

# CS409

# Software Testing

**TAN, Shin Hwei**

陈馨慧

Southern University of Science and Technology

Slides adapted from Introduction to Software Testing, Edition 2

# Part 1: Difficult To test objects

---

- System.in
- System.out
- How about checking contents of a file?

# How to test print to console (System.out)?

```
public class HelloWorld {  
    public static void printHello() {  
        System.out.println("Hello, World");  
    }  
}
```

# How to test print to console (System.out)?

```
@Test
public void printTest() {
    //Step1: Prepare to redirect output
    OutputStream os = new ByteArrayOutputStream();
    PrintStream ps = new PrintStream(os);
    System.setOut(ps);

    //Step2: need System.getProperty("line.separator") to
    //properly test for the next line
    HelloWorld.printHello();
    assertEquals("Hello World" +
                 System.getProperty("line.separator"),
                 os.toString());

    //Step3: Restore normal output
    PrintStream originalOut = System.out;
    System.setOut(originalOut);
}
```

# How to test main?

```
class App {
    @Test
    public void mainTest() {
        ...
        public static void main(String[] args) {
            //Step1: Prepare to redirect input
            String[] args = null;
            System.setIn(new ByteArrayInputStream(""));
            Application.main(args);
        }
        //Step2: construct test inputs
        String [] args = { "one", "two", "three" };
        Application.main(args);
    }
    protected void ...
}
```

# How to test file?

```
class Application
{...
public static void main (String[] args)
{
    String string = args[0];
    Application application = new Application();
    try {
        application.run(string);
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
protected void run(String string) throws Exception { }
}
```

# How to test file?

```
class App {
    ...
    public static void main(String[] args) {
        ...
    }
}

@Test
public void writesContentToFile() throws IOException {
    // arrange
    File output =
        temporaryFolder.newFolder("reports").toPath().resolve("output.txt").toFile();

    // act
    fileWriter.writeTo(output.getPath(), "test");

    // assert
    assertThat(output)
        .hasContent("test")
        .hasExtension("txt")
        .hasParent(resolvePath("reports"));
}

private String resolvePath(String folder) {
    return temporaryFolder.getRoot().toPath().resolve(folder).toString();
}
```

# How to test private method?

```
Method method = null;
try {
    method = Anagram.class.getDeclaredMethod("usage", new Class[]{});
} catch (NoSuchMethodException e) {
} catch (SecurityException e) {
}
method.setAccessible(true);
try {
    method.invoke(null);
} catch (IllegalAccessException e) {
} catch (IllegalArgumentException e) {
} catch (InvocationTargetException e) {
}
```

change it to be not private

**Program:**  
private static void usage() {  
  
 ...  
}

- **Use java reflection!**



# How to test private method?

```
Method method = null;
try {
    method = Anagram.class.getDeclaredMethod("usage", new Class[]{});
} catch (NoSuchMethodException e) {
} catch (SecurityException e) {
}
method.setAccessible(true);
try {
    method.invoke(null);
} catch (IllegalAccessException e) {
} catch (IllegalArgumentException e) {
} catch (InvocationTargetException e) {
}
}
```

change it to be not private

**Program:**  
private static void usage() {  
  
...  
}

# Invitation link for part 1

<https://classroom.github.com/a/8TQabGyd>

- Write unit tests for FileWriter to achieve 100% statement and branch coverage

# Shortcut- Template

```
private Method getMethod(Class c, String name) throws  
    Exception{ Method m = c.getDeclaredMethod(name);  
    m.setAccessible(true);  
    return m;  
}
```

```
private <T> void a(T a, T b) { assertEquals(a, b);  
}
```

```
private <T> void at(T a) { assertTrue((Boolean) a);  
}
```

```
private <T> void af(T a) { assertFalse((Boolean) a);  
}
```

```
private <T> void p(T x) { System.out.println(x);  
}
```

# Bonus Opportunity!

- Participate in Software Testing Competition!
- Sign up now open at: <http://www.mooctest.org/>
- You will receive 5 bonus points if you get into the final round (总决赛). Note that to get into the final round, you need to go through several stages:
  - (preliminary round)预选赛时间为10月24-25日
  - (province level)省赛时间为11月8日
  - (final round)总决赛时间为11月22日

Needs to specify the following:

- Email: [tansh3@sustc.edu.cn](mailto:tansh3@sustc.edu.cn)
- Class Number (编号): 753
- Name: 陈馨慧

# Test Driven Development (TDD)

---

# Recap: Extreme Programming (XP)

- Frequent releases
- Elements include:
  - Pair Programming
  - Stories/Use Cases
  - Continuous Integration
  - *Test Driven Development*

# History of Test Driven Development(TDD)

- Kent Beck
  - Software Engineers
  - Creator of XP
  - “rediscover” TDD



## Interesting information

码农Coding Peasant?

- Kent Beck was also a goat farmer!

# Kent Beck's rules

- Beck's concept of test-driven development centers on two basic rules:
  - Never write a single line of code unless you have a failing automated test.
  - Eliminate duplication.



# Informal Requirements

**Maintenance:** The Maintenance function records the history of items undergoing maintenance.

If the product is covered by warranty or maintenance contract, maintenance can be requested either by calling the maintenance toll free number, or through the web site, or by bringing the item to a designated maintenance station.

If the maintenance is requested by phone or web site and the customer is a US or EU resident, the item is picked up at the customer site, otherwise, the customer shall ship the item with an express courier.

If the maintenance contract number provided by the customer is not valid, the item follows the procedure for items not covered by warranty.

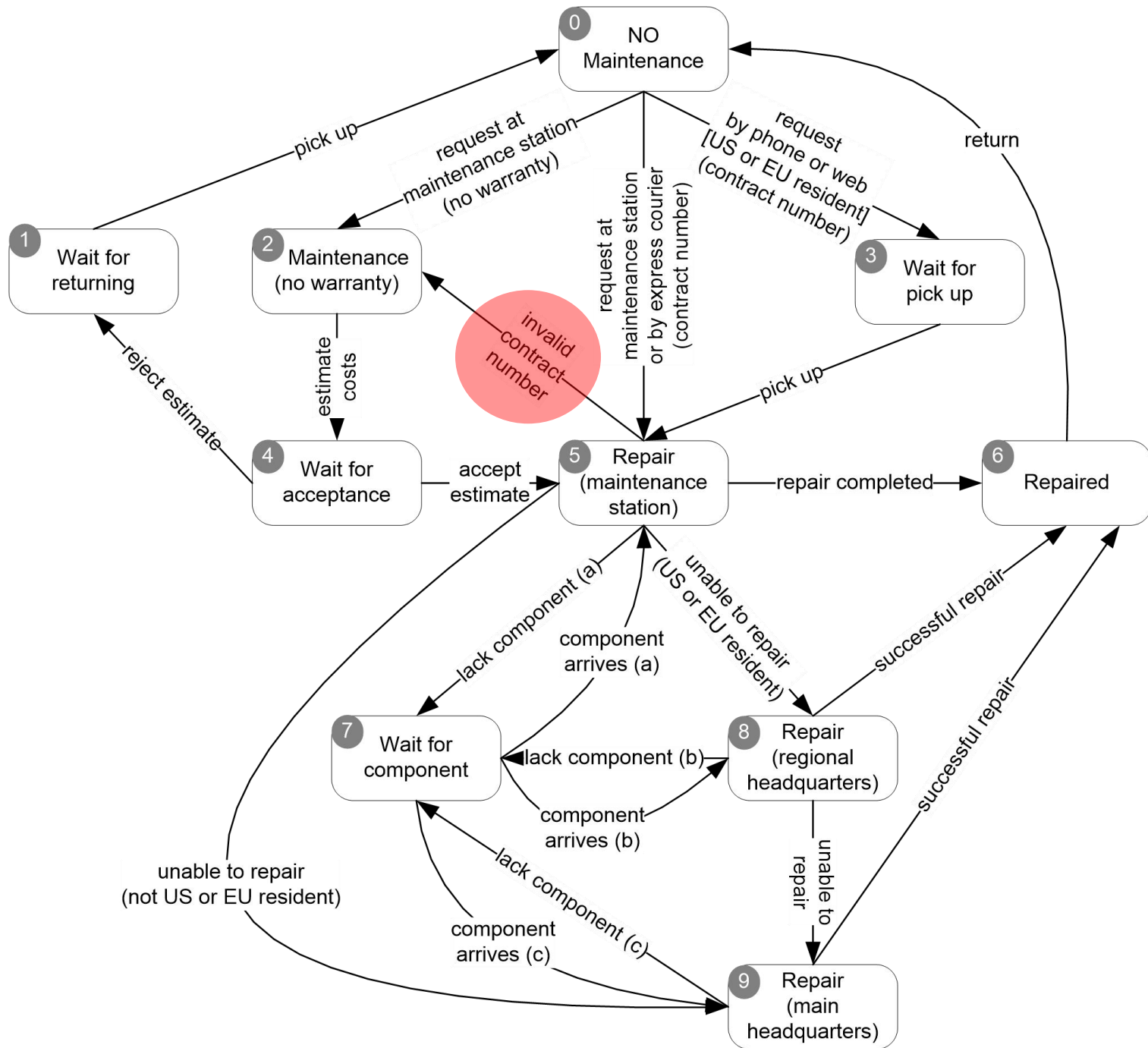
If the product is not covered by warranty or maintenance contract, maintenance can be requested

**If the maintenance contract number provided by the customer is not valid, the item follows the procedure for items not covered by warranty.**

maintenance main headquarters.

Maintenance is suspended if some components are not available.

Once repaired, the product is returned to the customer.



# Ambiguity in Informal Requirements

If the maintenance contract number provided by the customer is not valid

- Contract number cannot contain alphabets or special characters?
- Contract number must be 5 digits?
- Contract number cannot start with 0?

# Requirements based on Test Cases

*@Test*

```
public void testContractNumberCorrectLength() {  
    assertTrue(contract.isValidContractNumber("12345"));  
}
```

*@Test*

```
public void testContractNumberTooLong() {  
    assertFalse(contract.isValidContractNumber("53434434343"));  
}
```

*@Test*

```
public void testContractNumberNoSpecialCharacter() {  
    assertTrue(contract.isValidContractNumber("08067"));  
}
```

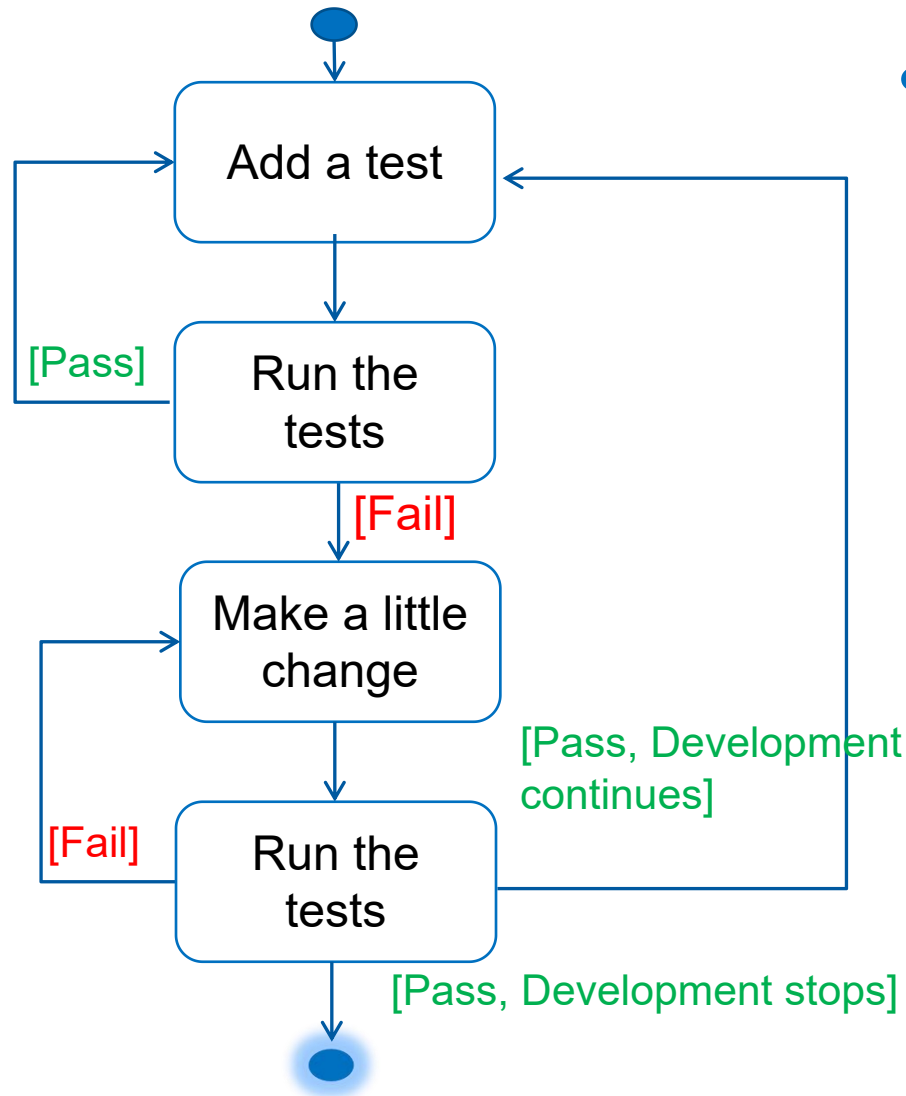
*@Test*

```
public void testContractNumberWithSpecialCharacter() {  
    assertFalse(contract.isValidContractNumber("98&67"));  
}
```

# Informal Requirements versus Test cases

- Test cases are more specific than requirements.
- But: How to develop code based on test cases?
  - Follow the steps in Test Driven Development(TDD)

# Steps in Test Driven Development (TDD)



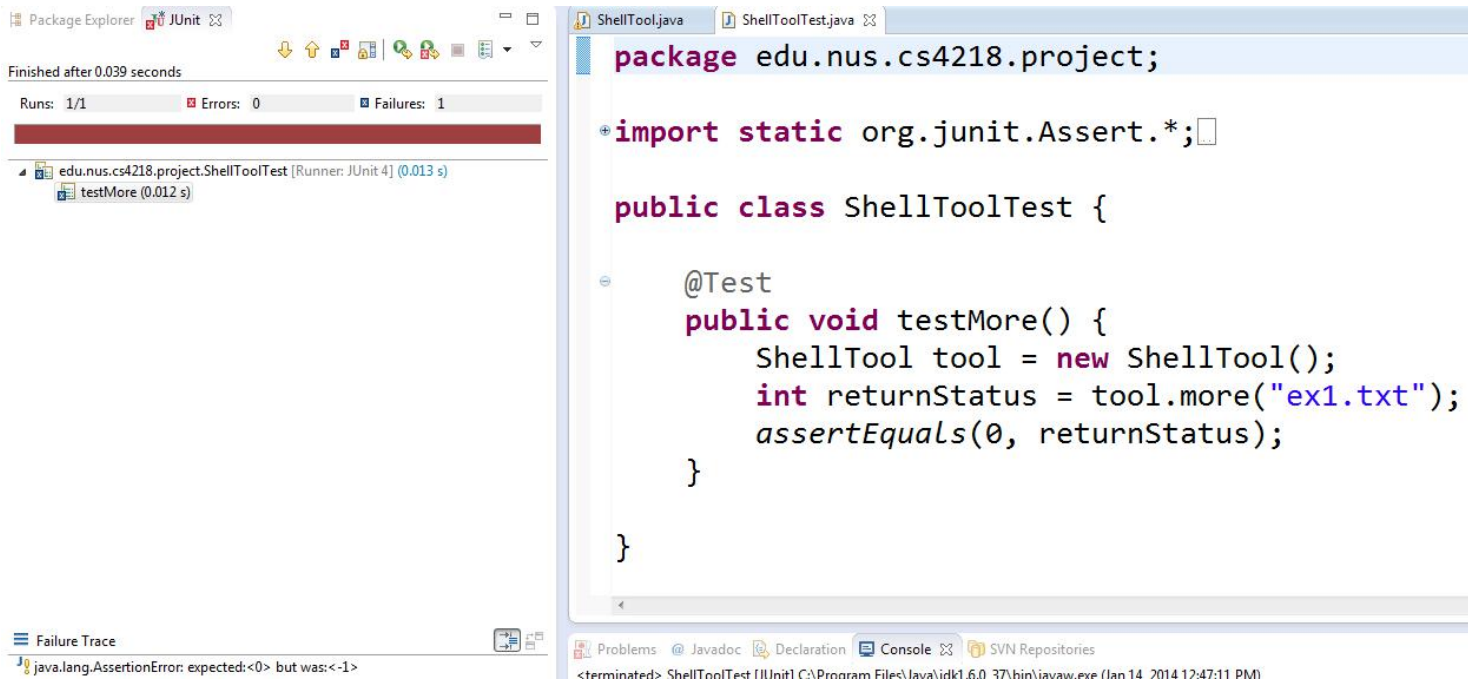
- The iterative process
  - Quickly add a test.
  - Run all tests and see the new one fail.
  - Make a little change to code.
  - Run all tests and see them all succeed.
  - Refactor to remove duplication.

# Test First in Your Assignment

- Write test for the newly added functionality
  - These test cases will serve as a specification for your implementation
  - These test cases should fail now because the corresponding methods are not implemented
  - Write minimal code to make the test pass
  - Add more tests

# Run the tests

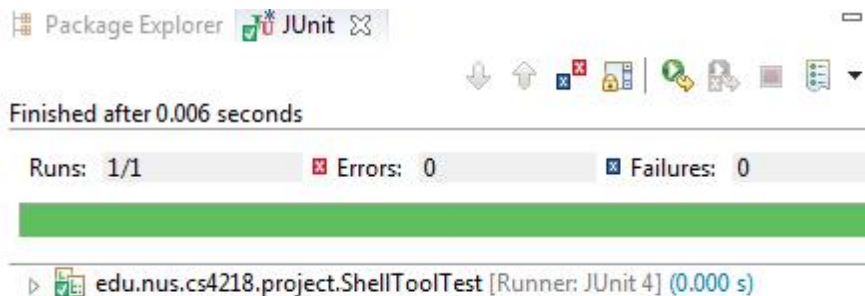
- Run the tests that your team gets to see the **failing** ones
  - Failing test cases indicates missing functionality





# Make them Pass

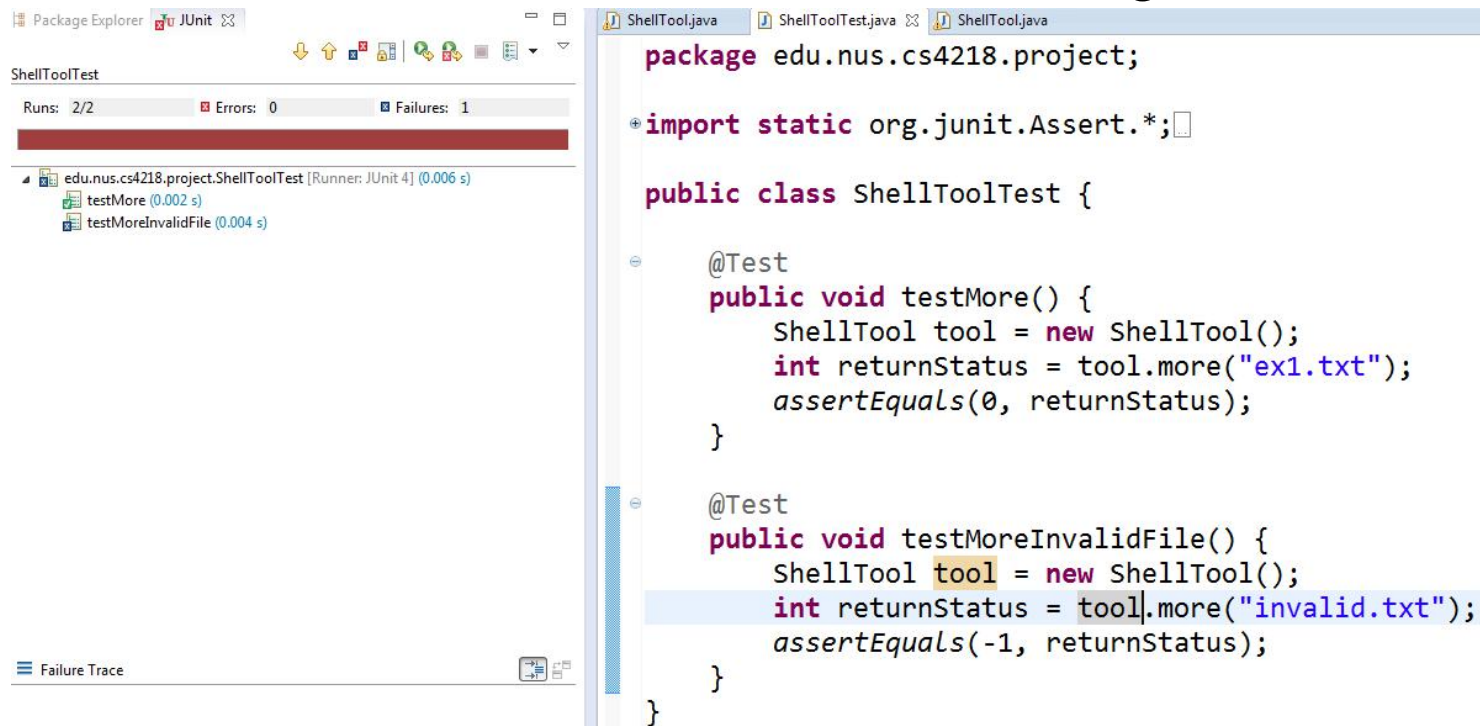
- Add code to make the failing tests **pass**
  - After implementing all the missing functionalities, all failing test cases should now pass



```
/**
 * Shows the first part of a file
 *
 * @param path
 * @return
 */
public int more(String path) {
    BufferedReader in = null;
    try {
        in = new BufferedReader(new FileReader(path));
        String line;
        try {
            while ((line = in.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    return 0;
}
```

# Add more tests

- Add more tests for
  - Newly added components or helper methods
  - Checking for corner cases
- Run the new set of tests to see the failing ones



The screenshot shows an IDE with two main panels. The left panel displays the 'JUnit' test runner for 'edu.nus.cs4218.project.ShellToolTest'. It shows 'Runs: 2/2', 'Errors: 0', and 'Failures: 1'. A red progress bar indicates the test results. Below this, a tree view shows the test methods: 'testMore (0.002 s)' and 'testMoreInvalidFile (0.004 s)'. The right panel shows the source code for 'ShellToolTest.java'. The code is as follows:

```
package edu.nus.cs4218.project;

import static org.junit.Assert.*;

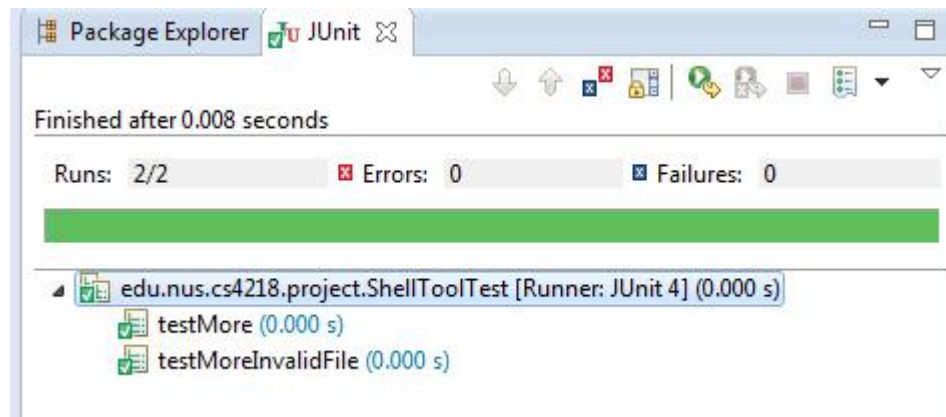
public class ShellToolTest {

    @Test
    public void testMore() {
        ShellTool tool = new ShellTool();
        int returnStatus = tool.more("ex1.txt");
        assertEquals(0, returnStatus);
    }

    @Test
    public void testMoreInvalidFile() {
        ShellTool tool = new ShellTool();
        int returnStatus = tool.more("invalid.txt");
        assertEquals(-1, returnStatus);
    }
}
```

# Make them Pass

- Add more code to make the added tests pass
  - After implementing all the helper methods and checking for corner cases, all new failing test cases should now pass



# Interview question

---

Can you pass the “Fizzbuzz” test?

# TDD Exercise

- tdd-lab in GitHub classroom
- <https://classroom.github.com/g/Rf2Mkwo7>

## Interview question

- Write a Java program, which checks if a given String is palindrome or not.
- Hint: A palindrome (回文(正反读都一样的词语)) is a word, phrase, or sequence that reads the same backwards as forwards

# TDD in a team

- Form a team of two/three students
  - Student A write a test to specify the requirement
  - Student A commits the test
  - Student B write minimal code for making the test pass
  - Student B commits the code
  - Student A write one more test
  - Student A commits the test
  - Student B write minimal code for making the new test pass
  - Student B commits the code
  - Repeat

Remember to commit at each step!

# Example Test Cases

- `testEmptyString()`
- `testIsPalindrome()`
- `testIsNotPalindrome()`