Theory: Reading files

16 minutes 0 / 5 problems solved

The standard Java class library provides several ways to read data from files. Some of them are quite old, others have appeared recently. In this topic, we will consider only two basic methods, which is quite enough for now. You can choose the one that is most suitable for you.

§1. Reading data using Scanner

It is possible to use `java.util.Scanner` to read data from files. This class is a high-level approach to read input data. It allows reading primitive types or strings by using regular expressions.

First of all, we need to create an instance of `java.io.File` and then an instance of `Scanner` passing the file object. After that, we can get data from the file by using the scanner in the same way as we read from the standard input.

Suppose you have a string called `pathToFile`. It keeps the path to a file that contains a sequence of numbers separated by spaces.

Let's create a file object and then a scanner to read data from the file.

```
File file = new File(pathToFile);

Scanner scanner = new Scanner(file); // it throws FileNotFoundException (checked)
```

When you create an instance of `Scanner` passing a file, you must handle the checked exception `FileNotFoundException`. You can also declare the method as throwing this exception.

Now, we can use methods of `Scanner` to read data as strings, integers and so on.

Let's read a line from the file

```
while (scanner.hasNext()) {
    System.out.print(scanner.nextLine());
}
```

This code reads each line from the file and outputs it to the standard output.

After using a scanner, we should close the object to avoid leaks. A convenient way to close scanners and handle exceptions is to use the **try-with-resources** statement as below. Read more about **try-with-resources** in the official tutorial.

The full example of `Scanner` usage is presented below:

```
File file = new File(pathToFile);

try (Scanner scanner = new Scanner(file)) {
    while (scanner.hasNext()) {
        System.out.print(scanner.nextLine() + " ");
    }
} catch (FileNotFoundException e) {
    System.out.println("No file found: " + pathToFile);
}
```

Then, for a file containing:

```
first line
second line
third line
```

The program outputs to console the following result:

Current topic:

Reading files

Topic depends on:

Stage 7
✕ Exception handling

✕ Files

Table of contents:

```
first line second line third line
```

The scanner also allows you to read integers, boolean, doubles, and other types. Methods have corresponding names like `nextInt`, `nextBoolean` and etc. In case of no new data is available any of `next` methods returns `java.util.NoSuchElementException`.

Instead of displaying the read data in the standard output, you can store it in an array or a string.

§2. Reading all text from a file as a single string

Since Java 1.7 there is a set of new classes and methods for handling files. Within this topic, we will confine ourselves to learning how to read an entire text file. Note that this method should be used only for *small text files*. By small we mean that their size is smaller than JVM available RAM. That's more than enough for learning and performing small tasks.

First of all, make the following imports:

```
import java.nio.file.Files;
import java.nio.file.Paths;
```

The `Files` class consists of methods that operate on files, the `Paths` class contains a set of methods that return a special object to represent the path to a file.

The following method returns all text from a specified file:

```

public static String readFileAsString(String fileName) throws IOException {
    return new String(Files.readAllBytes(Paths.get(fileName)));
}
```

Let's try to use the method `readFileAsString` to read the source code from the file `HelloWorld.java` and print it to the standard output. `HelloWorld.java` contains a traditional basic program mentioned in one of the earliest topics "The first program".

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class ReadingFileDemo {

 public static String readFileAsString(String fileName) throws IOException {
        return new String(Files.readAllBytes(Paths.get(fileName)));
    }


    public static void main(String[] args) {

        String pathToHelloWorldJava = "/home/username/Projects/hello-
world/HelloWorld.java";

        try {

            System.out.println(readFileAsString(pathToHelloWorldJava));

        } catch (IOException e) {

            System.out.println("Cannot read file: " + e.getMessage());

        }

    }

}
```

It prints the source code:

```java
package org.hyperskill;

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

Note that it is not difficult to modify the code above so that it would read the path to the target file from the standard input instead of hardcoding it.

Report a typo

385 users liked this theory. **14** didn't like it. **What about you?**

Start practicing

This content was created over 2 years ago and updated 3 days ago. Share your feedback below in comments to help us improve it!

Comments (15) Hints (0) Useful links (0) Show discussion