Interfaces Funcionales

Desde versiones anteriores conoces las interfaces SAM (Single Abstract Method interfaces). Es decir, interfaces que tienen un solo método abstracto; ejemplo de ellas encontramos a java.lang.Runnable

```
1
2
3
```

```
public interface Runnable {
   public abstract void run();
}
```

Usualmente utilizamos este tipo de interfaz de forma anónima

```
1
2
3
4
5
6
7
8
9
10
```

```
public class ServiceExample {
    public void doWork() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                System.out.println("Running and working
...");
            }
        }).start();
    }
}
```

Java 8 mantiene estas SAM interfaces pero las llama Functional Interfaces. Creando además una anotación específica @FunctionalInterface que le dará la instrucción al compilador para que verifique que la interfaz anotada cumple con los requisitos de una interfaz funcional.

Requisitos de una interfaz funcional.

Las interfaces funcionales tienen la característica de contar con un solo método abstracto.

También pueden contar con más de un método siempre y cuando estos métodos sobrescriban métodos de la clase Object o sean métodos predeterminados, default.

Los métodos predeterminados (marcados con "default") no son abstractos, por lo cual una interfaz funcional puede definir varios métodos predeterminados.

Por ejemplo, veamos esta interfaz 'MyInterface'. Contiene un método funcional **"doSomething",** dos métodos de la clase Object **'toString'** y **'equals'** y un método default **'doSomething2'**

```
1
2
3
4
5
6
7
```

```
@FunctionalInterface
public interface MyInterface {
   public void doSomething();
   default public void doSomething2() {
System.out.print("Hi");};
   public String toString();
   public boolean equals(Object o);
}
```

Si cometemos el error de intentar agregar dos métodos a nuestra interfaz funcional recibiremos este error "multiple non-overriding abstract methods found"

```
1
2
3
4
```

```
@FunctionalInterface
public interface MyInterface {
   public void doSomething(); // metodo funcional
```

```
   default public void doSomething2() {
System.out.print("Hi");}; // metodo default
   public void doSomethingWrong(); // otro metodo
funcional. mal!
   public String toString(); // sobreescribe
   public boolean equals(Object o); // sobreescribe
}
```

Expresiones lambda

Las expresiones lambda son una forma de crear funciones anónimas y que puedes utilzar en dónde el párametro recibido sea una interfaz funcional

Una expresión lambda tiene esta forma

(parameters) ->expression

(parameters) -> { statements; }

Estas son algunas expresiones lambda de ejemplo:

- () -> 5 // directamente devuelve un valor, return 5
- x -> 2 * x // duplica el valor de x y lo retorna
- (x, y) -> x – y // toma dos valores y retorna su diferencia, aqui se infiere el tipo int
- (int x, int y) -> x + y // toma dos enteros y retorna su suma
- (String s) -> System.out.print(s) // toma un string y lo imprime en la consola

Observa lo siguiente:

- Si se tiene un solo parámetro no es obligatorio usar los paréntesis
  x -> 2 * x
- Si no se tienen parámetros o si hay dos o más parámetros debemos usar paréntesis
  () -> 5
  (x, y) -> x – y
- Si la expresión lambda es de una línea no se requieren llaves ni return.
  (String s) -> System.out.print(s)

Las expresiones lambda son funciones anónimas que puedes utilizarlas en donde el parámetro sea una interface funcional.

Por ejemplo, la interface Comparator es una interfaz funcional en el que usas su método abstracto ‘compare’ con expresiones lambda.

```
@FunctionalInterface
public interface Comparator<T> {
int compare(T o1, T o2);
}
```

Puedes entonces crear un comparator utilizando una lambda expresión con esta interfaz funcional de este modo.

```
// JAVA 8 with lambda
Comparator<UserModel> userModelComparator = (UserModel
u1, UserModel u2) ->
u1.getPosition().compareTo(u2.getPosition());

// JAVA 7 lo harias de este modo
Comparator<UserModel> userModelComparator = new
```

```
Comparator<UserModel>() {
   @Override
   public int compare(UserModel u1, UserModel u2) {
       return
u1.getPosition().compareTo(u2.getPosition());
   }
};
```

Algunos ejemplos de uso de lambdas

Prueba estos ejemplos simples en tu IDE para comprender mejor el uso de lambdas y cómo puedes simplicar significativamente tu código

```
public class LambdaExample {

    public static void main(String[] args) {

        List<UserModel> userModels = getUsers();

        // JAVA 8 sort list using lambda
        userModels.sort((UserModel u1, UserModel u2)->
u1.getPosition().compareTo(u2.getPosition()));

        // JAVA 8 Foreach using lambda
        userModels.forEach((userModel)-
>System.out.println(userModel));

        // JAVA 8 Foreach using lambda and reference
method
        userModels.forEach(System.out::println);

        // JAVA 8 Foreach using lambda with sentences
        userModels.forEach(user->{
            if("Gustavo".equals(user.getName())){
                System.out.println("Hi Gustavo!");
            }else{
                System.out.println(user.getName());
            }
        });

        // JAVA 8 Simple Runnable using lambda
        new Thread(() -> System.out.println("Hi
thread")).start();

        // JAVA 8 Runnable using lambda
        Runnable task = () -> {
            userModels.forEach(user->{
                System.out.println( "Hi " +
user.getName() );
            });
        };
        // start the thread
        new Thread(task).start();


    }

    static class UserModel  {

        String name;
        String position;
```

```
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
```

```
        public UserModel(String name, String position)
{
            this.name = name;
            this.position = position;
        }

        public String getName() {return name;}

        public String getPosition() {
            return position;
        }
    }


    static List<UserModel> getUsers() {

        List<UserModel> userModels = new ArrayList<>();
        userModels.add(new UserModel("Gus","A"));
        userModels.add(new UserModel("Harrison","Z"));
        userModels.add(new UserModel("James","G"));
        //.. more..
        return userModels;

    }

}
```