compact1, compact2, compact3
java.util

# Class Optional<T>

java.lang.Object
  java.util.Optional<T>

```
public final class Optional<T>
extends Object
```

A container object which may or may not contain a non-null value. 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()` (return a default value if value not present) and `ifPresent()` (execute a block of code if the value is present).

This is a value-based class; use of identity-sensitive operations (including reference equality (`==`), identity hash code, or synchronization) on instances of `Optional` may have unpredictable results and should be avoided.

**Since:**
1.8

## Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

| Modifier and Type | Method and Description |
|---|---|
| static <T> Optional<T> | **empty**()<br>Returns an empty `Optional` instance. |
| boolean | **equals**(Object obj)<br>Indicates whether some other object is "equal to" this Optional. |
| Optional<T> | **filter**(Predicate<? super T> predicate)<br>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)<br>If a value is present, apply the provided `Optional`-bearing mapping function to it, return that result, otherwise return an empty `Optional`. |
| T | **get**()<br>If a value is present in this `Optional`, returns the value, otherwise throws `NoSuchElementException`. |
| int | **hashCode**()<br>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)<br>If a value is present, invoke the specified consumer with the value, otherwise do nothing. |
| boolean | **isPresent**()<br>Return `true` if there is a value present, otherwise `false`. |
| <U> Optional<U> | **map**(Function<? super T,? extends U> mapper)<br>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)<br>Returns an `Optional` with the specified present non-null value. |
| static <T> Optional<T> | **ofNullable**(T value)<br>Returns an `Optional` describing the specified value, if non-null, otherwise returns an empty `Optional`. |
| T | **orElse**(T other)<br>Return the value if present, otherwise return `other`. |
| T | **orElseGet**(Supplier<? extends T> other)<br>Return the value if present, otherwise invoke `other` and return the result of that invocation. |
| <X extends Throwable> T | **orElseThrow**(Supplier<? extends X> exceptionSupplier)<br>Return the contained value, if present, otherwise throw an exception to be created by the provided supplier. |
| String | **toString**()<br>Returns a non-empty string representation of this Optional suitable for debugging. |

### Methods inherited from class java.lang.Object

`clone`, `finalize`, `getClass`, `notify`, `notifyAll`, `wait`, `wait`, `wait`

## Method Detail

### empty

```
public static <T> Optional<T> empty()
```

Returns an empty `Optional` instance. No value is present for this Optional.

**API Note:**
Though it may be tempting to do so, avoid testing if an object is empty by comparing with `==` against instances returned by `Option.empty()`. There is no guarantee that it is a singleton. Instead, use `isPresent()`.

**Type Parameters:**
`T` - Type of the non-existent value

**Returns:**
an empty `Optional`

### of

```
public static <T> Optional<T> of(T value)
```

Returns an `Optional` with the specified present non-null value.

**Type Parameters:**
`T` - the class of the value

**Parameters:**
`value` - the value to be present, which must be non-null

**Returns:**
an `Optional` with the value present

**Throws:**
`NullPointerException` - if value is null

### ofNullable

```
public static <T> Optional<T> ofNullable(T value)
```

Returns an `Optional` describing the specified value, if non-null, otherwise returns an empty `Optional`.

**Type Parameters:**
`T` - the class of the value

**Parameters:**
`value` - the possibly-null value to describe

**Returns:**
an `Optional` with a present value if the specified value is non-null, otherwise an empty `Optional`

### get

```
public T get()
```

If a value is present in this `Optional`, returns the value, otherwise throws `NoSuchElementException`.

**Returns:**
the non-null value held by this `Optional`

**Throws:**
`NoSuchElementException` - if there is no value present

**See Also:**
`isPresent()`

### isPresent

```
public boolean isPresent()
```

Return `true` if there is a value present, otherwise `false`.

**Returns:**
`true` if there is a value present, otherwise `false`

### ifPresent

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

If a value is present, invoke the specified consumer with the value, otherwise do nothing.

**Parameters:**
`consumer` - block to be executed if a value is present

**Throws:**
`NullPointerException` - if value is present and `consumer` is null

### filter

```
public 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`.

**Parameters:**
`predicate` - a predicate to apply to the value, if present

**Returns:**
an `Optional` describing the value of this `Optional` if a value is present and the value matches the given predicate, otherwise an empty `Optional`

**Throws:**
`NullPointerException` - if the predicate is null

### map

```
public <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. Otherwise return an empty `Optional`.

**API Note:**
This method supports post-processing on optional values, without the need to explicitly check for a return status. For example, the following code traverses a stream of file names, selects one that has not yet been processed, and then opens that file, returning an `Optional<FileInputStream>`:

```
    Optional<FileInputStream> fis =
        names.stream().filter(name -> !isProcessedYet(name))
                      .findFirst()
                      .map(name -> new FileInputStream(name));
```

Here, `findFirst` returns an `Optional<String>`, and then `map` returns an `Optional<FileInputStream>` for the desired file if one exists.

**Type Parameters:**
`U` - The type of the result of the mapping function

**Parameters:**
`mapper` - a mapping function to apply to the value, if present

**Returns:**
an `Optional` describing the result of applying a mapping function to the value of this `Optional`, if a value is present, otherwise an empty `Optional`

**Throws:**
`NullPointerException` - if the mapping function is null

### flatMap

```
public <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`. This method is similar to `map(Function)`, but the provided mapper is one whose result is already an `Optional`, and if invoked, `flatMap` does not wrap it with an additional `Optional`.

**Type Parameters:**
`U` - The type parameter to the `Optional` returned by

**Parameters:**
`mapper` - a mapping function to apply to the value, if present the mapping function

**Returns:**
the result of applying an `Optional`-bearing mapping function to the value of this `Optional`, if a value is present, otherwise an empty `Optional`

**Throws:**
`NullPointerException` - if the mapping function is null or returns a null result

### orElse

```
public T orElse(T other)
```

Return the value if present, otherwise return `other`.

**Parameters:**
`other` - the value to be returned if there is no value present, may be null

**Returns:**
the value, if present, otherwise `other`

### orElseGet

```
public T orElseGet(Supplier<? extends T> other)
```

Return the value if present, otherwise invoke `other` and return the result of that invocation.

**Parameters:**
`other` - a `Supplier` whose result is returned if no value is present

**Returns:**
the value if present otherwise the result of `other.get()`

**Throws:**
`NullPointerException` - if value is not present and `other` is null

### orElseThrow

```
public <X extends Throwable> T orElseThrow(Supplier<? extends X> exceptionSupplier)
                                    throws X extends Throwable
```

Return the contained value, if present, otherwise throw an exception to be created by the provided supplier.

**API Note:**
A method reference to the exception constructor with an empty argument list can be used as the supplier. For example, `IllegalStateException::new`

**Type Parameters:**
`X` - Type of the exception to be thrown

**Parameters:**
`exceptionSupplier` - The supplier which will return the exception to be thrown

**Returns:**
the present value

**Throws:**
`X` - if there is no value present
`NullPointerException` - if no value is present and `exceptionSupplier` is null
`X extends Throwable`

### equals

```
public boolean equals(Object obj)
```

Indicates whether some other object is "equal to" this Optional. The other object is considered equal if:

- it is also an `Optional` and;
- both instances have no value present or;
- the present values are "equal to" each other via `equals()`.

**Overrides:**
`equals` in class `Object`

**Parameters:**
`obj` - an object to be tested for equality

**Returns:**
{code true} if the other object is "equal to" this object otherwise false

**See Also:**
`Object.hashCode()`, `HashMap`

### hashCode

```
public int hashCode()
```

Returns the hash code value of the present value, if any, or 0 (zero) if no value is present.

**Overrides:**
`hashCode` in class `Object`

**Returns:**
hash code value of the present value or 0 if no value is present

**See Also:**
`Object.equals(java.lang.Object)`, `System.identityHashCode(java.lang.Object)`

### toString

```
public String toString()
```

Returns a non-empty string representation of this Optional suitable for debugging. The exact presentation format is unspecified and may vary between implementations and versions.

**Overrides:**
`toString` in class `Object`

**Implementation Requirements:**
If a value is present the result must include its string representation in the result. Empty and present Optionals must be unambiguously differentiable.

**Returns:**
the string representation of this instance