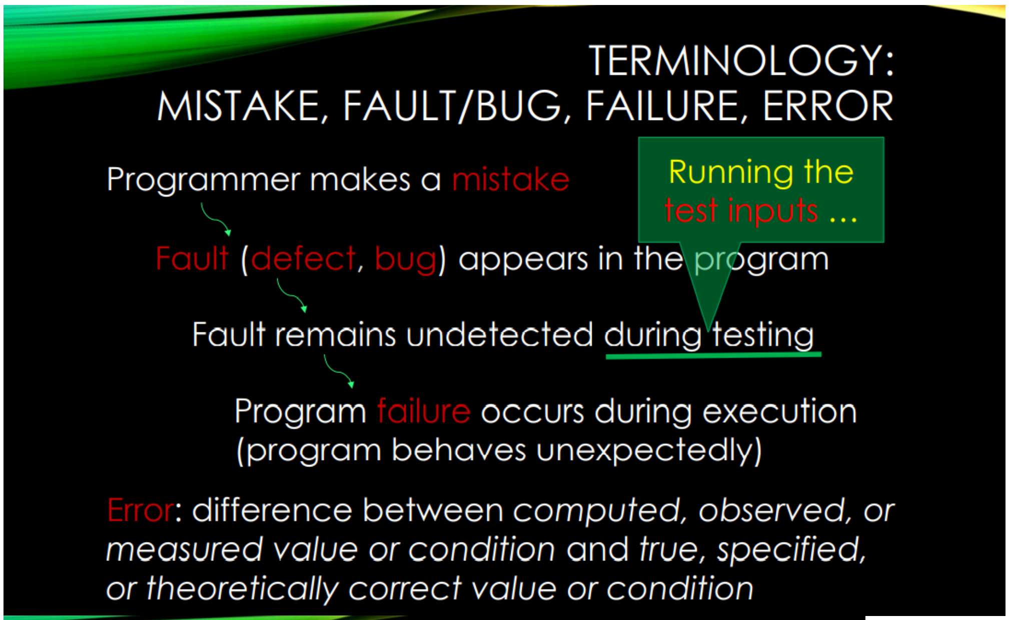
Lec4

WHY TEST?

- Improve quality - find bugs (有BUG的后果很严重，交易所，飞机，汽车，buang~)
- Measure quality
  - Prove there are no bugs? (Is it possible?)
  - Determine if software is ready to be released
  - Determine what to work on
  - See if you made a mistake
- Learn the software

WHAT IS A TEST?

- Run program with known inputs (test inputs/data), check results (with test oracles)
  - Tests pass (green) or fail (red)
- Tests can document faults
- Tests can document code
- Important terminology to remember:
  - **Mistake, fault (or defect, or bug), failure, error**
  - Oracle



TERMINOLOGY: MISTAKE, FAULT/BUG, FAILURE, ERROR

Running the test inputs …

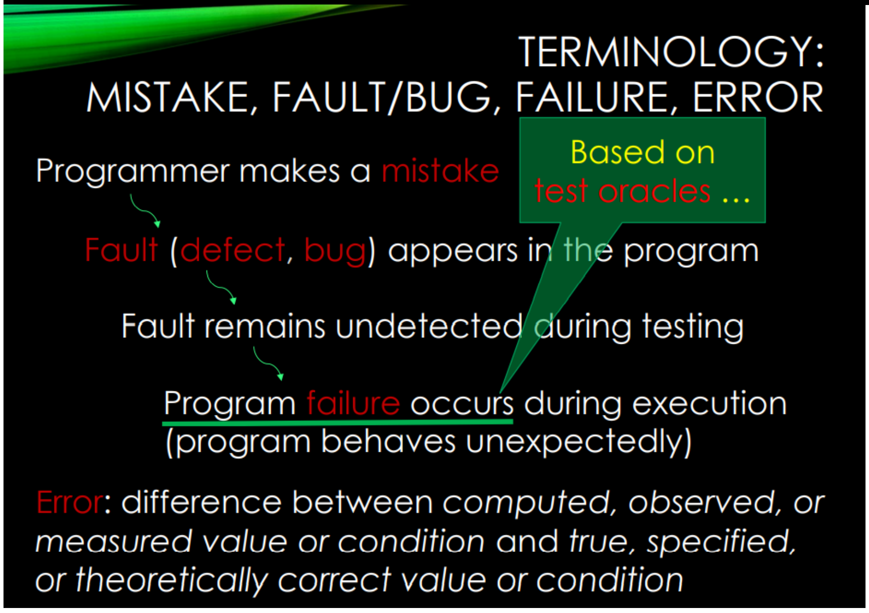
Programmer makes a mistake

Fault (defect, bug) appears in the program

Fault remains undetected during testing

Program failure occurs during execution (program behaves unexpectedly)

Error: difference between *computed, observed, or measured value or condition* and *true, specified, or theoretically correct value or condition*



TERMINOLOGY: MISTAKE, FAULT/BUG, FAILURE, ERROR

Based on test oracles …

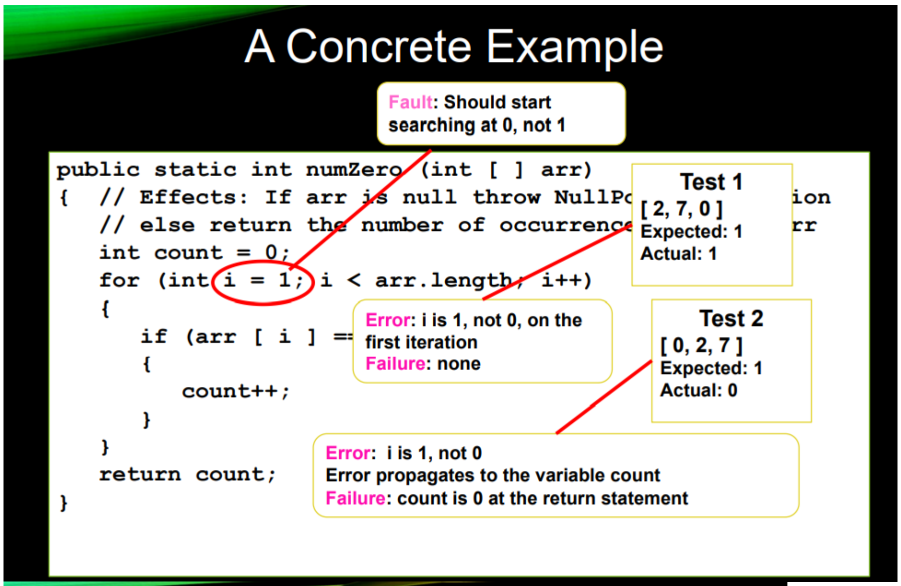
Programmer makes a mistake

Fault (defect, bug) appears in the program

Fault remains undetected during testing

Program failure occurs during execution (program behaves unexpectedly)

Error: difference between *computed, observed, or measured value or condition* and *true, specified, or theoretically correct value or condition*



A Concrete Example

Fault: Should start searching at 0, not 1

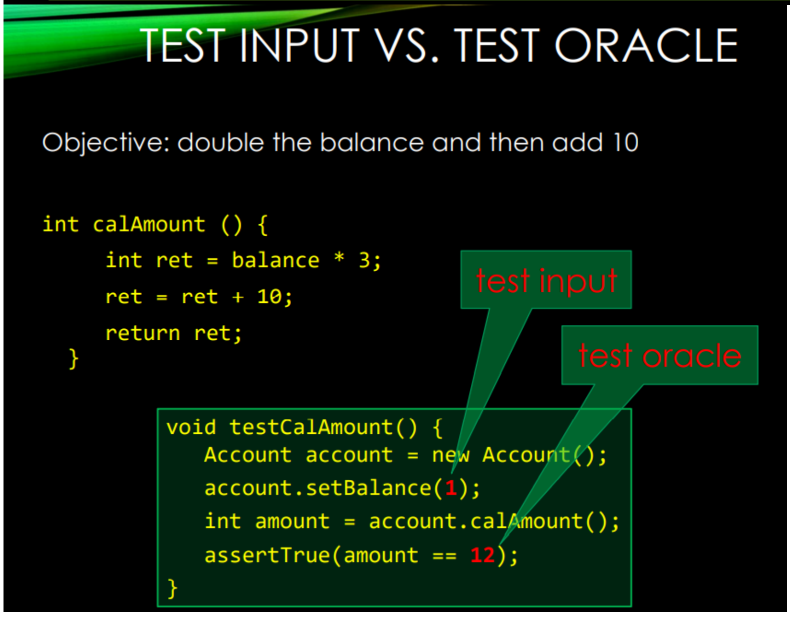
```
public static int numZero (int [ ] arr)
{  // Effects: If arr is null throw NullPo        ion
   // else return the number of occurrence        rr
   int count = 0;
   for (int i = 1; i < arr.length; i++)
   {
      if (arr [ i ] ==
      {
         count++;
      }
   }
   return count;
}
```

Test 1
[2, 7, 0]
Expected: 1
Actual: 1

Error: i is 1, not 0, on the first iteration
Failure: none

Test 2
[0, 2, 7]
Expected: 1
Actual: 0

Error: i is 1, not 0
Error propagates to the variable count
Failure: count is 0 at the return statement



TEST INPUT VS. TEST ORACLE

Objective: double the balance and then add 10

```
int calAmount () {
     int ret = balance * 3;
     ret = ret + 10;
     return ret;
  }
```

```
void testCalAmount() {
   Account account = new Account();
   account.setBalance(1);
   int amount = account.calAmount();
   assertTrue(amount == 12);
}
```

test input

test oracle

1个测试用例 = 1个测试输入（test input） + 1个test oracle。test input（测试输入）是用来执行程序的，测试oracle是用来检查测试执行的正确性的。且test oracle通常是以可执行的assertions语句（比如在JUnit test 框架中）的形式出现的。

JUNIT BASIC

- Open source (junit.org) Java testing framework used to write and run repeatable **automated tests**
- A structure for writing test drivers
- JUnit features include:
  - **Assertions** for testing expected results
  - **Sharing common test data** among tests
  - **Test suites** for easily organizing and running tests
  - **Test runners**, both graphical and textual
- JUnit is widely used in industry
- Can be used as stand alone Java programs (from command line) or from an IDE such as IntelliJ or Eclipse

JUNIT TESTS

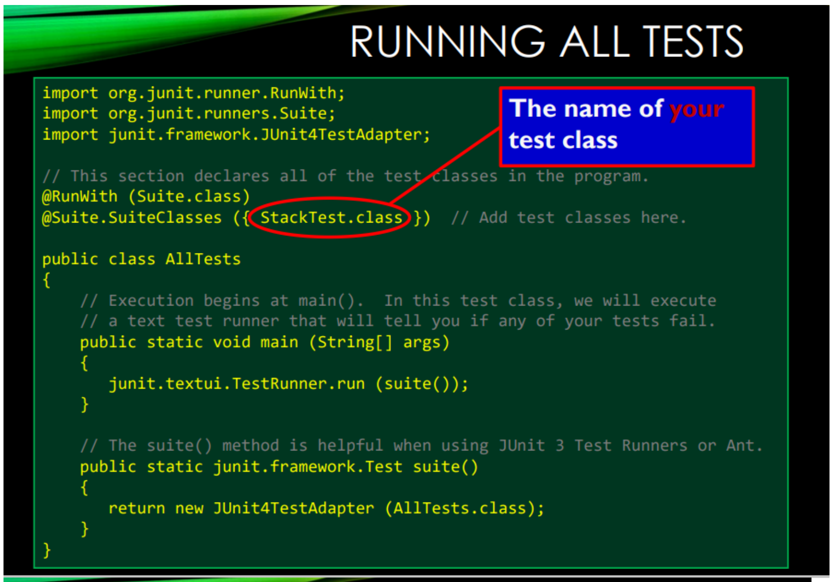
- JUnit can be used to test ...
    - ... an entire object
    - ... part of an object – method or interacting methods
    - ... interaction between several objects
- Primarily **unit & integration testing**, **not system testing**
- Each test is embedded into one test method
- A test class contains one or more test method
- Test classes include:
    - A test runner to run the tests - main()
    - A collection of test methods
    - Methods to set up the state before and update the state after each test and before and after all tests

WRITING TESTS FOR JUNIT

- Need to use methods of `junit.framework.assert` class
- Each test method checks a condition (assertion) and reports to the test runner whether the test succeeded or failed
- The test runner uses the result to report to the user (in command line mode) or update the display (in an IDE)
- All of the methods **return void**
- A few representative methods (of `junit.framework.assert`):
    - assertTrue([String message], boolean condition)
    - assertEquals([String message], Object expected, Object actual)
    - assertNull([String message], Object)
    - Fail(String)

JUNIT TEST FIXTURES

- A **test fixture** is the **state** of the test
    - Objects and variables used by more than one test
    - Initializations (prefix values)
    - Reset values (postfix values)
- Different tests can use objects without sharing state
- Objects in fixtures declared as instance variables
- They should be initialized in a `@Before` method
    - JUnit runs them *before* every `@Test` method
- Can be deallocated or reset in an `@After` method
    - JUnit runs them *after* every `@Test` method



RUNNING ALL TESTS

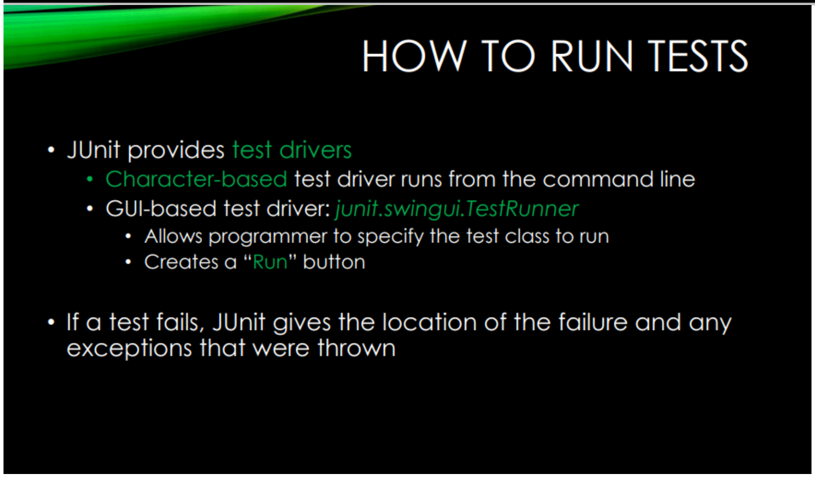
```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import junit.framework.JUnit4TestAdapter;

// This section declares all of the test classes in the program.
@RunWith (Suite.class)
@Suite.SuiteClasses ({ StackTest.class }) // Add test classes here.

public class AllTests
{
    // Execution begins at main(). In this test class, we will execute
    // a text test runner that will tell you if any of your tests fail.
    public static void main (String[] args)
    {
        junit.textui.TestRunner.run (suite());
    }

    // The suite() method is helpful when using JUnit 3 Test Runners or Ant.
    public static junit.framework.Test suite()
    {
        return new JUnit4TestAdapter (AllTests.class);
    }
}
```

The name of your test class



HOW TO RUN TESTS

- JUnit provides test drivers
  - Character-based test driver runs from the command line
  - GUI-based test driver: *junit.swingui.TestRunner*
    - Allows programmer to specify the test class to run
    - Creates a "Run" button
- If a test fails, JUnit gives the location of the failure and any exceptions that were thrown

JUNIT高级主题 ↓

ASSERTION PATTERN

How to decide if your test passes?

- State Testing Patterns
  - Final State Assertion (Most Common Pattern: Arrange-Act-Assert.)

    Assumptions (Preconditions) Limit Values Appropriately

    Action Performs Activity Under Scrutiny

    Assertions (Postconditions) Check Result
  - Guard Assertion (Assert Both Before and After The Action (Precondition Testing))
  - Delta Assertion (Verify a Relative Change to the State)
  - Custom Assertion (Encodes Complex Verification Rules)
- Interaction Assertion
  - Verify Expected Interactions
  - Heavily used in Mocking tools

PARAMETERIZED TESTS

How to describe and run very similar tests?

- Parameterized unit tests call constructor for each logical set of data value
    - Same tests are then run on each set of data values
    - List of data values identified with `@Parameters` annotation

```
import org.junit.*;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;
import static org.junit.Assert.*;
import java.util.*;

@RunWith(Parameterized.class)
public class ParamTest {
    public int sum, a, b;
    public ParamTest (int sum, int a, int b) {
        this.sum = sum; this.a = a; this.b = b;
    }
    @Parameters public static Collection<Object[]> parameters() {
        return Arrays.asList (new Object [][] {{0, 0, 0}, {2, 1, 1}});
    }
    @Test public void additionTest() { assertEquals(sum, a+b); }
}
```

有参数列表的测试

```
import org.junit.*;
import org.junit.runner.RunWith;
import static org.junit.Assert.*;
import static org.junit.Assume.*;
import org.junit.experimental.theories.DataPoint;
import org.junit.experimental.theories.DataPoints;
import org.junit.experimental.theories.Theories;
import org.junit.experimental.theories.Theory;
import java.util.*

@RunWith(Theories.class)
public class SetTheoryTest {
    @Theory public void removeThenAddDoesNotChangeSet(
    Set<String> set, String string) { // Parameters!
            assumeTrue(set.contains(string)) ; // Assume
        Set<String> copy = new HashSet<String>(set); // Act
        copy.remove(string);
        copy.add(string);
        assertTrue (set.equals(copy)); // Assert
    // System.out.println(“Instantiated test: " + set + "," + string);
    }

    // 参数怎么来?
    // All combinations of values from @DataPoint(format is an array) annotations
where assume clause is true
```

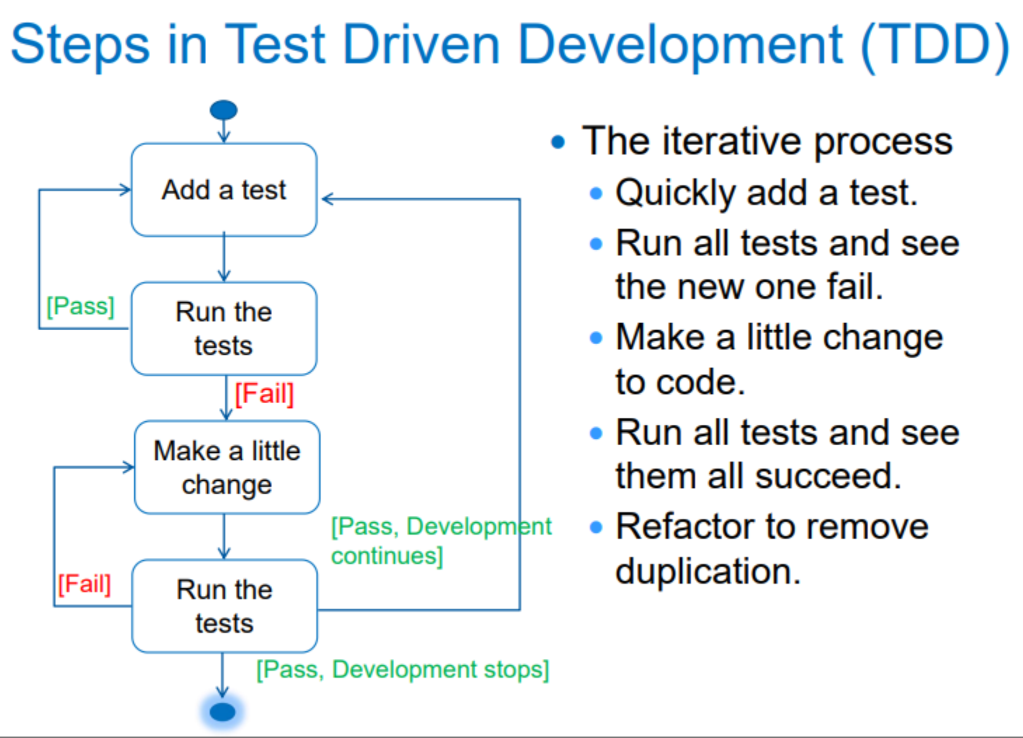
```java
    // Four (of nine) combinations in this particular case
    @DataPoints
    public static String[] string = {"ant", "bat", "cat"};
    @DataPoints
    public static Set[] sets = {
        new HashSet(Arrays.asList("ant", "bat")),
        new HashSet(Arrays.asList("bat", "cat", "dog", "elk")),
        new HashSet(Arrays.asList("Snap", "Crackle", "Pop"))
    };
}
```

Test Driven Development (TDD)

One of the practices in XP

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.



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.