Lab 6. Files and Exceptions

Authors: Yida Tao

Reference: Core Java Volume I. Cay S. Horstmann

Working with Files

Please download `FilesExample.java`. You should execute the code and make sure that you understand each of the output.

To fully understand Java File I/O mechanism and available APIs, please also refer to the official documentation.

Exception Flow

Download and modify `ExceptionDemo.java` as follows. Observe how the results differed.

- Case 1: The code throws no exceptions. In this case, the program first executes all the code in the `try` block. Then, it executes the code in the `finally` clause. Afterwards, execution continues with the first statement after the `finally` clause.

```
try
{
    InputStream in = new FileInputStream("exist-file");
    System.out.println("End of try.");
}
catch (IOException e)
{
    System.out.println("Catch begins.");
}
finally
{
    System.out.println("Finally.");
}
System.out.println("After finally.");
```

- Case 2a: The code throws an exception that is caught in a `catch` clause - in our case, an `IOException`. For this, the program executes all code in the `try` block, up to the point at which the exception was thrown. The remaining code in the `try` block is skipped. The program then executes the code in the matching `catch` clause, and then the code in the `finally` clause. If the `catch` clause does not throw an exception, the program executes the first line after the `finally` clause.

```
try
{
    InputStream in = new FileInputStream("nonexist-file");
    System.out.println("End of try.");
}
catch (IOException e)
```

```
        {
            System.out.println("Catch begins.");
        }
        finally
        {
            System.out.println("Finally.");
        }
        System.out.println("After finally.");
```

- Case 2b: If the `catch` clause throws an exception, then the exception is thrown back to the caller of this method after `finally` clause executes.

```
    public static void main(String[] args) throws FileNotFoundException {

        try
        {
            InputStream in = new FileInputStream("nonexist-file");
            System.out.println("End of try.");
        }
        catch (IOException e)
        {
            System.out.println("Catch begins.");
            InputStream in = new FileInputStream("nonexist-file");
            System.out.println("End of catch");
        }
        finally
        {
            System.out.println("Finally.");
        }
        System.out.println("After finally.");

    }
```

- Case 3: The code throws an exception that is not caught in any `catch` clause. Here, the program executes all code in the `try` block until the exception is thrown. The remaining code in the try block is skipped. Then, the code in the `finally` clause is executed, and the exception is thrown back to the caller of this method.

```
        try
        {
            InputStream in = new FileInputStream("exist-file");
            String s = null;
            s.length();

            System.out.println("End of try.");
        }
        catch (IOException e)
        {
            System.out.println("Catch begins.");
```

```
            InputStream in = new FileInputStream("nonexist-file");
            System.out.println("End of catch");
        }
        finally
        {
            System.out.println("Finally.");
        }
        System.out.println("After finally.");

    }
```

Method Call Chain and Stack Trace

A stack trace is a listing of all pending method calls at a particular point in the execution of a program. You have almost certainly seen stack trace listings — they are displayed whenever a Java program terminates with an uncaught exception.

You could use the `StackWalker` class that yields a stream of `StackWalker.StackFrame` instances, each describing one stack frame.

The `StackWalker.StackFrame` class has methods to obtain the file name and line number, as well as the class object and method name, of the executing line of code. The `toString` method yields a formatted string containing all of this information.

Run `StackTraceTest.java` to print the stack trace of a recursive function.