Java 8 Optional Type

GURUTEAM
specialist ICT learning

What is wrong with Null?

- Given the following structure

- What does the NullPointerException indicate?

```
public class NullPointerExample {
    public static void main(String[] args) {
        Trader t = new Trader("ABC123", "John");
        ...
        System.out.println(
            t.getPortfilio().getAsset(0).getSymbol().getValue());
    }
}
```

```
Exception in thread "main" java.lang.NullPointerException
        at ms.optional.NullPointerExample.main(NullPointerExample.java:12)
```

Optional<T>

- Sometimes need to represent "no value"
  - Map lookup
  - empty collection/stream
- Optional is an alternative to null
  - reduces likelihood of NullPointerException
  - reduces need for special handling of results
- Type advertises the possibility of no value
  - but can wrap a value
- Defined in java.util

Optional<T> Variables

- Can specify variables of type Optional<T>

```
class Processor {
  public Integer calc(Optional<Integer> x, Optional<Integer> y) {
      System.out.println("'x' is present: " + x.isPresent());
      System.out.println("'y' is present: " + y.isPresent());

      // Optional.orElse - returns the value if present otherwise
      // returns the default value passed.
      Integer v1 = x.orElse(new Integer(0));

      // Optional.get - gets the value, value should be present
      // otherwise will throw NoSuchElementException
      Integer v2 = y.get();
      return v1 + v2;
  }
}
```

Working with Optional

- Use `of` and `ofNullable` factory methods

```
public class OptionalExample1 {
    public static void main(String[] args) {

        Processor p = new Processor();

        Optional<Integer> a1 = Optional.of(3);
        Optional<Integer> a2 = Optional.of(5);
        System.out.println("p.calc(a1, a2):" + p.calc(a1, a2));

        Optional<Integer> a3 = Optional.ofNullable(null);
        System.out.println("p.calc(a3, a2): "
                           + p.calc(a3, a2));
      }
}
```

```
'x' is present: true
'y' is present: true
p.calc(a1, a2):8
'x' is present: false
'y' is present: true
p.calc(a3, a2): 5
```


Example – Map lookup

```
…
 Map<String, String> capitals = new HashMap<>();
 capitals.put("UK", "London");
 capitals.put("USA", "Washington DC");
 capitals.put("India", "New Delhi");

 Map<String, Integer> populations = new HashMap<>();
 populations.put("London", 8_000_000);
 populations.put("Washington DC", 6_000_000);
 populations.put("Beijing", 20_000_000);
…
```

```
…
 System.out.println(capitals.get("UK").toUpperCase());
…
 System.out.println(capitals.get("China").toUpperCase());
…
```

```
LONDON
Exception in thread "main"
java.lang.NullPointerException
       at Optional1.main(Optional1.java:34)
```

Example – Map lookup

- Change retrieval of element from Map
  - do not return `null`, but give indication of "no value"

```
…
  public static Optional<String> getCap ( String key ) {
    return Optional.ofNullable(capitals.get(key));
  }
…
```

- `Optional<String>` type has two possibilities
  - a `String` value
  - `empty`

```
…
 System.out.println(getCap("UK"));
 System.out.println(getCap("China"));
…
```

```
Optional[London]
Optional.empty
```


`Optional<T>` functions

- Similar approach to processing Streams
  - no special handling for `empty` case

```
…
 System.out.println(getCap("UK").map(String::toUpperCase));
 System.out.println(getCap("China").map(String::toUpperCase));
…
```

```
Optional[LONDON]
Optional.empty
```

- Value can be extracted

```
…
System.out.println(getCap("UK").map(String::toUpperCase)
                               .orElse(""));
System.out.println(getCap("China").map(String::toUpperCase)
                               .orElse(""));
…
```

```
LONDON
```

Chaining Optional functions

- Chain methods returning `Optional<T>` with `flatMap()`

```
…
 public static Optional<Integer> getPop ( String key ) {
  return Optional.ofNullable( populations.get(key) );
 }
…
```

```
…
 System.out.println( getCap("UK")                    // Optional<String>
                        .flatMap(populations::getPop)
                        .orElse(-1));                // Optional<Integer>

 System.out.println( getCap("China")
                        .flatMap(populations::getPop)
                        .orElse(-1));
…
```

```
8000000
-1
```


Using `Optional<T>` with Streams

- Provides convenient way of handling empty Stream

```
…
 public static int largest(List<Integer> l ) {
  return l.stream()
          .sorted((x,y) -> y - x)
          .collect(Collectors.toList())
          .get(0);
 }
…
```

```
…
 List<Integer> values = Arrays.asList(1,-4, -3, 4, -10, 11, 0);
 List<Integer> empty = new ArrayList<>();
…
 System.out.println( largest(values) );
 System.out.println( largest(empty) );
…
```

```
11
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index: 0, Size: 0
        at java.util.ArrayList.rangeCheck(ArrayList.java:653)
        at java.util.ArrayList.get(ArrayList.java:429)
  …
```

Using `Optional<T>` with Streams

- Provides convenient way of handling empty Stream

```
…
 public static int largestPositive(List<Integer> l ) {
  return l.stream()
          .filter(i -> i >= 0 )
          .sorted((x,y) -> y - x)
          .collect(Collectors.toList())
          .get(0);
 }
…
```

```
…
 List<Integer> values = Arrays.asList(1,-4, -3, 4, -10, 11, 0);
 List<Integer> allNegative = Arrays.asList(-1, -4, -3, -4,-10, -11);
…
 System.out.println( largestPositive(values) );
 System.out.println( largestPositive(allNegative) );
…
```

```
11
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index: 0, Size: 0
        at java.util.ArrayList.rangeCheck(ArrayList.java:653)
        at java.util.ArrayList.get(ArrayList.java:429)
…
```


Using `Optional<T>` with Streams

- Use alternative method to extract head of Stream

```
…
 public static Optional<Integer> safeLargest(List<Integer> l ) {
  return l.stream()
          .sorted((x,y) -> y - x)
          .findFirst();
 }
…
```

```
…
public static Optional<Integer> safeLargestPositive(
                                        List<Integer> l ) {
  return l.stream()
          .filter(i -> i >= 0 )
          .sorted((x,y) -> y - x)
          .findFirst();
 }
…
```

Using `Optional<T>` with Streams

- Values now processed successfully

```
…
 System.out.println( safeLargest(values)
                                    .map( n -> n * n)
                                    .orElse(-1) );
 System.out.println( safeLargest(empty)
                                    .map( n -> n * n)
                                    .orElse(-1) );
 System.out.println( safeLargestPositive(values)
                                    .map( n -> n * n)
                                    .orElse(-1) );
 System.out.println( safeLargestPositive(allNegative)
                                    .map( n -> n * n)
                                    .orElse(-1) );
…
```

```
121
-1
121
-1
```


Method Summary

Method	Type	Use
empty	static	Returns empty Optional instance
of	static	Returns Optional containing non-null value
ofNullable	static	Returns Optional containing non-null or empty
get	instance	Returns value or throws Exception
orElse	instance	Returns value if present or returns alternative
orElseGet	instance	Returns value if present or invokes supplied function, returns value from that
orElseThrow	instance	Returns value if present or throws specified Exception
isPresent	instance	Returns true if Option is not empty
ifPresent	instance	If value is present invoke supplied function
map	instance	Applies function to contents
flatMap	instance	Applies supplied function to contents and flattens the result
filter	instance	Returns a value if optional meets criteria