

Object Oriented Programming

Week 7 Part 2 Defining Exceptions

Lecture

- Throwing Exceptions
- Finally clause

Throwing exceptions

- A method may throw an exception
- For example we could check that the string is not empty and that it has only digits before passing it to `Integer.parseInt`
 - This example is silly because `Integer.parseInt` already does this test.
 - It do, however illustrate how to throw an exception

Example: Throw an exception

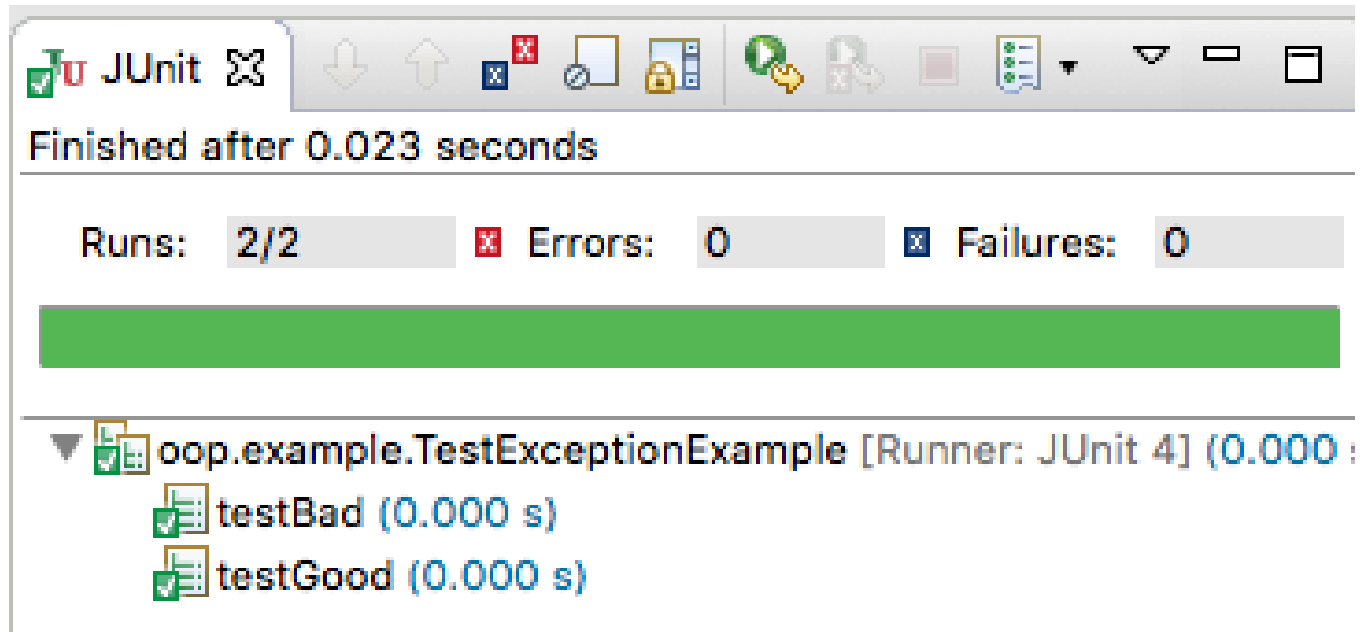
```
public class ExceptionExample {  
    public ExceptionExample() {  
    }  
  
    public int string2Int(String s)  
    throws NumberFormatException {  
        for (char c: s.toCharArray()) {  
            if (!Character.isDigit(c) ||  
                s.length() == 0) {  
                throw new NumberFormatException(  
                    "For input string: \"" + s + "\"");  
            }  
        }  
        return Integer.parseInt(s);  
    }  
}
```

Make sure non-empty string is all digits

If it is throw a new NumberFormatException
Parameter is message

This example is silly. `Integer.parseInt` already does this test and throws a `NumberFormatException`. It shows how to throw an exception and sets up a later example.

Example Result



Both tests still pass because we did not change the behavior of the class

A Better Example

- Suppose we wanted to read a value from a file and then convert what we read to an int.
- In this case we could have both a `NumberFormatException` and a `FileNotFoundException`

Common Exceptions

- In package java.lang
 - NullPointerException
 - NumberFormatException
 - ArithmeticException
 - SecurityException
- In Package java.io
 - FileNotFoundException
 - IOException

Finally clause

- The block following the finally call is guaranteed to run even if there is an exception
- Allows the function to clean up anything partial results
 - Particularly useful when opening files
 - Files can be closed when there is an exception

Better Example

Open file may throw FileNotFoundException →

Reading may throw IOException →

Catch exceptions and throw them →

Finally block is guaranteed to run →

Clean up files if opened →

```
public int readInt(String filename)
throws NumberFormatException,
FileNotFoundException, IOException {
    BufferedReader inputStream = null;
    String in = new String();
    int returnValue;

    try {
        inputStream = new BufferedReader
            (new FileReader(filename));
        in = inputStream.readLine();
        returnValue = Integer.parseInt(in);
    } catch (NumberFormatException e) {
        throw e;
    } catch (FileNotFoundException e) {
        throw e;
    } catch (IOException e) {
        throw e;
    } finally {
        if (inputStream != null) {
            inputStream.close();
        }
    }
    return returnValue;
}
```

Testing readInt

Succeed if returns 1

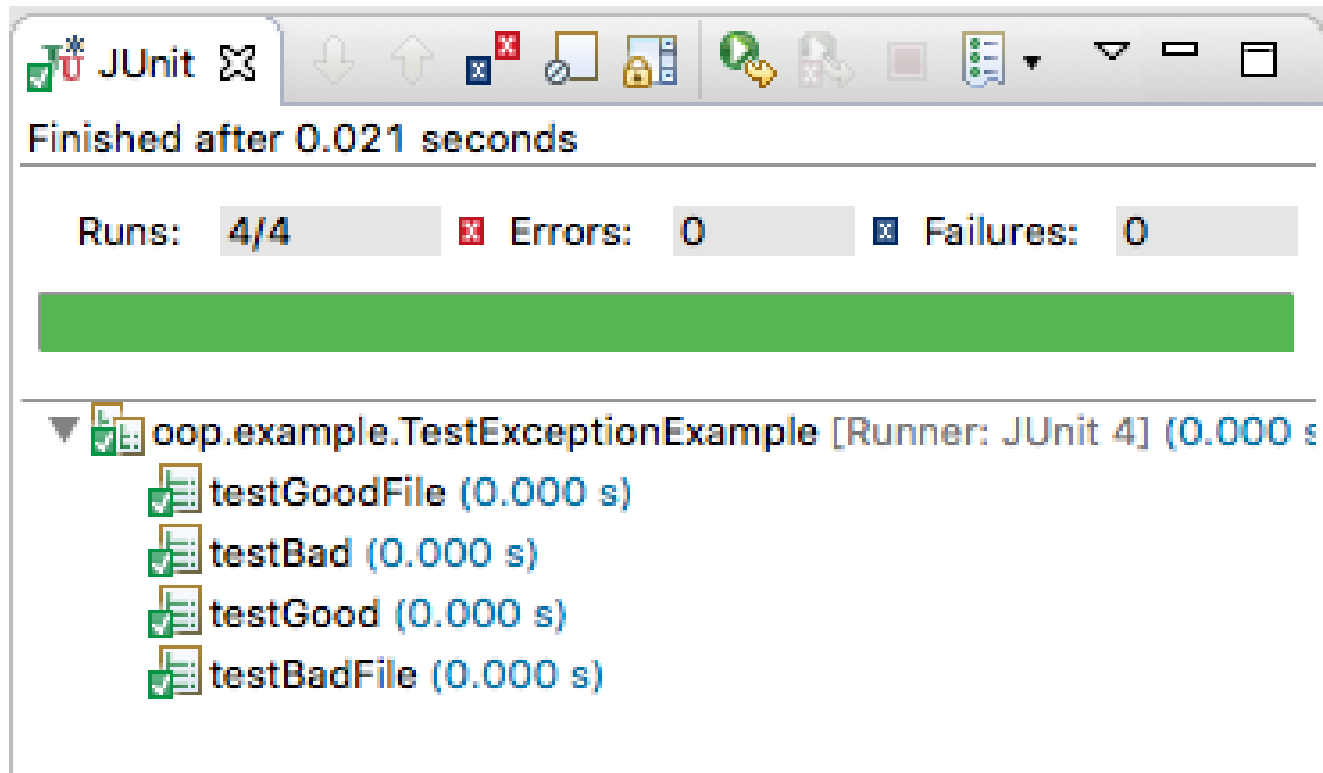
Fail if exceptions thrown

```
@Test
public void testGoodFile() {
    ExceptionExample ex = new ExceptionExample();
    try {
        assertEquals(1, ex.readInt("./good.txt"));
    } catch (FileNotFoundException e) {
        fail("good.txt missing in testGoodFile.");
    } catch (IOException e) {
        fail("good.txt has bad input should be 1");
    }
}
```

Succeed if read "one"

```
@Test
public void testBadFile() {
    ExceptionExample ex = new ExceptionExample();
    try {
        assertEquals(1, ex.readInt("bad.txt"));
    } catch (NumberFormatException e) {
        assertEquals("For input string: \"one\"", e.getMessage());
    } catch (FileNotFoundException e) {
        fail("bad.txt missing testGoodFile.");
    } catch (IOException e) {
        fail("good.txt has bad input should be one");
    }
}
```

Results



The image shows a JUnit test results window. At the top, the title bar says "JUnit" with a close button. Below the title bar, there's a status bar that says "Finished after 0.021 seconds". The main area displays the test results: "Runs: 4/4", "Errors: 0", and "Failures: 0". A green progress bar is shown below the summary. The test suite is "oop.example.TestExceptionExample [Runner: JUnit 4] (0.000 s)". It contains four tests: "testGoodFile (0.000 s)", "testBad (0.000 s)", "testGood (0.000 s)", and "testBadFile (0.000 s)". All tests are marked as passed with green checkmarks.

JUnit

Finished after 0.021 seconds

Runs: 4/4 Errors: 0 Failures: 0

oop.example.TestExceptionExample [Runner: JUnit 4] (0.000 s)

- testGoodFile (0.000 s)
- testBad (0.000 s)
- testGood (0.000 s)
- testBadFile (0.000 s)

Refactoring

- Exceptions form a hierarchy
 - All at Exception
 - All FileNotFoundExceptions are IOExceptions
- A Superclass exception matches its subclasses
- Since we only re-throw the exceptions, we can match them with “catch”

Refactored Example

```
public int readInt(String filename)
throws NumberFormatException,
FileNotFoundException, IOException {
    BufferedReader inputStream = null;
    String in = new String();
    int returnValue;

    try {
        inputStream = new BufferedReader
            (new FileReader(filename));
        in = inputStream.readLine();
        returnValue = Integer.parseInt(in);
    } catch (Exception e) {
        throw e;
    } finally {
        if (inputStream != null) {
            inputStream.close();
        }
    }
    return returnValue;
}
```

Only one “catch”



Refactored Example

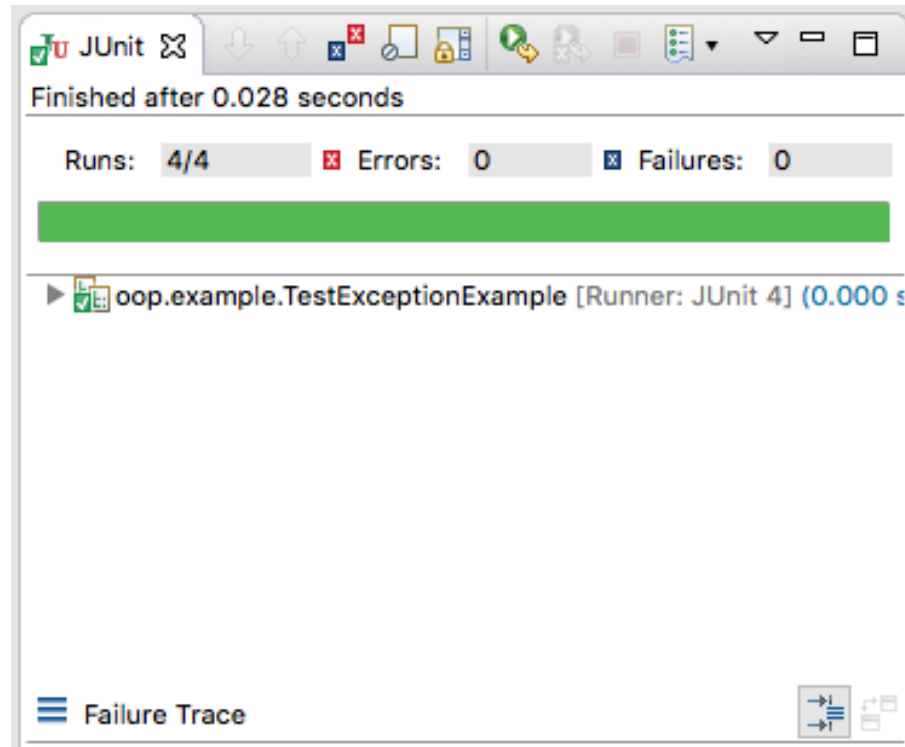
```
public int readInt(String filename)
throws NumberFormatException,
FileNotFoundException, IOException {
    BufferedReader inputStream = null;
    String in = new String();
    int returnValue;

    try {
        inputStream = new BufferedReader
            (new FileReader(filename));
        in = inputStream.readLine();
        returnValue = Integer.parseInt(in);
    } catch (Exception e) {
        throw e;
    } finally {
        if (inputStream != null) {
            inputStream.close();
        }
    }
    return returnValue;
}
```

Only one “catch”



Tests still run



Overriding and Exception

- When overriding a method, it may not add exceptions
- It may throw
 - The same exceptions
 - Subclasses of the exceptions thrown
- It may throw fewer exceptions including none