Lecture 20: Anonymous Inner Class V/S Lambda Expressions Part1:

We can replace Anonymous Inner class with Lambda expression but this is not possible all the time.

Anonymous Inner Class:

Runnable r = **new** Runnable() {

@Override

**public** **void** run() {

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

System.***out***.println("We are inside a Run Method of Runnable Interface");

}

};

Thread Demo using Anonymous Inner Class

**package** com.durgaSoft.section2.lecture20;

**public** **class** Example1 {

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

//Anonymouse Inner class:

Runnable r = **new** Runnable() {

@Override

**public** **void** run() {

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

**for**(**int** i=0;i<10;i++) {

System.***out***.println("Child Thread");

}

}

};

Thread th = **new** Thread(r);

th.start();

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

**for**(**int** i=0;i<10;i++) {

System.***out***.println("Main Thread");

}

}

}

Using Lambda Expression:

**package** com.durgaSoft.section2.lecture20;

**public** **class** Example2 {

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

//Anonymouse Inner class:

Runnable r = () ->{

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

**for**(**int** i=0;i<10;i++) {

System.***out***.println("Child Thread");

}

};

Thread th = **new** Thread(r);

th.start();

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

**for**(**int** i=0;i<10;i++) {

System.***out***.println("Main Thread");

}

}

}

Lecture 21: Anonymous Inner Class V/S Lambda Expressions Part2:

Anonymous InnerClass is more powerful than lambda expression:

Case1:

Anonymous InnerClass which implements a Concrete Class:

**package** com.durgaSoft.section2.lecture20;

**class** Test {

}

**public** **class** Example3 {

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

Test test = **new** Test() {

**public** **void** testMethod() {

System.***out***.println("we are in TEst method");

}

};

}

}

Case 2:

Anonymous InnerClass with implements Abstract class:

**package** com.durgaSoft.section2.lecture21;

**abstract** **class** Test2 {

}

**public** **class** Example2 {

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

Test2 test2 = **new** Test2() {

**public** **void** print() {

System.***out***.println("We are in Print method");

}

};

}

}

Case 3:

Anonymous Inner class which implements an Interface, which contains multiple methods.

**package** com.durgaSoft.section2.lecture21;

//Anonymous inner class that implements

//an interface which contains multiple

//methods

**interface** Test3{

**public** **void** m1();

**public** **void** m2();

**public** **void** m3();

}

**public** **class** Example3 {

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

Test3 test3 = **new** Test3() {

@Override

**public** **void** m1() {

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

System.***out***.println("We are in method m1");

}

@Override

**public** **void** m2() {

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

System.***out***.println("We are in method m2");

}

@Override

**public** **void** m3() {

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

System.***out***.println("We are in method m3");

}

};

}

}

Case 4:

Anonymous Inner class which implements an Interface, which contains only single method.

**package** com.durgaSoft.section2.lecture21;

//Anonymous inner class that implements

//an interface which contains only single

//

**interface** Test4{

**public** **void** m1();

}

**public** **class** Example4 {

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

Test4 test3 = **new** Test4() {

@Override

**public** **void** m1() {

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

System.***out***.println("We are in method m1");

}

};

}

}

We can implement only case 4 using lambda expression in other above scenarios we cannot implement lambda expresssions.

Lecture 22 Anonymous Inner Class V/S Lambda Expressions Part3:

this keyword works differently in Lambda Expresssion.

While working with Anonymous Inner Class

**package** com.durgaSoft.section2.lecture22;

**public** **class** Example1 {

**int** x =888;

**public** **void** m2() {

Interf4 interf = **new** Interf4() {

**int** x =999;

@Override

**public** **void** m1() {

System.***out***.println("We arein m1 method Interf4 "+ **this**.x); //Output = 999

}

};

interf.m1();

}

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

Example1 example1 = **new** Example1();

example1.m2();

}

}

**interface** Interf4{

**public** **void** m1();

}

In anonymous inner class this refers to current class objects.

While working with lambda expression:

**package** com.durgaSoft.section2.lecture22;

**public** **class** Example2 {

**int** x =888;

**public** **void** m2() {

Interf5 interf = () -> {

**int** x =999;

System.***out***.println("We arein m1 method Interf4 "+ **this**.x); //Output = 888

};

interf.m1();

}

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

Example2 example1 = **new** Example2();

example1.m2();

}

}

**interface** Interf5{

**public** **void** m1();

}

Inside Lambda expression we cannot define any instance variables. All the variables declared in the lambda expressions are local variables only.

So we can say that this keyword in Lambda expressions refer to the outer variables of the class. Whereas this in the anonymous class refers to the inner class variables.

Lecture 23 Anonymous Inner Class V/S Lambda Expressions Part4:

|  |  |  |
| --- | --- | --- |
| S.I | Anonymous Inner Class | Lambda Expression |
|  |  |  |
| 1 | It’s a Class without a name | It’s a Function without a name |
|  |  |  |
| 2 | Anonymous Inner class can extend abstract and concrete class | Lambda expression cant extend abstract and concrete classes |
|  |  |  |
| 3 | Anonymous Inner class can implement an interface that contains any number of abstract methods | Lambda expression can implement an interface which contains Single abstract method (Functional Interface) |
|  |  |  |
| 4 | Inside Anonymous Inner class we can declare instance variables | Inside Lambda expressions we can’t declare instance variables whatever variables declared are considered as local variables |
|  |  |  |
| 5 | Anonymous Inner class can instantiated | Lambda expression cannot be instantiated |
|  |  |  |
| 6 | Inside Anonymous Inner class this always referes t current Anonymous Inner class object but not outer class object | Inside Lambda expression this always refers to current outer class object, i.e. enclosing class object |
|  |  |  |
| 7 | Anonymous Inner class is the best choice if we want to handle multiple methods | Lambda expression is the best choice if we want to handle interface with Single Abstract method (Functional Interface) |
|  |  |  |
| 8 | For the Anonymous Inner class, at the time of compilation a separate .class file will be generated | For the Lambda expression at the time of compilation, no separate .class file will be generated. |
|  |  |  |
| 9 | Memory will be allocated on demand whenever, we are creating objects | Lambda expression will reside in permanent memory of JVM (method area) |

Consider the below example:

We can access enclosing class and method variables:

**package** com.durgaSoft.section2.lecture23;

**public** **class** Example1 {

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

Test test = **new** Test();

test.m2();

}

}

**interface** Interf6{

**public** **void** m1();

}

**class** Test{

**int** x=10;

**public** **void** m2() {

**int** y=20;

Interf6 interf = () ->{

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

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

};

interf.m1();

}

}

The local variables which are referenced in lambda expression are final or effectively final.

Eg:

We are able to change the class level variable:

**public** **void** m2() {

**int** y=20;

Interf7 interf = () ->{

x=888;

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

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

};

interf.m1();

}

O/P

888

20

If we change the local variable i.e y then we get Compile time error.

**public** **void** m2() {

**int** y=20;

Interf7 interf7 = () ->{

y=888;

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

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

};

interf7.m1();

}

i.e Local variables reference from the lambda expression must be final or effectively final.

Advantage of Lambda Expressions:

* We can enable functional programming in Java
* We can reduce length of the code so that readability will be improved.
* We can resolve complexity of Anonymous Inner class for some extent
* We can handle procedures/functions just like values
* We can pass procedure/functions as arguments
* Easier to use updated APIs and Libraries.
* Enable support for parallel processing