Java 8 New Features

Mohammed Husain Bohara

Compare Java 8 & Java 10 Features

Java 8 New Features

1. Lambda Expressions
2. Functional Interfaces
3. Default Methods
4. Predefined Functional interfaces
   Predicates
   Functions
   Supplier
   Consumer

   5. Double Colon Operator (Constructor and Method references)
   6. Stream API
   7. Date and Time API(Joda Time API)
      Nashorn java script

Lambda Expression

- History of lambda calculus- 1930

Example of a Lambda Expression

- The lambda expression

  λ x . (+ x 1) 2

  represents the application of a function λ x . (+ x 1) with a formal parameter x and a body + x 1 to the argument 2. Notice that the function definition λ x . (+ x 1) has no name; it is an *anonymous function*.
- In Java 8, we would represent this function definition by the Java 8 lambda expression `x -> x + 1`.

Which one is the first programming language which uses lambda expression?

- C
- C++
- C#
- Python
- Java
- Ruby
- Java script
- Or any one else

LISP

What about Java 8?

- Features from the lambda calculus such as lambda expressions have been incorporated into many widely used programming languages like C++ and now very recently Java 8.
- In 2014- Java is too late to add such feature

Why they have added such feature?

- Java is OOP language
- Functional programming Languages features was not enable

Functional programming language example

```
a=f()
f(f1)
```

What is lambda expression?

- It is just anonymous function/nameless function.

means

- No name
- No return type
- No modifiers

Example-1

```
public void m1()
{
        System.out.println("Hello");
}
```

Convert into lambda expression

```
() -> { System.out.println("Hello"); }
```

OR

```
() -> System.out.println("Hello");
```

Example-2

```
public void add(int a, int b)
{
        System.out.println(a+b);
}
```

Convert into lambda expression

```
(int a, int b) -> System.out.println(a+b);
```

OR

```
( a,  b) -> System.out.println(a+b);
```

Based on the context, Compiler guess type automatically

Example-3

```
public void square(int n)
{
        return n*n;
}
```

Convert into lambda expression

```
(n) -> n*n;
```

OR

```
n -> n*n;
```

Based on the context, Compiler guess type automatically

Rules to write lambda expression

1. Any number of arguments
   zero, one, 2….
2. For one-argument lambda expression parenthesis are optional
3. (a,b) -> SOP(a+b);
4. (a,b) -> {
   SOP(a+b);
   SOP(a-b);
   }

How to call/invoke lambda expressions?

- Functional Interfaces concepts must required to use lambda expression.

What is the Functional Interface and how it is different from normal interface?

What is the Functional Interface?

- We have used Runnable interface in multithreading

- Some other interfaces like

1. Callable
2. Comparable
3. Serializable
4. RandomAccess
5. Cloneable
6. SingleThreadModel

1. Serializable
2. RandomAccess
3. Cloneable
4. SingleThreadModel

Called Marker Interfaces- the interface does not contain any method

What is the Functional Interface?

- The interface which contain only single abstract method- prior to java 1.8

- Java 1.8 version onward- default and static methods

Runnable → run()

Callable → call()

Comparable → compareTo()

The annotation to explicitly show Functional Interface

@FunctionalInterface

Example of interface

```
interface InterfaceExample
{
    public abstract void m1();

    default void m2()
    {

    }
    static void m3()
    {

    }
    public abstract void m4();

}
```

Example of Functional Interface

```
interface InterfaceExample
{
    public abstract void m1();

    default void m2()
    {

    }
    static void m3()
    {

    }
    public abstract void m4();

}
```

```
@FunctionalInterface

interface InterfaceExample
{
    public abstract void m1();

    default void m2()
    {

    }
    static void m3()
    {

    }
    public abstract void m4();

}
```

Output

```
C:\Users\sony\Desktop\javaProgram\Java8>javac InterfaceExample.java
InterfaceExample.java:1: error: Unexpected @FunctionalInterface annotation
@FunctionalInterface
^
  InterfaceExample is not a functional interface
    multiple non-overriding abstract methods found in interface InterfaceExample

1 error
```

Will it compile successfully?

```
@FunctionalInterface

interface A
{

}
```

```
@FunctionalInterface

interface A
{
        public void m1();
}

@FunctionalInterface
interface B
{

}
```

```
@FunctionalInterface

interface A
{
        public void m1();
}

@FunctionalInterface
interface B extends A
{

}
```

Is it Functional Interface?

```
interface A
{
  public void m1();
}
```

Yes, if interface contain single abstract method then interface is by default Functional Interface

How to invoke the lambda expression

- To invoke the lambda expression some reference is required
- Functional Interface provide the reference
- You can not use lambda expression without Functional Interface

Example- what will be printed here?

```
interface A
{
    public void m1();
}

class Demo implements A
{
    public void m1()
    {
        System.out.println("m1 method of A interface");
    }
}

class CallerClass
{
    public static void main(String[] args) {
        Demo d = new Demo();
        d.m1();
    }
}
```

```
interface A
{
    public void m1();
}

class Demo implements A
{
    public void m1()
    {
        System.out.println("m1 method of A interface");
    }
}

class CallerClass
{
    public static void main(String[] args) {
        A a = new Demo();
        a.m1();
    }
}
```

Using lambda function

```
interface A
{
    public void m1();
}
class CallerClass
{
    public static void main(String[] args) {
        A a = ()->System.out.println("m1 method of A interface");
        a.m1();
    }
}
```

```
C:\Users\sony\Desktop\javaProgram\Java8>java -cp . CallerClass
m1 method of A interface
```

Using { } - in lambda expression

```
interface A
{
    public void m1();
}
class CallerClass
{
    public static void main(String[] args) {
        A a = ()->{System.out.println("m1 method of A interface");};
        a.m1();
    }
}
```

Lambda expression can be used with Functional interface only

```
interface A
{
    public void m1();
    public void m2();
}
class CallerClass
{
    public static void main(String[] args) {
        A a = ()->System.out.println("m1 method of A interface");
        a.m1();
    }
}
```

```
C:\Users\sony\Desktop\javaProgram\Java8>java -cp . CallerClass

C:\Users\sony\Desktop\javaProgram\Java8>javac A.java
A.java:12: error: incompatible types: A is not a functional interface
                A a = ()->System.out.println("m1 method of A interface");
                      ^
    multiple non-overriding abstract methods found in interface A
1 error
```

Example-2

```
interface AddInterface
{
    public void add(int a, int b);
}
class DemoAdd implements AddInterface
{
    public void add(int a, int b)
    {
        System.out.println("The sum is: "+(a+b));
    }
}

class TestAdd
{
    public static void main(String[] args) {
        AddInterface a = new DemoAdd();
        a.add(100,200);
    }
}
```

Convert into lambda expression-

```
interface AddInterface
{
    public void add(int a, int b);
}

class TestAdd
{
    public static void main(String[] args) {
        AddInterface i = (a,b)->System.out.println("The sum is: "+(a+b));
        i.add(100,200);
    }
}
```

Method with return type- lambda expression

```
interface AddInterface
{
    public int add(int a, int b);
}

class TestAdd
{
    public static void main(String[] args) {
        AddInterface i = (a,b)->a+b;
        System.out.println(i.add(100,200));
    }
}
```

Using return keyword in lambda expression

```
interface AddInterface
{
    public int add(int a, int b);
}

class TestAdd
{
    public static void main(String[] args) {
        AddInterface i = (a,b)->{return a+b;};
        System.out.println(i.add(100,200));
    }
}
```

Lambda expression use in – multithreading also

```
class MyRunnable implements Runnable
{
    public void run()
    {
        for(int i=0; i<10; i++)
        {
            System.out.println("child thread");
        }
    }
}
class ThreadDemo
{
    public static void main(String[] args) {
        MyRunnable r = new MyRunnable();
        Thread t = new Thread(r);
        t.start();
        for(int i=0; i<10; i++)
        {
            System.out.println("main thread");
        }
    }
}
```

Runnable is Functional Interface- use lambda expression

```
class ThreadDemo
{
    public static void main(String[] args) {
        Runnable r = ()->{
            for(int i=0; i<10; i++)
            {
                System.out.println("child thread");
            }
    };
        Thread t = new Thread(r);
        t.start();
        for(int i=0; i<10; i++)
        {
            System.out.println("main thread");
        }
    }
}
```

Any Question