Optional in Java

Optional

Optional is an object that can be think of as a container to hold other objects. It can contain null or an instance of the class.

Why is this object needed? Well there are many places in your code that can return the object or null, i.e. your custom find method for example. And if you call method on the null object this is where you will be seeing `NullPointerException`.

So the normal way of dealing with this type of project is just to add a if-statement check like such:

```
Cat c = Database.findCat("Lucy");
if (c != null) {
c.meow();
} else {
System.out.println("No cat found");
}
```

Optional is created to be more explicit telling the programmer that hey the method that returns the value you ask for might or might not return null, so you have to be prepared to deal with it. No worries, optional also provides some API that deals with null pretty easily.

Let's take a look at some of them.

ofNullable and of

To actually create an optional object you wouldn't use the constructor but instead use the static method, `ofNullable` or `of`.

They accept in the object that you want to put into the Optional container.

Use `ofNullable` if the object can be null.

Use `of` if the object you are 100% will not be null. Otherwise it will be a `NullPointerException` :)

public T get()

Now you created your Optional object how do you get the object out of it if it has any? These following API are use to do that.

Now this method shouldn't really be used since the creator of Optional regret adding this into the API because it is so tempting to use it like such.

```
Optional<Cat> catOpt = Optional.ofNullable(cat);
catOpt.get().meow();
```

What if `cat` is null? You will again get `NullPointerException`

public boolean isPresent()

You can use this method to tell whether or not the value is present, it will return `true` if it is present, otherwise `false`.

You can use `isPresent()` with `get()`

```
Optional<Cat> catOpt = Optional.ofNullable(cat);
if (catOpt.isPresent()) {
catOpt.get().meow();
}
```

But this is pretty much the same god damn code without Optional right? Right, but you make it clear with Optional that you the programmer acknowledged the fact that the object you get out from the catOpt could be null, as oppose to without Optional, you aren't sure if it can be null or not.

Rather than using `get()` and `isPresent()`, there are other more useful API methods.

public T orElse(T other)

This method can be called on an Optional object to retrieve the value inside it, if it doesn't contain one, i.e. it is null, then the `other` is returned.

As an example:

```
Optional<Cat> catOpt = Optional.ofNullable(cat);
Cat catOutTheBox = catOpt.orElse(new Cat("Default"));
System.out.println(catOutTheBox.getName());
```

Now as you can see, if the cat exists in the Optional object, then it will return that and that cat's name from Optional will be printed. However, if `cat` is null, then a default cat is returned with name "Default".

public Optional<U> map(mapper)

Now this method is super interesting because you can use it to map your Optional object into something else and get a Optional object of that type back.

In more formal terms, if the value exists in the called Optional object, it will apply the mapping function (could also be a lambda function, and if the result is non-null, it will return an Optional object of that particular type. Otherwise, if the value doesn't exist, return an empty Optional of that type.

Let's look at the example:

```
Optional<Cat> catOpt = Optional.ofNullable(catOpt);
Optional<String> outOpt = catOpt.map(Cat::getName);
```

In this case, the map function calls getName on the Cat object inside the Optional if it exists, and return out an Optional of type String because getName returns a String. So ultimately you get back an Optional that may or may not contains a String depending on whether or not `catOpt` contains the Cat object.

Revision #1
Created 2023-04-15 16:14:08 UTC by Tamarine
Updated 2023-04-15 21:02:13 UTC by Tamarine