Java Optional Class Methods with Examples

author: Ramesh Fadatare

java 8 optional class

In this tutorial, we will take a look into how to use the Optional class to avoid null checks and NullPointerException.

Well, the NullPointerException is one of the well-known exceptions that frequently occurs in day-to-day project work right and NullPointerException is like a close friend to the java programmer because whenever we write a program or whenever we work on java project then this null pointer exception will frequently occur all right.

Video

Java 8 has introduced a new Optional utility class in `java.util` package. This class can help in avoiding null checks and NullPointerException exceptions.

You can view Optional as a single-value container that either contains a value or doesn't (it is then said to be "empty").

Let's understand some of the frequently or commonly used Java 8 Optional Class methods with examples.

Creating Optional Objects

empty() Method

To create an empty Optional object, we simply need to use its `empty()` static method:

```
Optional<Object> emptyOptional = Optional.empty();
```

of() Method

The `of()` static method returns an Optional with the specified present non-null value.

```
Optional<String> emailOptional = Optional.of("ramesh@gmail.com")
```

ofNullable() Method

The `ofNullable()` static method returns an Optional describing the specified value, if non-null, otherwise returns an empty Optional.

```
Optional<String> stringOptional = Optional.ofNullable("ramesh@gm
```

Here is the complete example with output:

```
import java.util.Optional;

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

        String email = "ramesh@gmail.com";

        // of, empty, ofNullable
        Optional<Object> emptyOptional = Optional.empty();
        System.out.println(emptyOptional);

        Optional<String> emailOptional = Optional.of(email);
        System.out.println(emailOptional);

        Optional<String> stringOptional = Optional.ofNullable(email);
        System.out.println(stringOptional);
    }
}
```

Output:

```
Optional.empty
Optional[ramesh@gmail.com]
Optional[ramesh@gmail.com]
```

Get Value from Optional

get() Method

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

```
package com.java.lambda.optional;

import java.util.Optional;

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

        String email = "ramesh@gmail.com";
        Optional<String> stringOptional = Optional.ofNullable(email);
```

```
}
```

Output:

```
ramesh@gmail.com
```

Checking Value Presence

isPresent() Method

The `isPresent()` method returns true if there is a value present, otherwise false.

```
import java.util.Optional;

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

        String email = "ramesh@gmail.com";
        Optional<String> stringOptional = Optional.ofNullable(email);
        if(stringOptional.isPresent()){
            System.out.println(stringOptional.get());
        }else{
            System.out.println("no value present");
        }
    }
}
```

Output:

```
ramesh@gmail.com
```

Retrieve Default Value

orElse() Method

The orElse() method returns the value if present, otherwise return other (default value).

In the below example, orElse() method return default value because Optional contains null value:

```
import java.util.Optional;

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

        String email = null;
        Optional<String> stringOptional = Optional.ofNullable(email);
        String defaultOptional = stringOptional.orElse("default@gmail.co
        System.out.println(defaultOptional);
    }
}
```

Output:

```
default@gmail.com
```

In the below example, orElse() method return actual value because Optional contains actual value:

```
import java.util.Optional;

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

My Udemy Course - Spring Boot + A

About Me

Hi, I am Ramesh Fadatare. I am VMWare Professional for Spring and Spring Bo

I am founder and author of this blog v JavaGuides, a technical blog dedicated Java/Java EE technologies and Full-St development.

All the articles, guides, tutorials(2000 with me if you have any questions/qu at About Me.

Top YouTube Channel (75K+ Subscri YouTube channel for free videos and Channel

My Udemy Courses - https://www.ude

Connect with me on Twitter, Facebook, andStackOverflow

Follow Me on Twitter

Follow @FadatareRamesh

Facebook Likes and Shares

```
        Optional<String> stringOptional = Optional.ofNullable(email);
        String defaultOptional = stringOptional.orElse("default@gmail.co
        System.out.println(defaultOptional);
    }
}
```

Output:

```
ramesh@gmail.com
```

orElseGet() Method

The `orElseGet()` method returns the value if present, otherwise invoke other and return the result of that invocation.

In the below example, orElseGet() method return default value because Optional contains null value:

```
import java.util.Optional;

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

        String email = null;
        Optional<String> stringOptional = Optional.ofNullable(email);
        String defaultOptional2 = stringOptional.orElseGet(() -> "defaul
        System.out.println(defaultOptional2);
    }
}
```

Output:

```
default@gmail.com
```

In the below example, orElse() method return actual value because Optional contains actual value:

```
import java.util.Optional;

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

        String email = "ramesh@gmail.com";
        Optional<String> stringOptional = Optional.ofNullable(email);
        String defaultOptional2 = stringOptional.orElseGet(() -> "defaul
        System.out.println(defaultOptional2);
    }
}
```

Output:

```
ramesh@gmail.com
```

Handling Exceptions with Optional

orElseThrow() Method

The `orElseThrow()` method returns the contained value, if present, otherwise throw an exception to be created by the provided supplier.

In the below example, we pass a `null` value to the Optional object so `orElseThrow()` method throws an exception to be created by the provided supplier.

```
import java.util.Optional;

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

        String email = null;
        Optional<String> stringOptional = Optional.ofNullable(email);
        String optionalObject = stringOptional.orElseThrow(() -> new Ill
        System.out.println(optionalObject);
    }
}
```

Output:

```
Exception in thread "main" java.lang.IllegalArgumentException: Email is
        at com.java.lambda.optional.OptionalDemo.lambda$main$0(Optional
        at java.base/java.util.Optional.orElseThrow(Optional.java:403)
        at com.java.lambda.optional.OptionalDemo.main(OptionalDemo.java
```

In the below example, we pass a `non-null` value to the Optional object so `orElseThrow()` method returns a value from the Optional object:

```
import java.util.Optional;

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

        String email = "ramesh@gmail.com";
        Optional<String> stringOptional = Optional.ofNullable(email);
        String optionalObject = stringOptional.orElseThrow(() -> new Ill
        System.out.println(optionalObject);
    }
}
```

Output:

```
ramesh@gmail.com
```

Optional filter() and map() Methods

filter() Method

If a value is present, and the value matches the given predicate, return an `Optional` describing the value, otherwise return an empty `Optional` .

```
import java.util.Optional;

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

        // without Optional
        String result = "abc";
        if(result != null && result.contains("abc")){
            System.out.println(result);
        }

        // with Optional
        Optional<String> optionalStr = Optional.of(result);
        optionalStr.filter(res -> res.contains("abc"))
                .ifPresent((res) -> System.out.println(res));
    }
}
```

```
abc
abc
```

map() Method

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.

```
import java.util.Optional;

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

        String result = " abc ";
        if(result != null && result.contains("abc")){
            System.out.println(result);
        }

        Optional<String> optionalStr = Optional.of(result);
        optionalStr.filter(res -> res.contains("abc"))
                .map(String::trim)
                .ifPresent((res) -> System.out.println(res));
    }
}
```

Output:

```
 abc
abc
```

Related Java 8 Features

- Java 8 Lambda Expressions
- Java 8 Functional Interfaces
- Java 8 Method References
- Java 8 Stream API
- Java 8 Optional Class
- Java 8 Collectors Class
- Java 8 StringJoiner Class
- Java 8 Static and Default Methods in Interface

java 8 optional class

JAVA 8 OPTIONAL CLASS

Free Courses on YouTube

Learn Spring Boot
Learn Spring Boot 3
Learn Spring MVC
Learn Spring Data JPA
Learn Spring Boot React
Learn Java Collections
Learn Java 8 in 4 Hours
Learn 25+ Spring Boot Annotations
Spring Boot Kafka Microservices
Learn Building Spring Boot Projects
Learn Spring Boot REST API
Learn Event-Driven Microservices
Learn Spring Boot + Kafka
Learn Spring Boot + Angular
Learn Spring Boot + Thymeleaf

Top Tutorials

Learn Java
Learn Java 8
Learn Spring Boot
Learn Spring Framework
Learn Spring MVC
Learn Spring Security
Learn Spring Data JPA
Learn Spring Annotations
Learn Microservices
Learn REST API
Learn JPA
Learn Hibernate ORM
Learn JSP
Learn Servlet
Learn JUnit
Learn Thymeleaf

Top Quizzes

Java Quiz
Java 8 Quiz
Java Coding Quiz
Spring Boot Quiz
Spring Quiz
Spring MVC Quiz
Spring Data JPA Quiz
Hibernate Quiz
Miroservices Quiz
REST API Quiz
JavaScript Quiz
JavaScript Coding Quiz
React JS Quiz
Kotlin Quiz
SQL Quiz
CSS Quiz
JSON Quiz

My Bestseller Udemy Courses

Spring 6 and Spring Boot 3 for Beginners (Includes Projects)
Building Real-Time REST APIs with Spring Boot
Building Microservices with Spring Boot and Spring Cloud
Full-Stack Java Development with Spring Boot 3 & React
Testing Spring Boot Application with JUnit and Mockito
Master Spring Data JPA with Hibernate
Spring Boot + Apache Kafka - The Quickstart Practical Guide
Spring Boot + RabbitMQ (Includes Event-Driven Microservices)
Spring Boot Thymeleaf Real-Time Web Application - Blog App