**Lambda илэрхийлэл**

JDK8 хувилбар дээр нэмэгдсэн олон ойлгголтын нэг нь бол энэ Lambda илэрхийлэл юм.

Бид аливаа програмыг бичихдээ бие биендээ хамааралтай програмчлалын аргаар хийх гэж байгаа бол өөрөөр хэлбэл төрөл бүрийн класс болон интерфейсийн функцүүдийг ашиглан бүрэн ажиллагаатай програм бичдэг. Тиймээс бид энэ аргыг ашиглах нь төдийлөн цаг авах ба үүнээс өөр хялбар аргаар хийх арга бол JDK8 дээр нэмэлтээр гарч ирсэн lambda хэмээх шинэ ойлголтыг ашиглан функцүүдийн хоорондын харилцааг амарчилж өгсөн ойлголтыг ашиглаж програм бичих илүү хялбар төд арга юм.

Swing бол java хэл ашигласан GUI бичих зориулалттай library юм. Ихэнхдээ бид өөрсдийн Swing component даа event оруулж өгхийг хүсдэг жишээ нь ActionEvent, KeyListenerгэх мэт. Энэхүү event юу хэлж байна гэхлээр бид 1 блок код функцлуу оруулж өгмөөр байна тэгээд ямар нэгэн event болход тэрхүү код ажилж баймаар байна. Гэхдээ Java 8 гархаас өмнө энийг хэрхэн хийдэг байсан бэ?

button.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent event) {

System.out.println(“Товчлуур дарагдлаа”);

}

});

Дээрх жишээнээс харахын бол бид товчлуур дарагдхад “Товчлуур дарагдлаа” гэсэн текст хэвлэхийг хүсэж байна. Залхуу програмистууд эсвэл хурдан кодоо бичих хүмүүст бол энэ бүх бичиглэл их цаг зав үрсэн ажил байдаг. Харин lambda ашиглан энэ кодчлолыг хэрхэн өөрчилж бичиж байна гэхээр.

button.addActionListener(event ->

System.out.println(“Товчлуур дарагдлаа”)

);

Дээрх жишээнээс хархад lambda –ш ашигласан жишээ код нь өмнө бичиж байсан кодчилолоос ямар хялбар нь харагдаж байна.

Lambda ихэнх хэрэггүй бичлэгийг хаяж хүнд ойлгомжтой уншигдах байдлыг хадгалж үлддэг. Ихэнх тохиолдолд Java –д нэргүй функц код дамжуулхад заавал тэрхүү хүлээн авж буй объект эсвэл функц interface –н объектыг параметердээ байрлуулж байх ба тэрхүү interface нэг method заралж өгсөн байх шаардлагатай байдаг. Java дээр бол энгийн нэргүй функц код параметер болгон дамжуулна гэдэл бол их удаан бичиглэлтэй бас жинхэнэ ёсоор функц код дамжуулахгүй байгаа юм харин объект дамжуулж байгаа учир JVM –д heap санах ойг бас л үр ашиггүй үрэгдэж байгаа юм.

Lambda гийн тусламжтайгаар бид объект биш харин жинхэнэ блок код дамжуулж байгаа юм өөрөөр хэлбэл нэргүй функц.

Кодонд lambda орсон үгүйг шалгах их амархан. **‘->’** ийм нэг бичиглэл байвал lambda тэрхүү кодонд орсон байна гэсэн үг. Одоо харин сүүлчийн жишээгээс өөрөөр хэрхэн lambda –г ашиглаж болох боломжыг үзеэ.

1. Runnable r = () ->System.out.println(“Hello”);
2. ActionListener al = event ->System.out.println(“Hello”);
3. Runnable r2 = () -> {

System.out.println(“Hello”);

System.out.println(“World”);

};

1. BinaryOperator<Long> add = (x, y) -> x + y;
2. BinaryOperator<Long>addExplicit = (Long x, Long y) -> x + y;

Жишээ 1: Энэхүү жишээн дээр ямарч **argument** байхгүй **lambda** илэрхийлэл бичич болхыг харуулж байна. ‘()’ хоосон дугуй хаалтаар энэхүү функцэнд ямарч argument байхгүй гэдгийг тодорхойлно. Код нь бол lambda Runnable class –г хэрхэн зарлаж байгааг харуулж байна. Runnable class бол өөрөө run гэсэн функцтэй ба тэр нь argument агуулдаггүй void буцах төрөлтэй юм.

Жишээ 2: Урд нь авсан жишээ шиг гэхдээ жишээ 1 нэгээс ялгаатай зүйлээрээ 1 argument авч байгаа ба гадуур нь дугуй хаалт хэрэггүй болж байгаа.

Жишээ 3: Эхний 2 жишээн дээр 1 мөр код бичигдэж байгаа бол энэхүү жишээн дээр олон мөрт коп бичиглэлтийг бий болгож байна. Энгэхийн тулд ‘{}’ угалзан хаалтанд кодоо оруулж өгхөд хангалттай.

Жишээ 4: Хэрэв 1 дээш argument агуулвал энэхүү жишээн дээр шиг бичигдэж болно. ‘()’ дугуй хаалтанд argument –ийн нэрийг бичхэд хангалттай. Кодонд бол 2 argument авч тэрхүү 2-г нэмсэн функцыг зохиож байгаа билээ.

Жишээ 5: Урдны бүх argument-тэй жишээнүүд бичигдэхэд программист тэрхүү argument-уудямарөгөгдлийнтөрөлтэйггарцгүй мэдэж байгаа үед ажшиглаж байна. Харин 2 өөр төрлийн argument агуулж байхад ялгахад хэцүү болж эхэлдэг учир энэхүү жишээнд бичигдэж байгаа шиг өгөгдлийн төрөл заралж өгч болно.

**Утга ашиглах**

Урд нь нэргүй inner class ашиглаж байсан бол нэгэн хувьсагчид функцны буцалт утгыг оноож өгөхийг оролдсон байх. Үүнийг хийхийн тулд хувьсагчинд final гэсэн түлхүүр үг оноож өгдөг. Энэ түлхүүр үг ашигласан хувьсагчид ерөнхийдөө анхан тавигдсан утгаас өөр утга шинээр заралж өгч болохгүй гэсэн үг. Бас ийм хувьсагч ашиглахан өөрчлөлт орогүй жинхэнэ утгыг нь ашиглаж байна гэсэн үг.Доорх жишээ lambda ашиглахгүйгээр хэрхэн зарлах арга зам.

final String name = getUsername();  
 button.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent event) {  
System.out.println(“hi ” + name);  
}

});

Дээрх жишээ lambda дээрх бичиглэл

String name = getUsername();  
 button.addActionListener(event ->System.out.println(“hi ”+name));

Java 8 гархаас өмнө иймэрхүү асуудалд хувьсагч заавалчгүй final түлхүүр үг агуулсан байх хэрэгтэй байдаг байсан үгүй бол compiler алдаа заана. Шинээр Java 8 нд зарлагдсан бүх хувьсагч автоматаар final түлхүүр үг агуулж байх ба шинээр утга оноож өгхөд түлхүүр үг арилах ба final түлхүүр үгийг бичиглэлтэйн хамт ашиглавал жинхэнэ утгаар final хувьсагч ашиглана.

**Functional Interfaces**

Функцианал интерфайс гэдэг нь lambda хэлбэрээр ашиглагддайг ганц abstract функцтай интерфайсыг хэлнэ.

Шинэ интерфайс зарлалтын түлхүүр үг байхгүйч их хэрэглэгддэг интерфайсуудын нэг. Жишээ нь Swing –ийн ActionListener interface хэрхэн зарлагдсан

public interface ActionListener extends EventListener {  
 public void actionPerformed(ActionEvent event);  
}

ActionListenerинтерфайс ганц л функцтэй ба эцэгд нь ямарч функц байдгүй. Тийм болохлээр ActionListenerбол функцниал интерфайс юм.

**Төрөл болон дүгнэлт**

Өөрөөр хэлбэл type inference энэ нь lambda –д ашиглагддаг ба анх java 7 нэмэгдэж ирсэн. Java 7 –г илүү сайн мэдэх байх ‘<>’ энэхүү хаалтанд өгөгдлийн төрөлийг оруулж generic argument-тай харицахад ихэнхдээ ашиглагддаг.

Map<String, Integer>oldWordCounts = new HashMap<String, Integer>();  
Map<String, Integer>diamondWordCounts = new HashMap<>();

Дээрх жишээгээр үзвэл 2 янзаар generic хувьсагч зарлаж өгч болох аргууд. Гэхдээ 2 жишээ нь java 7 дээр ажиллахгүй зөвхөн java 8 ажиллах юм. Generic хувьсагч зарлалт нь lambda дээр хэрхэх бичигдэхийг хархыг хүсвэл жишээ нь

public interface Predictate<T> {  
 Boolean test(T t);  
}

Predicate<Integer> atleast5 = x -> x > 5;

Дээрх жишээг шинжвэл Predicate гэсэн generic интерфайс үүсгэж дотрон Boolean утга буцаагч функц зарлаж өгсөн байна. Дараагийн мөр кодноос тэрхүү интерфайс объектыг үүсгэж generic өгөгдөлд Integer оруулж өгч lambda ашиглан хувьсагч 5 тооноос илүү, бага байнуу гэсэн Boolean утгыг шууд хадгалж байна.

Дахиад нэгэн жишээ харья илүү нарийн төвөгтэй.

BinaryOperator<Long>addLongs = (x, y) -> x + y;

Энэхүү жишээ бол ажиллахгүй. Бичиглэл бүгд зөвч бидний compiler энийг уншиж чадахгүй. Яагаад вэ гэвэл BinaryOperatorбол функцианал generic интерфайс. Функц нь Long өгөгдөл буцааж байгаа бас 2 Long өгөгдөлтэй argument авч байгаа. Гэхдээ энэхүү кодыг compiler яагад авахгүй байна вэ гэвэл бидний объект generic argument зөв авсанч compiler өгөгдлийн утгыг объект болгон авж харин буцах утга нь бас Long төрөл байхад энэ 2 нийлэхгүй алдаа гарч байгаа юм.

Энэхүү жишээ ч ажиллахгүй ч ирээдүйн код бичилтэнд иймэрхүү алдааг анхаарж байхаар сануулга болгон татсан юм.