Optional in Java 8: Part 2

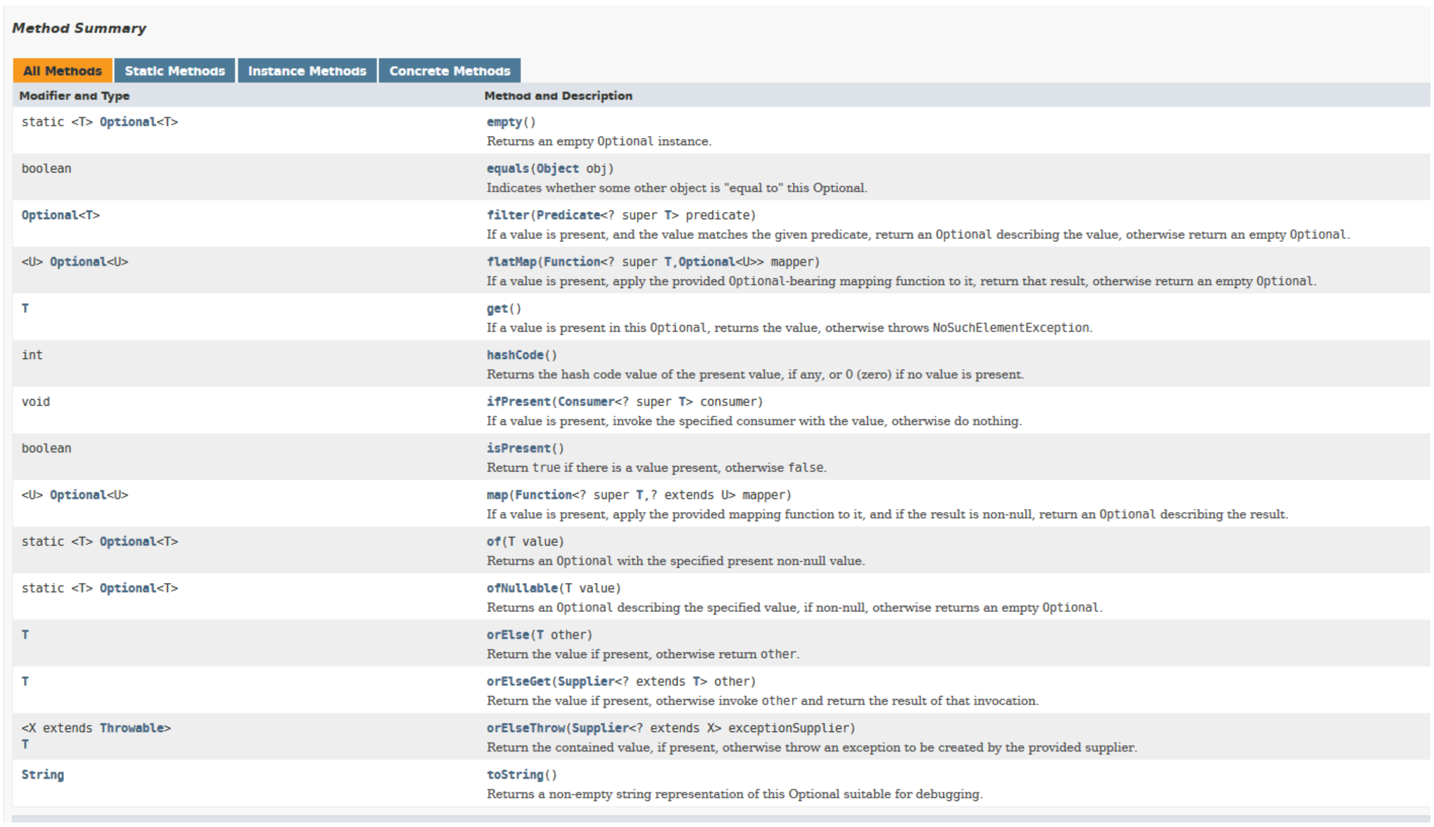
In this lesson, we will look at some of the methods added in Optional class and discuss their functionalities.

We'll cover the following

In the previous lesson, we looked at the `Optional<T>` class. You learned what an `Optional` is and how to create it.

In this lesson, we will look at all the operations that we can perform using an `Optional`.

Below is the list of methods available in the `Optional` class.



Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
static <T> Optional<T>	empty() Returns an empty Optional instance.
boolean	equals(Object obj) Indicates whether some other object is "equal to" this Optional.
Optional<T>	filter(Predicate<? super T> predicate) If a value is present, and the value matches the given predicate, return an Optional describing the value, otherwise return an empty Optional.
<U> Optional<U>	flatMap(Function<? super T,Optional<U>> mapper) If a value is present, apply the provided Optional-bearing mapping function to it, return that result, otherwise return an empty Optional.
T	get() If a value is present in this Optional, returns the value, otherwise throws NoSuchElementException.
int	hashCode() Returns the hash code value of the present value, if any, or 0 (zero) if no value is present.
void	ifPresent(Consumer<? super T> consumer) If a value is present, invoke the specified consumer with the value, otherwise do nothing.
boolean	isPresent() Return true if there is a value present, otherwise false.
<U> Optional<U>	map(Function<? super T,? extends U> mapper) If a value is present, apply the provided mapping function to it, and if the result is non-null, return an Optional describing the result.
static <T> Optional<T>	of(T value) Returns an Optional with the specified present non-null value.
static <T> Optional<T>	ofNullable(T value) Returns an Optional describing the specified value, if non-null, otherwise returns an empty Optional.
T	orElse(T other) Return the value if present, otherwise return other.
T	orElseGet(Supplier<? extends T> other) Return the value if present, otherwise invoke other and return the result of that invocation.
<X extends Throwable> T	orElseThrow(Supplier<? extends X> exceptionSupplier) Return the contained value, if present, otherwise throw an exception to be created by the provided supplier.
String	toString() Returns a non-empty string representation of this Optional suitable for debugging.

1) isPresent()

The `isPresent()` method is used to check if the optional contains a value or if it is null.

The method `isPresent()` returns the value true in case the id of the `Optional` objects contains a non-null value. Otherwise, it returns a false value.

```
Optional<Person> optional = getPerson();
if(optional.isPresent()){
        System.out.println(optional.get.getName())
}
```

2) ifPresent(Consumer<? super T> consumer)

Here is the syntax of `ifPresent()` method.

```
public void ifPresent(Consumer<? super T> consumer)
```

It takes in a `Consumer` as a parameter and returns nothing. When `ifPresent()` is called, if a value is present, the specified consumer is invoked with the value. Otherwise, nothing happens.

```
import java.util.HashMap;
import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

public class StreamDemo {

    Map<Integer, Employee> empMap = new HashMap<>();

    public void populateEmployee() {
        empMap.put(123, new Employee("Alex", 23, 12000));
    }

    public Optional<Employee> getEmployee(Integer employeeId) {
        // Before returning the employee object we are wrapping it into an Optional
        return Optional.ofNullable(empMap.get(employeeId));
    }

    public static void main(String[] args) {
        StreamDemo demo = new StreamDemo();
        demo.populateEmployee();
        Optional<Employee> emp = demo.getEmployee(123);
        emp.ifPresent(System.out::println);
    }
}

class Employee {
    String name;
    int age;
    int salary;

    Employee(String name) {
        this.name = name;
    }

    Employee(String name, int age, int salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public int getSalary() {
        return salary;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", salary=" + salary +
                '}';
    }
}
```

3) get()

The `get()` method returns a value if it is present in this `Optional`. Otherwise, it throws NoSuchElementException.

It is risky to use this method without checking if the value is present or not using `isPresent()` method.

```
import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

public class OptionalDemo {

    public static void main(String[] args) {

        Optional<String> optional = Optional.ofNullable(null);
        // This will throw exception because optional contains a null value.
        System.out.println(optional.get());
    }
}
```

4) orElse(T other)

This method returns the value present in the optional. If no value is present, then a default value provided as a parameter is returned.

```
import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

public class OptionalDemo {

    public static void main(String[] args) {

        Optional<String> optional = Optional.ofNullable(null);
        // This will return the default value.
        System.out.println(optional.orElse("default sting"));
    }
}
```

5) orElseGet(Supplier<? extends T> other)

This method returns the value present in the optional. If no value is present, then the value calculated from the supplier provided as a parameter is returned.

```
import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

public class OptionalDemo {

    public static String getDefaultValue(){
        return "default";
    }

    public static void main(String[] args) {

        Optional<String> optional = Optional.ofNullable(null);
        // This will return the default value.
        System.out.println(optional.orElseGet(OptionalDemo::getDefaultValue));
    }
}
```

6) orElseThrow(Supplier<? extends T> other)

This method returns the value present in the optional. If no value is present, then it throws the exception created by the provided supplier.

```
import java.util.Optional;

public class OptionalDemo {

    public static void main(String[] args) {

        Optional<String> optional = Optional.ofNullable(null);
        // This will throw exception
        try {
            System.out.println(optional.orElseThrow(() -> new Exception("Resource not found.")));
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
}
```

7) Optional<T> filter(Predicate<? super T>

predicate)

The `filter()` method is used to check if the value in our optional matches a particular condition. If yes, then the optional with the value is returned. Otherwise, an empty optional is returned.

```java
import java.util.Optional;

public class OptionalDemo {

    public static void main(String[] args) {

        Optional<String> optional = Optional.ofNullable("orange");
        // Since the filter condition is matched, this will return the optional.
        System.out.println(optional.filter(str -> str.equals("orange")));

        // Since the filter condition is not matched, this will return empty optional.
        System.out.println(optional.filter(str -> str.equals("apple")));

    }
}
```

8) map(Function<? super T, ? extends U> mapper)

As per Java docs, *"if a value is present, apply the provided mapping function to it, and if the result is non-null, return an `Optional` describing the result. Otherwise, return an empty `Optional`."*

```java
import java.util.*;

public class StreamDemo {

    public static void main(String[] args) {

        // Creating an Optional of Employee object.
        Optional<Employee> optional = Optional.of(new Employee("Adam", 54, 20000));

        optional
                .map(emp -> emp.getSalary()) // Fetching the salary from employee object.
                .filter(sal -> sal > 10000) // Checking if the salary is greater than 10000.
                .ifPresent(System.out::println);
    }
}

class Employee {
    String name;
    int age;
    int salary;
    Employee(String name) {
        this.name = name;

    }

    Employee(String name, int age, int salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public int getSalary() {
        return salary;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", salary=" + salary +
                '}';
    }
}
```

9) `flatMap(Function<? super T, Optional<U&>> mapper)`

Similar to the `map()` method, we also have the `flatMap()` method as an alternative for transforming values.

The difference is that the map transforms values only when they are unwrapped, whereas flatMap takes a wrapped value and unwraps it before transforming it.

Let's take the same example that we discussed while looking at `map()`. There is a slight modification though. The `getSalary()` method will return `Optional<Address>`, so the return type of **optional.map(emp -> emp.getSalary())** operation will be `Optional<Optional<Integer>>` .

```
Optional<Optional<Integer>> op1 = optional.map(emp -> emp.getSalary());
```

If we don't need a nested `Optional`, then we can use a `flatMap()`.

```
Optional<Integer> op1 = optional.flatMap(emp -> emp.getSalary());
```

Here is the complete code example.

```
import java.util.*;

public class OptionalDemo {

    public static void main(String[] args) {

        // Creating an Optional of Employee object.
        Optional<Employee> optional = Optional.of(new Employee("Adam", 54, 20000));
        optional.flatMap(emp -> emp.getSalary())
                .filter(sal -> sal > 10000)
                .ifPresent(System.out::println);
    }
}

class Employee {
    String name;
    int age;
    int salary;

    Employee(String name) {
        this.name = name;
    }

    Employee(String name, int age, int salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public Optional<Integer> getSalary() {
        return Optional.of(salary);
    }

    @Override
    public String toString() {
        return "Employee{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", salary=" + salary +
                '}';
    }
}
```

Let’s complete a quiz to review the concepts.

1 `Optional.of()` method throws `NullPointerException` if passed parameter is null?

2 What is `Optional` object used for?

3 Which method can be used to check null on an `Optional` variable in Java 8?

Retake Quiz

In the next lesson, we will learn about slicing operations in `Stream`.