Table of Contents

Author: ungcode
Email: ungcode076@gmail.com
Git: https://github.com/ungcode/java_space.git
Youtube Channel: https://www.youtube.com/channel/UCkIL5-6T1eefpG4aB-JVsTw?view_as=subscriber

Help us create more content. Hit the **like button** and **Subscribe** to the channel

With this guide I intend to teach you Java 8 in easy and clear manner.
For better clarity, follow the guide as the content is listed in Table of Contents

Java 8 - Clear and Quick Guide

Functional Interface

Lambda Expressions

Function and BiFunction Interface

Method Reference

`Functional` Interface

`Available from Java 1.8`

What you should know before learning `Lambada expressions` is the **Functional Interface**

What is a `Functional` Interface ?

An interface which has `no more than one` abstract method.

Use `@FunctionalInterface` to declare an interface as `functional interface`.

Let's see in practice:

```
package com.ung.java.functionalInterface;

/**
 * @author A.M
 * @since Java 1.8
 */
@FunctionalInterface
public interface Greeting {

    void perform(String message); ①

}

public class GreetingImpl implements Greeting {

    // usage I
    @Override
    public void perform(String message) {

        System.out.println(message);

    }

    // usage II
    public Greeting greeting() {

        return new Greeting() {

            @Override
            public void perform(String message) {

                System.out.println(message);
            }
        };
    }

}
```

```
public class Execute {

    public static void main(String[] args) {

        GreetingImpl greeting = new GreetingImpl();
        greeting.perform("Good Morning"); ②
        greeting.greeting().perform("Good Afternoon"); ③

        Greeting g = new GreetingImpl();
        g.perform("Good Night"); ④

    }
}
```

① Functional interface must have only one abstract method

② Output: Good Morning

③ Output: Good Afternoon

④ Output: Good Night

Lambda Expressions

What is a `Lambda` Expression?

1. Lambda expression enables functional programming

Why should you use Lambda expressions ?

1. To provide the implementation of Functional interface
2. Less coding.

This is what you need to be aware of while defining lambda expressions

1. Syntax
   - `parameter` → *expression body*
2. Characteristics
   - **Type declaration** → parameter type are optional
   - **Parenthesis around parameter**
     a. No need to declare a single `parameter` in `parenthesis`.
     b. `Parentheses` are `required` for `multiple parameters`
   - `Curly braces` → ` curly braces` are optional If the `body contain`s a `single statement`
   - Return keyword
     a. The compiler automatically returns the value if the body has a single expression

b. `Curly braces` are required to indicate that expression returns a value.

Let's see Lambda expresions in practice:

```
package com.ung.java.lambda;

@FunctionalInterface
public interface Operation {

    int perform(int x,int y);
}

@FunctionalInterface
public interface Greeting {

    String perform(String message);
}

public class Lambda {


    public static int execute (int x,int y, Operation operation) {

        return operation.perform(x, y);
    }

   public static String execute (String message, Greeting greeting) {

        return greeting.perform(message);
    }

}

public class Execute {

public static void main(String[] args) {

    // With optional parameter type
    Operation addition = (x,y)->x+y;

    // With parameter type
    Operation subtraction = (int x,int y)->x-y;

    // With curly braces
    Operation multiplication = (x,y)->{return x+y;};

    // Without  curly braces and return statement
    Operation division = (int x,int y)->x/y;

    Greeting greeting = (message)-> {return message;};
```

```
    System.out.println("addition: "+ Lambda.execute(8, 2, addition));

    System.out.println("subtraction: "+ Lambda.execute(8, 2, subtraction));

    System.out.println("multiplication: "+ Lambda.execute(8, 2, multiplication));

    System.out.println("division: "+ Lambda.execute(8, 2, division));

    System.out.println("greeting: "+ Lambda.execute("Good morning",greeting));


    }
}
```

Function and BiFunction Interface

java.util.function package contains several `functional interfaces` for general purpose, mostly used by the JDK, but also available to be used by user code as well. In this guide, I will quickly show you how to use two functional interfaces of this package. I expect that by understanding this two functional interfaces you will be able to easily work with other `functional interfaces of java.util.function package`.

Function Interface

Syntax:
Interface Function<T,R>
Parameters:
T → the argument to the function
R → the result of the function

EARTH

```
public class FunctionExample {


    public static void main(String[] args) {

        Planet planet = new Planet();

        //function with apply method
        Function<List<Planet>, Boolean> isEarth  = planets ->planets.get(2).getName()
.equals("EARTH");

        System.out.println("is earth: "+ isEarth.apply(planet.getPlanets()));

        //function with andThen and apply method
        Function<Boolean, Planet>showDetails  = check ->planet.getFeatures(check);

        System.out.println("Earth fearures: "+isEarth.andThen(showDetails).apply
(planet.getPlanets()));



    }
}
```

BiFunction Interface

Syntax:
Interface BiFunction<T,U,R>
Parameters:
T - the first argument to the function
U - the second argument to the function
R - the result of the function

Let's see BiFunction Interface in practice:

```
/**
 * Let's see some java  Bifunction Interfaces
 *
 * @author A.M
 *
 */
public class BiFunctionExample {


    public static void main(String[] args) {

        // 1. BiFunction with apply
```

```
        BiFunction<String, String, String> person = (name,username)->name.concat
(username);

        String fullname = person.apply("John", " Doe");

        System.out.println("After apply: "+fullname);

        // 2. BiFunction with andThen
        Function<String, String> location  = address -> address.contains("John") ?
address : null;
        String address = person.andThen(location).apply("Hauer Landstr 20", " John");

        System.out.println("andThen: "+address);

        // 3. BiFunction with compute

        System.out.println("Initial countries: "+ Planet.getCountries());

        BiFunction<Integer, String, String> newCountry = (key,country) -> country ==
null ? Planet.defaultCountry : country;

        Planet.getCountries().compute(1, newCountry);
        Planet.getCountries().compute(2, newCountry);
        Planet.getCountries().compute(5, newCountry);

        System.out.println("---------------------- ");
        System.out.println("After compute : "+ Planet.getCountries());

        //4. BiFunction with computeIfPresent
        newCountry = (key, country) -> country.concat(Planet.defaultCountry);
        Planet.getCountries().computeIfPresent(4, newCountry);
        Planet.getCountries().computeIfPresent(7, newCountry);

        System.out.println("---------------------- ");
        System.out.println("After computeIfPresent: "+ Planet.getCountries());


        // 5. with merge(key, value, BiFunction)

        BiFunction<String, String, String> merge = (key, country) -> country.concat
(Planet.defaultCountry);
        Planet.getCountries().merge(1, " Peru ", merge);
        Planet.getCountries().merge(2, " Brazil ", merge);
        Planet.getCountries().merge(4, " Mexico ", merge);

        // add if key does not exist
        Planet.getCountries().merge(8, " UK ", merge);

        System.out.println("HashMap using merge() => " + Planet.getCountries());

    }
```

```
}
```

Author: ungcode
Email: ungcode076@gmail.com
Git: https://github.com/ungcode/java_space.git
Youtube Channel: https://www.youtube.com/channel/UCkIL5-6T1eefpG4aB-JVsTw?view_as=subscriber

Help us create more content. Hit the **like button** and **Subscribe** to the channel