Java 8 Static Methods and Default Methods

Anonymours Inner Class VS Lambda Expression

Preliminary Notes:

Anonymous Inner class is more powerful than lambda expression because of below reasons.

Anonymous Inner Classes

1. Anonymous Inner class can extend a normal and abstract class.
2. Anonymous Inner class can implement an interface which contains any number of abstract methods.

Lambda Expression :

1. Lambda expression can implement an interface which must contain only one abstract method, such interface(which contains only one abstract method) is also called Functional Interface.

Conclusion :

Anonymous Inner class != Lambda Expression
Anonymous Inner class > Lambda Expression (check preliminary notes above)

Default Methods(Virtual Extension Method/Defender Method) :

Need of default methods :

It provides backward compatibility, i.e. if we want to add new methods to the interface, without affecting the implementation classes, then we should go for **default methods**.

Imp Points :

1. Every other feature of java 8 is dependent on the concept of default method.
2. Default method has default implementation, the implementing class has to provide its requirement specific implementation.
3. Object class methods are not available to implement in functional interfaces as default methods. Reason being that they are already implemented in Object class.

Default Methods and Multiple Inheritance

We can write similar default methods in two interfaces and inherit them in same implementing class, and can have interface specific behaviour. Let's see below code.

****

```
interface Left{
	default void m1(){
		System.out.println("Left Interface method...");
	}
}

interface Right{
	default void m1(){
		System.out.println("Right Interface method...");
	}
}

public class DefaultMethodImplementWRTMultipleInheritance implements
Left, Right{
	public static void main(String[] args){
		DefaultMethodImplementWRTMultipleInheritance d = new
DefaultMethodImplementWRTMultipleInheritance();
		d.m1();
	}

//Here, is we wont override default method, the complier will give error.
//Hence overriding an abstract method is must.
	public void m1(){
		System.out.println("Own implementation..."); //Prints Own
Implementation
		Left.super.m1(); //Prints "Left Interface method…"
		Right.super.m1(); //Prints "Right Interface method…"
	}
}
```

****

Static Methods :

Why static methods come in Java 8?

To define general utility methods, where everything is static, then there is no need to define those methods in class, if we do so then, we need to create an object and it will impact the performance. Then to solve this Java 8 come up with **static methods in interface** concept, where we can define static methods in interface and use them in implementing classes. These static methods in interface are termed as utility methods.

For e.g.

```
//Below is an interface with one static/utility method name m1
Interface Interf{

        Static void m1(String msg){
                System.out.println(msg);
        }
}
//Create N no of classes implementing above interface i.e. interf, and which will have m1 method by default.

Class c1 implements Interf{

        public static void main(String[] args){
        System.out.println(m1("Hello"));

}
Class c2 implements Interf{

        public static void main(String[] args){
        System.out.println(m1("Hello"));

}
Class c3 implements Interf{

        public static void main(String[] args){
        System.out.println(m1("Hello"));

}
```

…
Class cN implements Interf{

```
public static void main(String[] args){
System.out.println(m1("Hello"));
```

}

How to call interface static methods?

By using interface name only, see below example.

```
Interface interf{
    Static void m1(){
        System.out.println("Hello…");
)
Public class Test implements interf{
    Public static void main(String[] args){
        Interf.m1();
    }
}
```

Conclusion:

1. Java 8 onwards, we can define static methods in interface too.
2. We can have N no of static methods in interface from Java 1.8.
3. We can call interface static methods by using interface name only, please refer above example.