Java 8: Does “Optional” class alleviate the pain of ubiquitous NullpointerException?

Posted on November 8, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

The most prevalent runtime exception (aka unchecked) exception in Java is the ***NullPointerException***. You need to code defensively to avoid **NullPointerException** as shown below. Later, we will see how the class ***java.util.Optional*** that was introduced in Java 8 will alleviate this problem to some extent.

Pre Java 8 without the **java.util.Optional** class

```
1  public class Person {
2
3   private String name;
4   private int age;
5
6   private HealthInsurance insurance;
7
8   //getters and setter
9  }
10
```

```
1 public class HealthInsurance {
2
3  private String name;
4  private Extra[] extras;
5
6  //getters and setters
7 }
8
```

```
1 public class Extra {
2
3  private String name;
4
5  //getters and setters
6 }
7
```

Now, here is the main class that invokes printExtras(Person person), where you need to check for null

```
1  public class PersonTest {
2
3   public static void main(String[] args) {
4
5    Person person1 = new Person();
6    person1.setName("John");
7    printExtras(person1);
8
9    Person person2 = new Person();
10   person2.setName("Peter");
11
12   HealthInsurance healthIns = new HealthInsuranc
13   Extra[] healthExtras = { new Extra("Physio"),
14   healthIns.setExtras(healthExtras);
15
16   person2.setInsurance(healthIns);
17   printExtras(person2);
18
19  }
20
21  private static void printExtras(Person person)
22   if (person != null &amp;amp;&amp;
23    person.getInsurance() != null
24    &amp;amp;&amp; person.getInsurance().getExtras()
25
```

```
26     Extra[] extras = person.getInsurance().getExt
27     for (Extra extra : extras) {
28      System.out.println(extra.getName());
29     }
30    }
31
32   }
33 }
34
```

In Java 8 with **java.util.Optional** class

Take note of the Optional class.

```
1  package com.java8.examples;
2
3  import java.util.Optional;
4
5  public class Person {
6
7   private String name;
8   private int age;
9   private Optional<HealthInsurance> insurance = O
10
11  public String getName() {
12   return name;
13  }
14  public void setName(String name) {
15   this.name = name;
16  }
17  public int getAge() {
18   return age;
19  }
20
21  public void setAge(int age) {
22   this.age = age;
23  }
24
25
26  public Optional<HealthInsurance> getInsurance()
27   return insurance;
28  }
29  public void setInsurance(Optional<HealthInsuran
30   this.insurance = insurance;
31  }
32 }
33
```

```
1  package com.java8.examples;
2
3  import java.util.Optional;
4
5  public class  HealthInsurance{
6
7   private String name;
8   private Optional<Extra[]> extras = Optional.emp
9
10  public String getName() {
11   return name;
```

```
12  }
13
14  public void setName(String name) {
15   this.name = name;
16  }
17
18  public Optional<Extra[]> getExtras() {
19   return extras;
20  }
21
22  public void setExtras(Optional<Extra[]> extras)
23   this.extras = extras;
24  }
25 }
26
```

```
1  package com.java8.examples;
2
3  public class Extra {
4
5   private String name;
6
7   public Extra(String name) {
8    super();
9    this.name = name;
10  }
11
12  public String getName() {
13   return name;
14  }
15 }
16
```

Finally, the main class that make use of the Optional.of methods, and the printExtras(Optional person) is much simpler with functional programming and lambda expressions.

```
1  package com.java8.examples;
2
3  import java.util.Arrays;
4  import java.util.Collections;
5  import java.util.List;
6  import java.util.Optional;
7  import java.util.function.Consumer;
8
9  public class PersonTest {
10
11  public static void main(String[] args) {
12
13   Person person1 = new Person();
14   person1.setName("John");
15
16   Optional<Person> p1 = Optional.of(person1);
17   printExtras(p1);
18
19   Person person2 = new Person();
20   person2.setName("Peter");
21
```

```
22    // health ins and extras are supplied
23    HealthInsurance healthIns = new HealthInsuranc
24    Extra[] healthExtras = { new Extra("Physio"),
25
26    healthIns.setExtras(Optional.of(healthExtras))
27    person2.setInsurance(Optional.of(healthIns));
28
29    Optional<Person> p2 = Optional.of(person2);
30    printExtras(p2);
31
32   }
33
34   private static void printExtras(Optional<Person
35    person
36         .flatMap(Person::getInsurance)
37         .flatMap(HealthInsurance::getExtras)
38         .ifPresent((p) -> {for(int i = 0; i < p.le
39                           System.out.println(p[i].
40                   }
41                           });
42
43       }
44 }
45
```

Both approaches outputs

Physio
Optical
Chiro

Here are the convenient methods that the **Optional** class support

- ifPresent( ): returns true if a value is present in the optional.
- get( ): returns a reference to the item contained in the optional object, if present, otherwise throws a ***NoSuchElementException***.
- ifPresent(Consumerconsumer): passes the optional value, if present, to the provided ***Consumer*** (lambda expression or method reference).
- orElse(T other): returns the value, if present, otherwise returns the value in other.
- orElseGet(Supplier other): returns the value if present, otherwise returns the value provided by the Supplier (lambda expression or method reference).
- orElseThrow(Supplier exceptionSupplier): returns the value if present,

otherwise throws the exception provided by the ***Supplier*** **(**lambda expression or method reference**)**

Note: In Java you cannot simply get rid of the null references that have historically existed, but before Java 8, you need to rely on your code and proper documentation to understand if a reference is optional. ***java.util.Optional*** class has been added to deal with optional object references. The intention with the Optional class is not to replace every null-able reference, but to help in the creation of more robust APIs you could tell if you can expect an optional reference by reading the signature of a method.

Popular Posts

♦ 11 Spring boot interview questions & answers

825 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

765 views

18 Java scenarios based interview Questions and Answers

400 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

388 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

295 views

♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

285 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

279 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

239 views

001B: ♦ Java architecture & design concepts interview questions & answers

201 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ Java 8: What is currying? Does Java 8 support currying?

03: ♦ Functional interfaces and Lambda expressions Q&A ›

Posted in Java 8, member-paid

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category

© Disclaimer

 ↑