**Java-8 – Anonymous Inner Class vs Lambda Expression**

* **Part\_04:**

**Difference between Anonymous Inner class & Lambda expression**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Anonymous Inner Class** | **Lambda Expression** |
| 1 | It is a class without name. | It is a function without name (Anonymous function). |
| 2 | Anonymous inner class can extend abstract and concrete classes. | Lambda expression can’t 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 variable. | Inside Lambda expression we can’t declare instance variable. Whatever variables declared are considered as local variable. |
| 5 | Anonymous inner class can be instantiated. | Lambda expression cannot be instantiated. |
| 6 | Inside anonymous inner class, this always refers current anonymous inner class object but not outer object. | Inside lambda expression, this always refers current outer class object, that is enclosing class object. |
| 7 | Anonymous inner class is best choice if we want to handle multiple methods. | Lambda expression is a 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 object. | Lambda expression will reside in permanent memory of JVM(Method Area). |

* **Conclusions:**

Inside lambda expression we can access enclosing class variables and including method (local) variables also.

interface Interf{

public void m1();

}

class Test{

int x = 10;

public void m2(){

int y = 20;

Interf interf = () ->{

System.out.println(x);

System.out.println(y);

};

Interf.m1();

}

public static void main(String[] args){

Test t = new Test();

t.m2();

}

}

The local variables which are referenced from Lambda expression are final or effectively declared final.

interface Interf{

public void m1();

}

class Test{

int x = 10;

public void m2(){

int y = 20;

Interf interf = () ->{

X = 888; // No error

Y = 999; // Error

System.out.println(x);

System.out.println(y);

};

Interf.m1();

}

public static void main(String[] args){

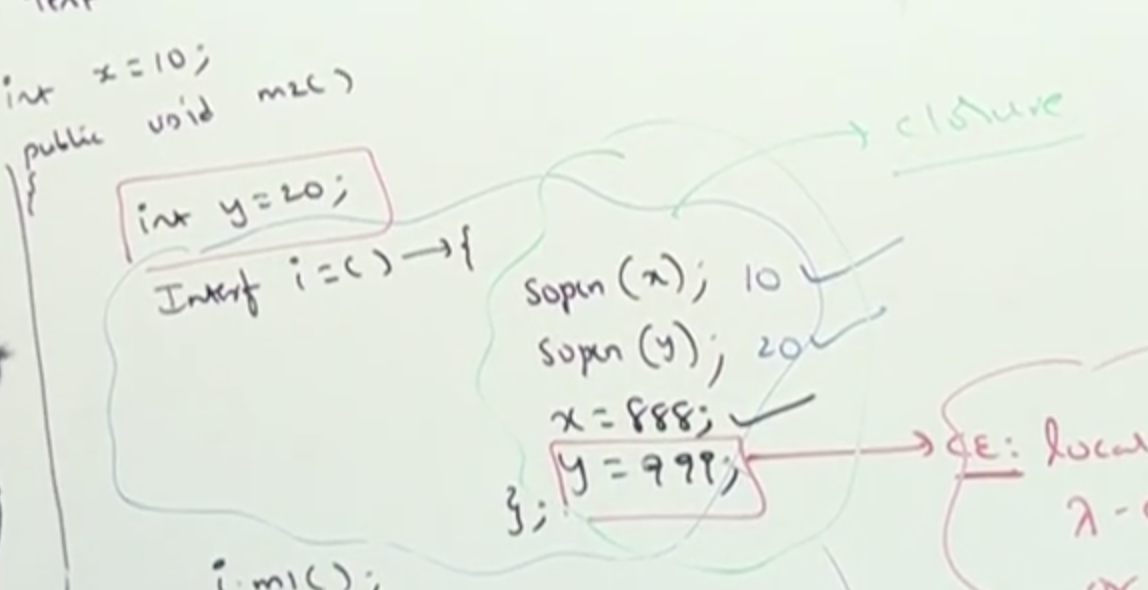
Test t = new Test();

t.m2();

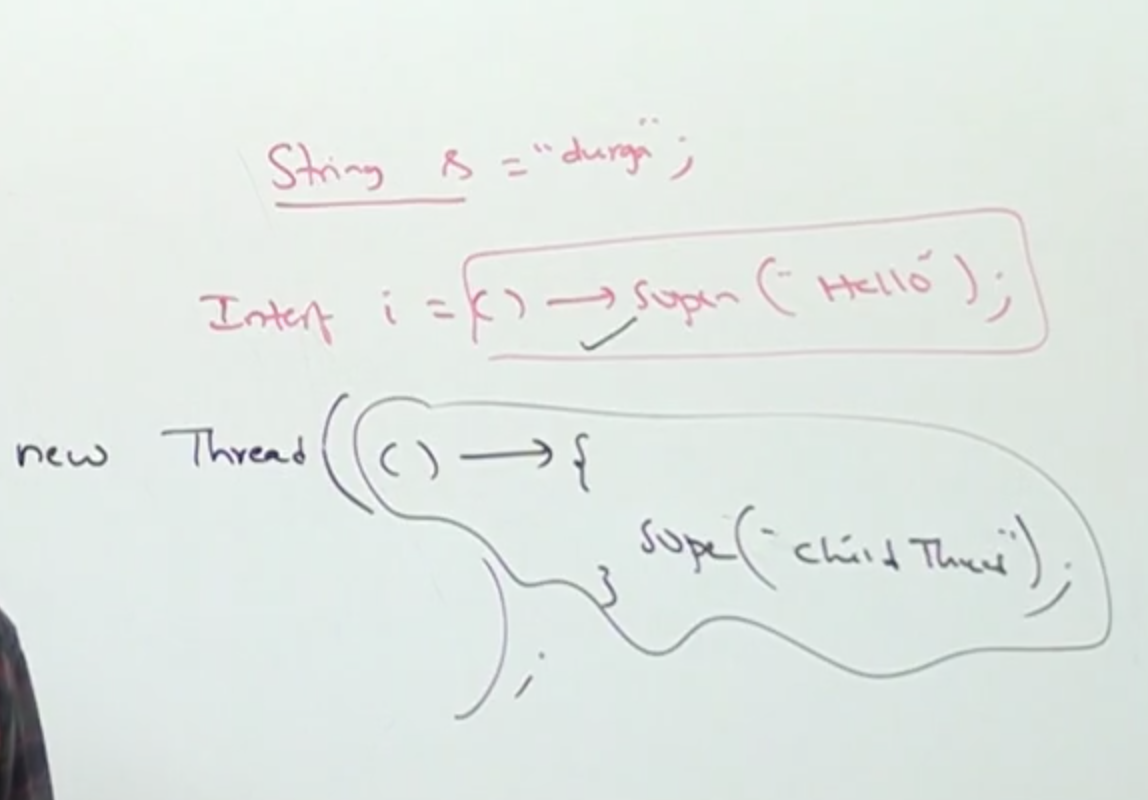
}

}

CE: local variable referenced from a lambda expression must be final or effectively final.



* **Advantages of Lambda expressions:**

****

1. We can enable functional programming in Java.
2. We can reduce length of the code so that readability will be improved.
3. We can resolve complexity of anonymous inner classes until some extent.
4. We can handle procedure / functions just like values.
5. We can pass procedure/functions as arguments.
6. Easier to use updated API’s and libraries.
7. Enables support for parallel processing.