**Функц блон классыг гүнзгийрүүлэх нь**

**Хэт ачаалалттай функц**

Класс дотор ижил нэртэй функцүүд байж болдог ба ийм функцүүд нь аргументаараа ялгагддаг. Ийм функцүүдийг хэт ачаалалттай ( overloaded ) функцүүд гэдэг. Ижил нэртэй ижил аргументтай функцүүд байж болохгүй.

**class** OverloadDemo {

**void** test() {

System.***out***.println("No parameters");

}

**void** test(**int** a) {

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

}

**void** test(**int** a, **int** b) {

System.***out***.println("a and b: " + a + " " + b);

}

**double** test(**double** a) {

System.***out***.println("double a: " + a);

**return** a\*a;

}

}

**class** Overload {

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

OverloadDemo ob = **new** OverloadDemo();

**double** result;

ob.test();

ob.test(10);

ob.test(10, 20);

result = ob.test(123.25);

System.***out***.println("Result of ob.test(123.25): " + result);

}

}

**Үр дүн:**No parameters  
a: 10  
a and b: 10 20  
double a: 123.25  
Result of ob.test(123.25): 15190.5625  
----------------------------------------------------------------------------------  
**class** OverloadDemo {  
 **void** test() {  
 System.***out***.println("No parameters");  
 }  
 **void** test(**int** a, **int** b) {  
 System.***out***.println("a and b: " + a + " " + b);  
 }  
 **void** test(**double** a) {  
 System.***out***.println("Inside test(double) a: " + a);  
 }  
}

**class** Overload {

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

OverloadDemo ob = **new** OverloadDemo();

**int** i = 88;

ob.test();

ob.test(10, 20);

ob.test(i); // this will invoke test(double)

ob.test(123.2); // this will invoke test(double)

}

}

**Үр дүн:**No parameters  
a and b: 10 20  
Inside test(double) a: 88.0  
Inside test(double) a: 123.2  
----------------------------------------------------------------------------------**Хэт ачаалалттай байгуулагч функц**  
**class** Box {  
 **double** width;  
 **double** height;  
 **double** depth;  
 Box(**double** w, **double** h, **double** 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** OverloadCons {  
 **public** **static** **void** main(String args[]) {  
 Box mybox1 = **new** Box(10, 20, 15);  
 Box mybox2 = **new** Box();  
 Box mycube = **new** Box(7);  
 **double** vol;  
 vol = mybox1.volume();  
 System.***out***.println("Volume of mybox1 is " + vol);  
 vol = mybox2.volume();  
 System.***out***.println("Volume of mybox2 is " + vol);  
 vol = mycube.volume();  
 System.***out***.println("Volume of mycube is " + vol);  
 }  
}  
**Үр дүн:**Volume of mybox1 is 3000.0  
Volume of mybox2 is -1.0  
Volume of mycube is 343.0

**Функцийн параметр дамжуулалт**

Жава хэлэнд хоёр төрлийн параметр ( аргумент ) дамжуулалт байдаг. Утгаар дамжуулалт (pass by value), заалтаар дамжуулалт ( pass by reference ) гэсэн.

**Утгаар дамжуулалт:** Дамжуулагдаж байгаа параметрийн утгыг хуулаад функцэд дамжуулана. Өгөгдлийн үндсэн төрлүүд болох (byte, short, int, long, double, float,boolean, char) гэсэн төрлүүд утгаар дамжигддаг.

**Заалтаар дамжуулалт:** Дамжуулж байгаа өгөгдлийн эх хувийг дамжуулана. Дуудагдсан функц энэ эх хувийг өөрчилж болно. Утгаар дамжуулахыг бодвол илүү хурдан байдаг. Жава хэлэнд бүх объектууд заалтаар дамжигддаг. Жава хэлэнд массивыг объект гэж үздэг ба массив нь заалтаар дамжих юм.

**Утгаар дамжуулалтын жишээ:**

**class** Test {

**void** meth(**int** i, **int** j) {

i \*= 2;

j /= 2;

}

}

**class** CallByValue {

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

Test ob = **new** Test();

**int** a = 15, b = 20;

System.***out***.println("a and b before call: " +

a + " " + b);

ob.meth(a, b);

System.***out***.println("a and b after call: " +

a + " " + b);

}

}

**Үр дүн:**

a and b before call: 15 20

a and b after call: 15 20

**Заалтаар дамжуулалтын жишээ:**

**class** Test {

**int** a, b;

Test(**int** i, **int** j) {

a = i;

b = j;

}

// pass an object

**void** meth(Test o) {

o.a \*= 2;

o.b /= 2;

}

}

**class** PassObjRef {

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

Test ob = **new** Test(15, 20);

System.***out***.println("ob.a and ob.b before call: " + ob.a + " " + ob.b);

ob.meth(ob);

System.***out***.println("ob.a and ob.b after call: " + ob.a + " " + ob.b);

}

}

**Үр дүн:**

ob.a and ob.b before call: 15 20

ob.a and ob.b after call: 30 10

**Рекурс:**

Жава хэл нь Рекурсыг дэмждэг юм. Рекурс гэдэг нь функц нь дотроосоо өөрийгөө дуудахыг хэлнэ. Функцийн биед функцээс гарах нөхцлийг хийж өгөхгүй бол хязгааргүй цикль болно. Энэ нь функц дуудах бүрт утгуудаа **стект** хадгалах ба функцээс гарах нөхцөл биелэх үед стекээс хамгийн сүүлд орсон нь эхэлж гарах зарчмаар хадгалагдсан утгууд гарч дуусна.

**Стек гэдэг нь:** тооцоолох явцад өргөн хэрэглэгддэг мэдээлэл хадгалах тогтолцооны нэг нь стекийн зохион байгуулалт юм.

**class** Factorial {

// this is a recursive method

**int** fact(**int** n) {

**int** result;

**if**(n==1) **return** 1;

result = fact(n-1) \* n;

**return** result;

}

}

**class** Recursion {

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

Factorial f = **new** Factorial();

System.***out***.println("Factorial of 3 is " + f.fact(3));

System.***out***.println("Factorial of 4 is " + f.fact(4));

System.***out***.println("Factorial of 5 is " + f.fact(5));

}

}

**Үр дүн:**

Factorial of 3 is 6

Factorial of 4 is 24

Factorial of 5 is 120

**Хандалтын удирдлага:**

Объект хандалтын зарчмын нэг чухал тулгуур ойлголт нь өгөгдлийн далдлалт ( объектийн биеэ даасан байдал ) болох тухай ярилцаж байсан. Өгөгдлийн далдлалт гэдэг нь объектууд өөрсдийн дотоод мэдлэг, үйлдлийг бусдадаа хандагдахгүй нууц байдалд дотроо хадгалагдана гэсэн санаатай. Өөрөөр хэлбэл объектууд нь бусад ( бусад объектууд ) хэрэгтэй үйлдлүүдээ бусдадаа өгөөд зөвхөн өөртөө л хамаатай өгөгдөл үйлдүүдээ бусдаас нууж мэдэгдэхгүй байлгана гэсэн үг. Тиймээс амжилттай систем бүрээхийн гол эхлэл нь өгөгдлийн зөв далдлалт байна гэсэн үг. Өгөгдлийн далдлалтыг ОХП –үүдэд төрөл бүрээр хэрэгжүүлдэг боловч ихэнх нь **хандалтын тодорхойлогч – access modifier** ашигладаг. Хандалтын тодорхойлогч гэж юу вэ ?

Энэ нь объектийн ямар өгөгдөл, үйлдэл нь бусдад ямар хэмжээгээр хандагдахгүй, харагдахгүй буюу далд байхыг тэмдэглэх арга юм. Өөрөөр хэлбэл гишүүн өгөгдөл бүрийг бусдад яаж хандагдахыг хандалтын тодорхойлогч зааж өгдөг. Жава хэлний хандалтын тодорхойлогчид бол **public, private, protected** юм. Үүн дээр нэмэгдээд **package** буюу **default** хандалт орж болно.

Private

Package буюу default Сумны чиглэлд өгөгдлийн далдлалт өснө.

Protected

Public

Классын гишүүн өгөгдөл, функцийн хандалтыг зааж өгөхдөө дээрх түлхүүр үгсийг уг өгөгдөл функцийн зарлалтын өмнө нь тавьж өгдөг.

**Private хандалт:** зөвхөн уг объект өөрөө л хандаж ашиглаж чадах хамгийн хамгаалалттай өгөгдөл юм.

**Package хандалт:** нэг пакеж дотор байгаа бусад бүх классууд хандаж ашиглаж чадах өгөгдөл. Хэрэв хандалтын тодорхойлогчийг нь бичээгүй бол уг өгөгдөл package хандалттай өгөгдөл болно. Өөрөөр хэлбэл өмнөх жишээний ondor гэсэн өгөгдөл нь пакэж хандалттай байна.

**Protected хандалт:** Пакеж хандалттай адилхан. Гэхдээ өөр пакеж дотор удамшуулж хэрэглэвэл хандагддаг.

**Public хандалт:** програмын дурын хэсгээс ( бүх пакеж дотроос ) бүх объектууд энэ өгөгдөлд чөлөөтэй хандаж чадна.

**Private хандалт:** Доорх жишээнд бол зөвхөн тухайн классын time гишүүнд зөвхөн тухайн классын гишүүн функцүүд хандаж байна.

**class** Clock {

**private** **long** time = 0;

**public** **long** getTime() {

**return** **this**.time;

}

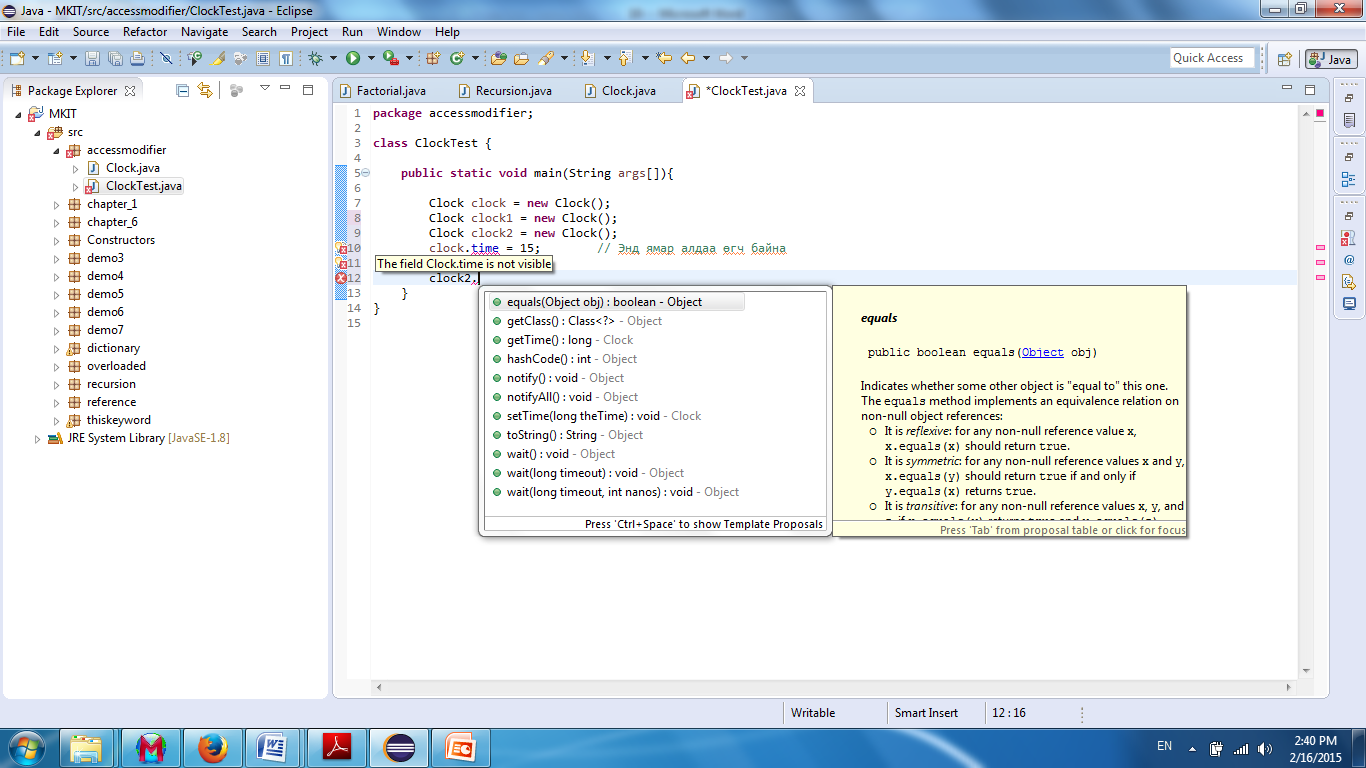
**public** **void** setTime(**long** theTime) {

**this**.time = theTime;

}

}

Доорх жишээнд бүр алдаа өгөөд гишүүд нь гарч ирэхгүй байна.



Тухайн классын гишүүн өгөгдөлд хандах гэсэн чинь болохгүй алдаа гарч байна. Яагаад гэвэл **private** хандалттай гишүүд рүү зөвхөн тухайн классын гишүүд хандана гэсэн үг. Тэгвэл тухайн классын private хандалттай гишүүд рүү яаж хандах вэ, яаж утга олгох вэ гэсэн асуулт гарч ирж байна. Энэ зүйлийг шийдэх арга байгаа юу ? байгаа. Юу вэ гэхээр тухайн классын гишүүн фунцийг тодорхойлж өгөхдөө public хандалттай үйлчилгээ тавьж өгөх хэрэгтэй. Энэ нь манай жишээнд **public** **long** getTime(),**public** **void** setTime() гэсэн үйлчилгээ тавьж өгсөн нь харагдаж байна.

**Package хандалт:**

Доорх жишээнд ClockPackage, ClockTestPack гэсэн хоёр класс accessmodifier гэсэн нэг пакежд байгаа болохоор хандаж болж байна.

**package** accessmodifier;

**class** ClockPackage {

//this is package (deafult)

**long** time = 0;

}

**package** accessmodifier;

**class** ClockTestPack {

ClockPackage clock = **new** ClockPackage();

**public** **long** readClock(){

**return** clock.time;

}

}

---------------------------------------------------------------------------------------------------------------------------

Доорх жишээнд ClockPackage, ClockTestPack гэсэн хоёр класс маань өөр өөр пакежд байгаа тул хандагдахгүй байна.

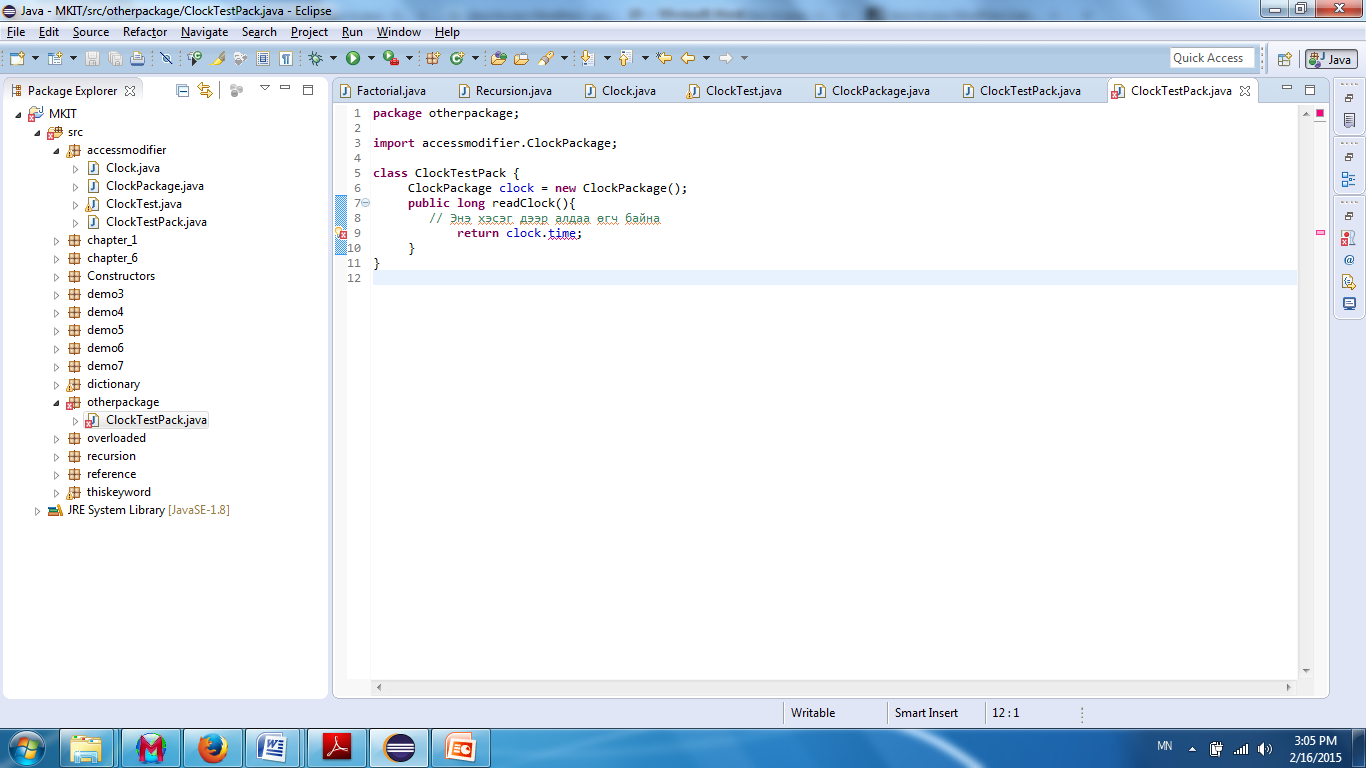
**package** accessmodifier;

**public** **class** ClockPackage {

//this is package (deafult)

**long** time = 0;

}



**Protected хандалт:** Удамшилын үед хандаж болж байна.

**package** accessmodifier;

**class** ClockProtected {

// time in milliseconds

**protected** **long** time = 0;

}

**package** accessmodifier;

**class** CloskTestProtected **extends** ClockProtected{

**public** **long** getTimeInSeconds(){

**return** **this**.time / 1000;

}

}

---------------------------------------------------------------------------------------------------------------------------

Харин одоо удамшаагүй үед хандаж болохгүй жишээг авч үзье.

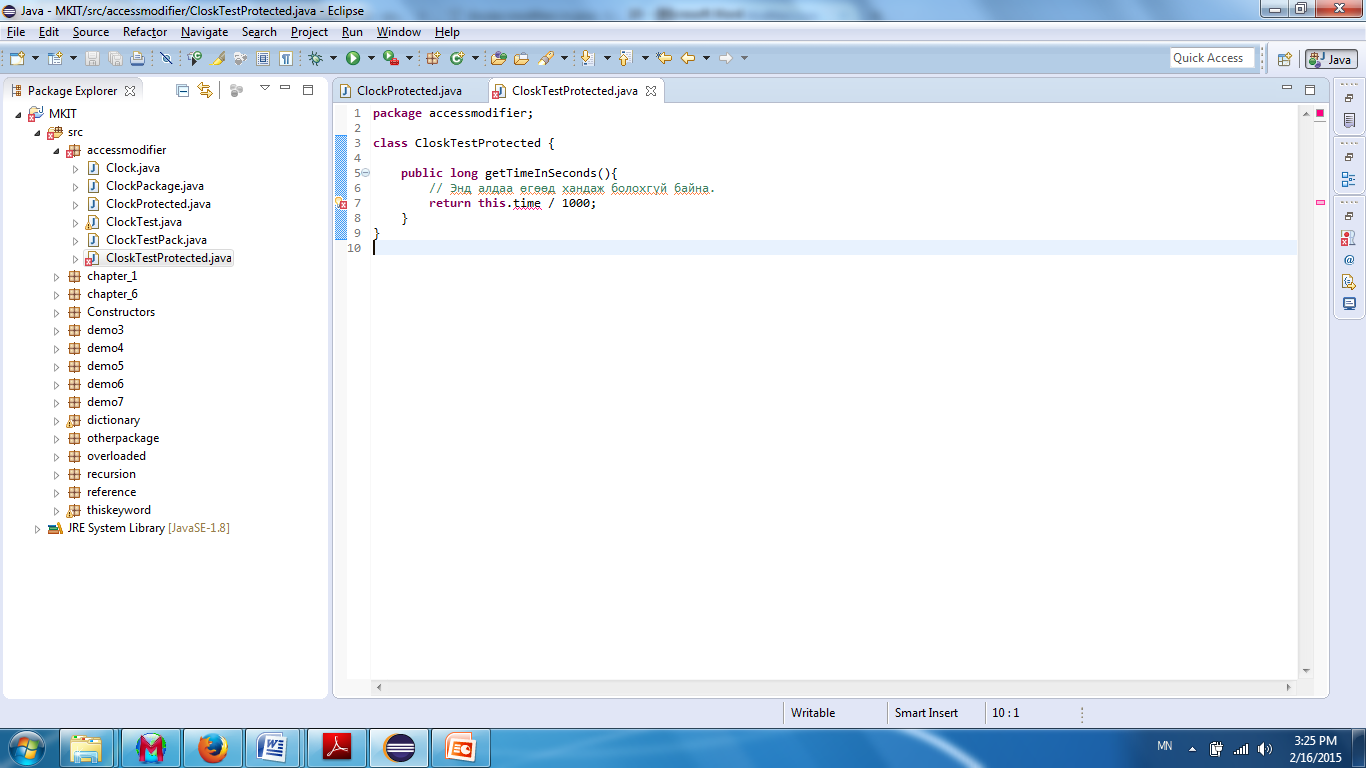
**package** accessmodifier;

**class** ClockProtected {

// time in milliseconds

**protected** **long** time = 0;

}



**Public хандалт:** Энэ нь бүх хэсгээс хандаж болно гэсэн үг.

**package** accessmodifier;

**class** ClockPublic {

**public** **long** time = 0;

}

**package** accessmodifier;

**class** ClockTestPublic {

ClockPublic clcok = **new** ClockPublic();

**public** **long** readClock(){

**return** clcok.time;

}

}

За ингээд дээрх бүгдийг дүгнэж хэлхэд эдгээр 4 хандалтыг яг ямар үед хэрэглэх вэ гэхээр. **Public** хандалтыг аль болох хэрэглэхгүй байх нь зүгээр. Энэ нь хамгийн энгийн хандалт буюу хандалтын ямар нэг хязгаар байхгүй. Энэ хандалтыг ОХП –д хэрэглэдэггүй. **Package** хандалт тйим ч сайн биш програмистууд энэ хандалтыг тоолгүй хэрэглэдэг. Нэг пакежд байгаа бүх классууд бие биенийхээ ийм хандалттай өгөгдлүүд рүү хандаж ашиглаж чадна. Хэрэв гишүүн өгөгдөл, функцүүдийн өмнө ямар нэг түлхүүр үг бичээгүй бол энэ өгөгдөл, функц нь пакеж хандалттай болдог. **Protected** хандалтыг зайлшгүй л биш бол хэрэглэхгүй байсан нь дээр. Энийг бас өөрөөр өөр пакежд байгаа класс руу удамшдаг пакеж хандалттай өгөгдлүүд гэж хэлж болно. **Private** хандалт хамгийн зөв програмчлал эндээс эхэлнэ. Энэ нь хамгийн далд хэлбэр юм. Ийм өгөгдлүүд классын гаднаас хандагдахгүй, удамшихгүй. Ийм өгөгдлийн утгыг зөвхөн тухайн объект өөрөө л мэднэ. Өөрөө л өөрчилж чадна. Үүнээс хойш классын бүх өгөгдлийг **private** –аар зарлаж хэвших хэрэгтэй.Тиймээс классын гишүүн өгөгдлүүдийн хувьд public, package, protected хандалтын тодорхойлогчид бараг хэрэглэгддэггүй.

**Санах дүрэм:** Классыг бүтээхдээ бүх гишүү хувьсагчдийг **private** хандалттайгаар тавьна. Аажмаар бусад классуудыг бичиж байх явцад **private** өгөгдлийг ашиглах боломжгүй болж ирвэл (зайлшгүй шаардлагатай тохиолдолд) уг **private** хандлагыг эхлээд **package**, болохгүй бол **protected** болохгүй бол **public** гэж өөрчилнө.

Тэгэхээр энэ тодорхойлогчид хэрэггүй гэсэн үг үү? Эдгээрийг гишүүн функцүүдэд өргөн хэрэглэдэг. Гишүүн функцийн хандалтыг заахдаа уг функцийн зарлалтын өмнө хандалтын тодорхойлогчийг нь тавидаг. Хэрэв тавиагүй бол **package** хандалттай функц болдог. Обьектууд зөвхөн өөрийн өгөгдлүүдийг бусдаас далдлах нь хангалтгүй. Өөрийн дотоод үйл ажиллагаагаа бусдаас далд байлгах хэрэгтэй. Гишүүн функцүүдийг хандалтын хэлбэрээр нь хоёр ангилж болох юм. **Дотоод функцүүд** буюу **private** **функцүүд**. **Гадаад функцүүд** буюу **public, protected, package** хандалттай функцүүд гэж. Тэгэхээр классыг зохиохдоо гишүүн функцүүд нь ямар хандалттай байхыг хэрхэн мэдэх вэ гэсэн асуулт гарч ирж байна.

Тиймээс дотоод функцүүдийн шинж нь

* Зөвхөн өөрөө дотроо хийдэг үйлдлүүд
* Бусад обьектууд уг обьектоор хийлгэхийг шаардах албагүй функцүүд
* Үр дүн нь уг обьект дотор үлддэг үйлдлүүд

Гадаад функцүүдийн шинж нь

* Бусад обьектууд уг обьектоор гүйцэтгүүлэхийг хүсэх үйлдлүүд
* Үр дүн нь уг обьектоос гадна бий болж байдаг функцүүд

Гадаад функцүүд нь уг обьектийн бодит ертөнцөд үзүүлэх үйлчилгээнүүд нь юм.

* Үйлчилгээнүүд

Дотоод функцүүд нь уг үйлчилгээнүүдийг гүйцэтгэхэд шаардлагатай дотоод процессууд юм.

* Дотоод процессууд
* Обьектийн үйлчилгээнүүд нь дотроо дотоод процессуудаа ашиглана (ашиглахгүй үйлчилгээ ч байж болно).
* Дотоод процессууд нь өөр дотоод процессуудаа ашиглаж болно.

Обьектийн үйлчилгээнүүдийг нь (public, protected, package хандалттай функцүүд) ашиглаж бусад обьектууд уг обьектод мэдээлэл дамжуулах, ямар нэг боловсруулалт хийлгэж үр дүнг нь авдаг.

Харин нэг обьект өөр обьектийн дотоодод болж буй процессуудыг (private функцүүд) харах, ашиглах боломжгүй.

Ийнхүү обьект нь өөрийн өгөгдлүүд, дотоод процессуудаа бусдаас далд байлгах ба бусдад зөвхөн өөрийн хийдэг үйлчилгээнүүдийг л бусдад өгдөг.

* Энэ нь хар хайрцаг зарчим юм.
* Үүний ачаар програм нь: Уян хатан, Алдаанд тэсвэртэй,Өөрчлөлт сайжруулалтанд хялбар,Өргөтгөх боломж өндөр,Ойлгоход хялбар, Багаар ажиллах боломжтой систем болдог.

**Static түлхүүр үг:** Энэ түлхүүр үгийг классын гишүүн өгөгдөл болон функцүүдэд ашигладаг.

**package** accessmodifier;

**class** UseStatic {

**static** **int** *a* = 3;

**static** **int** *b*;

**static** **void** meth(**int** x) {

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

System.***out***.println("a = " + *a*);

System.***out***.println("b = " + *b*);

}

**static** {

System.***out***.println("Static block initialized.");

*b* = *a* \* 4;

}

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

*meth*(42);

}

}

**Үр дүн:**

Static block initialized.

x = 42

a = 3

b = 12

**class** StaticDemo {

**static** **int** *a* = 42;

**static** **int** *b* = 99;

**static** **void** callme() {

System.***out***.println("a = " + *a*);

}

}

**class** StaticByName {

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

StaticDemo.*callme*();

System.***out***.println("b = " + StaticDemo.*b*);

}

}

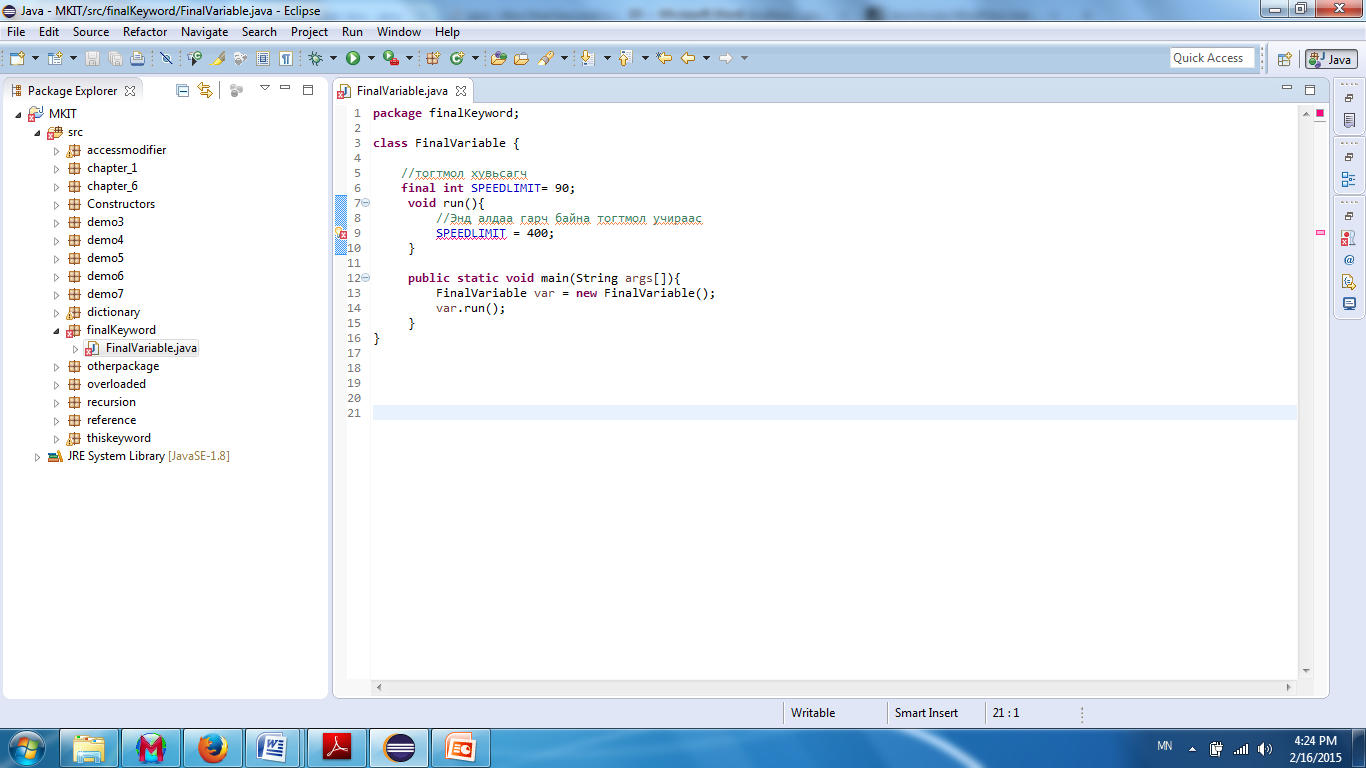
**Үр дүн:**

a = 42

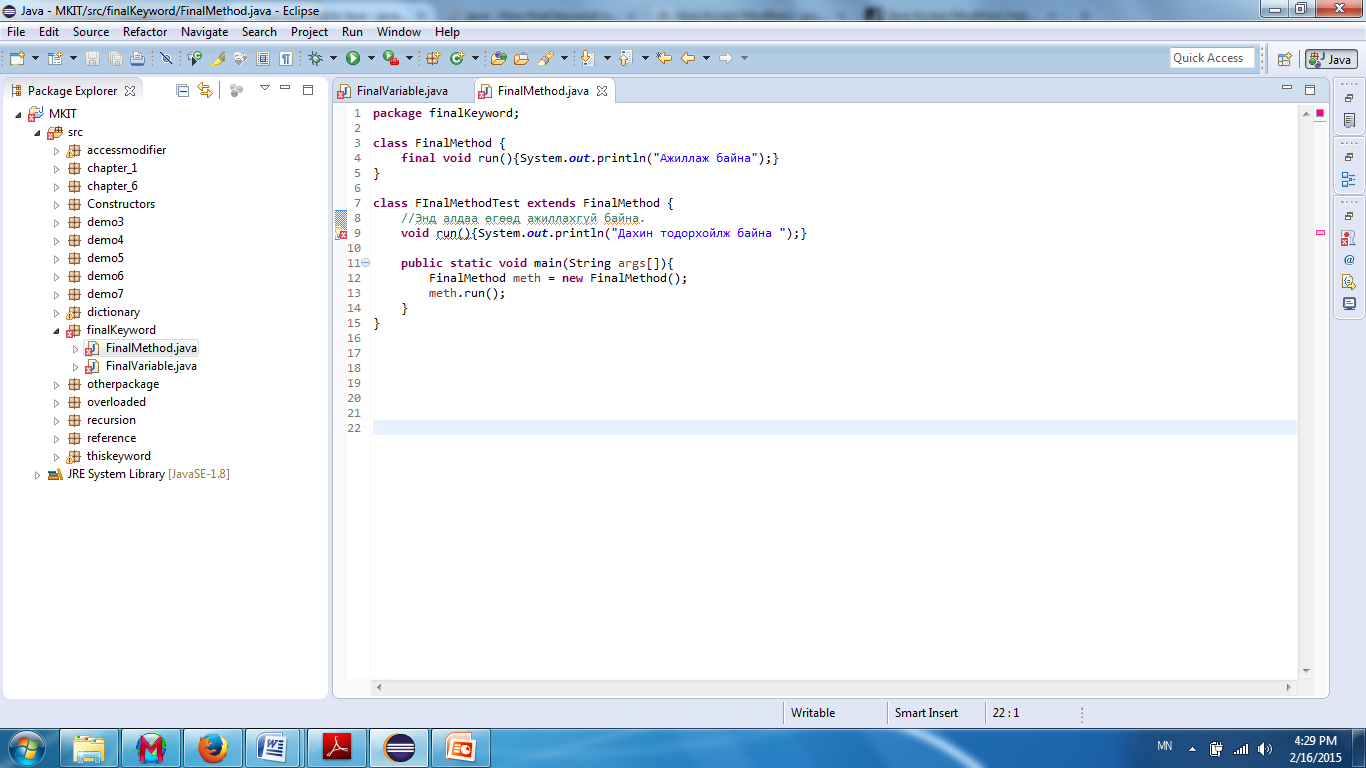
b = 99

**Final түлхүүр үг:** Бид жава хэлэнд програм бичиж байх явцад тогтмол өгөгдлийг зарлах шаардлага гардаг. Тэгвэл жава хэлэнд тогтмол өгөгдлийг зарлахдаа **final** түлхүүр үгийг ашигладаг. Тогтмол хувьсагчийн нэрийг бүх үсгийг нь томоор бичих бичигдээгүй дүрэм байдаг. Тогттол гишүүд нь анхны утгатай байх ба энэ утгыг нь өөрчилж болохгүй. Мөн энэ ойлголтыг класс болон функц дээр хэргэлэж болдог. Тогтмол классууд удамшдаггүй, тогтмол фунцүүд нь дахин тодорхойлогддоггүй.

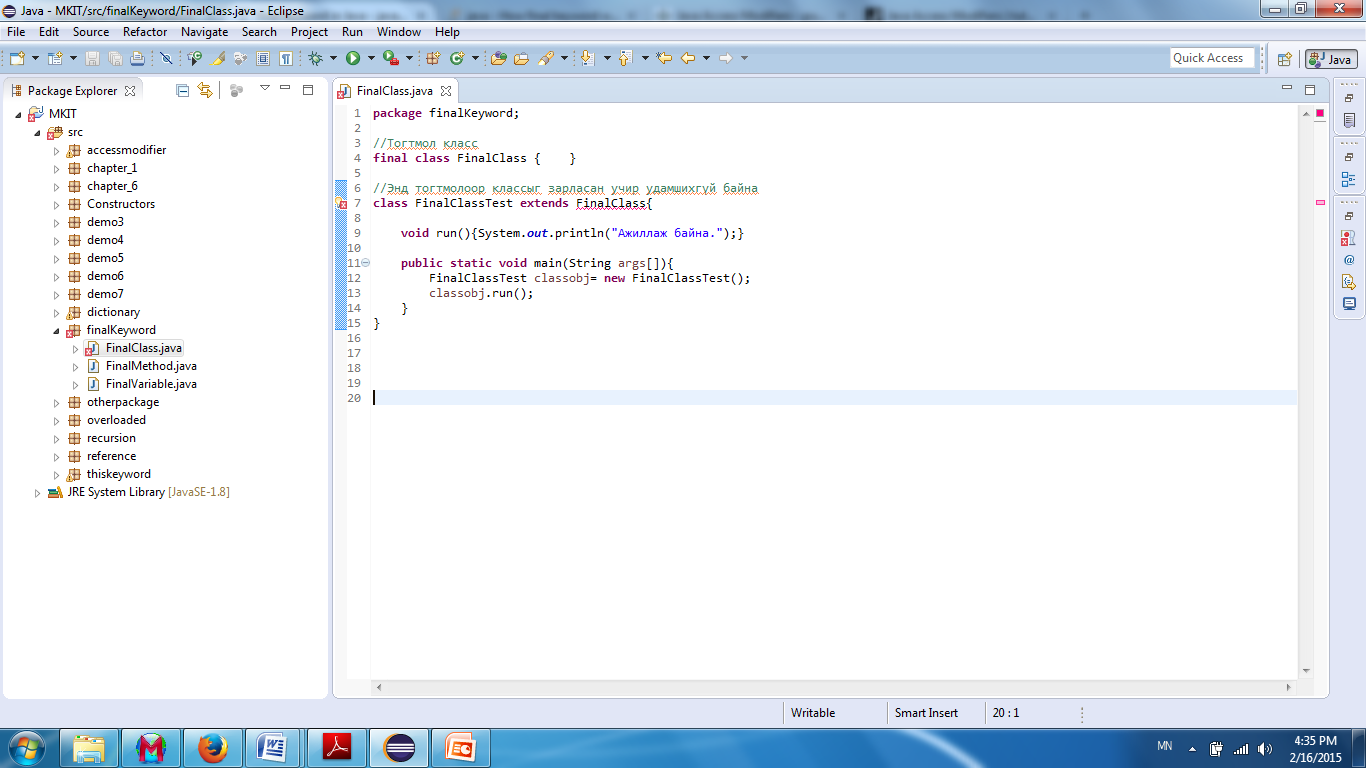
**Тогтмол хувьсагч:**



**Тогтмол функц:**



**Тогтмол класс:**

****