Java 8 Optional Class

Difficulty Level : Medium Last Updated : 15 Apr, 2019

Java Optional Class : Every Java Programmer is familiar with NullPointerException. It can crash your code. And it is very hard to avoid it without using too many null checks.
Java 8 has introduced a new class Optional in java.util package. It can help in writing a neat code without using too many null checks. By using Optional, we can specify alternate values to return or alternate code to run. This makes the code more readable because the facts which were hidden are now visible to the developer.

```
// Java program without Optional Class
public class OptionalDemo{
    public static void main(String[] args) {
        String[] words = new String[10];
        String word = words[5].toLowerCase();
        System.out.print(word);
    }
}
```

Output :

```
Exception in thread "main" java.lang.NullPointerException
```

To avoid abnormal termination, we use the Optional class. In the following example, we are using Optional. So, our program can execute without crashing.

The above program using Optional Class

```
// Java program with Optional Class
import java.util.Optional;
public class OptionalDemo{
    public static void main(String[] args) {
        String[] words = new String[10];
        Optional<String> checkNull =
                      Optional.ofNullable(words[5]);
        if (checkNull.isPresent()) {
            String word = words[5].toLowerCase();
            System.out.print(word);
        } else
            System.out.println("word is null");
    }
}
```

Output :

```
word is null
```

Prerequisite : Java 8 Optional Class

Optional is a container object which may or may not contain a non-null value. You must import ***java.util package*** to use this class. If a value is present, **isPresent()** will return true and **get()** will return the value. Additional methods that depend on the presence or absence of a contained value are provided, such as **orElse()** which returns a default value if value not present and **ifPresent()** which executes a block of code if the value is present. This is a ***value-based*** class, i.e their instances are :

- Final and immutable (though may contain references to mutable objects).
- Considered equal solely based on equals(), not based on reference equality(==).
- Do not have accessible constructors.

Static Methods : Static methods are the methods in Java that can be called without creating an object of class. They are referenced by the class name itself or reference to the Object of that class.
Syntax :

```
public static void geek(String name)
{
 // code to be executed....
}

// Must have static modifier in their declaration.
// Return type can be int, float, String or user-defined data type.
```

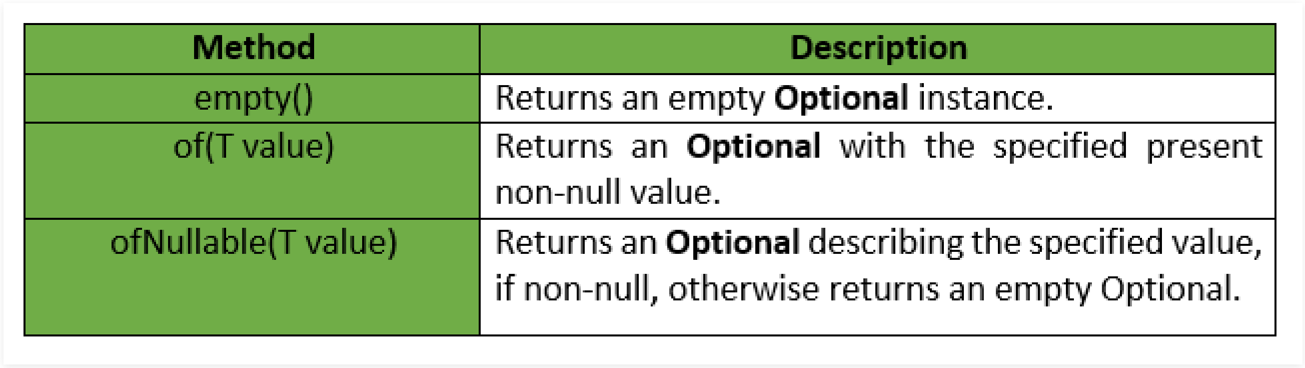
Important Points : Since Static methods belong to the class, so they can be called to without creating the object of the class. Below given are some important points regarding Static Methods :

- Static method(s) are associated with the class in which they reside i.e. they can be called even

without creating an instance of the class.
- They are designed with the aim to be shared among all Objects created from the same class.
- Static methods can not be overridden. But can be overloaded since they are resolved using

  static binding by the compiler at compile time.

Following table shows the list of Static Methods provided by Optional Class :



Method	Description
empty()	Returns an empty **Optional** instance.
of(T value)	Returns an **Optional** with the specified present non-null value.
ofNullable(T value)	Returns an **Optional** describing the specified value, if non-null, otherwise returns an empty Optional.

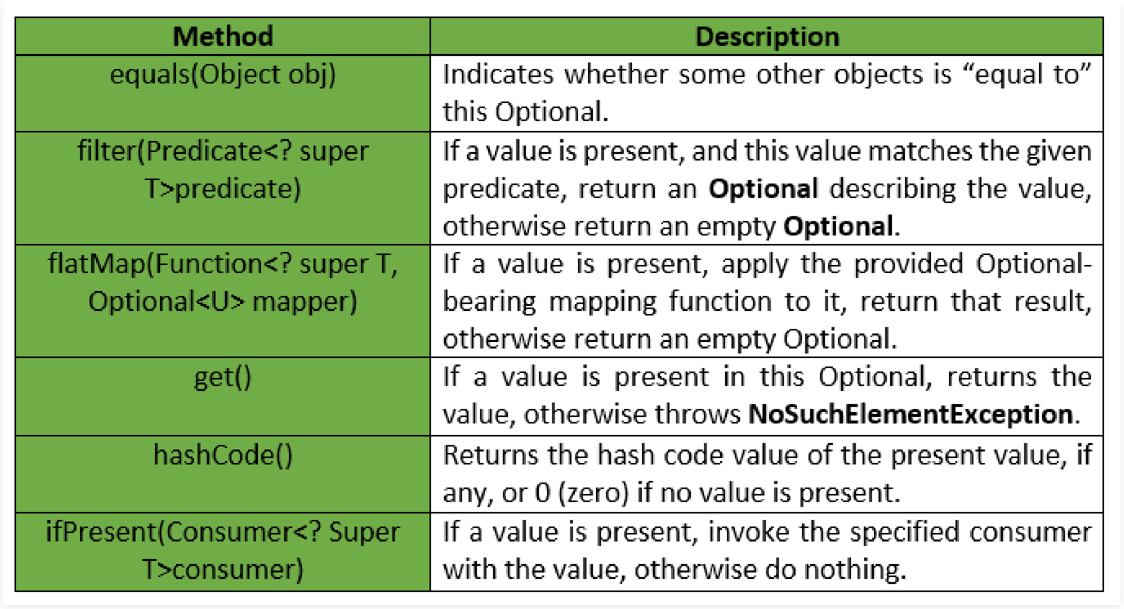
Instance Methods: Instance method are methods which require an object of its class to be created before it can be called. To invoke an instance method, we have to create an Object of the class within which it is defined.
Syntax :

```
public void geek(String name)
{
 // code to be executed....
}
// Return type can be int, float String or user defined data type.
```

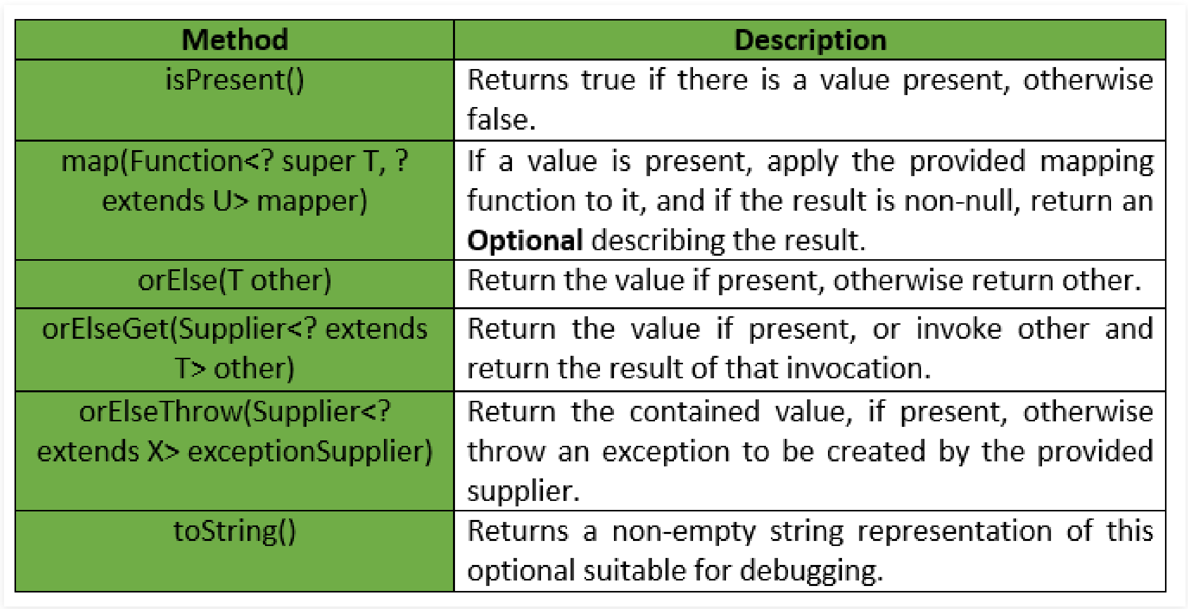
Important Points : Instance Methods can be called within the same class in which they reside or from the different classes defined either in the same package or other packages depend on the access type provided to the desired instance method. Below given are some important points regarding Instance Methods :

- Instance method(s) belong to the Object of the class not to the class i.e. they can be called after creating the Object of the class.
- Every individual Object created from the class has its own copy of the instance method(s) of that class.
- They can be overridden since they are resolved using dynamic binding at run time.

Following table shows the list of Instance Methods provided by Optional Class :

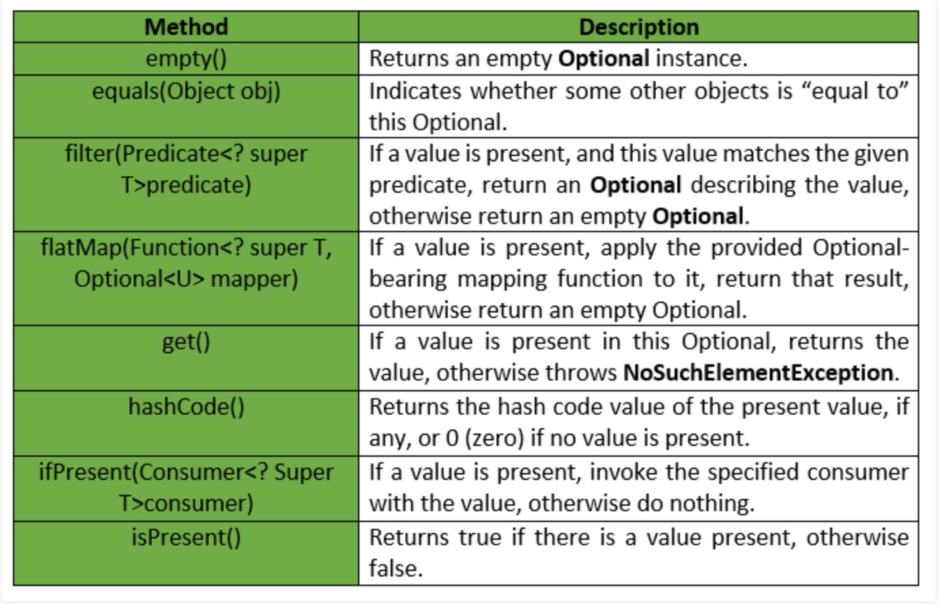


Method	Description
equals(Object obj)	Indicates whether some other objects is "equal to" this Optional.
filter(Predicate<? super T>predicate)	If a value is present, and this value matches the given predicate, return an **Optional** describing the value, otherwise return an empty **Optional**.
flatMap(Function<? 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.
get()	If a value is present in this Optional, returns the value, otherwise throws **NoSuchElementException**.
hashCode()	Returns the hash code value of the present value, if any, or 0 (zero) if no value is present.
ifPresent(Consumer<? Super T>consumer)	If a value is present, invoke the specified consumer with the value, otherwise do nothing.

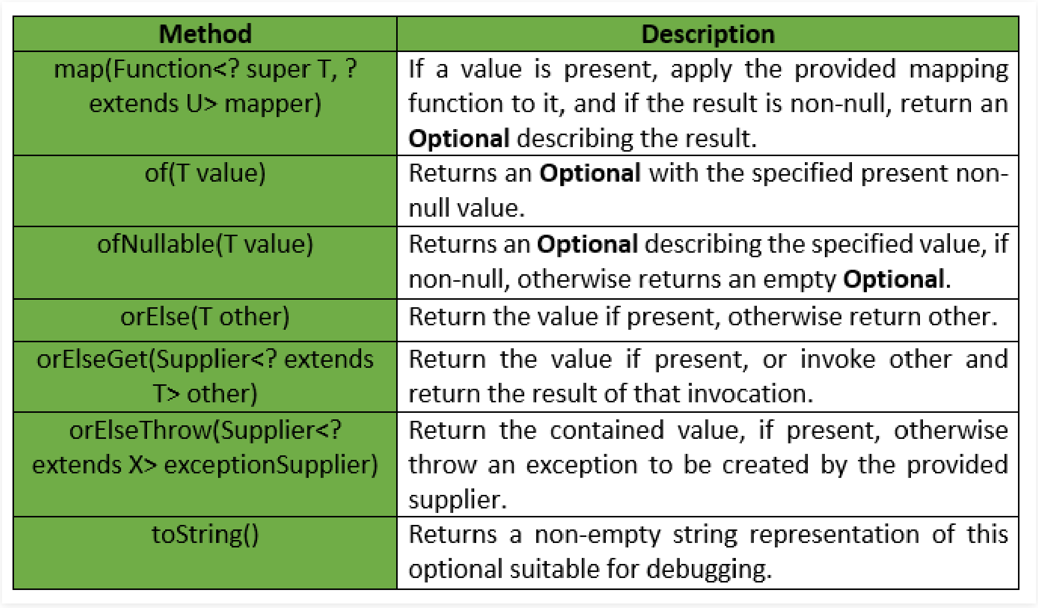


Method	Description
isPresent()	Returns true if there is a value present, otherwise false.
map(Function<? super T, ? extends U> mapper)	If a value is present, apply the provided mapping function to it, and if the result is non-null, return an **Optional** describing the result.
orElse(T other)	Return the value if present, otherwise return other.
orElseGet(Supplier<? extends T> other)	Return the value if present, or invoke other and return the result of that invocation.
orElseThrow(Supplier<? extends X> exceptionSupplier)	Return the contained value, if present, otherwise throw an exception to be created by the provided supplier.
toString()	Returns a non-empty string representation of this optional suitable for debugging.

Concrete Methods : A concrete method means, the method have **complete definition** but it can be overridden in the inherited class. If we make this method **final**, then it can not be overridden. Declaring method or class "final" means it's implementation is complete. It is compulsory to override the abstract methods. ***Concrete Methods can be overridden in the inherited classes if they are not final.***Following table shows the list of Concrete Methods provided by Optional Class :



Method	Description
empty()	Returns an empty **Optional** instance.
equals(Object obj)	Indicates whether some other objects is “equal to” this Optional.
filter(Predicate<? super T>predicate)	If a value is present, and this value matches the given predicate, return an **Optional** describing the value, otherwise return an empty **Optional**.
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.
get()	If a value is present in this Optional, returns the value, otherwise throws **NoSuchElementException**.
hashCode()	Returns the hash code value of the present value, if any, or 0 (zero) if no value is present.
ifPresent(Consumer<? Super T>consumer)	If a value is present, invoke the specified consumer with the value, otherwise do nothing.
isPresent()	Returns true if there is a value present, otherwise false.



Method	Description
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.
of(T value)	Returns an **Optional** with the specified present non-null value.
ofNullable(T value)	Returns an **Optional** describing the specified value, if non-null, otherwise returns an empty **Optional**.
orElse(T other)	Return the value if present, otherwise return other.
orElseGet(Supplier<? extends T> other)	Return the value if present, or invoke other and return the result of that invocation.
orElseThrow(Supplier<? extends X> exceptionSupplier)	Return the contained value, if present, otherwise throw an exception to be created by the provided supplier.
toString()	Returns a non-empty string representation of this optional suitable for debugging.

Below given are some examples :

Example 1 :

```java
// Java program to illustrate
// optional class methods
import java.util.Optional;

class GFG {

    // Driver code
    public static void main(String[] args)
    {

        // creating a string array
        String[] str = new String[5];

        // Setting value for 2nd index
        str[2] = "Geeks Classes are coming soon";
        // It returns an empty instance of Optional class
        Optional<String> empty = Optional.empty();
        System.out.println(empty);

        // It returns a non-empty Optional
        Optional<String> value = Optional.of(str[2]);
        System.out.println(value);
    }
}
```

Output :

```
Optional.empty
Optional[Geeks Classes are coming soon]
```

Example 2 :

```
// Java program to illustrate
// optional class methods
import java.util.Optional;

class GFG {

    // Driver code
    public static void main(String[] args)
    {

        // creating a string array
        String[] str = new String[5];

        // Setting value for 2nd index
        str[2] = "Geeks Classes are coming soon";

        // It returns a non-empty Optional
        Optional<String> value = Optional.of(str[2]);

        // It returns value of an Optional.
        // If value is not present, it throws
        // an NoSuchElementException
        System.out.println(value.get());

        // It returns hashCode of the value
        System.out.println(value.hashCode());

        // It returns true if value is present,
        // otherwise false
        System.out.println(value.isPresent());
    }
}
```

Output :

```
Geeks Classes are coming soon
1967487235
true
```

Reference : Java 8 Optional Class

This article is contributed by **loginakanksha** and **Sahil Bansal** . If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.