Java 8 Optional – Replace your get() calls

June 19, 2016 (https://reversecoding.net/java-8-optional-replace-get-examples/) Mario Pio Gioiosa (https://reversecoding.net/author/mariocidio/) Java 8 (https://reversecoding.net/category/java-8/)

Optional class were introduced in order to prevent *NullPointerException*, but method *get()* used to retrieve the value inside the Optional might still throw a *NoSuchElementException*.

Different name, same issue?

Calling *get()* without checking that value is actually present it's a bug. So we should always write something like that in order to use *get().*

```
Optional<String> myString = Optional.ofNullable(getNullableString());
  if(myString.isPresent()){
      doSomething(myString.get());
  }
```

But are Optional really meant to be used in this way? No.

Writing block of i*sPresent/get* it's not so different from writing a classic null check.

```
String myString = getNullableString();
  if(myString != null){
      doSomething(myString);
  }
```

Let's see how we can really benefit from Optional object.

1. Optional *orElse* example

It returns the value if is present, or the other specified otherwise.

Let's see an example:

```
    @Test
    public void orElse_whenNamePresent_ThenName(){
        Optional<String> petName = Optional.of("Bobby");

        assertEquals("Bobby", petName.orElse(""));
    }
```

```
    @Test
    public void orElse_whenNameNotPresent_ThenEmptyString(){
        Optional<String> petName = Optional.empty();

        assertEquals("", petName.orElse(""));
    }
```

As you can see we haven't called *get()* and we've made the code easier and more readable compared to the *isPresent/get* version:

```
    @Test
    public void isPresentGet_whenNamePresent_ThenName(){
        Optional<String> petNameOptional = Optional.of("Bobby");

        String petName = "";
        if(petNameOptional.isPresent()){
             petName = petNameOptional.get();
        }

        assertEquals("Bobby", petName);
    }

    @Test
    public void isPresentGet_whenNameNotPresent_ThenEmptyString(){
        Optional<String> petNameOptional = Optional.empty();

        String petName = "";
        if(petNameOptional.isPresent()){
             petName = petNameOptional.get();
        }

        assertEquals("", petName);
    }
```

2. Optional *orElseThrow* example

It returns the value if is present, or throws the specified exception otherwise.

```
    @Test
    public void elseOrThrow_whenNamePresent_ThenName(){
        Optional<String> petName = Optional.of("Bobby");

        assertEquals("Bobby", petName.orElse(""));
    }

    @Test(expected=IllegalArgumentException.class)
    public void elseOrThrow_whenNameNotPresent_ThenIllegalArgEx(){
        Optional<String> petName = Optional.empty();

        petName.orElseThrow(IllegalArgumentException::new);
    }
```

3. Optional *filter* example

filter() is useful to specify other conditions on our object. It returns an Optional containing the value if is not empty and satisfy the specified predicate, an empty Optional otherwise.

In this example we want that the name not only is different from *null* but also that is not empty or made of only empty spaces.

```
    @Test
    public void filter_whenNameNotEmpty_thenName(){
        Optional<String> petNameOpt = Optional.of("Bobby");

        String petName = petNameOpt.filter(name -> !name.trim().isEmpty())
                                   .orElseThrow(IllegalArgumentException::new);

        assertEquals("Bobby", petName);
    }
```

And those are the tests for the null and the empty name:

```
@Test(expected=IllegalArgumentException.class)
    public void filter_whenNameNotPresent_thenIllegalArgEx(){
        Optional<String> petNameOpt = Optional.empty();

        petNameOpt.filter(name -> !name.trim().isEmpty())
                  .orElseThrow(IllegalArgumentException::new);
    }

    @Test(expected=IllegalArgumentException.class)
    public void filter_whenNameEmpty_thenIllegalArgEx(){
        Optional<String> petNameOpt = Optional.of(" ");

        petNameOpt.filter(name -> !name.trim().isEmpty())
                  .orElseThrow(IllegalArgumentException::new);
    }
```

4. Optional *ifPresent* example

IfPresent (https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html#ifPresent-java.util.function.Consumer-), that it's different from *isPresent*, accept a function, a Consumer, and executes it only if the value is present.

So instead of writing something like:

```
if(optional.isPresent){
  doSomething(optional.get)
}
```

You can write:

```
optional.ifPresent(val -> doSomething(val))
```

or if you prefer:

```
optional.ifPresent(this::doSomething)
```

But let's have a look to a proper example.

We define a Pojo class, useful also for the following examples, that represents a Loyalty card.

```
public class LoyaltyCard {

    private String cardNumber;

    private int points;

    public LoyaltyCard(String cardNumber, int points){
        this.cardNumber = cardNumber;
        this.points = points;
    }

    public int addPoints(int pointToAdd){
        return points += pointToAdd;
    }

    //Getters

}
```

We want to add 3 points to the loyalty card if loyalty card is actually present.

Node: In the following example we're going to use Mockito to mock LoyaltyCard class. Don't worry if you are not familiar with Mockito, I'll add some comments to the code.

```
    @Test
    public void ifPresent_whenCardPresent_thenPointsAdded(){
        LoyaltyCard mockedCard = mock(LoyaltyCard.class);
        Optional<LoyaltyCard> loyaltyCard = Optional.of(mockedCard);

        loyaltyCard.ifPresent(c -> c.addPoints(3));

        //Verify addPoints method has been called 1 time and with input=3
        verify(mockedCard, times(1)).addPoints(3);
    }
```

5. Optional *map* example

map() it's a method that we use to transform an input in a different output. In this case nothing changes except that the map operation will be executed only if the value is actually present, otherwise it returns an empty *Optional*.

In this example we want to retrieve the number of points of our loyalty card if we have it, otherwise number of point will return 0.

```
    @Test
    public void map_whenCardPresent_thenNumber(){
        LoyaltyCard mockedCard = mock(LoyaltyCard.class);
        when(mockedCard.getPoints()).thenReturn(3);

        Optional<LoyaltyCard> card = Optional.of(mockedCard);

        int point = card.map(LoyaltyCard::getPoints)
                        .orElse(0);

        assertEquals(3, point);
    }
```

```
    @Test
    public void map_whenCardNotPresent_thenZero(){
        Optional<LoyaltyCard> card = Optional.empty();

        int point = card.map(LoyaltyCard::getPoints)
                        .orElse(0);

        assertEquals(0, point);
    }
```

6. Optional *flatMap* example

flatMap() it's really similar to *map()* but when output is already an *Optional* it doesn't wrap it with another *Optional*. So instead of having *Optional<Optional<T>>* if will just return *Optional<T>*.

Let me clarify it using an example. Let's define a new class, called *Gift*.

```
public class Gift {

    private String name;

// Constructor and getters

}
```

And let's define a new method to our LoyaltyCard class that returns an *Optional* containing the last *Gift* chosen. Since we are going to mock the result of this method, we don't really care about its implementation.

```
    public Optional<Gift> getLastGift(){
        //whatever
        return Optional.empty();
    }
```

We can now create a mocked *Gift* with name “Biography of Guybrush Threepwood”, put it into an *Optional* and make *getLastGift* return it. So if we write:

```
card.map(LoyaltyCard::getLastGift)
```

Output will be an *Optional<Optional<Gift>>* that is not what we want, so flatMap will unwrap this double level and leave only an *Optional<Gift>.*

```
    @Test
    public void flatMap_whenCardAndLastGiftPresent_thenName(){
        Gift mockedGift = mock(Gift.class);
        when(mockedGift.getName()).thenReturn("Biography of Guybrush Threepwood");

        LoyaltyCard mockedCard = mock(LoyaltyCard.class);
        when(mockedCard.getLastGift()).thenReturn(Optional.of(mockedGift));
        Optional<LoyaltyCard> card = Optional.of(mockedCard);

        String giftName = card.flatMap(LoyaltyCard::getLastGift)
                              .map(Gift::getName)
                              .orElse("");

        assertEquals("Biography of Guybrush Threepwood", giftName);
    }
```

Writing this solution by using *isPresent/get* would have meant using a nested if: one for check that card was present and another of checking the gift. Harder to read, easier to fail.

7. Optional *ifPresentOrElse ?*

Unfortunately this is yet to come 🙂 It will be available in Java 9 (http://mail.openjdk.java.net/pipermail/core-libs-dev/2015-February/031223.html).

Until then we have to write something like:

```
        if(optional.isPresent()){
            doSomething(optional.get());
        } else {
            doSomethingElse();
        }
```

There are cases in which you are allowed to use *get()* and *isPresent()* but use them with a grain of salt.

Resources:

JavaDoc (https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html)

Short Tutorial By Example (http://www.javaspecialists.eu/archive/Issue238.html)

Share this:

(https://reversecoding.net/java-8-optional-replace-get-examples/?share=twitter&nb=1)

(https://reversecoding.net/java-8-optional-replace-get-examples/?share=facebook&nb=1)

(https://reversecoding.net/java-8-optional-replace-get-examples/?share=linkedin&nb=1) More

Related

(https://reversecoding.net/spring-mvc-requestparam-binding-request-parameters/) Spring MVC @Requestparam - Binding request parameters (https://reversecoding.net/spring-mvc-requestparam-binding-request-parameters/)
In this article, we are going to see several examples on how to get request parameters with Spring MVC, how to bind them to different objects, how to use @RequestParam annotation and when the annotation is not needed.
February 26, 2017
In "Spring MVC"

(https://reversecoding.net/java-8-list-to-map/) Java 8 Stream - From List to Map (https://reversecoding.net/java-8-list-to-map/)
An example to convert a List<?> to a Map<K,V> using Java 8 Stream. Java 8 - Collectors.toMap() Let's define a Pojo class: public class Person { private String email; private String name; private int age; public Person(String email,
June 4, 2016
In "Java 8"

(https://reversecoding.net/java-8-convert-list-string-comma/) Java 8 - Convert List to String comma separated (https://reversecoding.net/java-8-convert-list-string-comma/)
Convert a List<String> in a String with all the values of the List comma separated in Java 8 is really straightforward. Let's have a look how to do that. In Java 8: We simply can write String.join(..), pass a delimiter and an
May 22, 2016
In "Java 8"

example (https://reversecoding.net/tag/example/) filter (https://reversecoding.net/tag/filter/) flatMap (https://reversecoding.net/tag/flatmap/) Java 8 (https://reversecoding.net/tag/java-8/) map (https://reversecoding.net/tag/map/) Optional (https://reversecoding.net/tag/optional/) orElse (https://reversecoding.net/tag/orelse/) orElseThrow (https://reversecoding.net/tag/orelsethrow/)

One thought to “Java 8 Optional – Replace your get() calls”

DEV
July 25, 2016 at 6:57 pm (https://reversecoding.net/java-8-optional-replace-get-examples/#comment-2)

In the case we want a check for NullPointer/NoSuchElement exceptions, why not stuff a default value in the read mutator and be done with it?
E.g., petName = petNameOptional.get().orElse("");

Thus, ever null/not found.

REPLY (HTTPS://REVERSECODING.NET/JAVA-8-OPTIONAL-REPLACE-GET-EXAMPLES/?REPLYTOCOM=2#RESPOND)

LEAVE A REPLY

Enter your comment here...

Java 8 Stream – From List to Map (https://reversecoding.net/java-8-list-to-map/)

Java 8 Comparator – How to sort a List (https://reversecoding.net/java-8-comparator-how-to-sort-a-list/)

Privacv