**Удамшил**

Объект хандалтат програмчлалын түлхүүр элементүүдийн нэг удамшил юм. ОХП бол бодит амьдарлыг ажиглан түүний бүтэц үйл ажиллагааны давуу талуудыг програмчлалд оруулж ирсэн зүйл юм.Жишээ нь хүн обьект хандалтат сэтгэлгээтэй байдаг. Тэр сэтгэлгээр програмчлалд оруулах боломжийг ОХП олгодог.Амьдралд төрөл бүрийн үйл ажиллагаа нь өөр олон сонирхолтой зарчим дээр тулгуурлан явагдаж байдгийн нэг нь удамшил юм. Жишээ нь ямар ч хүн дор хаяж дараах чадваруудыг төрөхдөө эзэмшсэн байдаг: төрөх, үхэх, хооллох, унтах, харах, сонсох, амьсгалах, айх гэх мэт. Эдгээр шинж чанар бүх амьтадад бий. Хүн өөрөө амьтан болохоор амьтан бүгдэд байх энэхүү үндсэн чадваруудыг гендээ удамшуулан авсан байдаг. Тийм болохоор хүн бүрт хэрхэн амьсгалах, унтах, сонсох гэх мэт зүйлийг заах шаардлаггүй байдаг. Ийм зорилгоор бурхан хүнийг бүтээсэн бөгөөд үүнийгээ ашиглан бүх амьтад маш хялбархан бүтээсэн байна. Эндээс төрөл бүрийн амьтан бүрт дээрх амьтдын үндсэн үйлдлүүдийг тус бүрийнх нь генд нь дахин бичих шаардлага алга болно. Ингэснээр нэг бичсэн кодыг бусдад дахин давтан бичих зүйлийг хааж өгсөнөөр улам уян хатан байна гэсэн үг юм. Харин амьтад өөрсдийнх нь онцлог үйлдлүүдийг нэмж генд нь суулгаад ажил дуусна. Жишээ нь баавгайд бол өвөлдөө ичих үйлдлийг генд нь хийнэ. Энэ давуу талыг програмчлах боломжыг объект хандалтат хэлүүдэд суулгаж өгсөн бөгөөд “Удамшил” хэмээх ойлголтоор хэрэгжүүлдэг. ОХП –н хэлэнд нэг класс нөгөөгөөс удамшин түүний гишүүн өгөгдөл, функцүүдийг хүлээн авах боломжтой байдаг.Нэг классыг нөгөөгөөс удамшуулахдаа **extends** түлхүүр үгийг хэрэглэнэ. Удамшлын харьцаанд удамшиж байгаа классыг дэд класс буюу - sub класс, удамшуулж байгаа классыг нь эх класс буюу - super класс гэж нэрлэдэг. Тиймээс өмнөх жишээнд А класс нь sub класс, B класс нь А классын super класс нь юм.Удамшиж буй класс нь эх классынхаа гишүүн өгөгдөл болон гишүүн функцүүдийг удамшуулан хүлээн авна. Өөрөөр хэлбэл удамшиж орж ирсэн өгөгдөл функцүүдийг яг өөрийнх нь дотор хийж өгсөн мэт ашиглах боломжтой. Зарим төрлийн гишүүн өгөгдөл болон функцүүд нь удамшдаггүй. Энэ талаар өгөгдлийн далдлалт буюу **( data hiding )**тухай лекцээр авч үзсэн.

Жишээ нь А классаас B классыг удамшуулбал дараах байдалтай бичигдэнэ :

**class** A {

**int** i, j;

**void** showij() {

System.***out***.println("i and j: " + i + " " + j);

}

}

**class** B **extends** A {

**int** k;

**void** showk() {

System.***out***.println("k: " + k);

}

**void** sum() {

System.***out***.println("i+j+k: " + (i+j+k));

}

}

**class** SimpleInheritance {

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

A superOb = **new** A();

B subOb = **new** B();

superOb.i = 10;

superOb.j = 20;

System.***out***.println("Contents of superOb: ");

superOb.showij();

System.***out***.println();

subOb.i = 7;

subOb.j = 8;

subOb.k = 9;

System.***out***.println("Contents of subOb: ");

subOb.showij();

subOb.showk();

System.***out***.println();

System.***out***.println("Sum of i, j and k in subOb:");

subOb.sum();

}

}

**Үр дүн:**

Contents of superOb:

i and j: 10 20

Contents of subOb:

i and j: 7 8

k: 9

Sum of i, j and k in subOb:

i+j+k: 24

**Удамшил ба гишүүдэд хандах нь**

Хүү класс нь өөрийн эх классын бүх гишүүдийг агуулдаг боловч эх классд private –аар зарлагдсан гишүүдэд хандаж чаддаггүй. Жишээ нь:

**class** A {

**int** i;

**private** **int** j;

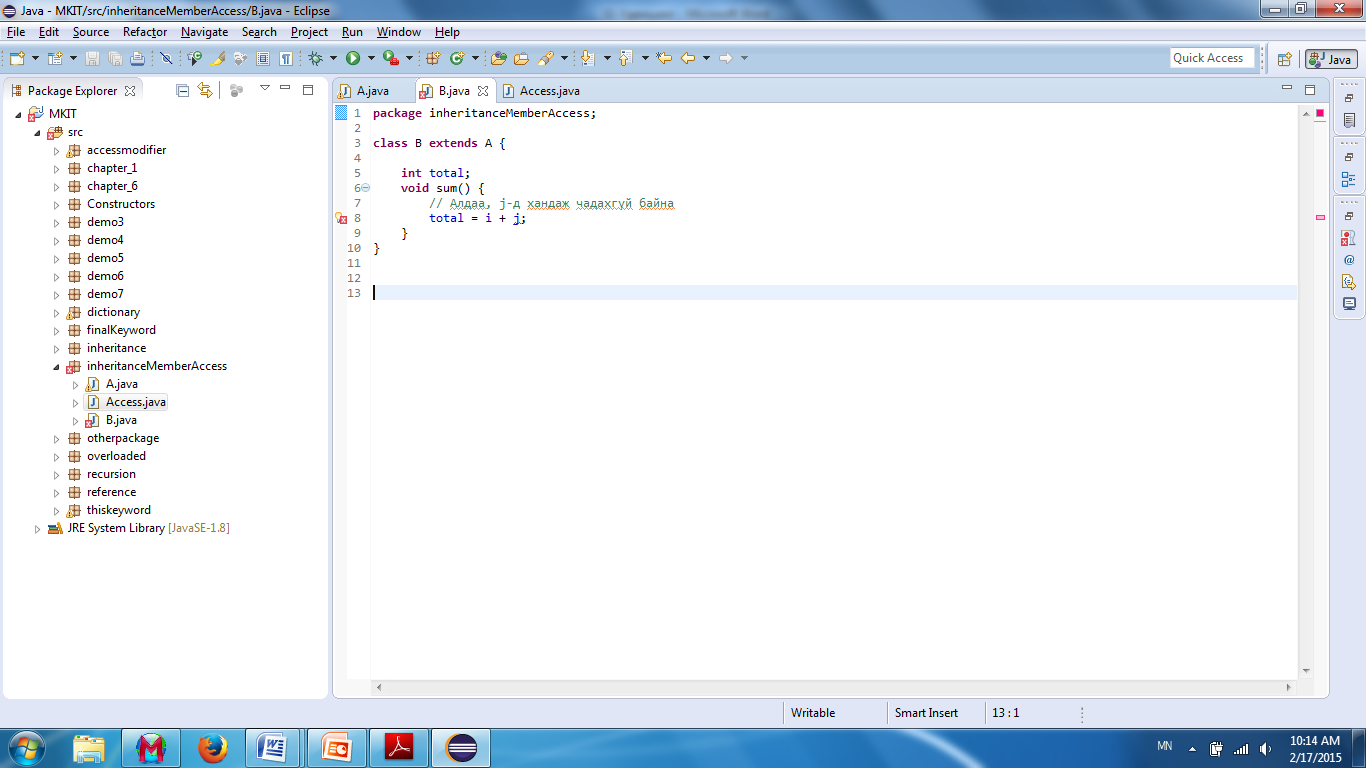
**void** setij(**int** x, **int** y){

i = x;

j = y;

}

}



**class** Access {

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

B subOb = **new** B();

subOb.setij(10, 12);

subOb.sum();

System.***out***.println("Total is " + subOb.total);

}

}

Дээрх програм нь ажиллахгүй яагаад гэвэл B классын sum ( ) функц дотор j хувьсагчийг хэргэлсэн байгаа болхоор.

**class** Box {

**double** width;

**double** height;

**double** depth;

// construct clone of an object

Box(Box ob) { // pass object to constructor

width = ob.width;

height = ob.height;

depth = ob.depth;

}

// constructor used when all dimensions specified

Box(**double** w, **double** h, **double** d) {

width = w;

height = h;

depth = d;

}

// constructor used when no dimensions specified

Box() {

width = -1; // use -1 to indicate

height = -1; // an uninitialized

depth = -1; // box

}

// constructor used when cube is created

Box(**double** len) {

width = height = depth = len;

}

// compute and return volume

**double** volume() {

**return** width \* height \* depth;

}

}

**class** BoxWeight **extends** Box{

**double** weight; // weight of box

// constructor for BoxWeight

BoxWeight(**double** w, **double** h, **double** d, **double** m) {

width = w;

height = h;

depth = d;

weight = m;

}

}

**class** DemoBoxWeight {

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

BoxWeight mybox1 = **new** BoxWeight(10, 20, 15, 34.3);

BoxWeight mybox2 = **new** BoxWeight(2, 3, 4, 0.076);

**double** vol;

vol = mybox1.volume();

System.***out***.println("Volume of mybox1 is " + vol);

System.***out***.println("Weight of mybox1 is " + mybox1.weight);

System.***out***.println();

vol = mybox2.volume();

System.***out***.println("Volume of mybox2 is " + vol);

System.***out***.println("Weight of mybox2 is " + mybox2.weight);

}

}

**Үр дүн:**

Volume of mybox1 is 3000.0

Weight of mybox1 is 34.3

Volume of mybox2 is 24.0

Weight of mybox2 is 0.076

Эх классын хувьсагчид нь хүү классынхаа объектуудыг зааж болдог.

**class** RefDemo {

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

BoxWeight weightbox = **new** BoxWeight(3, 5, 7, 8.37);

Box plainbox = **new** Box();

**double** vol;

vol = weightbox.volume();

System.***out***.println("Volume of weightbox is " + vol);

System.***out***.println("Weight of weightbox is " + weightbox.weight);

System.***out***.println();

// assign BoxWeight reference to Box reference

plainbox = weightbox;

vol = plainbox.volume(); // OK, volume() defined in Box

System.***out***.println("Volume of plainbox is " + vol);

/\* Доорх кодчилолд алдаа гарсан байна яагаад гэвэл

plainbox -д weight гишүүнийг тодорхойлоогүй \*/

// System.out.println("Weight of plainbox is " + plainbox.weight);

}

}

**Super заагч**

Удамшлын харьцаанд удамшиж байгаа классыг дэд класс буюу – sub class, удамшуулж байгаа классыг нь эх класс буюу – super class гэж нэрлэдэг. Удамшиж буй класс нь эх классынхаа гишүүн өгөгдөл болон гишүүн функцүүдийг удамшуулан хүлээн авна. Өөрөөр хэлбэл удамшиж орж ирсэн өгөгдөл функцүүдийг яг өөрийнх нь дотор хийж өгсөн мэт ашиглах боломжтой. Энэ нь хоёр хэрэглээтэй **эхний хэрэглээ нь**: эх классын байгуулагчыг хүү класс дотороос дуудах мөн функцийг нь дуудахад **хоёр дахь хэрэглээ нь:** удамшин орж ирсэн өгөгдлийг ялган харуулахад хэрэглэгддэг.

**Эхний хэрэглээний бичигдэх хэлбэр:**  *super ( arg-list );*

**class** Box { **double** width; **double** height;  
 **double** depth;  
 Box(Box ob) { // pass object to constructor  
 width = ob.width;  
 height = ob.height;  
 depth = ob.depth; }  
 Box(**double** w, **double** h, **double** d) {  
 width = w;  
 height = h;  
 depth = d; }  
 Box() {  
 width = -1; // use -1 to indicate  
 height = -1; // an uninitialized  
 depth = -1; // box }  
 Box(**double** len) {  
 width = height = depth = len; }  
 **double** volume() {  
 **return** width \* height \* depth; }  
}  
**class** BoxWeight **extends** Box{   
 **double** weight; // weight of box  
 BoxWeight(**double** w, **double** h, **double** d, **double** m) {  
 **super**(w, h, d); // эх классын байгуулагчийг дуудаж байна  
 weight = m;  
 }  
}

**Хоёр дахь хэрэглээний бичигдэх хэлбэр:** *super.member*

Энэ нь хүү класс дотроос эх классын гишүүн өгөгдөлд хандан утга олгоход хэрэглэгдэнэ. Өөрөөр хэлбэл хүү классын хувьсагч нь эх классын хувьсагчийн нэртэй ижил байх тохиолдолд тэдгээрийг хооронд нь ялгахын тулд хэрэглэгдэнэ.

**class** A { **int** i; }

**class** B **extends** A{

**int** i; // this i hides the i in A

B(**int** a, **int** b) {

**super**.i = a; // энд эх классын гишүүн рүү хандаж байна.

i = b; // i in B }

**void** show() {

System.***out***.println("i in superclass: " + **super**.i);

System.***out***.println("i in subclass: " + i); }

}

**class** UseSuper {

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

B subOb = **new** B(1, 2);

subOb.show();

}

}

**Үр дүн:** i in superclass: 1 i in subclass: 2

**Функцийн дахин тодорхойлолт**

Эх классд тодохойлсон функцийг нь түүнээс удамшсан хүү классд яг ижил нэртэй, аргументтайгаар тодорхойлсон байвал үүнийг функцийн дахин тодорхойлолт гэнэ. Өөрөөр хэлбэл удамшиж ирсэн функцийг өөрчлөн дахин зарлахыг функцийн дахин тодорхойлолт (**function override**) гэнэ.

Бид өмнө нь хэт ачаалалттай функц (**function overloading**) гэж юу болохыг үзсэн. Энэ нь нэг класс дотор **ижил нэртэй** гэхдээ **ялгаатай төрлийн аргументтай** функцүүдийг хэлнэ.

Дахин тодорхойлогдсон функцүүд нь эх класс дотор болон түүнээс удамшсан хүү класс дотор байрлах ижил нэртэй мөн ижил аргументтай функцүүд юм.

**class** A {

**int** i, j;

A(**int** a, **int** b) {

i = a;

j = b;

}

// display i and j

**void** show() {

System.***out***.println("i and j: " + i + " " + j);

}

}

**class** B **extends** A{

**int** k;

B(**int** a, **int** b, **int** c) {

**super**(a, b);

k = c;

}

// display k – this overrides show() in A

**void** show() {

System.***out***.println("k: " + k);

}

}

**class** Override {

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

B subOb = **new** B(1, 2, 3);

subOb.show(); // this calls show() in B

}

}

**Үр дүн:** k: 3

**Динамик функцийг илгээх буюу Полиморизмын тухай ойлголт**

Энэ ойлголт нь удамшлын үед дахин тодорхойлогдсон фунцүүдийн хувьд яригддаг Жава хэлний гол ойлголтуудын нэг юм. Энэ нь удамшлын нэг төвшинд байгаа классуудыг бүгдийг нь эх классын заагч ашиглан хандах арга юм. Энэ нь яг програм ажилж байх явцад дахин тодорхойлогдсон функцийг дуудах механизм юм. Үүнийг хожуу холболт буюу виртуаль функц ч гэж бусад хэлэнд ярьдаг.

**class** A {

**void** callme() {

System.***out***.println("Inside A's callme method");

}

}

**class** B **extends** A{

// override callme()

**void** callme() {

System.***out***.println("Inside B's callme method");

}

}

**class** C **extends** A{

// override callme()

**void** callme() {

System.***out***.println("Inside C's callme method");

}

}

**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

}

}

**Үр дүн**

Inside A's callme method

Inside B's callme method

Inside C's callme method

**Абстракт класс**

Абстракт гэдэг нь хийсвэр гэсэн үг юм. Энгийн классаас ялгагдах хоёр онцлог байдаг.

* Абстракт классын объектийг үүсгэж болдоггүй.
* Абстракт класс нь дотроо абстракт функцүүдийг агуулж болдог. Абстракт функц гэдэг нь кодгүй зөвхөн зарлалт хэлбэрээр байдаг функцийг хэлнэ. Өөрөөр хэлбэл интерфэйс (энэ ойлголтыг хожим үзнэ)доторх функцүүд шиг.

Классыг зарлахдаа өмнө нь abstract гэсэн түлхүүр үг тавьбал уг класс абстракт класс болно. Абстракт классыг ашиглахдаа түүнээс удамшсан класс бичдэг. Хэрэв түүнээс удамшсан нэг ч класс байхгүй бол түүнийг абстракт байлгах шаардлаггүй юм. Хэрэв абстракт классаас дахин өөр абстракт класс удамшвал уг удамшсан класс дотор эх абстракт классын абстракт функцийг имплемент хийх албагүй. Учир нь нэгэнт хоёулаа абстракт тул абстракт функцүүд нь кодгүйгээрээ байх боломжтой. Абстракт классаас удамшсан класс нь түүний абстракт функцүүдийг имплемент хийх ёстой. Абстракт классын обьектийг програмист өөрөө үүсгэж болохгүй ч абстракт класст байгуулагч функц хийж өгч болдог. Энэ байгуулагч ямар хэрэгтэй вэ? Обьектийг нь үүсгэхгүй шүү дээ? Удамшсан классын обьект үүсэх үед түүний эх классын обьектууд эхлээд үүсдэгийг санана уу. Тиймээс абстракт классаас удамшсан энгийн классын обьектийг үүсгэхэд уг абстракт классын байгуулагч дуудагддаг. Хийсвэр обьектуудыг хийх өөр нэг арга нь абстракт класс юм. Ер нь ямар нэг зүйл хийсвэр байх тусам түүний гүйцэтгэж буй үйл ажиллагаанууд нь тодорхой зүйл хийдэггүй, харин тийм ийм зүйл хийдэг байх ёстой гэсэн “дүрэм” хэлбэрээр байдаг. Абстракт класс нь интерфэйсийн адил нийлмэл удамшлыг хэрэгжүүлж чадахгүй. Учир нь Java-д нэг класс нэг л классаас удамшдаг. Тэгвэл абстракт классыг хэрэглэх шалтгаан юу юм бэ? Абстракт классын оронд зүгээр энгийн класс ашиглачихвал болохгүй гэж үү? Хариулт нь : Болно. Гэхдээ ингэвэл нөгөө л уян хатан бус, алдаа гарах боломжтой, хөшүүн байдлаар бичигдэх боломжтой болж ирдэг.

Бидэнд одоо дараах гурван зүйлийн аль нэгийг хэрэглэн өгөгдлийн шинэ төрлийг бүтээх боломж байна :

* Энгийн класс
* Абстракт класс
* Интерфэйс

Энэ гурвыг хийсвэр бодитойгоор нь зэрэгцүүлэн харвал дараах байдалтай байна. Өөрөөр хэлбэл интерфэйс нь 100% хийсвэр өгөгдлийн төрөл бол абстракт класс нь бодитой болон хийсвэр ойлголтууд холилдсон, харин энгийн класс нь 100% бодитой үйл ажиллагаануудыг агуулна.

Энгийн класс

Абстракт класс

Интерфейс

Бодитой Хийсвэр

