Lambda expression

Lambda expression is a new and important feature of Java which was included in Java SE 8. It provides a clear and concise way to represent one method interface using an expression. It is very useful in collection library. It helps to iterate, filter and extract data from collection.

The Lambda expression is used to provide the implementation of an interface which has functional interface. It saves a lot of code. In case of lambda expression, we don't need to define the method again for providing the implementation. Here, we just write the implementation code.

Lambda expression provides implementation of ***functional interface***. An interface which has only one abstract method is called functional interface. Java provides an annotation **@*FunctionalInterface***, which is used to declare an interface as functional interface.

Why use Lambda Expression?

- It enables functional programming (To provide the implementation of function interface).
- Using Lambda expression we get readable and concise code.
- It also enables us easier-to-use API's and libraries.
- I also enables support for parallel processing (Multithreading).

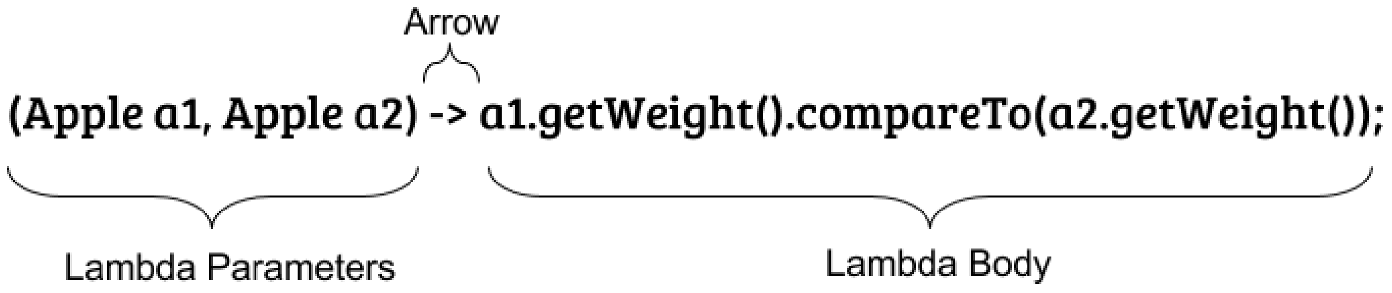
Java lambda expression is consisted of three components.

1) Argument-list: It can be empty or non-empty as well.

2) Arrow-token: It is used to link arguments-list and body of expression.

3) Body: It contains expressions and statements for lambda expression.

(Argument-list) -> { body }


Arrow
(Apple a1, Apple a2) -> a1.getWeight().compareTo(a2.getWeight());
Lambda Parameters
Lambda Body

- No modifier, no return type, no function name, no interface.
- A lambda expression can have zero or any number of arguments.
- If number of argument is only one then even we can omit function parentheses.
- If we define Lambda expression without curly braces then we can give only single line in method body. It is actually invalid to specify the return keyword when we have a one-liner lambda expression without {} curly braces.
- If we define lambda expression with curly braces then we can give multiple line method body.
- In Java lambda expression, if there is only one statement, we may or may not use return keyword. We must use return keyword when lambda expression contains multiple statements.

Important Points:

#. Java lambda expression is treated as a function, so compiler does not create .class file.

#. Lambda expression are used primarily to define inline implementation of functional interface that means an interface with a single abstract method. Some known Functional Interfaces in Java are **Comparator, Runnable, ActionListener, Callable,** etc.

Also note that an interface is still a functional interface **even if it defines one or more default methods**, as long as it defines a single abstract method.

#. A lambda expression can be passed as argument to a method or stored in a variable.

#. Technically, Lambda expressions do not allow us to do anything that we did not do prior to Java 8. It's just that we don't need to write clumsy code in order to use the behavior parameterization.

Let's look at some use case examples of java lambda expression.

A boolean expression: `(List list) -> list.isEmpty()`

Creating objects: `() -> new Apple()`

Consuming from an object: `(Apple a) -> { System.out.println(a.getWeight()); }`

Select/extract from an object: `(String s) -> s.length()`

Produce a single value by performing computation on two values: `(int a, int b) -> a * b`

Compare two objects: `(Apple a1, Apple a2) -> a1.getWeight().compareTo(a2.getWeight())`

Difference between normal interface implementation and functional programming.

```
@FunctionalInterface
interface Drawable{
    public void draw();     // Abstract method.

    /* register method */
    default void register(Drawable o){
        o.draw();
    }
}

class OldImplementation implements Drawable{

    @Override
    public void draw(){
        System.out.println("I old Implementation.");
    }

    public static void main(String...a){
        /* 1. Normal Way. Implement -> Override -> call the register
method. */
        OldImplementation oi = new OldImplementation();
        oi.register(oi);

        /* 2. We have also had another option via anonymous class. */
        Drawable d = new Drawable(){
            public void draw(){
                System.out.println("I Old implementation via anonymous
class.");
            }
        };

        d.draw();
    }
}

/* Let's check out the new implementation via lambda expression. */
class NewImplementation{

    public static void main(String...a){

        Drawable d2 = () -> {
            System.out.println("I new implementation");
            System.out.println("I new implementation With More lines.");
        };
        d2.draw();
    }
}
```

/* Different Examples of Lambda Expression */

```java
@FunctionalInterface
interface Sayable{
   public String say();
}

interface Sayable1{
   public String say(String name);
}

@FunctionalInterface
interface Addable{
    int add (int a, int b);  /* by default public and abstract */
}

/* via functional programming we don't need to even implement the
interface. */
class LambdaExpression{

public static void main(String...args){

    /* 1. With no parameter */
    Sayable s = () -> { return "Nothing to say";};
    System.out.println(s.say());

    /* 2. With Single parameter */
    Sayable1 s1 = (name)->{ return "Hello " + name; };
    System.out.println(s1.say("Sayeed"));

    /* Can Also write: omit parentheses */
    s1 = name -> { return "Hii " + name; };
    System.out.println(s1.say("Ahmad"));
    System.out.println(s1.say("Ahmad Sayeed"));

    /* 3. With Multiple Parameters. if single statement there then JVM
figure out itself what type of data has to return. */
    Addable ad1 = (int a, int b) -> (a+b);
    System.out.println("10+20: " + ad1.add(10, 20));

    /* we can also eliminate parameter type */
    ad1 = (a,b) -> (a+b);
    System.out.println("50+40: " + ad1.add(50, 40));

    /* With return keyword:  {} required */
    ad1 = (a,b) -> { return a+b; };
    System.out.println("35+25: " + ad1.add(35, 25));

    /* 4. With Multiple Statement */
    Sayable s2 = () -> {
        System.out.println("My First Statement");
        System.out.println("My Second Statement");
        System.out.println("My Third Statement");
        return "I am also returning an statement";
    };
    System.out.println(s2.say());
}
}
```

/* Lambda Expression with Threads */

```
class ThreadOne implements Runnable{

@Override
  public void run(){
    System.out.println(Thread.currentThread().getName() +" is entered");

    try{    Thread.sleep(2000); }
    catch(InterruptedException e){ e.printStackTrace(); }

    System.out.println(Thread.currentThread().getName() + " is dead now");
  }
}

class LambdaThread{

  public static void main(String...a){
    /* Thread Running with normal way: */
    ThreadOne to1 = new ThreadOne();
    Thread t1 = new Thread(to1, "One");  t1.start();
    Thread t2 = new Thread(to1, "Two");  t2.start();

    /* Or another older way is via anonymous */
    Runnable r1 = new Runnable(){
        public void run(){
            System.out.println("Anonymous Thread is running...");
        }
    };

    Thread t3 = new Thread(r1);  t3.start();

    /* Thread with lambda */
    Runnable r2 = ()->{
        System.out.println("Lambda Thread2 is running...");
    };
    Thread t4 = new Thread(r2);  t4.start();

    /* Just tried */
    new Thread(
        ()->{ System.out.println("Shor Lambda Thread is running...");  }
    ).start();
  }
}
```

/* Lambda Scope with Inner classes */

```
import java.util.function.Consumer;     /* an interface */

public class LambdaScopeTest{

  public int x = 0;

  /* Inner Class */
  class FirstLevel{
    public int x =1;

    void methodInFirstLevel(int x){
        Consumer<Integer> MyConsumer = (y) -> {
            System.out.println("X = " + x);
            System.out.println("Y = " + y);
            System.out.println("this.x = " + this.x);
            System.out.println("LambdaScopeTest.this.x = " +
LambdaScopeTest.this.x);
        };

        MyConsumer.accept(x);
    }
  }

  public static void main(String...a){
    LambdaScopeTest st = new LambdaScopeTest();
    LambdaScopeTest.FirstLevel f1 = st. new FirstLevel();
    f1.methodInFirstLevel(11);
  }
}
```

/* Lambda Expression with loops/collection. */

```
import java.util.*;

public class LambdaWithCollection{

    public static void main(String...args){

        String[] str = {"Ahmad", "Sayeed", "Asmal", "Babu"};

        /* create an ArrayList */
        List<String> names = Arrays.asList(str);

        System.out.println("\nvia lambda expression\n");
        names.forEach( (n)-> System.out.println(n) );

        System.out.println("\nusing (::) operator from java 8\n");

        names.forEach(  System.out::println);
    }
}
```

/* Another Example Of Lambda Expression

```
public class LambdaTest
{
  public static void main(String... s){
    LambdaTest tester=new LambdaTest();

     /* with type declaration */
    MathOperation addition=(int a,int b)->a+b;

    /* without type declaration */
    MathOperation subtraction=(a,b)->a-b;

    /* with return statement along with curly braces */
    MathOperation multiplication=(int a,int b)->{return a*b;};

    /* without return statement and without curly braces */
    MathOperation division=(int a,int b)->a/b;

    System.out.println("10+5= "+tester.operate(10,5,addition));
    System.out.println("10-5= "+tester.operate(10,5,subtraction));
    System.out.println("10*5= "+tester.operate(10,5,multiplication));
    System.out.println("10/5= "+tester.operate(10,5,division));

    GreetingService greetService1=message->System.out.println("hello
"+message);

    GreetingService greetService2=message->{System.out.println("hello
"+message);
    System.out.println("hai "+message);};

    greetService1.sayMessage("Mahesh");
    greetService2.sayMessage("Suresh");
  }

    interface MathOperation {
        int operation(int a,int b);
    }

    interface GreetingService {
        void sayMessage(String message);
    }

    private int operate(int a,int b,MathOperation mathOperation) {
        return mathOperation.operation(a,b);
    }
}
```

```
/* lambda with Event Listener */

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JTextField;

public class LambdaEventListenerExample {
  public static void main(String[] args) {

    JTextField tf=new JTextField();
    tf.setBounds(50, 50,150,20);
    JButton b=new JButton("click");
    b.setBounds(80,100,70,30);

    /* lambda expression implementing here. */
    b.addActionListener( (e) -> { tf.setText("hello swing"); } );

    JFrame f=new JFrame();
    f.add(tf);f.add(b);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setLayout(null);
    f.setSize(300, 200);
    f.setVisible(true);

  }
}
```