Theory: NPE

10 minutes 4 / 7 problems solved

Start practicing

§1. What is NPE

Java provides a special type of value called `null` to indicate that no actual value is assigned to a reference variable. This value may cause one of the most frequent exceptions called `NullPointerException` (often referred to as **NPE** for short). It occurs when a program attempts to use a variable with the `null` value. To avoid **NPE**, the programmer must ensure that the objects are initialized before the use.

Here is one interesting fact about the concept of `null` reference and errors associated with it. Not only is it not unique for Java, but in 2009, Tony Hoare, a British Computer Scientist who invented the concept of `null` reference, described it as **"billion-dollar mistake"**:

> I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

Let's look at some situations where **NPE** might occur and find out how to avoid it.

§2. NPE when invoking a method

Since `String` is a regular reference type, it's variable can be `null`. If we invoke a method or apply an operation to such variable, the code throws **NPE**.

In the following code, an uninitialized instance of `String` is created and then the method `length()` is invoked. The code throws **NPE** because the object `str` is actually `null`.

```
1 String someString = null; // a reference type can be null
2
3 int size = someString.length(); // NullPointerException (NPE)
```

The same exception will occur if we will use uninitialized instances of any other reference type, not only `String`.

To avoid the exception we should explicitly check whether a string is `null` or not and depending on the result perform different code. It's similar to the default value.

```
1
int size = str != null ? str.length() : 0; // if the string is empty, the size is
0
```

In the code above, when the given string is `null`, the size is set as 0. This way we won't get any exceptions.

§3. Comparing strings

A very common situation occurs when we try to compare a `String` variable and a string literal.

Current topic:

✓ NPE

Topic depends on:

✓ Ternary operator Stage 2

✗ What is an exception Stage 7

Topic is required for:

Array exceptions

Boxing and unboxing

Optional

File hierarchies

Table of contents:

```
String str = null;

if (str.equals("abc")) { // it throws NPE
    System.out.println("The same");
}
```

To avoid **NPE** here we can call the equals method on literal rather than the object:

```
 String str = null;

if ("abc".equals(str)) { // no NPE here
    System.out.println("The same");
}
```

But what if we have two variables of the type `String`? Any of them may happen to be `null`. In this case, we can use the special auxiliary class `java.util.Objects`.

```
String s1 = null;
String s2 = null;

if (Objects.equals(s1, s2)) { // no NPE here
    System.out.println("Strings are the same");
}
```

This approach is recommended in modern Java programming since it is easy for reading and does not throw **NPE**.

§4. Rules for avoiding NPE

We've considered a few cases when **NPE** may occur. Actually, there are more of such situations, and we will consider them in the next topics.

Here are several general rules on how to avoid **NPE** in your programs:

- for reference types, use a condition statement to check whether the given variable is `null` before using it;
- try to avoid assigning `null` to variables whenever possible;
- use NPE-safe features from the standard library.

These simple rules will help to reduce the number of places in your code that could throw this exception.

Report a typo

480 users liked this theory. **1** didn't like it. **What about you?**

Start practicing

Comments (11) Hints (0) Useful links (1) Show discussion