**Пакеж болон Интерфейс**

**Пакеж**

Хоорондоо холбоотой бүлэг класс болон интерфейсүүд юм. Ижил нэртэй классуудыг ялгахад пакеж нь тус болдог. Програмд пакежийг импорт хийж оруулан ашиглаж болдог. Жава пакежийн төрлүүд нь дотроо өмнө нь тодорхойлсон **predefined package** болон хэрэглэгчийн тодорхойлсон **user-defined package** гэсэн хоёр төрөлтэй байдаг. Энэ нь хөгжүүлэгч өөрөө нэр өгч нэрлэх ба нэрлэхдээ нэрийг нь жижиг үсэгээр бичих ба нэрийг нь бичихдээ java болон javax гэсэн жавагийн түлхүүр үгийг ашиглаж болохгүй. Пакежийн нэрийг тоо болон дундуур зураасаар эхэлж болохгүй харин доогуур зураасыг ашиглаж болно. Жишээ нь: **package < package name >;** доорх жишээнд polymorphism гэсэн нэртэй пакеж үүсгэж үүн дотроо Dispatch гэсэн класс зарласан байна.

**package** polymorphism;

**class** Dispatch {

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

A a = **new** A(); // object of type A

B b = **new** B(); // object of type B

C c = **new** C(); // object of type C

A r; // obtain a reference of type A

r = a; // r refers to an A object

r.callme(); // calls A's version of callme

r = b; // r refers to a B object

r.callme(); // calls B's version of callme

r = c; // r refers to a C object

r.callme(); // calls C's version of callme

}

}

Predefined package –нь JavaAPI дотор байдаг ба үндсэн хоёр бүлгийн пакеж байдаг. Үндсэн пакеж буюу java гэж эхэлдэг Өргөтгөл пакеж буюу javax гэж эхэлдэг. Жишээ нь: java.io, java.util, java.awt, javax.swing гэх мэт. Жава дахь пакежийн жагсаалтуудыг энэ холбоосоос харна уу. <http://docs.oracle.com/javase/8/docs/api/>

Мөн энэ ойлголт нь жава програм дотор хэрэглэгдэж байгаа классуудыг олоход тусалдаг ( компляторт). Жишээ нь: **import** javax.swing.JOptionPane; гэж бичиж мэдээлж байна.

**package** pack;

**import** javax.swing.JOptionPane;

**class** Addition {

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

String firstNumber, secondNumber;

**int** first, second, sum;

firstNumber = JOptionPane.*showInputDialog*("Эхний тоогоо оруулна уу");

secondNumber = JOptionPane.*showInputDialog*("Хоёр дахь тоогоо оруулна ");

first = Integer.*parseInt*(firstNumber);

second = Integer.*parseInt*(secondNumber);

sum = first + second;

JOptionPane.*showMessageDialog*( **null**,"Нийлбэр = "+ sum, "Үр дүн", JOptionPane.***INFORMATION\_MESSAGE***);

System.*exit*(0);

}

}

**Интерфейс**

Ер нь амьдрал дээрээс хархад хоёр төрлийн объектууд оршин байдаг. Амьдралд бодитой оршин байдаг объектууд, бодитой оршин байдаггүй объектууд ( компьютерийн санах ойд ). Бодитой оршин байдаггүй объектууд бол хийсвэр объектууд юм. Бодит байдалд хийсвэр обьектууд нь бодит обьектуудаас дутахгүй олон байдаг.Нэгэнт програм нь бодит амьдрал дээрх системийг програмчлан хийдэг тул програмистууд эдгээр хийсвэр обьектуудыг програмдаа ашиглах шаардлагатай болдог.Амьдралд хийсвэр обьектууд ямар үүрэгтэй вэ? Эдгээр хийсвэр обьектууд байдаггүй байсан бол амьдралд үйл ажиллагаа хэрхэн явагдах байсан бол ?

Жишээ нь: **Амьтан** гэсэн хийсвэр объект буюу хийсвэр ойлголтыг бид хэрэглэдэггүй байг гэж үзье. Дараах амьдралын хэсгийг ажиглая.

**Дорж бол амьтны хүрээлэнгийн ажилтан:** Амьтны хүрээлэн гэдэг зүйлийг үнэндээ ойлгуулах боломжгүй болно.

**Болд тэнд шинээр ажилд орсон байг:** Бас Ажил гэдэг ойлголтыг хэрэглэхгүй бол энэ өгүүлбэрийг ч хэлэх боломжгүй.

**Дорж Батад ажилаа танилцуулна. Хэрхэн танилцуулах бол ?**

**Асуудал 1**

Энэ бол Баавгай. Баавгай бол иддэг, уудаг, явдаг, төрдөг, үхдэг, хаздаг.

Энэ бол Хандгай. Хандгай нь иддэг, уудаг, явдаг, төрдөг, үхдэг.

Энэ бол Болжмор. Болжмор нь иддэг, уудаг, явдаг, төрдөг, үхдэг, нисдэг.

Энэ бол Матар. Матар нь иддэг, уудаг, явдаг, төрдөг, үхдэг, хаздаг.

Энэ бол Тогоруу. Тогоруу нь иддэг, уудаг, явдаг, төрдөг, үхдэг, нисдэг.

.........

Ямар асуудал энд үүсч байна вэ ?

**Дорж гэнэт нэг амьтны нэрийг мартжээ. Хэрхэн асуух вэ ?**

Амьтан, эд юм, зэрэг хийсвэр ойлголтуудыг хэрэглэхгүйшүүдээ

**Асуудал 2**

**Дорж Батад дараах зүйлийг хэлэхийг хүсэв.**

Бүх амьтдаа өдөрт хоёр удаа хооллох хэрэгтэй. Хэрхэн хэлэх вэ ?

Өдөр бүр Баавгайг хоёр удаа хооллох хэрэгтэй.  
Өдөр бүр Хандгайг хоёр удаа хооллох хэрэгтэй.  
Өдөр бүр Матарыг хоёр удаа хооллох хэрэгтэй.  
Өдөр бүр Тогорууг хоёр удаа хооллох хэрэгтэй.  
......... ямар асуудал үүсч байна.

Гэх мэт асуудал үүсч байгааг энэ амьдралын хэсгээс гарч ирж байна. За тэгэхээр Амьтан гэдэг үг хэрэглэхгүй бол бүтэхгүй байна. Дээрх жишээнээс харахад хэдийгээр амьдралд биетэйгээр оршин байдаггүй ч хийсвэр объект гэдэг зүйл маань амьдралд ямар их чухал үүрэгтэйг харж байна. Ер нь хийсвэр объектууд бодит объектуудаас илүү ихээр амьдралд хэрэглэгддэг нь мэдэгдэж байна. Үүнтэй адилаар ОХП нь бодит амьдрал дээрх зүйлийг програмчлах зорилгоор үүссэн ба бид програмдаа энэ хийсвэр ойлголтыг хэрэгжүүлэхгүй бол мөн дээрх амьдралын хэсэг дээр гарч байгаа асуудал шиг програмд ч мөн адилхан гарах юм. Тэгэхээр хийсвэр объектууд гэж юу вэ гэсэн асуулт гарч ирж байна.

Хийсвэр объектууд нь дараах онцлогуудтай:

* Ямар нэг ойлголтыг илэрхийлнэ: Амьтан, Машин, Дугуй ...
* Тухайн ойлголтод ямар ямар үйлдэл байх ёстойг дүрэм хэлбэрээр заасан байна: Амьтан бол явдаг, иддэг... Машин бол араагаа солидог, хурдалдаг.... Дугуй бол араа солих, унах.... гэх мэт
* Хийх үйлдлүүдийг нь хэрхэн хийгдэхийг огт заадаггүй.

Тиймээс хийсвэр обьектууд нь тэр зүйлийн хийх ёстой үйлдлүүдийг заасан дүрмүүдийг агуулсан обьектууд юм. Ийм хийсвэр төрлүүдийг програмчлалын хэлэнд жишээ нь Java дээр яаж хэрэгжүүлэх вэ? Жава хэлэнд хийсвэр объектийг **ИНТЕРФЕЙС** гэсэн ойлголтыг ашиглан хийдэг. Интерфейсийг үүсгэх нь энгийн класс үүсгэхтэй төстэй. Класс гэсэн нэрийн оронд **interface** гэсэн түлхүүр үгийг хэрэглэдэг. Үүн дотор тухайн интерфейсийн гүйцэтгэх ёстой үйлдлүүдийг заасан кодгүй функцүүдийг бичиж өгдөг. Интерфейсийг бас өөр үгээр тайлбарлаж болно. интерфейс бол бусад классуудад биелүүлэхийг тушаасан командуудын цуглуулга юм. Дээр бид жава нь энгийн удмашлыг дэмждэг харин нийлмэл удамшлыг дэмждэггүй гэж ярьж байсан. Тэгэвл нийлмэл удамшлыг жава д хэрэгжүүлэх арга нь интерфейс юм. Нийлмэл удамшлыг хэрэгжүүлэхдээ **implements** гэсэн түлхүүр үгийг ашиглана. Өөрөөр хэлбэр интерфэйсийг имплемент хийсэн классууд уг интерфэйс доторх бүх функцүүдийг имплемент хийсэн байх ёстой хэрвээ функцүүдийг нь оруулж ирэхгүй бол алдаа гарах юм. Энэ нь нөгөө талаас програмистуудад бусад функцийг бүрэн оруулж ирэхгүй байх зүйлийг хааж байга хэрэг юм. Интерфэйс доторх функцүүдийг абстракт функцүүд гэж нэрлэнэ.

*access* interface *name* {

*return-type method-name1*(*parameter-list*);

*return-type method-name2*(*parameter-list*);

*type final-varname1* = *value*;

*type final-varname2* = *value*;

//...

*return-type method-nameN*(*parameter-list*);

*type final-varnameN* = *value*;

}

Доорх жишээ нь амьтан гэсэн ойлголтыг ямар байхыг зааж өгсөн байна.

**interface** Animal {

**void** toroh();

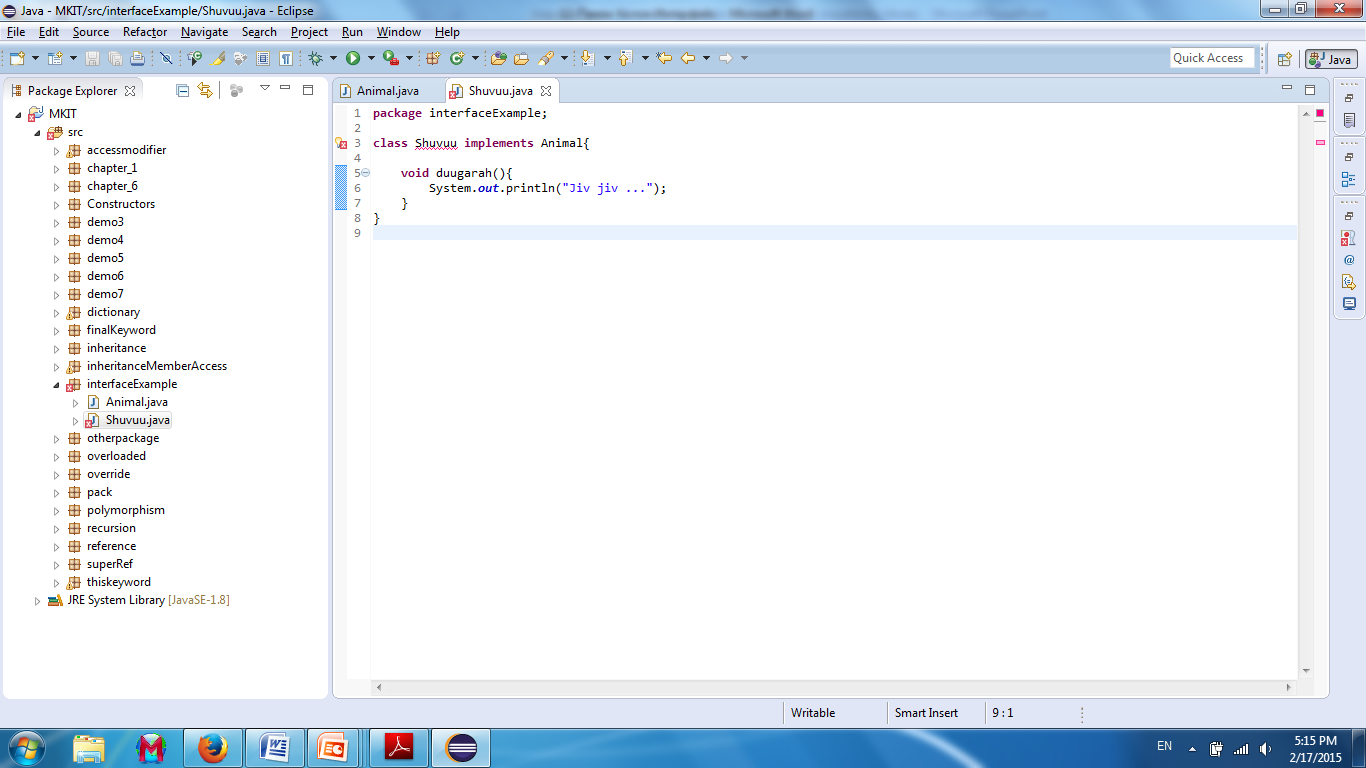
**void** hoolloh();

**void** yavah();

**void** harah();

}

Доорх жишээ нь алдаа гарсаар л байна. Animal интерфейсийн функцүүдийг оруулж ирэхгүй бол энэ алдаа байсаар л байна. Тиймээс Animal интерфейсийн бүх функцүүдийг нь оруулж ирэх хэрэгтэй.



Оруулж ирсэн байдал:

**class** Shuvuu **implements** Animal{

**void** duugarah(){

System.***out***.println("Jiv jiv ...");

}

@Override

**public** **void** toroh() {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** hoolloh() {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** yavah() {

// **TODO** Auto-generated method stub

}

@Override

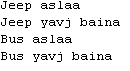
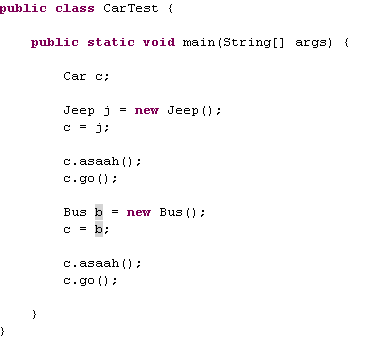
**public** **void** harah() {

// **TODO** Auto-generated method stub

}

}

**Санах зүйл:** Интерфейсийн функцүүд нь **public** эсвэл **package** хандалтын аль нэгээр л зарлагддаг. Private, protected функцүүд байж болохгүй. Мөн интерфейсийн объектийг шууд үүсгэх боломжгүй. Учир нь хийсвэр объектууд бол амьдралд бодитойгоор оршин байдаггүй объектууд юм. Хэдий тийм боловч интерфейсийн заагчийг хэрэглэж болдог. Интерфейс дотор хийсвэр ( кодгүй ) функцүүдээс гадна **тогтмол** өгөгдлүүд байж болно. Ийм өгөгдлүүдийг интерфейс дотор зүгээр л класс дотор зарладаг шиг зарлаж хийж өгдөг. Эдгээр өгөгдлүүдэд заавал анхны утга тавьж өгнө. Эдгээр өгөгдлүүд тогтмол тул уг интерфейсийг имплемент хийсэн бусад классуудад уг өгөгдлүүдийн утгыг өөрчлөх боломжгүй.



Дээрх Jeep, Bus гэсэн хоёр класс маань Car интерфейсийг имплемент хийсэн тул энэ хоёрийн объектийг бас Car гэсэн утгаар хандаж болно гэсэн санаа гарч байна. Програмыг хийж байх явцад нэг классын объектийг төрөл бүрээр үүсгэх шаардлага гардаг. Энэ тохиолдолд өөр өөр байгуулагч функц хэрэгтэй болно. Өөрөөр хэлбэл классын байгуулагч функцийг хэт ачаалалттай болгоно гэсэн үг. Нэг класс дотор ижил нэртэй олон функц тавьж өгөхийг хэт ачаалалттай гэж үзсэнийг санана уу. Жишээ нь: А класс Б классаас удамшсан Б класс нь С классаас удамшсан байг. Тэгвэл А классын объектийг байгуулахад эхлээд С классын объект дараа нь Б классын объект дараа нь А классын объект үүсдэг. Энэ юу гэсэн үг вэ ? Амьдралд дээрх жишээгээр тайлбарлая. Жишээ нь хүн өөрөө эцэг эхээсээ үүснэ эцэг эх нь өвөг эцэг эхээсээ гэх мэтээр үүснэ. Би үүсэхийн тулд эхлээд манай аав ээж хоёр үүснэ. Манай аав ээж хоёр үүсэхийн тулд манай аав ээж хоёрын аав ээж нар .... үүснэ гэх мэтээр үүснэ.

Интерфейс нь дараах гурван тохиолдолд голчлон хэрэглэгддэг.

* Нийлмэл удамшлыг хэрэгжүүлэх – Нэг классыг өөр өөр өнцөгөөр харах боломж
* Классын агуулалтад, функцийн аргументад – Програмын уян хатан байдлыг нэмэгдүүлнэ
* Полиморфизмд – Програмын өргөтгөгдөх боломж, бага кодоор ихийг бүтээх, програмд өөрчлөлт оруулалгүйгээр нэмэлт хийх боломж.

Хэрэв нэг класс нэг болон түүнээс дээш тооны классын классаас удамшиж байвал түүнийг нийлмэл удамшил гэнэ.

Олон классаас удамших нийлмэл удамшлын дутагдал: Юуны өмнө Жава нь нийлмэл удамшлыг дэмждэггүй. Зарим хэлүүд дэмждэг – C++, Python …

**Ромбын проблем:**   
Хуурай газар болон усны амьтан классууд дээр Амьтан классаас удамшин орж ирсэн Үхэх функцийг дахин тодорхойлж эдгээр амьтдыг өөр өөрөөр үхдэг болгосон гэж үзье.

**Асуудал :** Хоёр нутагтан дээр Үхэх функцийг дуудвал Хуурай газрын амьтны үхэх нь дуудагдах уу? Усны амьтны үхэх нь дуудагдах уу?

Ер нь бол үүнийг нийлмэл удамшил дэмждэг хэлүүд төрөл бүрээр шийддэг. Жишээ нь C++ ийн хувьд хуурай газрын амьтан болон, усны амьтан нь тусдаа Амьтан гэсэн хоёр обьекттой байх ба үүнийгээ виртуаль удамшил хэмээх ойлголтоор шийддэг. Үхэх функцийг нь дуудахдаа эх классын нэрийг хэрэглэнэ.

Амьтан

Үхэх ( )

Хуурай газарын амьтан

Явах ( )

Усны амьтан

Сэлэх ( )

Хоёр нутагтан

Програмыг debug хийхэд хүндрэлтэй бараг боломжгүй болгодог. Програмын класс хооронд маш хүчтэй хамаарал үүсгэдэг. Энэ нь програмын уян хатан байдлыг багасгана. Өөрөөр хэлбэл удамшлын дээр байгаа класст өөрчлөлт ороход санамсаргүйгээр өөр доор байгаа классын үйл ажиллагаа буруу явагдаж болдог. Энэ нь энгийн удамшлын хувьд ч ажиглагдана. Java, C# зэрэг хэлүүд нь дээрх асуудлуудаас болоод нийлмэл удамшлыг шууд зөвшөөрдөггүй. Харин нийлмэл удамшлыг Интерфэйс ашиглан шийддэг. Нэг класс нь хэд л болон хэдэн интерфэйс имплемент хийж болно. Нөгөө талаас интерфэйсүүд нь өөр интерфэйсүүдээс удамшиж болдог. Интерфэйс ашиглан нийлмэл удамшил хэрэгжүүлэх үед нэгэнт интерфэйс дэх функцүүд нь кодгүй тул удамшиж буй классууд нь түүнийг имплемент хийх шаардлагатай. Абстракт класс бол ингэх шаардлаггүй.

“Амьтан” жишээний хувьд дараах интерфэйсүүд байж болох юм :

* ГазараарЯвдагЗүйлс
* УсаарЯвдагЗүйлс
* ТэнгэрээрНисдэгЗүйлс

Дараа нь Хуурай газрын амьтад, усны амьтад, шувууд дээрх интерфэйсүүдээс хэрэгтэйгээ имплемент хийн авна. Дээрх интерфэйсүүд нь зөвхөн амьтдад ч биш тээврийн хэрэгсэлд ч хэрэглэгдэж болох тул програмын логикийг илүү хялбар, кодыг илүү бага болгох давуу талтай. Интерфэйсүүд нь өөр интерфэйсээс удамшиж болдог ба ингэснээр нийлмэл удамшил жинхэнэ утгаар бий болно. Интерфэйсийг өөр интерфэйсээс удамшуулахдаа **EXTENDS** түлхүүр үгийг хэрэглэнэ. Нэг интерфэйс олон интерфэйсээс удамшиж болох ба энэ тохиолдолд таслалаар тусгаарлан зааж өгнө. Интерфэйс нь зөвхөн интерфэйсээсээ л удамшина.

**interface** A {

**void** meth1();

**void** meth2();

}

**interface** B **extends** A{

**void** meth3();

}

**class** MyClass **implements** B{

@Override

**public** **void** meth1() {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** meth2() {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** meth3() {

// **TODO** Auto-generated method stub

}

}

**class** IFExtend {

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

MyClass ob = **new** MyClass();

ob.meth1();

ob.meth2();

ob.meth3();

}

}

Интерфэйсийг ашигласан үед ромбын проблем үүсэхгүй. Учир нь интерфэйс доторх функц нь кодгүй.

**Доорх зурагт:**  **Д** интерфэйс дотор **Явах()** гэсэн дүрэм зарлагдсан байна. Энэ нь **A** класс, **Е** интерфэйс рүү дамжигдах бөгөөд **А** класс дотор түүнийг дахин тодорхойлж болох бол **Е** интерфэйс дотор болохгүй. Учир нь интерфэйс доторх функц кодгүй байна. Тиймээс **Б** класс дээрээс явах функцийг дуудвал **А** классын явах функц дуудагдана.

Б класс

Д интерфейс

Явах( )

А класс

Е интерфейс

Хэрэв **А** класс дотор **Явах( )** функцийг тодорхойлоогүй бол (А класс нь абстракт байх тохиолдол) **Б** класст **Явах( )** функцийг бичиж өгөх ёстой болно. Нэгэнт удамшин орж ирж буй **Явах( )** функцүүд нь кодгүй дүрэм төдий тул алийг нь дуудах тухай яригдахгүй. Жава нь зөвхөн нэг л классаас удамшдаг тул ямар ч удамшин орж ирсэн функц нь нэг бол нэг л эх класст кодтой байна эсвэл кодгүй дүрэм хэлбэрээр байх юм. Тиймээс хоёр ижил кодтой функцийг алийг нь дуудах вэ гэсэн **ромбын проблем** үүсэхгүй юм.

Д интерфейс

Явах( )

Е интерфейс

Х интерфейс

Б класс

Дээрх тохиолдолд ч ялгаагүй ромбын проблем үүсэхгүй. Мэдээж **Х** интерфэйс, **Е** интерфэйст удамшин орж ирсэн **Явах( )** функц нь кодгүй дүрэм байх ба **Б** класс түүнийг бичиж өгөх ёстой болно. Тиймээс **Б** класс дээрээс **Явах( )** функцийг дуудвал түүний өөрийнх нь функц л дуудагдана. Ромбын асуудал нь удамшсан классууд удамшиж орж ирсэн функцийг дахин тодорхойлсон үед л үүсдэг болохыг анхаарна уу. Интерфэйсийг нийлмэл удамшилд хэрэглэх нь класст өмнө дурдаж байсан шиг хүчтэй хамаарал үүсэхгүй. Өөрөөр хэлбэл нэгэнт **Д, Х, Е** интерфэйсүүд дотор байгаа функцүүд кодгүй тул эдгээрийн үйл ажиллагаа өөрчлөгдөнө гэж байхгүй ба өөрөөр хэлбэл **Б** классын хувьд эдгээр интерфэйсүүдээс болж буруу ажиллах боломж байхгүй болох юм.