Theory: What is an exception

10 minutes 0 / 5 problems solved

Some errors in your code do not prevent the program from running. In this case, the program will only crash while trying to execute a "broken" line: a line with an error called an **exception**. Thus, exceptions are the errors detected during the program **execution** (at runtime), whereas syntax errors are those detected during **compiling** the program into byte-code. An exception interrupts the normal execution of a program.

Let's consider several kinds of exceptions and how to avoid them.

§1. ArithmeticException

Suppose you are writing a program that reads two integers from the standard input and then outputs the result of the integer division of the first number by the second one. Look at the code below.

```
import java.util.Scanner;

public class ArithmeticExceptionDemo {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int a = scanner.nextInt();
        int b = scanner.nextInt();



        System.out.println(a / b); // an exception is possible here!

        System.out.println("finished");

    }

}
```

If the user passes `9` and `3` as the input values, the program outputs `3` as well as the `"finished"` string. But if the second number is `0`, the program throws an exception because of division by zero.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
  at ArithmeticExceptionDemo.main(ArithmeticExceptionDemo.java:11)
```

As you can see, the program fails with the `ArithmeticException` and the `"finished"` string is not printed at all. All the code **before** the exception is executed properly, and everything **after** is not.

The displayed message shows the cause of the exception (`"/ by zero"`), the file and the line number where it has occurred (`ArithmeticExceptionDemo.java:11`). The provided information is useful for developers, but it is not very meaningful for the end-users of the program.

To avoid the exception, we can check the value before the division, and, if the value is zero, print a message. Here is another version of the program. If the second number is zero, the program will print the "**Division by zero!**" string.

Current topic:
What is an exception Stage 7

Topic depends on:
✓ Conditional statement Stage 2
✓ Errors in programs Stage 7

Topic is required for:
Hierarchy of exceptions Stage 7
✓ NPE
Match results

Table of contents:
↑ What is an exception
§1. ArithmeticException
§2. NumberFormatException
§3. Conclusion
Feedback & Comments

```
import java.util.Scanner;

public class ArithmeticExceptionDemo {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int a = scanner.nextInt();
        int b = scanner.nextInt();



        if (b == 0) {

            System.out.println("Division by zero!");

        } else {

            System.out.println(a / b);

        }

        System.out.println("finished");

    }

}
```

Look at some input examples. Let's start with non-zero integers:

```
8 4
```

The program still works in the same way and the result is:

```
2
finished
```

Now if we try to input zero as a divider:

```
3 0
```

The result is:

```
Division by zero!
finished
```

As you can see, the new version of the program does not throw an exception and always successfully finishes. Additionally, it prints a user-friendly message instead of the standard message.

§2. NumberFormatException

Another situation to consider is when you are trying to convert a string into an integer number. If the string has an unsuitable format, the code will throw an exception.

The following program reads a number from the standard input and then outputs the number that follows it.

```
import java.util.Scanner;

public class NumberFormatExceptionDemo {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();


int number = Integer.parseInt(input); // an exception is possible here!

        System.out.println(number + 1);

    }

}
```

It works pretty well if the input line is a correct integer number. But if the input is not correct (e.g. `"121a"`), the program will fail:

```

Exception in thread "main" java.lang.NumberFormatException: For input string: "121
a"

 at java.base/java.lang.NumberFormatException.forInputString(NumberFormatExceptio
n.java:65)
    at java.base/java.lang.Integer.parseInt(Integer.java:652)
    at java.base/java.lang.Integer.parseInt(Integer.java:770)
    at NumberFormatExceptionDemo.main(NumberFormatExceptionDemo.java:9)
```

This message displays the type of exception (`NumberFormatException`) and the passed input string. The place where the exception occurred is shown in the last line of the message:

- the filename is `NumberFormatExceptionDemo.java`;
- the `main` method;
- the line 9.

All the previous lines of the message show the positions inside the `parseInt` method that is the part of the standard library.

To avoid this exception, it is possible to check the input string by using a regular expression. In case the input is not correct we can output a warning message. The following program does this:

```
import java.util.Scanner;

public class NumberFormatExceptionDemo {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();


if (input.matches("\\d+")) { // it checks the input line contains only dig
its

            int number = Integer.parseInt(input);

            System.out.println(number + 1);

        } else {

            System.out.println("Incorrect number: " + input);

        }

    }

}
```

If the input line is "121a", the program will not stop, and it will print the message:

```
Incorrect number: 121a
```

Don't be sad if you do not know regular expressions. Just remember this trick.

If you have trouble understanding what the exception is, you can always copy-paste its name and google it. Moreover, you're strongly encouraged to do it, as 99% of troubles that learners encounter have already been solved on professional forums.

§3. Conclusion

- Exceptions do not prevent a program from being compiled and run, but the program crashes as soon as the line with an exception is being executed.
- There are many types of exceptions.
- You can use control statements to avoid some kinds of exceptions (like `ArithmeticException` or `NumberFormatException`) in your programs.
- There is a general approach to handle exceptions and even throw them by yourself that you will learn in the next lesson.

Report a typo

521 users liked this theory. **5** didn't like it. **What about you?**

Start practicing

Comments (12) Hints (0) Useful links (0) Show discussion