чадахгүй гэдгийг тодорхойлно. Гэвч унадаг дугуй нь ертөнцтэй өөр үүдээр харилцана. Жишээлбэл, агуулах дахь унадаг дугуйг хөрөнгийн хөтөлбөрөөр хянаж болно. Хэрэв агуулахын бараанууд нь үнэ, барааны дугаар зэрэг чухал мэдээллээр хангах аваас хөрөнгийн хөтөлбөр ямар ангийн гишүүнтэй харьцаж байгаагаа мэдэх албагүй. Үүднээс өөр ямар ч холбоогүй эдгээр гишүүдийн шүтэлцээг эвдэхийн оронд хөрөнгийн хөтөлбөр нь харилцааны протокол бүрдүүлж болно. Энэ протокол нь ямар нэгэн Үүд дотор тодорхойлогдсон тогтмолууд болон аргуудын олонлог хэлбэртэй байна. Хөрөнгийн үүд нь жижиглэнгийн үнэ тогтоох, авах, барааны дугаар олгох зэрэг аргуудыг хэрэгжүүлэхгүйгээр тодорхойлж болно.

Энэхүү хөрөнгийн хөтөлбөр дотор ажиллахын тулд , унадаг дугуй анги нь дурдсан Үүдийг хэрэгжүүлэх замаар протоколоо зөвшөөрөх ёстой. Ямар нэгэн үүдийг ямар нэгэн анги хэрэгжүүлэх үед тухайн анги нь тухайн Үүд дотор тодорхойлогдсон бүх аргыг хэрэгжүүлэхээр зөвшөөрсөн болно. Тиймээс, унадаг дугуй анги нь жижиглэнгийн үнэ тогтоох болон авах, барааны дугаар олгох хийгээд бусад аргуудыг хэрэгжүүлэх хэрэгтэй болно.

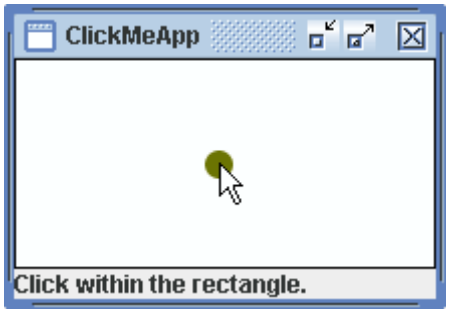
Үүдийг хэрэглэхдээ ангийн шатлал дотор хаана ч ямар ч анги хэрэгжүүлэх боломжтой тийм нийтлэг төрхийн протоколыг тодорхойлдог. Үүд нь дараах ач холбогдолтой:

- Ангийн шатлалыг зохиомлоор эвдэхгүйгээр хоорондоо хамааралгүй ангиудыг адилтгана.
- Нэг болон түүнээс дээш ангиудад хэрэгжүүлэх аргуудыг зарлана.
- Ангийг нь нарийвчлан үзэлгүйгээр тусагдахууны ангийн үүдийг авч үзнэ.

Эдгээр ухагдахуунууд нь хэрхэн код руу хөрвүүлэгдэх вэ?

Одоо та Обьект Хандалтат Прорамчлалын ухагдахууны талаар тодорхой ойлголттой болсон тул эдгээр ухагдахуунууд нь хэрхэн код руу хөрвүүлэгдэх талаар авч үзье. Дараах дүрс нь ClickMe нэртэй гаршуулсан GUI компонентыг дүрсэлсэн ClickMeApp

нэртэй жишээгээр үзүүлсэн хэрэглэгчийн график интерфейсийн (GUI) зураг юм. Таныг ClickMe компонент дотор хулганаа товших үед (click хийх) толбо бий болж байна.



ClickMeApp жишээ нь том биш боловч та програмчлалын талаар туршлага багатай бол энэ код нь сүрдмээр байх болно. Бид таныг энэ жишээнд гарах бүхнийг байгаагаар нь ойлгоно гэж найдахгүй байгаа бөгөөд энэ хэсэгт дэлгэрэнгүй тайлбарлаагүй болно. Энэ санаа нь танд зарим эх кодыг болон таны дөнгөж сурсан нэр томъёог мөн ухагдахуунуудтай нэгтгэхийг илэрхийлнэ. Та дараагийн бүлгүүдэд дэлгэрэнгүй үзэх болно.

Жишээг эмхэтгэх болон гүйлгэх

ClickMeApp жишээ нь ClickMeApp.java, ClickMe.java, ба Spot.java 3 эх файлтай. Эдгээр эх файлууд нь ClickMeApp, ClickMe, ба Spot гэдэг 3 ангид агуулагддаг. Нэмж хэлэхэд, энэ жишээ нь эдгээр ангиудыг ашиглахаас гадна Java платформоор хангагдсан зарим ангиудыг ашиглана.

Та ClickMeApp.java, ClickMe.java, ба Spot.java файлуудыг ClickMeApp.java дахь Java эмхтгүүрээр (javac) эмхэтгэж болно.

Та Java Web Start эсвэл java коммандыг ашиглан ClickMeApp жишээг хэрэглэгдэхүүн болгон гүйлгэж болно. Урдах эмхэгтгэсэн хувилбарыг Java Web Start –р гүйлгэхдээ толилууруаа энэ URL руу зааж өгнө.

http://java.sun.com/docs/books/tutorialJWS/java/concepts/ex5/ClickMeApp.jnlp

---

Санамж: ClickMeApp нь J2SE 5.0. –д танилцуулсан API д түшиглэсэн. Энэ нь JDK –н өмнөх хувилбаруудад болон гүйх үед гүйлгэж эсвэл эмхэтгэж болохгүй.

Жишээн дэх обьектууд

ClickMeApp жишээнд олон обьектууд оролцож байна. Хамгийн илэрхий дэлгэцэн дээр дүрслэгддэг хэсэг нь GUI ийг агуулсан цонх, хэрэглэгчийн хийх зүйлийг тодорхойлсон хаяг, эхний хоосон тэгш өнцөгтийн зурдаг гаршуулсан компонент, гаршуулсан компонентоор зурагдсан толбо зэрэг юм.

Эдгээр обьектууд хэрэглэгчийг хэрэглэгдэхүүнийг ажиллуулах үед үүсдэг. Хэрэглүүрийн main арга бүхэл хэрэглэгдэхүүний турш ажиллах обьектыг үүсгээд энэ обьект нь бусдад цонх, хаяг, гаршуулсан компонентыг үүсгэдэг.

Гаршуулсан компонентыг төлөөлж байгаа обьект нь дэлгэцэн дээрх толбыг төлөөлөх обьектыг дүрсэлдэг. Таныг хулганаа гаршуулсан компонент дээр товших болгонд компонент нь толбыг spot обьектын X Y байрлалыг өөрчилж толбыг дахин зурна. Толбо обьект нь өөрийгөө дүрсэлдэггүй бөгөөд гаршуулсан компонент нь толбо обьектод байгаа мэдээлэл дээр тулгуурлан дугуй дүрсийг зурдаг.

Өөр харагддаггүй обьектууд нь мөн хэрэглэгдэхүүнд оролцоно. Гаршуулсан компонент дах 3 өнгийг (хар, цагаан, ногоон) 3 обьект дүрсэлдэг, үзэгдэл (event) обьект нь хэрэглэгч хулганаа товших гэх мэт үйлдэлийг нь дүрсэлдэг.

Жишээн дэх ангиуд

Та хэрэглэгдэхүүнийг дүрслэж буй обьект нь ClickMeApp ангийн төл гэдгийг, гаршуулсан компонентыг дүрслэж буй обьект нь ClickMe –ын төл гэдгийг, мөн толбо нь Spot –ын төлөөр дүрслэгдэнэ гэдгийг гадарласан байхаа.

Дэлгэцэн дээр байгаа толбыг дүрслэх нь амархан учраас, үүнийг кодыг нь үзье. Spot анги нь 3 төл хувьсагч зарладаг: толбын радиусыг агуулсан хэмжээ, толбын хэвтээ байрлалыг агуулсан X , толбын босоо байрлалыг агуулсанY. Энэ нь өөр 2 аргыг болон байгуулагчийг (байгуулагч – ангиас үүссэн шинэ объектыг цэнэглэхийн тулд хэрэглэдэг дэд програм) зарладаг.

```
public class Spot {
    //instance variables
    public int x, y;
    private int size;

    //constructor
    public Spot() {
        x = -1;
        y = -1;
        size = 1;
    }

    //methods for access to the size instance variable
    public void setSize(int newSize) {
        if (newSize >= 0) {
            size = newSize;
        }
    }
    public int getSize() {
        return size;
    }
}
```

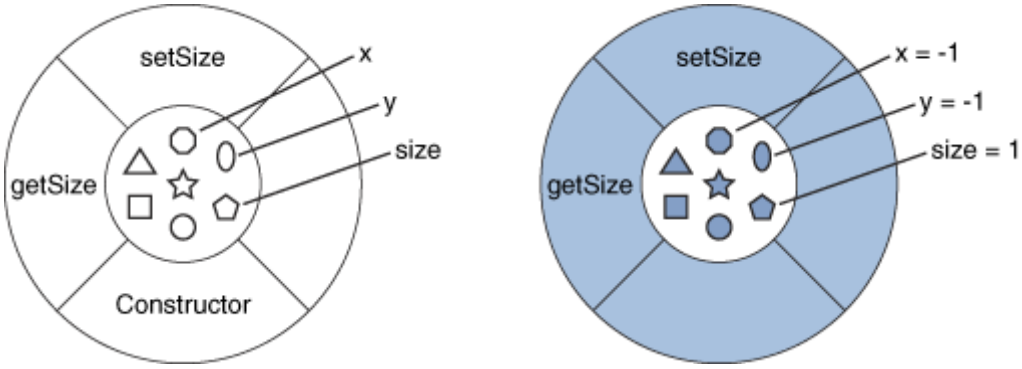
Та байгуулагчийг ангийнхаа нэртэй адил нэртэй байх учир таньж болно. Spot гэдэг байгуулагч нь тухайн обьектынхоо бүх 3 хувьсагчыг цэнэглэнэ. Хэрэглэгдэхүүн эхлэх үед толбо дэлгэцэн дээр гарахгүй үед X,Y хувьсагчидад -1 гэсэн утга олгоно. Size хувьсагч нь боломжит хувьсагч болсон үед 1 гэсэн утга олгоно.

setSize болон getSize аргууд нь бусад обьектуудад төл хувьсагчийн бодит хэмжээг өөрчлөх болон уншихыг бусад обьектуудад бодит хувьсагчид хандах боломжгүйгээр хангадаг. Хэрэглүүрийн ClickMe обьект нь Spot обьектыг хэрэглэхдээ толбыг зурах эсэх болон хаана зурахыг зааж өгнө. Spot обьектыг ClickMe анги хэрхэн үүсгэх болон зарлахыг дараах кодод харууллаа.

```
private Spot spot = null;
...
spot = new Spot();
```

Өгөгдлийн Spot төрөлтэй Spot нэртэй хувьсагчийг эхний мөрөнд зааж өглөө: энэ нь хувьсагчийн утгыг null болгоно. Обьектын заасан хувьсагчийн утга нь тодорхойгүй гэдгийг null гэсэн түлхүүр үг нь илэрхийлнэ. Дараагийн мөр нь обьектыг хэрхэн үүсгэхийг зааж байна. new түлхүүр үг нь обьектод санах ойн зай хуваарилна. Spot( ) нь өмнө үзсэн байгуулагчийг дуудна.

Дараах зургийн зүүн талд Spot анги, баруун талд шинээр цэнэглэгдсэн Spot –ын төл байна.



Жишээн дэх мэдээнүүд

Таны мэдэхээр А обьект нь В обьектыг ямар нэгэн зүйл хийлгэхийг хүсэхийн тулд мэдээ илгээдэг. Мэдээ нь 3 компоненттой.

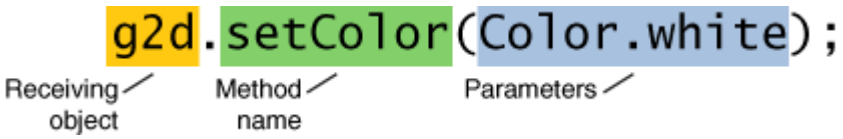
- Тухайн обьектод хаяглагдсан мэдээ
- Гүйцэтгэх гэж буй тухайн аргын нэр
- Аргад хэрэглэгдэх параметрууд

Доор ClickMe ангийн 2 мөр жишээ бичлээ.

```
g2d.setColor(Color.WHITE);
g2d.fillRect(0, 0, getWidth() - 1, getHeight() - 1);
```

Энэ 2 мэдээ нь хоёулаа ClickMe обьектын g2d (Graphics2D дэлгэцэн дээр зурахад чухал обьект) гэдэг обьектынх юм. Энэ обьект нь компонентыг зурах систем нь компонентыг өөрийгөө зурахад шаардлагатай мэдээллээр хангадаг. Эхний мөр нь өнгийг цагаан болгож, 2 дахь нь тэгш өнцөгтийг компонентын зайгаар дүүргэдэг, иймээс компонентын бүсийг цагаанаар будна.

Дараах зураг нь эхний мөрийн g2d илгээж буй мэдээний анхаарах хэсгийг үзүүлж байна.



Жишээн дэх удамшил

Өөрийгөө дэлгэцэн дээр зурахын тулд обьект нь компонент байх ёстой. Энэ нь обьект нь Java платформоор хангагдаж буй Component ангиас үүссэн төл байх ёстой.

ClickMe обьект нь ClickMe ангийн төл, ингэж зарлагдана:

```
public class ClickMe extends JComponent implements MouseListener {
   ...
}
```

extends Jcomponent –ын утгат хэсэг нь ClickMe –г Component ангиас үүссэн Jcomponent ангийн дэд анги болгоно. ClickMe нь өөрийгөө зурах чадварын багтааснаараа, ирмэгийг тойрсон стандарт хүрээтэй байх ба хулганы үзэгдлийг хүлээж авдаг сонсогчдыг бүртгэх зэргийг Component and Jcomponent –оос удамшуулж авдаг. Эдгээр чадвараас гадна ClickMe анги нь өөрийн тодорхой үүрэгтэй: өөрийн зурах код нь стандарт байрлалд байх ёстой.

```
public ClickMe() {
    ...
    setPreferredSize(aDimensionObject);
    setMinimumSize(anotherDimensionObject);
    ...
}

public void paintComponent(Graphics g) {
    ... // ClickMe's painting code here
}
```

Жишээн дэх интерфейсүүд

ClickMe компонент нь хулганы товшиход хариу өгөхдөө товшсон газар нь толбыг дэлгэцэнд гаргаж байна. Ямарч обьект нь компонент дээрх хулганы товшихыг илрүүлдэг. Энэ нь MouseListener интерфейсийг хэрэгжүүлэх хэрэгтэй ба хулганы сонсогч мэтээр компоненттой бүртгэнэ.

MouseListener интерфейс нь тус бүрдээ хулганы үзэгдлээс шалтгаалан өөр өөрөөр дуудагддаг 5 аргыг зарладаг. Тэдгээр нь хулганыг товших үед, хулганыг компонентоос гарах үед гэх мэт. Хэдийгээр ClickMe компонент нь хулганыг товших үед л хариулдаг боловч хулган сонсогчийн хэрэгжүүлэлт нь бүх 5 аргаа багтаасан байх хэрэгтэй. Обьектын сонирхоогүй үзэгдлүүдэд зориулсан аргууд нь хоосон байдаг.

ClickMe компонентын бүрэн кодыг доор үзүүллээ. Хулганы үзэгдлийн зохицуулалтанд оролцсон кодыг өнгөт дармалаар (colored boldface) харууллаа.

```
import javax.swing.BorderFactory;
import javax.swing.JComponent;
import java.awt.*;
import java.awt.event.*;

public class ClickMe extends JComponent
                  implements MouseListener {
    private Spot spot = null;
    private static final int RADIUS = 7;
    private Color spotColor = new Color(107, 116, 2); //olive

    /** Creates and initializes the ClickMe component. */
    public ClickMe() {
        addMouseListener(this);

        //Hint at good sizes for this component.
        setPreferredSize(new Dimension(RADIUS*30,
                                       RADIUS*15));
        setMinimumSize(new Dimension(RADIUS*4, RADIUS*4));

        //Request a black line around this component.
        setBorder(
            BorderFactory.createLineBorder(Color.BLACK));
    }

    /**
     * Paints the ClickMe component.  This method is
     * invoked by the Swing component-painting system.
     */
    public void paintComponent(Graphics g) {
        /**
         * Copy the graphics context so we can change it.
         * Cast it to Graphics2D so we can use antialiasing.
         */
        Graphics2D g2d = (Graphics2D)g.create();

        //Turn on antialiasing, so painting is smooth.
        g2d.setRenderingHint(
               RenderingHints.KEY_ANTIALIASING,
               RenderingHints.VALUE_ANTIALIAS_ON);

        //Paint the background.
        g2d.setColor(Color.WHITE);
        g2d.fillRect(0, 0, getWidth() - 1, getHeight() - 1);

        //Paint the spot.
        if (spot != null) {
            int radius = spot.getSize();
            g2d.setColor(spotColor);
```

```
            g2d.fillOval(spot.x - radius, spot.y - radius,
                         radius * 2, radius * 2);
        }
    }

    //Methods required by the MouseListener interface.
    public void mousePressed(MouseEvent event) {
        if (spot == null) {
            spot = new Spot();
            spot.setSize(RADIUS);
        }
        spot.x = event.getX();
        spot.y = event.getY();
        repaint();
    }
    public void mouseClicked(MouseEvent event) {}
    public void mouseReleased(MouseEvent event) {}
    public void mouseEntered(MouseEvent event) {}
    public void mouseExited(MouseEvent event) {}
}
```

API баримтжуулалт

Та ClickMe ын ажиллагааны талаар илүү ихийг мэдхийг хүсвэл түүний өвөг анги болох JComponent болон Component –н талаар илүү судлах хэрэгтэй. Та энэ мэдээллийг яаж олох вэ?

Java платформыг бүтээж буй бүх ангиуд дахь тодорхойлолтуудыг бүрдүүлсэн API баримтжуулалт дотроос та анги бүрийн талаар дэлгэрэнгүй тайлбарыг олох болно.

Java 2 платформын API баримтжуулалт нь java.sun.com дээр бий. Таны ашигладаг бүх хувилбаруудад зориулсан API баримтжуулалтыг өөрийн толилуурtаа тэмдэглээд авах нь хэрэгтэй шүү. Та эндээс API баримтжуулалт 5.0 –ыг олж болно.

http://java.sun.com/j2se/5.0/docs/api/

Java платформоос ClickMe ангид хэрэглэгдсэн бүх ангиуд болон интерфейсүүдийн талаар илүү сурахын тулд, дараах ангиудад зориулсан API баримтжуулалтыг харж болно.

- javax.swing.JComponent
- java.awt.Graphics
- java.awt.Graphics2D
- java.awt.Dimension
- java.awt.Color
- javax.swing.BorderFactory
- java.awt.event.MouseListener
- java.awt.event.MouseEvent

Эдгээрийг ашиглан хэрэглэгчийн график интерфейсийг (GUI) үүсгэх талаар Creating a GUI with JFC/Swing дээр бүрэн тайлбарласан байгаа.

Дүгнэлт

Энэ хэлэлцүүлэгт олон хэсгүүдийг өөрчилж чимсэн ба зарим зүйлүүдийг тайлбарлаагүй орхисон, гэхдээ обьект хандалтат ухагдахуунууд нь код дотор хэрхэн харагддаг талаар зарим ойлголттой болсон байхаа. Тэгэхээр одоо та дараах ерөнхий ойлголтуудыг эзэмших нь зүйтэй:

- Анги нь обьектуудад зориулсан эскиз зураг болох
- Обьектууд нь ангиудаас үүсдэг болох
- Обьектыг хэрхэн ангиас үүсгэх
- Байгуулагчууд гэж юу болох
- Обьектуудыг хэрхэн цэнэглэх
- Ангид зориулсан код нь ямар харагдах
- Анги хувьсагчууд болон анги аргууд гэж юу болох
- Ангийн өвөг ангийг хэрхэн мэдэх
- Интерфейс нь араншингийн протокол болох
- Интерфейсийг хэрэгжүүлэх гэдэг нь юу гэсэн утгатай болох

Асуулт болон дасгал

Асуулт

Дараах асуултанд хариулахын тулд Java 2 Platform дахь API баримтжуулалтыг хэрэглээрэй.

1. ClickMe компонент нь хүрээгээ тавих үедээ Color.BLACK -г хэрэглэдэг. Бусад ямар өнгийг үүн шиг нэрээр авч болох вэ?

2. Ягаан өнгийг дүрслэсэн Color обьектыг хэрхэн үүсгэх вэ? (Сануулга: ягаан нь улаан болон хөх хоёрын тэнцүү хэсгээс бий болсон.)

3. Дүүргэсэн тэгш өнцөгтийг зурахад ямар Graphics аргыг хэрэглэх вэ?

4. Текстийг зурахад ямар Graphics аргыг хэрэглэх вэ?

Дасгал

Сануулга: Энэ дасгалын гол санаа нь таныг код уншихад болон API баримтжуулалтыг хэрэглэхэд туслана. Бид танд хэрхэн код бичих талаар ямар нэгэн дэлгэрэнгүй мэдээллийг өгөөгүй байгаа. Гэхдээ хэр ихийг хийж чадахаа та гайхах байхаа.

Дээрх асуултаас сурсанаа ашиглан дараах CliclMe компонентын өөрчлөлтийг гүйцэтгэ.

1. ClickMe -г өөрчлөхдөө ногоон толбын оронд улаан тэгш өнцөгт зур.
2. ClickMe -г өөрчлөхдөө ногоон толбын оронд өөрийн нэрээ ягаанаар хэвлэ.

ClickMe компонентыг хэрэглэж эмхэтгэхээ санаарай. ClickMe.java, ClickMeApp.java, болон Spot.java файлууд хэрэгтэй шүү. java ClickMeApp командыг ашиглан програмаа гүйлгэж болно.

Хэлний үндсүүд

Дараах BasicsDemo програм нь 1 -с 10 хүртэлх тоонуудыг нэмэн үр дүнг дэлгэцэнд хэвлэж байна.

```
public class BasicsDemo {
    public static void main(String[] args) {
        int sum = 0;
        for (int current = 1; current <= 10; current++) {
            sum += current;
        }
        System.out.println("Sum = " + sum);
    }
}
```

Уг програмын гарц нь:

```
 Sum = 55
```

Дээрх шиг програм нь хэдийгээр жижиг ч гэсэн Java хэл дээрх хувьсагч, оператор, болон удирдлагын урсгалын хэллэгүүдийг багтаасан үндсэн чухал зүйлүүдийг хэрэглэсэн байна.

- Хувьсагчууд
- Операторууд
- Илэрхийллүүд, хэллэгүүд, болон блокууд
- Удирдлагын урсгалын хэллэгүүд

Хувьсагч

Аливаа тусагдахуун өөрийн төлөвийг хувьсагч дотор хадгална.

Тодорхойлолт: Хувьсагч бол төлөөллөөр нэршсэн өгөгдөхvvний нэг нэгж юм.

Хөтөлбөр дотор ашиглагдах бvх хувьсагчийн нэр болон төрлийг тодорхой зааж өгөх ёстой. Хувьсагчийн нэр нь хууль ёсны төлөөлөл – vсгээр эхэлсэн Unicode тэмдэгийн хязгааргvй цуваа – байх ёстой. Хувьсагч дотор агуулагдаж буй өгөгдөхvvн pvv хандахын тулд хувьсагчийн нэр хэрэглэнэ. Ямар утга хадгалах болон утга дээр ямар vйлдэл хийж болохыг хувьсагчийн төрөл заана. Хувьсагчийн нэр, төрөл заахдаа гол төлөв дараах хэлбэртэй хувьсагчийн мэдэгдэл бичнэ.

- Өгөгдлийн төрлүүд
- Хувьсагчийн нэрүүд
- Цар хүрээ
- Хувьсагч цэнэглэх
- Төгс хувьсагч
- Дүгнэлт
- Асуулт болон дасгал

Хувьсагч нь мэдэгдэлд өгөгдсөн төрөл болон нэрээс гадна буйр агуулна. Хувьсагчийн энгийн нэр ашиглаж болох кодын хэсэг бол хувьсагчийн буйр юм. Хувьсагчийн буйр нь хувьсагчийн мэдэгдлийн байрлалаас, өөрөөр хэлбэл кодын бусад хэсгvvдтэй харьцуулахад мэдэгдэл хаана байгаагаас хамааран далдуур тодоройлогдоно.
Дор vзvvлсэн *MaxVariablesDemo* хөтөлбөр нь өөрийн main арга дотор төрөл бvрийн найман хувьсагч зарласан байна. Хувьсагчийн мэдэгдлийг улаанаар тодруулав.

public class *MaxVariablesDemo* {
```
public static void main(String args[]) {

// бvхэл
byte largestByte = Byte.MAX_VALUE;
short largestShort = Short.MAX_VALUE;
int largestInteger = Integer.MAX_VALUE;
long largestLong = Long.MAX_VALUE;

// бодит тоо
float largestFloat = Float.MAX_VALUE;
double largestDouble = Double.MAX_VALUE;

// бусад эгэл төрлvvд
char aChar = 'S';
boolean aBoolean= true;

// бvгдийг vзvvл
System.out.println("Хамгийн их byte утга бол " + largestByte);
System.out.println("Хамгийн их short утга бол " + largestShort);
System.out.println("Хамгийн их integer утга бол " + largestInteger);
System.out.println("Хамгийн их long утга бол " + largestLong);

System.out.println("Хамгийн их float утга бол " + largestFloat);
System.out.println("Хамгийн их double утга бол " + largestDouble);

if (Character.isUpperCase(aChar)) {
System.out.println("Тэмдэг " + aChar + " бол том vсэг.");
} else {
System.out.println("Тэмдэг " + aChar + " бол жижиг vсэг.");
}
System.out.println("aBoolean утга нь " + aBoolean);
}
}
```

Энэ хөтөлбөрийн гарц:

Хамгийн их byte утга бол 127
Хамгийн их short утга бол 32767
Хамгийн их integer утга бол 2147483647
Хамгийн их long утга бол 9223372036854775807
Хамгийн их float утга бол 3.40282e+38
Хамгийн их double утга бол 1.79769e+308
Тэмдэг S бол том vсэг.
aBoolean утга нь true

Дараагийн хэсгvvдэд өгөгдөхvvний төрөл, нэр, буйр, цэнэглэл болон төгс хувьсагч зэрэг хувьсагчийн олон талыг авч vзнэ. *MaxVariablesDemo* хөтөлбөр нь одоогоор таны судалж амжаагvй, тvvнчлэн энэ хэсэгт тайлбарлаагvй байгаа.

MAX_VALUE тогтмолууд болон if-else хэллэг гэсэн хоёр гишvvн хэрэглэж байна.

MAX_VALUE тогтмол бvр нь Java хорсоос хангагдах аль нэгэн тоон анги дотор тодорхойлогдсон , тухайн тоон төрлийн хувьсагч руу олгож болох хамгийн дээд утга байна.

MaxVariablesDemo хөтөлбөр дотор хэрэглэгдэх if-else хэллэгийн талаар энэ хичээлийн удирдлагын урсгалын хэллэг хэсэг дотор vзэх болно.

Өгөгдлийн төрлүүд

Аливаа хувьсагч өгөгдөхvvний төрөл агуулах ёстой. Хувьсагчийн төрөл нь тухайн хувьсагчийн агуулж болох утга болон тvvн дээр хийж болох vйлдлvvдийг тодорхойлно. Жишээлбэл, MaxVariablesDemoхөтөлбөр доторхи int largestInteger мэдэгдлээр largestInteger нь өгөгдөхvvний бvхэл (int) төрөл агуулна гэдгийг зарлаж байна. Бvхэл нь зөвхөн бvхэл тоон утга эерэг болон сөрөг агуулна. Бvхэл хувьсагч дээр нэмэх vйлдэл гэх мэтийн арифметик vйлдэл хийж болно.

Java хөтөлбөрлөлийн хэл нь эгэл ба заалтуур гэсэн өгөгдөхvvний хоёр ангилалтай.

Эгэл төрлийн хувьсагч нь тухайн төрөлд тохирсон хэмжээ болон формат бvхий ганц утга агуулна. Жишээлбэл, int утга нь хоёртын нэмэлт код хэмээн нэрлэгдэх формат бvхий 32 битийн өгөгдөхvvн байдаг бол char утга нь Unicode тэмдэг маягаар форматлагдсан 16 битийн өгөгдөхvvн байна.

Дараах хvснэгтэнд Java-ийн дэмждэг бvх эгэл төрлийг тvлхvvр vгээр нь хэмжээ, формат, тодорхойлолтын хамт тусгав. MaxVariablesDemo хөтөлбөр нь эгэл төрөл бvрээс нэг хувьсагч зарласан байна.

Тvлхvvр vг	Тодорхойлолт	ХэмжээФормат
	(бvхэл тоо)	
byte	Byte – урттай бvхэл	8 бит нэмэлт код
short	Богино бvхэл	16 бит нэмэлт код
int	Бvхэл	32 бит нэмэлт код
long	Урт бvхэл	64 бит нэмэлт код
	(бодит тоо)	
float	Дан нарийвчлалтай бодит	32 бит
double	Давхар нарийвчлалтай бодит	64 бит
	(бусад төрлvvд)	
char	Нэг тэмдэг	16 бит Unicode тэмдэг
boolean	Булын утга (vнэн эсвэл худал)	true / false

Санамж: Бусад хэлvvдэд өгөгдөхvvний эгэл төрлийн формат болон хэмжээ нь тухайн хөтөлбөр ямар хөрс дээр ажиллаж байгаагаас хамаарч болно. Харин хөтөлбөрлөлийн

хэл нь өөрийн өгөгдөхvvний эгэл төрлийн хэмжээ болон форматыг зvйлчилсэн байдаг. Тиймээс, тогтолцооны хамаарлын талаар санаа амар байж болно.

Хөтөлбөр дотор литерал эгэл утга шууд ашиглаж болно. Жишээ нь бvхэл хувьсагч руу 4 гэсэн утга олгохдоо ингэж бичиж болно:

int anInt = 4;

4 гэсэн цифр бол литерал бvхэл утга юм. Энд төрөл бvрийн эгэл төрлийн литерал утгуудыг тусгав.

Литерал	Өгөгдөхvvний төрөл
178	int
8864L	long
37.266	double
37.266D	double
87.363F	float
‘c’	double
true	char
false	boolean

Ерөнхийдөө аравтын цэггvй бичигдсэн цифрийн цуваа бол бvхэл юм. Ийм тооны ард ‘L’юмуу‘l’ бичих замаар урт бvхэл зааж болно.‘1’цифртэй андуурахгvйн тулд‘L’ хэрэглэх нь зохимжтой. Аравтын цэг бvхий цифрийн цуваа бол double төрөл юм. Ийм тооны ард ‘F’ юмуу ‘f’ бичиж float утга зааж болно. Литерал тэмдэг утга нь дан хашилт дотор бичигдсэн дурын Unicode тэмдэг байж болно. Хоёр boolean литерал бол true болон false юм.

Эгэл төрлөөс ялгаатай нь, заалтуур төрлийн хувьсагчийн утга нь хувьсагчаар төлөөлvvлсэн утга юмуу утгын олонлогийн заалтуур хаяг байна. Бусад хэлvvдэд заалтуурыг ойн хаяг буюу заагч гэж нэрлэдэг. Java хөтөлбөрлөлийн хэл нь хаягийг бусад хэлvvдэд хэрэглэдгийн адил шууд хэрэглэдэггvй. Харин тvvний оронд хувьсагчийн нэр хэрэглэнэ.

Хувьсагчийн нэр

Хөтөлбөр нь хувьсагчийн утга руу хувьсагчийн нэрээр хандана. Жишээ нь largestByte хувьсагчийн утга дэлгэц дээр гаргахдаа, MaxVariablesDemo хөтөлбөр нь largestByte гэдэг нэр ашиглана. Ганц төлөөллөөс бvрдэх largestByte зэрэг нэрийг энгийн нэр гэнэ. Vvний эсрэг хэлбэр болох хам нэр нь нэг ангиас нөгөө анги буюу тусагдахуун доторхи гишvvн хувьсагчтай харьцахад хэрэглэгдэнэ.
Java хөтөлбөрлөлийн хэлэнд энгийн нэрийн хувьд дараах зvйл vнэн байх ёстой:

1. Энэ нь хууль ёсны төлөөлөл байх ёстой. Төлөөлөл нь vсгээр эхэлсэн Unicode тэмдгvvдийн хязгааргvй цуваа.
2. Энэ нь тvлхvvр vг, булийн литерал (true, false), null нөөцлөгдсөн vг байх ёсгvй.
3. Энэ нь өөрийн буйр дотор хосгvй байна. Нэг хувьсагч нь өөр буйр дотор зарлагдсан өөр нэг хувьсагчтай ижил нэртэй байж болно. Зарим vед, кодын багтсан блокууд дотор

зарлагдсан хувьсагчууд ижил нэртэй байж болно.

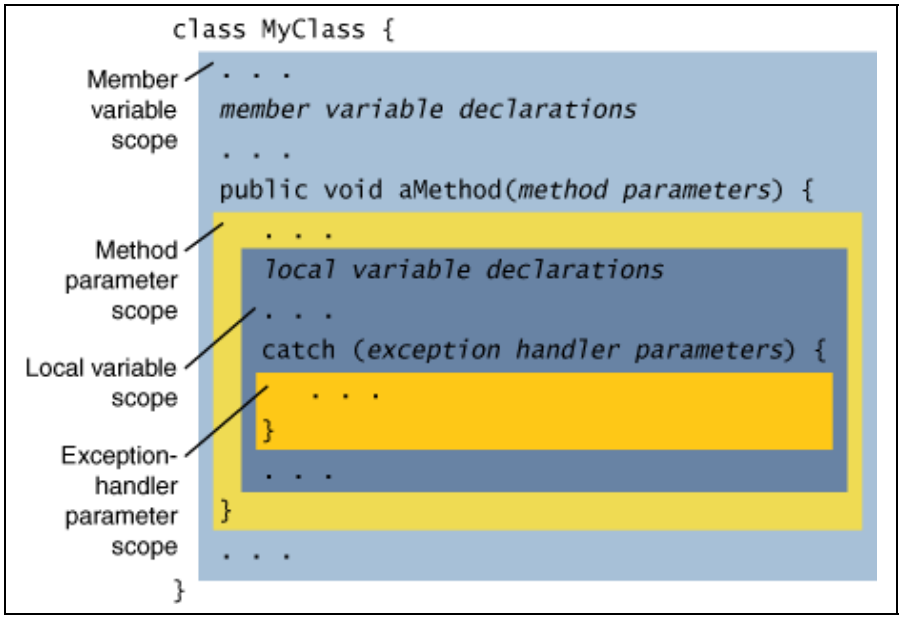
Заншил ёсоор, хувьсагчийн нэр жижиг үсгээр, ангийн нэр том үсгээр эхэлнэ. Хэрэв хувьсагчийн нэр хоёр ба дээш тооны үгээс бүрдсэн бол эхнийхээс бусад бүх үгийг том үсгээр эхлүүлэн залгаж бичнэ. Жишээ нь isVisible гэх мэт

Доогуур зураасыг нэрийн аль ч хэсэгт хэрэглэж болно, гэхдээ заншил ёсоор тогтмол доторхи үгсийг зааглахад хэрэглэнэ.Учир нь, заншил ёсоор тогтмол нь бүгд том үсгээр бичигддэг бөгөөд тиймээс жижиг үсгээр зааг үүсгэдэггүй.

Цар хүрээ

Хувьсагчийн цар хүрээ нь програм доторх муж юм. Хоёрдугаарт, цар хүрээ нь хувьсагч дахь санах ойг систем нь хэзээ үүсгэх болон устгахыг тогтоодог. Цар хүрээ нь зөвхөн гишүүн хувьсагчууд руу хэрэглэгдэх ба хувьсагчийг зарласан ангийн гаднаас хэрэглэж болох эсэхийг тогтооно. Харагдах байдал нь хандалт өөрчлөгчөөр тавигдана. Илүү мэдээллийг Controlling Access to Members of a Class хэсгээс хараарай.

Програм доторх хувьсагчийн зарлалтын байрлал нь өөрийн цар хүрээг байгуулдаг. Дараах зурагт харуулж байгаагаар цар хүрээний дөрвөн зэрэглэл байдаг.



Гишүүн хувьсагч нь анги юмуу обьектын гишүүн вм. Энэ нь ангийн дотор боловч ямар ч арга болон байгуулагчаас гадна зарлагдана. Гишүүн хувьсагчийн цар хүрээ нь тухайн ангийн бүрэн зарлалт юм. Тийм ч гэсэн, гишүүний зарлалт нь гишүүн цэнэглэх илэрхийлэлд хэрэглэхээс өмнө бий болно. Гишүүн хувьсагчийг зарлах талаар Declaring Member Variables хэсгээс хараарай.

Кодын блок дотор локал хувьсагчуудыг зарладаг. Локал хувьсагчийн цар хүрээ нь өөрийн зарлалтыг зарлагдсан кодын блокын төгсгөл хүртэл өргөтгөгдөнө. MaxVariablesDemo жишээнд, main арга дотор зарлагдсан бүх хувьсагчууд локал хувьсагч юм. Тэр програм дахь хувьсагч бүрийн цар хүрээ нь хувьсагчийн зарлалтаас эхлэн main аргын төгсгөл хүртэл өргөтгөгдөж байна – програмын код дахь сүүлийн угалзан баруун хаалт “}” хүртэл.

Параметрууд нь арга болон байгуулагчийн зүй ёсны аргументууд бөгөөд арга болон байгуулагч руу утга дамжуулахад хэрэглэдэг. Classes and Inheritance бүлгийн Defining Methods хэсэгт параметраар арга руу утга дамжуулах талаар хэлэлцэнэ.

Гажуудал засагч параметрууд нь параметртай адилхан, гэхдээ арга болон байгуулагчаас илүү гажуудал засах аргументууд юм. Гажуудал засагч параметрын цар хүрээ нь catch хэллэгийг дагасан { болон } хоорондох кодын блок юм. Handling Errors with Exceptions бүлэг нь алдааг засахад гажуудлыг ашиглах талаар хэлэлцэх бөгөөд параметртай гажуудал засагчийг хэрхэн бичихийг харуулна.

Дараах код жишээг авч үзье.

```
if (...) {
    int i = 17;
    ...
}
System.out.println("The value of i = " + i);    //error
```

Сүүлчийн мөрөнд локал хувьсагч i нь цар хүрээнээсээ гарсан байгаа тул эмхэтгэгдэхгүй. Хувьсагч i –н цар хүрээ нь { болон } хооронд кодын блокт байна. Хувьсагч i нь хаалтыг } хаасны дараа оршин тогтнохоо болино. Дээрхээс харахад нэг бол хувьсагчын зарлалтыг if хэллэгийн гадна зарлана эсвэл println аргыг if хэллэг дотор дуудвал дээрх код нь ажиллана.

Хувьсагч цэнэглэх

Дотоод хувьсагч болон гишүүн хувьсагч зарлах үедээ олговрын хэллэгээр цэнэглэж болно. Тэдгээр хувьсагчийн өгөгдөхүүний төрөл болон олгож буй хувьсагчийн төрөл ижил байх естой. *MaxVariablesDemo* хөтөлбөр нь өөрийн бүх дотоод хувьсагчийг зарлах үедээ тэдгээрийг цэнэглэж байна. Энэ хөтөлбөрийн дотоод хувьсагчийн мэдэгдэл цэнэглэж буй кодыг улаанаар тодруулав.

```
// бүхэл
byte largestByte = Byte.MAX_VALUE;
short largestShort = Short.MAX_VALUE;
int largestInteger = Integer.MAX_VALUE;
long largestLong = Long.MAX_VALUE;

// бодит тоо
float largestFloat = Float.MAX_VALUE;
double largestDouble = Double.MAX_VALUE;

// бусад эгэл төрлүүд
char aChar = 'S';
boolean aBoolean = true;
```

Хэмжигдэхүүн болон үгүйсгэл эзэмдэгчийн хэмжигдэхүүнийг ийм замаар цэнэглэж болохгүй. Хэмжигдэхүүний утгыг дуудагч нь тогтооно.

Төгс хувьсагч

Дурын буйр доторхи хувьсагчийг төгс хувьсагч болгон зарлаж болно. Төгс хувьсагчийн утга нь цэнэглэгдсэний дараа өөрчлөгдөхгүй. Ийм хувьсагч нь хөтөлбөрлөлийн бусад хэлүүд дэх тогтмолтой төстэй.

Төгс хувьсагч зарлахдаа хувьсагчийн мэдэгдэл дотор төрлийн өмнө final түлхүүр үг бичнэ:

```
final int aFinalInt = 0;
```

Өмнөх хэллэг нь нэг дор төгс хувьсагч зарлаад цэнэглэж байна, цаашдаа aFinalVar хувьсагч руу утга олгохоор оролдвол эмхэтгүүрийн алдаанд хүргэнэ. Шаардлагатай бол төгс дотоод хувьсагчийн цэнэглэлийг тусгаарлаж болно. Дотоод хувьсагч зарлаад сүүлд нь дараах маягаар цэнэглэж болно:

```
final int blankfinal;
...
blankfinal = 0;
```

Зарлагдсан боловч цэнэглэгдээгүй төгс дотоод хувьсагчийг хоосон төгс гэнэ. Урьдын адил, нэгэнт төгс дотоод хувьсагч цэнэглэгдсэн бол дахин өөр утга авч болохгүй, түүнчлэн blankfinal руу утга олгохыг оролдвол эмхэтгэх үеийн алдаа гарна.

Дүгнэлт

Хувьсагч зарлах үедээ хувьсагчийн нэр болон өгөгдөхүүний төрлийг тодорхой заана. Java хөтөлбөрлөлийн хэл нь эгэл ба заалтуур гэсэн өгөгдөхүүний төрлийн хоёр хэлбэртэй: Эгэл төрлийн хувьсагч нь утга агуулна. Энэ хүснэгтээр өгөгдөхүүний бүх эгэл төрлийн хэмжээ ба форматыг харуулав.

Түлхүүр үг	Тодорхойлолт	Хэмжээ/Формат
	(бүхэл тоо)	
byte	Byte – урттай бүхэл	8 бит нэмэлт код
short	Богино бүхэл	16 бит нэмэлт код
int	Бүхэл	32 бит нэмэлт код
long	Урт бүхэл	64 бит нэмэлт код
	(бодит тоо)	
float	Дан нарийвчлалтай бодит	32 бит IEEE 754
double	Давхар нарийвчлалтай бодит	64 бит IEEE 754
	(бусад төрлүүд)	
char	Нэг тэмдэг	16 бит Unicode тэмдэг
boolean	Булын утга үнэн эсвэл худал	true / false

Хувьсагчийн мэдэгдлийн байрлал нь тухайн хувьсагчийг энгийн нэрээр нь ашиглах бололцоо олгох кодын хэсэг тусгасан хувьсагчийн буйрыг далдуур тогтооно. Гишүүн хувьсагчийн буйр, дотоод хувьсагчийн буйр, хэмжигдэхүүний буйр болон үгүйсгэл эзэмдэгчийн хэмжигдэхүүний буйр гэсэн буйрын дөрвөн хэлбэр байна. Олговрын үйлдэхүүн (=) ашиглан хувьсагчийн мэдэгдэл дотор хувьсагчийн анхны утга өгч болно.

Хувьсагчийг төгс болгон зарлаж болно. Цэнэглэгдсэний дараа төгс хувьсагчийн утгыг өөрчилж болохгүй.

Асуулт болон дасгал

Асуулт

Дараахь Хувьсагчийн нэрсийн аль нь зов бэ

int
anInt
i
i1
1
thing1
1thing
ONE-HUNDRED
ONE_HUNDRED
something2do

BasicDemo хөтөлбөрийн талаар дараах асуултанд хариул.

1. Хөтөлбөр дотор зарлагдсан хувьсагч бүрийн нэр юу вэ? Аргын хэмжигдэхүүн бас хувьсагч болохыг санаарай.

2. Бүх өгөгдөхүүний бүхэл төрлийн доод утга харуулахаар MaxVariablesDemo хөтөлбөрийг шинэчлэн бич.

3. Том үсэг танихад isUpperCase-ийн оронд хэрэглэгдэх Character ангийн аргын нэрийг тааж чадах уу? IsUpperCase -ийн оронд тэр аргаа хэрэглэхээр MaxVariablesDemo хөтөлбөрийг өөрчил.

Дасгал

1. aBoolean өөр утга авахаар MaxVariablesDemo хөтөлбөрийг өөрчил

2. Өгөгдөхүүний бүхэл төрөл нэг бүрийн доод утга харуулдаг байхаар MaxVariables хөтөлбөрийг өөрчил. Та эдгээр хувьсагчийн нэр юу болохыг таах юмуу эсвэл API бичиглэл ашиглаж болно.

3. Том үсэг танихдаа isUpper –ийн оронд хэрэглэж болох Character анги дахь аргын нэрийг тааварлаж байна уу? MaxVariablesDemo хөтөлбөр өөрчлөхдөө isUpper-ийн оронд тэр аргаа ашиглаарай.

Оператор

Оператор нь нэг хоёр эсвэл гурав үйлдэгдэхүүн дээр үйлдэл хийдэг. Нэг үйлдэгдэхүүнтэй операторыг унари оператор гэнэ. Жишээлбэл: ‘+ +’ нь үйлдэгдэхүүний утгыг нэгээр нэмэгдүүлдэг **унари оператор** юм. Хоёр үйлдэгдэхүүнтэйг **бинари оператор** гэнэ. Жишээлбэл: ‘=’ бол баруун гар талын үйлдэгдэхүүний утгыг зүүн гарын үйлдэгдэхүүн рүү олгодог бинари оператор юм.

Эцэст нь гурван үйлдэгдэхүүнтэйг **тринари оператор** гэнэ. Java хэлэнд нэг тринари оператор байдаг нь if-else-ийн товчилсон хэлбэр ‘?;’ юм.

Унари оператор нь дагавар юмуу угтварын аль алийг хэрэглэдэг.

operator op
үйлдэгдэхүүний өмнө оператор байвал угтвар буюу prefix

op operator
операторын өмнө үйлдэгдэхүүн байвал дагавар буюу profix
Бүх бинари операторууд нь завсрын тэмдэглэгээ ашигладаг. Өөрөөр хэлбэл 2 үйлдэгдэхүүний дунд нь оператор нь байдаг.

op1 operator op2
Тринари оператор нь завсрынх байдаг. Энэ операторын гишүүн болгон нь 2 үйлдэгдэхүүний завсар байдаг.

op1 op2 : op3
Оператор нь үйлдэл гүйцэтгэхээс гадна утга буцаадаг. Утга буцах болон түүний төрөл нь тухайн үйлдэгхүүний төрөл, операторын төрлөөс хамаарна. Жишээлбэл: "+,-"зэрэг арифметик операторууд нь арифметик үйлдлийн үр дүн болох тоо буцаадаг. Арифметик операторуудын буцаах өгөгдлийн төрөл нь үйлдэгдэхүүний төрлөөс хамаарна. 2 бүхэл тоог нэмбэл үр дүн нь бүхэл тоо байх жишээтэй. Үйлдэгдэж гүйцэтгэхийг хариу бодож гаргана гэнэ.

Операторуудыг дараах ангилалд хуваадаг.

1. Арифметик оператор
2. Харьцуулах болон болзолт оператор
3. Шилжүүлэх болон логик оператор
4. Утга олгох оператор
5. Бусад

Арифметик оператор

Java хэлэнд бүхэл болон бутархай тоотой ажилладаг төрөл бүрийн арифметик операторууд байдаг. Үүнд +,-,*,/,%. Java хэлэнд арифметик бинари операторыг үзүүллээ.

Бинар арифметик оператор

Оператор	Хэрэглээ	Тайлбар
+	`op1 + op2`	op1, op2 –г нэмэх бас хэлхээг нэгтгэх
-	`op1 - op2`	op1 –с op2 –г хасах
*	`op1 * op2`	op1 болон op2 –г үржүүлэх
/	`op1 / op2`	op1 –г op2 –т хуваах
%	`op1 % op2`	op1 –г op2 –т хуваасны үлдэгдлийг тооцох

2 бүхэл 2 бутархай тоог тодорхойлж төрөл бүрийн арифметик үйлдэл гүйцэтгэх 5 оператор ашигласан жишээ програмыг харууллаа. Түүнчлэн энэ оператор нь хэлхээг нэгтгэхдээ "+" үйлдэл ашиглажээ. Арифметик үйлдлүүдийг дармалаар үзүүллээ.

```
public class ArithmeticDemo {
    public static void main(String[] args) {

        //a few numbers
        int i = 37;
        int j = 42;
        double x = 27.475;
        double y = 7.22;
        System.out.println("Variable values...");
        System.out.println("    i = " + i);
        System.out.println("    j = " + j);
        System.out.println("    x = " + x);
        System.out.println("    y = " + y);

        //adding numbers
        System.out.println("Adding...");
        System.out.println("    i + j = " + (i + j));
        System.out.println("    x + y = " + (x + y));

        //subtracting numbers
        System.out.println("Subtracting...");
        System.out.println("    i - j = " + (i - j));
        System.out.println("    x - y = " + (x - y));

        //multiplying numbers
        System.out.println("Multiplying...");
        System.out.println("    i * j = " + (i * j));
        System.out.println("    x * y = " + (x * y));

        //dividing numbers
        System.out.println("Dividing...");
        System.out.println("    i / j = " + (i / j));
        System.out.println("    x / y = " + (x / y));

        //computing the remainder resulting from dividing numbers
        System.out.println("Computing the remainder...");
        System.out.println("    i % j = " + (i % j));
        System.out.println("    x % y = " + (x % y));

        //mixing types
        System.out.println("Mixing types...");
        System.out.println("    j + y = " + (j + y));
```

```
            System.out.println("    i * x = " + (i * x));
        }
}
```

Энэ програмын гарц нь:

```
Variable values...
        i = 37
        j = 42
        x = 27.475
        y = 7.22
Adding...
        i + j = 79
        x + y = 34.695
Subtracting...
        i - j = -5
        x - y = 20.255
Multiplying...
        i * j = 1554
        x * y = 198.37
Dividing...
        i / j = 0
        x / y = 3.8054
Computing the remainder...
        i % j = 37
        x % y = 5.815
Mixing types...
        j + y = 49.22
        i * x = 1016.58
```

1 арифметик үйлдэл дотор бүхэл болон бутархай тоонуудыг үйлдэгдэхүүн хийхэд бутархай гарч байгааг анхаар. Энэ үйлдэл хийгдэхээс өмнө бүхэл тоо нь бутархай тоо болж байна (дамаар). Үйлдэгдэхүүний өгөгдлийн төрлөөс хамааран арифметик үйлдлийн буцаах өгөгдлийн төрлийг доорхи хүснэгтээс харууллаа. Үйлдэл хийхээс өмнө шаардлагатай үйлдлийг хийж байна.

Арифметик үйлдлийн үр дүнгийн төрлүүд

Өгөгдлийн төрөл	**Үйлдэгдэхүүний өгөгдлийн төрөл**
`long`	Аль нэг нь float юмуу double, ядаж нэг нь long байна.
`int`	Аль нэг нь float эсвэл double, аль нэг нь long байна.
`double`	Ядаж нэг нь double байна.
`float`	Ядаж нэг нь float, аль нэг нь double байна.

"+,-" үйлдлүүд нь бинари үйлдлээс гадна унари байна. Үүнийг доор үзүүллээ.

Унари арифметик оператор

оператор	Хэрэглээ	Тайлбар
+	+op	Нэгээр нэмэгдүүлэх
-	-op	Нэгээр хорогдуулах

Үйлдэгдэхүүнийг нэгээр нэмэгдүүлдэг "+ +", нэгээр хорогдуулдаг "- -" арифметикийн хураангуй 2 оператор байдаг. "+ +" юмуу "- -"-ийн аль аль нь үйлдэгдэхүүний өмнө, хойно байж болно. Угтвар хувилбартай ++op / --op нэмэгдүүлсэн эсвэл хорогдуулсаны дараа үйлдэгдэхүүн дээрээ үйлдлээ хийдэг. Дагавар хувилбар дээр op++ / op-- үйлдлээ гүйцэтгэсний дараа нь нэмэх хасах үйлдлийг хийдэг.

SortDemo гэсэн програм нь "++"-ийг 2,"--"-ийг нэг удаа ашигласан байна.

```
public class SortDemo {
    public static void main(String[] args) {
        int[] arrayOfInts = { 32, 87, 3, 589, 12, 1076,
                              2000, 8, 622, 127 };

        for (int i = arrayOfInts.length; --i >= 0; ) {
            for (int j = 0; j < i; j++) {
                if (arrayOfInts[j] > arrayOfInts[j+1]) {
                    int temp = arrayOfInts[j];
                    arrayOfInts[j] = arrayOfInts[j+1];
                    arrayOfInts[j+1] = temp;
                }
            }
        }

        for (int i = 0; i < arrayOfInts.length; i++) {
            System.out.print(arrayOfInts[i] + " ");
        }
        System.out.println();
    }
}
```

Энэ програм нь нэг төрлийн олон утга хадгалах чадвартай, тогтмол урттай бүтэц болон бүл рүү 10-н бүхэл тоо оруулаад дараа нь эрэмбэлдэг. arrayOfInts гэдэг бүлрүү 10 бүхэл тоо оруулаад үүсгэснийг дармалаар харууллаа. Бүлийнхээ гишүүдийнхээ тоог авахыг arrayOfInts.length –рүү хийж байна. Бүлийн нэгж гишүүнтэй харьцахдаа арга тэмдэглэхээ ашигладаг:arrayOfInts[index] энд index нь тухайн гишүүний бүл доторх байрлалыг харуулсан. Энэхүү дугаар 0-с эхэлж байгааг анхаар.

Програмын гарц нь өсөх дарааллаар байрлуулсан бүхэл тооны жагсаалт юм.

3 8 12 32 87 127 589 622 1076 2000

SortDemo програмын эрэмблэх 2 давхар давталтын гадна талын давталтыг удирдахдаа"--"-ийг хэрхэн ашигласныг авч үзье. Гаднах давталтыг удирдах хэллэгийг үзүүллээ.

```
for (int i = arrayOfInts.length; --i >= 0; ) {
    ...
}
```

For хэллэг бол давталтын хэллэг юм. Энэ кодын хамгийн чухал хэсэг нь "--" үйлдлийн утга 0-с их буюу тэнцүү байх үед давталтын хамгийн сүүлчийн алхмын i нь 0-тэй тэнцүү үед тохиолдоно гэдгийг "--"-ийн угтвар хувилбар харуулж байна. Хэрвээ бид дагавар нөхцлийг хэрэглэвэл энэ давталтын сүүлчийн хувилбар i нь -1 байх үед бүлийн index болж чадахгүй. Энэ програмд байгаа бусад 2 давталт нь "++"-ийн дагавар хувилбарыг ашиглаж байна. Энэ 2 тохиолдолд ямар хувилбар ашиглах нь ач холбогдолгүй. Учир нь энэ операторын буцах утга нь юунд ч хэрэглэхгүй үед дагавар хувилбарыг хэрэглэдэг. "++", "--"-ийг доорхь хүснэгтэнд нэгтгэн харууллаа.

Арифметик операторууд

Оператор	Хэрэглээ	Тайлбар
`++`	`op++`	Нэмэгдүүлэхийн өмнө үйлдэглэхүүнийг хийнэ
`++`	`++op`	Нэмэгдүүлсний дараа үйлдэгдэхүүнийг хийнэ
`--`	`op--`	Хорогдуулахын өмнө үйлдэгдэхүүнийг хийнэ
`--`	`--op`	Хорогдуулсны дараа үйлдэгдэхүүнийг хийнэ

Харьцуулах болон болзолт оператор

Харьцуулах оператор нь 2 утгыг харьцуулж, тэдгээрийн хоорондох харьцааг тогтооно.

Жишээлбэл: тэнцүү биш оператор нь "!=" харьцуулж буй 2 үйлдэгдэхүүн хоорондоо тэнцэхгүй нөхцөлд үнэн утга буцаана. Харьцуулах операторыг доорх хүснэгтээр харууллаа.

Харьцуулах оператор

Оператор	Хэрэглээ	Тайлбар
`>`	`op1 > op2`	`op1` нь `op2-с их` бол үнэн
`>=`	`op1 >= op2`	`op1` нь `op2-с их буюу тэнцүү` бол үнэн
`<`	`op1 < op2`	`op1 нь op2-с бага` бол үнэн
`<=`	`op1 <= op2`	`op1 нь op2-с бага буюу тэнцүү` бол үнэн

==	op1 == op2	op1, op2 нь тэнцүү бол үнэн
!=	op1 != op2	op1, op2 нь тэнцүү биш бол үнэн

3 бүхэл тоо тодорхойлж тэдгээрийг хооронд нь харьцуулах RelationalDemo жишээг доор үзүүллээ.

```
public class RelationalDemo {
    public static void main(String[] args) {

        //a few numbers
        int i = 37;
        int j = 42;
        int k = 42;
        System.out.println("Variable values...");
        System.out.println("    i = " + i);
        System.out.println("    j = " + j);
        System.out.println("    k = " + k);

        //greater than
        System.out.println("Greater than...");
        System.out.println("    i > j = " + (i > j)); //false
        System.out.println("    j > i = " + (j > i)); //true
        System.out.println("    k > j = " + (k > j)); //false;
                                                      //they are equal

        //greater than or equal to
        System.out.println("Greater than or equal to...");
        System.out.println("    i >= j = " + (i >= j)); //false
        System.out.println("    j >= i = " + (j >= i)); //true
        System.out.println("    k >= j = " + (k >= j)); //true

        //less than
        System.out.println("Less than...");
        System.out.println("    i < j = " + (i < j)); //true
        System.out.println("    j < i = " + (j < i)); //false
        System.out.println("    k < j = " + (k < j)); //false

        //less than or equal to
        System.out.println("Less than or equal to...");
        System.out.println("    i <= j = " + (i <= j)); //true
        System.out.println("    j <= i = " + (j <= i)); //false
        System.out.println("    k <= j = " + (k <= j)); //true

        //equal to
```

```
        System.out.println("Equal to...");
        System.out.println("    i == j = " + (i == j)); //false
        System.out.println("    k == j = " + (k == j)); //true

        //not equal to
        System.out.println("Not equal to...");
        System.out.println("    i != j = " + (i != j)); //true
        System.out.println("    k != j = " + (k != j)); //false
    }
}
```

Програмын гаралтанд

```
Variable values...
    i = 37
    j = 42
    k = 42
Greater than...
    i > j = false
    j > i = true
    k > j = false
Greater than or equal to...
    i >= j = false
    j >= i = true
    k >= j = true
Less than...
    i < j = true
    j < i = false
    k < j = false
Less than or equal to...
    i <= j = true
    j <= i = false
    k <= j = true
Equal to...
    i == j = false
    k == j = true
Not equal to...
    i != j = true
    k != j = false
```

Харьцуулах операторыг болзолт оператортой хослуулан илүү нарийн төвөгтэй шийдвэр гаргах илэрхийлэл байгуулах нь элбэг байдаг. Доорх хүснэгтэнд үзүүлснээр хосмог (бинари) 5 болон (инари) 1 нийт 6 харьцуулах оператор байдаг. (Java хэлэнд)

Болзолт оператор

Оператор	Хэрэглээ	Тайлбар
&&	op1 && op2	op1 болон op2 нь хоёул үнэн байвал үнэн
||	op1 || op2	op1 болон op2 -н аль нэг нь үнэн байвал үнэн
!	!op	op нь худлаа байвал үнэн
&	op1 & op2	op1 болон op2 нь хоёул boolean бөгөөд хоёул үнэн байвал үнэн, хэрэв үйлдэгдэхүүнүүд тоо бол AND үйлдлээр гүйцэтгээрэй
|	op1 | op2	op1 болон op2 нь хоёул boolean бөгөөд аль нэг нь үнэн байвал үнэн, хэрэв үйлдэгдэхүүнүүд тоо бол inclusive OR үйлдлээр гүйцэтгээрэй
^	op1 ^ op2	op1 болон op2 нь ялгаатай байвал үнэн

Тухайлбал: && оператор нь болзолт AND (ба) үйлдэл гүйцэтгэдэг. 2 харьцуулалт 2-лаа үнэн утга буцаж буй хэсгийг тогтооход 2 өөр харьцуулах операторoo && оператортой хамт хэрэглэнэ. Бүлийн (массив) индекс дээд доод хязгаар дотор оршин буй хэсгийг шалгахад доор үзүүллээ. Индексийн утга 0-с их бас урьдчилан тодорхойлсон NUM_ENTRIES утгаас бага байна уу гэдгийг шалгаж байна.

0 <= index && index < NUM_ENTRIES

Зарим тохиолдолд болзолт оператор нь 2-дох үйлдэгдэхүүнийг тооцоолохдоо орхиж болдог.

(numChars < LIMIT) && (...)

&& оператор нь зөвхөн 2 үйлдэгдэхүүн нь 2-л үнэн үед үнэн утга буцаадаг. Иймд numChars хувьсагчийн утга LIMIT тогтоохдоо тэнцүү буюу их үед && операторын зүүн талын үйлдэгдэхүүний утга худлаа болно. Иймд баруун үйлдэгдэхүүнийг тооцоолохгүйгээр && операторыг хариуг шууд авж болно. Урсгалаас хойш утга шийдлэх эсвэл тооцоолол хийх зэрэг үйлдлүүдийг баруун талын үйлдэгдэхүүн рүү хийж буй нөхцөлд ийм тохиолдол нь чухал ач холбогдолтой.

Шилжүүлэх болон логик оператор

Шилжүүлэх оператор нь эхний үйлдэгдэхүүнийг зүүн болон баруун тийш битээр нь шилжүүлэх үйлдэл гүйцэтгэдэг. Дараах хүснэгт нь Java хэлэнд байдаг шилжүүлэх операторуудыг харууллаа.

Шилжүүлэх оператор

Оператор	Хэрэглээ	Тайлбар
<<	op1 << op2	op2 –н хэмжээгээр op1 –г зүүн тийш нь битээр шилжүүлэн, баруун талыг 0 битээр дүүргэнэ
>>	op1 >> op2	op2 –н хэмжээгээр op1 –г баруун тийш нь битээр шилжүүлэн, зүүн талыг дээд битээр дүүргэнэ

>>>	op1 >>> op2	op2 –н хэмжээгээр op1 –г баруун тийш нь битээр шилжүүлэн, зүүн талыг 0 битээр дүүргэнэ

Бүх шилжүүлэх оператор нь хоёрдох үйлдэгдэхүүний тоогоор эхний үйлдэгдэхүүнийг битээр нь шилжүүлдэг. Аль тийш нь шилжүүлэхийг нь шилжүүлэх операторын сум нь өөрөө заадаг.

Жишээ нь, дараах хэллэг нь 13 гэсэн бүхэл тооны бит утгыг нь баруун тийш 1 байрлалаар шилжүүлж байна.

```
13 >> 1;
```

13 бүхэл тоо хоёртын тооллын системд 1101 байна. Баруун гар тийш 1 байрлал шилжүүлснээр үр дүнд нь 1101 тоо нь 110 болно (110 нь аравтын тооллын системд 6 гэсэн тоо байна.).

```
  1101     //13
& 1100     //12
------
  1100     //12
```

Зүүн талын тоог шилжүүлэхэд хоосон орнуудыг тэгээр “0” дүүргэдэг.

Доорх хүснэгт нь Java хэлэнд байдаг бит үйлдэгдэхүүн дээр хийгддэг 4 операторыг харууллаа.

Логик оператор

Оператор	Хэрэглээ	Үйл ажиллагаа
&	op1 & op2	Үйлдэгдэхүүнүүд нь number байвал бит AND, boolean байвал болзолт AND
|	op1 | op2	Үйлдэгдэхүүнүүд нь number байвал бит OR, boolean байвал болзолт OR
^	op1 ^ op2	Бит exclusive OR (XOR)
~	~op2	Бит үгүйсгэл

Үйлдэгдэхүүнүүд нь тоон төрөл байх үед, & оператор нь үйлдэгдэхүүнүүдийн харгалзах бит бүрд AND үйлдэл гүйцэтгэнэ. AND үйлдэл нь хэрэв үйлдэгдэхүүнүүдийн харгалзах 2 бит нь 1 байх үед үр дүн нь 1 болно.

Доорх хүснэгтээр харуулав.

Бит AND функц

op1 дахь бит	харгалзах op2 дахь бит	Үр дүн

0	0	0
0	1	0
1	0	0
1	1	1

13 болон 12 гэсэн утгуудад AND үйлдэл хийвэл (13 & 12 гэж), үр дүнд нь 12 гарна. Яагаад гэвэл 12 нь хоёртын тооллын системд 1100 байх ба 13 нь 1101 байна.

```
  1101     //13
& 1100     //12
------
  1100     //12
```

Хэрэв хоёр бит хоёулаа 1 бол, AND үйлдэл нь битийн үр дүнг 1 болгоно, эсвэл бусад тохиолдолд 0 болно. Иймд дээрх жишээний хоёр үйлдэгдэхүүний эхний хоёр орнууд нь 1 байгаа тул үр дүнгийн битүүд нь 1 болно. Үйлдэгдэхүүнүүдийн сүүлийн хоёр орнуудын аль нэг нь 0 эсвэл хоёулаа 0 байгаа тул үр дүн нь 0 болно.

Үйлдэгдэхүүн нь тоон төрөл байх үед нэг үйлдэгдэхүүн нь OR гэсэн үйлдэл гүйцэтгэнэ, мөн нөгөө үйлдэгдэхүүн нь XOR үйлдэл гүйцэтгэнэ. OR нь 2 битийн аль нэг нь 1 бол үр дүн нь 1 байна. Доорх хүснэгтээр харууллаа

Бит inclusive OR функц

op1 дахь бит	харгалзах op2 дахь бит	Үр дүн
0	0	0
0	1	1
1	0	1
1	1	1

XOR нь 2 бит нь ялгаатай бол үр дүн нь 1, эсрэг тохиолдолд 0 байна. Доорх хүснэгтээр харууллаа.

Бит exclusive OR (XOR) функц

op1 дахь бит	харгалзах op2 дахь бит	Үр дүн
0	0	0
0	1	1
1	0	1

1	1	0

Эцэст нь үгүйсгэх үйлдэгдэхүүн ‘~’ нь үйлдэгдэхүүний бит бүрийн утгыг 0 бол 1, 1 бол 0 болгож урвуулдаг. Жишээлбэл:1011 гэсэн тоонд ‘~’ үгүйсгэл хийвэл 0100 болно.

Бусад туйлуудын хооронд, бий үйлдэгдэхүүнүүд нь Boolean төлөвүүд дээр ажилладаг. Жишээлбэл: таны програмд цөөн тооны Boolean төлөв элдэв компонентоос бүрдсэн байдгийг илэрхийлж байдаг байх.(visible, draggable гэх мэт)

Бүх төлөвүүдийг хадгалахын тулд тусдаа Boolean хувьсагч тодорхойлсон нь дээр байдаг. Бүх төлөвүүдэд flag гэсэн нэг хувь тодорхойлогдож байдаг. Flag доторх бүх битүүд нь төлөвүүдийн нэгнийх нь одоогийн байдлыг төлөөлдөг. Дараа нь төлөвүүдийн утгыг тавихад болон авахад бит үйлдэгдэхүүнийг хэрэглээрэй.

Эхлээд програмынхаа төлөвүүдийг заасан тогтмолоос анхны утгыг олгоно. Эдгээр төлөвүүд зөвхөн нэг төлөвт хэрэглэгдэж буй(бит бүрийг) гэдгийг батлахын тулд төлөвүүдээ өөр өөр flag-с хувь тодорхойлж, битүүдийг нь төлөв бүрийн одоогийн байдлыг зааснаар тохиолдоно. Дараах жишээ код нь flag-ийн анхны утгыг 0 болгож байна. Энэ нь бүх төлөвүүдийн утга нь худлаа гэсэн үг(ямар ч бит нь утга аваагүй).

```
  static final int VISIBLE = 1;
static final int DRAGGABLE = 2;
static final int SELECTABLE = 4;
static final int EDITABLE = 8;

int flags = 0;
```

Ямар нэг зүйлийг харагддаг болгох үед VISIBLE төлөвт утга олгохдоо дараах хэллэгийг ашигладаг.

```
flags = flags | VISIBLE;
```

Харагдацыг нь шалгахын тулд дараахыг бичээрэй

```
  (flags & VISIBLE) == VISIBLE) {
      ...
}
```

Энд BitwiseDemo нэртэй бүрэн програм байна.

```
public class BitwiseDemo {

      static final int VISIBLE = 1;
      static final int DRAGGABLE = 2;
      static final int SELECTABLE = 4;
      static final int EDITABLE = 8;

      public static void main(String[] args) {
            int flags = 0;

            flags = flags | VISIBLE;
            flags = flags | DRAGGABLE;
```

```
            if ((flags & VISIBLE) == VISIBLE) {
                if ((flags & DRAGGABLE) == DRAGGABLE) {
                    System.out.println("Flags are Visible "
                                                + "and Draggable.");
                }
            }

            flags = flags | EDITABLE;

            if ((flags & EDITABLE) == EDITABLE) {
                System.out.println("Flags are now also Editable.");
            }
        }
}
```

Програмын гаралтанд:

Flags are Visible and Draggable.
Flags are now also Editable.

Утга олгох оператор

Нэг утгыг нөгөөд олгохдоо (=) гэх үндсэн үйлдэгдэхүүнийг хэрэглэнэ. MaxVariablesDemo програм нь өөрийн бүх локал хувьсагчаасаа (=) ашиглан анхны утгуудыг нь олгож байна.

```
//integers
byte largestByte = Byte.MAX_VALUE;
short largestShort = Short.MAX_VALUE;
int largestInteger = Integer.MAX_VALUE;
long largestLong = Long.MAX_VALUE;

//real numbers
float largestFloat = Float.MAX_VALUE;
double largestDouble = Double.MAX_VALUE;

//other primitive types
char aChar = 'S';
boolean aBoolean = true;
```

Java хэл нь арифметик, шилжүүлэх, эсвэл бит үйлдлүүдийг утга олгох үйлдэлтэй хамтруулсан нэг үйлдлээр биелүүлэхийг зөвшөөрсөн цөөн тооны товч олгуур операторуудтай.

```
i = i + 2;
```

+= үйлдэл нь ашигласнаар уг хэлийг товчилж болно.

```
i += 2;
```

Дээрх 2 мөрөнд яг адил үйлдэл гүйцэтгэж байна. Доорх хүснэгт нь товчхон олгуур үйлдэгдэхүүнийг яг дүйцэхүйц үйлдэгдэхүүнтэй нь харгалзан жагсаалаа.

Утга олгох товч оператор

Оператор	Хэрэглээ	Адил шинж
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2
&=	op1 &= op2	op1 = op1 & op2
|=	op1 |= op2	op1 = op1 | op2
^=	op1 ^= op2	op1 = op1 ^ op2
<<=	op1 <<= op2	op1 = op1 << op2
>>=	op1 >>= op2	op1 = op1 >> op2
>>>=	op1 >>>= op2	op1 = op1 >>> op2

Бусад оператор

Java хэлэнд мөн дараахь операторуудыг хэрэглэдэг.

Оператор	Тайлбар
`?:`	Хэрэв-эсвэл хэллэгийн товч
`[]`	Бүлийг зарлах, бүлийг үүсгэх, болон бүлийн гишүүдэд хандахад хэрэглэнэ
`.`	Нарийвчилсан нэрийг бүрдүүлэхэд хэрэглэнэ
`(` *params* `)`	Хэмжигдэхүүнүүдийн таслалаар тусгаарласан жагсаалтыг хязгаарлана
`(` *type* `)`	Утгыг тодорхойлогдсон төрөл рүү хөрвүүлнэ
`new`	Шинэ обьект эсвэл шинэ бүлийг үүсгэнэ
`instanceof`	Үйлдэгдэхүүнүүдийн эхнийх нь хоёрдохынхоо төл мөн эсэхийг тогтооно

Дүгнэлт

Арифметик оператор

Бинар арифметик оператор

Оператор	Хэрэглээ	Тайлбар
`+`	`op1 + op2`	op1, op2 –г нэмэх бас хэлхээг нэгтгэх
`-`	`op1 - op2`	op1 –с op2 –г хасах
`*`	`op1 * op2`	op1 болон op2 –г үржүүлэх
`/`	`op1 / op2`	op1 –г op2 –т хуваах
`%`	`op1 % op2`	op1 –г op2 –т хуваасны үлдэгдлийг тооцох

Арифметик үйлдлийн үр дүнгийн төрлүүд

Өгөгдлийн төрөл	Үйлдэгдэхүүний өгөгдлийн төрөл
`long`	Аль нэг нь float юмуу double, ядаж нэг нь long байна.
`int`	Аль нэг нь float эсвэл double, аль нэг нь long байна.
`double`	Ядаж нэг нь double байна.
`float`	Ядаж нэг нь float, аль нэг нь double байна.

Унари арифметик оператор

оператор	Хэрэглээ	Тайлбар
+	+op	Нэгээр нэмэгдүүлэх
-	-op	Нэгээр хорогдуулах

Арифметик операторууд

Оператор	Хэрэглээ	Тайлбар
++	op++	Нэмэгдүүлэхийн өмнө үйлдэглэхүүнийг хийнэ
++	++op	Нэмэгдүүлсний дараа үйлдэгдэхүүнийг хийнэ
--	op--	Хорогдуулахын өмнө үйлдэгдэхүүнийг хийнэ
--	--op	Хорогдуулсны дараа үйлдэгдэхүүнийг хийнэ

Харьцуулах болон болзолт оператор

Харьцуулах оператор

Оператор	Хэрэглээ	Тайлбар
>	op1 > op2	op1 нь op2-с их бол үнэн
>=	op1 >= op2	op1 нь op2-с их буюу тэнцүү бол үнэн
<	op1 < op2	op1 нь op2-с бага бол үнэн
<=	op1 <= op2	op1 нь op2-с бага буюу тэнцүү бол үнэн
==	op1 == op2	op1, op2 нь тэнцүү бол үнэн
!=	op1 != op2	op1, op2 нь тэнцүү биш бол үнэн

Болзолт оператор

Оператор	Хэрэглээ	Тайлбар
&&	op1 && op2	op1 болон op2 нь хоёул үнэн байвал үнэн
||	op1 || op2	op1 болон op2 -н аль нэг нь үнэн байвал үнэн
!	!op	op нь худлаа байвал үнэн
&	op1 & op2	op1 болон op2 нь хоёул boolean бөгөөд хоёул үнэн байвал үнэн, хэрэв үйлдэгдэхүүнүүд тоо бол AND үйлдлээр гүйцэтгээрэй
|	op1 | op2	op1 болон op2 нь хоёул boolean бөгөөд аль нэг нь үнэн байвал үнэн, хэрэв үйлдэгдэхүүнүүд тоо бол inclusive OR үйлдлээр

		гүйцэтгээрэй
^	op1 ^ op2	op1 болон op2 нь ялгаатай байвал үнэн

Шилжүүлэх болон логик оператор

Шилжүүлэх оператор

Оператор	Хэрэглээ	Тайлбар
<<	op1 << op2	op2 –н хэмжээгээр op1 –г зүүн тийш нь битээр шилжүүлэн, баруун талыг 0 битээр дүүргэнэ
>>	op1 >> op2	op2 –н хэмжээгээр op1 –г баруун тийш нь битээр шилжүүлэн, зүүн талыг дээд битээр дүүргэнэ
>>>	op1 >>> op2	op2 –н хэмжээгээр op1 –г баруун тийш нь битээр шилжүүлэн, зүүн талыг 0 битээр дүүргэнэ

Логик оператор

Оператор	Хэрэглээ	Үйл ажиллагаа
&	op1 & op2	Үйлдэгдэхүүнүүд нь number байвал бит AND, boolean байвал болзолт AND
|	op1 | op2	Үйлдэгдэхүүнүүд нь number байвал бит OR, boolean байвал болзолт OR
^	op1 ^ op2	Бит exclusive OR (XOR)
~	~op2	Бит үгүйсгэл

Бит AND функц

op1 дахь бит	харгалзах op2 дахь бит	Үр дүн
0	0	0
0	1	0
1	0	0
1	1	1

Бит inclusive OR функц

op1 дахь бит	харгалзах op2 дахь бит	Үр дүн
0	0	0
0	1	1
1	0	1
1	1	1

Бит exclusive OR (XOR) функц

op1 дахь бит	харгалзах op2 дахь бит	Үр дүн
0	0	0
0	1	1
1	0	1
1	1	0

Утга олгох оператор

Утга олгох товч оператор

Оператор	Хэрэглээ	Адил шинж
+=	`op1 += op2`	`op1 = op1 + op2`
-=	`op1 -= op2`	`op1 = op1 - op2`
*=	`op1 *= op2`	`op1 = op1 * op2`
/=	`op1 /= op2`	`op1 = op1 / op2`
%=	`op1 %= op2`	`op1 = op1 % op2`
&=	`op1 &= op2`	`op1 = op1 & op2`
|=	`op1 |= op2`	`op1 = op1 | op2`
^=	`op1 ^= op2`	`op1 = op1 ^ op2`
<<=	`op1 <<= op2`	`op1 = op1 << op2`
>>=	`op1 >>= op2`	`op1 = op1 >> op2`
>>>=	`op1 >>>= op2`	`op1 = op1 >>> op2`

Бусад оператор

Оператор	Тайлбар
`?:`	Хэрэв-эсвэл хэллэгийн товч
`[]`	Бүлийг зарлах, бүлийг үүсгэх, болон бүлийн гишүүдэд хандахад хэрэглэнэ
`.`	Нарийвчилсан нэрийг бүрдүүлэхэд хэрэглэнэ
`(` *params* `)`	Хэмжигдэхүүнүүдийн таслалаар тусгаарласан жагсаалтыг хязгаарлана
`(` *type* `)`	Утгыг тодорхойлогдсон төрөл рүү хөрвүүлнэ
`new`	Шинэ обьект эсвэл шинэ бүлийг үүсгэнэ
`instanceof`	Үйлдэгдэхүүнүүдийн эхнийх нь хоёрдохынхоо төл мөн эсэхийг тогтооно

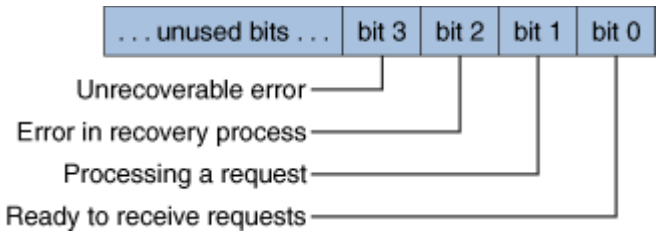
Асуулт болон дасгал

Асуулт

1. Дараах код мөрийг авч үзье:
   ```
   arrayOfInts[j] > arrayOfInts[j+1]
   ```
   Кодод ямар операторууд агуулав?
2. Дараах код мөрийг авч үзье:
   ```
   int i = 10;
   int n = i++%5;
   ```
   a. Код ажилласны дараа i болон n –н утгууд ямар болох вэ?
   b. Хэрэв утгыг нэмэгдүүлэгч дагавар (postfix) операторын (i++) оронд, угтвар (prefix) хувилбарыг (++i) хэрэглэвэл эцсийн үр дүнд нь i болон n нь ямар утгатай болох вэ?
3. Дараах код мөр ажилласны дараа i –н утга ямар болох вэ?
   ```
   int i = 8;
   i >>=2;
   ```
4. Дараах код мөр ажилласны дараа i –н утга ямар болох вэ?
   ```
   int i = 17;
   i >>=1;
   ```

Дасгал

1. Хөвөх таслалтай тоо тэг байгаа эсэхийг шалгадаг програм бич. (Сануулга: Хөвөх таслалтай тоонуудыг яг тэнцүүлэхэд хүнд бол ерөнхийдөө == тэнцэтгэл операторыг хэрэглээгүй нь дээр шүү)
2. Euro –р өгөгдсөн тоонуудыг US долларлуу тэнцүү тооцоолдог програм бич. Арилжааны тарифаар 1 US долларт ойролцоогоор 0.781162 Euro оноогдож байгаа. Хэрэв програмын харуулах тоонуудын форматыг өөрөө удирдах бол Formatting Numbers With Custom Formats сэдэв дэх DecimalFormat ангийг хэрэглэж болно.
3. Дан бүхэл тоон дахь битүүдийг ашиглан доор харуулсан true/false өгөгдлийг төлөөлөн програм бич.



Status нэртэй хувьсагчийг програмдаа оруулан, утгыг нь хэвлэх програм бичээрэй. Жишээ нь: Хэрэв Status нь 1 бол програм нь доорх шиг зүйл хэвлэдэг байх:

Ready to receive requests

a. Кодоо харуул
b. Status нь 8 байх үед гарц ямар байх вэ?
c. Status нь 7 байх үед гарц ямар байх вэ?

Илэрхийллүүд, хэллэгүүд, мөн блокууд

Програмын блокуудыг барихад хувьсагчууд мөн операторууд чухал байдгийг бид үзсэн. Тооцоолох мөн утга буцаах үйлдэл гүйцэтгэдэг илэрхийллийн кодын хэсгийн хувьсагчид болон операторуудыг та нэгтгэж болно. Тодорхой илэрхийллүүд нь үр дүнгийн нэгж болох хэллэгүүдийг бий болгодог. Та {} энэ хаалтыг ашиглан хэллэгүүдийн кодын блокыг үүсгэж болно.

Илэрхийлэл

Илэрхийлэл нь програмын ажлыг биелүүлдэг. Бусад зүйлд илэрхийлэл нь тооцоолох, хувьсагчийн утга тогтоох мөн програмын биелэлтийн урсгалыг удирдахад тусалдаг.

Илэрхийлэлд хоёр ажил хийгддэг: Тооцоолох мөн түүний үр дүнг буцаадаг илэрхийллийн элементүүдээр тодорхойлогдсон тооцооллыг биелүүлэх ажил юм.

Тодорхойлолт: Илэрхийлэл нь энгийн утга боддог хувьсагчид, операторууд мөн аргын дуудлагын бүлэг юм.

Өмнөх хэсэгт бид операторууд утга буцаадаг,мөн оператор нь илэрхийлэлд хэрэглэгддэг гэдгийг хэлэлцсэн. Энэ хэсэгхэн програмын жагсаалт нь илэрхийллийг үзүүлсэн байна.

```
...
//other primitive types
char aChar = 'S';
boolean aBoolean = true;

//display them all
System.out.println("The largest byte value is "
                + largestByte);
...

if (Character.isUpperCase(aChar)) {
    ...
}
```

Ямар ч илэрхийлэл нь утга буцаах гэх мэт үйлдэл гүйцэтгэдэг.Доорх хүснэгтээс харна уу.

MaxVariablesDemo –н зарим илэрхийллүүд

Илэрхийлэл	Үйлдэл	Буцаах утга
`aChar = 'S'`	Assign the character 'S' to the character variable `aChar`	The value of `aChar` after the assignment ('S')
`"The largest byte value is " + largestByte`	Concatenate the string `"The largest byte value is "` and the value of `largestByte` converted to a string	The resulting string: `The largest byte value is 127`
`Character.isUpperCase(aChar)`	Call the method `isUpperCase`	The return value of the method: `true`

Илэрхийлэлд хэрэглэгдэж байгаа элементүүдээс хамааран илэрхийллийн ямар өгөгдлийн төрөлтэй утга буцах нь тогтоогдоно. aChar = ‘S’ илэрхийлэл нь тэмдэгт буцаана. Яагаад гэвэл хувиарлагч оператор нь операнд, aChar мөн ‘S’ тэмдэгтүүдтэй адил өгөгдлийн төрөлтэй утга буцаадаг. Та бусад илэрхийллүүдээс харж болно. Илэрхийлэл нь булын мөн хэлхээний гэх мэт утгыг буцаадаг.

Java програмын хэл нь нийлмэл илэрхийллүүд мөн хэллэгүүд байгуулахыг зөвшөөрдөг ба тэдгээр нь өөр өгөгдлийн төрлийг сонгодог илэрхийллийн нэг хэсгээр хангагдсан ялгаатай арай жижиг илэрхийллүүд байна. Нийлмэл илэрхийллийн жишээ:

```
x * y * z
```

Дээрх жишээнд илэрхийллийн бодогдох дараалал чухал биш. Яагаад гэхээр үржих үйлдэл нь дарааллаас тусгаар байдаг ба хаанаас нь ч эхэлж бодсон үр дүн нь ижил гарна. Хэдий ийм боловч энэ нь бүх илэрхийллүүдийн хувьд үнэн биш байдаг. Жишээлбэл: доорх илэрхийлэл нь нэмэх үйлдэл ба хуваах үйлдлийн аль нь эхэлж биелэгдэхээс хамаарч өөр өөр үр дүн өгнө.

```
x + y / 100     //ambiguous
```

Та илэрхийллийн аль үйлдэл нь эхэлж бодогдохыг ( ) хаалт ашиглан зааж өгөх хэрэгтэй. Жишээлбэл: өмнөх илэрхийллийг тодорхой болгоё.

```
(x + y)/ 100    //unambiguous, recommended
```

Хэрвээ та нийлмэл илэрхийлэл биелүүлэх үйлдлээ тодорхой зааж, үзүүлж чадахгүй байгаа бол илэрхийлэл доторх операторуудынхаа биелэгдэх дарааллыг шийдэмгий тогтоож өгөх хэрэгтэй. Операторуудын хамгийн өндөр зэрэглэлтэй нь эхэлж бодогддог. Жишээ нь, хуваах оператор нь нэмэх операторааас өндөр зэрэглэлтэй. Тэгэхээр дараах хэллэгүүд нь адил юм.

```
x + y / 100
x + (y / 100) //unambiguous, recommended
```

Та нийлмэл илэрхийлэл бичиж байхдаа, аль оператор эхэлж бодогдохыг ( ) хаалт ашиглан тодорхой зааж өгөх хэрэгтэй. Энэ нь танд кодыг хялбар бичихэд дөхөм болно.

Доорх хүснэгт нь Java платформ дахь операторуудын биелэгдэх дарааллыг үзүүлсэн ба операторууд хүснэгтэнд биелэгдэх дарааллаараа жагссан байна. Хүснэгтийн хамгийн дээр байгаа оператор нь бусдаас өндөр зэрэглэлтэй байна. Өндөр зэрэглэлтэй операторууд нь харьцангуй доогуур зэрэглэлтэй операторуудаас өмнө биелэгдэнэ. Нэг шугаман дээр байгаа операторууд нь ижил зэрэглэлтэй. Илэрхийлэлд ижил зэрэглэлтэй операторууд байвал тэдгээрийн аль нь эхэлж биелэгдэхийг зохицуулдаг дүрэм байдаг. Бүх хоёртын операторуудаас гадна хувиарлах операторууд зүүнээс баруун тийш бодогддог.

Операторын биелэгдэх дараалал

postfix operators	*expr*++ *expr*--
unary operators	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
conditional	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Хэллэг

Хэллэгүүд нь үндсэн хэлнийхээ өгүүлбэр зүйтэй ойролцоогоор ижил байдаг. Хэллэг нь үр дүнгийн бүтэн нэгжээр бүтдэг. Доорх илэрхийлэлийн төрлүүдийн хойно (;) үүнийг тавьж төгсгөнө.

- Хувиарлах илэрхийллүүд
- Any use of ++ or --
- Аргын дуудлага
- Обьект үүсгэх илэрхийллүүд

Эдгээр хэллэгүүдийн төрлийг илэрхийллийн хэллэгүүд гэж нэрлэнэ. Энд хэсэг илэрхийллийн хэллэгүүдийн жишээ байна:

```
aValue = 8933.234;                      //assignment statement
aValue++;                               //increment       "
System.out.println(aValue);             //method call     "
Integer integerObject = new Integer(4); //object creation "
```

Эдгээр илэрхийллийн хэллэгүүдийн төрлүүд дээр нэмэгдээд хоёр өөр хэллэгүүдийн төрлүүд байна. Хэллэгийн мэдэгдэл энэ нь хувьсагч зарладаг. Та хэллэгийн мэдэгдлийн жишээг өмнө нь олон харж байсан.

```
double aValue = 8933.234;             //declaration statement
```

Дараагийн төрөл нь хэллэгийн удирдлагын урсгал юм. Энэ нь аль хэллэг биелэгдэх дарааллыг зохицуулж өгдөг. for давталт мөн if хэллэгүүд нь хоёулаа хэллэгийн удирдлагын урсгалын жишээ юм.

Блок

Блок бол тэгийн групп эсвэл {} энэ хаалтан доторх хаа ч хэрэглэгддэг зөвшөөрөгдсөн энгийн хэллэг байна. Доорх жагсаалт нь тус бүр энгийн хэллэг агуулсан програмын хоёр блокыг үзүүлсэн байна:

```
if (Character.isUpperCase(aChar)) {
    System.out.println("The character " + aChar
                       + " is upper case.");
} else {
    System.out.println("The character " + aChar
                       + " is lower case.");
}
```

Дүгнэлт

Илэрхийлэл нь энгийн утга боддог хувьсагчид, операторууд мөн аргын дуудлагын бүлэг юм. Та нийлмэл илэрхийллүүдэд хамрагдсан бүх операторуудаар илэрхийллүүдийг нэгтгэн нийлмэл илэрхийлэл бичиж болно. Та нийлмэл илэрхийлэл бичиж байхдаа, аль оператор эхэлж бодогдохыг ( ) хаалт ашиглан тодорхой зааж өгөх хэрэгтэй.

Хэрвээ та хаалт ашиглахыг сонгоогүй бол, Java платформ операторын биелэгдэх дарааллаар нийлмэл илэрхийллийг бодно. Java платформ дахь операторуудын биелэгдэх дарааллыг хүснэгтэнд үзүүлсэн байгаа.

Хэллэг нь үр дүнгийн бүтэн нэгжээс бүрдэх бөгөөд (;) – ээр төгссөн байна. Хэллэгийг гурван төрөлд хуваана: илэрхийллийн хэллэгүүд, хэллэгүүдийн мэдэгдэл, мөн хэллэгүүдийн удирдлагын урсгал.

Та тэгийн групп эсвэл олон хэллэгүүдийг хамтад нь { } хаалт ашиглан блок үүсгэж болно. Хэдийгээр бид блок үүсгэхэд { } хаалт ашиглахыг санал болгосон ч гэсэн хэрвээ блок дотор зөвхөн ганц хэллэг байвал энэ хаалт хэрэггүй.

Асуулт болон дасгал

Асуулт

1. i –н төрлийг int гэвэл, дараах илэрхийллүүд дэх өгөгдлийн төрлүүд юу вэ?

```
i > 0
i = 0
i++
(float)i
i == 0
"aString" + i
```

2. Дараах илэрхийллийг авч үзье.

```
i--%5>0
```

   a. i –н утга эхлээд 10 байсан гэвэл, дээрх илэрхийллийн үр дүн ямар байх вэ?
   b. Дээрх илэрхийллийг уншихад хялбар болгон өөрчил, гэхдээ үр дүнг нь адилхан байлга.

Дасгал

1. Дээрх асуултны хариултыг бататгасан програм бич.

Хэллэгүүдийн удирдлагын урсгал

Та програм бичихдээ хэллэгүүдээ файлдаа бичиж өгнө. Үүний дараа интерпретер эдгээр хэллэгүүдийг зүүнээс баруун, дээрээс доош харагдах байдлаар хэллэгүүдийн удирдлагын урсгалгүйгээр биелүүлнэ. Та хэллэгүүдийн удирдлагын урсгалыг програмдаа ашиглахдаа хэллэгүүдийг нөхцөлтэйгээр биелүүлэх, дахин дахин хэллэгүүдийн блокыг биелүүлэх мөн өөр аргаар энгийн, дараалсан удирдлагын урсгалыг өөрчилж болно. Жишээлбэл: доорх кодын хэсгийг харна уу. if хэллэг нь Character.is.UpperCase (aChar) – ын утгыг System.out.println хэллэгийн тусламжтайгаар нөхцөлтэйгээр биелүүлнэ:

```
char aChar;
...
if (Character.isUpperCase(aChar)) {
   System.out.println("The character " + aChar
                   + " is upper case.");
}
```

Java програмын хэл нь доорх хүснэгтэн жагсаалтан дахь хэллэгүүдийн удирдлагын урсгалаар хангагдсан байдаг.

Хэллэгүүдийн удирдлагын урсгал

Хэллэгийн төрөл	Түлхүүр үг
looping	`while`, `do-while`, `for`
decision making	`if-else`, `switch-case`
exception handling	`try-catch-finally`, `throw`
branching	`break`, `continue`, `label:`, `return`

Та доорх хэсгээс хэллэгүүдийн удирдлагын урсгалын тодорхойлтыг харж болно.

```
control flow statement details {
    statement(s)
}
```

Хэрвээ блок нь зөвхөн нэг хэллэг агуулж байвал { } энэ хаалт шаардлагагүй. Тийм боловч бид энэ { } хаалтыг хэрэглэхийг танд зөвлөж байна. Яагаад гэвэл код бичихэд хялбар мөн кодыг хувиргах үеийн алдаанаас сэрэмжилдэг.

while болон do – while хэллэгүүд

Та while хэллэгийг ашиглавал, энэ нь нөхцөл үнэн болох хүртэл хэллэгийн блокыг нөхцөлтэйгээр биелүүлнэ. while хэллэгийн өгүүлбэр зүй нь:

```
while (expression) {
    statement
}
```

Эхлээд while хэллэг нь илэрхийллийг бодоод булын утга буцаах ёстой. Хэрвээ илэрхийлэл үнэн утга буцаавал while хэллэг нь блок дахь хэллэгүүдийг бодно. while хэллэг нь илэрхийлэл худал утга буцаах хүртэл илэрхийлэл ба блокыг биелүүлэх болно.

Доорх while хэллэг ашигласан жишээг WhileDemo гэж нэрлэе.

```
public class WhileDemo {
    public static void main(String[] args) {

        String copyFromMe = "Copy this string until you " +
                            "encounter the letter 'g'.";
        StringBuffer copyToMe = new StringBuffer();

        int i = 0;
        char c = copyFromMe.charAt(i);

        while (c != 'g') {
            copyToMe.append(c);
            c = copyFromMe.charAt(++i);
        }
```

```
        System.out.println(copyToMe);
    }
}
```

Java програмын хэл нь while хэллэгтэй төстэй do – while хэмээх өөр нэгэн хэллэгээр хангаж өгсөн. do – while хэллэгийн өгүүлбэр зүй нь:

```
do {
    statement(s)
} while (expression);
```

Энэ хэллэг нь давталтын эхний илэрхийллийг бодохын оронд сүүлийн илэрхийллийг боддог бөгөөд ийм маягаар do – while хэллэг нь нэг л удаа өчүүхэн зүйлийг биелүүлдэг. Энд өмнөх програмыг do – while хэрэглэн дахин бичсэн байна.

```
public class DoWhileDemo {
    public static void main(String[] args) {

        String copyFromMe = "Copy this string until you " +
                            "encounter the letter 'g'.";
        StringBuffer copyToMe = new StringBuffer();

        int i = 0;
        char c = copyFromMe.charAt(i);

        do {
            copyToMe.append(c);
            c = copyFromMe.charAt(++i);
        } while (c != 'g');
        System.out.println(copyToMe);
    }
}
```

for хэллэг

for хэллэг нь утгуудын хэмжээг дахин давтах арга замыг нээж өгсөн. Энэ нь 5,0 шиг ерөнхий төрөлтэй бөгөөд та энгийн бүлүүд мөн коллекцуудыг дахин давтахад ашиглаж болно. for хэллэгийн ерөнхий төрөл нь:

```
for (initialization; termination; increment) {
    statement(s)
}
```

Initialization илэрхийлэл нь давталтын эхний илэрхийлэл ба энэ нь давталтын эхэнд ганц удаа биелэгдэнэ. termination илэрхийлэл нь давталт дуусахад тодорхойлогдоно. Илэрхийлэл худал бодсон үед давталт дуусна. Эцэст нь, increment илэрхийлэл нь давталтын давтагдах бүрт дуудагдаж байдаг. Эдгээр бүх бүрэлдэхүүн хэсгүүд нь туслах чанарын үйлдлүүд юм. Үнэн хэрэгтээ, хязгааргүй давталт бичихдээ та тэдгээр гурван илэрхийллийг орхих хэрэгтэй.

```
for ( ; ; ) {    //infinite loop
    ...
}
```

for давталт нь үргэлж бүл эсвэл хэлхээн дэх тэмдэгтийн элементүүдийг дахин давтахад ашиглагддаг. Доорх ForDemo нэртэй энгийн жишээнд for хэллэгийг бүл мөн тэднийг хэвлэх элементүүдийг дахин давтахад хэрэглэж байна.

```
public class ForDemo {
    public static void main(String[] args) {
        int[] arrayOfInts = { 32, 87, 3, 589, 12,
                              1076, 2000, 8, 622, 127 };

        for (int i = 0; i < arrayOfInts.length; i++) {
            System.out.print(arrayOfInts[i] + " ");
        }
        System.out.println();
    }
}
```

Програмын гарц нь: `32 87 3 589 12 1076 2000 8 622 127`.

Та for давталтын initialization илэрхийлэлтэй хамт локаль хувьсагч зарлаж болно. Хувьсагчийн хэмжээг for хэллэгээр зохицуулагддаг termination мөн increment илэрхийлэлүүдэд хэрэглэгддэг блокын төгсгөлийн мэдэгдлээр өргөтгөдөг. Хэрэв хувьсагч нь for давталтыг давталтын гадна талд хэрэггүй гэж үзвэл, энэ нь initialization илэрхийлэл дэх хамгийн сайн хувьсагчийн зарлагаа болно. i, j мөн k нь үргэлж for давталтын удирдлаганд ашиглагддаг. Эдгээрийг for давталтын initialization илэрхийлэлтэй хамт зарлах ба энэ нь тэдний амьдралын хором мөн алдаануудын цөөрөлтийг хэмждэг.

Түүвэрүүд болон бүлүүдийг нэмэгдүүлэн давтах нь

5.0 д онцгойлон түүвэрүүд болон бүлүүдэд шинэ илэрхийлэл үүссэн. Энд зарим кодыг нь ForEachDemo -аас авсан.Энэ нь өмнөх кодын хэсгийг авсантай ижил зүйл (ForDemo -аас авсан).

```
public class ForEachDemo {
    public static void main(String[] args) {
        int[] arrayOfInts = { 32, 87, 3, 589, 12,
                              1076, 2000, 8, 622, 127 };

        for (int element : arrayOfInts) {
            System.out.print(element + " ");
        }
        System.out.println();
    }
}
```

Энэ илэрхийллээс та яг үүн шиг өмнөх хэсгийг уншиж болно.

For each `int element` *in* `arrayOfInts`...

Нэмэгдүүлэх мөчид илэрхийллийн гүйх үе нь түүврүүдийг ашиглах юм. (Collection интерфейсийг хэрэгжүүлэх ангиуд.) Энд хуучин илэрхийлэл нь түүврийн турш давтана.

```
//This is ugly. Avoid it by using enhanced for!
void cancelAll(Collection<TimerTask> c) {
    for (Iterator<TimerTask> i = c.iterator(); i.hasNext(); )
        i.next().cancel();
}
```

<TimerTask> гэсэн хачин кодод одоохондоо санаа зовсны хэрэггүй. Та үүний талаар сурах болно. Мөн Generics болон Collections түүврүүдийн тухай сурах болно. Санаа нь for давталтыг хэрэглэснээр илэрхийллийг нэмэгдүүлэхээс зугтаж болно.

```
//This is much prettier.
void cancelAll(Collection<TimerTask> c) {
    for (TimerTask t : c)
        t.cancel();
}
```

Nest давталтын үед илэрхийллийг нэмэгдүүлэх нь илүү аятайхан байдаг. Яагаад гэвэл хэрэггүй кодоос зугтаж болно. Жишээ нь :

```
for (Suit suit : suits) {
    for (Rank rank : ranks)
        sortedDeck.add(new Card(suit, rank));
}
```

Харамсалтай нь илэрхийлэл нэмэгдүүлэх нь бүх газарт ажилладаггүй.Хэрэв танд массивийн индексэд нэвтрэх хэрэгтэй бол нэмэгдүүлэх нь таны төлөө ажиллахгүй. Хэдий тийм боловч илэрхийлэл нэмэгдүүлэх нь бясаануудыг цөөлж, таны кодыг цэвэрхэн харагдуулна.

if / else хэллэгүүд

if хэллэг нь хэсэг хэмжүүр дээр суурилагдсан, бусад хэллэгүүдийг сонгон биелүүлэх боломжийг танд олгоно. Жишээ нь: DEBUG хэмээн нэрлэгдэх булын хувьсагчийн утганд суурилсан, мэдээллийг хэвлэдэг програм гэж бодъё. Хэрвээ DEBUG үнэн бол таны програм хувьсагчийн мөн x – ийн гэх мэт утгыг хэвлэнэ. Өөрөөр хэлбэл: таны програм хэвийн ажиллаж байна. Кодын хэсэг нь доорх шиг харагдаж магадгүй:

```
if (DEBUG) {
    System.out.println("DEBUG: x = " + x);
}
```

Энэ бол if хэллэгийн жирийн хувилбар юм. Блок нь нөхцөл үнэн үед биелэгддэг if хэллэгээр зохицуулагдана. Ерөнхийдөө, if хэллэгийн энгийн төрөл доорх маягаар бичигдэнэ:

```
if (expression) {
    statement(s)
}
```

Хэрвээ таны илэрхийлэл худал гарвал хэллэгүүдийн өөр тохиргоо танд хэрэгтэй. Энэ нь else хэллэг юм. Өөр нэгэн жишээг авч үзье. Хэрэглэгч сэрэмжлүүлэх цонхонд ОК товч эсвэл өөр товч дарах эсэхээс хамааран таны програм ялгаатай өөр үйлдлүүдийг хийх хэрэгтэй болно. таны програм үүнийг if хэллэг болон else хэллэгийн хамтаар хийж чадна.

```
. . .
   //response is either OK or CANCEL depending
   //on the button that the user pressed
   . . .
if (response == OK) {
   //code to perform OK action
} else {
   //code to perform Cancel action
}
```

else блок нь if хэсэг худал үед биелэгдэнэ. else хэллэгийн өөр нэгэн төрөл болох else if хэллэг нь өөр илэрхийлэлд суурилан хэллэг биелүүлдэг. if хэллэгтэй дагалдах else – ийн тоо нь зөвхөн ганц байна. Дараах жишээ нь тестийн үнэлгээний утганд суурилсан IfElseDemo хэмээх үнэлгээ тогтоох програм юм. 90% ба түүнээс дээш A, 80% ба түүнээс дээш B гэх мэт:

```
public class IfElseDemo {
   public static void main(String[] args) {

      int testscore = 76;
      char grade;

      if (testscore >= 90) {
         grade = 'A';
      } else if (testscore >= 80) {
         grade = 'B';
      } else if (testscore >= 70) {
         grade = 'C';
      } else if (testscore >= 60) {
         grade = 'D';
      } else {
         grade = 'F';
      }
      System.out.println("Grade = " + grade);
   }
}
```

Програмын гарц нь:

```
Grade = C
```

Таны testscore –ийн утга нь нийлмэл if хэллэг дэх илэрхийллүүдийн нэгээс нь л их байвал хангалттай: 76 >= 70, 76 >=60. runtime system нь if хэллэгийн үндсэн нөхцлүүдийг бодохгүйгээр нэг удаагийн нөхцлөөр хангагдсан зохистой хэллэгүүдийг биелүүлж нийлмэл if хэллэгийг боловсруулна.

Java програмын хэл нь if хэллэгийн авсаархан хувилбар болох ?: ийм операторoop хангадаг. MaxVariablesDemo програмаас энэ хэллэгийг буцаая:

```
if (Character.isUpperCase(aChar)) {
    System.out.println("The character " + aChar
                       + " is upper case.");
} else {
    System.out.println("The character " + aChar
                       + " is lower case.");
}
```

Энд та яаж ?: операторыг ашиглан хэллэгийг дахин бичих вэ:

```
System.out.println("The character " + aChar + " is " +
                   (Character.isUpperCase(aChar) ? "upper" :
                        "lower") + "case.");
```

Хэрвээ isUpperCase арга нь үнэн гэж буцаавал ?: оператор нь upper хэлхээг буцаана. Эсрэгээр энэ нь lower хэлхээг буцаана. Үр дүн нь бусад мэдээлэл үзүүлэх хэсгүүдтэй холбогдсон байна. if хэллэг нь хоёрдогч println аргыг дууддаг. Үүнийг ашиглавал таны код хялбар уншигдах болно.

switch хэллэг

switch хэллэгийг бүхэл тоон илэрхийлэл эсвэл тоочих төрөл дээр суурилсан нөхцөлтэй хэллэгүүдийг биелүүлэхэд хэрэглэнэ. Доорх жишээ нь он сар өдрийн сарыг үзүүлдэг SwitchDemo хэмээх програм юм. Энэ нь switch хэллэгийг ашигласан бөгөөд month –ийн утганд суурилсан сарын нэрийг хэвлэдэг програм юм:

```
public class SwitchDemo {
    public static void main(String[] args) {

        int month = 8;
        switch (month) {
            case 1:  System.out.println("January"); break;
            case 2:  System.out.println("February"); break;
            case 3:  System.out.println("March"); break;
            case 4:  System.out.println("April"); break;
            case 5:  System.out.println("May"); break;
            case 6:  System.out.println("June"); break;
            case 7:  System.out.println("July"); break;
            case 8:  System.out.println("August"); break;
            case 9:  System.out.println("September"); break;
            case 10: System.out.println("October"); break;
            case 11: System.out.println("November"); break;
            case 12: System.out.println("December"); break;
            default: System.out.println("Not a month!"); break;
        }
    }
}
```

switch хэллэг нь энэ илэрхийллийг боддог бөгөөд case доторх month –ийн утга мөн зохистой хэллэгийг биелүүлдэг. Тиймээс дээрх програмын гарц нь August юм. Мэдээж хэрэг та if хэллэгийг ашиглан гүйцэтгэж чадна:

```
int month = 8;
if (month == 1) {
    System.out.println("January");
} else if (month == 2) {
    System.out.println("February");
}
. . . //and so on
```

Та if хэллэг эсвэл switch хэллэгийн алийг нь хэрэглэхээ яг таг шийдэх хэрэгтэй. Та аль нэгийг нь сонгохдоо хурд мөн бусад хүчин зүйлсийг харгалзаарай. if хэллэг нь утгуудын эсвэл нөхцөлүүдийн хэмжээн дээр суурилсан зүйлсийг шийдвэрлэдэг бол switch хэллэг нь зөвхөн энгийн бүхэл эсвэл тоочих утганд суурилсаныг шиидвэрлэхэд хэрэглэдэг. Мөн түүнчлэн ямарч case хэллэгийн утга нь цор ганц байх ёстой.

switch хэллэгийн өөр нэгэн чухал заагч нь break хэллэг бөгөөд бүх case –ын ард бичигдэнэ. Бүх break хэллэг нь switch хэллэгийг төгсгөдөг ба удирдлагын урсгал нь switch блокын эхний хэллэгтэй хамт үргэлжилнэ. break хэллэгүүд нь зайлшгүй шаардлагатай. Яагаад гэвэл case хэллэгүүд нь тэдэнгүйгээр уналтанд ордог. Илээр break хэллэггүйгээр удирдлага нь дараагийн case хэллэгүүдээр дэс дарааллан урсана. Доорх SwitchDemo2 жишээ нь яагаад case хэллэгүүд уналтанд орж байгааг үзүүлж байна.

```
public class SwitchDemo2 {
    public static void main(String[] args) {

        int month = 2;
        int year = 2000;
        int numDays = 0;

        switch (month) {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                numDays = 31;
                break;
            case 4:
            case 6:
            case 9:
            case 11:
                numDays = 30;
                break;
            case 2:
                if ( ((year % 4 == 0) && !(year % 100 == 0))
                     || (year % 400 == 0) )
                    numDays = 29;
```

```
                else
                    numDays = 28;
                break;
            default:
                numDays = 0;
                break;
        }
        System.out.println("Number of Days = " + numDays);
    }
}
```

Програмын гарц нь :

```
Number of Days = 29
```

Дээрх нөхцөлд break хэллэг шаардлагагүй. Яагаад гэвэл урсгал нь хаа сайгүй байгаа switch хэллэгийн гадна унана. Гэхдээ бид break хэллэгийг хэрэглэхийг зөвлөж байна. Энэ нь копийг хялбар болгож алдааг багасгадаг.

Эцэст нь та switch –ийн төгсгөл болох default хэллэгийг ашиглах хэрэгтэй. Гэхдээ бүх утгыг case хэллэгүүдийн нэгээр тодорхойлж болохгүй.

switch хэллэгүүдийн тоочих төрлүүд

switch хэллэгүүдэд ашиглагддаг 5,0 –д оруулсан өвөрмөц зүйл бол тоочсон төрөл юм. Бид энэ хэсэгт тэднийг хэрхэн switch хэллэгтэй хамт ашиглахыг үзэх болно. Энэ нь switch хэллэг дэхь бүхэл тоог ашиглахтай ойролцоо юм. Доорх SwitchEnumDemo жишээ код нь өмнө нь үзсэн SwitchDemo2 –ийн кодтой бараг адил юм.

```
public class SwitchEnumDemo {
    public enum Month { JANUARY, FEBRUARY, MARCH, APRIL,
                        MAY, JUNE, JULY, AUGUST, SEPTEMBER,
                        OCTOBER, NOVEMBER, DECEMBER }

    public static void main(String[] args) {
        Month month = Month.FEBRUARY;
        int year = 2000;
        int numDays = 0;

        switch (month) {
            case JANUARY:
            case MARCH:
            case MAY:
            case JULY:
            case AUGUST:
            case OCTOBER:
            case DECEMBER:
                numDays = 31;
                break;
            case APRIL:
            case JUNE:
```

```
            case SEPTEMBER:
            case NOVEMBER:
                numDays = 30;
                break;
            case FEBRUARY:
                if ( ((year % 4 == 0) && !(year % 100 == 0))
                     || (year % 400 == 0) )
                   numDays = 29;
                else
                   numDays = 28;
                break;
            default:
                numDays=0;
                break;
        }
        System.out.println("Number of Days = " + numDays);
    }
}
```

Энэ жишээнд Java хэл нь тоочих үйлдлийг яаж биелүүлдгийг үзүүлсэн байна.

Гажуудал засах хэллэгүүд

Java програмын хэл нь програмын тайлан болон алдаануудад тусалдаг exceptions (гажуудал) хэмээх механизмаар хангагдсан байдаг. Яг гэсэн үг вэ гэвэл? Энэ нь програмын жирийн урсгал нь тасалдсан мөн гүйлгэх үеийн орчин нь алдааны нарийн төрлүүдийг засдаг кодын блок болох exception handler хэмээх гажуудал засагчийг олохыг оролдоно. Гажуудал засагч нь алдаанаас сэргээхийг оролдох ба хэрэв алдаа нь сэргээгдэхгүй гэж тодорхойлогдсон бол програмаас гарна.

Гажуудлыг засах хэсэгт гурван хэллэг байдаг

- try хэллэг. Энэ нь гажуудал үүсэх магадлалтай хэллэгүүдийн блокыг таньдаг.
- catch хэллэг. Энэ нь try хэллэгтэй нэгдэж гажуудлийн тодорхой төрлийг засдаг хэллэгүүдийн төрлийг таньдаг.
- finally хэллэг нь try хэллэгтэй нэгдэж try блокт алдаа гараагүй эсвэл хайхрамжгүй биелүүлсэн эсэхийг хэллэгүүдийн блокоос таньдаг.

Эдгээр хэллэгүүдийн ерөнхий өгүүлбэр зүй нь:

```
try {
                statement (s)
} catch (exceptiontype name) {
        statement (s)
} finally {
        statement (s)
}
```

Энэ нь Java програмын хэл алдаануудыг мэдээлэх мөн засах хэллэгүүдээр хангагдсан гэсэн үг. Хэдийгээр тийм боловч runtime мөн шалгагдсан гажуудлууд мөн гажуудлын

ангиудыг шатлан захирах гэх мэтийн хооронд ялгаатай бусад хүчин зүйлс мөн ач холбогдол нь гажуудлын механизм дахь гажуудлын ялгаатай төрлүүдийг үзүүлдэг.

Салаалуулах хэллэгүүд

Java програмын хэл нь гурван салаалуулах хэллэгээр хангагдсан байдаг.

- break хэллэг
- continue хэллэг
- return хэллэг

break мөн continue хэллэгүүд нь шошготой мөн шошгогүй төрлүүдийг зогсооход хэрэглэгдэнэ. Хаяг хэллэгийн өмнөх тогтсон байрлал ба (:) –тэй хамт бичигдэнэ.

```
statementName: someJavaStatement;
```

break хэллэг

break хэллэг нь хоёр төрөлтэй. Хаяглагдсан мөн хаяглагдаагүй. Та switch –тэй хэрэглэгддэг break хэллэгийн шошгогүй төрлийг үзсэн. Шошгогүй break нь switch –ийг зогсоодог ба удирдлагын урсгал нь switch руу шууд дамжина. Та break хэллэгийн шошгогүй төрлийг for, while эсвэл do – while давталтыг дуусгахад ашиглаж болно. Доорх BreakDemo програм нь for давталт ашиглан бүл дэхь тодорхой утгыг хайна.

```
public class BreakDemo {
    public static void main(String[] args) {

        int[] arrayOfInts = { 32, 87, 3, 589, 12, 1076,
                              2000, 8, 622, 127 };
        int searchfor = 12;

        int i = 0;
        boolean foundIt = false;

        for ( ; i < arrayOfInts.length; i++) {
            if (arrayOfInts[i] == searchfor) {
                foundIt = true;
                break;
            }
        }

        if (foundIt) {
            System.out.println("Found " + searchfor
                               + " at index " + i);
        } else {
            System.out.println(searchfor
                               + "not in the array");
        }
    }
}
```

Хэрвээ утга олдвол break хэллэг нь for давталтыг зогсоох ба удирдлагын урсгал нь давталтыг дуусгаад програмын төгсгөл дэх print хэллэгийг дамжуулна.

Програмын гарц нь:

```
Found 12 at index 4
```

break хэллэгийн шошгогүй төрөл болох break нь switch, for, while эсвэл do – while зэргийг төгсгөхөд ашиглагдах ба шошготой төрөл нь break хэллэг доторх тодорхой шошгоор танигдаж, хэллэгийн гадна төгсдөг.

Дараах BreakWithLabelDemo програм нь өмнөхтэй ойролцоо бөгөөд гэхдээ хоёр хэмжигдэхүүнтэй бүл дэх утгыг хайдаг. Багтсан хоёр for давталт нь бүлд нэвтрэнэ. Хэрэв утга олдвол шошготой break нь for давталтын гаднах шошготой search хэллэгийг төгсгөнө.

```
public class BreakWithLabelDemo {
    public static void main(String[] args) {

        int[][] arrayOfInts = { { 32, 87, 3, 589 },
                                { 12, 1076, 2000, 8 },
                                { 622, 127, 77, 955 }
                              };
        int searchfor = 12;

        int i = 0;
        int j = 0;
        boolean foundIt = false;

    search:
        for ( ; i < arrayOfInts.length; i++) {
            for (j = 0; j < arrayOfInts[i].length; j++) {
                if (arrayOfInts[i][j] == searchfor) {
                    foundIt = true;
                    break search;
                }
            }
        }

        if (foundIt) {
            System.out.println("Found " + searchfor +
                               " at " + i + ", " + j);
        } else {
            System.out.println(searchfor
                               + "not in the array");
        }

    }
}
```

Програмын гарц нь:

```
Found 12 at 1, 0
```

Энэ өгүүлбэр зүй жоохон андуурч болно. break хэллэг нь шошготой хэллэгийг төгсгөдөг ба удирдлагын урсгал нь шошготой руу дамжихгүй. Удирдлагын урсгал нь дараагийн шошготой хэллэг (төгссөн) руу шууд дамжина.

continue хэллэг

Та continue хэллэгийг for, while эсвэл do–while давталтын тохиосон давталтыг алгасахад хэрэглэнэ. Шошгогүй төрөл нь давталтын биеийн хамгийн доорх хэсгийн төгсгөлийг алгасдаг ба давталтын удирдлагууд нь энэ давталтын давтах үйлдлийг алгасаад булийн илэрхийлэлийг боддог. Доорх ContinueDemo програм нь хэлхээний буферээс алхам алхмаар үсэг бүрийг шалгадаг. Хэрвээ тохиосон тэмдэгт нь ‘p’ биш бол continue хэллэг нь давталтыг зогсоох ба дараагийн тэмдэгт руу шилжинэ. Хэрвээ ‘p’ олдвол тоолуурыг нэгээр нэмэгдүүлээд жижиг ‘p’ –ыг том ‘P’ болгож хувиргана.

```
public class ContinueDemo {
    public static void main(String[] args) {

        StringBuffer searchMe = new StringBuffer(
            "peter piper picked a peck of pickled peppers");
        int max = searchMe.length();
        int numPs = 0;

        for (int i = 0; i < max; i++) {
            //interested only in p's
            if (searchMe.charAt(i) != 'p')
                continue;

            //process p's
            numPs++;
            searchMe.setCharAt(i, 'P');
        }
        System.out.println("Found " + numPs
                           + " p's in the string.");
        System.out.println(searchMe);
    }
}
```

Програмын гарц нь:

```
Found 9 p's in the string.
Peter PiPer Picked a Peck of Pickled PePPers
```

continue хэллэгийн шошготой төрөл нь гаднах давталтын тогтсон шошгоны тохиосон давталтыг алгасдаг. Доорх ContinueWithLabelDemo програм нь багтсан давталт ашиглан бусад хэлхээн дэх дэд хэлхээг хайна. Багтсан хоёр давталтанд дараах зүйл хэрэгтэй: нэг нь дэд хэлхээг зогсоох, нөгөө нь хэлхээ хайхыг зогсоох. Энэ програм нь continue –ийн шошготой төрлийг ашиглан гаднах давталтын давталтыг алгасч байна.

```
public class ContinueWithLabelDemo {
    public static void main(String[] args) {

        String searchMe = "Look for a substring in me";
        String substring = "sub";
```

```
        boolean foundIt = false;

        int max = searchMe.length() - substring.length();

    test:
        for (int i = 0; i <= max; i++) {
            int n = substring.length();
            int j = i;
            int k = 0;
            while (n-- != 0) {
                if (searchMe.charAt(j++)
                        != substring.charAt(k++)) {
                    continue test;
                }
            }
            foundIt = true;
                    break test;
        }
        System.out.println(foundIt ? "Found it" :
                                "Didn't find it");
    }
}
```

Програмын гарц нь:

```
Found it
```

return хэллэг

return хэллэг нь салаалуулах хэллэгүүдийн сүүлчийн хэллэг юм. Та return хэллэгийг тохиосон аргаас гарахад хэрэглэж болно. Удирдлагын урсгал нь жинхэнэ аргын дуудлагын хэллэг руу буцдаг. return хэллэг нь хоёр төрөлтэй: Нэг нь утга буцаадаг, нөгөө нь буцаадаггүй. Буцах утга нь return түлхүүр үгийн ард тавьсан утга юм. (эсвэл утга боддог илэрхийлэл)

```
return ++count;
```

Утгын өгөгдлийн төрөл нь аргын зарласан буцах утгын төрөл болох return –тэй таарах ёстой. Хэрвээ арга нь void гэж зарлавал return –ийн утга буцаадаггүй төрлийг ашиглана.

```
return;
```

Дүгнэлт

Java програмын хэл нь програмын удирдлагын урсгалд зориулж if – else, switch, гажуудал засах хэллэгүүд мөн салаалуулах хэллэгүүдийг байгуулж өгсөн.

Давталтууд

while хэллэгийг булын илэрхийлэл үнэн болоход хэллэгүүдийн блокын давталтыг зогсооход хэрэглэнэ. Илэрхийлэл нь давталтын эхнээс бодогдоно.

```
while (boolean expression) {
    statement(s)
}
```

do-while хэллэгийг булын илэрхийлэл үнэн болоход хэллэгүүдийн блокын давталтыг зогсооход хэрэглэх ба илэрхийлэл нь давталтын сүүлчийн өчүүхэн хэсгийг л боддог

```
do {
    statement(s)
} while (expression);
```

for хэллэгийг цэнэглэх илэрхийлэл, нөхцөл дуусгах илэрхийлэл мөн нэмэгдүүлэх илэрхийлэлүүд агуулсан хэллэгүүдийн блокын давталтыг дуусгахад хэрэглэнэ.

```
for (initialization ; termination ; increment) {
    statement(s)
}
```

Шийдвэр гаргах хэллэгүүд

Java програмын хэл нь if-else мөн switch хэмээх хоёр шийдвэр гаргах хэллэгтэй. if нь ерөнхий зорилготой хэллэг. Харин switch –ийг энгийн бүхэл утганд суурилсан олон сонголтуудыг шийдвэрлэхэд хэрэглэдэг.

Доорх энгийн блок хэллэг нь булийн илэрхийлэл үнэн болоход биелэгддэг үндсэн if хэллэг юм.

```
if (boolean expression) {
    statement(s)
}
```

Энд else хэмээх дагуул хэллэгтэй if хэллэг байна. if хэллэг нь эхний блок дахь булын илэрхийлэл үнэн байвал хэллэгийг биелүүлдэг. хэрэв илэрхийлэл нь худал байвал хоёрдох блокыг биелүүлдэг.

```
if (boolean expression) {
    statement(s)
} else {
    statement(s)
}
```

Та нийлмэл if хэллэгүүдийг байгуулахад else-if –ийг ашиглаж болно.

```
if (boolean expression) {
    statement(s)
} else if (boolean expression) {
    statement(s)
} else if (boolean expression) {
    statement(s)
} else {
    statement(s)
}
```

switch хэллэг нь бүхэл тоон илэрхийлэл эсвэл тоочсон илэрхийллийг боддог ба case хэллэгийг биелүүлдэг.

```
switch (integer expression) {
    case integer expression:
         statement(s)
         break;
    ...
    default:
         statement(s)
         break;
}
switch (expression of enum type) {
    case enum constant:
         statement(s)
         break;
    ...
    default:
         statement(s)
         break;
}
```

Гажуудал засах хэллэгүүд

Гажуудлыг засахад try, catch мөн finally хэллэгүүдийг ашиглана.

```
try {
    statement(s)
} catch (exceptiontype name) {
    statement(s)
} catch (exceptiontype name) {
    statement(s)
} finally {
    statement(s)
}
```

Салаалуулах хэллэгүүд

Хэсэг салаалуулах хэллэгүүд програм дахь удирдлагын урсгалыг шошготой хэллэг болгон өөрчилдөг. Шошго тавихдаа шошголох хэллэгийнхээ өмнө хэллэгийн нэрээ бичээд тодорхойлох хоёр цэг тавина.

```
statementName: someJavaStatement;
```

break хэллэгийн шошгогүй төрлийг switch, for, while эсвэл do-while хэллэгүүдийг дуусгахад хэрэглэнэ.

```
break;
```

break хэллэгийн шошготой төрлийг тогтсон шошготой гаднах switch, for, while эсвэл do-while хэллэгүүдийг дуусгахад хэрэглэнэ.

```
break label;
```

continue хэллэг нь давталт доторх тохиосон давталтыг дуусгахад мөн давталт удирддаг булын илэрхийлэл бодоход ашиглана.

```
continue;
```

continue хэллэгийн шошготой төрөл нь тогтсон шошготой давталтын тохиосон давталтыг алгасдаг.

```
continue label;
```

return хэллэгийг тохиосон аргыг дуусгахад хэрэглэнэ.

```
return;
```

Та return –ийн утгын төрлийг ашиглан аргын дуудлага руу утга буцааж болно.

```
return value;
```

Асуулт болон дасгал

Асуулт

1. SortDemo програмыг хар. Ямар удирдлагын урсгалын хэллэгүүд агуулж байна вэ?
2. Доорх кодын хэсэгт ямар алдаа байна вэ?

```
if (i = 1) {
    /* do something */
}
```

3. WhileDemo мөн DoWhileDemo програмыг хар. Хэрэв эдгээрийн copyFromMe хэлхээний утгыг golly see. this is fun. гэж өөрчлөвөл програмуудын гарц нь ямар ямар байх вэ?

Дасгал

1. Доорх кодын хэсгийг авч үеье.

```
if (aNumber >= 0)
    if (aNumber == 0) System.out.println("first string");
else System.out.println("second string");
System.out.println("third string");
```

   a. Хэрэв aNumber нь 3 гэж өгөгдвөл програмын гарц нь ямар байна гэж бодож байна вэ?
   b. aNumber –г 3 болгох кодыг агуулсан тест програм бич. Програмын гарц нь ямар байх вэ? Урьдаас мэдэж байсан уу? Яагаад гарц нь ийм байгааг тайлбарла. Өөрөөр хэлбэл кодын хэсэг дэх удирдлагыг урсгал нь юу вэ?
   c. Хоосон зай болон мөр шилжүүлэхийг ашиглан код хэсэг дэх удирдлагын урсгалыг ойлгоход хялбар болгон дахин формат тавь.
   d. Цаашид кодыг ойлгомжтой болгохын тулд { } хаалт ашигла.

Объектын үндэс ба өгөгдлийн энгийн объектууд

Эхний бүлэгт амьдралын мөчлөгийн тухай ерөнхий ойлголтыг авч үзнэ. Бүл төрлийн объектуудад хамааралтай мэдээллийг авч үзэхдээ объектыг хэрхэн байгуулах, объект яаж ашиглах болон хэрэгцээгүй объектыг систем хэрхэн цэвэрлэдэг тухай тайлбарлах болно.

Дараагийн бүлэгт үндсэн ангиудыг яаж ашиглах болон тэмдэгт өгөгдөл, тоон өгөгдөл болон бүлийг хэрхэн ашиглах талаар тайлбарлана.

Энэ нь дараах сэдвүүдийг агуулна.

- **Тэмдэгт өгөгдөл** - Нэг тэмдэг эсвэл олон тэмдэгтүүдтэй ажиллах, хадгалахдаа java.lang доторх Character, String, StringBuilder, StringBuffer 4 ангийг ашиглана.
- **Тоон өгөгдөл** - Number анги болон түүний дэд ангиуд нь өгөгдлийн эгэл төрлүүдийн объект хэлбэр юм. Java Platform дахь бүх тоон ангиудын дээд анги нь Number анги бөгөөд түүний дэд ангиуд нь java.lang доторх Byte, Double, Float, Integer, Short дэд ангиуд юм. Түүнчлэн илүү өндөр нарийвчлалтай олон оронтой тоотой ажиллахад зориулагдсан BingDecimal, BigInteger ангиуд нь java.math багц дотор тодорхойлогдсон байдаг.
- **Бүл** - Нэг төрлийн олон утгуудыг нэгтгэж нэг объект болгодог. Бүл нь Java хэлийг шууд дэмждэг. Бүл анги гэж байхгүй. Бүл нь объект ангиас удамшсан байдаг тул объект төрлийн дурын хувьсагч руу бүлийг хадгалж, утга олгож болно.

Java платформ нь зориулалтаар нь бүлэглэж багцалдаг. Тэмдэгт, хэлхээ болон тоон өгөгдлийг дүрслэхийн тулд өөрөө анги бичих шаардлагагүй харин java платформын бэлэн ангиудыг ашиглана. Энэ бүлэгт авч үзэж буй ангиуд нь java.lang –ын бүрэлдэхүүнд ордог. Java.lang-ын багц дахь бүх ангиуд таны програмд автоматаар нээгддэг тул эдгээрийг import хэлбэрээр нээх шаардлагагүй.

Объект амьдралын мөчлөг

Ихэнх java програм нь мэдээ илгээх замаар хоорондоо харилцдаг олон объектыг үүсгэдэг. Объектын харилцан үйлчлэлийн явцад дараах ажлыг хийдэг. Зурмал интерфейс, хөдөлгөөнт дүрс ажиллуулах болон сүлжээгээр дамжуулан мэдээлэл хүлээн авах болон илгээх зэргийг хийдэг. Объект нь үүргээ гүйцэтгэж дуусмагц түүний нөөцийг өөр объектуудад хуваарилдаг.

Нэг Point болон хоёр Rectangle нийт 3 объект үүсгэн CreateObjectDemo жижиг программыг дор үзүүлэв.

```
public class CreateObjectDemo {
        public static void main(String[] args) {
                // declare and create a point object
                // and two rectangle objects
                Point originOne = new Point(23, 94);
                Rectangle rectOne = new Rectangle(originOne, 100, 200);
                Rectangle rectTwo = new Rectangle(50, 100);
                // display rectOne's width, height, and area
```

```
                System.out.println("Width of rectOne: " +
                        rectOne.width);
                System.out.println("Height of rectOne: " +
                        rectOne.height);
                System.out.println("Area of rectOne: " + rectOne.area());
                // set rectTwo's position
                rectTwo.origin = originOne;
                // display rectTwo's position
                System.out.println("X Position of rectTwo: "
                        + rectTwo.origin.x);
                System.out.println("Y Position of rectTwo: "
                + rectTwo.origin.y);
                // move rectTwo and display its new position
                rectTwo.move(40, 72);
                System.out.println("X Position of rectTwo: "
                        + rectTwo.origin.x);
                System.out.println("Y Position of rectTwo: "
                        + rectTwo.origin.y);
        }
}
```

Энэ программыг эмхтгэхийн тулд 3 файл ашиглана. Энэ программ янз бүрийн объект үүсгэж түүнтэй харилцаж ажлаад үр дүнг нь хэвлэж байна. Энэ программын гарц нь :

```
Width of rectOne: 100
Height of rectOne: 200
Area of rectOne: 20000
X Position of rectTwo: 23
Y Position of rectTwo: 94
X Position of rectTwo: 40
Y Position of rectTwo: 72
```

Дээрхи жишээнд тулгуурлан программ доторх амьдралын мөчлөгийг доорх хэсгээр задалж харуулав. Программд объектыг яаж үүсгэх, яаж ашиглах талаар зохистой мэдлэгийг та олж авна. Түүнчлэн амьдралынх нь мөчлөг дууссан объектыг систем хэрхэн цэвэрлэдэг талаар мэдэж авна.

- Объектыг үүсгэх
- Объектыг хэрэглэх
- Хэрэгцээгүй объектыг цэвэрлэх
- Дүгнэлт
- Асуулт болон дасгал

Объектыг үүсгэх нь

Анги бол объектын эх загвар юм. Өөрөөр хэлбэл объектыг ангиас үүсгэнэ. CreateObjectDemo –г иш татсан дараах 3 хэллэг нь объект үүсгэж түүнийгээ хувьсагчид олгосон байна.

```
Point originOne = new Point(23, 94);
Rectangle rectOne = new Rectangle(originOne, 100, 200);
Rectangle rectTwo = new Rectangle(50, 100);
```

Эхний мөрөнд Point ангиас объект үүсгэж, 2,3-р мөрөнд Rectangle ангиас объект үүсгэж байна.

Эдгээр хэллэг бүр дараах 3 хэсгээс бүрдэж байна.

1. **Мэдэгдэл -** Хувьсагчийн нэрийг объектын төрөлтэй холбосон бүх хувьсагчийн нэрийг дармалдаж харууллаа.
2. **Төллөлт -** Объект үүсгэх java хэлний оператор бол New түлхүүр үг юм. Үүнийг өөрөөр хэлбэл объектыг төллүүлэх гэж нэрэлдэг.
3. **Цэнэглэлт -** New операторын ард байгуулагчийн дуудлагыг бичдэг. Жишээ нь : Point (23,94) нь Point –ын цорийн ганц байгуулагч юм. Энэ байгуулагч нь шинэ объектоо цэнэглэнэ.

Эдгээр үйл явцыг тус бүрд нь нарийвчлан авч үзье.

Объект заасан хувьсагч зарлах

Өмнөх хичээлийн “Хувьсагчид” хэсэгт үзсэнээр хувьсагч зарлахдаа дараах маягаар бичдэг.

```
type name
```

Type гэсэн төрлийн өгөгдхүүн рүү хандахдаа name гэсэн нэр ашиглана гэдгийг дээрх бичлэг харуулна. Java хэлэнд хувьсагчийн төрлийг эгэл төрөл ба заагч төрөл гэж 2 ангилна. Эгэл төрлийн (byte, short, int, long, char) хувьсагч нь ямагт тухайн төрлийнхөө ганц утгыг хадгалдаг. Харин заагч төрлийн хувьсагч нь илүү нийлмэл шинжтэй байдаг.

Эдгээрийг дараах мэт олон янзаар зарлаж болдог.

- Зарласан төрөл нь объектынхоо ангитай адил
  MyClass myObject `= new MyClass();`
- Зарласан төрөл нь объектынхоо ангийн өвөг анги нь байна.
  MyParent myObject `= new MyClass();`
- Зарласан төрөл нь тухайн объектын ангид хэрэгжүүлсэн интерфейс нь байна.
  MyInterface myObject `= new MyClass();`

Эсвэл объектыг, хувьсагчийг дангаар нь зарлаж болно.

```
MyClass myObject;
```

Ийм нөхцөлд объектыг үүсгэж утга олгох хүртэл myObject хувьсагчийн утга автоматаар NULL байдаг. Хувьсагчийн мэдэгдэл нь дангаараа объект үүсгэдэггүй гэдгийг санаарай. Объект үүсгэхийн тулд new оператор ашигладаг. Ямар ч объект заагаагүй хувьсагчийн энэ төлөвийг хоосон заагч утга гэдэг. (null reference) Хэрэв CreateObjectDemo дотор originOne –ын ийм маягаар зарласан бол доорх маягаар тайлбарлаж болно.

Заагч төрлийн хувьсагч нь объект заагчийг агуулж бас болно.

Анги төллүүлэх

New операторор анги төллүүлэхэд шинээр үүссэн объектод санах ойг хуваарилна.

Санамж: Анги төллүүлнэ гэдэг нь объект үүсгэнэ гэдэгтэй адил утгатай. Объект үүсгэхэд ангийн төл үүсдэг, тиймээс анги төллөнө гэсэн үг.

New оператор нь байгуулагчийн дуудлага гэсэн ганц аргументтай. Тухайн байгуулагчийн нэр нь төллүүлэх ангийнхаа нэр байдаг. Энэхүү байгуулагч нь үүссэн объектоо цэнэглэнэ.

New оператор нь үүссаэн объектынхоо заагчийг буцаадаг. Ингэхдээ энэхүү заагчийг тохирох хувьсагчид олгодог. Хэрэв энэ заагчийг хувьсагчид олгохгүй бол new оператор агуулсан хэллэг өндөрлөсний дараа тухайн объект руу хандах боломжоор болдог.

Объект цэнэглэх

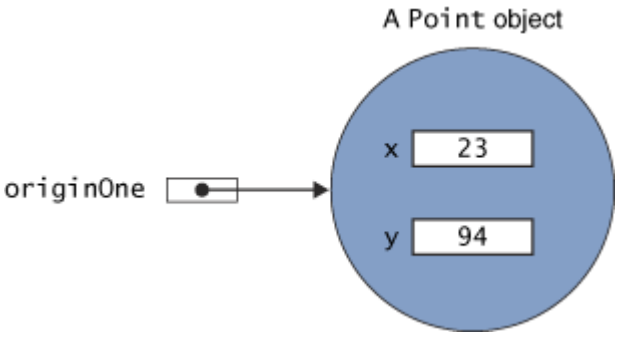
Point ангийн кодыг дор харуулав.

```
public class Point {
    public int x = 0;
    public int y = 0;
    //A constructor!
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

Энэ анги нь ганцхан байгуулагчтай юм. Байгуулагч нь ангийнхаа нэртэй адил нэртэй буцах төрөлгүй байгаа нь харагдаж байна. Int x, int y гэж бичсэнээс үзэхэд Point ангийн байгуулагч нь 2 бүхэл аргумент (хэмжигдэхүүн) –тэй ажээ: Эдгээр хэмжигдэхүүний утга 23 ба 94 байж болохыг доорх хэллэг харуулав.

```
Point originOne = new Point(23, 94);
```

Энэ код ямар үр дүнд хүрэхийг доорхи зургаар харуулав.



4 байгуулагчтай Rectangle ангийн кодыг дор харуулав.

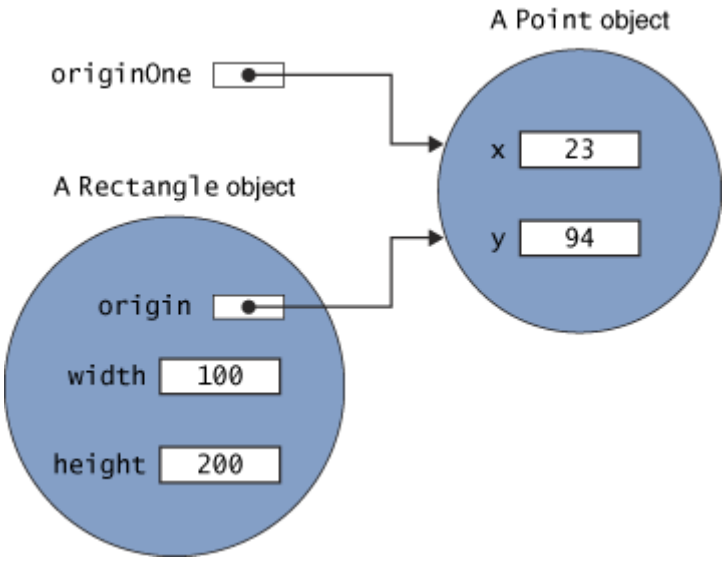
```
public class Rectangle {
    public int width = 0;
    public int height = 0;
    public Point origin;
```

```
    //Four constructors
    public Rectangle() {
        origin = new Point(0, 0);
    }

    public Rectangle(Point p) {
        origin = p;
    }

    public Rectangle(int w, int h) {
        this(new Point(0, 0), w, h);
    }

    public Rectangle(Point p, int w, int h) {
        origin = p;
        width = w;
        height = h;
    }

    //A method for moving the rectangle
    public void move(int x, int y) {
        origin.x = x;
        origin.y = y;
    }

    //A method for computing the area of the rectangle
    public int area() {
        return width * height;
    }
}
```

Эдгээр байгуулагч нь 4 тэгш өнцөгтийн анхны байрлал, өргөн, урт бүгдийг эсвэл хоосон гэсэн 4 янзаар тэгш өнцөгтийнхөө анхны утгыг олгож байна. Хэрэв анги нь олон байгуулагчтай бол эдгээрийн нэр нь адилхан байх боловч хэмжигдэхүүнийхээ тоо болон төрлүүд дээр нь тулгуурлан байгуулагчуудаа ялгана. Доорхи код ажиллахад Rectangle анги доторх Point төрлийн хэмжигдэхүүн, 2 бүхэл тоон хэмжигдэхүүн бүхий байгуулагчийг дуудна.

```
Rectangle rectOne = new Rectangle(originOne, 100, 200);
```

Энэ дуудлага нь originOne хувьсагчийн заасан Point объектыг тэгш өнцөгт origin хувьсагчид олгоно. Түүнчлэн өндөр 100, өргөн 200 гэсэн утгатай болж байна. Доорхи зурагт үзүүлснээр нэг объект хичнээн ч заагчтай байж болно.



Доорхи код нь өргөн, өндрийн анхны утгыг тодорхойлсон. Хоёр бүхэл тоон аргументтэй байгуулагчийг дуудаж утга олгож байна. Хэрэв энэ байгуулагч дахь кодыг гүнзгийрүүлэн харах юм бол x,y нь 0 байх Point объект шинээр үүсгэж байгааг харах болно.

```
Rectangle rectTwo = new Rectangle(50, 100);
```

Дараах хэллэгт орсон Rectangle байгуулагч нь ямар ч аргументгүй байгаа бөгөөд үүнийг аргументгүй байгуулагч гэнэ.

```
Rectangle rect = new Rectangle();
```

Хэрэв объект дотор ямар ч байгуулагч зарлаагүй бол ямар ч үүрэггүй оршмол байгуулагч хэмээх аргументгүй байгуулагчийг автоматаар үүсгэнэ. Тиймээс бүх ангиуд нь ядаж нэг байгуулагчтай байна.

Объектуудыг хэрэглэх

Объект байгуулсан бол түүнийгээ ямар нэг байдлаар ашиглах нь дамжиггүй. Түүнээс мэдээлэл авах, төлвийг нь өөрчлөх эсвэл үүгээр ямар нэг үйлдэл хийлгүүлэх гэх мэт.

Объектыг ашиглах 2 зам байдаг.

- Шууд харьцах буюу хувьсагчуудыг шалгах.
- Аргуудыг нь дуудах

Объектын хувьсагчидтай харьцах

Урт нэр буюу нарийвчилсан болон ерөнхий хэлбэрийг дор үзүүлэв.

```
objectReference.variableName
```

Цараа доторх төл хувьсагчтай харьцахдаа өөрөөр хэлбэл объектынхоо ангийн код дотор төл хувьсагчийн энгийн нэрийг ашиглана. Объектын ангийн гаднаас харьцахдаа заавал нарийвчилсан нэрийг ашиглана. Жишээ нь : CreateObjectDemo анги доторх код бол Rectangle ангийн кодын гадна байна. Тиймээс rectOne нэртэй Rectangle объектын

origin,урт, өргөн хувьсагчидтай харьцахдаа CreateObjectDemo анги нь rectOne.origin, rectOne.width, rectOne.height нэрийг ашиглана. Энэ програм нь rectOne объектыг width болон height утгыг хэвлэхдээ дурьдсан тэмдэглэгээг ашиглаж байна.

```
System.out.println("Width of rectOne: " + rectOne.width);
System.out.println("Height of rectOne: " + rectOne.height);
```

CreateObjectDemo анги доторх width болон height гэсэн энгийн нэрээр хандах нь буруу тэдгээр нь зөвхөн объект дотор тодорхойлогдсон тул эмхтгүүрийн алдаа өгнө.

RectTwo объектын тухай мэдээлэл гаргахад энэ програм нь төстэй код ашигласан байна. Нэг төрлийн объектууд төл хувьсагёийнхаа өөр өөрийн хуулбар агуулна. Тиймээс Rectangle объект бүхэл origin, width, height хувьсагчидтай байна. Объект заагчаар дамжуулан төл хувьсагчтай харьцахад тухайн тодорхойлсон объектын хувьсагчтай харьцана. CreateObjectDemo програм доторх rectOne болон rectTwo хоёр объектын дээрх гурван хувьсагчид нь хоорондоо ялгаатай юм.

Хувьсагчийн нарийвчилсан нэрний эхний хэсэг объект заагч нь заавал объектын заагч байх ёстой. Эсвэл өмнөх жишээнд үзүүлсэнчлэн энэ нь заагч хувьсагчийн нэр эсвэл объект заагч буцаах дурын илэрхийлэл ашиглаж болно. New оператор нь объектын заагч буцаадаг болохыг эргэж саная. Тиймээс шинээр үүссэн объектынхоо хувьсагч руу хандахдаа new операторооc буцаах утгыг ашиглаж болно.

```
int height = new Rectangle().height;
```

Энэ хэллэг нь Rectangle объект үүсгэмэгц үүний өндрийг шууд авсан байна. Үнэн хэрэгтээ энэ хэллэг нь Rectangle объектын оршмол өндрийг шалгаж байна. Нэгэнт заагчийн утгыг ямар нэг хувьсагчид хадгалаагүй тул энэ хэллэгийг биелүүлсний дараа шинээр үүссэн rect объект руу хандах ямар ч бололцоогүй болно. Энэ объект нь заагчаа алдаж түүний бүх нөөц чөлөөлөгдөнө.

Хувьсагчидтай харьцах нь

Өөр объект болон ангиудаас объектын хувьсагчтай шууд харьцах нь тохиромжгүй байдаг. Учир нь уг хувьсагчид замбараагүй утга олгож болзошгүй байдаг. Жишээ нь : өмнө үзсэн Rectangle ангийг авч үзье.

Энэ ангийг ашиглан өндөр болон өргөн сөрөг байх тэгш өнцөгт байгуулах боломжтой байгаа нь зарим хэрэглэгдэхүүний хувьд ойлгомжгүй байдаг. Хувьсагчтай шууд харьцахын оронд хувьсагчийн утгыг өөрчлөх, шалгах бололцоог бусад объектуудад олгосон тусгай арга ашиглах нь илүү тохиромжтой байдаг. Ийм арга нь тухайн төрлийн объектын хувьд хувьсагчийнх нь утга зөв байх нөхцлийг нь бүрдүүлдэг. Тиймээс Rectangle анги нь урт, өргөний утгыг тохируулахад зориулсан serWidth, setHeight, getHeight, getWidth аргуудыг тодорхойлох нь оновчтой. Энэ аргууд нь хэрэв өндөр, өргөнд сөрөг утга олгоход алдааны мэдээлэл өгдөг байж болно.

Хувьсагчидтай шууд харьцахын оронд аргаар дамжуулж харьцахын өөр нэг давуу тал бол урт, өргөний мэдээллийг хадгалах зориулалттай ашиглах хувьсагчийнхаа нэр болон төрлийг бусад объектуудад нөлөөлөхгүйгээр өөрчилж болдог тал юм. Гэвч бодит амьдрал дээр объектын хувьсагч руу шууд хандах илүү ашигтай байх нөхцөл байж болно.

Жишээ нь: Point болон Rect анги нь хоёулаа гишүүн хувьсагчидаа public хэмээн зарлаж нээсэн байна. Ингэснээр эдгээр ангиуд нь энгийн бөгөөд цомхон болсон байна. Тэдгээр нь ерөнхийдөө илүү тохиромжтой болсон байна. Зарим тэгш өнцөгтүүд сөрөг утгатай өндөр, урттай байхыг зөвшөөрч болох юм.

Гишүүн хувьсагч руу нь өөр ямар анги шууд харьцах боломжтойг тодорхойлсон хандалтыг удирдах механизм Java хэлэнд байдаг. Анги нь тухайн төрлийн объектын хувьд байж боломгүй утга олгохыг оролдсон бусад объектуудаас хувьсагчидаа хамгаалах ёстой. Иймэрхүү хувьсагчдын өөрчлөлтийг аргын дуудлагаар зохицуулах ёстой. Гишүүн хувьсагч руу нь чөлөөтэй хандах боломжтой байна гэдэг тэдгээрээр хувьсагчдыг шалгах, өөрчлөхөд ноцтой хор хөнөөл учрахгүй хэмээн тооцож болно. Хувьсагчийг нээлттэй болгосноор тэдгээр нь ангийнхаа API болдог. Тиймээс уг ангийг бичсэн хүн хувьсагчийнхаа нэр төрлийг өөрчилж болохгүй болдог.

Объектын аргыг дуудах

Объектын нарийвчилсан нэрийг ашиглан тухайн объектын аргыг дуудаж болно. Аргын нарийвчилсан нэрийг бичихдээ объектын заагчийг хойно цэг тавиад аргын нэрийг нэмж өгнө. Бас хаалтанд аргын авч болох аргументыг зааж өгч болно. Хэрэв уг арга аргумент шаарддаггүй бол хаалтыг хоосон үлдээнэ.

```
objectReference.methodName(argumentList);
```

эсвэл

```
objectReference.methodName();
```

Rectangle анги хоёр аргатай байна.

1. Тэгш өнцөгтийн талбай бодох
2. Тэгш өнцөгтийн байрлалыг өөрчлөх

Доор CreateObjectDemo –ын 2 аргыг дуудсан кодыг харуулав.

```
System.out.println("Area of rectOne: " + rectOne.area());
...
rectTwo.move(40, 72);
```

Эхний хэллэг нь rectOne-ын area аргыг дуудаж үр дүнг дэлгэцэнд хэвлэж байна. Хоёр дахь хэллэг нь rectTwo –г зөөх байна. Учир нь move арга нь уг объектын x,y утгыг шинээр олгож байна.

Төл хувьсагчтай адилаар объектын заагч нь объектыг зааж байх ёстой. Тухайн объектын хувьсагчийн нэрээр хандаж болох ба мөн объектын заагчийг буцаадаг ямар ч илэрхийллээр хандаж болно. New оператор нь объектын заагчийг буцаадаг учраас уг операторoop объектыг үүсгээд түүний аргыг дуудаж уг утгад хандаж болно.

```
new Rectangle(100, 50).area()
```

Дээрхи илэрхийлэл тэгш өнцөгтийг зааж буй объектын заагчийг буцаана. Дээр үзүүлснээр үүссэн тэгш өнцөгтийн area аргыг дуудаад түүний талбайг бодож байна. Зарим аргууд area арга шиг утга буцааж болно. Тиймээс үүнийг нь ашиглан илэрхийлэл бичиж болно. Дурын хувьсагчид буцаж ирж байгаа утгыг олгож дурын илэрхийлэл болон давталтанд ашиглаж болно. Доорхи код “area” аргын үр дүнд ирсэн утгыг ямар нэг хувьсагчид олгож байна.

```
int areaOfRectangle = new Rectangle(100, 50).area();
```

Тодорхой нэг объектын аргыг дуудах нь уг объект руу мэдээ илгээсэнтэй адил зүйл юм. Энэ тохиололд тухайн байгуулагчийн үр дүнд үүссэн “area” аргыг дуудаж буй объект нь тэгш өнцөгт юм.

Аргатай харьцах

Point болон Rectangle ангиуд дахь аргууд нь бүгд public –аар зарлагдсан байна. Тиймээс эдгээр нь бусад ангиуд хандахад нээлттэй юм. Зарим үед аргуудад хандах хандалтыг хязгаарлах нь тухайн ангид хэрэгтэй байдаг.

Жишээ нь: Тухайн нэг ангийн аргад зөвхөн түүний дэд анги л хандахыг зөвшөөрдөг байх. Хувьсагчиддаа хандах хандалтыг хянадагтай адил анги нь мөн аргууддаа хандах хандалтыг хянах механизмыг хэрэгжүүлж болдог.

Хэрэгцээгүй объектыг цэвэрлэх

Зарим объект хандалтад хэлэнд объектуудаа үүсгэж үүнийгээ нэг бүрчлэн бүртгэж, хэрэгцээгүй болсон үед нь өөрөө (программ зохиогч нь) устгах шаардлага гардаг. Санах ойг ингэж гардан зохион байгуулах нь төвөгтэй, алдаа гарах магадлал өндөр байдаг. Java платформ нь танд хүссэн хэмжээнийхээ объектыг үүсгэж болох бөгөөд үүнийг яаж устгах талаар санаа зовохгүй байх явдлыг олгодог. Хэрэгцээгүй болсон объектуудыг java –ын ажиллуулах орчин илрүүлээд автоматаар устгадаг. Үүнийг хог цуглуулах гэнэ. Тухайн объект руу заасан нэг ч заагч байхгүй болсон үед тэр объектыг устгана.

Хувьсагч цараанаас гармагц тухайн хувьсагч доторх заагч нь устдаг. Эсвэл тухайн хувьсагч руу null гэсэн тусгай утга олгох замаар объект заагчийг илээр устгаж болно. Нэг объект олон заагстай байж болохыг бид мэднэ, тиймээс объектыг устгахын тулд бүх заагчуудыг нь устгах хэрэгтэй.

Хог цуглуулагч

Заагчгүй болсон объектыг санах ойгоос үе үе чөлөөлдөг хог цуглуулагч java ажиллуулах орчин байдаг. Хог цуглуулагч нь энэ ажлаа автоматаар хийдэг, зарим үед System ангийн gc аргыг дуудах замаар хог цэвэрлэгчийг илээр ажиллуулж болно.

Жишээ нь : Хэрэв том хэмжээний хог үүсгэсний дараа, эсвэл их хэмжээний санах ой шаардсан кодын өмнө хог цуглуулагчийг зориудаар дуудаж болно. Тэгэхдээ ихэнх тохиолдолд хог цуглуулагчийг автоматаар ажиллуулах нь ашигтай байдаг.

Өндөрлөгөө

Хог цуглуулагч нь объектыг утстгахаасаа өмнө тухайн объектын finalize аргыг дуудаж уг объект өөрийгөө цэвэрлэх боломжийг олгоно. Үүнийг өндөрлөгөө гэнэ. Ихэнх програм зохиогчид finalize аргыг хэрэгжүүлэхэд санаа зовдоггүй. Ховорхон үед хог цуглуулагчийн хараа хяналтаас гадна орших нөөцийг чөлөөлөхөд finalize аргыг ашиглана. Finalize арга бол java платформын бүх ангиудын дээд өвөг болох объект ангийн нэг арга юм. Ямар ч анги тухайн төрлийнхөө (ангийнхаа) объектод өндөрлөгөө хийхийн тулд finalize аргыг дахин тодорхойлж болно. Хэрэв та уг аргыг дахин тодорхойлж байгаа бол бүх ажлаа хийж дууссаны дараа supre.finalize аргыг дуудах ёстой.

Дүгнэлт

Аливаа объект үүсгэхдээ new оператор болон байгуулагч ашиглана. New нь үүссэн объектынхоо заагчийг буцаана. Уг заагчийг хувьсагчид олгох юмуу шууд ашиглаж болно. Аливаа анги нь төл хувьсагч болон төл аргууд руу хандах хандалтыг зохицуулахдаа java платформын хандалтын механизмыг ашиглана. Өөр анги дотор тодорхойлогдсон нээлттэй төл хувьсагч болон төл аргатай харьцахын тулд нарийвчилсан нэрийг ашиглана. Төл хувьсагчийн нарийвчилсан нэр нь доорх хэлбэртэй.

```
objectReference.variableName
```

Аргын нарийвчилсан нэр нь :

```
objectReference.methodName(argumentList)
```

эсвэл

```
objectReference.methodName()
```

Хог цуглуулагч нь хэрэгцээгүй объектыг автоматаар цэвэрлэнэ. Нэг ч заагчгүй объектыг хэрэгцээгүй болсон объект гэдэг. Заагч агуулсан хувьсагч руу null утга олгох замаар заагчийг илээр устгаж болно.

Асуулт болон дасгал

Асуулт

1. Доорхи програмд ямар алдаа байна вэ?

```
public class SomethingIsWrong {
    public static void main(String[] args) {
        Rectangle myRect;
        myRect.width = 40;
        myRect.height = 50;
        System.out.println("myRect's area is " + myRect.area());
    }
}
```

2. Доорхи код нь нэг Point объект, нэг Rectangle объект үүсгэнэ. Кодыг ажиллуулсны дараа эдгээр объектуудын хичнээн заагч үлдэх вэ? Хоёр объект хоёулаа устах уу?

```
...
Point point = new Point(2,4);
Rectangle rectangle = new Rectangle(point, 20, 20);
point = null;
...
```

3. Үүсэгэсэн объектыг програм яаж устгадаг вэ?

Дасгал

1. Нэг дэхь асуултанд харуулсан SomeThingWrong()-ын алдааг зас
2. NumberHolder анги дор өгөгдлөө. Энэ ангийн төл үүсгэж, гишүүн өгөгдлийг нь цэнэглэж тэдгээрийн утгыг дэлгэцэнд гаргах код бич

```
public class NumberHolder {
    public int anInt;
    public float aFloat;
}
```

Тэмдэгтүүд ба Хэлхээнүүд

Java платформд тэмдэгт өгөгдөхүүнтэй харьцах зориулалттай дөрвөн анги байдаг.

- Character – Уг ангийн төл нь цорын ганц тэмдэгт утга агуулдаг. Энэ анги нь түүнчлэн ганц тэмдэгт өгөгдөлтэй харьцдаг олон аргыг агуулдаг.
- String – Олон тэмдэгтүүдээс бүрдсэн өөрчлөгддөггүй өгөгдөлтэй ажиллах зориулалттай анги.
- StringBuffer- Олон тэмдэгтээс бүрдсэн өөрчлөгддөг өгөгдөлтэй ажилладаг анги.
- StringBuilder – Зөвхөн нэг thread ашиглахад зориулагдсан StringBuffer ангийн цомхотгосон хэлбэр юм.

Энэ бүлэгт доорх сэдвүүдийг авч үзнэ.

- Тэмдэгтүүд
- Хэлхээ ба хэлхээн буфер
- Хэлхээ ба хэлхээн буфер үүсгэх
- Хэлхээ болон хэлхээн буферын уртыг авах
- Хэлхээ болон хэлхээн буферээс тэмдэгтийг индексээр авах
- Хэлхээн дотроос дэд хэлхээ буюу тэмдэгт хайх
- Хэлхээ болон хэлхээний хэсгүүдийг харьцуулах
- Тэмдэгт хэлхээтэй ажиллах
- Хэлхээн буферыг өөрчлөх
- Хэлхээ болон этхэтгүүр
- Дүгнэлт

Тэмдэгтүүд

Тэмдэгт төрлийн обьект нь ганц тэмдэгт утга агуулдаг. Char төрлийн хувьсагчийг обьект болгон хувиргахын тулд тэмдэгт обьектийг хэрэглэдэг. Жишээлбэл : Дамжуулсан хэмжигдэхүүнийхээ утгыг өөрчилдөг арга руу тэмдэгт утга дамжуулах эсвэл зөвхөн обьект хүлээн авах чадвартай ArrayList мэтийн өгөгдлийн бүтэц рүү тэмдэгт утга оруулах гэх мэт.

Цөөн тооны тэмдэгт обьект үүсгээд тэдгээрийнхээ тухай зарим мэдээллийг хэвлэх CharacterDemo програмыг доор орууллаа. Character ангитай холбоотой кодын хэсгийг дармалдаж орууллаа.

```
public class CharacterDemo {
   public static void main(String args[]) {
      Character a = new Character('a');
```

```
        Character a2 = new Character('a');
        Character b = new Character('b');

        int difference = a.compareTo(b);

        if (difference == 0) {
            System.out.println("a is equal to b.");
        } else if (difference < 0) {
            System.out.println("a is less than b.");
        } else if (difference > 0) {
            System.out.println("a is greater than b.");
        }
        System.out.println("a is "
                        + ((a.equals(a2)) ? "equal" : "not equal")
                        + " to a2.");
        System.out.println("The character " + a.toString() + " is "
            + (Character.isUpperCase(a.charValue()) ? "upper" : "lower")
            + "case.");
    }
}
```

Энэ програмын гарцыг доор үзүүллээ.

```
a is less than b.
a is equal to a2.
The character a is lowercase.
```

Дээрх жишээний Character.isUpperCase(a.CharValue()) код нь a гэдэг нэртэй character обьектын char утгыг авч байна. isUpperCase энэ аргын хэмжигдэхүүний төрөл нь char тул a.CharValue() гэж дуудах шаардлагатай болсон байна. Хэрвээ та JDK 5.0 –ийг ашиглаж байгаа бол автомат хувиргалтын давуу талыг ашиглан энэ арга руу character обьектыг шууд дамжуулж болно.

```
Character.isUpperCase(a)
```

CharacterDemo () програм нь Character ангид тодорхойлогдсон доорх байгуулагч болон аргуудыг дуудсан байна.

```
Character(char)
```

Аргумент нь өгсөн утгыг агуулсан character обьект өгсөн character ангийн цорын ганц байгуулагч. Character обьект үүсмэгц доторх утга нь дахин өөрчлөгддөггүй.

```
compareTo(Character)
```

Арга нь дуудагдаж буй обьект (манай жишээ a) болон арга руу нь дамжиж буй аргумент (манай жишээ b) 2 тэмдэгт обьектын утгуудыг харьцуулах төл арга.

Тухайн обьект доторх утга нь аргумент доторх утгаас их, бага эсвэл тэнцүү эсэхийг харуулсан бүхэл тоог энэ арга буцаадаг.

2 тэмдэгтийн тоон утгаараа ихтэй нь нөгөөгөөсөө их байна.

`equals(Object)`
Тухайн обьектын утгыг өөр обьект утгатай харьцуулах төл арга 2 обьектын утга хоёулаа тэнцүү бол энэ арга нь true буцаана.

`toString()`
Обьектыг хэлхээ болгон хөрвүүлдэг төл арга . Гарсан хэлхээ нь ганц тэмдэгтээс бүрдэх бөгөөд утга нь character обьект доторх утга байна.

`charValue()`
Character обьект доторх утгыг char утга болгон буцаадаг төл арга.

`isUpperCase(char)`
Өгөгдсөн char утга нь том үсэг эсэхийг шалгадаг анги арга. Энэ бол тэмдэгт өгөгдөхүүнтэй ажиллах зориулалттай character ангийн олон анги аргуудын зөвхөн нэг нь юм.

Character ангийн өргөн хэрэглэгддэг зарим аргуудыг доор хүснэгтээр харууллаа. Энэ ангийн бүх аргуудын бүрэн жагсаалтыг java.lang.Character гэсэн API тодорхойлолтоос хараарай.

Character ангийн өргөн хэрэглэгддэг зарим аргууд

Арга	Тайлбар
boolean isUpperCase(char) boolean isLowerCase(char)	Өгөгдсөн char утгыг том үсгээр эсвэл жижиг үсгээр байгаа эсэхийг нь тус тусдаа тогтооно
char toUpperCase(char) char toLowerCase(char)	Өгөгдсөн char утгыг том үсгээр эсвэл жижиг үсгээр буцаана
boolean isLetter(char) boolean isDigit(char) boolean isLetterOrDigit(char)	Өгөгдсөн char утгыг үсэг, тоо эсвэл үсэг тоо байгаа эсэхийг нь тус тусдаа тогтооно
boolean isWhitespace(char)[a]	Өгөгдсөн char утгыг Java платформд тодорхойлогдсон сул зай мөн эсэхийг тогтооно
boolean isSpaceChar(char)[b]	Өгөгдсөн char утгыг Юникод тодорхойлолтонд тодорхойлогдсон сул зайн тэмдэгт мөн эсэхийг тогтооно
boolean isJavaIdentifierStart(char)[c] boolean isJavaIdentifierPart(char)[d]	Өгөгдсөн char утгыг зөв идентификаторын эхлэлийн тэмдэгт болон зөв идентификатор хэсэг болох эсэхийг нь тус тусдаа тогтооно

a. Java платформ 1.1 –г гаргахад нэмсэн. isSpace(char) –г орлуулсан.
b. Java платформ 1.1 –г гаргахад нэмсэн.
c. Java платформ 1.1 –г гаргахад нэмсэн. isJavaLetter(char) –г орлуулсан.
d. Java платформ 1.1 –г гаргахад нэмсэн. isJavaLetterOrDigit(char) –г орлуулсан.

Хэлхээ болон хэлхээн буфер

Олон тэмдэгтээс бүрдэх тэмдэгт өгөгдөл болон хэлхээтэй ажиллах зориулалттай String ба String Buffer гэсэн 2 анги Java Platform -д ямагт багтсаар ирсэн. String анги нь утга нь өөрчлөгдөхгүй хэлхээтэй ажиллахад зориулагдсан. Жишээ: Таны бичсэн арга хэлхээ ашигладаг,тэр хэлхээ нь арга дотроо ямар нэгэн байдлаар өөрчлөгдөх ёсгүй бол энэ арга руу объект дамжуулах хэрэгтэй. String Buffer анги нь өөрчлөгдөх хэлхээтэй ажиллахад зориулагдсан ; тэмдэгт өгөгдлийн утга нь өөрчлөгдөх нь тодорхой үед String Buffer –г ашиглах хэрэгтэй. Гол төлөв тэмдэгт өгөгдлийг динамикаар байгуулах үед String Buffer –г ашиглана. Жишээ : файлаас тэмдэгт мөр унших гэх мэт.

JDK 5.0 – д шинээр нэмэгдсэн String Builder анги нь String Buffer –г хялбаршуулсан хувилбар юм.Түүнийг ашиглах нь String Buffer –г ашиглахтай адилхан гэвч түүнийг ганц thread ангид ашиглана.

Хэзээ ямар анги ашиглахаа шийдэхдээ доорхи зааварчилгааг баримтал.

- Та текстыг өөрчлөх шаардлага байхгүй бол хэлхээг ашиглана.
- Таны текстыг өөрчлөх гэхдээ зөвхөн single thread орчинд ашиглах бол string builder –г ашиглана.
- Таны текстыг өөрчлөх гэхдээ олон thread орчинд ашиглах бол string buffer –г ашиглана.

Хэлхээний тэмдэгтүүдийг оруулдаг StringsDemo програмыг доор харууллаа. Энэ програм нь хэлхээ (String) болон хэлхээн бүтээгч (String Builder) хоёрыг ашигласан байна. JDK 5.0 –аас өмнөх хувилбарыг ашиглаж буй хүмүүс хэлхээний бүтээгч ангийг хэлхээний буфераар соливол саадгүй ажиллах болно.

```
public class StringsDemo {
        public static void main(String[] args) {
                String palindrome = "Dot saw I was Tod";
                int len = palindrome.length();
                StringBuilder dest = new StringBuilder(len);
                for (int i = (len - 1); i >= 0; i--) {
                        dest.append(palindrome.charAt(i));
                }
                System.out.println(dest.toString());
        }
}
```

Уг програмын гарц нь:

```
doT saw I was toD
```

Хэлхээ ба хэлхээн буферыг үүсгэх

Хэлхээг гол төлөв хэлхээн литерал, тодруулбал давхар хашилт дотор бичигдсэн тэмдэгтийн цуваа маягаар зохион байгуулдаг. Жишээ нь : Java platform -д Gobbledygook утгатай String объект үүсгэхдээ “Gobbledygook” гэж бичнэ.

```
"Gobbledygook"
```

StringsDemo програмд дүрмийн дагуу хэлхээ үүсгэж түүндээ утга олгохдоо palindrome хувьсагчийг ашигласан байна.

```
String palindrome = "Dot saw I was Tod";
```

Та шинэ түлхүүр үг бас байгуулагч ашиглаж та хэлхээн объект үүсгэх өөр бусад Java объект үүсгэж чадна. Хэлхээн анги нь хэлхээний эхний утга, өөр тэмдэгт код ашиглах, тэмдэгтийн бүлээс хайх, байтын бүл, хэлхээн буфер, буюу хэлхээн байгуулагч зэргийг хангадаг. Доорх хүснэгтэнд хэлхээн ангийн байгуулагчийг харуулав.

* String анги дахь байгуулагчууд

Байгуулагч	Тайлбар
`String()`	Хоосон хэлхээ үүсгэнэ.

`String(byte[])`
`String(byte[], int, int)`
`String(byte[], int, int, String)`
 	
`String(byte[], String)`
	Хоёр бүхэл утга, тэр нь байт бүлийн шилжих утга бас урт, эхний утгыг барина. Тэмдэгтийн бүлийг тэмдэгтийн шиферээс тэмдэгтийн бүлрүү хөрвүүлэлт хийх тэмдэгтийг тодорхойлно.

`String(char[])` `String(char[], int, int)`	Char[ ] бүлээр хэлхээ байгуулна. Хоёр бүхэл утга, тэр нь char бүлийн шилжих утга бас урт, эхний утгыг барих.
`String(String)`	Хэлхээ үүсгэнэ. Тэр хэлхээ нь утгаа өөр хэлхээнээс авсан. Энэ байгуулагчийг ашигласнаар хэлхээн литералын утгыг recommend хийх биш харин 2 ижил утгатай хэлхээ үүсгэж байна.
`String(StringBuffer)`	Хэлхээ үүсгэнэ. Тэр хэлхээ нь утгаа өөр хэлхээн буфераас авсан.
`String(StringBuilder)`	Хэлхээ үүсгэнэ. Тэр хэлхээ нь утгаа өөр хэлхээн бүтээгчээс авсан.

* String анги нь уг хүснэгтэнд жагсаагаагүй бусад байгуулагчуудыг тодорхойлно. Тэдгээр байгуулагчуудын хэрэглээ нь сайшаагдаагүй.

Энэ жишээ нь тэмдэгтийн бүлээс хэлхээг үүсгэж байна.

```
char[] helloArray = { 'h', 'e', 'l', 'l', 'o' };
String helloString = new String(helloArray);
System.out.println(helloString);
```

Дээрх хэсэг кодын сүүлийн мөр нь ингэж хэвлэнэ: `hello`

String Buffer болон String Builder гэсэн 2 анги нь 2 ижил байгуулагчтай учир new командыг ашиглаж үүсгэнэ. Дор хэлхээн бүтээгчийн байгуулагчыг үзүүлэв.

StringBuffer анги дахь байгуулагчууд

Байгуулагч	Тайлбар
StringBuffer()	Анхны багтаамж нь 16 тэмдэгт байх хоосон хэлхээн буфер үүсгэнэ.

StringBuffer(CharSequence)	Өгөгдсөн CharSequence -тэй ижил тэмдэгтээс тогтсон хэлхээн буфер байгуулна. Энэ байгуулагч нь JDK 5.0 –д байдаг.
StringBuffer(int)	Багтаамжийг тодорхойлсон хоосон хэлхээн буфер үүсгэнэ.
StringBuffer(String)	Өгөгдсөн String –р цэнэглэгдсэн утга бүхий хэлхээн буфер үүсгэнэ. Багтаамж нь эх хэлхээний урт дээр 16 –г нэмсэнтэй тэнцүү.

StringsDemo програм нь хэлхээн бүтээгчийг dest ашиглан үүсгэхдээ буферын багтаамжийг тавьдаг байгуулагчийг хэрэглэж байна.

```
String palindrome = "Dot saw I was Tod";
int len = palindrome.length();
StringBuilder dest = new StringBuilder(len);
```

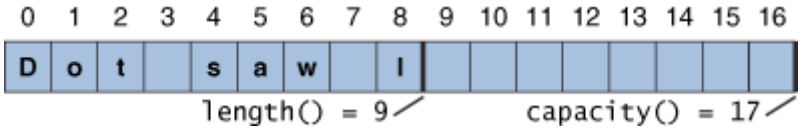
Дээрх код нь хэлхээн бүтээгчийг үүсгэхдээ анхны багтаамжийг palindrome нэрээр хэрэглэх хэлхээн урттай тэнцүүгээр авч байна. Энэ нь dest –д цорын ганц санах ойн хуваарилалт хийж байна. Яагаад гэвэл энэ нь хуулагдах тэмдэгтүүдийг агуулж чадахаар хэмжээтэй байна. Хэлхээн бүтээгчийг болон хэлхээн буферын багтаамжийг цэнэглэн, түүнд санах ойн хуваарилалтын хэдэн удаа хийх тоог багасгаж болно. Санах ойн хуваарилалт гэдэг нь харьцангуй үнэт ашиглалт учраас ингэснээрээ таны код илүү бүтээмжтэй болно.

Хэлхээн болон хэлхээн буферын уртыг авах

Обьектын талаар мэдээлэл авахад хэрэглэдэг аргуудыг хандагч аргууд гэнэ. Хэлхээ, хэлхээн буфер, хэлхээн бүтээгчтэй хамт хэрэглэдэг нэг хандагч арга нь length арга юм. Энэ арга нь тухайн обьектод агуулагдаж буй тэмдэгтүүдийн тоог буцаадаг. Дараах код хоёр мөр биелсэний дараа len нь 17 болно.

```
String palindrome = "Dot saw I was Tod";
int len = palindrome.length();
```

Нэмж хэлэхэд, StringBuffer болон StringBuilder ангиуд нь хэрэглэж байгаа зайнаас илүү хуваарилсан зайг буцаадаг capacity нэртэй аргатай байдаг. Жишээ нь: StringDemo програм дахь dest –д хэрэглэгдэж буй хэлхээн бүтээгчийн багтаамж нь хэдийгээр давталтын давталт бүрт урт нь нэгээр нэмэгдэж байгаа ч гэсэн хэзээ ч өөрчлөгдөхгүй. Дараах зураг нь есөн тэмдэгт нэмэгдсэний дараах багтаамж болон dest –н уртыг харууллаа.



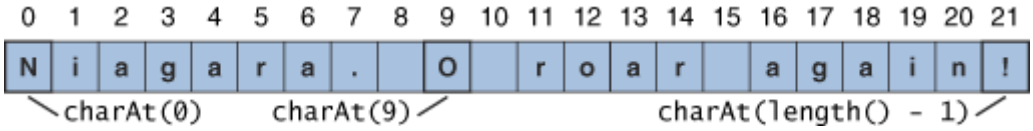
Хэлхээн буфер эсвэл хэлхээн бүтээгчийн урт нь түүний агуулах тэмдэгтүүдийн тоо байдаг. Түүний багтаамж нь хуваарилагдсан зайн дахь тэмдэгтүүдийн тоо байна. String анги нь capacity арга байдаггүй, учир нь хэлхээ өөрчлөгддөггүй.

Хэлхээ болон хэлхээн буфераас тэмдэгтийг индексээр авах

charAt хэмээх accessor аргыг дуудснаараа хэлхээ, хэлхээн буфер, хэлхээн бүтээгч доторх тодорхой индекс дэхь тэмдэгтийг авч болно. Индексийн эхний тэмдэгт нь 0, сүүлийнх нь `length()-1` байна. Жишээ нь: Дараах код нь хэлхээн дэхь 9 гэсэн индекс дэхь тэмдэгтийг авч байна.

```
String anotherPalindrome = "Niagara. O roar again!";
char aChar = anotherPalindrome.charAt(9);
```

Дараах зурганд үзүүлснээр индекс нь 0 –с эхлэн 9 –дэхь тэмдэгт нь ‘O’ байна.



Тодорхой индекс дэх тэмдэгтийг авахдаа charAt аргыг хэрэглэнэ. Дээрх зурагт харуулснаар хэлхээний сүүлийн тэмдэгтийн индексийг тооцоолж байна. Та length аргаас буцаасан утгаас нэгийг хасах ёстой.

Хэрэв та хэлхээ, хэлхээн буфер, эсвэл хэлхээн бүтээгчээс нэгээс олон тэмдэгт авах бол substring аргыг хэрэглэж болно. Дараах хүснэгтэнд үзүүлснээр substring арга нь хоёр хувилбартай.

`String`, `StringBuffer` болон `StringBuilder` ангиуд дахь * `substring` арга

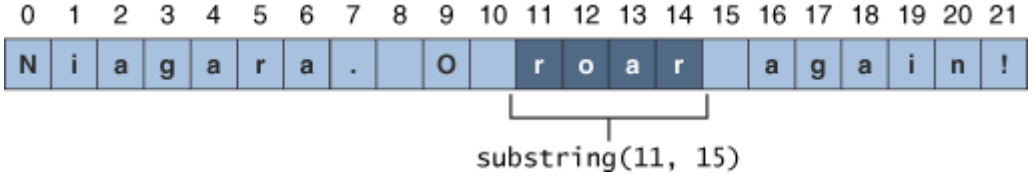
Арга	Тайлбар
`String substring(int)` `String substring(int, int)`	Уг хэлхээ, хэлхээн буфер, эсвэл хэлхээн бүтээгчийн дэд хэлхээ болох шинэ хэлхээг буцаана. Хоёрдох бүхэл тоон аргумент нь сүүлийн тэмдэгтийн `-1` индекс юм. Дэд хэлхээний урт нь эхний int –ээс хасах нь хоёрдох int юм. Хэрэв хоёрдох бүхэл тоог дамжуулахгүй бол, дэд хэлхээ нь эх хэлхээний төгсгөл хүртэл өргөтгөгдөнө.

- Substring аргууд нь Java 2 SDK 1.2 –т зориулан StringBuffer ангид нэмэгдсэн.

Дараах код нь хэлхээний 11 –с 15 хүртэлх индексийн тэмдэгтүүдийг дэд хэлхээ болгон авч байна.

```
String anotherPalindrome = "Niagara. O roar again!";
String roar = anotherPalindrome.substring(11, 15);
```

Хэлхээ, хэлхээн буфер, эсвэл хэлхээн бүтээгч дотроос хэсгийг нь авахдаа substring ашиглаарай.



Хэлхээн дотроос дэд хэлхээ юмуу тэмдэгт хайх

Хэлхээн доторх тодорхой тэмдэгт юмуу дэд хэлхээний байрлалыг буцаах indexOf болон lastIndexOf гэсэн хоёр хандагч арга string анги дотор байдаг. indexOf арга нь хэлхээний эхлэлээс төгсгөл рүү хайдаг бол lastIndexOf арга нь төгсгөлөөс эхлэл рүү хайдаг. Эдгээр аргуудын олон хэлбэрийг доор хүснэгтээр нэгтгэн харууллаа.

`String` анги доторх `indexOf` болон `lastIndexOf` аргууд

Арга	Тайлбар
`int indexOf(int)` `int lastIndexOf(int)`	Өгөгдсөн тэмдэгтийн эхний (сүүлийн) олдоцын индексийг буцаана
`int indexOf(int, int)` `int lastIndexOf(int, int)`	Өгөгдсөн индексээс эхлэн урагшаа (хойшоо) хайхдаа өгөгдсөн тэмдэгтийн эхний (сүүлийн) олдоцын индексийг буцаана
`int indexOf(String)` `int lastIndexOf(String)`	Өгөгдсөн хэлхээний эхний (сүүлийн) олдоцын индексийг буцаана
`int indexOf(String, int)` `int lastIndexOf(String, int)`	Өгөгдсөн индексээс эхлэн урагшаа (хойшоо) хайхдаа өгөгдсөн хэлхээний эхний (сүүлийн) олдоцын индексийг буцаана

StringBuffer болон StringBuilder ангиудад энэ хоёр арга байдаггүй. Хэрэв эдгээр обьект дээр ийм арга хэрэглэх шаардлага гарвал хэлхээний буферыг toString арга ашиглан хэлхээ болгон хувиргах хэрэгтэй.

Файлын нэрийг олон хэсэг болгон хуваахад lastIndexOf болон substring аргыг хэрхэн ашиглахыг доор үзүүлэхийг FileName анги харуулна.

Санамж: FileName анги доторх аргууд нь ямар ч алдаа шалгахгүй бөгөөд тэдгээрийн аргументууд нь директорын бүтэн нэр бол файлын нэр өргөтгөлтөйгөө байгаа хэмээн тооцсон. Хэрэв эдгээр аргууд нь үйлдвэрлэлийн код байсан бол аргументуудаа сайтар шалгах байсан.

```
// This class assumes that the string used to initialize
// fullPath has a directory path, filename, and extension.
// The methods won't work if it doesn't.
public class Filename {
    private String fullPath;
    private char pathSeparator, extensionSeparator;

    public Filename(String str, char sep, char ext) {
        fullPath = str;
        pathSeparator = sep;
        extensionSeparator = ext;
    }
```

```
    public String extension() {
        int dot = fullPath.lastIndexOf(extensionSeparator);
        return fullPath.substring(dot + 1);
    }

    public String filename() {
        int dot = fullPath.lastIndexOf(extensionSeparator);
        int sep = fullPath.lastIndexOf(pathSeparator);
        return fullPath.substring(sep + 1, dot);
    }

    public String path() {
        int sep = fullPath.lastIndexOf(pathSeparator);
        return fullPath.substring(0, sep);
    }
}
```

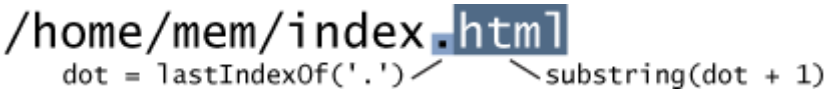
FileName обьектыг байгуулан бүх аргуудыг нь дуудсан жижиг програмыг доор харууллаа.

```
public class FilenameDemo {
    public static void main(String[] args) {
        final String FPATH = "/home/mem/index.html";
        Filename myHomePage = new Filename(FPATH,
                                    '/', '.');
        System.out.println("Extension = " +
            myHomePage.extension());
        System.out.println("Filename = " +
            myHomePage.filename());
        System.out.println("Path = " +
            myHomePage.path());
    }
}
```

Уг програмын гарц нь:

```
Extension = html
Filename = index
Path = /home/mem
```

Доорх зурганд харуулсанчлан бидний extension арга нь файлын нэр доторх цэгийн (.) сүүлчийн олдоцыг олохдоо lastIndexOf ашиглажээ. Дараа нь файлын нэрээс өргөтгөлийг нь ялгаж авахдаа substring нь lastIndexOf аргын буцах утгыг ашигласан байна. Цэгээс хэлхээний төгсгөл хүртэлх дэд хэлхээг авчээ.



Түүнчлэн extention арга нь substring аргын аргументыг `dot + 1` хэмээн бисэн байгааг анхаарна уу. Хэрэв цэг тэмдэгт нь мөрийн хамгийн сүүлийн тэмдэгт бол `dot + 1` нь

хэлхээний урттай тэнцэнэ. Тодруулбал хэлхээний урт нь хэлхээний хамгийн сүүлийн индексээс нэгээр илүү байх ёстой (учир нь индекс тэгээс эхлэдэг). Энэ бол substring –н алдаагүй зөв аргумент юм. Учир нь энэ арга нь хэлхээнийхээ уртаас илүүгүй индекс зааж авах бөгөөд энэ нь хэлхээний төгсгөл гэсэн санаа юмаа.

Хэлхээ болон хэлхээний хэсгүүдийг харьцуулах

String ангид хэлхээ болон хэлхээний хэсгүүдийг харьцуулах олон арга байдаг.

String анги дахь хэлхээг харьцуулах аргууд

Арга	Тайлбар

`boolean endsWith(String)`
`boolean startsWith(String)`
`boolean startsWith(String, int)`
	Аргынхаа аргументад өгсөн дэд хэлхээгээр уг хэлхээ эхлэх юмуу төгсөх тохиолдолд true буцаана. Хэлхээний аль цэгээс эхлэхийг бүхэл тоон аргумент нь заана.

`int compareTo(String)`
`int compareTo(Object)**`
`int compareToIgnoreCase(String)**`
	Хоёр хэлхээний утгыг харьцуулж уг хэлхээ нь аргументаасаа бага (хариу < 0), их (хариу > 0), эсвэл тэнцүү (хариу = 0) болохыг харуулсан бүхэл тоо буцаана. Харьцуулалт хийхээсээ өмнө object төрлийн аргыг хэлхээ болгон хөрвүүлдэг. com-pareToIgnore гэх арга үгийн том жижгийг тооцдоггүй. Иймд “a” болон “A” хоёрыг тэнцүү гэж үздэг.

`boolean equals(Object)` `boolean equalsIgnoreCase(String)`	Хэрэв энэ хэлхээ нь аргументтайгаа адилхан тэмдэгтүүдийн дараалал агуулж байвал true утга буцаана. Харьцуулалт хийхээсээ өмнө object төрлийн аргыг хэлхээ болгон хөрвүүлдэг. equalsIgnoreCase гэх арга үгийн том жижгийг тооцдоггүй. Иймд “a” болон “A” хоёрыг тэнцүү гэж үздэг.
`boolean regionMatches(int, String, int, int)` `boolean regionMatches(boolean, int, String, int, int)`	Энэхүү хэлхээний өгөгдсөн хэсэгт аргументууд өгөгдсөн хэсэгтэй тохирох эсэхийг шалгана. Үгийн том жижгийг тооцох эсэхийг boolean арга харуулна. Хэрэв true бол хэлхээг харьцуулахдаа үгийн том жижгийг тооцдоггүй.

* Аргууд Java 2 SDK 1.2 –т зориулан String ангид ** -тай хамт нэмэгдсэн.

Нэг хэлхээн дотроос хэлхээг хайхдаа RegionMatchesDemo програмыг доор харууллаа.

```
public class RegionMatchesDemo {
        public static void main(String[] args) {
                String searchMe = "Green Eggs and Ham";
                String findMe = "Eggs";
                int len = findMe.length();
                boolean foundIt = false;
                int i = 0;
                while (!searchMe.regionMatches(i, findMe, 0, len)) {
                        i++;
```

```
                        foundIt = true;
                }
                if (foundIt) {
                        System.out.println(searchMe.substring(i, i+len));
                }
        }
}
```

Уг програмын гарц нь: `Eggs`

Энэхүү програм нь SearchMe заагчыг заасан хэлхээн доторх тэмдэгтүүдтийг нэг нэгээр нь уншиж байна. Ингэхдээ энэ програм нь нэг тэмдэгт унших үедээ regionMatches аргыг дуудан хайж буй хэлхээтэй тохирох дэд хэлхээг тухайн тэмдэгтээс олдох хэсгийг шалгаж байна.

Тэмдэгт хэлхээтэй ажиллах

String ангид хэлхээг өөрчлөхөд зориулсан цөөн аргууд байдаг. Мэдээж хэлхээ өөрчлөгдөхгүй, тэгэхээр эдгээр аргууд үнэхээр юү хийх вэ гэвэл үр дүнг агуулсан хоёрдох хэлхээг үүсгэнэ, мөн буцаана. Дараах хүснэгтээр харууллаа.

Хэлхээтэй ажиллахад зориулсан String анги доторх аргууд

Арга	Тайлбар
`String concat(String)`	String аргументыг уг хэлхээний төгсгөлд холбоно. Хэрэв аргументын урт нь тэг (0) байвал эх хэлхээ обьект нь буцаагдана.
`String replace(char, char)`	Эхний аргументаар өгөгдсөн тэмдэгтийг хоёрдох аргументаар өгөгдсөн тэмдэгтээр солино. Хэрэв ямар нэг орлуулалт шаардлагагүй бол эх хэлхээн обьект нь буцаагдана.
`String trim()`	Хэлхээний төгсгөлөөс хоосон зайг хасна.
`String toLowerCase()` `String toUpperCase()`	Уг хэлхээг жижиг эсвэл том тэмдэгт рүү хөрвүүлнэ. Хэрэв ямар нэг хөрвүүлэлт шаардлагагүй бол эх хэлхээг буцаана.

Дараах BostonAccentDemo програм нь хэлхээг Boston –ны нутгийн аялга руу хөрвүүлэхдээ replace аргыг ашиглаж байна.

```
// This example demonstrates the use of the replace method
// in the String class.

public class BostonAccentDemo {
    private static void bostonAccent(String sentence) {
        char r = 'r';
        char h = 'h';
        String translatedSentence = sentence.replace(r, h);
        System.out.println(translatedSentence);
    }
```

```
    public static void main(String[] args) {
        String translateThis =
          "Park the car in Harvard yard.";
        bostonAccent(translateThis);
    }
}
```

replace арга нь sentence хэлхээний бүх “r” үсгүүдийг “h” үсгээр солиж байна. Гарц нь:

Pahk the cah in Hahvahd yahd.

Хэлхээн буферыг өөрчлөх

Хэлхээн буфер болон хэлхээн бүтээгч нь үүссэнийхээ дараа агуулга багтаамжаа өөрчилж болдог. StringBuffer болон StringBuilder ангиуд нь өөрсдийнхөө өгөгдлийг өөрчлөхөд зориулсан олон аргуудтай байдаг. Дараах хүснэгтэнд хэлхээн буферыг өөрчлөхөд хэрэглэдэг аргуудыг нэгтгэн харууллаа. StringBuilder ангид байдаг аргууд нь эдгээртэй адилхан гэхдээ хэлхээн буфер буцаахын оронд хэлхээн бүтээгч буцаадаг.

Хэлхээн буферыг өөрчлөхөд зориулсан * аргууд

Арга	Тайлбар

`StringBuffer append(boolean)`
`StringBuffer append(char)`
`StringBuffer append(char[])`
`StringBuffer append(char[], int, int)`
`StringBuffer append(double)`
`StringBuffer append(float)`
`StringBuffer append(int)`
`StringBuffer append(long)`
`StringBuffer append(Object)`
`StringBuffer append(String)`
	Тухайн аргументыг уг хэлхээн буферт нэмнэ. Нэмэх үйлдэл явагдахаас өмнө өгөгдөл нь хэлхээ рүү хөрвүүлэгдэнэ.

`StringBuffer delete(int, int)**` `StringBuffer deleteCharAt(int)**`	Уг хэлхээн доторх өгөгдсөн тэмдэгтийг усгана.

`StringBuffer insert(int, boolean)`
`. StringBuffer insert(int, char)`
`StringBuffer insert(int, char[])`
`StringBuffer insert(int, char[], int, int)**`
 	
`StringBuffer insert(int, double)`
`StringBuffer insert(int, float)`
`StringBuffer insert(int, int)`
`StringBuffer insert(int, long)`
`StringBuffer insert(int, Object)`
`StringBuffer insert(int, String)`
	Хоёр дах аргументыг хэлхээн буфер луу оруулна. Эхний аргумент нь тухайн өгөгдлийг оруулахаас өмнө индексийг заана.

`StringBuffer replace(int, int, String)**` `void setCharAt(int, char)`	Уг хэлхээн доторх өгөгдсөн тэмдэгтийг солино.
`StringBuffer reverse()`	Уг хэлхээн буфер доторх тэмдэгтүүдийн дарааллыг эсрэг эргүүлнэ.

* * тэмдэглэгдсэн * аргууд нь Java 2 SDK –д зориулж StringBuffer ангид нэмэгдсэн.

Та энэ бүлгийн эхэнд StringDemo програмд биелэгдэж байгаа append аргыг харсан. Дараах InsertDemo програм нь хэлхээн буфер руу хэлхээг оруулахдаа insert аргыг ашиглаж байна.

```
public class InsertDemo {
    public static void main(String[] args) {
        StringBuffer palindrome = new StringBuffer(
                        "A man, a plan, a canal; Panama.");
        palindrome.insert(15, "a cat, ");
        System.out.println(palindrome);
    }
```

```
}
```

Уг програмын гарц нь мөн л:

```
A man, a plan, a cat, a canal; Panama.
```

insert ашиглан оруулах өгөгдлийнхөө индексийг өмнө нь зааж өгч болно. Жишээнд, “a cat” –г зааж буй 15 нь “a canal” дахь эхний “a” –н өмнө орно. Өгөгдлийг хэлхээн буферын эхэнд оруулахдаа 0 индексийг ашиглана. Өгөгдлийг хэлхээн буферын адагт нь нэмэхдээ хэлхээн буферын одоогийн урттай тэнцүү индекс ашиглана эсвэл append ашиглана.

Хэрэв хэлхээн буферыг өөрчлөх үйл ажиллагаа нь одоогийн багтаамжийн хэмжээг өгсөж байгаа бол, хэлхээн буфер нь илүү санах ой хуваарилна. Өмнө дурдсанчлан санах ойн хуваарилалт гэдэг нь харьцангуй үнэт үйл ажиллагаа бөгөөд та хэлхээн буферын хэмжээг анхнаасаа ухаалагаар цэнэглэх юм бол таны код илүү үр бүтээмжтэй болно.

Хэлхээ болон эмхэтгүүр

Эмхэтгүүр нь String болон StringBuffer ангиудыг цаанаа үгчилсэн хэлхээнүүдтэй ажиллахад хэрэглэдэг. Үгчилсэн хэлхээг давхар хашилт дотор тусгайлж заадаг.

```
"Hello World!"
```

String обьектыг ашиглаад хэлхээг хаана ч хэрэглэж болно. Жишээ нь: System.out.println нь хэлхээний аргументыг хүлээн авдаг болохоор үгчилсэн хэлхээг түүнд хэрэглэхдээ:

```
System.out.println("Might I add that you look lovely today.");
```

Мөн хэлхээн аргуудыг үгчилсэн хэлхээнээс шууд ашиглаж болно.

```
int len = "Goodbye Cruel World".length();
```

Эмхэтгүүр нь үгчилсэн хэлхээ болгонд шинэ хэлхээн обьектыг автоматаар үүсгэдэг учраас, хэлхээг цэнэглэхийн тулд үгчилсэн хэлхээг ашиглахдаа:

```
String s = "Hola Mundo";
```

Энэ нь өмнөх байгуулагчтай адилхан. Гэхдээ өмнөх нь илүү бүтээмжтэй.

```
String s = new String("Hola Mundo"); //don't do this
```

Хэлхээнүүдийг холбохдоо “+” тэмдэг ашиглана.

```
String cat = "cat";
System.out.println("con" + cat + "enation");
```

Цаанаа, эмхэтгүүр нь хэлхээн буферуудыг холболт гүйцэтгэхэд хэрэглэнэ. Өмнөх жишээ нь эмхэтгэгдэхдээ:

```
String cat = "cat";
System.out.println(new StringBuffer().append("con").
append(cat).append("enation").toString());
```

Мөн “+” оператор ашиглан хэлхээн утганд уг хэлхээн бус хэлхээг нэмж болно.

```
System.out.println("You're number " + 1);
```

Эмхэтгүүр нь хэлхээ холбох үйл ажиллагааг биелүүлэхээсээ өмнө string бус утгыг (жишээн дэх 1 гэсэн бүхэл тоог) string обьект болгон цаагуураа хөрвүүлдэг.

Дүгнэлт

Дан ганц тэмдэгт агуулах бол Character обьектыг ашиглаарай, өөрчлөгдөхөөргүй тэмдэгтүүдийн дарааллыг агуулах бол String обьектыг ашиглаарай, мөн тэмдэгтүүдийн дарааллыг динамикаар өөрчлөх юмуу байгуулах бол StringBuffer юмуу StringBuilder ашиглаарай. Доор Palindrome нэртэй инээдтэй програм нь хэлхээг палиндром эсэхийг тогтоож байна. Уг програм нь String болон StringBuilder ангиудын олон аргыг ашигласан байна.

```
public class Palindrome {

   public static boolean isPalindrome(String stringToTest) {
      String workingCopy = removeJunk(stringToTest);
      String reversedCopy = reverse(workingCopy);

      return reversedCopy.equalsIgnoreCase(workingCopy);
   }

   protected static String removeJunk(String string) {
      int i, len = string.length();
       StringBuilder dest = new StringBuilder(len);
       char c;

       for (i = (len - 1); i >= 0; i--) {
           c = string.charAt(i);
           if (Character.isLetterOrDigit(c)) {
               dest.append(c);
           }
       }

      return dest.toString();
   }

   protected static String reverse(String string) {
       StringBuilder sb = new StringBuilder(string);

      return sb.reverse().toString();
   }

   public static void main(String[] args) {
      String string = "Madam, I'm Adam.";

      System.out.println();
```

```
        System.out.println("Testing whether the following "
                           + "string is a palindrome:");
        System.out.println("    " + string);
        System.out.println();

        if (isPalindrome(string)) {
            System.out.println("It IS a palindrome!");
        } else {
            System.out.println("It is NOT a palindrome!");
        }
        System.out.println();
    }
}
```

Уг програмын гарц нь:

```
Testing whether the following string is a palindrome:
    Madam, I'm Adam.

It IS a palindrome!
```

Асуулт болон дасгал

Асуулт

1. Дараах хэлхээн буферын анхны багтаамж нь ямар байна?

```
StringBuffer sb = new StringBuffer("Able was I ere I saw Elba.");
```

2. Дараах хэлхээг авч үзье.

```
String hannah = "Did Hannah see bees? Hannah did.";
```

   a. `hannah.length()` илэрхийлэлээр хэвлэгдсэн утга ямар байх вэ?
   b. `hannah.charAt(12)` дуудсан аргын буцаах утга нь ямар байх вэ?
   c. `hannah` –р хэрэглэгдсэн хэлхээн дэх `b` үсэгт хэрэглэгдэх илэрхийлэл бич.

3. Дараах илэрхийллээр буцааж буй хэлхээ нь хэр урт байх вэ?

```
"Was it a car or a cat I saw?".substring(9, 12)
```

4. Дараах ComputeResult нэртэй програмд, бүх дугаарласан мөрүүд биелсэний дараа `result` –н утга ямар байх вэ?

```
public class ComputeResult {
    public static void main(String[] args) {
        String original = "software";
        StringBuffer result = new StringBuffer("hi");
        int index = original.indexOf('a');

/*1*/   result.setCharAt(0, original.charAt(0));
/*2*/   result.setCharAt(1, original.charAt(original.length()-1));
/*3*/   result.insert(1, original.charAt(4));
/*4*/   result.append(original.substring(1,4));
```

```
/*5*/   result.insert(3, (original.substring(index, index+2) + " "));

        System.out.println(result);
    }
}
```

Дасгал

1. Дараах хоёр хэлхээг холбон `"Hi, mom."` хэлхээ болгох хоёр арга замыг харуул.

```
String hi = "Hi, ";
String mom = "mom.";
```

2. Өөрийн нэрний уртыг тооцоолон дэлгэцэнд хэвлэх програм бич.
3. Анаграм бол өөр үгийн эсвэл хэллэгийн үсгүүдийг сольж үүсгэсэн үг эсвэл хэллэг болно. Жишээ нь: “parliament” бол “partial men” –ын анаграм мөн “software” бол “swear oft” –ын анаграм болно. Ямар нэгэн хэлхээ нь өөр хэлхээний анаграм болох эсэхийг тогтоох програм бич. Хоосон зай болон тэмдэгтүүдийг (!, ?, …) тооцохгүй нь дээр шүү.

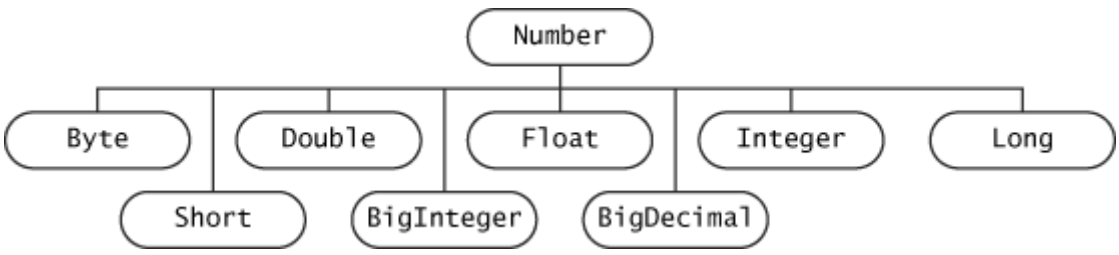
Тоонууд

Энэ бүлгийн эхэнд java.lang доторхи Number ангийг болон түүний дэд ангиудыг хэлэлцэнэ. Тодруулбал, яагаад эдгээр ангиуд нийтийн аргуудаас гаднах тэмдэгтүүд болон нийтийн доторхи анги хувьсагчдыг хэрэглэх тухай өгүүлж, мөн яаж төлүүдийг нь хэлхээ төрөл рүү хөрвүүлэхийг харуулна.

Нэмж хэлэхэд танд ажиллахад хэрэг болж мэдэх тоон төрөл доторхи бусад ангиудын талаар өгүүлнэ. Тохиолдлоор, хэрэв та дэлгэцэнд тоог тодорхой форматаар гаргах хэрэгтэй бол java.text дахь NumberFormat болон DecimalFormat ангиудыг ашиглаж болно. Хэрэв та JDK 5.0 болон түүнээс дээш хувилбарыг хэрэглэж байгаа бол, форматтай хэлхээг гарцын цацраг урсгал руу бичдэг PrintStream –ын printf боломж нь тохирсон аргыг тань хангаж өгнө. Мөн java.lang доторхи Math анги нь математик функцуудыг гүйцэтгэдэг анги аргыг агуулсан. Энэ ангид тригонометр функцууд, илтгэгч функцууд гэх мэт аргуудыг мөн багтаасан байдаг.

Тооны ангиуд

Дараах зураг нь Java платформ дээр хангагдсан тоон төрлийн ангиудын иерархийг харуулав.



Тоон ангийн талаар нэмж хэлэхэд, Java платформ нь Boolean, Character, ба Void гэх тоон ангиудтай хамтардаг type-wrapper гэдэг ангиудыг агуулсан байдаг.

Үндсэн өгөгдлийн төрлүүдийг хуулбарласан юм шиг type-wrapper ангиудыг заавал хэрэгтэй болсоныг гайхаж магадгүй. Type-wrapper ангиуд нь цөөн хэрэглээтэй.

- Обьект нь шаардлагатай эсэх нь хамаарахгүйгээр үндсэн төрлийн утгыг type-wrapper обьектод хадгалж болдог. Жишээ нь ArrayList анги зөвхөн обьектуудыг авдаг, тэгэхээр ArrayList –д тоонуудыг оруулахын тулд type-wrapper обьект дахь утга бүрийг ороож (өлгийдөж) обьектоо үүн лүү хангаж өгнө. Тэмдэгтийн талаар хэлэлцүүлэгийг өмнө үзсэн шиг, үндсэн төрлөөс обьект руу хөрвүүлэх нь JDK 5.0 –тай цуг ажиллаж эхэлснээр автоматаар явагддаг.
- Ангиуд нь өгөгдлийн төрлийн талаар ерөнхий мэдээллийг хангасан MIN_VALUE, MAX_VALUE мэтийн хэрэгцээтэй хувьсагчуудийг тодорхойлдог. Ангиуд нь утгуудыг ондоо төрөл рүү хөрвүүлэх, хэлхээ рүү хөрвүүлэх мэтийн хэрэглээтэй аргуудыг мөн тодорхойлдог.
- Ангиуд нь JVM доторхи ямар нэг обьект эсвэл ангийн талаархи мэдээллийг цуглуулахыг програмд зөвшөөрсөн Java ойлт (тусгал) –ын механизмд хэрэглэгддэг.

Үүнээс гадна BigInteger болон BigDecimal нь үндсэн өгөгдлийн төрлүүдийг өргөтгөхдөө тохиолдлын нарийн тоонуудад (үндсэн өгөгдлийн төрлүүдийн аль нэг рүү таардаггүй тоонуудад) зөвшөөрөхийн тулд өргөтгөсөн. Гэхдээ бусад ангиуд java.lang багцад, BigDecimal болон BigInteger нь java.math багцад байдаг.

Доор NumberDemo гэдэг хоёр float обьект болон нэг double обьектыг үүсгээд дараа нь compareTo болон equals –ыг ашиглан тэднийг харьцуулж байна.

```
public class NumberDemo {
    public static void main(String args[]) {
        Float floatOne = new Float(14.78f - 13.78f);
        Float floatTwo = Float.valueOf("1.0");
        Double doubleOne = new Double(1.0);

        int difference = floatOne.compareTo(floatTwo);

         if (difference == 0) {
         System.out.println("floatOne is equal to floatTwo.");
        } else if (difference < 0) {
         System.out.println("floatOne is less than floatTwo.");
        } else if (difference > 0) {
         System.out.println("floatOne is greater than floatTwo.");
        }

        System.out.println("floatOne is "
                        + ((floatOne.equals(doubleOne)) ?
                        "equal" : "not equal")
                        + " to doubleOne.");

    }
}
```

Уг програмын гарц нь магадгүй таныг жаахан гайхашруулах байх:

```
floatOne is equal to oneAgain.
floatOne is not equal to doubleOne.
```

floatOne болон dubleOne –д агуулагдсан хувьсагчууд нь хэдийгээр хоёулаа тоогоор 1 – тэй тэнцүү боловч, обьектууд нь өөр өөр төрөл учраас хоорондоо тэнцэхгүй гэж үзнэ.

Өмнөх жишээн дээр хэрэглэгдсэн comapreTo болон equals аргуудыг оролцуулаад, Number ангийн бүх дэд ангиудыг багтаасан төл аргуудыг доор хүснэгтээр харууллаа.

Тоон ангиудыг жирийн төл аргууд

Арга	Тайлбар

`byte byteValue()**`
`short shortValue()**`
`int intValue() long longValue()`
`float floatValue()`
`double doubleValue()`
	Энэ тоон обьектын утгыг byte, short, int, long, float, болон double үндсэн өгөгдлийн төрлүүд рүү хувиргана.

`int compareTo(Integer)***` `int compareTo(Object)***`	Энэ тоон обьектыг аргументтай харьцуулна. Энэ арга нь тоон обьектыг аргументаас бага, тэнцүү, эсвэл их гэдгийг харуулсан тоог буцаадаг.
`boolean equals(Object)`	Энэ тоон төрөл нь аргументтай тэнцүү эсэхийг тогтооно.

** Number ангиуд болон түүний дэд ангиудыг LDK 1.1 –д зориулан нэмсэн
*** Number дэд ангиудыг Java 2 SDK 1.2 –д зориулан нэмсэн

Бүлэг шиг, Number дэд ангиуд нь мөн зарим хэрэгтэй тогтмолуудыг агуулдаг. Тогтмолудд нь public static зарлагддаг учраас Integer.MIN_VALUE гэдэг шиг ангийн нэрээ тогтмолын нэртэй нь цэг “.” тавьж холбоно.

Доорх хүснэгт нь Float болон Double ангиуд дахь бусад хэрэгтэй тогтмолуудыг харууллаа.

Float болон Double ангиуд дахь бусад хэрэгтэй тогтмолууд

Арга	Тайлбар
`Float.NaN` `Double.NaN`	Number төрөл биш. Тухайн аргаар дамжуулсан аргументуудыг тодорхойлоогүй үед java.lang.Math анги дахь тодорхой аргуудаар буцаагдсан.
`Float.NEGATIVE_INFINITY` `Double.NEGATIVE_INFINITY`	float эсвэл double төрөл дахь сөрөг хязгааргүй утга.
`Float.POSITIVE_INFINITY` `Double.POSITIVE_INFINITY`	float эсвэл double төрөл дахь эерэг хязгааргүй утга.

Хэлхээг тоон төрөл рүү хөрвүүлэх

Заримдаа, хэлхээн обьект (жишээ нь хэрэглэгчийн оруулсан утга) дотроос тоон өгөгдөл авах шаардлага гардаг. Тоон type-wrapper ангиуд (byte, integer, double, float, long, болон short) нь valueOf гэсэн нэртэй хэлхээг тухайн төрлийн обьектууд руу хөрвүүлдэг анги аргатай байдаг. ValueOfDemo жижиг жишээ нь командын мөрнөөс хоёр хэлхээг аваад тоон төрөл рүү хөрвүүлээд, дараа нь утгууд дээр нь арифметик үйлдэл гүйцэтгэж байна.

```
public class ValueOfDemo {
```

```
    public static void main(String[] args) {

        //this program requires two arguments on the command line
       if (args.length == 2) {

          //convert strings to numbers
          float a = Float.valueOf(args[0]).floatValue();
          float b = Float.valueOf(args[1]).floatValue();

          //do some arithmetic
          System.out.println("a + b = " + (a + b) );
          System.out.println("a - b = " + (a - b) );
          System.out.println("a * b = " + (a * b) );
          System.out.println("a / b = " + (a / b) );
          System.out.println("a % b = " + (a % b) );
       } else {
          System.out.println("This program requires two command-line
arguments.");
       }
    }
}
```

Командын мөрний аргументад 4.5 болон 87.2 гэж хэрэглэхэд програмын гарцыг доор харууллаа.

```
a + b = 91.7
a - b = -82.7
a * b = 392.4
a / b = 0.0516055
a % b = 4.5
```

Тоон төрлийг хэлхээ рүү хөрвүүлэх

Заримдаа, тоон төрлийг хэлхээ рүү хөрвүүлэх шаардлага гардаг. Бүх ангиуд нь toString нэртэй аргыг Object ангиас удамшдаг. Дараах ToStringDemo програм нь тоог хэлхээ рүү хөрвүүлэхэд toString аргыг ашиглаж байна. Дараа нь прогрм аравтын таслалаас өмнөх болон хойших орнуудын тоог тооцоолсон зарим хэлхээ аргуудыг ашиглаж байна.

```
public class ToStringDemo {
    public static void main(String[] args) {
       double d = 858.48;
       String s = Double.toString(d);

       int dot = s.indexOf('.');
       System.out.println(s.substring(0, dot).length()
                       + " digits before decimal point.");
       System.out.println(s.substring(dot+1).length()
                       + " digits after decimal point.");
    }
}
```

Дээрх програмын гарц нь:

```
3 digits before decimal point.
2 digits after decimal point.
```

Уг програмаар дуудагдаж буй toString арга нь анги арга юм. Тоон төрлийн анги бүр нь тухайн төрлийн төлийг дуудсан toString гэдэг төл аргатай.

System.out.println аргыг тоонуудыг дэлгэцэнд гаргахын тулд toString аргыг заавал илэрхийгээр дуудаад байх албагүй. Java платформ нь toString –г далдуур дуудсанаараа хөрвүүлэлтийг зохион байгуулна.

Тоог форматлах

toString арга нь энгийн хөрвүүлэлтэнд тохирсон гэхдээ та гарцын форматыг таашаахгүй байж мэднэ. Тохиолдлоор програмд мөнгөн утга дүрсэлж буй хөвөх таслалтай тоон аравтын тооллын зөвхөн хоёр цэгтэй форматтай байх нь дээр. Програмын гарцын форматыг илүү хянах бол NumberFormat анги болон түүний дэд болох DecimalFormat –ыг хэрэглэж болно. Double мэтийн үндсэн төрөл болох тоонуудын форматыг тавих бол Double мэтийн тэдгээрт нь тохирсон ороогч (wrapper) обьектуудыг нь хэрэглээрэй. NumberFormat болон DecimalFormat ангиуд нь java.text багцад байдаг.

Доорх жишээ код нь Double форматыг тавьж байна. getNumberInstance нь NumberFormat –ын төлийг буцаадаг завод арга юм. Format арга нь Double –ыг аргумент мэтээр хүлээн авдаг ба хэлхээнд форматтай тоог буцаадаг.

```
Double amount = new Double(345987.246);
NumberFormat numberFormatter;
String amountOut;
numberFormatter = NumberFormat.getNumberInstance();
amountOut = numberFormatter.format(amount);
System.out.println(amountOut);
```

Кодын сүүлийн мөр нь 345, 987.246 гэж дэлгэцэнд хэвлэнэ.

Санамж : Өмнөх кодыг ажиллуулан таны харсан гарц нь магадгүй харагдаж байгаагаасаа өөр байх. Яагаад гэвэл NumberFormat болон DecimalFormat ангиуд нь тухайн орон нутгийн байрлалд таарах гарцыг эсгэдэг нутгийн хамааралтай байдаг. Байрлал нь тусгай газар зүйн, улс төрийн, соёлын орон нутгийг ялгаж тодорхойлсон обьект юм. Өмнөх кодод орон нутгийн байрлалыг нь яг илэрхий тавьж өгөөгүй, гэсэн ч тоон форматын обьект нь JVM –ны одоогийн даллагад оршмол (default) байрлалыг хэрэглэдэг. Оршмол байрлал нь US –ыг зааж байх тохиолдолд гарсан гарц нь таны үр дүн юм. Locale.getDefault аргыг ашиглан одоогийн оршмол байрлалийг аль байгааг мэдэж болно. Мөн Locale.getDefault –ийг ашиглан байрлалыг өөрчилж болно.

Хэрэв та тооны формат обьектыг үүсгэх үед JVM -ны одоогийн даллагад оршмол байрлалыг өөрчлөх сонголт нь байрлалыг тусгайлан заадаг. Оршмол байрлал ашиглахын оронд тооны форматын обьект нь дөнгөж үүсэн одоо заагдсан байгааг нь ашигладаг. Францыг заасан байрлалтай гарцыг эсгэсэн тооны формат обьектыг яаж үүсгэх вэ гэвэл:

```
NumberFormat                    numberFormatter                    =
NumberFormat.getNumberInstance(Locale.FRANCE);
```

Энэ тэмдэглэл нь DecimalFormat ангийг ашигласныг багтаан бүх форматын жишээнүүдэд хэрэглэгдэнэ. Илүү мэдээллийг International Trial –аас лавлаарай.

Мөнгөн утгыг форматлах

Хэрэв та бизнесийн хэрэглэгдэхүүн бичиж байгаа бол магадгүй мөнгөн утгыг форматлаж дэлгэцэнд гаргах хэрэгтэй байх.

Форматлагчыг үүсгэх getCurrencyInstance -ыг дуудахгүйгээр тоо шиг хэв маягаар форматлана. Формат аргыг дуудах үед тэр нь форматлагдсан тоо болон зохистой мөнгөн тэмдэгтийг агуулсан хэлхээг буцаана.

Дараах код нь мөнгөн утгыг яаж форматлахыг харууллаа.

```
Double currency = new Double(9876543.21);
NumberFormat currencyFormatter;
String currencyOut;

currencyFormatter = NumberFormat.getCurrencyInstance();
currencyOut = currencyFormatter.format(currency);
System.out.println(currencyOut);
```

Кодын сүүлийн мөр нь ,876,543.21 гэж дэлгэцэнд хэвлэнэ.

Зууны хувийг форматлах

NumberFormat ангийн аргуудыг мөн зууны хувийг форматлахад хэрэглэнэ. Орон нутгийн байрлалын тусгай форматлагчыг авахын тулд getPercentInstance аргыг дуудна. Энэ форматлагчаар 0.75 мэтийн аравтын бутархайг 75 % гэж дэлгэцэнд хэвлэдэг.

Дараах код нь зууны хувийг яаж форматлахыг харууллаа.

```
Double percent = new Double(0.75);
NumberFormat percentFormatter;
String percentOut;

percentFormatter = NumberFormat.getPercentInstance();
percentOut = percentFormatter.format(percent);
System.out.println(percentOut);
```

Кодын сүүлийн мөр нь 75% гэж дэлгэцэнд хэвлэнэ.

Printf боломжийг хэрэглэх нь

JDK 5.0 нь гарцыг форматлах ажлыг маш хялбар болгосон printf боломжийг танилцуулсан. Арга нь java.io.PrintStream –ээр тодорхойлогдсон ба дараах хэллэгийг агуулдаг .

```
public PrintStream printf(String format, Object... args)
```

Format гэх анхны аргумент нь args гэсэн дараагийн аргумент дахь обьектуудыг яаж форматлагдсаныг нь зааж буй форматын хэлхээ юм.

Энэ аргыг хэрэглэхийн тулд та эхлээд заавал формат хэлхээний өгүүлбэр зүйг ойлгох ёстой. Аз болоход энэ ангид зориулагдсан API –ын тодорхойлолтууд нь дэлгэрэнгүй баримт бичигтэй.

Форматын хэлхээ нь энгийн текстийг агуулж чадах хэлхээ юм. Форматын тодорхойлогчууд нь Object... args аргументуудыг форматлах онцгой тэмдэгтүүд байна. Object... args бичлэг нь varargs гэсэн нэртэй 5.0 өгүүлбэр зүй юм. Энэ нь аргументуудын тоо нь хувьсдаг гэсэн үг. API тодорхойлолт нь дараах жишээг өгдөг.

```
Calendar c = ...;
String s = String.format("Duke's Birthday: %1$tm %1$te,%1$tY",c);
```

Форматын тодорхойлогчууд нь танихад амархан байсан нь дээр.

Calendar обьектын С –д хэрэглэгддэгээс 3 нь: %1$tm, %1$te болон %1$tY ийм байна. Энэ жишээн дэх формат тодорхойлогчийг доор жагсаан харууллаа.

%	формат тодорхойлогчийн эхлэл
1$	эхний аргумент
tm	сар
te	сарын өдөр
tY	жил

Форматын тодорхойлогч нь энэ мэт санахад хэцүү онцгой тэмдэгтүүдийг хэрэглэдэг учраас API тодорхойлолтоос гүйцэт жагсаалтыг нь лавлах хэрэгтэй.

Хэрэглэж буй өөрийн өгүүлбэр зүйг нь ганц удаа хэрэглэж үзэхэд л printf арга нь тун тохиромжтой гэдгийг та мэдэх болно.

Тоог өөрийнхөөрөө форматлах

Аравтын тоонуудыг хэлхээ рүү форматлахдаа DecimalFormat ангийг хэрэглэж болно. Энэ анги нь эхлэл болон төгсгөл тэгүүдийг, prefix болон suffix, мянганы болон аравтын орны тусгаарлагчуудын дэлгэцэнд гаргахыг нь удирдах боломж олгосон. Аравтын тусгаарлагч мэтийн форматлагдсан тэмдэгтүүдийг өөрчлөхийг хүсвэл DecimalFormatSymbols –ыг DecimalFormat ангитай холбон хэрэглэж болно. Энэ ангиуд нь тоонуудын форматлалтанд маш их уян хатан байдлыг бий болгоно. Гэхдээ таны кодыг илүү ярвигтай болгож болно.

Дараах текст нь DecimalFormat болон DecimalFormatSysbols ангиудыг илэрхийлсэн жишээнүүдийг хэрэглэнэ. Энэ материал дахь код жишээнүүд нь DecimalFormatDemo нэртэй жишээ програмаас байгаа.

Хэв маягийг (pattern) байгуулах

DecimalFormat –н форматлах шинж чанарыг String хэв загвараар тодорхойлж заадаг. Форматлагдсан тоог яаж харагдуулахыг хэв загвар нь тодорхойлдог. Дараах жишээ нь String –ээс DecimalFormat –руу хэв маягын байгуулагч руу алгасан форматлагчийг

үүсгэж байна. Format арга нь Double утгыг аргумент мэтээр хүлээн авч форматлагдсан тоог String –д буцааж байна.

```
DecimalFormat myFormatter = new DecimalFormat(pattern);
String output = myFormatter.format(value);
System.out.println(value + " " + pattern + " " + output);
```

Кодын урьдах мөрөн дэх гарц нь дараах хүснэгтэнд тайлбарлагдана. Value гэдэг нь форматлагдах Double тоо юм. Pattern гэдэг нь форматлалтын шинж чанарыг тодорхойлсон String юм.

`DecimalFormatDemo` програмын гарц

Утга	Pattern	Гарц	Тайлбар
123456.789	###,###.###	123,456.789	Чагт тэмдэгт нь “#” орныг төлөөлнө, таслал нь бүлэг болгон хуваах тусгаарлагч тэмдэгт байх ба мөчлөг нь аравтын тусгаарлагчийн тэмдэгт байна.
123456.789	###.##	123456.79	Value нь аравтын цэгийн баруун тал руу 3 оронтой гэхдээ Pattern нь хоёр оронтой байна. Format арга нь дугуйлж цуглуулан үүнийг зохицуулна.
123.78	000000.000	000123.780	Pattern нь эхлэл болон төгсгөлийн тэгүүдийг тодорхойлно. Яагаад гэвэл “0” гэсэн тэмдэгт нь чагт “#” тэмдэгтийн оронд хэрэглэгдэнэ.
12345.67	$###,###.###	$12,345.67	Pattern –ны эхний тэмдэгт нь долларын тэмдэг “$” байна. Форматлагдсан Output дахь хамгийн зүүн талын оронг шууд болгодог.
12345.67	\u00A5###,###.###	¥12,345.67	Pattern нь Японы “¥”Иеныг 00A5 юникод утгаар мөнгөн тэмдэгт болгон тодорхойлогддог.

Форматлагч тэмдэгтүүдийг өөрчлөх

Format аргаар гаргасан форматлагдсан тоонуудад харагдаж буй тэмдэгтүүдийг өөрчлөхийн тулд DecimalFormatSymbols ангийг хэрэглэж болно. Энэ тэмдэгтүүд нь аравтын цэг, мянгатын тусгаарлагч, хасах тэмдэг, мөн хувийн тэмдэгийг бусаддаа агуулдаг.

Дараах жишээ нь тоо руу сонин формат хэрэглэснээрээ DecimalFormatSymbols ангийг илэрхийлж байна. Уг хэвийн бус формат нь SetDecimalSeparator, SetGroupingSeparator, болон SetGroupingSize аргуудыг дуудсаны үр дүн юм.

```
DecimalFormatSymbols unusualSymbols =
    new DecimalFormatSymbols(currentLocale);
unusualSymbols.setDecimalSeparator('|');
unusualSymbols.setGroupingSeparator('^');
```

```
String strange = "#,##0.###";
DecimalFormat weirdFormatter =
            new DecimalFormat(strange, unusualSymbols);
weirdFormatter.setGroupingSize(4);

String bizarre = weirdFormatter.format(12345.678);
System.out.println(bizarre);
```

Үүнийг гүйлгэхэд, уг жишээ нь тоог дараах хачин форматаар дэлгэцэнд хэвлэнэ:

```
1^2345|678
```

Үндсэн арифметикаас цааших

Java хэлэнд +, -, *, /, болон % мэтийн арифметик операторуудаар үндсэн тооцооллуудыг хийж болдог. Java платформ нь илүү нарийн математик тооцооллыг хийхэд зориулсан аргууд болон хувьсагчуудыг хангасан Math гэсэн ангийг java.lang багцад хангаж өгсөн.

Math анги дахь аргууд нь анги аргууд юм, тэгэхээр тэднийг ангиас нь шууд ингэж дуудаж болно:

```
Math.round(34.87);
```

Бидний үзэх гэж буй математикийн үндсэн функцуудын Math анги доторх аргуудыг доор хүснэгтээр тайлбарлан жагсаалаа.

Math анги доторх аргуудаар гүйцэтгэдэг математикийн үндсэн функцууд

Арга	Тайлбар

`double abs(double)`
`float abs(float)`
`int abs(int)`
`long abs(long)`
	Аргументын абсолют утгыг буцаана.

`double ceil(double)`	Математикийн бүхэл тоотой тэнцэх мөн аргументтай тэнцүү эсвэл их байх хамгийн бага double утгыг буцаана.
`double floor(double)`	Математикийн бүхэл тоотой тэнцэх мөн аргументтай тэнцүү эсвэл бага байх хамгийн их double утгыг буцаана.
`double rint(double)`	Математикийн бүхэл тоотой тэнцэх мөн аргументтай хамгийн ойр утгатай байх double утгыг буцаана.
`long round(double)` `int round(float)`	Аргын буцаадаг утгаар илэрхийлэгддэг шиг, хамгийн ойр long эсвэл int –г буцаана.

Доор BasicMathDemo програмд эдгээр аргуудын заримыг нь яаж хэрэглэхийг харууллаа.

```
public class BasicMathDemo {
    public static void main(String[] args) {
        double aNumber = -191.635;

        System.out.println("The absolute value of " + aNumber + " is " +
Math.abs(aNumber));
```

```
        System.out.println("The ceiling of " + aNumber + " is " +
Math.ceil(aNumber));
        System.out.println("The floor of " + aNumber + " is " +
Math.floor(aNumber));
        System.out.println("The rint of " + aNumber + " is " +
Math.rint(aNumber));
    }
}
```

Дээрх програмын гарц:

```
The absolute value of -191.635 is 191.635
The ceiling of -191.635 is -191
The floor of -191.635 is -192
The rint of -191.635 is -192
```

Math анги доторх үндсэн өөр хоёр арга бол min болон max юм. Доорх хүснэгтээр min болон max аргуудын ялгааг харуулж байна. Тэдгээр нь хоёр тоог харьцуулан ихийг нь эсвэл багыг нь буцаадаг.

Math анги дахь аргуудаар гүйцэтгэгддэг харьцуулах функцууд

Арга	Тайлбар

`double min(double, double)`
`float min(float, float)`
`int min(int, int)`
`long min(long, long)`
	Хоёр аргументын аль багыг нь буцаана.

`double max(double, double)`
`float max(float, float)`
`int max(int, int)`
`long max(long, long)`
	Хоёр аргументын аль ихийг нь буцаана.

Доорх MinDemo нь min аргыг ашиглан хоёр утгын аль нь бага болохыг тогтоож байна.

```
public class MinDemo {
    public static void main(String[] args) {

        double enrollmentPrice = 45.875;
        double closingPrice = 54.375;

        System.out.println("Your purchase price is: $"
                        + Math.min(enrollmentPrice, closingPrice));
    }
}
```

Дээрх програм нь бага утгыг нь зөвөөр гаргасан байна:

```
Your purchase price is: $45.875
```

Math ангид байдаг дараагийн аргууд бол зэргийн илтгэгч функцууд юм. Энд нэмж хэлэхэд, Math.E хэрэглэн натурал логарифмын үндсэн е тооны утгыг авч болдог.

Math анги дахь аргуудаар гүйцэтгэгддэг зэргийн илтгэгч функцууд

Арга	Тайлбар
`double exp(double)`	Аргументын зэрэгтэй е тооны натурал логарифм суурийг буцаана.
`double log(double)`	Аргументын натурал логарифмыг буцаана.
`double pow(double, double)`	Эхний аргументыг хоёр дахь аргументаар зэрэгт дэвшүүлэн утгыг нь буцаана.
`double sqrt(double)`	Аргументын язгуур дахь утгыг буцаана.

Дараах ExponentialDemo програм нь е тооны утгыг хэвлэн, өмнөх хүснэгт дэх аргуудыг дуудаж байна.

```
public class ExponentialDemo {
   public static void main(String[] args) {
      double x = 11.635;
      double y = 2.76;

      System.out.println("The value of e is " +
                         Math.E);
      System.out.println("exp(" + x + ") is " +
                         Math.exp(x));
      System.out.println("log(" + x + ") is " +
                         Math.log(x));
      System.out.println("pow(" + x + ", " + y + ") is " +
                         Math.pow(x, y));
      System.out.println("sqrt(" + x + ") is " +
                         Math.sqrt(x));
   }
}
```

ExponentialDemo –г гүйлгэхэд дараах гарцыг та харах болно.

```
The value of e is 2.71828
exp(11.635) is 112984
log(11.635) is 2.45402
pow(11.635, 2.76) is 874.008
sqrt(11.635) is 3.41101
```

Мөн Math анги нь дараах хүснэгтээр харуулсан тригнометрийн функцуудын цуглуулгыг хангаж өгсөн. Энэ аргуудаар дамжиж буй утга нь радианд илэрхийлэгддэг өнцөг юм. Градусыг радиан руу болон радианаас градус руу хөрвүүлдэг toDegrees болон toRadians аргуудыг хэрэглэж болно. Мөн тойргын радиусыг диаметрт нь харьцуулсан харьцаа болох pi тоотой бусдаасаа ойрхон байх double утгыг авахын тулд Math.PI хэрэглэж болно.

Math анги дахь аргуудаар гүйцэтгэгддэг зэргийн илтгэгч функцууд

Арга	Тайлбар
`double sin(double)`	Өгөгдсөн double утгын синусыг буцаана.

`double cos(double)`	Өгөгдсөн double утгын косинусыг буцаана.
`double tan(double)`	Өгөгдсөн double утгын тангенсийг буцаана.
`double asin(double)`	Өгөгдсөн double утгын арк синусыг буцаана.
`double acos(double)`	Өгөгдсөн double утгын арк косинусыг буцаана.
`double atan(double)`	Өгөгдсөн double утгын арк тангенсийг буцаана.
`double atan2(double)`	Тэгш өнцөгтийн координатуудыг (a, b) polar (r, theta) – руу хөрвүүлнэ.
`double toDegrees(double)*` `double toRadians(double)*`	Аргументыг градус руу эсвэл радиан руу хөрвүүлнэ.

* Java 2 SDK 1.2 –д зориулан Math ангид нэмэгдсэн.

Дараах TrignometricDemo програм нь дээрх аргуудыг ашиглахдаа 45 градусын өнцөгт тригнометрийн утгуудыг тооцоолж байна.

```
public class TrigonometricDemo {
    public static void main(String[] args) {
        double degrees = 45.0;
        double radians = Math.toRadians(degrees);

        System.out.println("The value of pi is " +
                           Math.PI);
        System.out.println("The sine of " + degrees +
                           " is " + Math.sin(radians));
        System.out.println("The cosine of " + degrees +
                           " is " + Math.cos(radians));
        System.out.println("The tangent of " + degrees +
                           " is " + Math.tan(radians));
        System.out.println("The arc sine of " +
                           Math.sin(radians) + " is " +
                      Math.toDegrees(Math.asin(Math.sin(radians))) +
                           " degrees");
        System.out.println("The arc cosine of " + Math.cos(radians) +
                           " is " +
                           Math.toDegrees(Math.acos(Math.cos(radians))) +
                           " degrees");
        System.out.println("The arc tangent of " +
                           Math.tan(radians) + " is " +
                           Math.toDegrees(Math.atan(Math.tan(radians))) +
                           " degrees");
    }
}
```

Энэ програмын гарц нь:

```
The value of pi is 3.141592653589793
The sine of 45.0 is 0.8060754911159176
The cosine of 45.0 is -0.5918127259718502
The tangent of 45.0 is -1.3620448762608377
```

```
The arc sine of 45.0 is NaN
The arc cosine of 45.0 is NaN
The arc tangent of 45.0 is 1.570408475869457
```

NaN бол үр дүн нь аргаар дамжуулж буй аргументад тодорхойлогдоогүй үед дэлгэцэнд гарна. NaN гэдэг нь Not a Number (тоо биш) –ийн товчилсон хэлбэр. Нарийвчилсан функцын үр дүн нь аргаар дамжуулж буй аргументад тодорхойлогдоогүй үед Math анги дахь элдэв аргууд дээрх NaN утгыг буцаадаг. Double болон Float ангиуд нь NaN нэртэй тогтмолуудыг агуулдаг. Аргын буцаах утгыг эдгээр тогтмолуудын нэгтэй нь харьцуулснаараа, таны програм NaN утгыг аргаас буцаагдсан эсэхийг нь тогтоож чадна. Тэгэхээр аргын математик үр дүн тодорхойлогдоогүй үед таны програм ямар нэгэн ухаалаг зүйл хийж чадна.

Math ангийн эцэст нь random арга байна. Энэ random арга нь 0.0 –оос 0.1 –н хооронд санамсаргүй согосон тоог буцаадаг. Гэхдээ хүрээ нь 0.0 –г багтаан 1.0 –г оруулдаггүй. Өөрөөр хэлбэл: 0.0 <= Math.random() < 1.0 байна. Random аргын буцаасан утга дээр арифметик үйлдэл гүйцэтгэн өөр хүрээн дэх тоог авч болдог. Жишээ нь, 1 –с 10 –н хооронд бүхэл тоог гаргахад ингэж бичнэ:

```
int number = (int)(Math.random() * 10 + 1);
```

10 –р үржүүлснээр боломжит утгын хүрээ нь 0.0 <= number < 10.0 болно. Дараа нь 1 –г нэмснээр боломжит утгын хүрээ нь 1.0 <= number < 11.0 болно. Эцэст нь тоог бүхэл тоо руу (int) хөрвүүлснээр 1 –с 10 –н хооронд бүхэл тоон утга болно.

Хэрэв та дан ганц тоо үүсгэх хэрэгтэй бол Math.random хэрэглэх нь сайн шүү. Хэрэв та санамсаргүй тоон цуваа үүсгэх хэрэгтэй бол java.util.Random –н төлийг үүсгэн, энэ обьект дахь аргуудыг дуудсан нь дээр шүү. Онлайн Бинго жишээн дэх RandomBag анги нь бинго бөмбөгнүүдийн санамсаргүй тоон цувааг үүсгэхдээ уг арга барилыг хэрэглэдэг. Math.random арга нь тоонуудаа үүсгэхдээ java.util.Random –н төлийг ашигладаг.

Дүгнэлт

Үндсэн өгөгдлийн төрлийн тоог агуулахын тулд Byte, Double, Float, Integer, Long, болон Short зэрэг Number ангиудын нэгнийх нь төлийг хэрэглэж болно. Мөн дурын тодорхой тоонуудад BigInteger болон BigDecimal –г хэрэглэж болно.

Number ангиуд нь арга замын олон зүйлд хэрэгтэй анги аргууд болон тогтмолуудыг агуулдаг. MIN_VALUE болон MAX_VALUE тогтмолууд нь тухайн төрлийн обьектоор агуулж болох хамгийн бага болон хамгийн их утгуудыг агуулдаг. ByteValue, ShortValue болон үүнтэй адил аргууд нь нэг тоон төрлөөс нөгөөд хөрвүүлдэг. valueOf арга нь хэлхээг тоо руу хөрвүүлдэг ба toString арга нь тоог хэлхээ рүү хөрвүүлдэг.

Хэрэглэгчийн дэлгэцэнд гарах тоог форматлахдаа, java.text багц дахь DecimalFormat ангийг хэрэглэнэ. DecimalFormat –г хэрэглэх үед аравтын тоонууд, зууны хувь, эсвэл мөнгөн утгад оршмол (default) форматыг авч болно. Эсвэл хэв маягуудыг (pattern) ашигласнаар форматыг өөрөө тохируулж (custom) болно. Хэрэв JDK 5.0 эсвэл дээш хувилбарыг ашиглаж байвал, printf боломж нь одоо хүртэл таны гарцыг форматлахад өөр утга санаа олгож өгсөн.

Math математик функцуудыг гүйцэтгэхэд зориулсан анги аргуудын олон зүйлийг агуулсан. Мөн синус, косинус тооцоолох зэрэг тригнометр функцуудыг багтаасан. Math нь логарифм тооцоололд зориулсан үндсэн арифметик функцуудыг мөн багтаасан. Эцэст нь Math нь random зэрэг аргыг санамсаргүй тоонуудыг гаргахад зориулан агуулсан байдаг.

Асуулт болон дасгал

Асуулт

1. Дараах асуултуудад хариулахын тулд API баримтжуулалт ашиглаарай.
   a. 16тын тооллын системд байгаа тоог илтгэсэн Int –г хэлхээрүү хөрвүүлэхэд ямар Integer арга хэрэглэж болох вэ? Жишээ нь: 65 бүхэл тоог “41” гэсэн хэлхээ рүү ямар арга хөрвүүлдэг вэ?
   b. 5тын суурь дээр илтгэсэн хэлхээг Int –лүү тэнцүүгээр хөрвүүлэхэд ямар 2 integer аргуудыг хэрэглэх вэ? Жишээ нь: “230” гэсэн хэлхээг 65 гэсэн бүхэл тоон утгаруу яаж хөрвүүлэх вэ? Энэ даалгаврыг гүйцэтгэхийн тулд хэрэглэх кодоо хараарай.
   c. Хөвөх таслалтай тоо нь тоо биш (NaN) онцгой утгатай эсэхийг илрүүлэхийн тулд ямар double аргыг хэрэглэж болох вэ?
2. Дараах илэрхийллийн утга ямар байх вэ? Мөн яагаад?
   ```
   new Integer(1).equals(new Long(1))
   ```

Дасгал

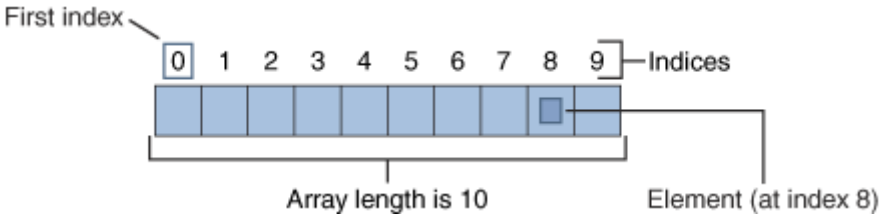
1. Хамгийн их утгын оронд хамгийн бага утгыг харуулахын тулд MaxVariablesDemo –г өөрчил. aChar болон aBoolean хувьсагчуудтай холбоотой бүх кодуудыг арилгаж болно. Гарц ямар байх вэ?
2. Командын мөрнөөс бүхэл тоон аргументуудын тодорхойлогдоогүй тоог уншдаг мөн тэднийг хамтад нь нэмдэг програм бич. Жишээ нь дараахыг оруулна гэж бодоорой:
   ```
   java Adder 1 3 2 10
   ```
   Програмаа 16 гэж хэвлээд дараа нь дуусдаг байхаар болго. Хэрэв хэрэглэгч зөвхөн ганц аргумент оруулбал алдааны мэдээ өгдөг байлга. ValueOfDemo програм дээр үндэслэж болно шүү.
3. Өмнөхтөй адил програм хий, гэхдээ дараах ялгаануудтай байна:
   - Бүхэл тоон аргументуудын оронд хөвдөг таслалтай аргументууд унших
   - Аравтын цэгийн баруун талруу яг 2 оронг ашиглан аргументуудын нийлбэрийг дэлгэцэнд гаргах

   Жишээ нь: дараахыг оруулна гэж бодоорой:
   ```
   java FPAdder 1 1e2 3.0 4.754
   ```

Програм нь 108.75 гэж дэлгэцэнд хэвлэнэ. Орон нутгийн байрлалаас хамааран аравтын цэгийн (.) оронд таслал (,) байж магадгүй шүү.

Бүл

Бүл бол нэг төрлийн олон утга агуулдаг бүтэц юм. Бүл үүсгэх үед үүлийн урт бүрэлддэг. Үүссэнийхээ дараа бүл нь тогтмол урттай бүтэц болдог.



Бүлийн гишүүн гэдэг нь бүл доторхи байрлалаараа хандагддаг бүлийн нэг гишүүн юм.

Төрөл бүрийн өгөгдлийг нэг бүтэц дотор хадгалах эсвэл бүтцийн хэмжээг нь динамикаар өөрчлөгдөх шаардлагатай болсон үед бүлийн оронд ArrayList гэх мэт Collection объектуудыг ашиглах хэрэгтэй. Энэ хэсэгт доорхи сэдвүүдийг авч үзнэ.

1. Бүлийг үүсгэх болон ашиглах
2. Объектын бүл
3. Бүлийн бүл
4. Бүл хуулбарлах
5. Дүгнэлт
6. Асуулт, дасгалууд

Бүлийг үүсгэх болон ашиглах

Бүл үүсгээд дотор нь утгуудыг нь оруулж дэлгэцэнд харуулах ArrayDemo програмыг үзүүллээ.

```
public class ArrayDemo {
    public static void main(String[] args) {
        int[] anArray;          // declare an array of integers

        anArray = new int[10];  // create an array of integers

        // assign a value to each array element and print
        for (int i = 0; i < anArray.length; i++) {
            anArray[i] = i;
            System.out.print(anArray[i] + " ");
        }
        System.out.println();
    }
}
```

Энэ програмын гарц нь:

```
0 1 2 3 4 5 6 7 8 9
```

Энд дараах сэдвүүдийг үзнэ.

- Бүл заасан хувьсагч зарлах
- Бүл үүсгэх
- Бүл цэнэглэх
- Бүлийн гишүүдэд хандах
- Бүлийн хэмжээг авах

Бүлтэй харьцах хувьсагчийг зарлах (Бүлд хандах)

Дараах энгийн програмын кодонд бүл хувьсагч зарлаж байна.

```
int[] anArray;          // declare an array of integers
```

Өөр төрлийн хувьсагч зарладагтай адил бүлийн зарлалт нь 2 хэсгээс бүрддэг.

1. Бүлийн төрөл
2. Бүлийн нэр

Бүлийн төрөл нь type[ ] –гэж бичигдсэн ба бүлийн гишүүдийн төрлийг заана. Мөн [ ] хаалт нь бүл гэдгийг зааж байна. Бүлийн бүх гишүүд нь int[ ] гэсэн ба энэ нь anArray нэртэй бүл нь бүхэл тоон өгөгдөл авна гэсэн үг юм.

Доор өөр өөр төрлийн өгөгдөлтэй бүлийн зарлалтыг харуулав.

```
float[] anArrayOfFloats;
boolean[] anArrayOfBooleans;
Object[] anArrayOfObjects;
String[] anArrayOfStrings;
```

Мөн дараах хэлбэрээр бүлийг бас зарлаж болно.

```
float anArrayOfFloats[]; //this form is discouraged
```

Гэвч энэ нь буруу хэлбэр юм, учир нь [ ]-хаалт нь бүлийн төрлийг заах ба мөн төрөлтэй хамт бичигдэхээс биш нэртэй хамт бичигддэггүй.

Бусад төрлийн хувьсагчдын зарлалттай адил бүл хувьсагчийн зарлалт нь ямар нэг бүлийг үүсгэхгүй ба бүлийн гишүүдэд ямар нэг санах ойг хуваарилдаггүй. Код нь бүлийг илээр үүсгэхэд anArray хувьсагчид олгох ёстой.

Бүлийг үүсгэх

Бүлийг Java-ийн new оператор ашиглан үүсгэдэг. Уг энгийн програмын дараагийн хэллэгт бүхэл тоон төрөлтэй 10 гишүүнтэй бүлийг санах ойд байрлуулж бүлийг дээр зарлагдсан anArray хувьсагчид олгож байна.

```
anArray = new int[10];  // create an array of integers
```

Ерөнхийдөө бүлийг new оператор, бүлийн гишүүдийн төрөл, [ ] –хаалтанд гишүүдийн тоо зэргийг ашиглан үүсгэнэ.

```
new elementType[arraySize]
```

Хэрэв new хэллэгийг уг энгийн програмаас хасвал эмхэтгүүр нь доорх алдааг өгч эмхтгэх ажил зогсоно.

```
ArrayDemo.java:4: Variable anArray may not have been initialized.
```

Бүлийг цэнэглэх

Бүлийг цэнэглэхдээ, үүсгэхдээ доорх хураангуй өгүүлбэр зүйг ашиглаж болно. Жишээ нь :

```
boolean[] answers = { true, false, true, true, false };
```

{ } – хаалтанд байгаа утгуудын тоо нь бүлийн уртыг заана.

Бүлийн гишүүнд хандах нь

Бүлийг санах ойд байрлуулсны дараа програм бүлийн гишүүдэд утга олгож байна.

```
for (int i = 0; i < anArray.length; i++) {
    anArray[i] = i;
    System.out.print(anArray[i] + " ");

}
```

Дээрхи хэсэг код нь бүлийн гишүүдэд хандаж утга олгох, утгыг нь авахдаа бүлийнн нэрэнд нь [ ] хаалт нэмж өгсөн байна. Уг хаалтанд доторхи утга нь хандах гишүүний индексыг заана.

Бүлийн хэмжээг авах

Бүлийн хэмжээг авахдаа дараах хэлбэрээр авч болно.

```
arrayname.length
```

Болгоомжлол: Java хэлийг эхлэн суралцаж буй хүмүүс length –ыг арга шиг хоосон хаалттай бичдэг. Энэ нь буруу юм, учир нь length нь арга биш юм. Харин бүх бүлүүд нь java платформд length шинж чанараар хангагддаг.

For давталт нь anArray хувьсагчийн гишүүн бүрт утга олготол давтах болно. For давталт нь anArray.lenght-г уг давталтаа дуусгахад ашилаж байна.

Объектуудын бүлүүд

Бүл нь энгийн төрлүүд шиг заагч төрөлтэй байж болно. Ихэнх бүлийг үүсгэхдээ энгийн төрөлтэй үүсгэдэг. Дараах ArrayOfStringsDemo програмд 3 хэлхээ (string) объектыг агуулсан бүлийг үүгэж уг 3 хэлхээг жижиг үсгээр хэвлэж байна.

```
public class ArrayOfStringsDemo {
    public static void main(String[] args) {
        String[] anArray = { "String One",
                             "String Two",
                             "String Three" };

        for (int i = 0; i < anArray.length; i++) {
            System.out.println(anArray[i].toLowerCase());
        }
    }
}
```

Уг програмын гарц нь :

```
string one
string two
string three
```

Санамж: Хэрэв та JDK 5.0 болон түүнээс дээшхи хувилбарыг хэрэглэж буй бол бүлээр дараах давталт нь боломжтой юм.

```
String[] anArray = {"String One","String Two","String Three"};
     for (String s : anArray) {
         System.out.println(s.toLowerCase());
      }
```

Уг програм нь бүлийн цэнэглэгч ашиглан нэг л хэллэгээр бүлийг үүсгэж, байрлуулж байна.

Дараагийн програмд ArrayOfIntegerDemo нь бүлийг integer объектуудтайгаар байгуулж байна. Уг програм нь нэг бүхэл тоон төрөлтэй объект үүсгээд for давталтын хугацаанд үүнийг бүлд байрлуулж байна гэдгийг анхаарах хэрэгтэй.

```
public class ArrayOfIntegersDemo {
    public static void main(String[] args) {
        Integer[] anArray = new Integer[10];

        for (int i = 0; i < anArray.length; i++) {
            anArray[i] = new Integer(i);
            System.out.println(anArray[i]);
        }
    }
}
```

Уг програмын гарц нь :

```
0
1
2
3
4
```

Дараах ганц мөр код нь бүлийг ямар нэг гишүүнгүйгээр үүсгэж байна.

```
Integer[] anArray = new Integer[5];
```

Энэ нь бидэнд алдаа авчрах магадлалтай юм. Объектууд агуулсан бүлийг ашиглахад шинэ програмистуудад дээрхи алдаа их гардаг. Дээрхи код ажиллахад anArray нэртэй бүл оршин байгаа эсэх болон 5 бүхэл тоон төрөлтэй объект байршуулах хангалттай зай байгааг үзнэ. Гэвч уг бүл нь ямар нэг объектыг агуулаагүй хоосон байна. Програм нь объектуудаа үүсгээд (илээр) түүнийг бүлдээ байршуулж өгөх ёстой. Энэ нь илэрхий мэт санагдаж болох юм. Гэвч ихэнх эхлэн суралцагчид дээрхи кодыг 5 хоосон объекттой бүл үүсгэж байна гэж тааварладаг. Тиймээс доорхи кодонд харуулсан шиг NullPointerException – д хүргэдэг.

```
Integer[] anArray = new Integer[5];
for (int i = 0; i < anArray.length; i++) {
        //ERROR: the following line gives a runtime error
        System.out.println(anArray[i]);
}
```

Бүл нь байгуулагч дотор үүсэх эсвэл өөр цэнэглэгчээр үүсгээд програмын аль нэг хэсэгт ашиглах үед дээрхи алдаа ихэнхдээ тохиолддог.

Бүлүүдийн бүлүүд

Бүл нь бүлийг агуулж болно. ArrayOfArraysDemo програм нь бүл үүсгээд цэнэглэгч ашиглан уг бүлээ 4 дэд бүлүүдтэйгээр байршуулж байна.

```
public class ArrayOfArraysDemo {
    public static void main(String[] args) {
        String[][] cartoons =
        {
             { "Flintstones", "Fred", "Wilma",
              "Pebbles", "Dino" },
            { "Rubbles", "Barney", "Betty",
               "Bam Bam" },
            { "Jetsons", "George", "Jane",
              "Elroy", "Judy", "Rosie", "Astro" },
            { "Scooby Doo Gang", "Scooby Doo",
              "Shaggy", "Velma", "Fred", "Daphne" }
        };

        for (int i = 0; i < cartoons.length; i++) {
             System.out.print(cartoons[i][0] + ": ");
            for (int j = 1; j < cartoons[i].length; j++) {
                 System.out.print(cartoons[i][j] + " ");
              }
```

```
            System.out.println();
        }
    }
}
```

Уг програмын гарц нь:

```
Flintstones: Fred Wilma Pebbles Dino
Rubbles: Barney Betty Bam Bam
Jetsons: George Jane Elroy Judy Rosie Astro
Scooby Doo Gang: Scooby Doo Shaggy Velma Fred Daphne
```

Дээрхи бүх дэд бүлүүд нь бүгд өөр өөр урттай гэдгийг анхаарах хэрэгтэй. Дэд ангиудын нэрнүүд нь `cartoons[0]`, `cartoons[1]` гэх мэт байна.

Объектуудын бүлүүдтэй адил бүлд дэд бүлүүдийг илээр үүсгэх ёстой. Тиймээс хэрэв цэнэглэгч ашиглаагүй бол ArrayOfArraysDemo2 програмын код шиг бичих хэрэгтэй.

```
public class ArrayOfArraysDemo2 {
    public static void main(String[] args) {
        int[][] aMatrix = new int[4][];

         //populate matrix
        for (int i = 0; i < aMatrix.length; i++) {
             aMatrix[i] = new int[5];   //create sub-array
             for (int j = 0; j < aMatrix[i].length; j++) {
                 aMatrix[i][j] = i + j;
             }
        }

        //print matrix
        for (int i = 0; i < aMatrix.length; i++) {
             for (int j = 0; j < aMatrix[i].length; j++) {
                 System.out.print(aMatrix[i][j] + " ");
             }
             System.out.println();
        }
    }
}
```

Уг програмын гарц нь:

```
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
```

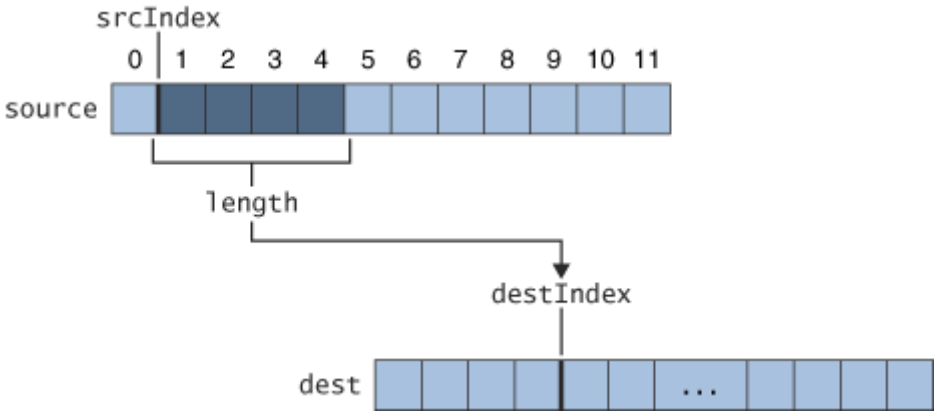
Бүл үүсгэхдээ үүнийхээ (бүлийнхээ) уртыг тотодорхойлж өгөх ёстой. Дэд бүлүүд агуулсан бүл үүсгэхдээ үндсэн бүлийнхээ уртыг тодорхойлох хэрэгтэй. Харин аль нэг дэд бүлийн уртыг тэднийг үүсэх хүртэл зааж өгөх (тодорхойлж өгөх) шаардлагагүй юм.

Бүлийг хуулбарлах

System ангийн arraycopy аргыг хэрэглэж нэг бүлээс нөгөө бүл рүү өгөгдөл хуулах нь зохимжтой юм. Уг arraycopy арга нь 4 аргументтай байдаг.

```
public static
    void arraycopy(Object source,
                   int srcIndex,
                   Object dest,
                   int destIndex,
                   int length,
```

Энд 2 объект аргумент нь хуулах болон хуулбарлаж авах объектыг заана. Гурван бүхэл тоон төрөрлтэй аргументууд нь хуулбарлах гишүүдийн тоо, болон хуулах бүлийн байрлал, эх бүлийн эхлэлийн байрлалыг заана. Дараах зураг нь хуулбарлалт нь яаж байрлалаа авч байгааг харуулж байна.



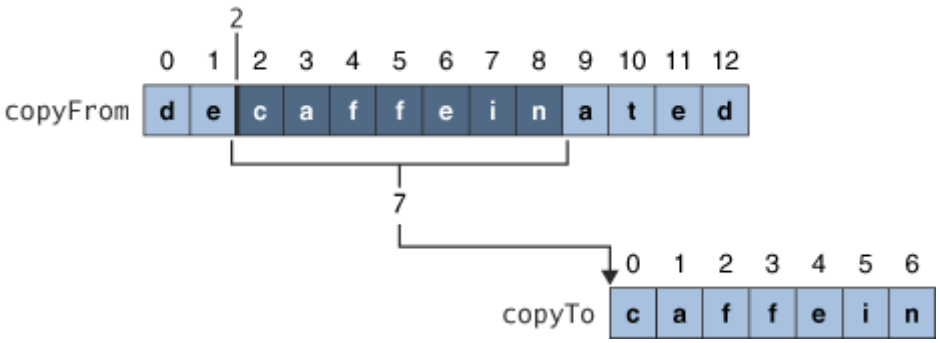
ArrayCopyDemo програм arraycopy аргыг ашиглан copyFrom бүлээс зарим гишүүдийг copyTo бүл рүү хуулбарлаж байна.

```
public class ArrayCopyDemo {
    public static void main(String[] args) {
        char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
                            'i', 'n', 'a', 't', 'e', 'd' };
        char[] copyTo = new char[7];

        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        System.out.println(new String(copyTo));
    }
}
```

Дээрхи програмын гарц нь:

```
caffein
```

Дээрхи жишээ програмд arraycopy аргыг дуудаж эх бүлийн 2 дахь гишүүнээс хуулбарлалт эхлэж байна. 0 –с эхлэн дахин дуудаад ‘c’ гэсэн гишүүнээс хуулбарлалт эхлэж байна. Arraycopy арга нь хуулбарлах copyTo бүлийн эхний гүшүүнээс (0 –дахь гүишүүнээс) эхлэн хуулбарлагдсан гишүүдийг тавиж байна. ‘c’, ‘a’, ‘f’, ‘f’, ‘e’, ‘i’, болон ‘n’ гэсэн 7 гишүүнийг хуулбарлана. Arraycopy арга нь “decaffeinated” дотроос “caffein” –г авч байна.



Arraycopy аргыг дуудахаас өмнө хуулбарлах бүлийг байрлуулсан байх ёстой бөгөөд хуулбарлагдах өгөгдлийг багтаахаар хангалттай хэмжээтэй байх ёстой.

Дүгнэлт

Бүл гэдэг нь ижил төрлийн олон обьектуудыг агуулж болдог тогтмол урттай өгөгдлийн бүтэц юм. Бүл нь обьектын ямар ч төрлийг агуулж чаддаг, бүл ч байж болно. Бүлийг зарлахын тулд, бүлийг агуулж болох обьектын төрлийг хэрэглээрэй.

Бүлийг үүсгэхдээ уртын хэмжээг нь зааж өгөх ёстой. Бүлийг үүсгэхэд new операторыг хэрэглэж болно эсвэл бүл цэнэглэгчийг хэрэглэж болно. Нэгэнт үүсгэсний дараа бүлийн хэмжээг өөрчилж болдоггүй. length атрибут ашиглан бүлийн уртын хэмжээг тавьж болдог.

Бүл доторх гишүүнд түүний индексээр нь ханддаг. Индекс нь 0 –р эхлэдэг бөгөөд бүлийн уртаас нэгийг хассан тоогоор дуусдаг.

Бүлийг хуулбарлахдаа System анги доторх arraycopy аргыг хэрэглэнэ.

Асуулт болон дасгал

Асуулт

1. Дараах бүл доторх Brighton –ны индекс юу вэ?

```
String[] skiResorts = {
    "Whistler Blackcomb", "Squaw Valley", "Brighton",
    "Snowmass", "Sun Valley", "Taos"
};
```

2. Бүл доторх Brighton хэлхээ рүү хандуулсан илэрхийлэл бич.
3. skiResorts.lenght илэрхийллийн утга юу вэ?
4. Бүл дэх сүүлчийн гишүүний индекс юу вэ?
5. skiResorts[4] илэрхийллийн утга юу вэ?

Дасгал

1. Дараах WhatHappens програм бөөс (bug) агуулсан байгаа. Түүнийг олж засварла.

```
//
// This program compiles but won't run successfully.
//
public class WhatHappens {
    public static void main(String[] args) {
        StringBuffer[] stringBuffers = new StringBuffer[10];

        for (int i = 0; i < stringBuffers.length; i ++) {
            stringBuffers[i].append("StringBuffer at index " + i);
        }
    }
}
```

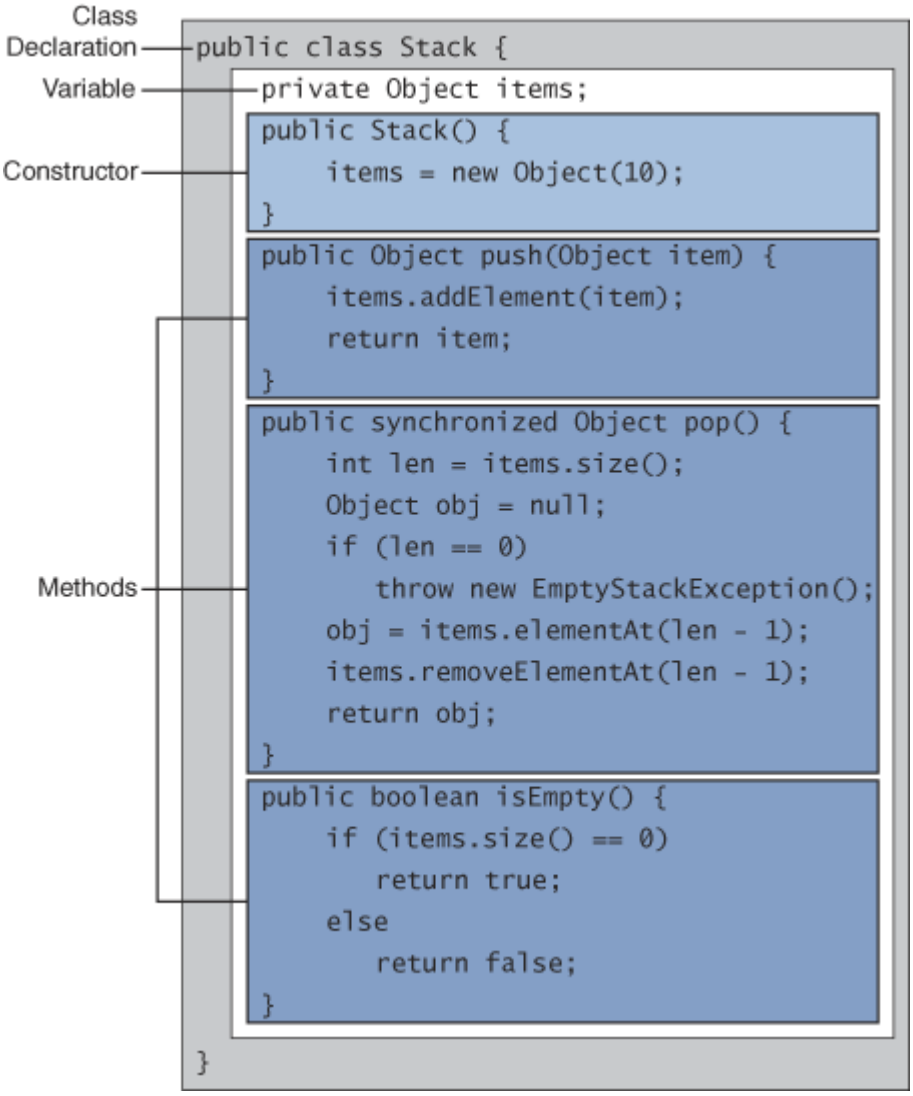
Ангиуд болон удамшил

Та өөрийн ангиудаа бичихийн тулд Java хэлний үндсүүдийн талаар болон обьектыг үүсгэх ашиглах талаар урд сурсан мэдлэг дээрээ тулгуурлан сурч болно. Энэ бүлэгт та гишүүн хувьсагчийг зарлах, арга бичих, өвөг ангиас хувьсагч болон арга удамшуулах, ангийг бусад анги дотор үүрлүүлэх зэргийг багтаан өөрийн ангиа хэрхэн тодорхойлох талаар мэдээллийг олж авах болно.

- Ангиудыг үүсгэх
- Удамшилыг зохицуулах
- Үүрлэсэн ангиуд
- Тоочих төрлүүд
- Женерик

Ангиудыг үүсгэх

Энэ хэсэгт ангийн үндсэн компонентуудыг харуулахдаа стекийг гүйцэтгэсэн жижиг жишээ ашиглана. Стек нь сүүлийн-орц-эхний-гарц (FILO) зарчмаар өгөгдлөө оруулдаг болон гаргадаг өгөгдлийн бүтэц юм. Доорх зураг нь stack ангийг харуулан уг ангийн гишүүдийг тогтоосон байна.



Ангийн тодорхойлолт нь ангийн мэдэгдэл болон ангийн эх бие гэсэн хоёр үндсэн компоненттой. Ангийн мэдэгдэл нь анги дахь кодын эхний мөр юм. Ангийн мэдэгдэл нь ядаж ангийнхаа нэрийг зарладаг.

Ангийн эх бие нь мэдэгдлийн дараах { } хаалтан дотор оршино. Ангийн эх бие нь ангиас үүссэн обьектуудын амьдралын мөчлөгийг хангах бүх кодуудыг багтаадаг. Тэдгээр нь шинэ обьектуудыг цэнэглэх байгуулагчууд, ангийн болон түүний обьектуудын нөхцлийг хангадаг хувьсагчуудын мэдэгдэлүүд, мөн ангийн болон түүний обьектуудын араншинг гүйцэтгэдэг аргууд зэрэг байна. Stack анги нь items жагсаалт болон стекийн оройн индекс болох top гэсэн хоёр гишүүн хувьсагчийг ангийн эх бие дотор тодорхойлдог. Stack анги нь мөн нэг аргумент болон pop, push, isEmpty гэсэн гурван аргыг авдаг ганц байгуулагчийг тодорхойлдог.

Анги зарлах

Төрөл бүрийн ангийн тодорхойлолтууд дараах нийтлэг хэлбэртэй байна.

```
class MyClass {
    //member variable and method declarations
}
```

Энэ кодын эхний мөрийг ангийн мэдэгдлийн мөр гэнэ. Дээрх ангийн мэдэгдэл нь хамгийн бага ангийн мэдэгдэл юм. Энэ нь ангийн мэдэгдэлд зайлшгүй шаардлагатай мэдээллүүдийг агуулж байна. Түүнээс гадна энэ ангид шууд тодорхойлогдоогүй олон шинж чанарууд байна. Тухайлбал MyClass ангийн шууд дээд өвөг нь Object анги юм. Ангийн мэдэгдэл дотор тухайн ангийн дээд анги хэрэгжүүлж байгаа үүд (interface) болон тухайн анги дэд анги үүсгэх боломжтой эсэх гэх мэт тухайн ангитай холбоотой дэлгэрэнгүй мэдээллийг тодорхойлж болно. Ангийн мэдэгдлийн бүх гишүүдийг бичигдэх дарааллаар доорх хүснэгтэнд харууллаа.

Ангийн мэдэгдлийн гишүүд

Гишүүн	Зориулалт
`public`	(Болзолт) Нээлттэй анги
`abstract`	(Болзолт) Төллөх боломжгүй анги
`final`	(Болзолт) Дэд анги үүсэх боломжгүй анги
`class` *`NameOfClass`*	Тухайн ангийн нэр
`extends` *`Super`*	(Болзолт) Ангийн өвөг анги
`implements` *`Interfaces`*	(Болзолт) Тухайн ангийн хэрэгжүүлж буй интерфейс

{
ClassBody
}
	Тухайн ангийн үйл ажилагаа

Хүснэгтийн баруун гар талд гишүүн бүрийн зориулалтуудыг тайлбарласан байна. Зайлшгүй гишүүдийн талаар харуулав. Бусад гишүүд нь болзолт бөгөөд тэдгээрийг тус тусад нь нэг мөрөнд бичсэн байна. (extends super) бол нэг гишүүн, гэхдээ програмдаа ингэж бичих шаардлагагүй анги болон үүдийн нэрийг бичихээр оруулав. Болзолт

гишүүдийг илээр зарлаагүй нөхцөлд ямар ч үүд хэрэгжээгүй. Object анги нээлттэй бус, хийсвэр бус, эцсийн бус, дэд анги автоматаар үүсдэг.

Ангийн мэдэгдлийн талаар дэлгэрэнгүй тайлбарыг доор үзүүллээ. Тухайн гишүүн ямар утгатай болох, яаж ашиглах, таны анги бусад ангиуд болон програмд хэрхэн нөлөөлөх, илүү дэлгэрэнгүй мэдээллийг энэ бүлгийн аль хэсгээс авч болох талаар бас тусгалаа.

- `public` – тухайн ангийг өөр дурын анги ашиглах боломжтой гэдгийг энэ хувиргуур зааж байна. Public хувиргуургүй ангийг зөвхөн тухайн багц доторх өөр ангиуд ашиглах боломжтой. Өөр дэлгэрэнгүй мэдээллийг Creating and Using Packages хэсгээс үзээрэй.

- `abstract` – төллөх боломжгүй ангийг зарлахдаа abstract хувиргуур ашиглана. Хийсвэр ангиудийг хэзээ яаж ашиглахад тохиромжтой болох талаар дэлгэрэнгүй мэдээллийг Writing Abstract Classes and Methods хэсгээс үзээрэй.

- `final` - дэд анги үүсгэх боломжгүй ангийг зарлахдаа final хувиргуур ашигладаг. Ийм анги бичих ямар шаардлага гардаг талаар Writing Final Classes and Methods хэсгээс үзээрэй.

- `class` *NameOfClass* - ангийн мэдэгдэл эхэлснийг class түлхүүр үг заадаг. Class түлхүүр бичигдсэний дараа ангийн нэр бичигдэнэ.

- `extends` *Super* - тухайн ангийн дээд анги нь өвөг болохыг заахдаа extends түлхүүр үгийг ашиглана. Дэд ангийн үүрэг зориулалт ач холбогдлын талаар Managing Inheritance хэсгээс үзнэ үү.

- `implements` *Interfaces* – үүд болон түүнээс дээш үүдийг хэрэгжүүлж байгааг зарлахдаа implements түлхүүр үгийг бичээд тухайн ангийг хэрэгжүүлж байгаа бүх үүдийн нэрийг таслалаар тусгаарлан бичнэ. Үүдийг яаж бичиж хэрхэн ашиглах талаар Creating and Using Interfaces хэсгээс үзээрэй.

Гишүүн хувьсагчийг зарлах

Стек доторх гишүүн хувьсагчийг тодорхойлохдоо доорх кодыг ашигласан болно.

```
private Object[] items;
private int top;
```

Уг мэдээлэл нь ямар нэг арга юмуу байгуулагч дотор биш харин ангийн бие дотор бичигдсэн байгаа учраас энэ код нь дотоод хувьсагч гэх мэт өөр төрлийн хувьсагчийг биш харин гишүүн хувьсагчийг зарлаж байна.

Зарласан гишүүн хувьсагчийнх нь нэр нь Items болон top болно. Тэдгээрийн харгалзах өгөгдлийн төрөл нь обьектуудын бүл болон int болно. Түүнчлэн Items болон top нь хаалттай гишүүд гэдгийг Private түлхүүр нь тодотгож байна.

Өөрөөр хэлбэл, зөвхөн stack анги тэдэнтэй харьцах чадвартай. Items болон top хувьсагчдын мэдэгдэл нь гишүүн хувьсагчдын мэдэгдлийн хялбар жишээ бөгөөд үүнээс нарийн төвөгтэй мэдэгдэл байж болно.

Өгөгдлийн төрөл, нэр, хандалтын төвшин төдийгүй тухайн хувьсагч анги хувьсагч эсэх, тогтмол эсэх гэх мэт нэмэлт шинж чанаруудыг бас тодорхойлж болно. Гишүүн хувьсагчдын мэдэгдлийн боломжит бүх гишүүдийг хүснэгтээр харуулав.

Хувьсагчын мэдэгдлийн гишүүд

Гишүүн	Зориулалт
accessLevel	(Болзолт) Хувьсагчийн хандалтын төвшин
`static`	(Болзолт) Анги хувьсагч
`final`	(Болзолт) Хувьсагч нь тогтмол
`transient`	(Болзолт) Хувьсагч нь `transient`
`volatile`	(Болзолт) Хувьсагч нь `volatile`
type name	Хувьсагчийн төрөл болон нэр

accessLevel – бусад ангиуд гишүүн хувьсагч руу хандах public, protected, package, мөн private хандалтын дөрвөн янзын төвшнөөр хандаж болно. Аргуудтай хандахад мөн энэ төвшнүүдийг ашиглана. Хандалтын төвшний талаар дэлгэрэнгүй мэдэхийг хүсвэл Controlling Access to Members of a Class хэсгээс үзээрэй.

`static` - төл хувьсагчийг биш харин анги хувьсагчийг зарлаж байна. Анги арга зарлахад бас static түлхүүр үгийг хэрэглэж болно. Төл болон анги аргын талаар дэлгэрэнгүйг Understanding Instance and Class Members хэсгээс үзнэ үү.

final - тухайн гишүүний утга өөрчлөгдөхгүй болохыг харуулна. Жишээ нь: утга нь тойргын уртыг диатметрт харьцуулсан харьцаа 3.141592653589793 тоо байх PI нэртэй тогтмол тодорхойлсон хувьсагчийн мэдэгдлийг харуулна.

Final double PI = 3.141592653589793;

Хэрэв таны программ энэ хувьсагчийн утгыг өөрчлөхийг оролдвол эмхэтгэлийн үеийн алдаа гарна. Тогтмолын нэрийг томоор бичих журам байдаг

`transient` – цувруулах боломжгүй гишүүн хувьсагчийг тэмдэглэдэг. Object Serialization хэсгээс дэлгэрэнгүйг уншина уу .

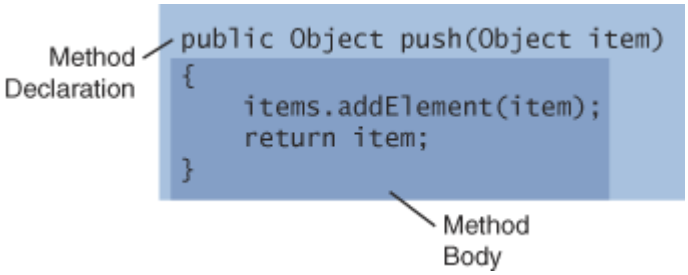
`volatile` – тухайн гишүүнд дээр optionation хийхийг хориглоно. Цөөн хүмүүс ашигладаг энэ онцлогийг бид энэ хэсэгт үзэхгүй.

Type – бусад хувьсагчийн адил гишүүн хувьсагч заавал төрөл агуулах ёстой.
Int, float, Boolean гэх мэт өгөгдлийн энгийн төрлийг ашиглаж болохоос гадна Object interface –н нэр гэх мэт заагч төрөл ашиглаж болно.

name – гишүүн хувьсагчын нэр нь дурын идентификатор байх бөгөөд жижиг үсгээр эхлэх журам байдаг. Нэг анги дотор хоёр адилхан анги байж болохгүй .

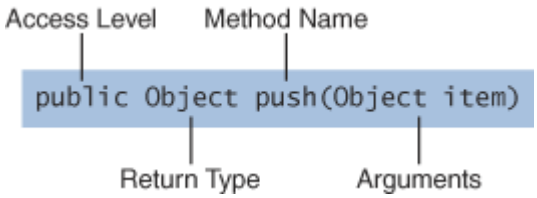
Арга тодорхойлох

Stack обьектын push доор зургаар харууллаа. Энэ арга нь обьект аргументыг stack –н орой дээр байрлуулаад уг обьектыг буцаана.



Аргын тодорхойлолт нь аргын мэдэгдэл аргын бие гэсэн 2 хэсгээс бүрдэнэ.

Доорх зурагт үзүүлсэн push аргын хандалтын төвшин, буцах төрөл, нэр , аргумент зэрэг тухайн аргын бүх шинж чанарыг аргументын шинж чанар дотор тодорхойлно. Аргын бие дотор тухайн арга доторх бүх үйл ажиллагааг тодорхойлно. Тухайн аргын хэрэгжүүлж байгаа бүх зааврууд энд байрладаг.



Аргын мэдэгдлийн зайлшгүй гишүүд нь тухайн Аргументын нэр буцах утга () хаалт байдаг. Тухайн аргын буцах төрөл, тухайн аргад шаардлагатай бүх аргументуудын тоо болон төрөл өөр ямар анги болон обьект энэ Аргыг дуудах боломжтой зэргийг аргын мэдэгдэл дотор бичиж болно. Аргын мэдэгдлийн боломжит бүх гишүүдийг дараах хүснэгтээр харууллаа.

Аргын мэдэгдлийн гишүүд

Гишүүн	Зориулалт
accessLevel	(Болзолт) Хандалтын төвшин
`static`	(Болзолт) Анги арга
`abstract`	(Болзолт) Хэрэгжилтгүй хоосон арга
`final`	(Болзолт) Дахин тодорхойлогдохгүй
`native`	(Болзолт) Өөр хэлээр бичигдсэн арга
`synchronized`	(Болзолт) Монитор зохицуулах шаардлагатай арга
returnType methodName	Буцах утгын төрөл болон нэр
`(` *paramList* `)`	Тухайн аргын Аргументын жагсаалт
`throws exceptions`	(Болзолт) Гажуудлын жагсаалт

accessLevel – дөрвөн янзын хандалтын төвшин байдаг бусад аргатай харьцахдаа public, protected, package, мөн private гэсэн дөрвөн төрлийн хандалтыг ашиглана. Дэлгэрэнгүй мэдээллийг Controlling Access to Members of a Class хэсгээс үзээрэй.

`static` – энэ арга нь төл арга биш анги арга гэдгийг static түлхүүр үг заана. Төл болон анги аргыг зарлах талаар Understanding Instance and Class Members хэсгээс үзээрэй.

`abstract` – хийсвэр арга нь хэрэгжилтгүй арга бөгөөд хийсвэр ангийн гишүүн байх ёстой. Хийсвэр арга бичих шалтгаан болон энэ арга нь дэд ангид яаж нөлөөлдөг талаар Writing Abstract Classes and Methods хэсгээс үзээрэй.

`final` – дэд анги дотор дахин тодорхойлох аргыг final түлхүүр үг хэрэглэнэ. энэ арга бичих шалтгаан болон энэ арга нь дэд ангид яаж нөлөөлдөг талаар Writing Final Classes and Methods хэсгээс үзээрэй.

`native` – өөр хэлээр бичигдсэн функцыг (Ж: C++ хэл) Java хэлээр бичигдсэн код дотор дуудахдаа ашиглана. Дэлгэрэнгүй мэдээллийг Java Native Interface хэсгээс үзээрэй.

`synchronized` – зэрэг ажиллаж байгаа Thread програмын функцүүд нэг ижил өгөгдөхүүн дээр ажиллах тохиолдол элбэг тохиолддог. Ийм мэдээлэл рүү зөрчилгүй хандахын тулд `synchronized` түлхүүр үгээр зарлах хэрэгтэй. Энэ аргын заглах талаар Threads: Doing Two or More Tasks At Once хэсэгт бичсэн байгаа.

returnType – Арга нь буцаах утгынхаа төрлийг заавал зарлах ёстой. Ямар нэг утга буцаахгүй тохиолдолд void түлхүүр үгийг хэрэглэх хэрэгтэй. Аргуудын буцах утгатай холбоотой асуудлуудаа Returning a Value from a Method хэсэгт тайлбарласан.

methodName – Аргын нэр нь дурын идентификатор байна. Аргыг нэрлэхдээ кодын журам нэр дахин тодорхойлогдох зэргийг тооцох хэрэгтэй Naming a Method хэсэгт энэ талаар тайлбарласан байгаа.

`(` *paramlist* `)` – Арга руу мэдээлэл дамжуулахдаа аргументуудыг нь ашиглана. Дэлгэрэнгүй мэдээллийг Passing Information into a Method or a Constructor хэсгээс авч үзнэ үү.

`throws` *exceptionList* – хэрвээ таны арга гажуудал ялгаруулвал аргын мэдэгдэл дотор нь тэдгээр гажуудлын төрлийг зааж өгөх хэрэгтэй. Дэлгэрэнгүй мэдээллийг Handling Errors with Exceptions хэсгээс авч үзнэ үү .

Аргын сигнатур нь аргын нэр ба хэмжигдэхүүний жагсаалт гэсэн 2 хэсгээс бүрдэнэ.

Аргыг нэрлэх

Хэдийгээр аргын нэр нь дурын идентификатор байх ёстой боловч аргын нэрийг хязгаарлах кодын журам гэж бий. Ерөнхийдөө аргын нэр нь үйл үг гэхдээ эхний үсэг нь жижиг харин залгаас үгнүүд нь том үсгээр эхэлсэн байна.

Байгуулагчууд нь ангийнхаа нэрээр нэрлэгддэг учир аргын нэр нь ангийн нэртэй ижил байж болохгүй.

toString
compareTo
isDefined
setX
getX

Яаж нэрлэх талаар Java –н архитектурын нэрийн журманд энэ талаар тодорхой заасан. Ихэнхдээ аргын нэр нь анги дотор давхцах ёсгүй гэхдээ аргын нэр нь тухайн анги юмуу дээд ангитайгаа давхцах гурван тохиолдол байдаг.

1. арга дахин тодорхойлох
2. арга задлах
3. нэр дахин тодорхойлох

Дээд анги дотроо тодорхойлогдсон аргатай адилхан сигнатур болон буцах төрөлтэй арга нь дээд ангийнхаа аргыг дахин тодорхойлох юмуу далдлах үүрэг гүйцэтгэдэг. Аргыг хэрхэн дахин тодорхойлох, далдлах талаар Overriding and Hiding Methods үзнэ үү.

Өөр өөр хэмжигдэхүүний жагсаалттай аргууд нэг нэртэй байж болно.Үүнийг нэрийн давхцал гэнэ.

Өөр хэл ашигласан нөхцөлд `drawString`, `drawInteger`, `drawFloat` гэх мэт арга болгондоо өөр өөр нэр өгөх хэрэгтэй болдог. Java хэлээр бичиж байгаа нөхцөлд дээрх аргууд нь бүгд адилхан нэртэй байж болно. Гэхдээ аргуудын аргументынх нь төрөл ялгаатай байх ёстой. Тиймээс өгөгдлийг зурах анги нь тус бүрдээ өөр өөр төрлийн аргумент агуулсан Draw нэртэй гурван арга зарлаж болно. Давхацсан аргууд нь тухайн арга руу дамжуулсан Аргументын тоо болон төрлөөрөө ялгагдана.

```
public class DataArtist {
    ...
    public void draw(String s) {
        ...
    }
    public void draw(int i) {
        ...
    }
    public void draw(float f) {
        ...
    }
}
```

Дээрх жишээнд аргументууд нь өөр өөр төрөлтэй байгаа учраас Draw (String s) Draw (int i) аргууд нь хоорондоо ялгаатай. Хооронд нь ялгаж чадахгүй учраас нэр нь адилхан аргументуудынх нь тоо болон төрөл нь адилхан хоёр арга байж болохгүй. Аргуудыг ялгахдаа буцах төрлийг тооцдоггүй учраас буцах төрөл нь ялгаатай гэхдээ сигнатур нь адилхан 2 арга тодорхойлж болохгүй.

Ангийн байгуулагч бичих

Бүх арга нь ядаж нэг байгуулагчтай байх ёстой. Байгуулагч нь тухайн төрлийн шинэ обьект цэнэглэхэд хэрэглэгддэг бөгөөд нэр нь ангийнхаа нэртэй адилхан байх ёстой. Жишээлбэл: stack ангийн цорын ганц байгуулагч нь stack юм.

```
public Stack(int size) {
    items = new Object[size];
}
```

Энэхүү жишээ болгон авсан байгуулагч нь бүхэл аргументынхаа утгаар стекийн анхны хэмжээг тохируулж байна. Байгуулагч бол арга биш тиймээс буцах утга гэж байхгүй. new оператор нь байгуулагчийг дуудаад шинээр үүссэн обьектыг автоматаар буцаана.

Байгуулагч дотор return хэллэгийг ашиглаж болохгүй тогтмол хэмжээтэй стек байгуулах өөр нэг байгуулагч.

```
public Stack() {
    items = new Object[10];
}
```

Энэ 2 байгуулагч нь хоёулаа адилхан нэртэй боловч хэмжигдэхүүнийх нь жагсаалт нь өөр өөр байна. Java платформд байгуулагчуудыг ялгахдаа аргументынх нь тоо болон төрөл дээр тулгуурладаг.

Дээд анги дотор аргументынх нь тоо болон төрөл нь адилхан байх 2 байгуулагч бичиж болохгүй. Java платформ ялгаж чадахгүй. Ингэсэн нөхцөлд эмхэтгэлийн үеийн алдаа гарна. Анги бичихдээ тухайн ангидаа тохирсон байгуулагчийг бичих хэрэгтэй.

Энэ хэсэгт авч үзсэн Rectangle ангийг санацгаая. Энэ анги нь шинээр үүсэх тэгш өнцөгт обьектыг олон янзын замаар цэнэглэж дөрвөн байгуулагч байгуулсан. Онц шаардлагагүй бол байгуулагч бичихгүй байж болно. Ийм байгуулагчгүй ангийн хувьд аргументгүй default (оршмол) байгуулагчийг автоматаар үүсгэнэ. Ийм оршмол байгуулагч нь юу ч хийдэггүй. Өөр ямар ангиуд байгуулагчийг дуудах боломжтой гэдгийг удирдахдаа байгуулагчийн мэдэгдэл дотор дараах хандалтын хувиргуурыг ашиглаж болно.

`private` – зөвхөн энэ анги энэ байгуулагчийг ашиглаж болно. Хэрэв анги доторх бүх байгуулагчууд хаалттай бол энэ анги төл үүсэж цэнэглэх нээлттэй анги аргыг factory (үйлдвэрлэгч арга) гэнэ. Бусад ангиуд энэ аргын төлийг үүсгэхдээ үйлдвэрлэгч ангийг ашиглана.

`protected` – энэ ангийн дэд ангиуд болон нэг багц доторх өөр ангиуд энэ ангийг ашиглана.

`public` – дурын анги энэ байгуулагчийг дуудаж болно.

no specifier – хувиргуургүй үед багцын хандалт. Энэ ангитай нэг багц дотор байгаа өөр ангиуд энэ байгуулагчийг ашиглаж болно.

Байгуулагчууд нь шинэ обьектыг цэнэглэх үүрэг гүйцэтгэдэг. Таны арга болон түүнээс үүссэн шинэ обьектуудыг цэнэглэх өөр арга замуудын талаар Initializing Instance and Class Members хэсгээс үзээрэй.

Арга болон байгуулагч руу мэдээлэл дамжуулах

Арга болон байгуулагч руу мэдээлэл нь тухайн арга болон байгуулагчийн аргументуудын тоо болон төрлийг заадаг.

Жишээ нь: зээлийн хэмжээ хүү зээлийн хугацаа болон зээлийн ирээдүйн утга дээр тулгуурласан банкны зээлийн сарын төлбөрийг тооцоолох доорх аргыг авч үзъе.

```
public double computePayment(double loanAmt, double rate,
                             double futureValue,
                             int numPeriods) {
    double I, partial1, denominator, answer;
    I = rate / 100.0;
```

```
    partial1 = Math.pow((1 + I), (0.0 - numPeriods));
    denominator = (1 - partial1) / I;
    answer = ((-1 * loanAmt) / denominator)
                - ((futureValue * partial1) / denominator);
    return answer;
}
```

Энэ арга нь дөрвөн параметртэй: зээлийн хэмжээ болон хүү, ирээдүйн утга урт. Эхний гурав нь бутархай тоо харин дөрөв дэхь нь бүхэл тоо юм. Эндээс харахад дурын арга юмуу байгуулагчуудын аргументууд нь таслалаар зааглагдсан хувьсагчийн мэдээлэл жагсаалт байх бөгөөд хувьсагчийн мэдээлэл нь төрөл нэр хоёрын хослол байна. computePayment аргын биеэс хархад аргументын утга руу хандахдаа аргументынхаа нэрийг ашиглана.

Аргументын төрөл

Арга болон байгуулагчруу дамжуулах аргумент нь өгөгдлийн дурын төрөл байж болно. Үүнд: ComputePayment арга дотор тодорхойлсон бутархай бүхэл гэх мэт өгөгдлийн эгэл төрөл болон анги бүл гэх мэт заагч төрлүүд байж болно. Аргумент нь бүл байх үйлдвэрлэгчийн аргын жишээ авч үзье.

```
public static Polygon polygonFrom(Point[] listOfPoints) {
    ...
}
```

Энэ жишээнд уг арга нь шинээр Polygon обьект үүсгээд points обьектын жагсаалтаар цэнэглэж байна. (points анги нь x,y координатийг тодорхойлно). Арга руу арга дамжуулж болохгүй. Харин обьект дамжуулаад тухайн аргынхаа аргыг дуудаж болно.

Аргументын нэр

Арга болон байгуулагчийн аргументыг зарлахдаа тухайн аргументдаа нэр өгдөг. Энэ нэр нь уг аргын бие дотроос аргументынхаа өгөгдөлтэй хандахдаа энэ нэрийг ашиглана.

Аргументын нэр нь цараа дотроо ганцхан байх ёстой. Тодруулбал: аргументын нэр нь тухайн арга юмуу байгуулагч доторх өөр аргументын нэртэй адил байх ёсгүй. Шинэ хувьсагчийг далдлах нь таны кодыг уншихад төвөгтэй болгодог бөгөөд зөвхөн байгуулагч юмуу арга дотор ямар нэг гишүүн хувьсагчийн утгыг олгоход хэрэглэдэг.

Жишээлбэл: дараах Circle анги болон түүний setOrgin аргыг авч үзье.

```
public class Circle {
    private int x, y, radius;
    public void setOrigin(int x, int y) {
        ...
    }
}
```

Circle анги нь x, y, Radius гэсэн гурван гишүүн хувьсагчтай. SetOrieng арга нь нэг аргументтай бөгөөд тэдгээрийн нэр нь гишүүн хувьсагчийнхаа нэртэй адилхан байна. Аргын аргумент нь гишүүн хувьсагчдаа далдалж байна.

Тиймээс аргын бие доторх x, y гэсэн нэр нь гишүүн хувьсагчийнхаа нэрийг биш харин аргументынхаа нэрийг заана. Гишүүн хувьсагчтай харилцахын тулд нарийвчилсан нэрийг ашиглана. Дэлгэрэнгүй мэдээлийг Using the this Keyword хэсгээс үзээрэй.

Утга дамжуулах

Аргументууд нь утгаар дамждаг. Арга юмуу байгуулагчийг дуудах үед тэдгээр нь дамжуулсан хувьсагчийнхаа утгыг авдаг. Аргумент нь эгэл төрөл байх үед утгаар дамжуулна гэдэг нь уг арга утгаа өөрчилж чадахгүй гэсэн санааг илэрхийлдэг. Харин аргумент нь заагч төрөл үед утгаар дамжуулна гэдэг нь тухайн арга нь дамжуулсан объектынхоо заагчийг өөрчилж чадахгүй гэхдээ уг обьектынхоо аргыг дуудах нээлттэй хувьсагчийг өөрчлөх боломжтой гэсэн санааг баримталдаг. Энэ ямар утгатай болохыг дэлгэрэнгүй тайлбарлахын тулд getRGBColor Аргыг авч үзье. Энэ арга нь Аргументынхаа утгыг олгох замаар гурван утга буцаахаар оролдож байна.

```
public class Pen {
    private int redValue, greenValue, blueValue;
    ...
    //This method does not work as intended.
    public void getRGBColor(int red, int green, int blue) {
        red = redValue;
        green = greenValue;
        blue = blueValue;
    }
}
```

Ингэх нь огт ажиллахгүй (энэ нь ямар ч үр дүнд хүрэхгүй) red, green болон blue хувьсагчид нь зөвхөн getRGBColor гэсэн арга дотор өөрчлөгддөг. Энэ арга дуудах үедээ эдгээр хувьсагч руу хандах учраас тэнд болсон бүх өөрчлөлтүүд зөвхөн тэнд болно.

Одоо getRGBColor аргаа шинэчилж бичъе.

```
public class RGBColor {
    public int red, green, blue;
}
```

Эхлээд RGB өнгөний улаан, ногоон, хөх утгуудыг хадгалах чадвартай RGBColor гэсэн обьектын цоо шинэ төрлийг үүсгэх хэрэгтэй. Дараа нь аргумент нь RGBColor обьект байхаар getRGBColor аргаар шинэчилж бичье. Энэ арга нь RGBColor Аргументынхаа улаан, хөх, ногоон гишүүн хувьсагчийг тохируулах замаар үзэгнийхээ тохирсон өнгийг буцаана.

```
public class Pen {
    private int redValue, greenValue, blueValue;
    ...
    public void getRGBColor(RGBColor aColor) {
        aColor.red = redValue;
        aColor.green = greenValue;
        aColor.blue = blueValue;
    }
}
```

aColor бол энэ обьектын царааны гадна орших заагч тул getRGBColor арга доторх RGBColor обьектруу хийх аливаа өөрчлөлт нь энэ аргыг буцаасны дараа хадгалагдан үлдэнэ.

Аргаас утга буцаах

Аргын буцах төрлийг ангийн мэдэгдэл дотор зарлаж өгдөг. Утга буцаахдаа аргын бие дотор Return хэллэг ашиглана. Void гэж зарлагдсан дурын арга утга буцаахгүй учраас Return хэллэгийг агуулахгүй. Void гэж зарлагдаагүй ямар ч арга дотор Return хэллэгийг ашиглах ёстой.

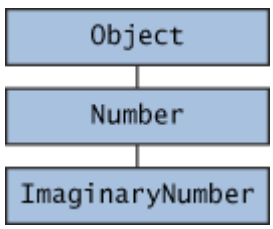
Stack анги доторх isEmpty Аргыг авч үзье.

```
public boolean isEmpty() {
      if (items.size() == 0) {
            return true;
      } else {
            return false;
      }
}
```

Буцах утгын өгөгдлийн төрөл нь аргынхаа зарласан буцах төрөлтэй адилхан байх ёстой. Boolean утга буцаана гэж зарлагдсан аргаас бүхэл төрлийн утга буцааж болохгүй. isEmpty аргын зарласан буцах төрөл нь Boolean бөгөөд энэ аргын хэрэгжилт нь шалгалтын үр дүнгээс хамааруулан true эсвэл false утга буцаана. IsEmpty арга нь эгэл төрөл буцааж байна. Арга нь заагч төрөл бас буцааж болно. Жишээ нь: Stack дотор зарлагдсан pop арга нь object заагч буцаана.

```
public Object pop() {
      if (top == 0) {
            throw new EmptyStackException();
      }
      Object obj = items[--top];
      items[top]=null;
      return obj;
}
```

Аргын буцах төрөл нь ангийн нэр байгаа нөхцөлд буцах обьектынх нь ангийн төрөл нь дараах зурганд үзүүлснээр ImaginaryNumber анги нь object ангийн дэд анги болох Java.lang.Number ангийн дэд анги юм.



Түүнчлэн number буцаах арга зарласан байна.

```
public Number returnANumber() {
    ...
}
```

Энэ арга нь object бус `ImaginaryNumber` буцааж чадна. `ImaginaryNumber` нь number ангийн дэд анги учир number болж чадна. Харин object нь заавал number байх албагүй энэ нь хэлхээ (String) юмуу өөр ямар нэг төрөл байж болно. Буцах төрөл нь үүд (interface) -ийн нэр бас байж болно. Энэ тохиолдолд буцах обьект нь заагдсан үүд (interface) -ийг заавал хэрэгжүүлэх ёстой.

This түлхүүр үг ашиглах

Төл арга юмуу байгуулагчийн дотор this түлхүүр үг нь тохиосон обьект өөрөөр хэлбэл арга юмуу байгуулагч нь дуудаж буй обьектын заагч нь болдог. Төл арга юмуу байгуулагч дотроос тохиосон обьектын дурын гишүүн рүү хандахдаа this түлхүүр үгийг ашиглана, ингэж ажиллах хамгийн түгээмэл шалтгаан нь арга юмуу байгуулагчийн аргумент нь гишүүн хувьсагчийг далдалсан үед тохиолддог.

Жишээлбэл: доорх HSBColor байгуулагч нь уг байгуулагчид дамжуулсан байгуулагчуудаар обьектуудынхаа гишүүн хувьсагчдыг цэнэглэж байна. Уг байгуулагчийн аргумент бүр нь тухайн обьектынхоо аль нэг гишүүн хувьсагчийг далдалж байгаа учраас энэ байгуулагч нь обьектын гишүүн хувьсагчтай харьцахдаа this түлхүүр үгийг ашиглаж байна.

```
class HSBColor {
    private int hue, saturation, brightness;
    public HSBColor (int hue, int saturation,
                     int brightness) {
        this.hue = hue;
        this.saturation = saturation;
        this.brightness = brightness;
    }
}
```

Байгуулагч дотроосоо анги доторх өөр нэг байгуулагчийг дуудахдаа this түлхүүр үгийг ашиглаж болно. Үүнийг байгуулагчийн далд дуудлага гэнэ.

Дээр үзүүлсэн ялгаатай Rectangle ангийн өөр нэг хэрэгжилтийг дор харууллаа .

```
class Rectangle {
    private int x, y;
    private int width, height;
    public Rectangle() {
        this(0, 0, 0, 0);
    }
    public Rectangle(int width, int height) {
        this(0, 0, width, height);
    }
```

```
    public Rectangle(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }
    ...
}
```

Энэ анги хэд хэдэн байгуулагчийг агуулж байна. Байгуулагч бүр нь тэгш өнцөгтийнхөө аль нэг гишүүн хувьсагчийг цэнэглэнэ. Анхны утга нь аргументаараа тодорхойлогдоогүй аль нэг гишүүн хувьсагчийн оршмол утгыг эдгээр байгуулагчууд тодорхойлж байна. Жишээ: аргументгүй байгуулагч нь 0 гэсэн утгыг оршмол утгаа болгон ашиглаж дөрвөн аргументтай байгуулагчаар дуудсан байна. Өмнө дурьдсанчлан эмхэтгүүр нь ямар байгуулагчийг дуудахаа шийдэхдээ аргументуудынхаа төрлийг ашигладаг.

Байгуулагчийн далд дуудлага нь тухайн байгуулагчийнхаа хамгийн эхний мөр байх ёстой.

Ангийн гишүүд рүү хандах хандалтыг удирдах

Тодорхой гишүүн хувьсагчийг ашиглах эсвэл тодорхой Аргыг дуудах боломжтой эсэхийг хандалтын төвшнөөр тодорхойлно. Гишүүн хувьсагч болон аргуудтай харьцах хандалтыг хандалтын private, public, protected, package гэсэн дөрвөн төвшнөөр тодорхойлно. Эдгээрийг доорх хүснэгтээр нэгтгэж харууллаа.

Хандах төвшинүүд

Хувиргуур	Анги	Багц	Дэд анги	Ертөнц
`private`	Y	N	N	N
no specifier	Y	Y	N	N
`protected`	Y	Y	Y	N
`public`	Y	Y	Y	Y

Эхний багана: анги өөрийн гишүүн рүү хандалтын төвшнөөс хамааруулж хандах боломжтой эсэхийг харуулав. Эндээс үзэхэд анги нь өөрийн гишүүд рүү ямар ч тохиолдолд хандана.

Хоёрдахь багана: тухайн ашигтай нэг багц дотор байгаа өөр ангиуд энэ ангийн гишүүд рүү хандах боломжтой эсэхийг харуулж байна.

Багц нь анги болон interface (үүд) ийг бүлэглэн хандалтын хамгаалалт нэрийн боловсронгуй зохицуулалт хийх бололцоог өгдөг. Багцын тухай дэлгэрэнгүй мэдээллийг Creating and Using Packages хэсгээс судлаж болно.

Гуравдахь багана: тухайн ангийн дэд ангиуд нь ямар багц дотор байгаагаас үл хамааран энэ ангийн гишүүн рүү хандах боломжтой эсэхийг харуулж байна.

Дөрөвдэхь багана: бүх ангиуд ангийн гишүүн рүү хандах боломжтой гэдгийг харуулж байна. Хандалтын төвшин нь хоёр замаар нөлөө үзүүлдэг.

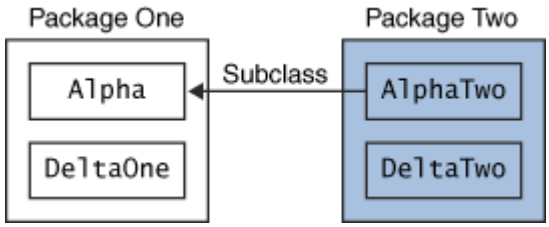
Бусад эх үүсвэрээс ирсэн ангиудыг ашиглах үед (Java платформ ашиглах үед) тэдгээр ангийн ямар гишүүнтэй харьцах боломжтойг хандалтын төвшин харуулдаг.

1. Өөрийн ангийг бичих үедээ Таны анги доторх бүх хувьсагч болон бүх аргын хандалтын төвшин ямар байхыг шийдэх хэрэг гарна.

Хандалтын төвшнийг API –тай холбон ойлгож болно: Хандалтын төвшин ангийн нээлттэй API –д шууд нөлөөлдөг бөгөөд тухайн ангийн ямар гишүүнийг өөр ангиуд ашиглах боломжтой гэдгийг харуулна.

Аргуудаа нэрлэхээс авахуулаад ангийнхаа API -д төлөвлөхөд (тооцоолоход) -өө гишүүдийн хандалтын төвшнийг тодорхойлоход чамгүй хугацаа зарах (анхаарал хандуулах) хэрэгтэй.

Хэд хэдэн багц ангийн гишүүн дээр хандалтын төвшинг авч үзье. Доорх зураг дээр хоорондоо холбоотой дөрвөн ангийг харуулсан байна.



Хандалтын ангийн төвшин

Бусад ангиас гишүүдтэй нь харьцахаар оролдож буй Item ангийн кодыг дор харуулав. Alpha анги нь хандалтын төвшин болгонд харгалзах нэг нэг гишүүн хувьсагч болон арга агуулна. Alpha нь One багц дотор байрлана.

```
package One;
public class Alpha {
      //member variables
      private   int iamprivate = 1;
                        int iampackage = 2;  //package access
      protected int iamprotected = 3;
      public    int iampublic = 4;

      //methods
      private void privateMethod() {
            System.out.println("iamprivate Method");
      }
      void packageMethod() { //package access
            System.out.println("iampackage Method");
      }
      protected void protectedMethod() {
```

```
        System.out.println("iamprotected Method");
    }
    public void publicMethod() {
        System.out.println("iampublic Method");
    }

    public static void main(String[] args) {
        Alpha a = new Alpha();
        a.privateMethod();   //legal
        a.packageMethod();   //legal
        a.protectedMethod(); //legal
        a.publicMethod();    //legal

        System.out.println("iamprivate: "
            + a.iamprivate);    //legal
        System.out.println("iampackage: "
            + a.iampackage);    //legal
        System.out.println("iamprotected: "
            + a.iamprotected"); //legal
        System.out.println("iampublic: "
            + a.iampublic);     //legal
    }
}
```

Өмнөх хүснэгтийн alpha баганад үзүүлснээр alpha анги нь өөрийнхөө бүх гишүүн хувьсагч болон бүх арга руу хандах боломжтой.

Энэ програмын гарц нь:

```
iamprivate Method
iampackage Method
iamprotected Method
iampublic Method
iamprivate: 1
iampackage: 2
iamprotected: 3
iampublic: 4
```

Гишүүний хандалтын төвшин нь тухайн гишүүнийхээ ямар төл бус хэнийг ямар анги хандах боломжтой гэдгийг тодорхойлдог. Тиймээс жишээлбэл, нэг ангийн төлүүд нэг нэгнийхээ хаалттай гишүүд рүү хандах боломжтой. Тэгэхлээр IomPrivate хувьсагчид хувьсагчид тохиосон alpha обьектыг (this) өөр обьекттой харьцуулах төл Аргыг alpha анги руу нэмлээ.

```
package One;
public class Alpha {
    ...
    public boolean isEqualTo(Alpha anotherAlpha) {
        if (this.iamprivate == anotherAlpha.iamprivate) {
```

```
        //legal
            return true;
        } else {
            return false;
        }
    }
}
```

Хандалтын багцын төвшин

Одоо alpha ангийнхаа нэг багц дотор орших Delta ангийг авч үзье. Энэ анги ямар арга болон хувьсагчтай харьцах боломжтой болохыг өмнөх хүснэгтийг багц гэсэн баганаас харж болно.

```
package One;
public class DeltaOne {
      public static void main(String[] args) }
            Alpha a = new Alpha();
            //a.privateMethod();  //illegal
            a.packageMethod();    //legal
            a.protectedMethod();  //legal
            a.publicMethod();     //legal
            //System.out.println("iamprivate: "
            //  + a.iamprivate);   //illegal
            System.out.println("iampackage: "
                  + a.iampackage);   //legal
            System.out.println("iamprotected: "
                  + a.iamprotected); //legal
            System.out.println("iampublic: "
                  + a.iampublic);    //legal
      }
}
```

DeltaOne анги нь alpha ангийн iamprivate болон privateMetod аргаас бусад гишүүдтэй харьцаж болж байна. Хэрвээ энэ кодоос тайлбаруудыг нь авааД эмхэтгэхийг оролдвол эмхэтгэлийн алдаа гарна.

Энэ програмыг хэвээр нь ажиллуулахад гарах гарц нь:

```
iampackage Method
iamprotected Method
iampublic Method
iampackage: 2
iamprotected: 3
iampublic: 4
```

Хандалтын дэд ангийн төвшин

Дараах AlphaTwo анги бол өөр багц доторх alpha ангийн дэд анги юм. Энэ анги ямар гишүүн хувьсагч болон аргуудтай харьцах боломжтой болохыг дээрх хүснэгтийн SubClass баганаас харж болно

```
package two;
import One.*;
public class AlphaTwo extends Alpha {
      public static void main(String[] args) {
            Alpha a = new Alpha();
            //a.privateMethod();   //illegal
            //a.packageMethod();   //illegal
            //a.protectedMethod(); //illegal
            a.publicMethod()       //legal

            //System.out.println("iamprivate: "
            //   + a.iamprivate);    //illegal
            //System.out.println("iampackage: "
            //   + a.iampackage);    //illegal
            //System.out.println("iamprotected: "
            //   + a.iamprotected);  //illegal
            System.out.println("iampublic "
                  + a.iampublic);      //legal

            AlphaTwo a2 = new AlphaTwo();
            a2.protectedMethod();   //legal
            System.out.println("iamprotected: "
                  + a2.iamprotected); //legal
      }
}
```

`AlphaTwo` анги нь alpha төл дотроо (дээд анги)-ийнхаа protectedMethod –ийг дуудах юмуу protected хувьсагчууд хандаж чадахгүй. Гэхдээ AlphaTwo ангийн төл эсвэл AlphaTwo ангийн дэд ангийн төлийн protectedMethod –г дуудах юмуу эсвэл protected хувьсагчууд нь хандах чадвартай. Өөрөөр хэлбэл, дэд анги нь өөрийнхөө хязгаарлагдмал гишүүд рүү харьцахдаа обьект заагчийн төрөл нь ангийнхаа эсвэл дэд ангийнхаа нэртэй адил байх тийм заагч ашиглах бололцоог хандалтын хязгаарлагдмал төвшин олгодог.

Энэ програмын гарц нь:

```
iampublic Method
iampublic: 4
iamprotected Method
iamprotected: 3
```

Хандалтын ертөнцийн төвшин

DeltaTwo анги нь Delta ангитай ямар ч холбоогүй бөгөөд түүнээс өөр багц дотор оршино. Дээрх хүснэгтийн ертөнц баганад харуулснаар DeltaTwo нь alpha ангийн зөвхөн нээлттэй гишүүдтэй харьцаж чадна.

```
package Two;
import One.*;
public class DeltaTwo {
      public static void main(String[] args) {
            Alpha alpha = new Alpha();
            //alpha.privateMethod();   //illegal
            //alpha.packageMethod();   //illegal
            //alpha.protectedMethod(); //illegal
            alpha.publicMethod();      //legal
            //System.out.println("iamprivate: "
            //   + a.iamprivate);      //illegal
            //System.out.println("iampackage: "
            //   + a.iampackage);      //illegal
            //System.out.println("iamprotected: "
            //   + a.iamprotected);    //illegal
            System.out.println("iampublic: "
                  + a.iampublic);       //legal
      }
}
```

DeltaTwo програмын гарц:

```
iampublic Method
iampublic: 4
```

Таны ангийн бусад програм зохиогчид ашиглаж байгаа үед таны кодыг эвдэх явдал гарч эвдрэх болзошгүй үүднээс сэргийлэхэд хандалтын төвшин ашиглаж болно. Хэзээ ямар хандалтын төвшин ашиглах болохыг дараах зөвлөгөөнд тусгалаа.

1. Аль болох хандалтыг хамгийн хатуу хоригийг ашиглана. Онц шаардлага гараагүй бол зөвхөн Private хандалтыг ашигла.
2. Тогтмолоос бусад тохиолдолд нээлттэй гишүүн хувьсагч зарлахыг цээрлэ. Нээлттэй гишүүн хувьсагчид нь төрөл бүрийн уг үндэс болдог. Зөвхөн арга дуудах замаар гишүүн хувьсагчийг өөрчлөх нөхцөлд өөрчлөлтийн талаар бусад анги болон обьектууд руу өөрчилж боллоо гэдгийг мэдээлж болно. Гишүүн хувьсагчдыг нээлттэй болгосон тохиолдолд ийм мэдэгдлийг хийх ямар ч боломжгүй. Хүчин чадлыг илт нэмэгдүүлсэн тохиолдолд хувьсагчийг нээлттэй болгож болно.
3. Хязгаарлагдмал болон багцын гишүүн хувьсагчийн тоог хязгаарлана.
4. Хэрэв гишүүн хувьсагч нь JavaBeansProperty байвал заавал хаалттай болно.

Төл болон анги гишүүдийг ойлгох

Language Basics хэсэгт төл болон анги гишүүдийн талаар товч ойлголт авсан. Анги болон төл гишүүдийн хэрхэн зарлаж, ашиглах талаар бид энд авч үзнэ.

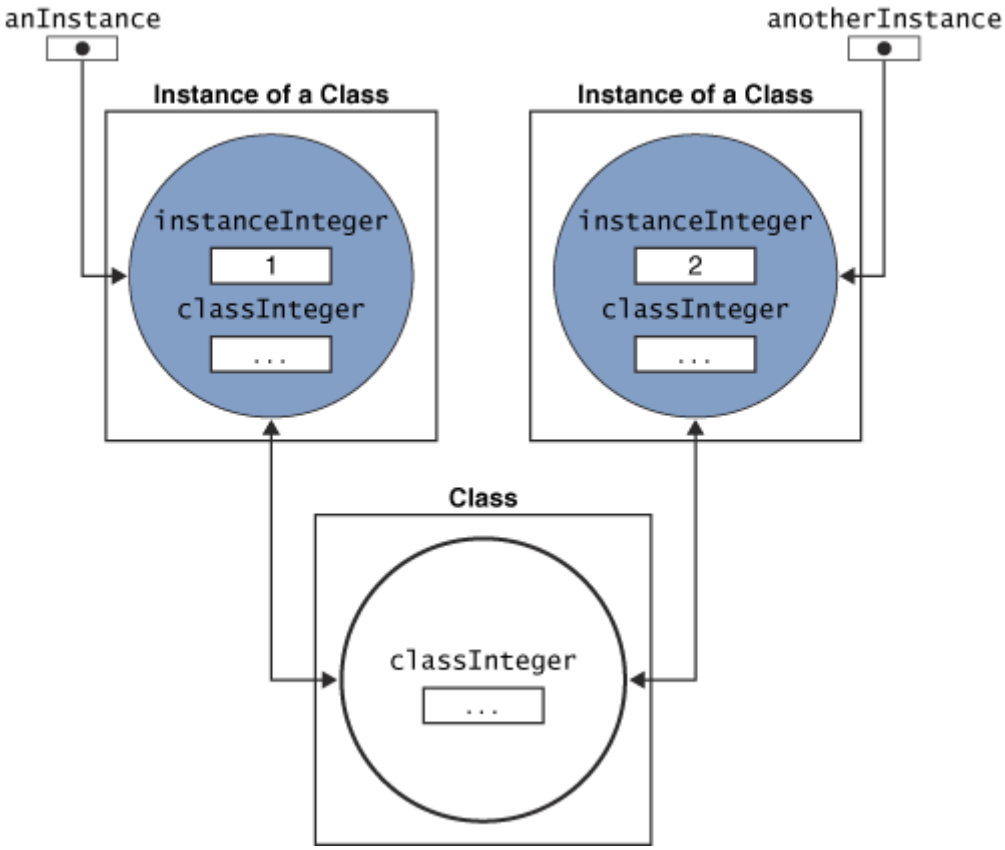
Төл гишүүн хувьсагч , төл арга, анги хувьсагч , анги арга болон main гэдэг анги арга зарласан AClass ангийг дор орууллаа.

```
public class AClass {

    public int instanceInteger = 0;
    public int instanceMethod() {
        return instanceInteger;
    }
    public static int classInteger = 0;
    public static int classMethod() {
        return classInteger;
    }

    public static void main(String[] args) {
        AClass anInstance = new AClass();
        AClass anotherInstance = new AClass();

        //Refer to instance members through an instance.
        anInstance.instanceInteger = 1;
        anotherInstance.instanceInteger = 2;
        System.out.println(anInstance.instanceMethod());
        System.out.println(
        anotherInstance.instanceMethod());

        //Illegal to refer directly to instance members
        //from a class method
        //System.out.println(instanceMethod());    //illegal
        //System.out.println(instanceInteger);     //illegal

        //Refer to class members through the class...
        AClass.classInteger = 7;
        System.out.println(classMethod());

        //...or through an instance.
        System.out.println(anInstance.classMethod());

        //Instances share class variables
        anInstance.classInteger = 9;
        System.out.println(anInstance.classMethod());
        System.out.println(anotherInstance.classMethod());
    }
}
```

Энэ програмын гарц нь

```
1
2
7
7
9
9
```

Энэ програм доторх обьект болон гишүүн хувьсагчууд өөр хоорондоо хэрхэн холбогдсоныг дараах зургаар харууллаа.



Онцгой тохиолдлыг эс тооцвол анги дотор тодорхойлогдсон гишүүд нь төл гишүүн болдог. Тиймээс InstanceInteger болон InstanceMethod гишүүд бол төл гишүүд мөн юм. Програм дотор үүссэн ангийн төл бүрт харгалзах төл хувьсагчийг ажиллуулах систем үүсгэдэг. Иймээс anInstance болон anotherInstance тус бүртээ өөр өөрийн InstanceInteger гишүүнийг агуулдаг. Зөвхөн төлийн заагчаар дамжуулан төл гишүүнтэй харьцах юм уу, төл аргыг дуудаж болно. Illegal тайлбартай кодын хэсгээс мөрийн эхэнд байгаа 2 ташуу зураасыг арилган, програмыг эмхэтгэх гэж оролдвол алдаа өгөх болно.

Анги гишүүнийг зарлахдаа статик хувиргуурыг ашиглана. Main аргаас гадна AClass анги нь ClassInteger болон ClassMethod гэж нэрлэгдэх нэг анги хувьсагч, нэг анги арга зарласан байна. Тухайн ангийн хичнээн төл үүсэхээс үл хамааран уг анги доторх анги хувьсагчийг зөвхөн удаа үүсгэдэг. Тухайн ангийг ашиглах эхний мөчид анги хувьсагчийг үүсгэдэг. Ангийн бүх төлүүд нь тухайн ангийнхаа анги хувьсагчийг хамтран эзэмшдэг. Анги хувьсагч руу хандахдаа нэг бол төлөөр нь эсвэл ангиар нь дамжуулж харьцдаг. Үүний нэгэн адил анги аргыг дуудахдаа ангиар нь эсвэл төлийн заагчаар нь дамжуулж харьцана. ClassInteger хувьсагчийн утга өөрчлөгдөхөд бүх төлүүдэд адилхан өөрчлөгдөнө.

Төл болон анги гишүүдийг цэнэглэх

Анги болон төл гишүүн хувьсагчийн анхны хувьсагчийг өгөхдөө мэдэгдлийг нь ашиглаж болно.

```
public class BedAndBreakfast {
    public static final int MAX_CAPACITY = 10;
    //initialize to 10
    private boolean full = false;
    //initialize to false
}
```

Өгөгдлийн эгэл төрлийн гишүүн хувьсагчийг цэнэглэхэд үүнийг ашиглахад тохиромжтой байдаг. Бүл болон обьект үүсгэхэд ч үүнийг бас ашиглаж болдог. Гэхдээ ингэж цэнэглэхэд хэд хэдэн сул талууд байдаг.

- Цэнэглэхэд кодыг бичихдээ зөвхөн утга хэллэгийг ашиглаж болдог. Жишээ нь If –Else хэллэгийг ашиглаж болохгүй.
- Цэнэглэх илэрхийлэл нь хяналттай гажуудлыг ялгаруулахаар зарлагдсан . Ямар ч аргыг дуудаж болдоггүй . Хэрэв цэнэглэх илэрхийлэх нь NullPointerException гэх мэт биелэгдэх үеийн гажуудал ялгаруулах аргыг дуудвал ийм алдааг сэргээн засварлах боломжгүй байдаг. Дэлгэрэнгүй мэдээлэлийг Handling Errors with Exceptions бүлгээс үзээрэй .

Хэрэв эдгээр сул талуудаас шалтгаалан гишүүн хувьсагчаа мэдэгдэл дотор нь цэнэглэх боломжгүй бол цэнэглэх кодоо өөр газар бичих хэрэгтэй. Анги гишүүн хувьсагчийг цэнэглэхдээ доорх хэсэгт үзүүлсэнээр цэнэглэх кодоо цэнэглэх статик блок дотор бичих хэрэгтэй. Харин төл гишүүн хувьсагчийг цэнэглэхдээ цэнэглэх кодоо байгуулагч дотор бичих хэрэгтэй.

Цэнэглэгч статик блокыг ашиглах

Цэнэглэх статик блокын жишээг дор харуулав.

```
import java.util.ResourceBundle;
class Errors {
    static ResourceBundle errorStrings;
    static {
        try {
            errorStrings =
              ResourceBundle.getBundle("ErrorStrings");
        } catch (java.util.MissingResourceException e) {
            //error recovery code here
        }
    }
}
```

Цэнэглэх статик блок нь угалзан хаалт дотор бичигдсэн ердийн блок код бөгөөд статик түлхүүр үгээр эхэлдэг. Хэрвээ цомог нь олдохгүй бол getBunde арга гажуудал ялгаруулах учраас ErrorStrings гэдэг нөөцийн цомгийг цэнэглэх статик блок дотор цэнэглэж байна. Энэ код нь алдааг нөхөн сэргээх үйлдэл хийх ёстой. Түүнчлэн ErrorString нь анги гишүүн учраас түүнийг байгуулагч дотор цэнэглэж болохгүй. Анги дотор хичнээн ч цэнэглэх статик блок байж болох бөгөөд тэдгээр нь ангийн биеийн хаана ч байж болно. Эх дотор бичигдсэн дарааллын дагуу (зүүнээс баруун, дээрээс доош) статик цэнэглэгчдийг дууддаг.

Төл гишүүдийг цэнэглэх

Дээр дурьдсан шалтгаанаар хувьсагчаа мэдэгдэл дотор нь цэнэглэх боломжгүй бол төл хувьсагчийг цэнэглэхдээ цэнэглэх кодыг ангийнх нь байгуулагч дотор нь бичиж өгнө.

Өмнөх жишээнд авсан ErrorStrings цомог нь анги хувьсагч биш төл хувьсагч байсан бол ErrorStrings цомгийг цэнэглэх кодыг дор үзүүлсэнчлэн байгуулагч дотор бичнэ.

```
import java.util.ResourceBundle;
import java.util.ResourceBundle;
class Errors {
    ResourceBundle errorStrings;
    Errors() {
        try {
            errorStrings =
              ResourceBundle.getBundle("ErrorStrings");
        } catch (java.util.MissingResourceException e) {
            //error recovery code here
        }
    }
}
```

Нэмэлт тайлбар

J2SE 5.0 шинээр нэмэгдсэн нэмэлт тайлбар бол онцгой нөхцөл байдалд эмхэтгүүр юу хийхийг заах бололцоог програм зохиогчдод олгодог.

Жишээлбэл эмхэтгэлийг зарим алдааны мэдээг хэвлэх шаардлагагүй эсвэл тодорхой арга юмуу ангийг дэд анги дотор нь заавал дахин тодорхойлох ёстой гэх мэт зааварчилгааг өгөхөд нэмэлт тайлбарыг ашигладаг. Нэмэлт тайлбарыг өөрөө тодорхойлж болох боловч ийм учир энэ хэрэгслийн талаар энд авч үзэхгүй. Харин Java 5.0 дотор бэлэн тодорхойлогдсон гурван нэмэлт тайлбарыг энд авч үзнэ.

Нэмэлт тайлбарыг хэрэглэхдээ @нэмэлт тайлбар гэсэн хэлбэр ашигладаг бөгөөд үүнийг арга эсвэл ангийг тодорхойлолт дотор хэрэглэж болно. Жишээлбэл:

```
@Override class Cat extends Pet {
}
```

эсвэл

```
@Override void feedTheDog() { }
```

Арга дотор хэрэглэгдэх бэлэн нэмэлт тайлбарыг бүх гурван төрлийн дараах жишээнд харууллаа.

```
import java.util.List;

class Food {}
class Hay extends Food {}
class Animal {
    Food getPreferredFood() {
        return null;
    }
```

```
    @Deprecated
    static void deprecatedMethod() {
    }
}
class Horse extends Animal {
    Horse() {
        return;
    }
    @Override //compiler error if getPreferredFood
              //overloaded, not overridden
    //Notice the return type is different
    //(covariant return type).
    Hay getPreferredFood() {
        return new Hay();
    }
    @SuppressWarnings({"deprecation", "unchecked"})
    void useDeprecatedMethod(List raw) {
        //deprecation warning - suppressed
        Animal.deprecateMethod();
        //unchecked warning - suppressed
        raw.add(new Horse());
    }
}
```

- java.lang.Override

Дээд анги доторх аргыг энэ анги дотор (заавал) дахин тодорхойлох ёстой гэдгийг override нэмэлт тайлбар зааж байна. Дээрх жишээнд харуулснаар Horse анги доторх getPrefferedFood арга нь Animal анги доторх getPrefferedFood аргыг дахин тодорхойлж байна гэдгийг харуулахад override нэмэлт тайлбарыг ашигласан байна. Энэ арга нь дээд ангиасаа (Food) ялгаатай төрөл (Hay) буцааж байгааг анхаарна уу. Үүнийг эсвэл буцах төрөл гэж нэрлэдэг. Дэлгэрэнгүй мэдээллийг Overriding and Hiding Methods Overriding and Hiding Methods хэсгээс үзээрэй. Хэрэв override нэмэлт тайлбартай аргыг дэд анги нь дахин тодорхойлохгүй бол алдааны мэдээлэл өгдөг.

- java.lang.Deprecated

Цаашид ашиглахаас татгалзах ёстой аргыг @deprecated нэмэлт тайлбараар тодотгож өгдөг. Өмнөх жишээнд Animal анги доторх deprecatedMethod –ыг тодорхойлохдоо deprecated нэмэлт тайлбарыг ашигласан байна. DeprecatedMethod аргыг дуудах юмуу дахин тодорхойлох гэж оролдвол эмхэтгэх үеийн сануулга өгнө.

- java.lang.SuppressWarnings

Эмхэтгүүрийн тухайлсан сануулгыг түдгэлзүүлэхийн тулд @SuppressWarnings нэмэлт тайлбарыг ашиглана. Дээрх жишээнд useDeprecatedMethod аргыг тодорхойлохдоо Deprecation, Unchecked гэсэн 2 эмхэтгүүрийн сануулга өгөхийг түдгэлзэх хэрэгтэй гэдэг нэмэлт тайлбарыг өгсөн байна.

Нэмэлт тайлбарын тухай дэлгэрэнгүй мэдээлэлийг JSR 175: A Metadata Facility for the Java Programming Language хэсгээс үзээрэй. Зөвхөн шаардлагатай үед сануулгаас түдгэлзэх ёстой гэдгийг анхааруулж байна. Жишээ нь Horse ангийг бүхэлд нь

SuppressWarnings нэмэлт тайлбараар тодорхойлж болохгүй учир нь энэ код доторх өөр бусад алдаанууд магадгүй нуугдсан байж болзошгүй.

Дүгнэлт

Ангийн тодорхойлолт нь ангийн мэдэгдэл бол ангийн бие гэсэн 2 хэсгээс бүрдэнэ. Ангийн мэдэгдлийн тухай дэлгэрэнгүй мэдээллийг Controlling Access to Members of a Class хэсэг доторх хүснэгтээс хараарай. Ангийн бие нь гишүүн хувьсагч, арга болон тухайн ангийн байгуулагчаас бүрдэнэ. Анги нь өөрийн төлвийн гишүүн хувьсагч дотор хадгалдаг бөгөөд араншингаар хэрэгжүүлэхдээ аргыг ашигладаг. Гишүүн хувьсагчийн мэдэгдэлийг бүх боломжит элементүүдийн жагсаалтуудыг Declaring Classes доторх хүснэгтээс харж болно. Харин аргын мэдэгдлийн бүх боломжит элементүүдийг Declaring Member Variables хэсгээс харж болно. Байгуулагч нь ангийн шинэ төлийг цэнэглэх үүрэгтэй бөгөөд ангийнхаа нэртэй адилхан нэртэй байдаг.

Гишүүн хувьсагч болон аргуудтай харьцах хандалтыг удирдахаас тухайн гишүүний мэдэгдэл дотор нь private юмуу public гэх мэт хандалтын хувиргуурыг ашигладаг.

Анги гишүүн хувьсагч юмуу анги аргыг тодорхойлохдоо тухайн гишүүнийхээ мэдэгдэл дотор static түлхүүр үгийг ашиглана. Зориудаар статик гэж зарлаагүй гишүүн бол төл гишүүн юм. Анги хувьсагчыг тухайн ангийн бүх төлүүд хамтран эзэмших бөгөөд тэдгээртэй ангийнх нь нэрээр дамжуулан харьцдаг. Ангийн төлүүд нь тус бүртээ өөр өөрийн төл хувьсагчийг авах бөгөөд тэдгээртэй төлийн заагчаар нь дамжуулж харьцана.

Асуулт болон дасгал

Асуулт

1. Доорх ангийг авч үзье.

```
public class IdentifyMyParts {
    public static int x = 7;
    public int y = 3;
}
```

a. IdentifyMyParts ангид хичнээн анги хувьсагч агуулсан байна? Нэрнүүд нь юу вэ?
b. IdentifyMyParts ангид хичнээн төл хувьсагч агуулсан байна? Нэрнүүд нь юу вэ?
c. Дараах кодын гарц ямар байх вэ?

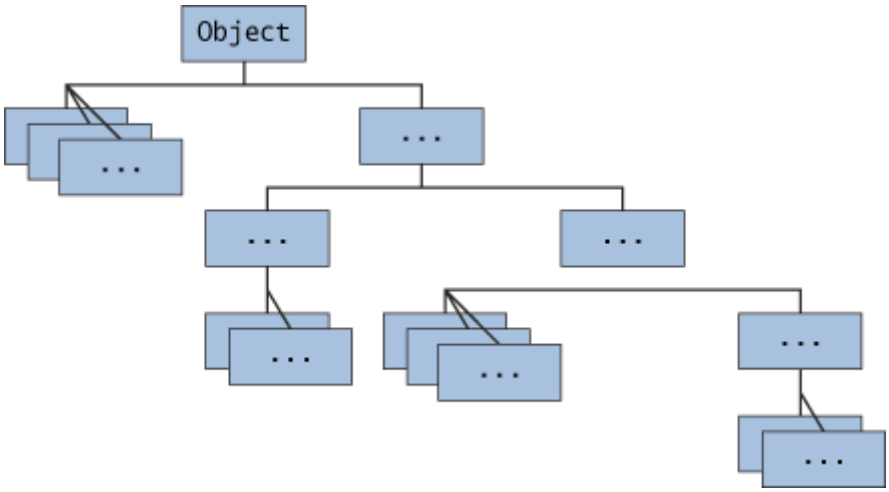
```
IdentifyMyParts a = new IdentifyMyParts();
IdentifyMyParts b = new IdentifyMyParts();
a.y = 5;
b.y = 6;
a.x = 1;
b.x = 2;
System.out.println("a.y = " + a.y);
System.out.println("b.y = " + b.y);
System.out.println("a.x = " + a.x);
System.out.println("b.x = " + b.x);
```

Дасгал

1. Төлүүд нь хөзрийн тавиур дээрх хөзөр дүрсэлсэн анги бич.
2. Төлүүд нь хөзрийн тавиур дүрсэлсэн анги бич.
3. Өөрийн deck болон card ангиудыг шалгах жижиг програм бич. Програм хөзрийн тавиур үүсгэн хөзрүүдээ харуулсан байхаар энгийн байлга.

Удамшлыг зохицуулах

Java.lang багц дотор тодорхойлсон object нь бүх ангиудад шаардагдах араншинг тодорхойлон хэрэгжүүлсэн байдаг. Доор зурагт үзүүлсэнчлэн object ангиас олон анги удамшиж тэдгээр ангиудаас өөр ангиуд удамших зэргээр object –н шатлал үүсгэнэ.



Бүх ангиудын ерөнхий анги нь шатлалын оройд нь байрлах оройн анги болно. Шатлалаас доошлох тусам обьектууд нь илүү тусгаарласан араншинтай болно. Өөр ангиас удамших замаар дэд анги үүснэ. Өвөг анги гэдэг тухайн ангийн шууд өвөг юмуу бусад удамшуулж буй ангиуд юм. Нэг нь зөвхөн ганц шууд дээд ангитай байна.

Дэд анги нь дээд ангийнхаа бүх гишүүн хувьсагчид болон аргыг удамшуулж авдаг. Тэгэхдээ дэд анги нь удамшиж авсан 1. хувьсагч болох 2. аргуудтай бүрэн харьцах албагүй.

Жишээ нь: дэд анги нь дээд ангиасаа удамшиж ирсэн хаалттай гишүүн өгөгдөлтэй харьцаж чадахгүй. Зарим хүмүүс үүнйи удамшил яваагүй хэмээн ойлгох тал бий энэ бол тийм биш юм. Ялангуяа дотоод анги ашиглах үед үүнийг мэдэх нь чухал. Учир нь дотоод анги нь агуулж буй ангийнхаа хаалттай гишүүнтэй хандах боломжтой байдаг. Байгуулагч бол гишүүн биш учир дэд ангидаа удамшихгүй.

Аргыг дахин тодорхойлох ба далдлах

Дээд анги доторх гол аргын сигнатур (нэр, гишүүн өгөгдлийн тоо) буцах төрөлтэйгөө адил тодорхойлсон дэд ангийн тал арга нь дээд ангийнхаа тухайн арга дахин тодорхойлдог. Аргын сигнатур бол аргынхаа нэр, хэмжигдэхүүнийхээ тоо болон төрлөөс бүрддэг. Дэд ангиудын аргыг дахин тодорхойлох чадвар нь өөртэйгээ (хамгийн төстэй) араншин бүхий дээд ангийг удамшуулан араншинг нь өөртөө тохируулан өөрчлөх боломжийг дэд ангиудад олгодог. Жишээ нь: object анги нь тухайн нэг төрлийн төлийн хэлхээ дүрслэлийг буцаадаг toString нэртэй төл аргыг агуулдаг. Бүх анги энэ аргыг удамшуулдаг. Object анги доторх энэ аргын хэрэгжилт нь дэд ангиудад ач

холбогдолгүй. Тэгэхээр ангийнхаа тухай илүү дэлгэрэнгүй мэдээллийг өсгөхийн тулд энэ аргыг дахин тодорхойлохыг зөвлөж байна. Энэ нь ялангуяа зүгшрүүлэлт хийхэд ач холбогдолтой байдаг. toString –г дахин тодорхойлох жишээг доор харууллаа.

```
public class MyClass {
    private int anInt = 4;
    //Overrides toString in Object class.
    public String toString() {
        return "Instance of MyClass. anInt = " + anInt;
    }
}
```

Дахин тодорхойлсон аргын нэр, хэмжигдэхүүнүүдийн тоо болон төрөл, буцах төрөл нь дахин тодорхойлуулж буй аргатайгаа яг адил байна. (Үнэн хэрэгтээ дэд ангийн буцах төрөл нь дээд ангийнхаа буцах төрлийн дэд анги байж болно.) Дахин тодорхойлсон арга нь дахин тодорхойлуулах аргаасаа ялгаатай. throws тодорхойлолт оруулж болно. Түүнчлэн дахин тодорхойлсон аргын хандалтын хувиргуур нь дахин тодорхойлуулж буй ангийнхаа хандалтын төвшингөөс дээд төвшинд байж болно. Жишээ нь: дээд анги доторх хязгаарлагдмал (protected) арга нь дэд ангидаа хаалттай байж болохгүй нээлттэй байж болно.

Хийсвэр анги болон аргуудын талаар The clone Method хэсгийг үзээрэй. Дээд ангийдаа final гэж зарлагдсан аргыг дэд ангид нь дахин тодорхойлж болохгүй. (Тодорхойлолт ёсоор эцсийн аргуудыг дахин тодорхойлж болохгүй)

Хэрэв та үүнийг дахин тодорхойлох гэж оролдвол эмхэтгүүр нь алдааны мэдээлэл өгнө. Эцсийн аргын дэлгэрэнгүй мэдээллийг Writing Final Classes and Methods хэсгээс авна уу.

Дээд ангидаа abstract гэж зарлагдсан ангийг дэд анги нь заавал дахин тодорхойлох ёстой эсвэл өөрийгө abstract болгон зарлах ёстой. Үүний дэлгэрэнгүй мэдээллийг Writing Abstract Classes and Methods хэсгээс авна уу.

Тухайн аргын аргументын тоо, төрлийг өөрчлөх замаар аргыг давхардуулах бололцоог олгодог болохыг бид мэднэ. Дээд анги доторх аргуудыг давхардуулах болоцоо бас бий. toString аргыг давхардуулах жишээг харууллаа.

```
public class MyClass {
    private int anInt = 4;
    //Overrides toString in Object class.
    public String toString() {
        return "Instance of MyClass. anInt = " + anInt;
    }
         //Overloads toString method name to provide
         //additional functionality.
    public String toString(String prefix) {
        return prefix + ": " + toString();
    }
}
```

Энэ жишээнд харуулснаар дээд ангийн аргыг давхардуулах замаар ангийн чадварыг баяжуулна (өргөжүүлнэ). Дээд анги доторх аргатай адилхан нэртэй арга бичих үед уг

аргаа дахин тодорхойлж байна уу эсвэл давхардуулж байна уу гэдгийг нягтлах үүднээс аргументын жагсаалт болон буцах төрлөө сайтар шалгах хэрэгтэй.

Хэрэв дэд анги нь ангийнхаа аргыг дээд ангийнхаа аргатай адил сигнатуртай тодорхойлсон бол дэд ангийн уг арга нь (тодорхойлсон) дээд ангийнхаа харгалзах аргыг далдлана. Далдлах, дахин тодорхойлох хоёрын хооронд том ялгаа бий. Үүнийг жишээн дээр авч үзье. Энэ жишээ нь хоёр ангиас бүрдэнэ. Эхний анги нь нэг төл арга, нэг анги арга агуулсан animal анги юм.

```
public class Animal {
    public static void hide() {
        System.out.println("The hide method in Animal.");
    }
    public void override() {
        System.out.println("The override method in Animal.");
    }
}
```

Хоёрдох анги буюу animal –н дэд анги нь cat гэдэг нэртэй.

```
public class Cat extends Animal {
    public static void hide() {
        System.out.println("The hide method in Cat.");
    }
    public void override() {
        System.out.println("The override method in Cat.");
    }

    public static void main(String[] args) {
        Cat myCat = new Cat();
        Animal myAnimal = (Animal)myCat;
        myAnimal.hide();
        myAnimal.override();
    }
}
```

Энэхүү cat анги нь animal доторх override нэртэй төл аргыг дахин тодорхойлж animal доторх hide() нэртэй анги аргыг далдлаж байна. Энэ ангийн main арга нь cat ангийн төл үүсгэж animal заагч болгон хувиргаад эцэст нь үүссэн төлийн hide болон override –г дуудаж байна.

Энэ програмын гарц нь:

```
The hide method in Animal.
The override method in Cat.
```

Энэ тохиолдолд далдлагдсан арга нь ажиллахдаа дээд анги доторх аргыг дуудаж, харин дахин тодорхойлсон арга нь ажиллахдаа дэд ангийнхаа аргыг дуудаж байна.

Анги аргын хувьд: Заагчийн аргыг дуудахдаа тухайн заагчийнхаа эмхэтгэх үеийн төрөл дотор тодорхойлсон аргыг дууддаг. Тиймээс animal анги дотор тодорхойлсон hide арга

дуудагдаж байна. Төл аргын хувьд заагчийн аргыг дуудахдаа тухайн заагчийнхаа {гүйлтийн үеийн} (runtime) төрөл дотор тодорхойлсон аргыг дууддаг (сонгодог).

Энэ жишээнд myAnimal заагчийн (гүйлтийн үеийн) төрөл нь cat юм. Тиймээс cat дотор тодорхойлсон override арга дуудагдаж байна. Анги арга төл аргаар дахин тодорхойлж болохгүй мөн төл аргыг анги аргаар далдлаж болохгүй. Дээд анги доторх аргатай адилхан сигнатуртай арга тодорхойлоход юу тохиолдохыг доор хүснэгтээр харуулав.

Аргыг өвөг ангийн арга шиг адил сигнатуртай тодорхойлох

	Өвөг ангийн Instance арга	**Өвөг ангийн Static арга**
Instance арга	Дахин тодорхойлно (буцах төрөл нь мөн л адилхан байх ёстой)	Эмхэтгэлийн үеийн алдааг үүсгэнэ
Static арга	Эмхэтгэлийн үеийн алдааг үүсгэнэ	Далдлана

Гишүүн хувьсагчийг далдлах

Хэрэв гишүүн хувьсагчийн нэр нь дээд ангийнхаа гишүүн хувьсагчийн нэртэй адилхан бол тэдгээрийн төрөл нь ялгаатай байсан ч хамаагүй дээд ангийнхаа гишүүн хувьсагчийг далдлана. Дэдд анги дотроос дээд ангийн гишүүн хувьсагчтай харьцахдаа ердийн нэрийг нь ашиглаж болохгүй. Үүний оронд super түлхүүр үгийг ашиглана. Ер нь гишүүн хувьсагчийг далдлахаас зайлсхийх нь дээр шүү.

Super түлхүүр үгийг ашиглах

Дээд ангийнхаа ямар нэг аргыг дахин тодорхойлсон нөхцөлд дахин тодорхойлуулсан аргаа дуудахдаа super түлхүүр үгийг ашиглана. Далдлагдсан гишүүн хувьсагч руу харьцахдаа super ашиглаж бас болно. Superclass гэдэг ангийг авч үзье.

```
public class Superclass {
    public boolean aVariable;

    public void aMethod() {
        aVariable = true;
    }
}
```

Энд aMethod аргыг дахин тодорхойлж aVariable хувьсагчийг далдалсан subclass нэртэй дэд ангийг харууллаа.

```
public class Subclass extends Superclass {
    public boolean aVariable; //hides aVariable in Superclass
    public void aMethod() { //overrides aMethod in Superclass
        aVariable = false;
        super.aMethod();
        System.out.println(aVariable);
        System.out.println(super.aVariable);
    }
```

```
}
```

Subclass анги доторх aVariable нэр бол Superclass анги доторх хувьсагчийг далдалсан Subclass анги дотор тодорхойлогдсон хувьсагч юм. Үүний нэгэн адил aMethod гэсэн нэр нь Superclass доторх аргыг дахин тодорхойлсон Subclass дотор тодорхойлогдсон арга юм. Иймд Superclass ангиас удамшсан aVariable болон aMethod –руу хандахдаа Subclass нь super ашиглан нарийвчилсан нэрийг ашиглана. Тиймээс Subclass ангийн aMethod арга доторх хэвлэх хэллэгүүд нь үүнийг хэвлэнэ.

```
false
true
```

Дээд ангийнхаа байгуулагчийг дуудахдаа байгуулагч дотроо super ашиглаж бас болно. Дүрс хөдөлгөх зориулалттай Thread ангийн дэд ангийн кодоос доор хэсэгчлэн харууллаа. AnimationThread ангийн байгуулагч нь кадрын хурд, зургуудын тоо зэрэг оршмол утгуудыг бүрдүүлээд дараа нь зургуудаа ачааллаж байна.

```
class AnimationThread extends Thread {
    int framesPerSecond;
    int numImages;
    Image[] images;

    AnimationThread(int fps, int num) {
        super("AnimationThread");
        this.framesPerSecond = fps;
        this.numImages = num;
        this.images = new Image[numImages];

        for (int i = 0; i <= numImages; i++) {
            ...
            // Load all the images.
            ...
        }
    }
    ...
}
```

AnimationThread ангийн дээд анги тодруулбал Thread ангийн байгуулагчийг дуудсан мөрийг дармалдаж харууллаа. Энд харуулсан Thread гэсэн байгуулагчийн string хүлээн авч Thread ангидаа нэр болгон өгч байна. Дээд ангийн байгуулагчийг илээр дуудах үед энэхүү дуудлага дэд ангийн байгуулагч доторх хамгийн эхний хэллэг байх ёстой. Дээд төвшний цэнэглэлтийг эхэлж хийх ёстой. Дээд ангийн байгуулагчийг илээр дуудаагүй үед байгуулагч бүрийн бүх хэллэгийн эхэнд аргументгүй байгуулагчийг автоматаар нэмдэг.

Обьектын ураг болох

Ангийн шатлалын модны оройд Object анги байрладаг. Бүх анги нь object ангийн шууд буюу дам ураг нь болдог. Энэ анги нь бүх обьектуудад нийтлэг төлөв болон араншинг тодорхойлсон байдаг. Тухайлбал өөрийгөө өөр нэг обьекттой харьцуулах хэлхээ рүү хөрвүүлэх, нөхцлийн хувийг хүлээх, нөхцлийн хувьсагчийг өөрчилснийг бусад обьектод мэдээллэх, обьектын ангийг буцаах гэх мэт юм.

Object анги доторх аргуудын жагсаалт:

- clone
- equals and hashCode
- finalize
- toString
- getClass
- notify, notifyAll, and wait

notify, notifyAll болон wait аргаас бусдыг нь энд тайлбарлалаа. Програм доторх бие биеэсээ хамааралгүй гүйж буй Thread –үүдийн үйл ажиллагааг зохицуулахад хэрэглэгддэг notify, notifyAll, болон wait аргын талаар Threads: Doing Two or More Tasks At Once дотор тайлбарласан байгаа.

clone арга

Байгаа обьектоос обьект үүсгэхэд clone аргыг хэрэглэнэ. clone үүсгэхдээ:

aCloneableObject.

Энэ аргыг дуудахдаа хувилагдаж буй обьект нь cloneable интерфейсийг хэрэгжүүлсэн эсэхийг шалгадаг. Хэрэв энэ аргыг хэрэгжүүлээгүй бол `CloneNotSupportedException` алдаа үүснэ. Хэдийгээр object анги нь clone аргыг хэрэгжүүлдэг боловч cloneable интерфейсийг хэрэгжүүлнэ гэж зарлаагүй байдаг тул энэ интерфейсээ хэрэгжүүлэхээ илээр зарлаагүй бол тоон ангийг хувилаж (clone хийж) болохгүй. Хэрэв clone аргыг дуудаж буй арга нь cloneable интерфейсийг хэрэгжүүлсэн бол object ангийн clone арга нь эх ангитай адилхан үүсэж гишүүн хувьсагчийг нь object –н гишүүн хувьсагчуудаа адил утгаар цэнэглэнэ.

Ангийг хувилагдах боломжтой болгох хамгийн хялбар зам бол ангийнхаа тодорхойлолт дотор implements cloneable гэдэг үгийг нэмэх явдал. Зарим обьектын хувьд object ангийн clone аргыг шууд ашиглахад хангалттай байдаг. Зарим ангиудад clone аргыг дахин тодорхойлох шаардлага гардаг.

Object –н бүл заасан гишүүн хувьсагч агуулсан stack ангийг авч үзье. Хэрэв stack нь object ангийн clone аргыг ашиглах бол эх stack болон түүний хуулбар нь нэг бүлрүү заана. Нэг stack –г өөрчлөхөд нөгөөдөх нь бас өөрчлөгдөнө гэсэн үг бөгөөд энэ нь хүсэх зүйл биш юмаа.

Бүлийг хувилахдаа эх stack болон хуулбар нь өөр өөр байхаар stack ангийн clone аргыг шинэчлэн бичсэнийг доор харууллаа.

```
public class Stack implements Cloneable {
   private Object[] items;
   private int top;
   ...
   // code for Stack's methods and constructor not shown
   protected Stack clone() {
      try {
         Stack s = (Stack)super.clone(); //clone the stack
         s.items = (Object)items.clone(); //clone the array
         return s; // return the clone
```

```
        } catch (CloneNotSupportedException e) {
           //This shouldn't happen because Stack is Cloneable.
            throw new InternalError();
        }
    }
}
```

Stack ангийн clone арга нь харьцангуй хялбар шийдэлтэй юм. Эхлээд object ангийн clone аргыг super.clone гэж ду удахад stack обьект үүсч цэнэглэгдэнэ. Энэ мөчид эх stack болон түүний хуулбар нь нэг жагсаалтруу зааж байна. Энэ анги нь stack буцаах бөгөөд энэ нь object.clone аргыг буцах төрлийн дэд анги байна.

Санамж: clone арга нь хуулбар үүсгэхдээ хэзээ ч new ашиглан байгуулагчийг нь дуудах ёсгүй. Харин уг арга нь super.clone аргыг дуудаж зөв төрлийн обьектыг үүсгэн, зөв утгуудаа хуулбарлах ёстой.

---

equals болон hashCode арга

equals арга нь хоёр обьектыг хоорондоо тэнцүү эсэхийг шалгаж тэнцүү бол true утга буцаана. Object анги доторх equals арга нь хоёр обьект хоорондоо тэнцүү эсэхийг шалгахдаа (==) операторыг ашиглана. Хэрэв харьцуулж буй обьектууд нь яг нэг обьектыг зааж байгаа бол true утга буцаана.

Зарим ангиудын хувьд нэг төрлийн хоёр өөр обьектын утга нь адил үед тэдгээрийг хоорондоо тэнцүү гэж үзэх тохиолдол гардаг. one ба anotherOne гэж хоёр integer обьектыг тэнцүү эсэхийг шалгасан кодыг авч үзье.

```
Integer one = new Integer(1);
Integer anotherOne = new Integer(1);
if (one.equals(anotherOne)) {
    System.out.println("objects are equal");
}
```

Энэ програм нь хэдийгээр one болон anotherOne хоёр нь хоёр өөр обьект зааж байгаа ч гэсэн обьектууд хоорондоо тэнцүү гэж харуулна. Тэдгээрийн агуулж буй бүхэл утгууд нь адил учраас тэдгээрийг тэнцүү гэж үзэж байна.

Тэнцүүлэх операторыг хэрэглэхэд тохиромжгүй байх үед equals аргыг дахин тодорхойлох шаардлагатай. equals аргыг дахин тодорхойлох үед hashCode аргыг бас дахин тодорхойлох хэрэгтэй.

hashCode аргыг буцах утга нь тухайн обьектын hash table доторх байрлалыг заасан бүхэл тоо байна. Обьект нь hashCode үүсгэх бүрдээ өөр өөр код биш харин нэг л hashCode үүсгэх ёстой. Гэхдээ олон обьект нэг hashCode –той байж болно. (обьект болгон өөр өөрийн hashCode –той байх албагүй.) Hash функцыг алдаагүй бичихэд төвөгтэй биш. Адилхан обьектуудад адилхан hashCode буцаадаг байхад л хангалттай. Hash функцыг оновчтой зохион байгуулах нь хүнд асуудал байдаг.

Зарим ангийн хувьд Hash функц нь тодорхой байдаг. Жишээ нь: integer обьектын hashCode нь бүхэл тоон утга байдаг.

Finalize арга

Object анги доторх finalize арга нь хог цуглуулахын өмнө обьектыг цэвэрлэхэд хэрэглэнэ. Энэ аргын тухай дэлгэрэнгүй мэдээллийг Cleaning Up Unused Objects хэсэгт үзсэн. Finalize аргын систем нь автоматаар дууддаг бөгөөд ихэнх ангиуд үүнийг дахин тодорхойлох шаардлагагүй байдаг. Тиймээс үүнийг анзаарахгүй орхиж болно.

toString арга

Object ангийн toString арга нь тухайн обьектынхоо string дүрслэлийг буцаана. Обьектыг тухайлбал double ангийн төлний бичвэр хэлбэрээр нь харахын тулд System.out.println дотор toString аргыг ашиглаж болно.

System.out.println(new Double(Math.PI).toString());

Обьектын string дүрслэл нь тухайн обьектоос бүрэн хамааралтай байдаг. Double обьектын хэлхээ дүрслэл нь тухайн обьектоосоо хамаарч өөр өөр байдаг. Double обьектын бичвэр хэлбэрт хувирласан бутархай тоо байдаг. Тиймээс дээрх код нь `3.14159` гэж хэвлэнэ.

Зүгшрүүлэх үед toString нь чухал үүрэг гүйцэтгэдэг. Тиймээс бүх анги дотроо энэ аргыг дахин тодорхойлох нь зүйтэй.

getClass арга

Энэ арга нь обьектын гүйлтийн үеийн (runtime) ангийг буцаана. Энэ арга нь class обьект буцаадаг бөгөөд энэ class обьектоор дамжуулан ангийн нэр, ангийн дэд анги, интерфейсүүдийн нэр (хэрэгжүүлсэн) гэх мэт ангийн тухай мэдээллийг авч болно. getClass арга нь дахин тодорхойлогддоггүй. Обьектынхоо ангийг аваад тухайн ангийнхаа нэрийг хэвлэсэн кодыг доор харуулав.

```
void PrintClassName(Object obj) {
    System.out.println("The Object's class is "
                       + obj.getClass().getName());
}
```

Эмхэтгэх үеийнх (compile time) нь төрлийг мэдэхгүйгээр ямар нэг ангийн төлийг шинээр үүсгэхэд class обьектыг хэрэглэх нь элбэг. Доорх жишээнд өгөгдсөн обьектын ангиар нь шинэ төл үүсгэх бөгөөд уг обьект нь object ангиас удамшсан дурын анги байж болно.

```
Object createNewInstanceOf(Object obj) {
    return obj.getClass().newInstance();
}
```

Ангийнхаа нэрийг мэдэж буй үед ангийнхаа нэрээс class обьектыг гарган авч байна. String ангийн class обьектын авах хоёр хувилбарыг доор үзүүлэв.

```
String.class
Class.forName("String")
```

Үүнээс эхнийх нь илүү үр ашигтай хувилбар юм.

Төгс анги болон аргыг бичих

Төгс ангиуд

Анги зарлахдаа төгс, өөрөөр хэлбэл тухайн ангийн дэд ангийг үүсгэх боломжгүй болгон тодорхойлж болно. Ингэх 2 шалтгаан байж болно.

1. Системийн аюулгүй байдлыг дээшлүүлэх
2. Програмын дизайныг илүү объект хандлагатай болгох

- **Аюулгүй байдал**

Энгийн дэд анги үүсгээд түүгээрээ жинхэнэ ангийг орлуулах замаар системийг өөрчлөх боломж хакеруудад байдаг. Дэд анги нь жинхэнэ ангиасаа бараг ялгагдахгүй, гэхдээ хувийн мэдээлэл авах эсвэл ямар нэг гэмтэл учруулах зэрэг хортой үйлдлүүд хийж болно. Иймэрхүү хөнөөлөөс сэргийлэхийн тулд ангиа төгс болгон зарлаж дэд ангийг үүсгэхийг нь хориглож болно. Яг ийм шалтгаанаар *string* анги нь төгс анги байдаг. Энэ анги нь Java системийн хамгийн чухал үйлдлийн нэг тул string биш *java.lang.string*-ийн төрлийн *string* байх ёстой гэсэн баталгаа шаарддаг. Ийм баталгааг бүрдүүлснээр бүх *string* хэлхээнүүд ямар нэгэн харш нөлөө үзүүлэхийг хаадаг.

Хэрвээ бичсэн ангийн дэд ангийг эмхэтгэхийг оролдвол энэ програмыг эмхэтгэхээс татгалзаж алдааны мэдээ өгнө. Мөн түүнчлэн системийг өөрчилсөн хэсгийг *bytecode*-ийн түвшинд давхар шалгадаг. Үүний тулд тухайн ангийг төгс ангийн дэд анги мөн эсэхийг шалгадаг.

- **Дизайн**

Програмын дизайныг илүү объект хандлагатай болгохын тулд ангийг төгс болгон зарлаж болно. Таны анги төгс төгөлдөр, өөрөөр хэлбэл дэд анги үүсгэх (салбарлуулах) шаардлагагүй үед төгс болгон зарлана.

Ангиа төгс болгон зарлахдаа ангийн мэдэгдэл дотрох *class* түлхүүр үгийн өмнө *final* гэж бичнэ. Жишээлбэл: *ChessAlgorithm* ангийн төгс болгон зарлахын тулд үүний мэдэгдлийг дараах маягаар бичнэ.

```
final class ChessAlgorithm {
    ...
}
```

ChessAlgorithm ангиас дэд анги үүсгэх аливаа оролдлогын үр дүнд эмхэтгэлийн алдаа гарна.

Төгс аргууд

Хэрвээ ангиа бүхэлд нь төгс болгон зарлахын оронд ангийнхаа зарим нэг аргуудыг төгс болгон зарлаж болно. Тухайн аргыг дэд анги дотор нь дахин тодорхойлох боломжгүй гэдгийг заахын тулд аргын дотор *final* түлхүүр үгийг ашиглана. Объектын ангийн зарим арга нь төгс зарим нь ердийн байдаг. Объектын төрлийг тогтвортой байлгахад чухал үүрэгтэй өөрчлөгдөх ёсгүй аргуудыг төгс болгон зарладаг. Жишээлбэл: *ChessAlgorithm*

ангийг бүхэлд нь төгс болгохын оронд зөвхөн nextmove аргыг төгс болгон зарлаж болно.

```
class ChessAlgorithm {
    ...
    final void nextMove(ChessPiece pieceMoved,
                        BoardLocation newLocation) {
        ...
    }
    ...
}
```

Хийсвэр анги болон арга бичих нь

Хийсвэр ангиуд

Зарим үед төлөвлөх боломжгүй хийсвэр ухагдахууныг анги дотор дүрслэх шаардлага гардаг. Жишээлбэл: Хүнсийг авч үеъе. Яг үнэндээ хүнс гэж бодитой зүйл байдаггүй. Харин лууван, алим, талх гэж байдаг. Хүнс нь бидний идэх боломжтой зүйлсийг уртасгасан хийсвэр ухагдахуун юм. Тийм болохоор хүнс нь бодитойгоор байдаггүй. Үүний нэг адилаар төлөвлөх боломжгүй хийсвэр ухагдахууныг засварлах шаардлага программ бичих үед гардаг. Жишээлбэл: Тоо гэсэн хийсвэр ухагдахууныг тусгасан number анги байна. Тоог загварчлах хандлага нь зөв, гэхдээ ерөнхий тоон объект үүсгэх нь утгагүй хэрэг. Харин Number анги нь Integer болон Float гэх мэт ангиуд нь дээд ангийн үүрэг гүйцэтгэж тэдгээр дэд ангиуд нь тодорхой төрлийн тоог хэрэгжүүлэх нь илүү оновчтой. Төллүүлэх боломжгүй хийсвэр ухагдахууныг тусгасан Number мэтийн ангийг хийсвэр анги гэнэ. Хийсвэр ангиас зөвхөн дэд ангийг үүсгэж болно. Хийсвэр анги зарлахдаа ангийг мэдэгдэл дотрох class түлхүүр үгийн өмнө abstruct гэж бичнэ.

```
abstract class Number {
    ...
}
```

Хэрвээ хийсвэр ангийг төллүүлэх гэж оролдвол эмхтгүүр алдааны мэдээ өгдөг.

Хийсвэр аргууд

Хийсвэр анги нь хийсвэр аргууд буюу хэрэгжилтгүй аргууд агуулж болно. Энэ нөхцөлд хийсвэр анги нь дэд ангидаа зориулсан программын бүх interface-ийг тодорхойлно, гэхдээ тэдгээр аргуудын хэрэгжилтийг дэд ангиуд дотор нь гүйцээж бөглөдөг. Амьдрал дээр хийсвэр анги нь ядаж нэг аргыг бүрэн юмуу хэсэгчилсэн байдаг. Зөвхөн хийсвэр аргын мэдэгдлүүдээс бүрдсэн хийсвэр ангийг interface болгон тодорхойлох нь дээр. Interface-ийн тухай дэлгэрэнгүйг Interfaces and Packages –с харна уу.

Дотроо хийсвэр аргыг агуулсан хийсвэр ангийг хэрхэн байгуулах жишээг авч үзъе. Объект хандалттай зурах хэрэглэгдэхүүн дотор тойрог, дөрвөлжин, шугам, муруй зэргийг зурж болно. Эдгээр график объектууд нь нийтлэг төлөв(байрлал, хүрээ) болон араншин(шилжилт, хэмжээ өөрчлөх, зурах)-тай байдаг. Ийм нийтлэг төстэй байдлыг нь

ашиглан доорх зурагт агуулсанчлан бүх объектоо GraphicObject гэх объектоос удамшуулж болно.

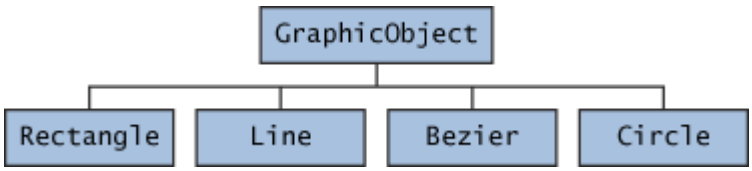


График объектод нь олон талаараа хоорондоо ялгаатай байх нь ойлгомжтой: тойрог зурах нь дөрвөлжин зурахаас ялгаатай гэх мэт. Иймэрхүү маягийн ялгаатай төлөв болон араншинг бүх график объектод нийтлэг маягаар ашиглаж болохгүй. Нөгөө талаас бүх график объектод нь өөрсдийгөө хэрхэн зурахаа мэддэг байх ёстой. Эдгээр нь зөвхөн зурах аргаар ялгаатай. Энэ бол хийсвэр дээд ангийг өчих гол үндэслэл болно.

Эхлээд бүх дэд ангиуддаа хэрэглэгдэх тохиосон байрлал moveto арга зэрэг гишүүн хувьсагч болон аргуудыг агуулсан GraphcObject гэсэн хийсвэр ангийг агуулна. Мөн түүнчлэн график объект нь бүх дэд ангиудад өөр өөр замаар хэрэгжүүлэх draw хийсвэр аргуудыг зарлана. График объект нь ойролцоогоор дараах хэлбэртэй байна.

```
abstract class GraphicObject {
    int x, y;
    ...
    void moveTo(int newX, int newY) {
        ...
    }
    abstract void draw();
}
```

Circle болон Rectangle гэх мэт GraphicObject-ийн хийсвэр бус дэд анги бүр нь draw аргыг хэрэгжүүлэх ёстой.

```
class Circle extends GraphicObject {
    void draw() {
        ...
    }
}
class Rectangle extends GraphicObject {
    void draw() {
        ...
    }
}
```

Хийсвэр анги дотор заавал хийсвэр арга байх хэрэггүй. Харин дотроо хийсвэр арга агуулсан эсвэл дээд ангидаа зарлагдсан хийсвэр аргыг хэрэгжүүлээгүй, эсвэл interface хэрэгжүүлээгүй ангыг хийсвэр анги болгон зарладаг.

Дүгнэлт

Тухайн Object ангиас бусад бүх анги зөвхөн ганц дээд ангитай байна. Анги нь өөрийн бүх дээд ангийг гишүүн үүсгэгч болон аргуудыг удамшуулж авдаг. Дэд анги нь удамшуулан авсан дээд анги болон аргаа далдалж болно. Дээд анги дотрох аргатайгаа адил signature-тай аргыг зарлахад ямар нөлөө үзүүлхийг Overriding and Hiding Methods доторх хүснэгтээс хараарай.

Object анги нь ангын шатлалын дээд анги юм. Бүх ангиуд нь энэ ангийн ураг бөгөөд түүний бүх аргыг удамшуулж авдаг. Object ангиас удамшдаг гол аргууд нь `toString`, `equals`, `clone`, `getClass`, `wait`, `notify`, `notifyAll` зэрэг болно.

Ангийн мэдэгдэл дотор final түлхүүр үг бичигдсэн байвал тухайн ангиас дэд анги үүсэхийг хориглож болно. Үүний нэг адилаар мөн арга тодорхойлох замаар тухайн аргыг дэд анги дотор дахин тодорхойлохоос сэргийлж болно. Хийсвэр ангийг төллүүлж болохгүй, зөвхөн дэд анги үүсгэж болно.

Хийсвэр анги нь хийсвэр аргууд, өөрөөр хэлбэл зарлагдсан хирнээ хэрэгжүүлээгүй аргууд оруулж болно. Хийсвэр аргуудыг дэд анги дотор нь хэрэгжүүлдэг.

Асуулт

Дараах 2 ангийг авч үзъе

```
public class ClassA {
    public void methodOne(int i) {
    }
    public void methodTwo(int i) {
    }
    public static void methodThree(int i) {
    }
    public static void methodFour(int i) {
    }
}

public class ClassB extends ClassA {
    public static void methodOne(int i) {
    }
    public void methodTwo(int i) {
    }
    public void methodThree(int i) {
    }
    public static void methodFour(int i) {
    }
}
```

a. Аль нэг арга нь дээд анги доторх аргаа дахин тодорхойлж байна вэ?
b. Аль нэг арга нь дээд анги доторх аргаа далдалж байна вэ?
c. Бусад аргууд нь юу хийж байна вэ?

2. Өмнөх дасгалуудад бичсэн `Card`, `Deck`,болон `DisplayDeck` ангиудаа авч үзъе. Object ангийн ямар аргуудыг бүх ангиуд дахин тодорхойлох ёстой вэ?

Дасгал

1. 2-р асуултанд хариулсан аргынхаа хэрэгжилтийг бич.
2. Хийсвэр анги бич. Ядаж 2 хийсвэр бус дэд ангийг бич.

Багтсан ангиуд

Нэг ангийг өөр нэг ангийн нэг гишүүн болгон зарлаж болно. Энэ ангийг багтсан анги гэнэ. Энэ жишээг доор харуулав.

```
class EnclosingClass {
    ...
    class ANestedClass {
```

```
        ...
    }
}
```

Хоёр ангийн хооронд зориудын холбоо ашиглахын тулд багтсан ангийг ашиглана. Багтсан ангийг багтаасан ангид байрлуулахаас өөр аргагүй үед л нэг ангийг нөгөө ангид байрлуулах хэрэгтэй.

Жишээ нь : Бичвэрийн курсорыг зөвхөн бичвэрийн компонент доор л тодорхойлж болно. Багтсан анги нь онцгой давуу эрх эдэлдэг. Ийм анги нь багтаасан ангийнхаа бүх гишүүд рүү тэр ч байтугай хаалттай гишүүнд ч хандах чадалтай байдаг. Гэхдээ ийм давуу эрх нь тийм ч онцгой зүйл биш юм. Энэ нь хаалттай гэдэгтэйгээ зөрчилдөхгүй. Хандалтын хувиргуур нь зөвхөн багтаасан ангийн гадна талын ангиудад зориулагдсан байдаг. Багтсан анги багтаасан ангийнхаа бүх гишүүн рүү хагдаж болно. Бусад гишүүдийн адил багтсан ангийг Static болгон зарлаж болно. Багтсан static ангийг зүгээр л багтсан static анги гэж нэрлэнэ. Харин static бус багтсан ангийг дотоод анги гэж нэрлэнэ.

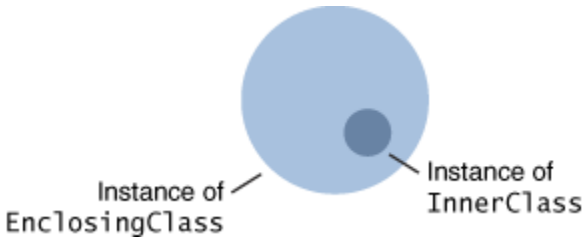
```
class EnclosingClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}
```

Анги арга болон анги хувьсагч хэмээн нэрэлдэг static арга болон static хувьсагчид нь нэгэн адил static багтсан анги нь багтаасан ангитайгаа холбогддог. Анги аргын нэгэн адил static багтсан анги нь багтаасан анги доторх төл хувьсагч болон арга руу шууд хандаж болохгүй харин заагчаар (обьект) нь дамжуулан харьцдаг. Төл арга болон төл хувьсагчийн нэгэн адил дотоод анги нь багтаасан ангийнхаа төлтэй холбогддог бөгөөд обьектынхоо төл хувьсагч болон аргуудтай шууд харьцаж болно. Дотоод анги нь төлтэй холбогддог учир дотроо static гишүүн агуулж болохгүй. Багтсан анги болон дотоод анги хоёрын ялгааг дараах байдлаар тайлыбарлаж болно. Багтсан анги нэр томъёо нь хоёр ангийн хоорондох цутгамал холбоог харуулдаг. Өөрөөр хэлбэл нэг ангийн код нь нөгөө ангийнхаа код дотор бичигдсэн байдаг. Нөгөө талаас дотоод анги гэдэг нь нэр томъёо, 2 ангийн төлүүдийн хоорондох холбоог харуулдаг. Дараах жишээг авч үзье.

```
class EnclosingClass {
    ...
    class InnerClass {
        ...
    }
}
```

InnerClass ангийг EncloseingClass –анги дотор тодорхойлсон нь энэ хоёр ангийн хоорондох харилцаа холбоо нь сонирхолтой зүйл биш юм. Харин InnerClass анги нь

зөвхөн EnclosingClass ангийн төл дотор орших чадвартай. Түүнчлэн багтаасан төлийнхөө төл хувьсагч, болон аргуудтай шууд харьцах чадвартай байдаг нь сонирхолтой юм. Энэ санааг дараах зургаар харуулав.



Java платформын сан дотор багтсан ангийн 2 хэлбэр хоёулаа тохиолддог. Гэхдээ багтсан ангийн хамгийн түгээмэл хэлбэр нь дотоод анги байдаг.

Дотоод ангиуд

Энэ анги ямар ач холбогдолтой вэ? гэдгийг харуулахын тулд Stack ангийг дахин авч үзье. Java.util.Iterator интерфейс ашиглан Stack доторх элементүүдийг тоочих бололцоог бусад ангиудад олгох тийм шинж чанар энэ ангид нэмэх шаардлагатай болсон гэж үзье. Энэ интерфейс нь дараах 3 аргын тодорхойлолтыг агуулна.

```
public boolean hasNext();
public Object next();
public void remove();
```

Iterator интерфейсийг нь эрэмблэгдсэн олонлог доторх элементүүдийг дэс дараалан унших бололцоог олгодог. Үүнийг дараах маягаар ашиглана.

```
while (hasNext()) {
    next();
}
```

Iterator интерфейс нь хэдэн сул талтай учраас Stack анги нь өөртөө iterator интерфейсийг хэрэгжүүлж болохгүй.

Жишээ нь: Iterator интерфейсийн зарим сул талыг дурьдвал :

1. Stack доторх гишүүн рүү 2 обьект зэрэг хандаж болохгүй, учир аль нь Next() аргыг дуудаж байгаа нь ойлгомжгүй болно.
2. Iterator интерфейс дотор эхлүүлэх арга тодорхойлогдоогүй учраас тоочилтыг дахин эхлүүлэх боломжгүй байдаг.
3. Iterator интерфейс дотор эхлэл рүүгээ буцах арга байдаггүй учраас тоочилтыг ганцхан удаа хэрэглэж болно.

Тиймээс туслах анги ашиглан энэ ажлыг хийх хэрэгтэй. Туслагч анги нь Stack-н гишүүд рүү хандаж боломжтой, хандахдаа шууд ханддаг байх ёстой. Учир нь Stack –ын нээлттэй интерфейс нь зөвхөн LIFO хандалтыг дэмждэг. Яг ийм нөхцөлд дотоод ангийг ашиглана. Stack-ын гишүүдийг тоочих зориулалттай StackIterator нэртэй туслагч анги тодорхойлсон Stack ангийн жишээг дор харууллаа.

```
public class Stack {
   private Object[] items;

   //code for Stack's methods and constructors
   not shown

   public Iterator iterator() {
      return new StackIterator();
   }
   class StackIterator implements Iterator {
      int currentItem = items.size() - 1;

      public boolean hasNext() {
         ...
      }
      public Object next() {
         ...
      }
      public void remove() {
         ...
      }
   }
}
```

StackIterator анги нь Stack-ын Items төл хувьсагч руу шууд хандаж байгааг анхаарна уу!

Дотоод ангийг гол төлөв энэ жишээнд харуулсанчлан туслагч ангийг бүтээхэд хэрэглэдэг. Үзэгдэл боловсруулах механизмыг маш өргөн хэрэглэдэг учир хэрэглэгчийн интерфейсийн үзэгдлийг боловсруулахаар шийдсэн бол дотоод ангийн тухай сайн судлах хэрэгтэй. Дотоод ангийг нэргүйгээр тодорхойлж болно. Үүнийг anonymous анги гэнэ.

Anonymous анги ашиглан iterator үүсгэсэн Stack ангийн өөр нэг жишээг дор харуулав.

```
public class Stack {
   private Object[] items;

   //code for Stack's methods and constructors
   not shown

   public Iterator iterator() {
      return new Iterator() {
         int currentItem = items.size() - 1;
         public boolean hasNext() {
            ...
         }
         public Object next() {
            ...
         }
         public void remove() {
            ...
```

```
            }
        }
    }
}
```

Ийм ангийг ашиглах нь кодыг ойлгоход илүү төвөгтэй болгодог. Маш цомхон анги (зөвхөн нэг эсвэл хоёр аргатай) эсвэл зориулалт нь тодорхой (үзэгдэл боловсруулах анги) ангиудын хувьд anonymous ангийг ашиглах хэрэгтэй.

Багтсан ангийн тухай бусад мэдэгдэл

Бусад ангиудын адил багтсан ангийг Abstract юмуу final болгон зарлаж болно. Дотоод ангийн хувьд энэ 2 хувиргуурын зориулалт нь бусад ангиудынхтай адил. Мөн түүнчлэн Private, Protected, Public хувиргууруудыг бусад ангитай адилхнаар ашиглаж болно.

Ямар ч багтсан ангийг ямар ч код блок дотор зарлаж болно. Арга юмуу код блок дотор тодорхойлсон багтаасан анги нь тухайн цараа доторх final болон дотоод хувьсагч руу хандаж болно.

Дүгнэлт

Өөр анги дотор тодорхойлсон ангийг багтсан анги гэнэ. Ангийн бусад гишүүдийн нэгэн адил багтсан анги нь Static юмуу Static бус байж болно. Static бус багтсан ангийг дотоод анги гэнэ. Дотоод ангийн төл нь зөвхөн багтаасан ангийнхаа төл дотор орших бөгөөд багтаасан ангийнхаа бүх гишүүд рүү үүний дотор хаалттай гишүүд рүү хандах чадвартай.

Асуулт болон дасгал

Асуулт

1. Эхний баганад өгөгдсөн тодорхойлолтыг ашиглан багтсан ангийн хамгийн тохиромжтой төрлийг 2-р баганаас ол.

a. Энэхүү багтсан ангийг хэрэглэгч нь багтаасан ангийн төл юмуу багтаасан ангийн дэд ангийн төл хувьсагч байна. b. Энэ багтсан ангийг хэн ч ашиглаж болно. c. Энэхүү багтсан ангийг зөвхөн зарласан ангийн төлүүд л ашиглах боломжтой бөгөөд тодорхой нэг төл нь үүнийг олон удаа ашиглаж болно. d. Энэхүү бичил багтсан ангийг зөвхөн нэг удаа хэрэглэх бөгөөд ямар нэг интерфейс хэрэгжүүлсэн обьектыг үүсгэхэд хэрэглэнэ. e. Энэхүү багтсан анги нь багтаасан ангийнхаа (багтаасан ангийнхаа төлийн тухай биш) мэдээллийг агуулах бөгөөд түүнийг зөвхөн багтаасан анги юмуу багтаасан ангийнх нь дэд ангиуд ашиглах боломжтой. f. Өмнөхтэй адил боловч дэд ангиуд нь ашиглахгүй.	1. Anonymous анги 2. Хязгаарлагдмал дотоод анги 3. Нээлттэй Static багтсан анги 4. Хязгаарлагдмал static багтсан анги 5. Хаалттай static багтсан анги 6. Хаалттай дотоод анги

2. Problem.java програмыг эмхтгэж болохгүй байна. Түүнийг эмхэтгэхийн тулд юу хийх ёстой вэ? Яагаад?

3. Вох ангийн танилцуулгыг ашиглан дараах асуултанд хариул

    a. Вох дотор ямар static багтсан анги тодорхойлсон байна вэ?
    b. Вох дотор ямар дотоод анги тодорхойлсон байна вэ?
    c. Вох ангийн дотоод ангийн дээд анги нь юу вэ?
    d. Вох ангийн багтсан ангиудаас алийг нь дуртай ангиасаа дуудаж болох вэ?
    e. Вох анги доторх Filler ангийн төлийг яаж үүсгэх вэ?

Дасгал

1. Эхлээд InnerClassDemo.java файлыг татаж ав.

    a. Уг програмаа эмхтгээд, гүйлгэ.

    b. InnerClassDemo-ын хуулбарыг үүсгэ. Дотор нь ActionListener интерфейсийг хэрэгжүүлсэн MyActionListener нэртэй дотоод ангийг нэмж оруул. ActionListener интерфейс дотор ганцхан арга тодорхойлсон байдаг. Энэ аргын хэрэгжилт дотор дараах кодыг бич.
    `quit();`
    Дараах кодын өмнөх 2 ташуу зураасыг арилга.

    ```
    //button.addActionListener(new MyActionListener());
    ```

    Одоо програмаа эмхтгээд, гүйлгэ. InnerClassDemoангийн эхний хувилбар болон сүүлийн хувилбарын хооронд ямар ялгаа байна вэ?

    c. 1b дасгалын ажлаар үүсгэсэн програмынхаа хуулбарыг хий. ActionListener хэрэгжилтээ anonymous дотоод анги болгон өөрчил. (Зөвлөмж : Энэ програм дотор тодорхойлсон WindowAdapter төрлийн өөр нэг anonymous дотоод ангийг санаа авахдаа ашиглаж болно.)

2. Class1.java –г татаж ав.

    a. Class1.java програмаа эмхтгээд, гүйлгэ. Гарц нь юу байх вэ?

    b. Class1 болон InnerClass1 гэсэн дотоод ангийнх нь дэд ангиудыг тодорхойлсон Class2.java нэртэй файл үүсгэ. (Эдгээр дэд ангиудаа Class2 болон InnerClass2 гэж нэрлэ)
    InnerClass2 – нь getAnotherString аргыг дахин тодорхойлохдоо “InnerClass2 version of getAnotherString invoked” бичвэрийг буцаах ёстой. Class2 – нь нэг байгуулагч, нэг арга тодорхойлох ёстой. Үүнд :

        - Хэмжигдэхүүнгүй байгуулагч нь удамшсан іс төл хувьсагчийг InnerClass2-ын төл болгон цэнэглэнэ.
        - Main арга нь Class2-ын төл үүсгээд, үүссэн төлийнхөө DisplayStrings аргыг дуудаж байна.

    Class2- програмыг гүйлгэхэд гарц нь юу байх вэ?

Тоочих төрлүүд

Тодорхой тооны тогтмолуудаас утга нь бүрддэг төрлийг тоочих төрөл гэнэ. Нийтлэг жишээнд хойд, өмнө, өрнө, дорно гэсэн утгуудаас бүрдэх compass- ын чиглэлүүд эсвэл даваа, мягмар, лхагва, пүрэв, баасан, бямба, ням зэргээс бүрддэг долоо хоногийн өдрүүд байдаг.

Java хэлэнд тоочих төрлийг тодорхойлохдоо enum түлхүүр үгийг ашиглана.

Жишээ нь : 7 хоногийн өдрүүдийг тоочсон төрлийг дараах маягаар зарлана.

```
enum Days { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
           FRIDAY, SATURDAY };
```

Тоочих төрлүүдийн утгуудыг бичихдээ том үсгээр бичдэг. Ямар нэг тодорхой тооны тогтмолыг дүрслэх хэрэгцээ гарах үед тоочих төрлийг хэрэглэх хэрэгтэй.

Манай нарны аймгийн системийн гараг, долоо хоногийн өдрүүд, хөзрийн моднууд зэрэг төгсгөлөг (тодорхой тоотой) юмс түүнчлэн цэсний сонголтууд, ойролцоо тоо руу хөрвүүлэх, тушаалын мөрийн дарцагнууд гэх мэт. Эмхтгэх үед бүх утгууд нь тодорхой байдаг олонлог зэргийг энд жишээ болгон авч үзэж байна. Бусад хэлний тоочих төрөл нь зүсээ хувиргасан бүхэл тоо байдгаас Java хэлнийх ялгаатай байдгаараа хүчтэй. Enum мэдэгдэл нь анги тодорхойлдог (Enum төрөл гэж нэрлэгдэх).

Хамгийн гол шинж чанарууд нь:

- Хэвлэгдэх утгууд нь илүү дэлгэрэнгүй
- Төрлийн зөрчилгүй
- Өөрийнхөө нэрийн орчинд оршдог.
- Тогтмолуудын олонлогийг нь цаг ямагт хувиршгүй байх албагүй.
- Тоочих тогтмол руу шилжих боломжтой.
- Бичиглэсэн дарааллынх нь дагуу тоочих төрлийн бүх утгыг агуулсан бүл буцаах values static арга агуулдаг. Энэ аргыг тоочих төрлийн бүх утгыг уншихад зориулагдсан for-each хэллэг дотор хамт ашиглах нь элбэг байдаг.
- Арга болон талбар нэмэх, интерфейс хэрэгжүүлэх гэх мэтчилэн сайжруулж болно.
- Object ангийн бүх аргуудыг хэрэгжүүлдэг. Тэдгээр нь харьцуулах болон сериалжуулах боломжтой.

Нарны аймгийн бүх гарагийг тусгасан Planet хэмээх тоочих төрлийг доор жишээ болгон харууллаа. Planet анги нь mass болон radius гэсэн 2 тогтмол шинж чанартай. Тогтмол бүр нь байгуулагдахдаа байгуулагч руу нь дамжуулан mass болон radius –ын хэмжигдэхүүний утгаар цэнэглэгдэнэ. Тоочих төрлийн байгуулагч нь хаалттай байгааг анхаарна уу? Хэрэв үүнийг нээлттэй болговол эмхтгүүрээс алдааны мэдээлэл өгнө.

```
public enum Planet {
   MERCURY (3.303e+23, 2.4397e6),
   VENUS   (4.869e+24, 6.0518e6),
   EARTH   (5.976e+24, 6.37814e6),
   MARS    (6.421e+23, 3.3972e6),
   JUPITER (1.9e+27,   7.1492e7),
   SATURN  (5.688e+26, 6.0268e7),
```

```
    URANUS  (8.686e+25, 2.5559e7),
    NEPTUNE (1.024e+26, 2.4746e7),
    PLUTO   (1.27e+22,  1.137e6);

    private final double mass;   //in kilograms
    private final double radius; //in meters
    Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
    }
    public double mass()   { return mass; }
    public double radius() { return radius; }

    //universal gravitational constant  (m3 kg-1 s-2)
    public static final double G = 6.67300E-11;

    public double surfaceGravity() {
        return G * mass / (radius * radius);
    }
    public double surfaceWeight(double otherMass) {
        return otherMass * surfaceGravity();
    }
}
```

Дурьдсан 2 шинж чанараас гадна Planet анги нь гараг бүр дээрх объектын гадаргуугийн таталцал болон жинг тооцох аргуудыг агуулж байна. Дэлхий дээрхи таны жинг аваад бусад бүх гараг дээр таны жинг тооцож хэвлэх програмын жишээг дор харуулав.

```
public static void main(String[] args) {
    double earthWeight = Double.parseDouble(args[0]);
    double mass = earthWeight/EARTH.surfaceGravity();
    for (Planet p : Planet.values()) {
        System.out.printf("Your weight on %s is %f%n",
                          p, p.surfaceWeight(mass));
    }
}
```

Энэ програмын гарц нь:

```
$ java Planet 175
Your weight on MERCURY is 66.107583
Your weight on VENUS is 158.374842
Your weight on EARTH is 175.000000
Your weight on MARS is 66.279007
Your weight on JUPITER is 442.847567
Your weight on SATURN is 186.552719
Your weight on URANUS is 158.397260
Your weight on NEPTUNE is 199.207413
Your weight on PLUTO is 11.703031
```

Тоочих төрлийн нэг сул тал бол хэдийгээр тоочих төрөл нь анги мөн боловч тоочих төрлийн шатлал үүсгэж болохгүй. Өөрөөр хэлбэл, нэг тоочих төрлийг өөр нэг тоочих төрөл удамшуулан өргөжүүлж болохгүй. Эцэст нь java.util багц дотор EnumSet,

EnumMap гэсэн тоочих төрлийг хэрэгжүүлсэн тусгай зориулалтын Set болон Map байдаг.

Асуулт болон дасгал

1. “Асуултууд болон анги үүсгэх дасгалууд” хичээлд үзсэн card ангийг өөрчлөн хөзрийн моднууд ба тоочих төрлөөр илэрхийлдэг болгон өөрчил.

2. Deck ангийг бас өөрчлөн бич.

Ерөнхий төрлүүд

Анги юмуу интерфэйсийг нэг удаа тодорхойлоод тэдгээрээ олон янзын төлөөр төллүүлэх боломжийг ерөнхий төрлүүд олгодог. Ерөнхий төл нь (заримдаа хэмжигдэхүүнтэй төл гэж нэрлэдэг) нэг болон түүнээс дээш тооны төрлийн хэмжигдэхүүнтэй байдаг. Ерөнхий төлийг ашиглахийн тулд төлийн хэмжигдэхүүн бүрт түүнд харгалзах жинхэнэ төлийг зааж өгөх шаардлагатай байдаг бөгөөд ингэснээр тухайн ерөнхий төл нь тодорхой явцуу төрөлтэй ажиллах боломжтой болдог.

Ерөнхий төрлийг тодорхойлох болон ашиглах

Ерөнхий төрлийг тодорхойлохдоо төрлийнх нь нэрийн ард төрлийн хэмжигдэхүүнүүдийг бичиж өгдөг. Төрлийн хэмжигдэхүүн гэдэг нь <””> гурвалжин хаалтан доторх нэрсийн таслалаар зааглагдсан жагсаалт байдаг. Төрлийн хэмжигдэхүүнүүдийг том үсгээр бичих журам байдаг. Төрлийн хэмжигдэхүүнүүдийг тухайн төрлийн аргууд дотор, нэг бол аргын хэмжигдэхүүний жагсаалт доторх аргументын төрөл маягаар эсвэл буцах утгынх нь төрөл маягаар тус тус ашиглаж болно.

Ангийн нэрийн ард бичигдсэн <T> гэдэг төрлийн хэмжигдэхүүнийг pushаргийн аргументын төрөл рор ангийн буцах төрөл болгон хэрхэн ашигласныг доорх жишээнээс харна уу!

```
public class Stack2<T> {
   private ArrayList<T> items;
   ...
   public void push(T item) {}
   public T pop() {}
   public boolean isEmpty() {}
}
```

Ерөнхий төрлийг хамгийн өргөн ашигладаг анги нь цуглуулгууд байдаг. Үнэн хэлэндээ ерөнхий төрлүүдийг Java хэлэнд нэвтрүүлэх гол үндэслэл нь цуглуулгууд байдаг, учир нь цуглуулга дээр хийх үйлдлүүдийн төрлийн аюулгүй байдлыг эмхэтгэх үед шалгах бололцоог ерөнхий төрлүүд олгодог. Цуглуулга дотор хадгалагдах объектын төрлийг зааж өгдөг болсноор:

- Цуглуулга руу объект нэмэх дурын үйлдлийг эмхэтгүүрээр хянах боломжтой болсон
- Цуглуулгаас гарган авах объектын төрөл нь урьдчилан мэдэгдэж байгаа учраас төрөл хувиргах үйлдэл хийх шаардлагагүй болсон. Тиймээс төлөө буруу хувиргаж гүйлгэх үеийн алдаа гарах эрсдэлээс ангижирсан.

Өмнө нь хэлсэнчлэн ерөнхий төрлийг ашиглахдаа аргын хэмжигдэхүүнүүдийг жинхэнэ утгаар орлуулдагийн нэгэн адил төрлийн хэмжигдэхүүний оронд жинхэнэ төрлийг бичиж өгдөг. Жинхэнэ төрөл нь эгэл төрөл байж болохгүй, заавал заагч байх ёстой.

Жишээлбэл: String төрлийн аргументтайгаар stack2 ангийг төллүүлэн, “hi” хэлхээг түлхэн оруулах (push) болон татаж авах (pop) жишээг доор харууллаа.

```
Stack2<String> s = new Stack2<String>();
s.push("hi");
String greeting = s.pop(); //no cast required here
```

Ерөнхий төрөл биш стектэй гэвэл:

```
Stack2 s = new Stack2();
s.push("hi");
String greeting = (String)s.pop(); //cast required here
```

Ерөнхий төрөл ашиглаагүй бол энэ жишээ дараах хэлбэртэй байх байсан. Ерөнхий төрлийг төллүүлэх үед уг ерөнхий төрлийг арилгах замаар, товчоор хэлэхэд төрлийн хэмжигдэхүүн болон жинхэнэ төлтэй холбоотой бүх мэдээллийг эмхэтгүүр арилгасан байдаг. Тухайлбал, stack2 <String> нь stack2 төл болон хувирдаг. Үүссэн төлийг нь ердийн төл гэнэ. Төл арилгасны үр нөлөө нь гэвэл гүйлгэх үед жинхэнэ төлийг нь төл хөрвүүлэхэд ашиглах боломжгүй юмуу эсвэл instanceof аргийг хэрэглэх боломжгүй болдог юм.

Ерөнхий төрлүүдийн хоорондох харилцан холбоо

Объект нь String ангийн дээд анги учраас stack2<object> нь stack2<String> ангийн дээд анги болно хэмээн таамаглаж болох юм. Үнэн хэрэгтээ ерөнхий төрлийн төлөвүүдийн хооронд ийм хамаарал байдаггүй. Ерөнхий төрөлийн төлүүдийн хооронд дээд-дэд ангийн холбоо байдаггүй нь ондооссон аргуудыг бичихэд хүндрэл учруулдаг.

Цуглуулга дотор агуулагдах объектуудын төрлөөс үл хамааруулан объектуудын цуглуулгыг хэвлэн харуулах арга бичих хэрэгтэй болдог:

```
public void printAll(Collection<Object> c) {
    for (Object o : c) {
        System.out.println(o);
    }
}
```

Хэлхээний жагсаалт үүсгэн бүх хэлхээнүүдээ хэвлэхэд дээрх аргаа ашиглах шаардлагатай боллоо гэж үзье:

```
List<String> list = new ArrayList<String>();
...
printall(list);   //error
```

Хэрэв энэ кодыг ашиглахаар оролдвол хамгийн сүүлийн хэллэг дээр эмхэтгэлийн алдаа гарна. Arraylist<String> нь collection<object> -ийн дээд төрөл биш учраас хэдийгээр энэ хоёр төрөл нь удамшлынхаа шатлалаар холбогдсон 2 өөр жинхэнэ төлтэй. Нэг ерөнхий төрлийн төлүүд мөн боловч Arraylist<String> гэсэн төлийг хэвлэх аргын хэмжигдэхүүн

болгон дамжуулж болохгүй. Нөгөө талаас, төлийн хэмжигдэхүүн нь адилхан, удамшлынхаа шатлалаар холбогдсон ерөнхий төрлүүдийн төлүүд нь хоорондоо зохицдог.

```
public void printAll(Collection<Object> c) {
    for (Object o : c) {
        System.out.println(o);
    }
}
List<Object> list = new ArrayList<Object>();
...
printall(list);   //this works
```

list<object> нь collection<object> -доо зохицно. Учир нь хоёулаа <object> гэсэн ижил төрлийн хэмжигдэхүүнтэй. Ерөнхий төрлүүдийнх нь хооронд дээд ба дэд анги гэсэн холбоо оршин байдаг.

Wildcard төрлүүд

Өмнө нь үзсэн PrintAll аргийн нь дурын төрлийн гишүүдтэй цуглуулга авдаг байхаар өөрчлөхийг хүсвэл collection<?> бичлэгийг ашиглана.

```
public void printAll(Collection<?> c) {
    for (Object o : c) {
        System.out.println(o);
    }
}
```

<?> төрлийг wildcard төрөл гэнэ. Буцаах төрөл ямагт объект байдаг тул ийм цуглуулгаас объект авахад хүндрэл учирдаггүй. Гэхдээ ийм цуглуулга руу шинээр объект нэмж болдоггүй. Учир нь ? тэмдэг нь тодорхойгүй төрлийг заадаг бөгөөд таны нэмэх объектын төрөл нь тодорхойгүй төрлийн дэд төрөл болж чадна гэдгийг урьдчилан мэдэх боломжгүй. Зөвхөн null утга бүх төрлийн утга болж чаддаг. Wildcard төрлийг төрлийн хязгаарлалт тавьж болдог. Төрлийн аргументын талаар гүйцэд мэдээлэлгүй үед хязгаарлагдмал wildcard төрлийг ашиглахад тохиромжтой байдаг.

Жишээлбэл: Ангийн шатлал нь геометр дүрс болон (shape) түүний дэд төрлүүдээс (дугуй, дөрвөлжин г.м) бүрддэг байг. Тэдгээр объектыг заасан зурах программ drawAll аргыг дуудан тэдгээр дүрсүүдээр цуглуулгыг дуудан зурах ёстой болог: drawAll аргаар дамжуулж буй ерөнхий цуглуулгын төрлийн аргумент нь shape:

```
public void drawAll(Collection<Shapes> shapes) {
    for (Shape s: shapes) {
        s.draw();
    }
}
```

Ангийн дэд төрөл (жишээ нь: дугуй) байж болохгүй тул энэ аргийг бүрэн ашиглахад бэрхшээл учирч байна.

Энэ нь drawAll –руу дамжуулсан ерөнхий өгөгдлийн төрлийн цуглуулга дахь төрөл аргумент мэтээр Shape –н дэд ангид (жишээ нь Circle -д) дамжуулах нь зүй ёсны бус юм. Энэ арга нь хэрэгцээтэй байх нөхцөлийг хязгаарласан: жишээ нь Collection<Circle>

-тэй дуудагдаж чадахгүй. Shape –н дэд ангийг төрөл аргумент мэтээр дамжуулах боломжтой болгохын тулд, дүрс хэлбэрийн цуглуулгын төрөл параметрийг wildcard мэтээр мэдэгдэж болно. Хэдийгээр тийм ч, төрөл аргумент нь хэлбэрийн ямар нэгэн төрөл гэдгийг мэдсэнээс хойш, Shape өвөг ангиар дараах шиг зааглагдах нь дээр.

```
void drawAll(Collection<? extends Shapes> shapes) { ... }
```

Энэ нь drawAll –д Shape –н ямарч дэд ангийн цуглуулгуудыг хүлээн авахад нь боломж олгоно.

Дүгнэхэд, дээд хүрээтэй wildcard нь `<? extends Type>` мэтээр заагддаг бөгөөд *type* –н дэд төрөл болох бүх төрлүүдийг төлөөлдөг. Мөн энэ нь доод хүрээтэй wildcard –г шаардах боломжтой. Доод хүрээтэй wildcard нь `<? super Type>` мэтээр заагддаг бөгөөд *type* –н өвөг төрөл болох бүх төрлүүдийг төлөөлдөг. Обьектыг үл мэдэгдэх төрлийн цуглуулга руу нэмэх нь боломжгүй гэдгийг санаарай. Мөн обьектыг хүрээтэй үл мэдэгдэх төрлийн цуглуулга руу нэмэх нь зүй ёсны бус шүү.

Ерөнхий төрлийн аргуудыг тодорхойлох болон ашиглах

Төрлүүдээс гадна аргууд нь бас хувьсагч хэрэглэдэг. Байгуулагч шиг сайн статик болон статик бус аргууд нь төрөл параметртэй байж болно.

Аргын төрөл параметрүүдийг зарлах өгүүлбэр зүй нь ерөнхий төрөл зарладаг шиг адтлхан. Төрөл параметр нь өнцөгт хаалт [] –аар хязгаарлагдах бөгөөд аргын буцах төрлийн өмнө нь бий болдог. Жишээ нь: дараах `Collections` арга нь `<? super T>` төрлийн `List` –г `T` төрлийн обьектуудаар дүүргэж байна.

```
static <T> void fill(List<? super T> list, T obj)
```

Ерөнхий төрлийн аргууд нь арга дахь нэг эсвэл олон аргументуудын төрлийн дунд эсвэл түүний буцах төрлийн (эсвэл хоёулаа) дунд хараат бус байдлыг илэрхийлэхийн тулд танд төрөл параметруудыг ашиглахыг зөвшөөрнө. Ерөнхий төрлийн аргуудын төрөл парамертууд нь ерөнхийдөө ямар ч ангийн болон интерфейс түвшний төрөл параметруудаас харьяат бус байдаг. Collections ангиар тодорхойлогдсон алгоритмууд (Algorithms бүлэгт тайлбарласан байгаа) нь ерөнхий төрлийн аргуудын олон элбэг хэрэглээг бүтээдэг.

Ерөнхий төрлийн аргууд болон ерөнхий төрлүүдийн нэг ялгаа нь гэвэл ерөнхий төрлийн аргууд нь ердийн аргууд шиг дуудагддаг. Төрөл параметруудыг дүүрэн аргын дуудлага доторх шиг дуудлагын сэдвээс бодож гаргасан.

```
public static void main(String[] args) {
    List<String> list = new ArrayList<String>(10);
    for (int i = 0; i < 10; i++) {
        list.add("");
    }
    String filler = args[0];
    Collections.fill(list, filler);
    ...
}
```

Ерөнхий төрлүүдийг уламжлал кодтой ашиглах

Энэ бүлгийн эхэнд яригдсан Cat анги нь Cat обьектуудын цуглуулгыг буцаадаг getLitter хэмээх уламжлал аргатай.

```
public Collection getLitter(int size) {
    ArrayList litter = new ArrayList(size);
    for (int i = 0; i < size; i++) {
        litter.add(i, new Cat());
        return litter;
    }
}
```

Энэ объект нь анхдагч төрөл; Цуглуулгад багтсан объектийн төрлүүд нь тодорхойлогдоогүй байна. Ийм нөхцөл байдал generics үүсэхээс өмнө бүх аргуудад тохиолддог байсан. Cat гэдэг програм шинээр бичихэд цуглуулгаас аль ч төрлийн litter –р дуудахад энэ нь cat –аар тодорхойлогдоно.

```
public static void main(String[] args) {
Collection<Cat> litter = myCat.getLitter(3);
for (Cat c : litter) {
    System.out.println(c.getColor());
}
```

Дээрх Cat.java програмыг эмхэтгэхэд танд доорх анхааруулгыг өгнө:

```
Note: Cats.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

Анхааруулга: Cats.java эмхэтгэхэд аюулгүй бус (unchecked) баталгаатай ажиллагаа хангагдаагүй (unsafe).
Анхааруулга: Xlint-тэй дахин эмхэтгэх: дэлгэрүүлж шалгаж чадсангүй.

Энэ мэдээллийг хангасан Xlint:checked –г ашиглах нь:

```
% javac -Xlint:unchecked Cats.java
Cats.java:5: warning: [unchecked] unchecked conversion
found   : java.util.Collection
required: java.util.Collection<Cat>
             Collection<Cat> litter = myCat.getLitter(3);
                                                      ^
```

Нэмж хэлэхэд, хэрэв Cat нь ерөнхий төрлүүдийг дэмждэг эмхэтгэгчээр дахин эмхэтгэгдсэн бол үз дүнд нь дараах алдаа гарна:

```
% javac -Xlint:unchecked Cat.java
Cat.java:19: warning: [unchecked] unchecked call to
add(int,E) as a member of the raw type java.util.ArrayList
                           litter.add(i, new Cat());
                                      ^
```

Хэдийгээр код зөв боловч, эмхэтгэх үйлдэл нь баталгаатай зөв байна гэсэн баталгаа байхгүй, энэ аргаар тодорхойлогдсон цуглуулгын аль ч төрлүүдийг ашиглаж чадна гэсэн анхааруулгийг харуулна. Хэдийд ч та unchecked гэсэн анхааруулгийг авч болох юм, таньд өгч буй үйлдлийн анхааруулга нь зөв эсэхийг магадлаж шалгаж байх ёстой.

Эцэст нь, Stack2 ангийн бүрэн бичлэгийг харъя:

```
public class Stack2<T> implements Cloneable {
    private ArrayList<T> items;
    private int top=0;

    public Stack2() {
        items = new ArrayList<T>();
    }
    public void push(T item) {
        items.add(item);
        top++;
    }

    public T pop() {
        if (items.size() == 0)
        throw new EmptyStackException();
        T obj = items.get(--top);
        return obj;
    }

    public boolean isEmpty() {
        if (items.size() == 0)
            return true;
        else
            return false;
    }

    protected Stack2<T> clone() {
        try {
            Stack2<T> s = (Stack2<T>)super.clone();
            // Clone the stack
            s.items =  (ArrayList<T>)items.clone();
            // Clone the list
            return s; // Return the clone
        } catch (CloneNotSupportedException e) {
            //This shouldn't happen because Stack is Cloneable
            throw new InternalError();
        }
    }
}
```

Энэ уламжлалын аргаар дуудаж буй уламжлуулах аргууд нь дээд анги ба жагсаалтад багтсан байдаг. clone аргууд нь уламжлал аргууд юм учир нь тэд ерөнхий аргуудыг тодорхойлж бэлэн болгосон явдал юм.

Та stack2 –г эмхэтгэхэд дараах анхааруулгийг авна:

```
% javac -Xlint:unchecked Stack2.java
Stack2.java:32: warning: [unchecked] unchecked cast
found   : java.lang.Object
required: Stack2<T>
        Stack2<T> s = (Stack2<T>)super.clone(); //clone the stack
```

```
                                        ^
Stack2.java:33: warning: [unchecked] unchecked cast
found   : java.lang.Object
required: java.util.ArrayList<T>
        s.items =  (ArrayList<T>)items.clone(); //clone the list
                                        ^
2 warnings
```

Эдгээр анхааруулгууд нь эмхэтгэх үйлдэл нь зөв алдаагүй гэсэн баталгаагүйг харуулж байна. Нэг үгээр хэлвэл удашсан аргууд нь объектийн төрлөөрөө объект нь дуудагдаж тодорхойлогддог, эмхэтгүүр нь цуглуулгаас дуудагдсан Stack2<T> –г шалгаж чаддаггүй. Хэдий тийм боловч удамшлын аргын үүрэг хүлээдэг, дамжуулалт нь зөв, эдгээр нь зөв боловч энэ нь unchecked баталгаагүй гэсэн анхааруулгыг үүсгэж байдаг.

Эдгээр нь ерөнхий төрөл ба удамшлын кодтой ажиллахад маш их нарын учиртай байдаг.

Интерфейсүүд болон багцууд

Энэ бүлгээр бид 2 чухал Java програмын хэлний анги шатлалаар холбогдоогүй анги хоорондын холбоог удирдахад туслах болно. Эхлээд та бичиж сурах болон обьектуудын хоорондын холбооны протоколын интерфейсийг хэрэглэх болно. Хоёрт, та ангиуд болон интерфейсүүдийг багц руу яаж нэгтгэхийг сурах болно.

Интерфейсүүдийг үүсгэх болон ашиглах

What Is an Interface? хэсгээс интерфейсүүдийн танилцуулгыг хараарай. Энэ хэсэг нь интерфейсүүдийг үүсгэх болон ашиглахыг илүү харуулна, мөн ангийн оронд яагаад интерфейсийг хэрэглэх ёстойг хэлж өгнө.

Интерфейс нь анги шатлалын хаана нэгтээ байгаа ямар ч ангийг хэрэгжүүлэх араншингийн протоколыг тодорхойлно. Интерфейс нь биегүй аргыг тодорхойлно. Интерфейсд тодорхойлогдсон бүх аргуудыг хэрэгжүүлэх интерфейсийн зөвшөөрлийг хэрэгжүүлдэг анги байна.

Тодорхойлолт: Интерфейс бол аргын тодорхойлолтуудын цуглуулгаар нэрлэгддэг (хэрэгжүүлэлтгүйгээр).

Яагаад гэвэл интерфейс нь хэрэгжүүлэлтгүй энгийн жагсаалт мөн түүнчлэн хийсвэр, аргууд билээ. Та интерфейс нь хийсвэр ангиас ялгаатай байгаад гайхаж магадгүй юм. Ялгаанууд нь олон талын ач холбогдолтой.

- Интерфейс нь хийсвэр ангийн чадах ямар ч аргуудыг хэрэгжүүлэхгүй.
- Анги нь олон интерфейсүүдийг хэрэгжүүлж чадна, харин зөвхөн ганц өвөг ангитай байдаг.
- Интерфейс нь анги шатлалын хэсэг нь биш юм. Холбогдоогүй ангиуд нь ижил интерфейсүүдийг хэрэгжүүлж чадна.

Энэ хэсэгт ашиглах жишээг үзье. Хөрөнгийн үнийн мэдээг хардаг анги бичлээ гэж бодьё. Энэ анги нь хөрөнгийн утга өөрчлөгдөх үед бусад ангиудыг бүртгүүлж зарлахыг зөвшөөрдөг. Эхлээд таны анги StockApplet –г дуудна, арга нь бусад обьектуудыг бүртгүүлж зарласнаар хэрэгжүүлнэ.

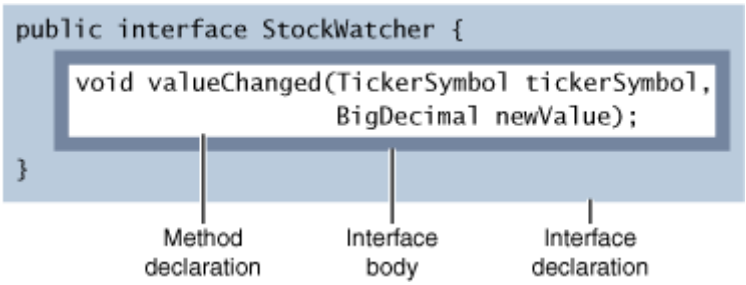
```
public class StockMonitor {
    public void watchStock(StockWatcher watcher,
                        TickerSymbol tickerSymbol,
                        BigDecimal delta) {
    ...
    }
}
```

Уг аргын эхний аргумент нь StockWatcher обьект юм. StockWatcher нь дараагийн бүлэгт харах кодын интерфейсийн нэр юм. Интерфейс нь valueChanged гэх нэг аргыг зарлаж байна. Зарлагдах гэж буй обьект нь уг интерфейсийг хэрэгжүүлэх ангийн төл байх ёстой ба ингэснээрээ valueChanged аргыг хэрэгжүүлнэ. Бусад хоёр аргумент нь хөрөнгийн үнийн тэмдэгтийг болон харагчийн хангалттай гэж авч үзэх өөрчлөлтийн хэмжээг хангаж өгнө. StockMonitor анги нь өөрчлөлтийг илрүүлэх үед, харагчийн valueChanged аргыг дуудна.

Бүх бүртгэгдсэн обьектууд valueChanged аргыг хэргжүүлэхийг warchStock арга нь өөрийн эхний аргументын өгөгдлийн төрлөөр баталгаажуулна. Бүртгэгдсэнүүд нь тодорхой аргыг хэрэгжүүлж байгаа нь зөвхөн чухал учираас интерфейсийн өгөгдлийн төрлийг эхэд хэрэглэх нь зохистой. Хэрэв StockMonitor нь ангийн нэрийг өгөгдлийн төрөл шиг хэрэглэсэн бол, хэрэглэгчүүд дээрх ангийн харилцаа холбоог хиймлээр албадах (хүчлэх) байсан. Анги нь зөвхөн ганц өвөг ангитай учраас, энэ нь обьектуудын ямар төрөл уг үйлчилгээг ашиглаж чадахыг хязгаарлах байсан. Анги шатлалын хаа ч байгаа ямар ч анги нь уг үйлчилгээг ашиглахыг зөвшөөрснөөрөө, интерфейс хэрэглэн, бүртгэгдсэн обьектуудын анги нь төлд зориулсан ямар нэгэн – Applet эсвэл Thread – байж болно.

Интерфейсийг тодорхойлох

Доорх зурагт интерфейсийн тодорхойлолтонд: зарлалт болон их бие гэсэн хоёр компонент байдгийг харуулж байна. Интерфейсийн зарлалтанд (интерфейсийн нэр болон бусад интерфейсүүдээр өргөтгөх эсэх зэрэг) интерфейсийн тухай олон аттрибутуудыг зарладаг. Интерфейсийн их бие нь тухайн интерфейст зориулсан тогтмол болон аргын зарлалтуудыг агуулдаг.



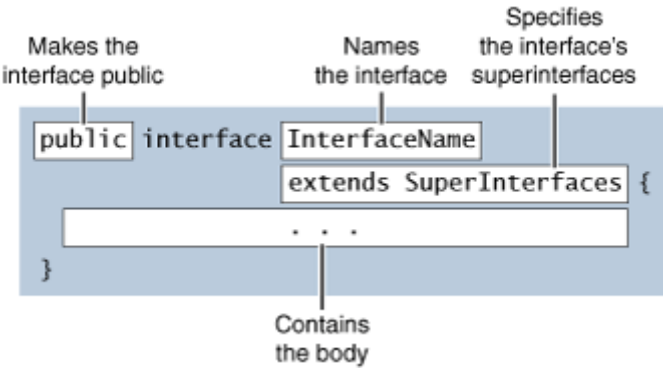
Интерфейс тодорхойлолтын бүтэц болон StockWatcher интерфейс.

```
public interface StockWatcher {
    void valueChanged(TickerSymbol tickerSymbol,
                      BigDecimal newValue);
}
```

Дээрх зурагт харуулсан интерфейс нь өмнө дурьдсан StockWatcher интерфейс юм. Энэ интерфейс нь valueChanged аргыг зарлаж байна гэхдээ хэрэгжүүлээгүй. Уг интерфейсийг хэрэгжүүлдэг ангиуд нь тухайн аргадаа зориулсан хэрэгжүүлэлтийг хангаж өгсөн.

Интерфейсийн зарлалт

Дараах зурагт интерфейсийн зарлалтын бүх боломжит компонентуудыг харууллаа.



Интерфейс зарлалтын боломжит компонентууд болон тэдгээрийн зорилго

Интерфейсийн зарлалтанд interface гэсэн түлхүүр үг болон интерфейсийн нэр хоёр заавал шаардлагатай байдаг. Хандалтын хувиргуур public нь ямар ч багц доторх ямар ч ангиар хэрэглэгдэнэ гэдгийг тогтооно. Хэрэв интерфейсээ public гэж заагаагүй тохиолдолд, тухайн интерфейсийн багц дотор тодорхойлогдсон ангиудаас хандах боломжтой болно.

Интерфейс зарлалтанд өвөг интерфейсүүдийн жагсаалт байж болно. Ангийг өргөтгөж болдог шиг интерфейсийг бусад интерфейсээр өргөтгөж болдог. Хэдийгээр ангийг өөр ганц ангиар өргөтгөдөг ч гэсэн, интерфейсийг хэдэн ч интерфейсээр өргөтгөж болдог. Өвөг интерфейсүүдийн жагсаалт нь өргөтгөгдөж буй бүх интерфейсүүдийн хооронд таслалаар “,” заагласан байна.

Интерфейсийн их бие

Интерфейсийн биед интерфейс дотор багтаасан бүх аргуудын зарлалтуудыг агуулдаг. Интерфейс нь дотроо зарласан аргуудад хэрэгжүүлэлтийг хангаагүй учираас интерфейс

доторх аргын зарлалт нь цэгтэй таслалтай (;) байдаг. Интерфейс дотор зарлагдсан бүх аргууд нь далдуур public болон abstratc байна.

Интерфейс дотор аргын зарлалтын дээр нь тогтмолын зарлалт байдаг. Интерфейс дотор тодорхойлогдсон бүх тогтмол утгууд нь далдуур public, static, болон final байна.

Интерфейс доторх гишүүн зарлалт нь зарим зарлалт өөрчлөгчийн хэрэглээг хориглодог. Интерфейс дотор гишүүн зарлалтанд transient, volatine, болон synchronized ашиглаж болохгүй. Мөн интерфейсийн гишүүдийг зарлах үед private болон protected хувиргуурыг хэрэглэж болохгүй.

Санамж: Java платформын өмнөх хувилбар нь интерфейс зарлалтанд болон интерфейс доторх аргын зарлалтанд abstract хувиргуур хэрэглэхийг зөвшөөрдөг байсан. Гэхдээ энэ нь шаардлагагүй, яагаад гэвэл интерфейсүүд болон тэдгээрийн аргууд нь далдуур abstract (хиймэл) байдаг. Интерфейс зарлалтанд болон интерфейс доторх аргын зарлалтанд abstract –г хэрэглээгүй нь дээр шүү.

Интерфейсийг хэрэгжүүлэх

Интерфейс нь араншингийн протоколыг тодорхойлдог. Интерфейсийг хэрэгжүүлж буй анги нь тухайн интерфейсээр тодорхойлогдож буй протокол руу ADHERE. Интерфейсийг хэрэгжүүлэх ангийг зарлахдаа ангийнхаа зарлалт дотор implements хэсгийг багтаана. Анги нь нэгээс олон интерфейсийг хэрэгжүүлж чаддаг (Java платформ нь интерфейсийн нийлмэл удамшлыг дэмждэг). Иимээс implement түлхүүр үг нь тухайн ангиар хэрэгжүүлэх интерфейсүүдийн жагсаалтыг таслалаар (,) тусгаарласан ардаа бичдэг.

Тохиролцоо: implement утгат хэсэг нь extends утгат хэсгийн ард ордог.

Доорх хэрэглүүрийн жишээний хэсэг нь StockWatcher интерфейсийг хэрэгжүүлж байна.

```
public class StockApplet extends Applet
                    implements StockWatcher {

   public void valueChanged(TickerSymbol tickerSymbol,
                       BigDecimal newValue) {
      switch (tickerSymbol) {
         case SUNW:
            ...
            break;
         case ORCL:
            ...
            break;
         case CSCO:
            ...
            break;
         default:
            // handle unknown stocks
            ...
            break;
      }
```

```
    }
}
```

Анги нь интерфейсийг хэрэгжүүлэх үед, энэ нь гол нь гэрээнд гарын үсэг зурж байна гэсэн үг. Анги нь хэрэгжүүлэх гэж буй интерфейс болон өвөг интерфейс дотор зарлагдсан бүх аргуудыг хэрэгжүүлэх ёстой, эсвэл анги нь abstract гэж зарлагдах ёстой. Тухайн анги доторх аргын сигнатур нь –нэр мөн аргументуудын тоо болон төрөл нь– интерфейс дотор харагдаж байгаа аргын сигнатуртай заавал таарч байх ёстой. StockApplet нь StockWatcher интерфейсийг хэрэгжүүлж байна, иймд хэрэглүүр нь valueChanged аргад хэрэгжүүлэлтийг хангаж өгнө. Тэр арга нь хэрэглүүрийн дэлгэцийг эсвэл эсрэг тохиолдолд уг мэдээллийн хэрэглээг хуурамчаар өөрчилж сайжруулж байна.

Интерфейсийг төрлөөр ашиглах

Шинэ интерфейсийг тодорхойлох үед шинэ лавлах өгөгдлийн төрөл тодорхойлдог. Бусад ямар ч өгөгдлийн төрлийн нэрийг ашиглаж болох хаана ч интерфейсийн нэрүүдийг ашиглаж болно. StockMonitor анги доторх watchStock аргын эхний аргумент дэх өгөгдлийн төрөл нь StockWatcher байна:

```
public class StockMonitor {
    public void watchStock(StockWatcher watcher,
                           TickerSymbol tickerSymbol,
                           BigDecimal delta) {
    ...
    }
}
```

Төрөл нь интерфейсийн нэр байх лавлах хувьсагч руу зөвхөн тухайн интерфейсийг хэрэгжүүлэх ангийн төлийг шилжүүлж болно. Тэгэхээр зөвхөн StockWatcher интерфейсийг хэрэгжүүлэх ангийн төлүүд нь хөрөнгийн үнийн өөрчлөлтөнд зарлагдахын тулд бүргэгдэж чадна. StockWatcher обьектууд нь valueChanged гэсэн аргатай болох нь баттай байна.

Анхаар! Интерфейсийг ихэсгэж болохгүй

StockWatcher –руу зарим функционалиудыг нэмэхэд, жишээ нь хөрөнгийн үнийн утга өөрчлөгдсөн эсэхийг тооцохгүйгээр одоогийн ханшийг илтгэсэн арга нэмэхэд:

```
public interface StockWatcher {
    void valueChanged(TickerSymbol tickerSymbol,
                      BigDecimal newValue);
    void currentValue(TickerSymbol tickerSymbol,
                      BigDecimal newValue);
}
```

Хэрэв энэ өөрчлөлтийг оруулбал, хуучин StockWatcher интерфейсийг хэрэгжүүлж буй бүх ангиуд нурна. Яагаад гэвэл, тэд нар интерфейсийг дахин хэрэгжүүлэхгүй юм. Уг интерфейсийг хэрэглэж буй программистууд нилээд эсэргүүцэх болов уу.

Интерфейсийнхээ бүх хэрэглээг урьдчилан харж анхнаас нь зөв бүрэн тодорхойлохыг бодоорой. Та магадгүй кодоо нураах эсвэл интерфейсүүдийг дараа нь үүсгэх хэрэгтэй

болох байхаа. Жишээ нь, шинэ арга зарласан StockWatcher –н дэд интерфейс болох StockTracker –г үүсгэж болно.

```
public interface StockTracker extends StockWatcher {
    void currentValue(TickerSymbol tickerSymbol,
                      BigDecimal newValue);
}
```

Одоо таны кодын хэрэглэгчүүд шинэ эсвэл хуучин интерфейсийг сонгож болно.

"Static Import" байгууламж

Бусад төрлүүдэд хэрэглэгддэг статик тогтмолууд болон аргуудыг зарим төрөл тодорхойлдог. Жишээ нь, java.lang.Math анги нь PI тогтмол болон cos аргыг тодорхойлдог.

```
public static final double PI 3.141592653589793
public static double cos(double a)
```

Уг обьектуудыг өөр ангиас хэрэглэхдээ ердөө ангийн нэрийг урд нь тавиарай.

```
double r = Math.cos(Math.PI * theta);
```

Ангийн нэрийг дахин дахин урд нь тавих нь кодыг үймүүлж мэднэ. Үүнээс зайлсхийхийн тулд программистууд заримдаа интерфейс рүү болон интерфейсийн удам руу статик обьектыг тавьж өгдөг. Уг Constant Interface Antipattern гэх практик нь сайшаагддаггүй.(Joshua Bloch -ын Effective Java номонд Constant Interface Antipattern дээрээс илүү мэлээллийг авч болно.) Үүнийг Java програмчлалын муу практик гэж авч үздэг. Учир нь, интерфейсийг анги хэрэгжүүлэх үед энэ нь ангийнхаа нээлттэй API –гийн хэсэг болдог. Өөр ангийн статик гишүүдийг ашиглах гэх мэт нарийвчилсан хэрэгжүүлэлтүүд нь нээлттэй API –руу урсдаггүй байх нь дээр шүү.

5.0 хувилбар нь энэ байдалд зориулж статикаар оруулах байгууламж гэх өөр шийдвэр танилцуулсан. Энгийн import –той төстөй уг байгууламж нь, статик обьектуудыг агуулж байгаа төрлөөс удамшуулахгүйгээр, статик обьектууд руу бэлтгэгдээгүй хандалтыг зөвшөөрдөг. (Стандарт import –н талаар илүү мэдээллийг Using Package Members хэсгээс хараарай.)

Обьектуудыг тус тусад нь оруулж ирж болдог:

```
import static java.lang.Math.PI;
```

эсвэл бүлэг шиг:

```
import static java.lang.Math.*;
```

Ангийг нэг л оруулахад, нэгэнт обьектууд нь бэлтгэлгүйгээр хэрэглэгдэж болно. Жишээ нь, өмнөх хэсэг код нь ийм болно:

```
double r = cos(PI * theta);
```

Санамж: Статик импортыг (бүгдэд нь шаардлагатай бол) маш цөөн хэрэглээрэй. Нэг эсвэл хоёр ангиас цөөхөн статик обьектуудад дахин дахин хандах хэрэгтэй болсон нөхцөлд энэ нь хэрэгцээтэй. Тусгай статик обьектыг аль анги нь тодорхойлж байгааг кодын уншигчид мэдэхгүйгээс болж статик импортыг хэтэрхий хэрэглэх нь кодыг унших болон сайжруулахад төвөгтэй болгоно. Ёсоор нь хэрэглэхэд, ангийн нэрүүдийн давхацлыг хассанаараа энэ нь кодыг илүү унших боломжтой болгоно.

Дүгнэлт

Интерфейс нь хоёр обьектын хоорондох харилцааны протоколыг тодорхойлдог. Интерфейсийн тодорхойлолт нь зарлалт болон их биеийг агуулдаг. Defining an Interface хэсэг нь интерфейс зарлалтын бүх боломжит компонентуудыг харуулсан байгаа. Интерфейсийн их биед аргуудад зориулсан зарлалтуудыг багтаадаг, гэхдээ хэрэгжүүлдэггүй. Мөн интерфейст тогтмолын зарлалтуудыг агуулж болдог. Интерфейсийг хэрэгжүүлж буй анги нь интерфейс дотор зарлагдсан бүх аргуудыг хэрэгжүүлэх ёстой. Төрлийг хэрэглэж болдог хаана ч интерфейсийн нэрийг хэрэглэж болдог.

Асуулт болон дасгал

Асуулт

1. java.lang.CharSequence интерфейсийг хэрэгжүүлдэг анги нь ямар аргуудыг хэрэгжүүлэх ёстой вэ?

2. Дараах интерфейст ямар алдаа байна?

```
public interface SomethingIsWrong {
    public void aMethod(int aValue){
        System.out.println("Hi Mom");
    }
}
```

3. Дээрх интерфейсийг зас.

4. Дараах интерфейс нь зөв үү?

```
public interface Marker {
}
```

Дасгал

1. java.lang багц дотор олдсон CharSequence интерфейсийг хэрэгжүүлдэг анги бич. Өгөгдөл мэтээр ашиглахдаа уг номноос догол мөрүүдийн нэгийг нь сонгоорой.

2. Та клиентүүдрүүгээ одоогийн огноо болон цагийг тогтмол мэдээллэдэг цагын сервер бичсэн гэж бод. Клиентүүд дээрх тусгай протоколыг албадахад хэрэглэх серверийн интерфейс бич.

Багцуудыг үүсгэх болон ашиглах

Хэв шинжийг хийх нь хэрэглэх болон хайхаас амар. Мөн нэрний зөрчил, удирдлагад нэвтрэх, программистуудын багууд нь багцуудтай холбогдохоос зайлсхийж болно.

Тодорхойлолт: Багц нь нэрийн орон зайн менежментийг болон хандалтын хамгаалалтыг хангадаг хамааралтай төрлүүдийн цуглуулга юм. Төрлүүд гэдэг нь ангид, интерфейст, тоочилтонд, мөн нэмэлтэнд хэрэглэгддэг. Тоочилтын талаар дэлгэрэнгүй мэдээллийг Enumerated Types хэсгээс хараарай.

Java платформын хэсгийн хэв шинжүүд нь ангиудын боодол ялгаатай багцуудын гишүүн: гол ангиуд нь java.lang –д, унших болон бичих ангиуд (оролт ба гаралт) нь java.io-д оршино. Та ч өөрийнхөө хэв шинжийг багцууд руу хийж болно.

Ангиудын утгыг хараад та яагаад багц руу эдгээрийг хийхийг хүсэж байгаагаа шалга. График обьектуудын (дугуй, дөрвөлжин, шугам, цэг) түүврийг дахин тодорхойлсон багийг бичсэн гэж үзье. Мөн та интерфейсийг бичсэн. Draggable ангиуд нь хэрэглэгч хулганаар чирэхийг хэрэгжүүлдэг.

```
//in the Graphic.java file
public abstract class Graphic {
    . . .
}

//in the Circle.java file
public class Circle extends Graphic implements Draggable {
    . . .
}

//in the Rectangle.java file
public class Rectangle extends Graphic implements Draggable {
    . . .
}

//in the Draggable.java file
public interface Draggable {
    . . .
}
```

Та эдгээр ангиуд болон интерфейсийг багцлах хэдхэн шалтгаан байдаг.

- Та болон бусад программистууд нь эдгээр холбогдсон хэв шинжүүдийг амархан хянах чадна.
- Та болон бусад программистууд нь график функцуудыг холбодог хэв шинжүүдийг хаанаас олохоо мэднэ.
- Таны хэв шинжүүдийн нэрүүд нь бусад багцуудын хэв шинжүүдийн нэрүүдтэй зөрөхгүй. Яагаад гэвэл Багц нь шинэ нэрийн зай үүсгэдэг.
- Та багцуудын өөр хоорондоо ямар нэг хязгаарлалтгүйгээр нэвтрэхийг одоохондоо зөвшөөрч чадахгүй.

Багц үүсгэх

Та багц үүсгэхдээ төрлөө (анги, интерфейс, тоочилт эсвэл бичлэг) тавьж өгнө. Үүндээ аль нэгэн тодорхойлогдсон сорс файлын эхэн дэхь багц хэллэгийг тавьж өгнө. Жишээ нь: доорх код нь Circle.java сорс файл доторх circle ангийг graphics банц руу хийж байгааг харуулжээ.

```
package graphics;

public class Circle extends Graphic implements Draggable {
    . . .
}
```

graphics багцын Circle анги нь нийтийн хандалттай гишүүн юм. Та graphic багцын анги эсвэл интерфейс тодорхойлдог бүх сорс файлын эхэн дэхь багц хэллэгийг агуулж байх ёстой. Та түүнчлэн Rectangle.java дахь хэллэгийг бас агуулж байх хэрэгтэй.

```
package graphics;

public class Rectangle extends Graphic implements Draggable {
    . . .
}
```

Багц хэллэгийн цар хүрээ нь бүтэн сорс файл, бүх ангиуд, интерфейсүүд, тоочсон төрлүүд мөн Circle.java дахь Rectangle.java тодорхойлогдсон бичлэгүүд нь бүгд graphics багцын гишүүн юм. Хэрвээ та энгийн сорс файлд олон ангиудыг тавибал эдгээрийн зөвхөн ганц нь нийтийн хандалттай байх ба сорс файлын үндсэн нэрийг ашиглах болно. Зөвхөн нийтийн хандалттай багцын гишүүдэд багцын гаднаас хандаж болдог.

Санамж: Зарим эмхэтгүүрүүд нь нэг нийтийн хандалттай ангиас илүү олныг хүлээн зөвшөөрдөг. Хэдийгээр тийм боловч бид нэг файлд нэг нийтийн хандалттай анги байхыг зөвшөөрдөг. Яагаад гэвэл, нийтийн хандалттай ангиуд нь бүх эмхэтгүүрүүдэд ажиллах болон хайхад хялбар болдог.

Хэрвээ та багц хэллэгийг ашиглахгүй бол таны нэр байхгүй default package дахь төрөл үгүй болно. Ерөнхийдөө заяамал багц нь зөвхөн жижиг эсвэл түр зуурын хэрэглэгдэхүүнүүд эсвэл таныг дөнгөж хөгжиж байхад зориулагдсан. Өөрөөр хэлбэл, ангиуд, тоочсон төрлүүд мөн бичлэгүүд нь нэрлэсэн багцад хамаарна.

Багцыг нэрлэх

Дэлхийн бүх программистууд Java программын хэлийг ашиглан ангиуд, интерфейсүүд, тоочсон төрлүүд мөн бичлэгүүдийг бичиж байна. Хоёр программист ялгаатай хоёр ангид ижил нэр ашиглаж байгаа нь гайхалтай. Баримт: өмнөх жишээнд, java.awt багцанд Rectangle анги байхад дахин Rectangle ангийг тодорхойлж байгааг үзүүлсэн. Эмхэтгүүр нь ижил нэртэй хоёр ангийг хүлээн зөвшөөрч байна. Яагаад? Яагаад гэвэл тэд ялгаатай багцуудынх ба ямар ч анги багцынхаа нэрийг агуулж байдаг. graphics багц дахь Rectangle ангийн бүтэн нэр нь graphics.Rectangle бөгөөд java.awt багц дахь Rectangle ангийн бүтэн нэр нь java.awt.Rectangle юм.

Журам: Компаниуд багцынхаа нэрийг хэрэглэхдээ интернэт эзэмших нэрийгээ урвуугаар нь хэрэглэдэг. com.company. Тухайн нэг компанийн хувьд нэрийн давхцал үүсвэл үүнийг зохицуулах журам байдаг ба энэ нь компанийн нэрийн араас газар орон эсвэл төлөвлөх нэрийг тавьж өгөх явдал юм.

Багц гишүүдийг ашиглах

Тодорхойлогдсон public хандалттай багц гишүүдэд багцын гаднаас хандах боломжтой. Энэ багцын гадна талаас public багц гишүүнийг ашиглахдаа та доорхоос нэг эсвэл нэлээн хэдийг ашиглах хэрэгтэй.

- урт (мэргэшсэн) нэртэй гишүүн хэрэглэх
- багц гишүүдыг оруулах
- гишүүдийн бүрэн багцыг оруулах

Багц гишүүнийг нэрээр хэрэглэх

Одоог хүртэл энэ хичээл дэх жишээнүүд нь Rectangle мөн StockWatcher гэх мэт тэдний энгийн нэрүүдээр хэрэглэгддэг төрлүүд байдаг. Ижил багцанд таны бичсэн код тэр гишүүн нь эсвэл нөгөө гишүүн нь чухал байсан бол та багц гишүүний энгийн хэрэглэж болно. Хэдийгээр та ялгаатай багцаас гишүүн хэрэглэхийг хүсэвч тэр багц нь чухал биш ба та багц нэрийг агуулдаг гишүүний мэргэшсэн нэрийг хэрэглэх ёстой. Энэ нь өмнөх жишээний багц дахь Rectangle анги зарлахад зориулсан хэмжээтэй нэр юм.

```
graphics.Rectangle
```

Та доорх урт нэрийг graphics.Rectangle-ийн жишээ болгон ашиглаж болно.

```
graphics.Rectangle myRect = new graphics.Rectangle();
```

Багц гишүүнийг оруулах

Тохиосон файлдаа онцолсон гишүүнээ оруулахдаа import хэллэгийг файлынхаа явцад ямар нэгэн анги, эсвэл интерфейс тодорхойлогчийн өмнө тавих ба хэрвээ тэнд нэг байвал багц хэллэгийн дараа тавина. Энд өмнөх хэсэгт байгуулсан graphics багцаас Circle ангийг яаж оруулж буйг харуулав.

```
import graphics.Circle;
```

Та одоо энгийн нэрээр Circle ангийг хэрэглэж болно.

```
Circle myCircle = new Circle();
```

Хэрвээ та graphics багцаас цөөн гишүүдийг нь ашигласан бол энэ хандлага нь сайн ажиллана.

Бүтэн багц оруулах

Тодорхой багцанд агуулагдах төрлийг import хэллэгтэй мөн (*) тэмдэгтийг ашиглан оруулна.

```
import graphics.*;
```

Одоо та богино нэрээр graphics багцын ямар нэгэн анги эсвэл интерфейсыг хэрэглэж болно.

```
Circle myCircle = new Circle();
Rectangle myRectangle = new Rectangle();
```

import хэллэг дэх од тэмдэглэгээ нь зөвхөн багц доторх бүх ангиудыг тодорхойлоход ашиглагддаг. Үүнийг багц доторх ангиудын дэд тохиргоог тааруулахад ашиглахгүй.

```
import graphics.A*;     // does not work
```

Үүний оронд энэ нь алдаа үүсгэдэг. Та import хэллэгтэй хамт оруулахдаа зөвхөн энгийн багц гишүүн эсвэл бүтэн багц л оруулна.

Санамж: Бусад оруулалтын нийтлэг төрөл нь таныг зөвхөн анги болон анги дахь public хандалтыг оруулахыг зөвшөөрдөг. Жишээ нь: хэрвээ graphics.Rectangle анги нь Rectangle.DoubleWide мөн Rectangle.Square шиг ашигтай ангиудыг дотроо агуулдаг бол та Rectangle мөн ангиуд дотор энэ замыг оруулж болно.

```
import graphics.Rectangle.*;
```

Танд зориулж Java эмхэтгүүр нь 3 бүтэн багцыг оруулж өгсөн.

- заяамал багц
- java.lang багц
- заяамал тохиосон багц

Санамж: Багцууд нь шатлан захирдаггүй. Жишээ нь: java.util.* оруулалт нь regex.Pattern шиг pattern анги хэрэглэгдэхгүй. Та үргэлж java.util.regex.Pattern эсвэл энгийн pattern (хэрвээ та java.util.regex-ыг оруулсан) хоёуланг нь хэрэглэх ёстой.

Нэрсийн давхардлыг арилгах

Нэг багц дахь гишүүн өөр багц дахь гишүүний нэртэй ижил ба хоёр багц нь импортлогдсон үед та мэргэшсэн нэрээр гишүүнд хандах ёстой. Жишээ нь: өмнөх жишээнд graphics багц дахь Rectangle нэртэй ангийг тодорхойлсон байна. java.awt багц нь Rectangle ангийг агуулдаг. Хэрвээ graphics болон java.awt хоёуланг нь импортолж оруулж ирсэн бол дараах нь эргэлзээтэй болно.

```
Rectangle rect;
```

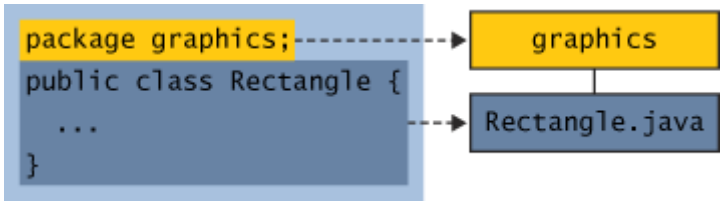
Ийм нөхцөлд яг аль Rectangle ангийг ашиглахаа гишүүний мэргэшсэн нэрийг ашиглан ялгаж өгнө.

```
graphics.Rectangle rect;
```

Source болон анги файлуудыг удирдах

Java платформын олон гүйцэтгэлүүд нь шаталсан файлын системүүд дэхь Source болон анги файлуудын удирдлагад найддаг.Хэдийгээр Java хэлний дүрэм нь үүнд хэрэг болохгүй.

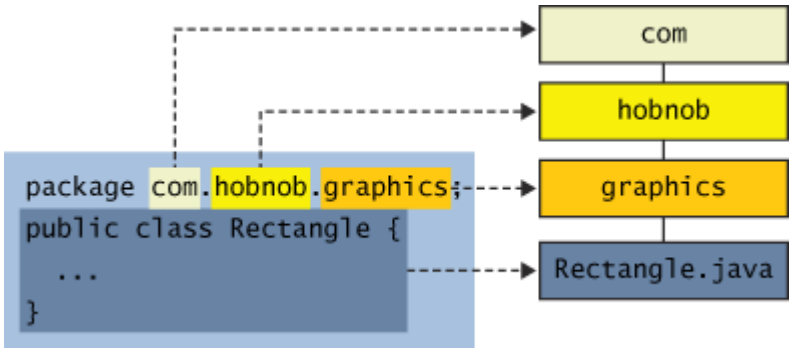
Стратеги нь: Энгийн нэртэй , .java өргөтгөлтэй файлын ангид source кодыг бичиж мөн нэмэлт тайлбар бичнэ.Үүний дараа файлаа багц болон төрөлтэй хамааралтай лавлахад хийнэ.Жишээ нь : Rectangle ангитай source код нь Rectangle.java нэртэй файл байх ёстой.Мөн Файл нь Graphics нэртэй лавлах байх ёстой. Graphics лавлах нь файлын системын хаана ч байж болно. Доор яаж ажилладагийг харууллаа.



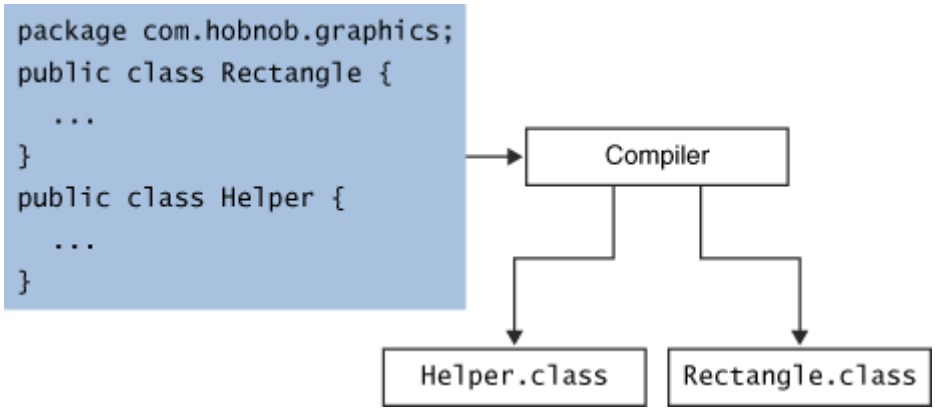
Файлд бэлтгэсэн багцын гишүүний нэр болон замын нэр нь ижил байна. Microsoft Windows-д файлын нэр нь (\ ) тусгаарлагч зураасаар төсөөлөгддөг.

class name `graphics.Rectangle`
pathname to file `graphics\Rectangle.java`

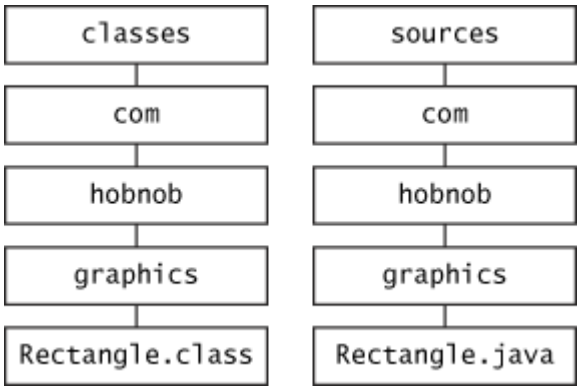
Тохиролцоогоор компани нь багцын нэрүүд дэхь интернет домайн нэрийг цуцалдаг. hobnob.com гэсэн интернетийн домайн нэртэй компани нь урьдчилаад com.hobnob гэсэн багц нэртэй байна. Багцын нэрийн компонент бүр нь дэд лавлахтай байдаг. Тэгэхлээр, хэрэв Hobnob нь rectangle.java файлыг агуулдаг graphics багцтай бол доор үзүүлсэнчлэн дэд лавлахуудын залгаануудыг агуулж байдаг.



Файлаа хөрвүүлэх үед анги болон интерфейсд бүрт ялгаатай гаралттай файл үүсгэж тодорхойлдог.Гаралтын файлын үндсэн нэр нь анги эсвэл интерфейсийн нэр бөгөөд өргөтгөл нь .class байна. Доор зургаар үзүүлэв.



.java , .class файлууд нь багцын нэрийг агуулсан лавлахуудын залгаануудад байж болно. Хэдийгээр энэ нь source шиг ижил лавлахтай байх албагүй. Source болон анги лавлахуудыг доор үзүүлсэнээр тусад нь жагсааж болно.



Үүнийг хийж өгснөөр ангиудын лавлахыг бусад программистуудад кодоо мэдүүлэлгүйгээр өгч болно.

Мөн Java хөрвүүлэгч дэхь –d тохиргоог ашиглан анги файлуудын хаана байгааг доор үзүүлсэнээр тодорхойлж болно.

```
javac -d classes  sources\com\hobnob\graphics\Rectangle.java
```

Яагаад бүх асуудал нь лавлахууд болон файлын нэрүүдийн тухай байдаг юм вэ?

Source болон ангийн файлуудыг удирдах байдал нь Java Virtual Machine (JVM) болон хөрвүүлэгч нь програмын хэрэглэсэн бүгдийг олж чадна.Хөрвүүлэгч нь таны програмын шинэ ангийг хөрвүүлэх тэр тохиолдолд энэ нь задарсан нэрүүдтэй ангийг олох ёстой.Үүнтэй адилаар JVM нь таны програмын шинэ ангийг ачаалах тэр тохиолдолд энэ нь дээрх аргуудыг хүсэх ангийг олох ёстой.

Хөрвүүлэгч болон JVM хоёулаа таны ангийн байрлах зам бүрийн лавлах эсвэл JAR аас ангиудыг хайдаг.

Тодорхойлолт: Ангийн зам нь анги файлуудыг хайдаг JAR файлууд эсвэл директоруудын дараалсан жагсаалт юм.

Лавлах бүрт байгаа ангиудын замын жагсаалт нь багцын лавлахууд харгалзах дээд түвшний лавлахууд харагдана. Дээд түвшний лавлахуудаас хөрвүүлэгч болон JVM нь ангийн багц болон ангийн нэр дээр үндэслэн замын үлдсэнийг нь байгуулж чадна. Жишээ нь : ангийн зам нь лавлахын бүтэцэд нэвтрэхдээ өмнө үзүүлсэн диаграмын

classes аар холбогдоно. Гэхдээ com –аар эсвэл com –тай ямар нэг лавлахууд биш. Хөрвүүлэгч болон JVM хоёулаа бүтэн багцын нэртэй хамт .class файлын замын нэрийг байгуулдаг.

Хөрвүүлэгч болон JVM нь байгаа лавлах болон JAR файлыг агуулсан Java платформын ангийн файлуудыг хайдаг. Өөрөөр хэлбэл, байгаа лавлах болон Java платформын ангийн файлууд нь таны ангийн замд автоматаар суудаг. Зарим ангиуд нь дээрх 2 байрлалаас олдоно. Энэ нь та өөрийнхөө ангийн замд санаа зовох хэрэггүй юм. Зарим тохиолдолд та өөрийнхөө ангийн замыг зааж өгч болно.

Дүгнэлт

Багцыг үүсгэхдээ дотор нь төрлийг (анги, интерфейс, тоочилт эсвэл тайлбар) нь оруулдаг. Анги болон интерфейсийг багц руу оруулахад, package хэллэгийг тухайн анги болон интерфейсийн эх код дотор эхний хэллэг болгон оруулна. Интерфейсийн эсвэл ангийн source болон анги файлын замын нэр нь багцынхаа нэрийг дүрслэдэг.

Өөр багц доторх анги болон интерфейсийг ашиглахад дараах гурван сонголт байдаг.

- Тухайн ангийн эсвэл интерфейсийн мэргэшлийн бүрэн нэрийг ашиглаж болно.
- Тухайн ангийг эсвэл интерфейсийг импортлож оруулж болно.
- Тухайн анги эсвэл интерфейс нь гишүүн нь байх багцыг бүтнээр нь импортлож оруулж болно.

Та өөрийн ангийн замыг тохируулах ёстой. Ингэснээрээ эмхэтгүүр болон ухварлуур нь таны ангиудад болон интерфейсүүдэд зориулсан source болон анги файлуудыг олж чадна.

Асуулт болон дасгал

Асуулт

1. Та өөрийгөө зарим ангиудыг бичсэн гэж бод. Доор хүснэгтэнд жагсааснаар тэдгээрийг гурван багцад хуваах нь дээр гэж сүүлд нь та шийдэв. Ялангуяа, тэдгээр ангиудыг одоогоор нэг заяамал тохиосон багцад байгаа гэж бод.

Багцын нэр	**Ангийн нэр**
`mygame.server`	`Server`
`mygame.shared`	`Utilities`
`mygame.client`	`Client`

   a. Анги бүрийг зөв багцад оруулахын тулд source файл болгон руу ямар код мөрийг нэмэх вэ?

   b. Директорын бүтцэд хийхийн тулд та хөгжүүлэлтийн директор дотроо дэд директоруудыг үүсгэх хэрэгтэй бөгөөд source файлуудыг дэд директорт нь зөв оруулах хэрэгтэй. Ямар дэд директоруудыг үүсгэх ёстой вэ? Ямар дэд директорт source файл болгон орох вэ?

c. Source файлуудыг алдаагүй зөв эмхэтгэхийн тулд тэдгээрт ямар нэгэн өөрчлөлт хийх хэрэгтэй гэж бодож байна уу? Хэрэв тийм бол, юуг өөрчлөх вэ?

Дасгал

1. Дараах source файлуудыг татаж ав.

- `Client`
- `Server`
- `Utilities`

a. Татаж авсан source файлуудыг ашиглан асуулт 1 –т хийсэн өөрчлөлтүүдээ хэрэгжүүл.

b. Залруулсан source файлуудаа эмхэтгэ. (Сануулга: Хэрэв та эмхэтгүүрийг команд мөрнөөс дуудаж байгаа бол, дөнгөж үүсгэсэн mygame директорыг агуулж буй директороос эмхэтгүүрийг дуудаарай.)

Кодлолын ерөнхий асуудлуудыг шийдэх

Энэ бүлэгт танд Java хэлийг сурч байх үед тулгарах ерөнхий асуудлуудыг багтаасан. Асуудал бүрийн ард боломжит хариунуудыг нь жагсаалаа.

Асуудал: Эхмэтгүүр нь ангийг олж чадахгүй байна гэсэн алдаа илрүүлдэг.

- Тухайн анги болон түүний багцыг импортлож оруул.
- Орчны CLASSPATH хувьсагчийг хэрэв тохируулсан бол түүнийг болиул.
- Ангийн нэрийг бичихдээ түүний яг зарласан шиг нь адилхан бич.
- Хэрэв таны ангиуд багцуудад байгаа бол, Managing Source and Class Files хэсэгт дүрслэсэн шиг тэднийг зөв дэд директоруудад байрлуул.
- Мөн зарим программистууд ангийн нэрээ .java файлын нэрээсээ өөр нэрээр хэрэглэдэг. Тэгэхээр та файлын нэрээр бус ангийнхаа нэрээр хэрэглэ. Энэ тохиолдолд та нэрүүдийг нь адилхан өгвөл энэ асуудалд орохгүй.

Асуудал: Ухварлуур нь миний ангиудын нэгийг нь олж чадахгүй байна гэж хэлдэг.

- Ангийн нэрийг (ангийн файлын нэрийг бус) ухварлуурт тусгайлан нэрлэ.
- Орчны CLASSPATH хувьсагчийг хэрэв тохируулсан бол түүнийг болиул.
- Хэрэв таны ангиуд багцуудад байгаа бол, Managing Source and Class Files хэсэгт дүрслэсэн шиг тэднийг зөв дэд директоруудад байрлуул.
- .java файлын байрласан директор дотроос ухварлуурыг дууд.

Асуудал: Миний програм ажиллахгүй байна! Ямар алдаа байна вэ?

Анхан шатны Java программистуудын гаргадаг ерөнхий алдаануудыг доор жагсаан харууллаа. Эдгээрийн аль нэг нь танд саад учруулж байгаа шүү.

- Та switch хэллэгэнд case хэллэг бүрийн ард break түлхүүр үг хэрэглэхээ мартсан уу?

- Та харьцуулах оператор == хэрэглэх үедээ утга олгох оператор = хэрэглэсэн үү?

- Таны давталтууд дахь төгсгөлийн нөхцөл зөв үү? Та нэг давталтыг хэтэрхий эрт эсвэл хэтэрхий сүүлд дуусгаагүй гэдгээ хараарай. Энэ нь, өөрийн нөхцөлдөө < эсвэл <= болон > эсвэл >= алийг нь хэрэглэхээ шалгаарай.

- Бүлийн индекс 0 –с эхэлдэг гэдгийг санаарай, иймд цааш үргэлжлэхдээ:

```
for (int i = 0; i < array.length; i++)
    . . .
```

- Та хөвөх таслалтай тоонуудыг харьцуулахдаа == хэрэглэж байна уу? Бутархай тоо нь бодит зүйлийн ойролцоо утга гэдгийг санаарай. Хөвөх таслалтай тоонууд дээр болзолт логик үйлдэл гүйцэтгэх үед их эсвэл бага (> болон <) операторууд нь илүү зохистой.

- Танд битүүмжлэл, удамшил, эсвэл бусад обьект хандалтат програмчлал болон загварын ухагдахуунууд дээр бэрхшээл учирсан бол Object-Oriented Programming Concepts хэсгээс мэдээллийг аваарай.

- Хэллэгүүдийн блокууд угалзан хаалтаар {} хашигдсан гэдгийг шалгаарай. Дараах код мөрүүдийг захлан харуулснаас болж зөв юм шиг харагдаж байна, гэхдээ угалзан хаалт орхигдсон учраас мөрийг захласны хэрэг гарсангүй.

```
for (int i = 0; i < arrayOfInts.length; i++)
    arrayOfInts[i] = i;
    System.out.println("[i] = " + arrayOfInts[i]);
```

- Та зөв болзолт оператор хэрэглэж байна уу? Та && болон || операторуудыг ойлгон, тэдгээрийг зохистой зөв хэрэглэсэн гэдгээ шалгаарай.

- Та үгүйсгэл операторыг хэрэглэдэг үү? Маш их? Энэгүйгээр болзлуудыг илэрхийлэхийг оролд. Сайн хийвэл алдаагүй бөгөөд будилаангүй болно.

- Та do-while хэрэглэж байна уу? Хэрэв тийм бол do-while нь ядаж нэг удаа гүйцэтгэдэг, гэхдээ төстөй while давталт нь огт гүйцэтгэдэггүй байж болдог гэдгийг мэдэх үү?

- Та аргументын утгын аргаас өөрчлөх гэж байна уу? Java доторх аргументууд утгаар дамждаг бөгөөд арга дотор өөрчлөгдөж чадахгүй.

- Та хэллэгийг дуусгахдаа нэмэлт цэгтэй таслал (;) санаандгүй нэмсэн үү? Хэллэгийн төгсгөлд нэмэлт цэгтэй таслал тавихгүй шүү:

```
for (int i = 0; i < arrayOfInts.length; i++) ;
    arrayOfInts[i] = i;
```

Товъёог

1.