

# OPTIOINAL IN JAVA

Prepared by [github.com/all-mp](https://github.com/all-mp)

I

## WHAT IS OPTIONAL ?

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**.

It is a **public final class** and used to deal with **NullPointerException** in Java application.

By using Optional, we can specify alternate values to return or alternate code to run.

II

## HOW TO DEFINE OPTIONAL ?

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

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

How to use ?

```
Optional<String> naser = Optional.of( value: "Naser");
```

```
Optional<String> nullVal = Optional.of( value: null);  
// throws java.lang.NullPointerException
```

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

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

How to use ?

```
Optional<String> Naser = Optional.ofNullable( value: "NASER");
```

```
Optional<String> nullVal = Optional.ofNullable( value: null);
```

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

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

How to use ?

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

get

**public T get()**

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

How to use ?

```
Car optima = new Car( name: "Optima", year: 2020);  
Optional<Car> optionalCar = Optional.of(optima);
```

```
Car car = optionalCar.get();
```

isPresent

**public boolean isPresent()**

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

How to use ?

```
Car optima = new Car( name: "Optima", year: 2020);  
Optional<Car> optionalCar = Optional.of(optima);
```

```
boolean present = optionalCar.isPresent();
```

orElse

**public T orElse(T other)**

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

How to use ?

```
Car optima = new Car( name: "Optima", year: 2020);  
Optional<Car> optionalCar = Optional.of(optima);
```

```
Car orElseCar = optionalCar.orElse(new Car( name: "Maxima", year: 2022));
```

orElseGet

**public T orElseGet(Supplier<? extends T> supplier)**

Supplier

What is Supplier ?

```
@FunctionalInterface  
public interface Supplier<T> {  
  
    T get();  
}
```

How to Define ?

```
Supplier<Car> supplier = new Supplier<Car>() {  
    @Override  
    public Car get() {  
        return new Car( id: 1, name: "Toyota", year: 2016);  
    }  
};
```

```
Supplier<Car> supplier = () -> new Car( id: 1, name: "Toyota", year: 2016);
```

orElseGet

If a value is present, returns **the value**, otherwise returns the result **produced by the supplying function**.

How to use ?

```
Car orElseGetCar = optionalCar.orElseGet(  
    () -> new Car( name: "Maxima", year: 2022)  
);
```

III

## OPTIONAL METHODS

orElseThrow

**public T orElseThrow()**

If a value is present, returns the value, otherwise throws **NoSuchElementException**.

introduced in **Java 10**

How to use ?

```
Car optima = new Car( name: "Optima", year: 2020);  
Optional<Car> optionalCar = Optional.of(optima);
```

```
Car car1 = optionalCar.orElseThrow();
```

**public <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**.

```
Car optima = new Car( name: "Optima", year: 2020);  
Optional<Car> optionalCar = Optional.of(optima);
```

```
Car car2 = optionalCar.orElseThrow(RuntimeException::new);
```

ifPresent

**public void ifPresent(Consumer<? super T> action)**

Consumer

Subtopic

```
@FunctionalInterface  
public interface Consumer<T> {  
  
    Performs this operation on the given argument.  
    Params: t - the input argument  
    void accept(T t);  
}
```

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

How to use ?

```
Car optima = new Car( name: "Optima", year: 2020);  
Optional<Car> optionalCar = Optional.of(optima);
```

```
optionalCar.ifPresent(car -> car.setName("naser"));
```

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.

How to use ?

```
Car optima = new Car( name: "Optima", year: 2020);  
Optional<Car> optionalCar = Optional.of(optima);
```

```
Optional<Car> naser = optionalCar.filter(  
    car -> car.getName().equals("Optima"));
```

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.

How to use ?

```
Car optima = new Car( name: "Optima", year: 2020);  
Optional<Car> optionalCar = Optional.of(optima);
```

```
Optional<String> carName = optionalCar.map(Car::getName);
```

flatMap