Lambda Expressions

Introduction

- The Project Lambda is part of Java 8. It provides us with the capability to avoid the annoying syntax of declaring anonymous inner class just for the purpose of passing over a functionality to another part of our program.

Introduction

- When dealing with an interface that includes one method only and the sole purpose of that interface is to allow us passing over functionality to another part of our program we can use the lambda expression as an alternative for the anonymous inner class.

Syntax

- The lambda expression includes a comma separated list of formal parameters enclosed within parentheses (the data type of the parameters can be usually omitted). If there is only one parameter then we can omit the parentheses.

```
ob -> ob.age<40
```

Syntax

- The body includes a single expression or a statement block.
- If we use a block then we can include the return statement.

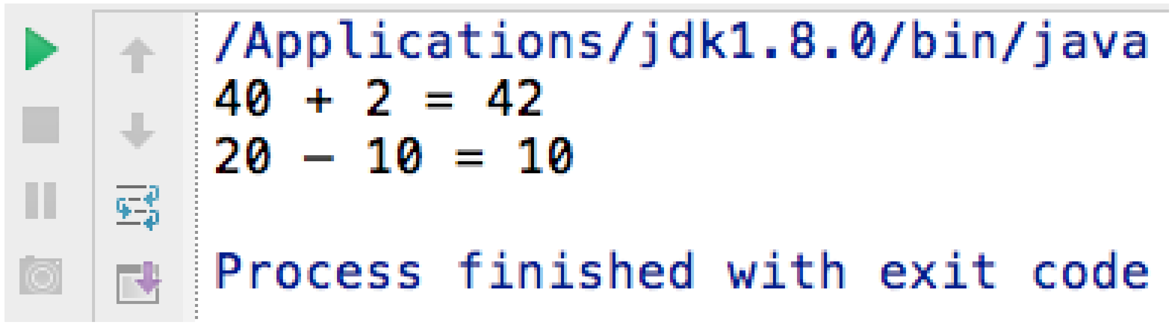
Simple Demo

```
package com.lifemichael.samples;

public class SimpleLambdaExpressionDemo {

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

    public int calculate(int a, int b, MathOperation op) {
        return op.execute(a, b);
    }
```

Simple Demo

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

        SimpleLambdaExpressionDemo myApp =
                new SimpleLambdaExpressionDemo();
        MathOperation addition = (a, b) -> a + b;
        MathOperation subtraction = (a, b) -> a - b;
        int num = addition.execute(40, 2);
        System.out.println("40 + 2 = " + num);
        System.out.println("20 - 10 = " +
                myApp.calculate(20, 10, subtraction));
    }
}
```

Simple Demo



```
/Applications/jdk1.8.0/bin/java
40 + 2 = 42
20 – 10 = 10

Process finished with exit code
```

The Output

Methods References

- There are four types of methods references: reference to static method, reference to an instance method of a particular object, reference to an instance method of an arbitrary object of a particular type and reference to constructor.

Reference to Static Method

- We can refer a static method by specifying the name of the class following with "::" and the name of the static method.

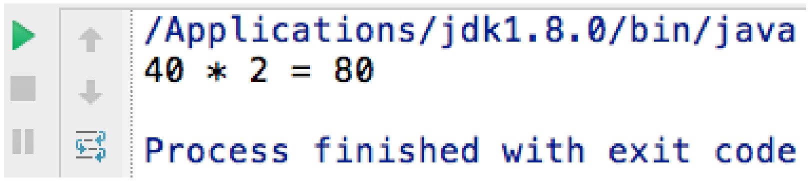
Reference to Static Method

```
package com.lifemichael.samples;

public class SimpleLambdaExpressionDemo {

    interface MathOperation {
        int execute(int a, int b);
    }
```

Reference to Static Method

```
    public static void main(String... args) {
        SimpleLambdaExpressionDemo myApp =
                new SimpleLambdaExpressionDemo();
        MathOperation op = Utils::calcu;
        int num = op.execute(40, 2);
        System.out.println("40 * 2 = " + num);
    }
}

class Utils
{
    public static int calcu(int a,int b)
    {
        return a*b;
    }
}
```

Reference to Static Method



```
/Applications/jdk1.8.0/bin/java
40 * 2 = 80

Process finished with exit code
```

The Output

Reference to Instance Method

- We can refer a specific instance method by specifying the reference for the object following with "::" and the name of the instance method.

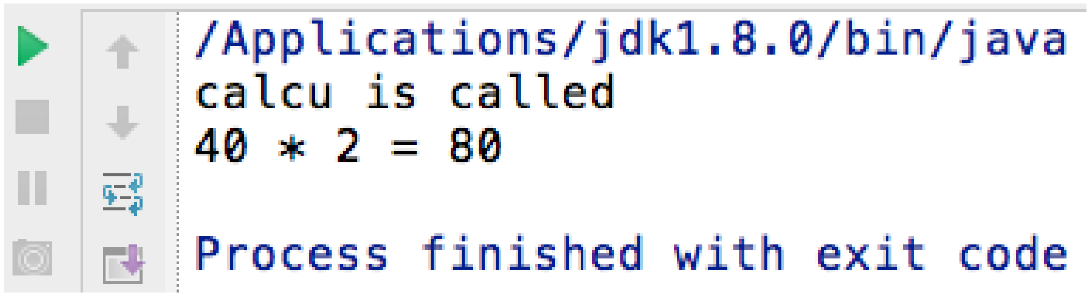
Reference to Instance Method

```
package com.lifemichael.samples;

public class SimpleLambdaExpressionDemo {

    interface MathOperation {
        int execute(int a, int b);
    }
```

Reference to Instance Method

```
    public static void main(String... args)
    {
        SimpleLambdaExpressionDemo myApp =
                new SimpleLambdaExpressionDemo();
        Utils utils = new Utils();
        MathOperation op = utils::calcu;
        int num = op.execute(40, 2);
        System.out.println("40 * 2 = " + num);
    }
}

class Utils
{
    public int calcu(int a,int b)
    {
        System.out.println("calcu is called");
        return a*b;
    }
}
```

Reference to Instance Method



```
/Applications/jdk1.8.0/bin/java
calcu is called
40 * 2 = 80

Process finished with exit code
```

The Output

Reference to Instance Method of an Arbitrary Object

- We can refer a specific instance method without been specific about the object on which it should be invoked.
- Instead of specifying the reference for a specific object we should specify the name of the class followed by :: and the name of the function.

Reference to Instance Method of an Arbitrary Object

```
public class SimpleLambdaExpressionDemo {

    public static void main(String... args) {
        Student[] students = {
                new Student(123123,98,"dave"),
                new Student(234233,88,"ron"),
                new Student(452343,82,"ram"),
                new Student(734344,64,"lara")
        };
        Arrays.sort(students,Student::compareTo);
        for(Student std : students) {
            System.out.println(std);
        }
    }
}
```

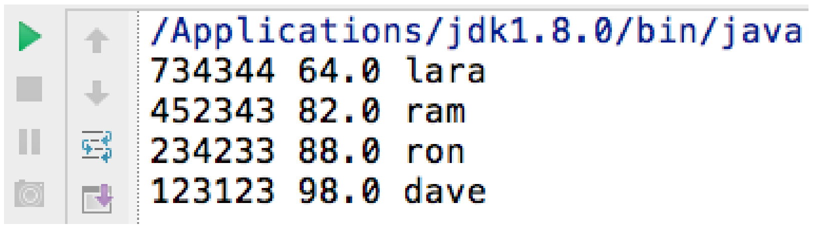
Reference to Instance Method of an Arbitrary Object

```
class Student implements Comparable<Student>
{
    private double average;
    private int id;
    private String name;

    Student(int id, double avg, String name)
    {
        this.average = avg;
        this.id = id;
        this.name = name;
    }
```

Reference to Instance Method of an Arbitrary Object

```
    @Override
    public String toString() {
        return id+" "+average+" "+name;
    }

    @Override
    public int compareTo(Student o) {
        if(this.average>o.average) {
            return 1;
        }
        else if(this.average<o.average) {
            return -1;
        }
        else return 0;
    }
}
```

Reference to Instance Method of an Arbitrary Object



```
/Applications/jdk1.8.0/bin/java
734344 64.0 lara
452343 82.0 ram
234233 88.0 ron
123123 98.0 dave
```

The Output

Reference to Constructor

- We can refer the default constructor in a class by writing the name of the class following with `::new`.

YouTube

Reference to Constructor

```
public class ReferenceToConstructor
{
    public static void main(String... args) {
        Student []students = new Student[8];
        populate(students, Student::new);
        for(int i=0; i<students.length; i++) {
            System.out.println(students[i]);
        }
    }
    public static void populate(Student[] vec, StudentsGenerator ob) {
        for(int i=0; i<vec.length; i++) {
            vec[i] = ob.create();
        }
    }
}
```

Reference to Constructor

```
interface StudentsGenerator {
    public Student create();
}

class Student implements Comparable<Student>
{
    private static int counter = 1;
    private static String[] names = {"david","moshe","anat","jake"};
    private double average;
    private int id;
    private String name;

    Student() {
        id = counter++;
        name = names[(int)(names.length*Math.random())];
        average = (int)(100*Math.random());
    }
```

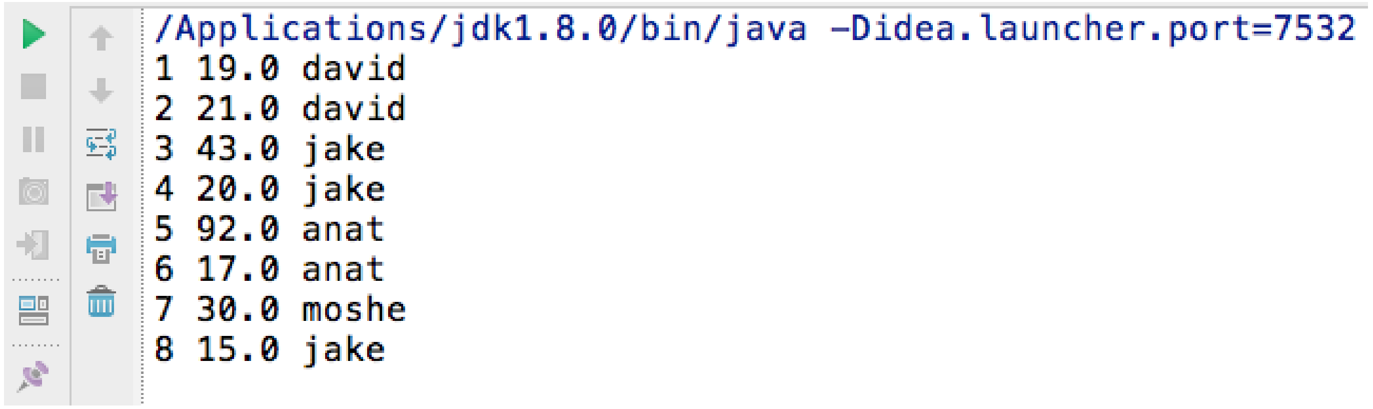
Reference to Constructor

```
Student(int id, double avg, String name)
{
    this.average = avg;
    this.id = id;
    this.name = name;
}

@Override
public String toString()
{
    return id+" "+average+" "+name;
}
```

Reference to Constructor

```
    @Override
     public int compareTo(Student o) {
          if(this.average>o.average) {
               return 1;
          }
          else if(this.average<o.average) {
               return -1;
          }
          else return 0;
     }
}
```

Reference to Constructor



```
/Applications/jdk1.8.0/bin/java -Didea.launcher.port=7532
1 19.0 david
2 21.0 david
3 43.0 jake
4 20.0 jake
5 92.0 anat
6 17.0 anat
7 30.0 moshe
8 15.0 jake
```

The Output