Object Oriented Programming

Week 7 Part 1
Exceptions

Lecture

- Overview of Exception
- How exceptions solve unexpected occurrences
- Catching exceptions

Exceptions Overview

Unexpected Occurances

- Unexpected things happen
  - Example: “new” fails because out of memory
  - They are difficult to program for
- When unexpected things happen either
  - Halt the program and produce information to debug
  - Alter the course of the program to avoid the problem

Function return

- What does the function do when something unexpected happens.
- You could return something like “null” or “false”, but then the calling function would need to expected these results.
  - Just hands the problem to the calling function

Exceptions are Problems

- It is difficult to get information on exceptions.
  - They cause the program to fail, erasing all evidence
- In C, you are likely to see only “segmentation fault”
- Debugging involves stepping through the program line by line until you find the problem.
  - The IDE makes this easier

Exceptions are worse for users

- Exceptions are bad when writing programs
- They are even worse when they happen to someone who is using the program
- The Java exception mechanism allows you to recover from run-time errors.

Java Exception Mechanism

Java Exception Mechanism

- Java allows a function to “throw” an “exception” or “error” when something unexpected happens
- The throw stops the function and transfers control to the first function in the calling stack that can “catch” the exception or error.
- The function that catches the exception or error can perform recovery.

Exceptions Defined

- A *exception* is an event that occurs during the execution of a program that disrupts the normal flow of instructions.
  - https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html

- Exceptions are also called run-time errors
  - They happen while the program is running
  - Java reduced exceptions by providing ample static checking
    - i.e. Using the compiler to check before it runs

Recoverable and Unrecoverable

- Java distinguishes unrecoverable errors from recoverable errors
  - Unrecoverable errors are problems that should cause the program to halt.
    - E.g., out of memory.
    - Conditions probably indicate the program cannot continue
  - Recoverable errors are problems that the program can handle
    - E.g., file not found
    - The user may have mistyped the name

Exceptions and Errors

- Java defines exceptions to deal with recoverable errors and errors to deal with unrecoverable errors.
- Methods may throw either exceptions or errors.
  - The must declare exceptions they throw, but not errors.
  - Java methods may throw errors
    - e.g., divide by zero.

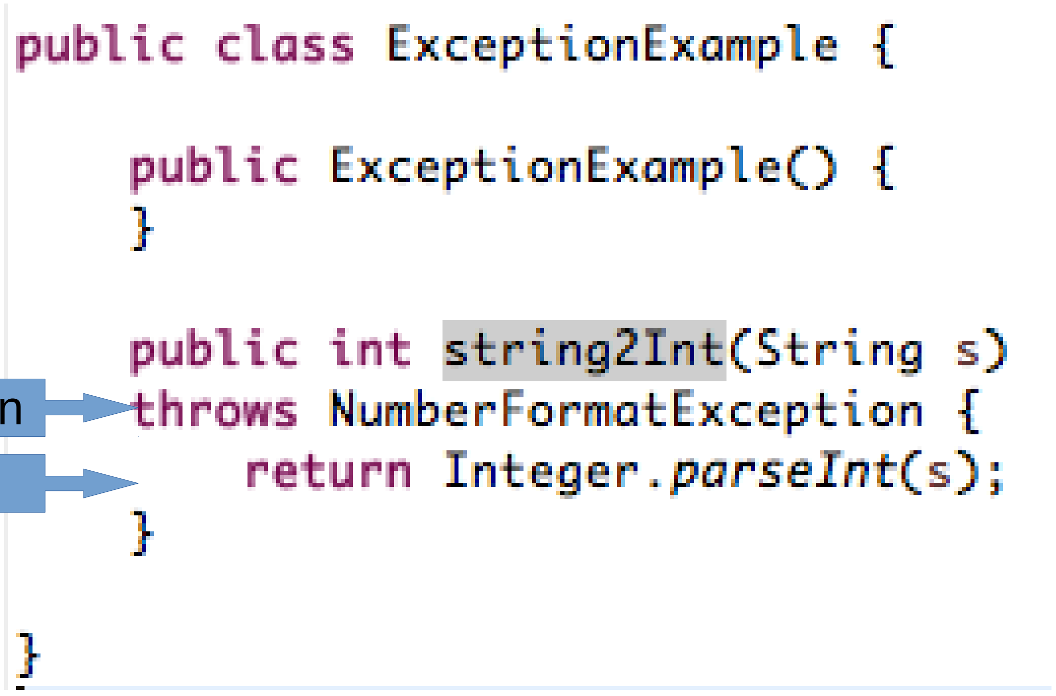
Subclass of Throwable

- Both errors and exceptions may be thrown
  - Throwing an error signals that the program should end
  - Throwing an exception indicates that the program may continue

Methods throw Exceptions

- Build in function throw exceptions
- The thrown exceptions are in turn thrown by the method that calls them
- Methods should declare what exceptions the throw

Example: Throwing an Exception



```
public class ExceptionExample {

    public ExceptionExample() {
    }

    public int string2Int(String s)
    throws NumberFormatException {
        return Integer.parseInt(s);
    }

}
```

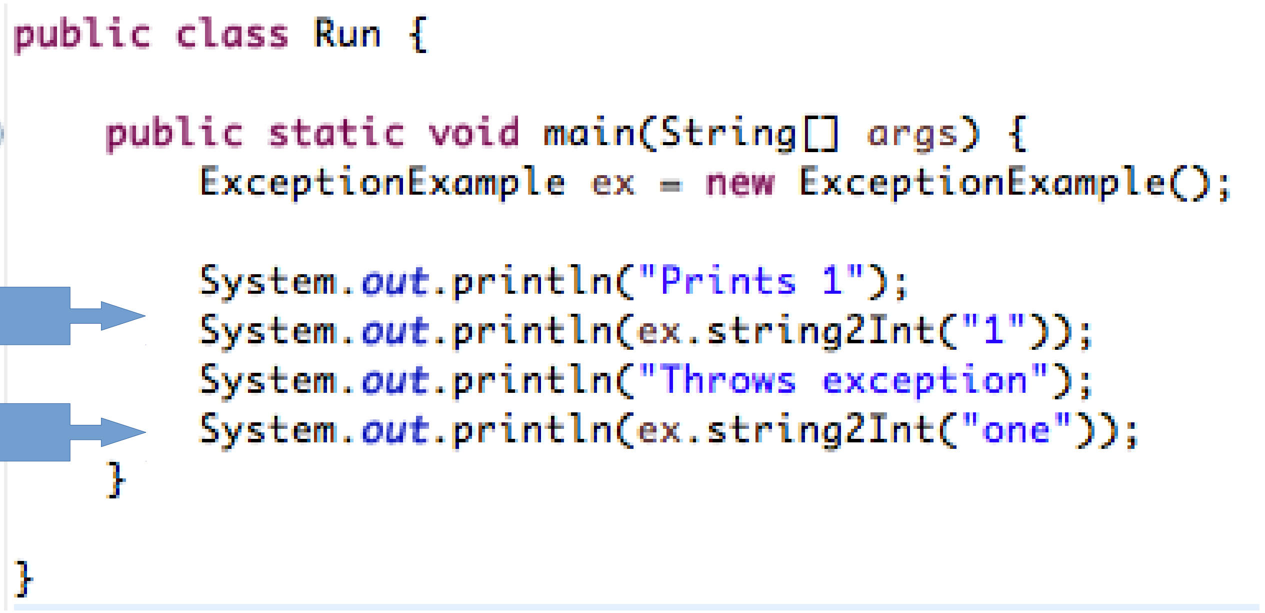
Declares that it throws NumberFormatException

NumberFormatException thrown by parseInt

Example: Running

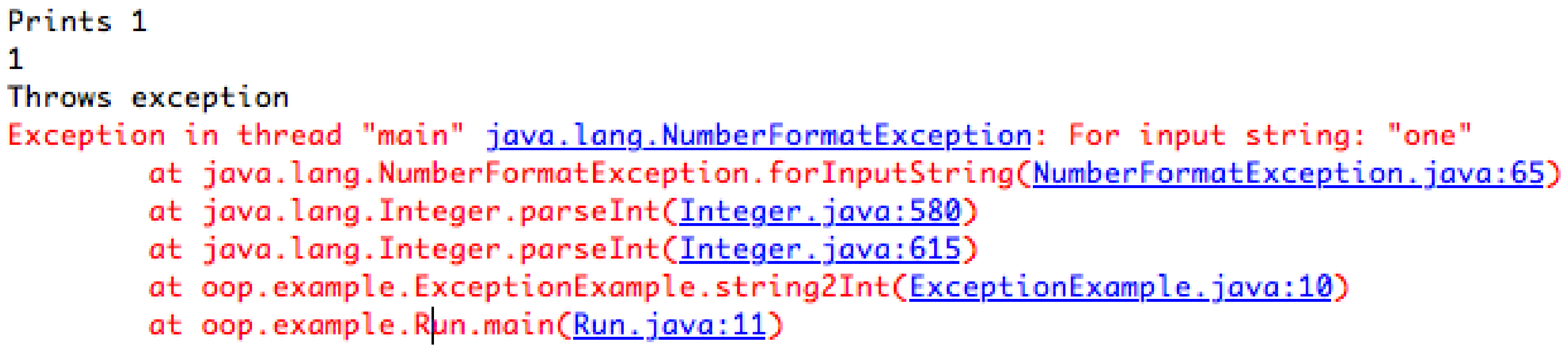
String2Int return 1

String2Int throws exception



```
public class Run {

    public static void main(String[] args) {
        ExceptionExample ex = new ExceptionExample();

        System.out.println("Prints 1");
        System.out.println(ex.string2Int("1"));
        System.out.println("Throws exception");
        System.out.println(ex.string2Int("one"));
    }

}
```

Output

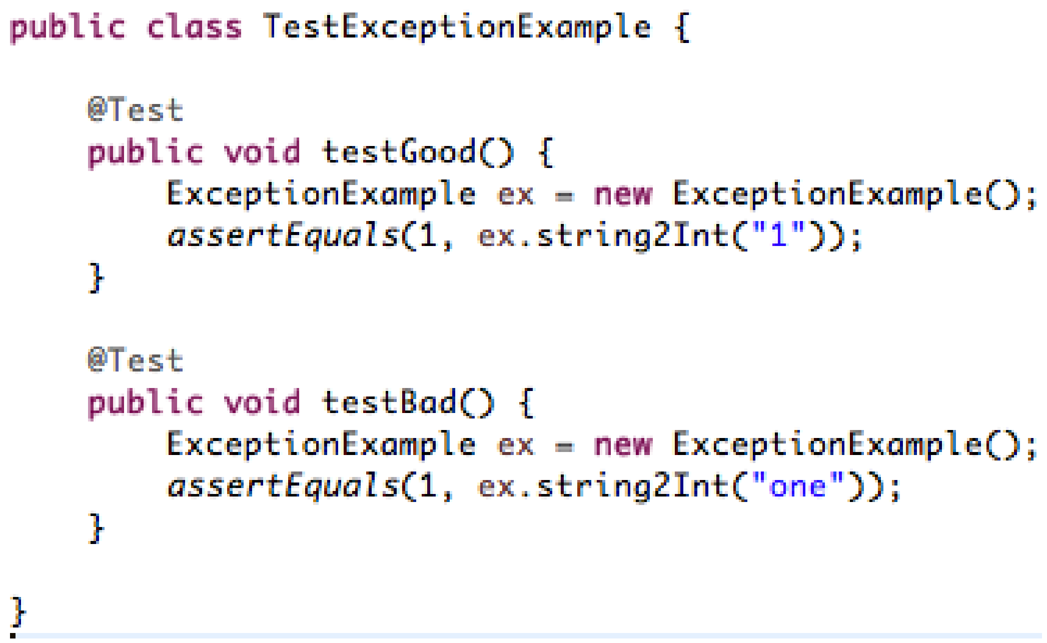


```
Prints 1
1
Throws exception
Exception in thread "main" java.lang.NumberFormatException: For input string: "one"
	at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
	at java.lang.Integer.parseInt(Integer.java:580)
	at java.lang.Integer.parseInt(Integer.java:615)
	at oop.example.ExceptionExample.string2Int(ExceptionExample.java:10)
	at oop.example.Run.main(Run.java:11)
```

Capturing Example in Test Case

Doesn't throw exception

Does throw exception



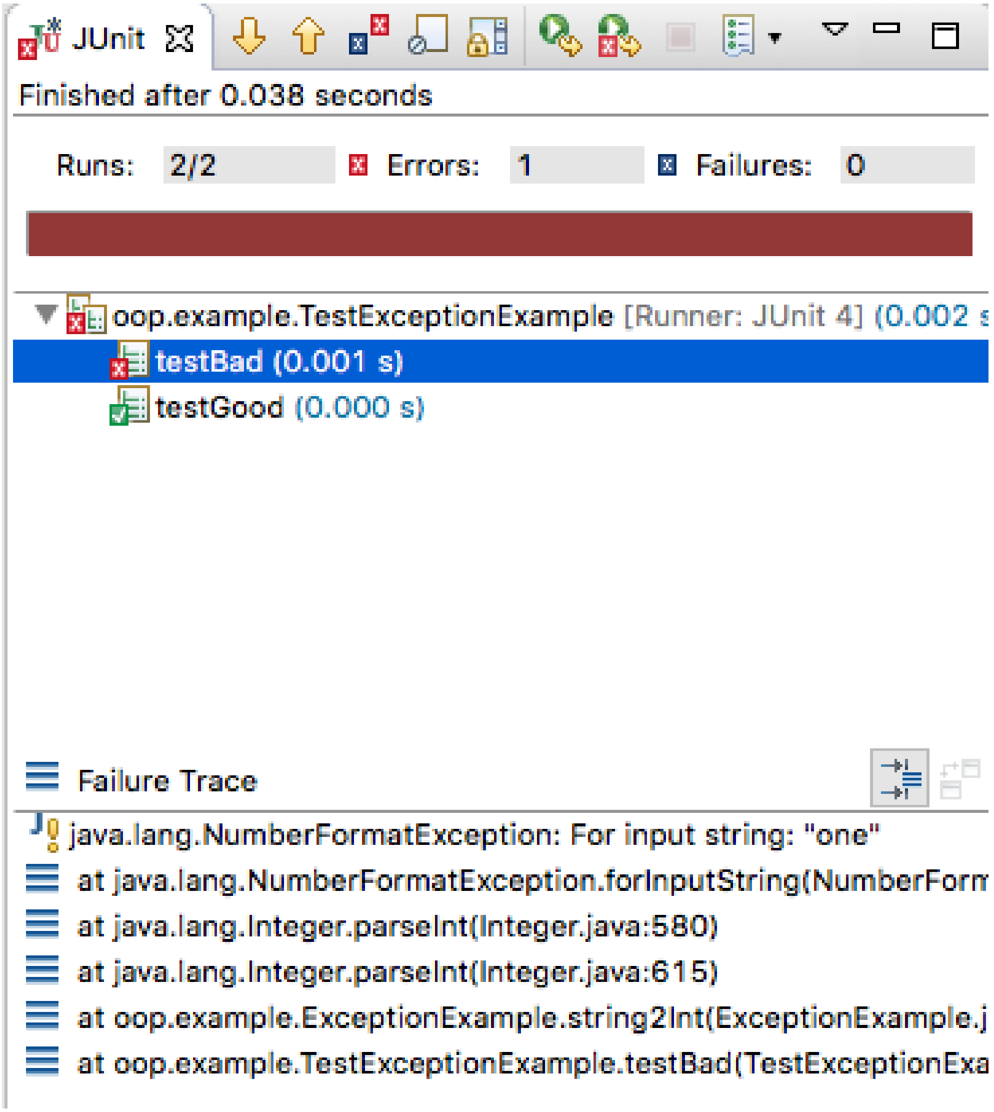
```
public class TestExceptionExample {

    @Test
    public void testGood() {
        ExceptionExample ex = new ExceptionExample();
        assertEquals(1, ex.string2Int("1"));
    }

    @Test
    public void testBad() {
        ExceptionExample ex = new ExceptionExample();
        assertEquals(1, ex.string2Int("one"));
    }

}
```

Results of Running Test Case

Throws exception

Doesn't throw exception

Exception thrown

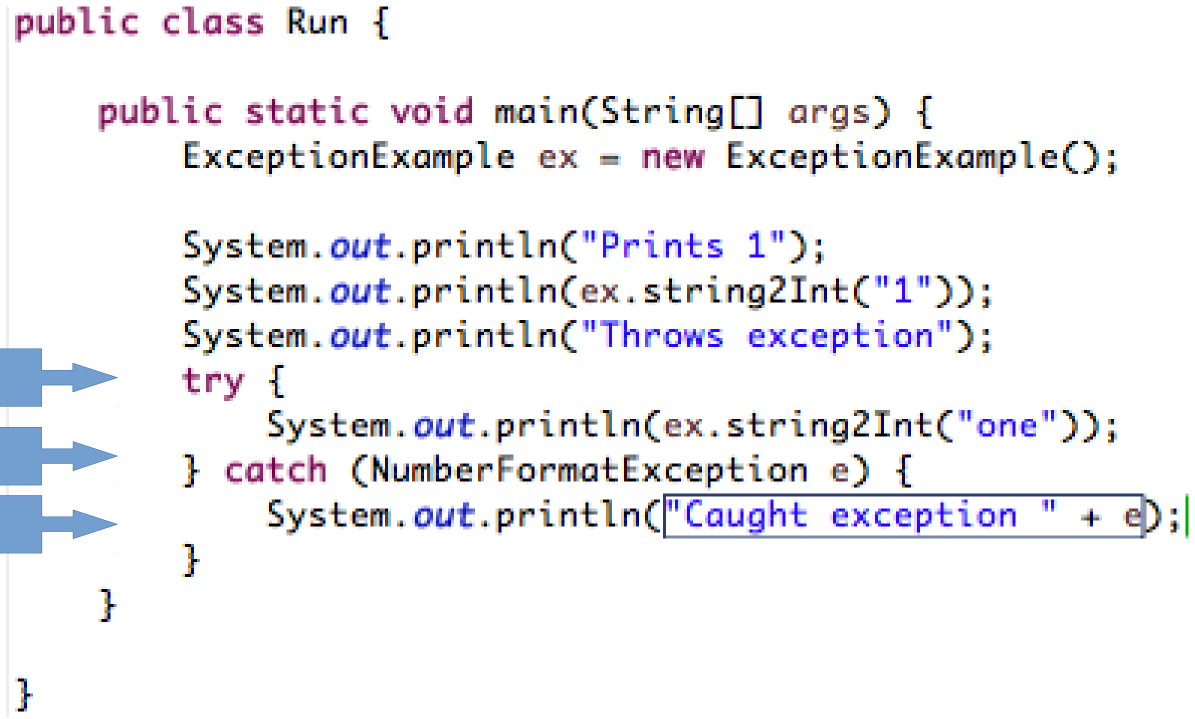


```
Finished after 0.038 seconds

Runs: 2/2    Errors: 1    Failures: 0

oop.example.TestExceptionExample [Runner: JUnit 4] (0.002 s
    testBad (0.001 s)
    testGood (0.000 s)

Failure Trace
java.lang.NumberFormatException: For input string: "one"
at java.lang.NumberFormatException.forInputString(NumberForm
at java.lang.Integer.parseInt(Integer.java:580)
at java.lang.Integer.parseInt(Integer.java:615)
at oop.example.ExceptionExample.string2Int(ExceptionExample.j
at oop.example.TestExceptionExample.testBad(TestExceptionExa
```

Catching a Thrown Object

- A “try” block defines a section of code in which exceptions might be thrown
- A “try” block may be followed by a “catch” statement that defines actions to do if an exception occurs

Example



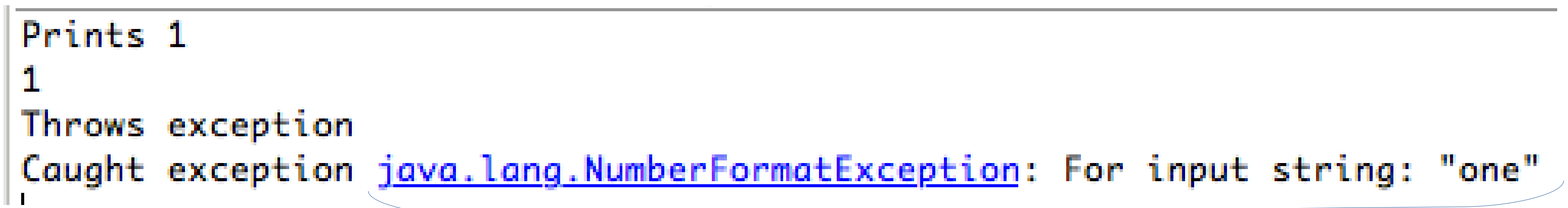
```
public class Run {

    public static void main(String[] args) {
        ExceptionExample ex = new ExceptionExample();

        System.out.println("Prints 1");
        System.out.println(ex.string2Int("1"));
        System.out.println("Throws exception");
        try {
            System.out.println(ex.string2Int("one"));
        } catch (NumberFormatException e) {
            System.out.println("Caught exception " + e);
        }
    }

}
```

Try block surrounds method

Catches NumberFormatException

Prints message if caught

Output



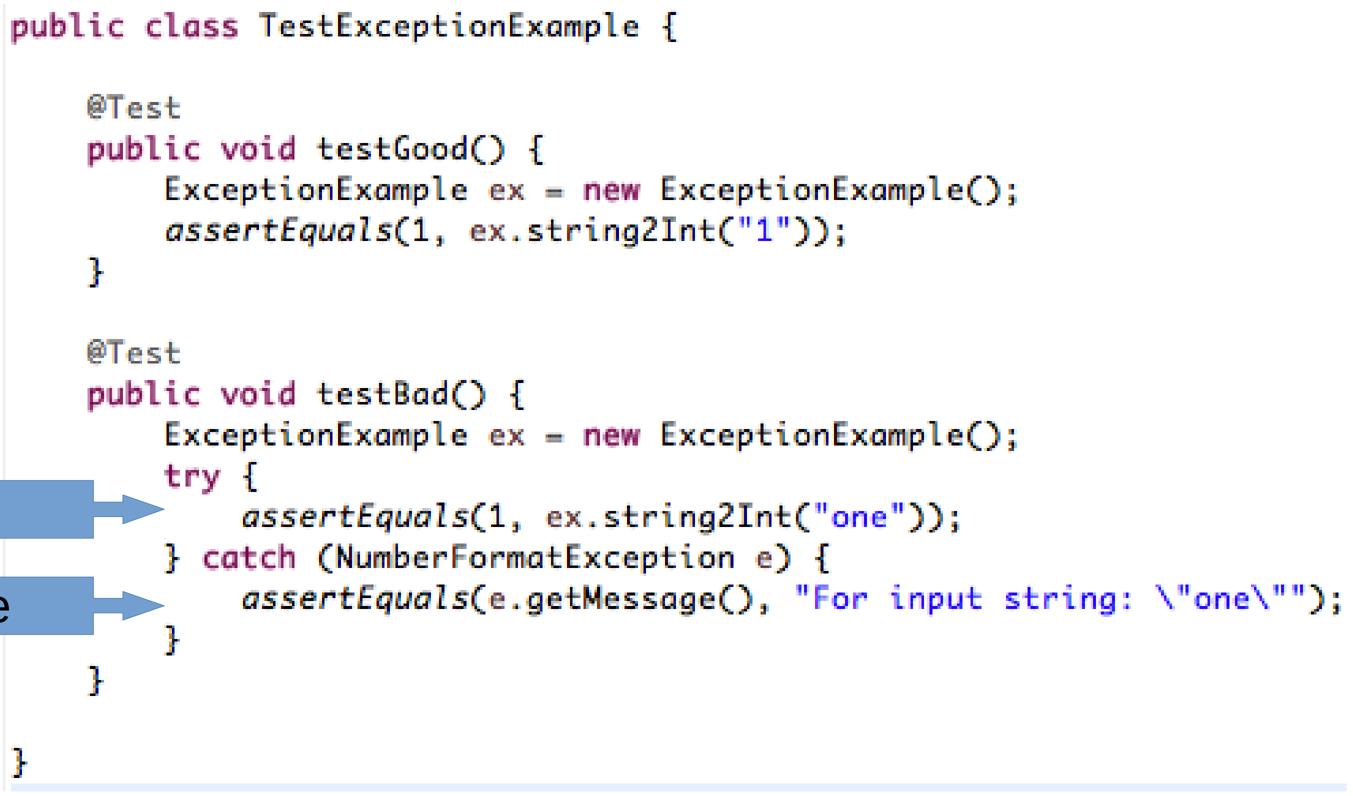
```
Prints 1
1
Throws exception
Caught exception java.lang.NumberFormatException: For input string: "one"
```

e.toString()

JUnit tests functions throw exception

- You can test that a function throws an exception in JUnit
  - Put the expressions that should throw an exception in a “try” bock.
  - Catch the expected exception
  - Test the message the exception carries

JUnit Test Example



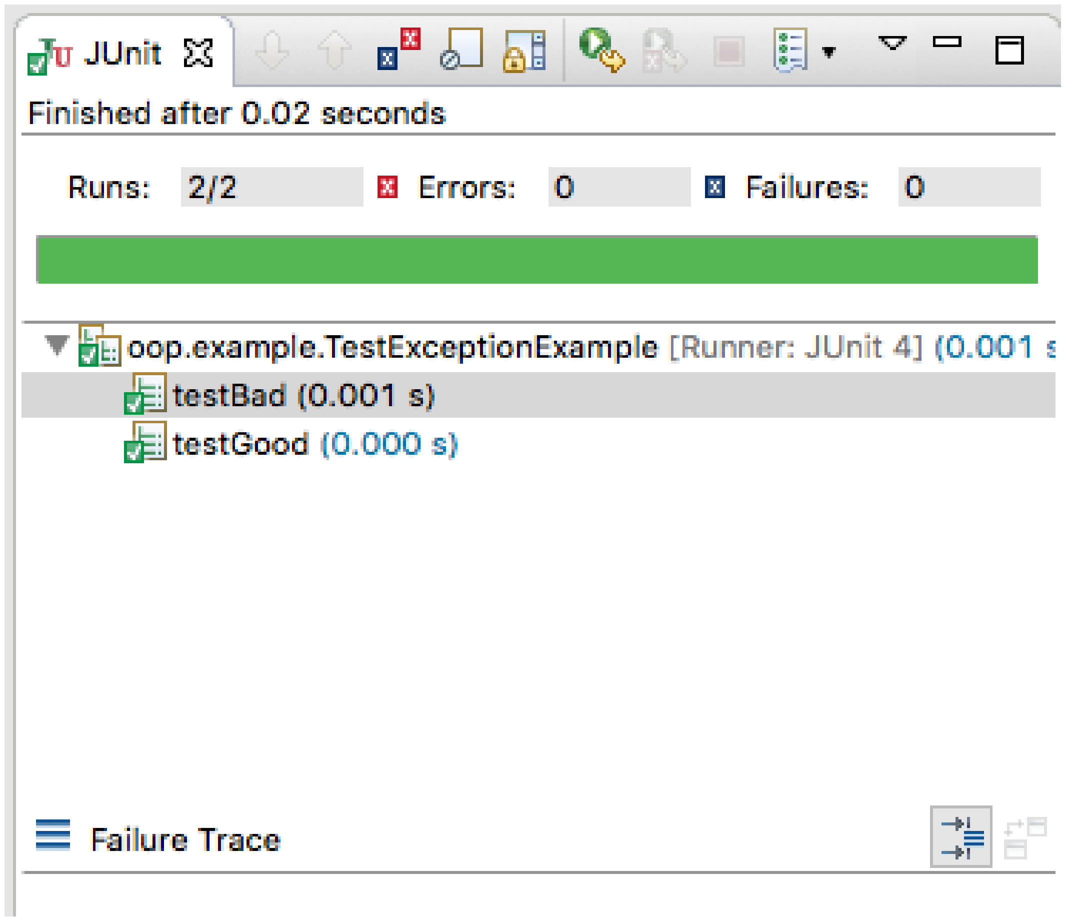
```java
public class TestExceptionExample {

    @Test
    public void testGood() {
        ExceptionExample ex = new ExceptionExample();
        assertEquals(1, ex.string2Int("1"));
    }

    @Test
    public void testBad() {
        ExceptionExample ex = new ExceptionExample();
        try {
            assertEquals(1, ex.string2Int("one"));
        } catch (NumberFormatException e) {
            assertEquals(e.getMessage(), "For input string: \"one\"");
        }
    }

}
```

Try block. AssertEquals skipped

Catch. AssertEquals on message

The first AssertEquals is skipped because string2Int throws an exception halting the “try” block.
The second AssertEquals fails because the “catch” block catches exception and it carries the message 'For input string: “one”'

JUnit Test Output



```
Finished after 0.02 seconds

Runs: 2/2    Errors: 0    Failures: 0

oop.example.TestExceptionExample [Runner: JUnit 4] (0.001 s
    testBad (0.001 s)
    testGood (0.000 s)
```