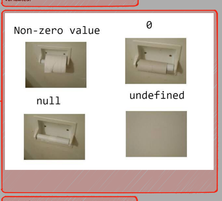
Optional created by github.com/farzadafi

Null

null is a special literal that represents the absence of a value. It is often used to indicate that a reference variable does not currently refer to any object in memory.

null is not the same as an object with a value of 0, false, or an empty string. null is a special value that represents the absence of any value, whereas 0, false, and an empty string are actual values that can be stored in variables.



Attempting to use a null reference can result in a NullPointerException at runtime, which is a type of unchecked exception that is thrown when a program attempts to access an object through a null reference.

It is important to always check if a reference variable is null before attempting to use it, to avoid these runtime errors.

Optional define

introduced in java 8 :)

Optional is a container object that may or may not contain a non-null value.

It is used to represent the absence of a value in a type-safe manner,

and provides a way to avoid null checks and NullPointerExceptions in your code.

Optional doesn't miracle, just prevents Null pointer Exception, you have to attention to your code still

Creaate and Vs Null

create

- Optional.empty() — Returns an empty Optional object.
- Optional.of(value) — Returns an Optional object containing the specified non-null value.
- Optional.ofNullable(value) — Returns an Optional object containing the specified value, or an empty Optional if the value is null.
- Optional.of VS Optional.ofNullable — Optional.of assumes that the value passed to it is non-null, while Optional.ofNullable allows for the value to be null and creates an empty Optional in that case. Therefore, Optional.of is useful when you know for sure that a value is non-null, while Optional.ofNullable is useful when a value may be null and you want to handle both cases in a uniform manner.
  It's important to note that both Optional.of and Optional.ofNullable return an Optional instance

Null

```
public String findString(String str, List<String> strings) {
    for (String s : strings
    ) {
        if (s.equals(str))
            return s;
    }
    return null;
}
```

Optional

```
public Optional<String> findString(String str, List<String> strings) {
    for (String s : strings
    ) {
        if (s.equals(str))
            return Optional.of(s);
    }
    return Optional.empty();
}
```

method

get()

If a value is present, returns the value. Otherwise, throws a NoSuchElementException.

```
Optional<String> optional = Optional.of( value: "farzad");
String value = optional.get();
System.out.println(value);  // todo -> farzad
```

isPresent()

It is used to determine whether an Optional instance contains a non-null value or not.

```
Optional<String> optional = Optional.of( value: "farzad");
if (optional.isPresent())
    System.out.println(optional.get());  //todo -> farzad
else
    System.out.println("Optional is empty");
```

isEmpty()

It was introduced in Java 11 and is used to determine whether an Optional instance is empty or not.

This method provides a more concise and readable alternative to calling !isPresent().

```
Optional<String> optional = Optional.of( value: "farzad");
if (optional.isEmpty())
    System.out.println("Optional is empty");
else
    System.out.println(optional.get());  //todo -> farzad
```

ifPresentOrElse(Consumer<? super T> action, Runnable emptyAction)

If a value is present, performs the given action with the value. Otherwise, performs the given empty-based action.

```
Optional<String> optional = Optional.of( value: "farzad");
optional.ifPresentOrElse(System.out::println,  // todo - consumer to be executed if value is present
        () -> System.out.println("Optional is empty") // todo - Runnable to be executed if value is absent
);
```

In this context, the Runnable passed as the second argument to ifPresentOrElse() method is a functional interface that defines a single method called run(), which takes no arguments and returns no value.

Using a Runnable in this way allows you to define a block of code that can be executed in a deferred manner, which can be useful in cases where you want to execute different blocks of code depending on whether a value is present or absent from the Optional.

filter(Predicate<? super T> predicate)

If a value is present, and the value matches the given predicate, returns an Optional describing the value. Otherwise, returns an empty Optional.

```
Optional<String> optional = Optional.of( value: "farzad");
Optional<String> filteredOptional = optional.filter(value -> value.contains("far"));
if (filteredOptional.isPresent())
    System.out.println(optional.get());  //todo farzad
else
    System.out.println("Optional is empty");
```

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

If a value is present, applies the given mapping function to the value and returns an Optional describing the result. Otherwise, returns an empty Optional.

```
Optional<String> optional = Optional.of( value: "farzad");
Optional<Integer> lengthOptional = optional.map(String::length);
if (lengthOptional.isPresent())
    System.out.println("Length of string: " + lengthOptional.get());  //todo Length of string: 6
else
    System.out.println("Optional is empty");
```

flatMap(Function<? super T, ? extends Optional<? extends U>> mapper)

If a value is present, applies the given mapping function to the value and returns the result. Otherwise, returns an empty Optional.

```
Optional<String> optional = Optional.of( value: "farzad");
Optional<Integer> lengthOptional = optional.flatMap(value -> Optional.of(value.length()));
if (lengthOptional.isPresent())
    System.out.println("Length of string: " + lengthOptional.get());  //todo Length of string: 6
else
    System.out.println("Optional is empty");
```

The flatMap() method is used to apply a function to the value contained in an Optional and return the result as an Optional. Unlike map(), the function passed to flatMap() returns an Optional.

```
Optional<String> optional = Optional.of( value: "farzad");
Optional<Integer> lengthOptionalMap = optional.map(String::length);

Optional<Integer> lengthOptionalFlatMap = optional.flatMap(value -> Optional.of(value.length()));
```

or(Supplier<Optional<T>>)

If the original Optional instance is not empty, the method simply returns the original Optional. If the original Optional instance is empty, the method invokes the Supplier to obtain a new Optional instance. If the Supplier returns a non-empty Optional, the or method returns that Optional. If the Supplier returns an empty Optional, the or method returns an empty Optional.

```
Optional<String> optional = Optional.empty();
String result = optional.or(() -> Optional.of( value: "default")).get();
System.out.println(result);  // todo default
```

stream()

The stream method of the Stream interface is used to convert a stream source into a Stream object for further processing using stream operations. When applied to an Optional object, the stream method returns a Stream object that contains either zero or one elements, depending on whether the Optional object is empty or contains a value.

```
Optional<String> optional = Optional.of( value: "farzad");
optional.stream().forEach(System.out::println);
```

orElse(T other)

If a value is present, returns the value. Otherwise, returns the specified other value.

```
Optional<String> optional = Optional.empty();
String result = optional.orElse( other: "default");
System.out.println(result);  // todo default
```

orElseGet(Supplier<? extends T> other)

If a value is present, returns the value. Otherwise, returns the result of the given supplier function.

```
Optional<String> optional = Optional.empty();
String result = optional.orElseGet(() -> {
    // Some expensive computation
    return "default";
});
System.out.println(result);  // todo default
```

orElseThrow(Supplier<? extends X> exceptionSupplier)

If a value is present, returns the value. Otherwise, throws an exception produced by the given supplier function.

```
Optional<String> optional = Optional.empty();
String result = optional.orElseThrow(
        () -> new IllegalArgumentException("Value is empty"));
```

primitive data type

Optional is Generic, and Java create this classes for primitive data type :)

- java.util.OptionalInt

```
int[] arr = {1, 2, 3, 4, 5};
OptionalInt optionalInt = Arrays.stream(arr).filter(n -> n % 2 == 0).findFirst();

if (optionalInt.isPresent())
    System.out.println("First even number found: " + optionalInt.getAsInt());
else
    System.out.println("No even numbers found");
```

- java.util.OptionalLong
- java.util.OptionalDouble