JUnit

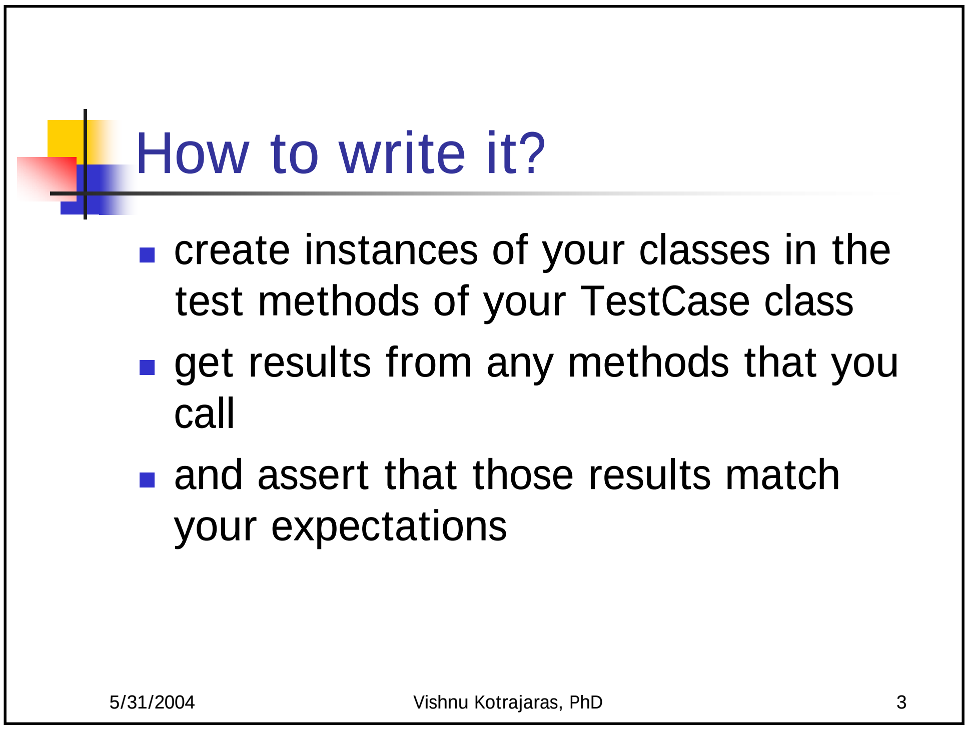
Vishnu

5/31/2004 Vishnu Kotrajaras, PhD 1

Introduction

- unit tests = public classes that extend the junit.framework.TestCase
- methods with names beginning with the word "test"
- provides methods to easily assert things about your own classes
- as well as the ability to run a group of tests

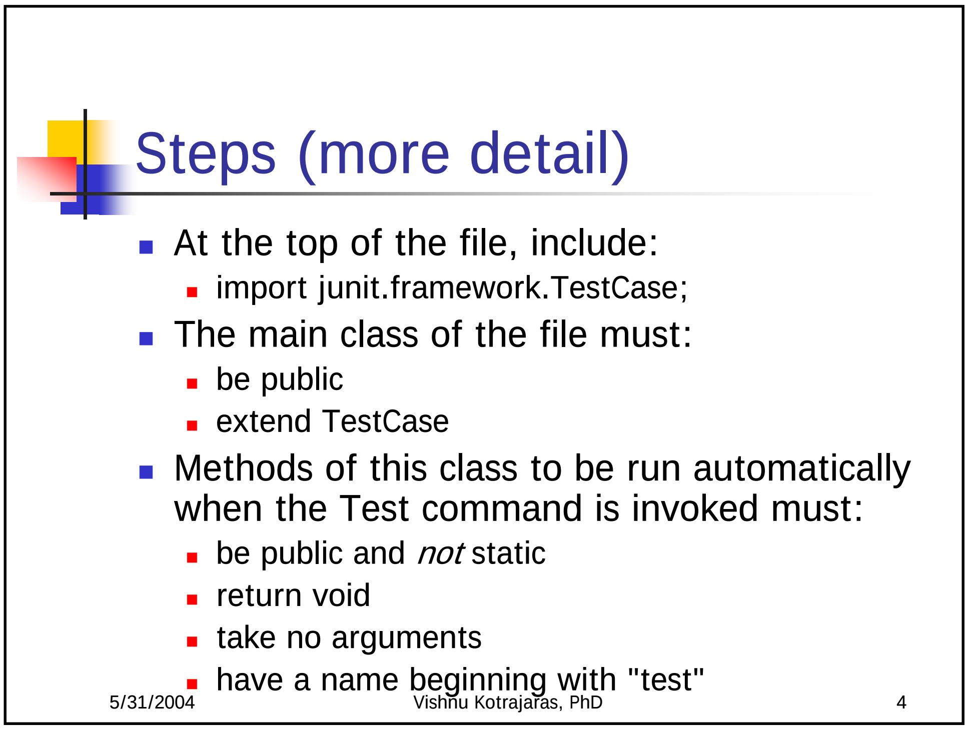
5/31/2004 Vishnu Kotrajaras, PhD 2



How to write it?

- create instances of your classes in the test methods of your TestCase class
- get results from any methods that you call
- and assert that those results match your expectations

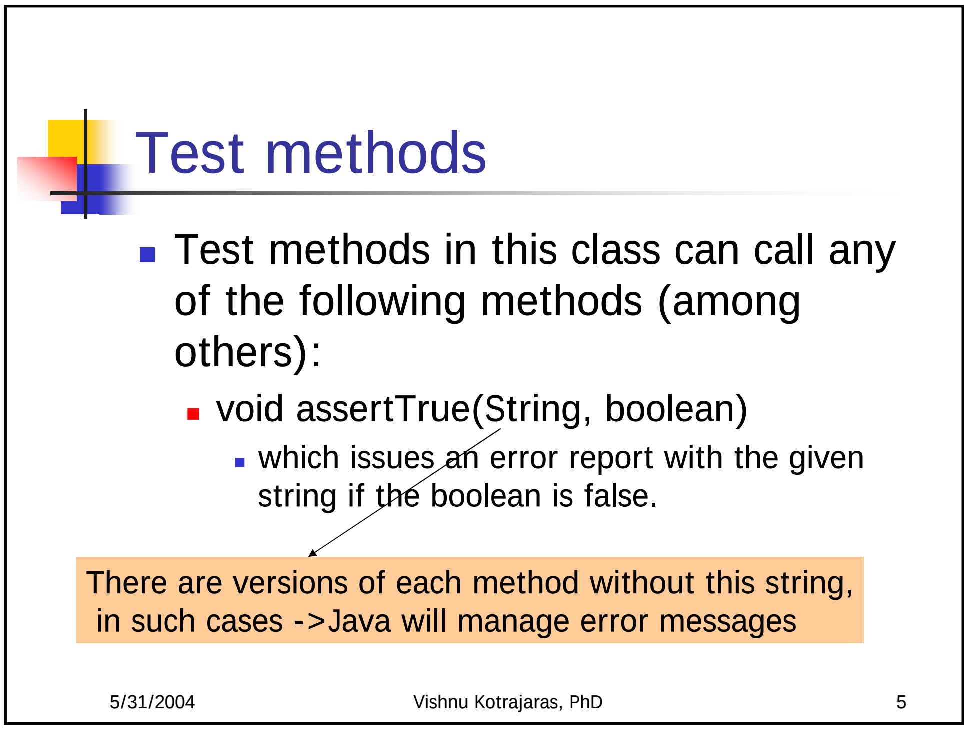
5/31/2004 Vishnu Kotrajaras, PhD 3



Steps (more detail)

- At the top of the file, include:
  - import junit.framework.TestCase;
- The main class of the file must:
  - be public
  - extend TestCase
- Methods of this class to be run automatically when the Test command is invoked must:
  - be public and *not* static
  - return void
  - take no arguments
  - have a name beginning with "test"

5/31/2004 Vishnu Kotrajaras, PhD 4

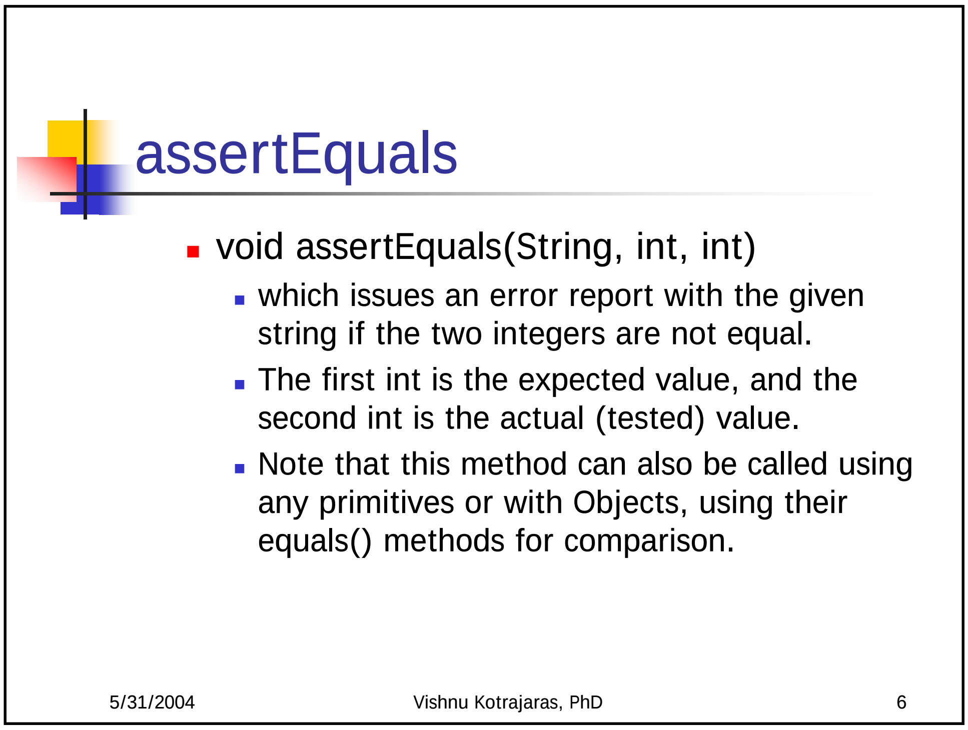


Test methods

- Test methods in this class can call any of the following methods (among others):
  - void assertTrue(String, boolean)
    - which issues an error report with the given string if the boolean is false.

There are versions of each method without this string, in such cases ->Java will manage error messages

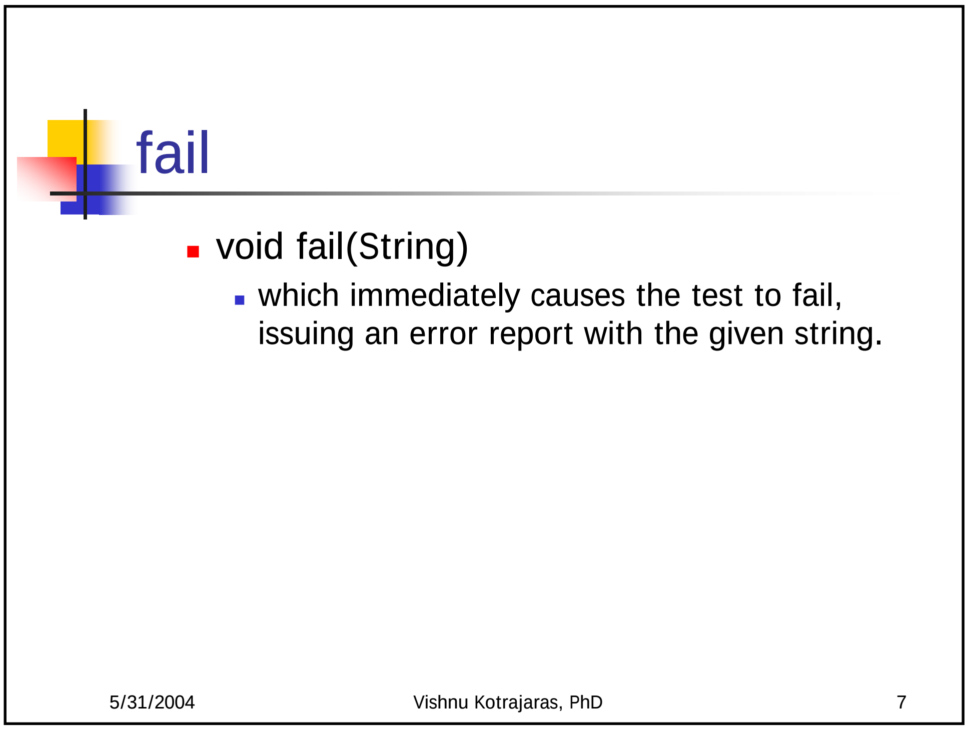
5/31/2004 Vishnu Kotrajaras, PhD 5



assertEquals

- void assertEquals(String, int, int)
  - which issues an error report with the given string if the two integers are not equal.
  - The first int is the expected value, and the second int is the actual (tested) value.
  - Note that this method can also be called using any primitives or with Objects, using their equals() methods for comparison.

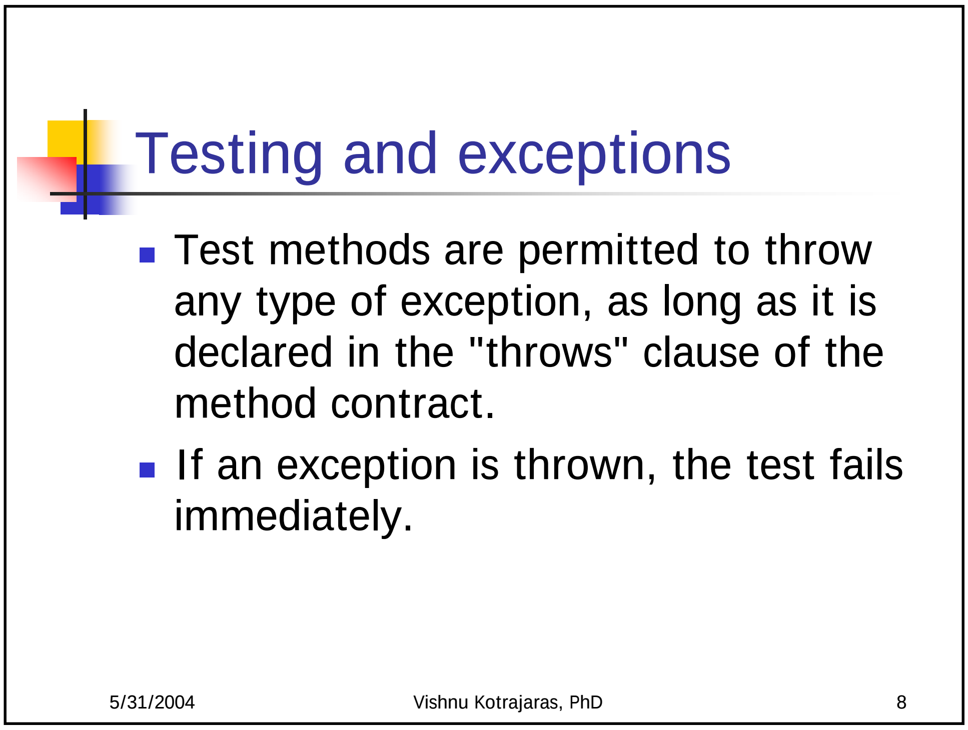
5/31/2004 Vishnu Kotrajaras, PhD 6



fail

- void fail(String)
  - which immediately causes the test to fail, issuing an error report with the given string.

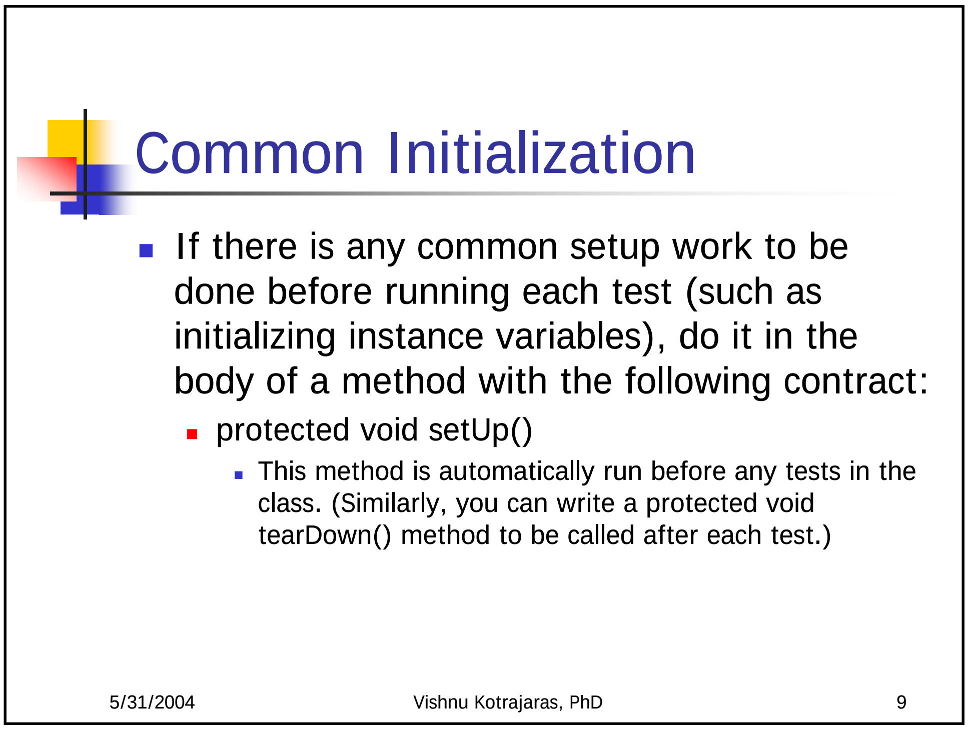
5/31/2004 Vishnu Kotrajaras, PhD 7



Testing and exceptions

- Test methods are permitted to throw any type of exception, as long as it is declared in the "throws" clause of the method contract.
- If an exception is thrown, the test fails immediately.

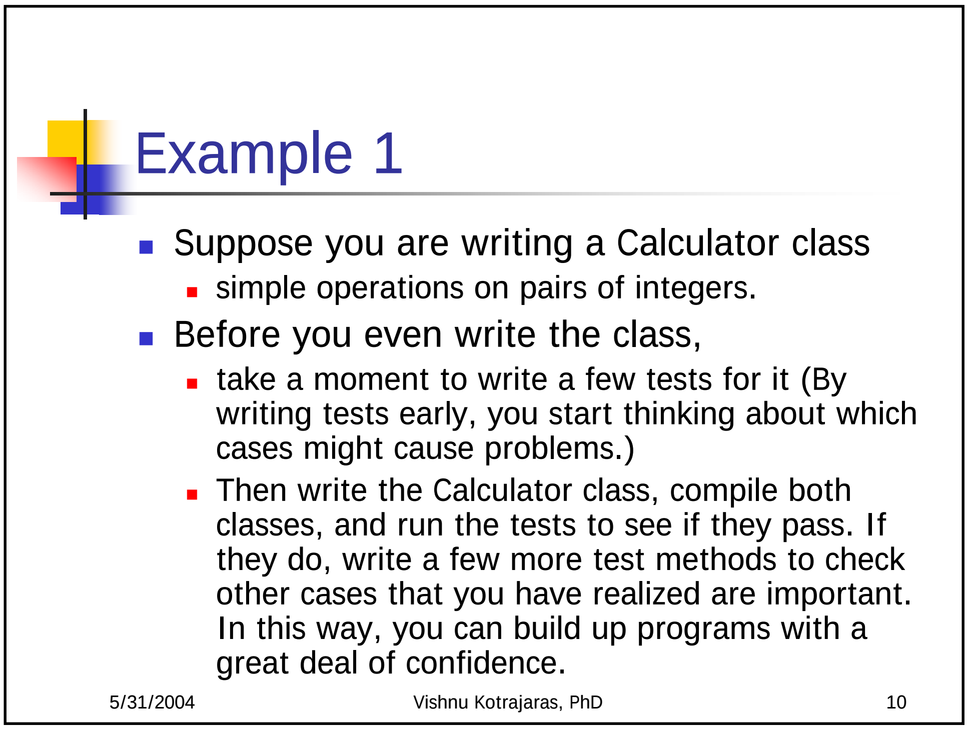
5/31/2004 Vishnu Kotrajaras, PhD 8



Common Initialization

- If there is any common setup work to be done before running each test (such as initializing instance variables), do it in the body of a method with the following contract:
  - protected void setUp()
    - This method is automatically run before any tests in the class. (Similarly, you can write a protected void tearDown() method to be called after each test.)

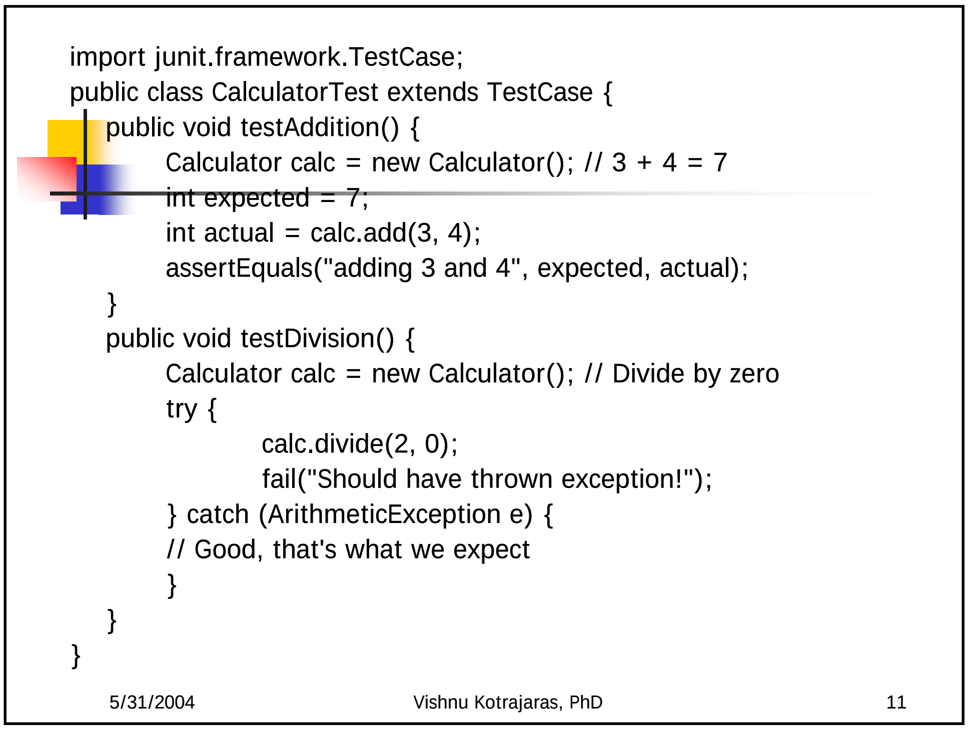
5/31/2004 Vishnu Kotrajaras, PhD 9



Example 1

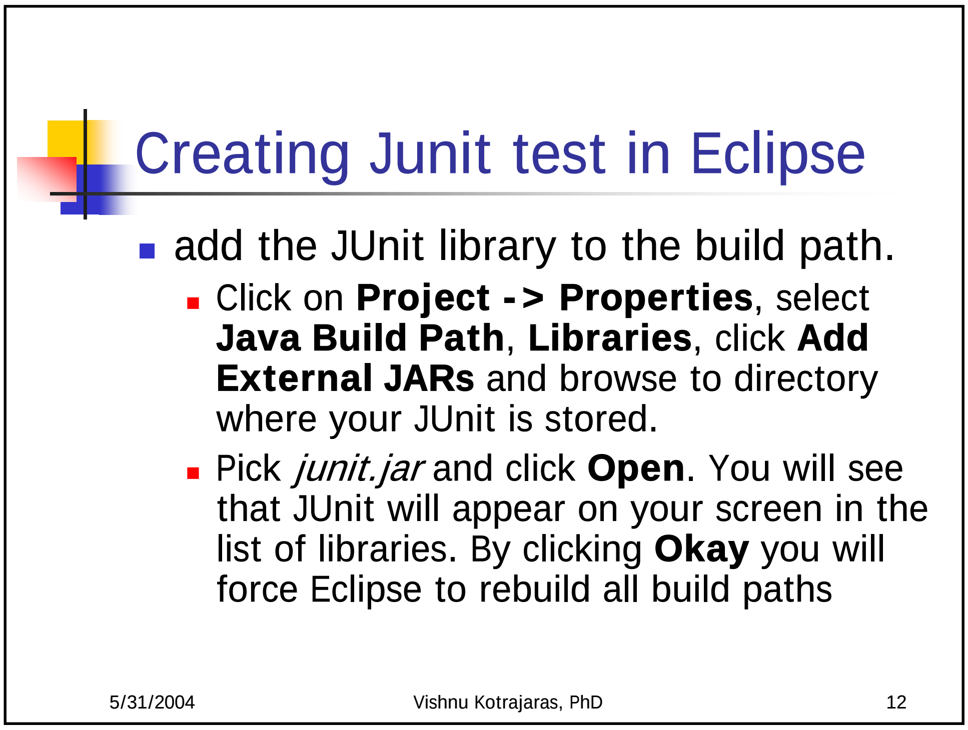
- Suppose you are writing a Calculator class
  - simple operations on pairs of integers.
- Before you even write the class,
  - take a moment to write a few tests for it (By writing tests early, you start thinking about which cases might cause problems.)
  - Then write the Calculator class, compile both classes, and run the tests to see if they pass. If they do, write a few more test methods to check other cases that you have realized are important. In this way, you can build up programs with a great deal of confidence.

5/31/2004 Vishnu Kotrajaras, PhD 10



```
import junit.framework.TestCase;
public class CalculatorTest extends TestCase {
    public void testAddition() {
        Calculator calc = new Calculator(); // 3 + 4 = 7
        int expected = 7;
        int actual = calc.add(3, 4);
        assertEquals("adding 3 and 4", expected, actual);
    }
    public void testDivision() {
        Calculator calc = new Calculator(); // Divide by zero
        try {
                calc.divide(2, 0);
                fail("Should have thrown exception!");
        } catch (ArithmeticException e) {
        // Good, that's what we expect
        }
    }
}
```

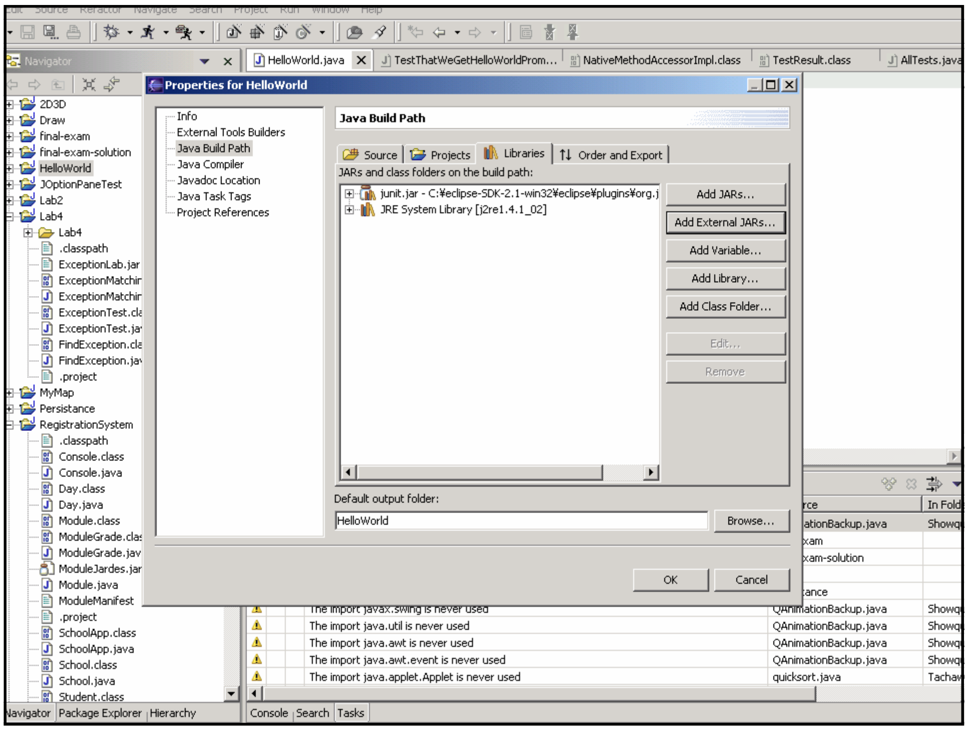
5/31/2004 Vishnu Kotrajaras, PhD 11



Creating Junit test in Eclipse

- add the JUnit library to the build path.
  - Click on **Project -> Properties**, select **Java Build Path**, **Libraries**, click **Add External JARs** and browse to directory where your JUnit is stored.
  - Pick *junit.jar* and click **Open**. You will see that JUnit will appear on your screen in the list of libraries. By clicking **Okay** you will force Eclipse to rebuild all build paths

5/31/2004 Vishnu Kotrajaras, PhD 12



Properties for HelloWorld

- Info
- External Tools Builders
- Java Build Path
- Java Compiler
- Javadoc Location
- Java Task Tags
- Project References

Java Build Path

Source | Projects | Libraries | Order and Export

JARs and class folders on the build path:

- junit.jar - C:¥eclipse-SDK-2.1-win32¥eclipse¥plugins¥org.j
- JRE System Library [j2re1.4.1_02]

Default output folder:

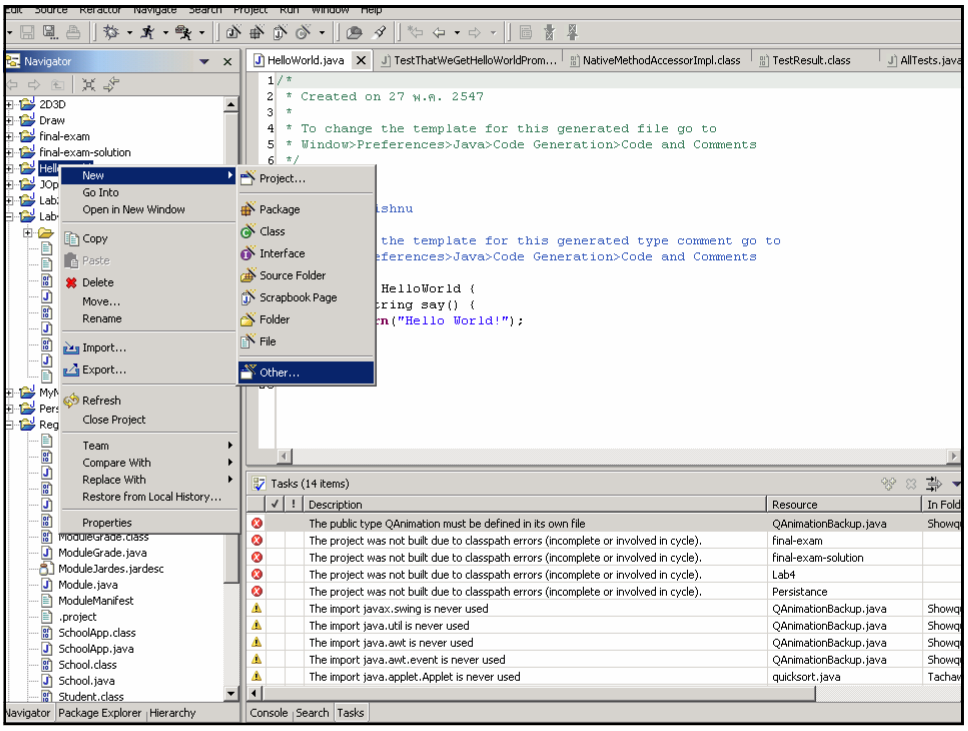
HelloWorld

```
The import java.util is never used          QAnimationBackup.java
The import java.awt is never used           QAnimationBackup.java
The import java.awt.event is never used     QAnimationBackup.java
The import java.applet.Applet is never used quicksort.java
```

Let us test HelloWorld

- To create such a test, right-click on the ProjectWithJUnit title, select **New -> Other**, expand the "Java" selection, and choose **JUnit**.
- On the right column of the dialog, choose **Test Case**, then click **Next**.

5/31/2004 Vishnu Kotrajaras, PhD 14

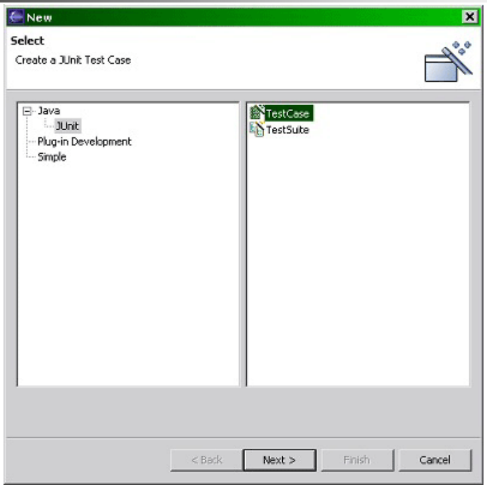


```
/*
 * Created on 27 พ.ค. 2547
 *
 * To change the template for this generated file go to
 * Window>Preferences>Java>Code Generation>Code and Comments
 */
```

```
ishnu

 the template for this generated type comment go to
eferences>Java>Code Generation>Code and Comments

 HelloWorld {
tring say() {
rn("Hello World!");
```

Tasks (14 items)

✓	!	Description	Resource	In Fold
	●	The public type QAnimation must be defined in its own file	QAnimationBackup.java	Showq
	●	The project was not built due to classpath errors (incomplete or involved in cycle).	final-exam	
	●	The project was not built due to classpath errors (incomplete or involved in cycle).	final-exam-solution	
	●	The project was not built due to classpath errors (incomplete or involved in cycle).	Lab4	
	●	The project was not built due to classpath errors (incomplete or involved in cycle).	Persistance	
	⚠	The import javax.swing is never used	QAnimationBackup.java	Showq
	⚠	The import java.util is never used	QAnimationBackup.java	Showq
	⚠	The import java.awt is never used	QAnimationBackup.java	Showq
	⚠	The import java.awt.event is never used	QAnimationBackup.java	Showq
	⚠	The import java.applet.Applet is never used	quicksort.java	Tachav



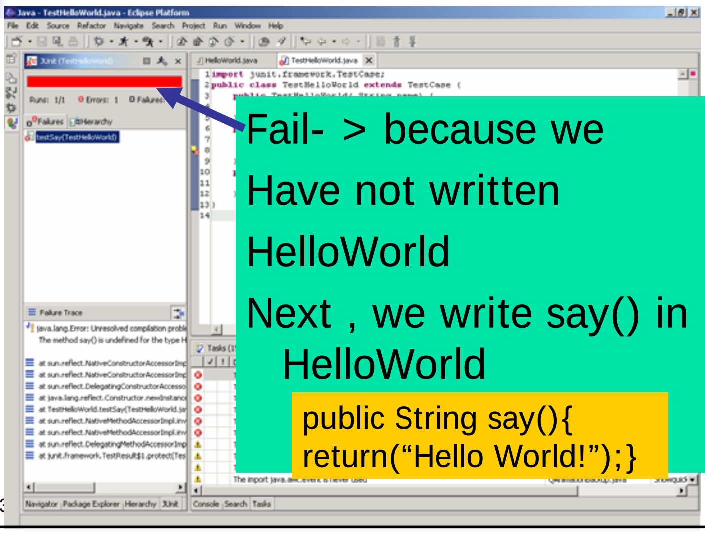
5/31/2004 Vishnu Kotrajaras, PhD 16

- Type in the name of our yet-to-be written class HelloWorld into the **Test class** field, and choose a name for our **Test case** -- for example, TestHelloWorld (yes, it looks long, but it clearly indicates what it does.) Click on **Finish**.

```
import junit.framework.TestCase;
public class TestHelloWorld extends TestCase {
    public TestHelloWorld( String name) {
        super(name);
    }
    public void testSay() {
        HelloWorld hi = new HelloWorld();
        assertEquals("Hello World!", hi.say()
    }
    public static void main(String[] args) {
        junit.textui.TestRunner.run( TestHelloWorld.class);
    }
}
```

text output and Eclipse IDE uses that to create its own graphic presentation. Normally we don't need this because eclipse evokes it automatically

Run ->Run as Junit Test


Java - TestHelloWorld.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
JUnit (TestHelloWorld)
Runs: 1/1 Errors: 1 Failures:
Failures Hierarchy
testSay(TestHelloWorld)
Failure Trace
HelloWorld.java
TestHelloWorld.java
Tasks
Navigator Package Explorer Hierarchy JUnit
Console Search Tasks

```
import junit.framework.TestCase;
public class TestHelloWorld extends TestCase {
```

```
java.lang.Error: Unresolved compilation proble
The method say() is undefined for the type H
at sun.reflect.NativeConstructorAccessorImp
at sun.reflect.NativeConstructorAccessorImp
at sun.reflect.DelegatingConstructorAccesso
at java.lang.reflect.Constructor.newInstance
at TestHelloWorld.testSay(TestHelloWorld.ja
at sun.reflect.NativeMethodAccessorImpl.inv
at sun.reflect.NativeMethodAccessorImpl.inv
at sun.reflect.DelegatingMethodAccessorImp
at junit.framework.TestResult$1.protect(Tes
```

Fail- > because we

Have not written

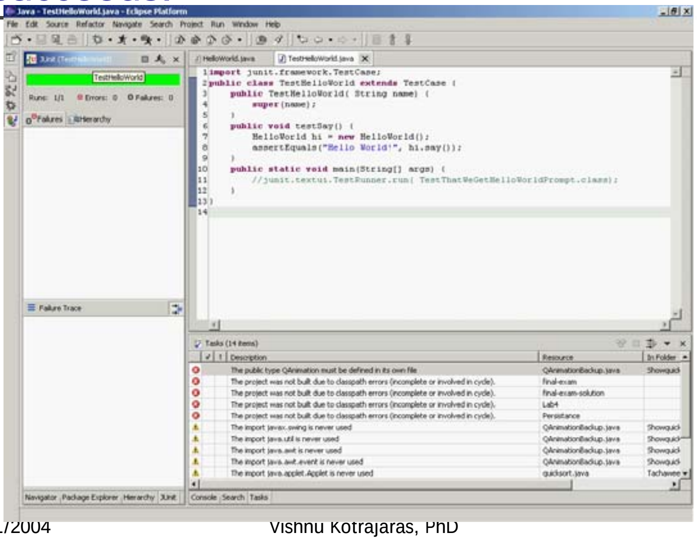
HelloWorld

Next , we write say() in
HelloWorld

```
public String say(){
return("Hello World!");}
```

5/3 19

Run Junit again. This time the test succeeds.


Java - TestHelloWorld.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
JUnit (TestHelloWorld)
TestHelloWorld
Runs: 1/1 Errors: 0 Failures: 0
Failures Hierarchy
Failure Trace
HelloWorld.java
TestHelloWorld.java
Tasks (14 items)
Navigator Package Explorer Hierarchy JUnit
Console Search Tasks

```
import junit.framework.TestCase;
public class TestHelloWorld extends TestCase {
    public TestHelloWorld( String name) {
        super(name);
    }
    public void testSay() {
        HelloWorld hi = new HelloWorld();
        assertEquals("Hello World!", hi.say());
    }
    public static void main(String[] args) {
        //junit.textui.TestRunner.run( TestThatWeGetHelloWorldPrompt.class);
    }
}
```

	✓	!	Description	Resource	In Folder
●			The public type QAnimation must be defined in its own file	QAnimationBackup.java	Showquick
●			The project was not built due to classpath errors (incomplete or involved in cycle).	final-exam	
●			The project was not built due to classpath errors (incomplete or involved in cycle).	final-exam-solution	
●			The project was not built due to classpath errors (incomplete or involved in cycle).	Lab4	
●			The project was not built due to classpath errors (incomplete or involved in cycle).	Persistance	
⚠			The import javax.swing is never used	QAnimationBackup.java	Showquick
⚠			The import java.util is never used	QAnimationBackup.java	Showquick
⚠			The import java.awt is never used	QAnimationBackup.java	Showquick
⚠			The import java.awt.event is never used	QAnimationBackup.java	Showquick
⚠			The import java.applet.Applet is never used	quicksort.java	Tachawee

5/31/2004 Vishnu Kotrajaras, PhD 20

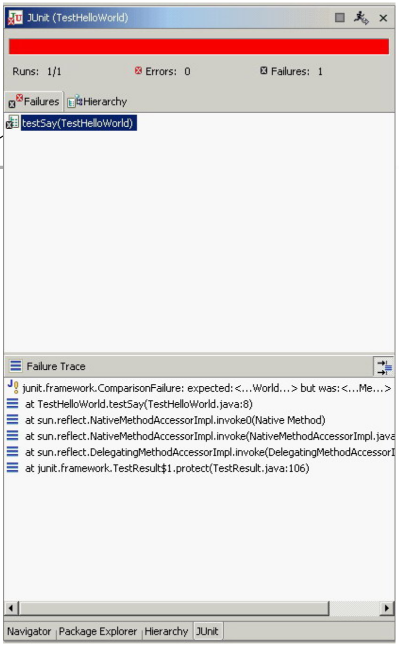
What about unexpected value?

- Edit the assertEquals() to change the expected return value from "Hello World!" to "Hello Me!".

5/31/2004 Vishnu Kotrajaras, PhD 21

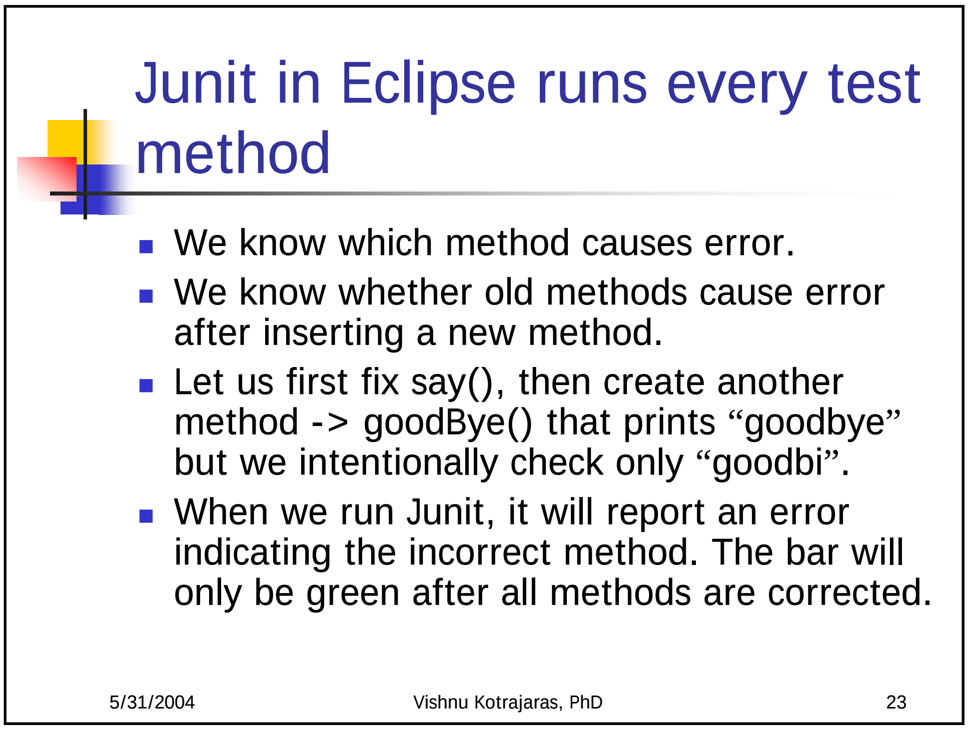
Run Junit again

Double click to go to that Method, or line in the method



```
junit.framework.ComparisonFailure: expected:<...World...> but was:<...Me...>
at TestHelloWorld.testSay(TestHelloWorld.java:8)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorI
at junit.framework.TestResult$1.protect(TestResult.java:106)
```

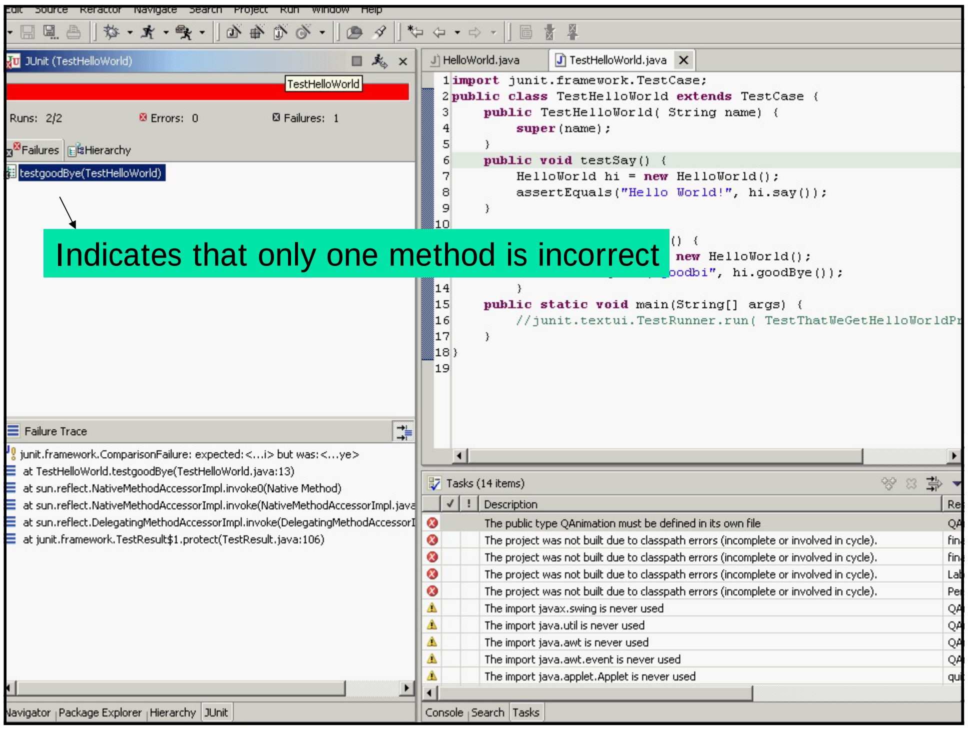
5/31/2004 Vishnu Kotrajaras, PhD 22



Junit in Eclipse runs every test method

- We know which method causes error.
- We know whether old methods cause error after inserting a new method.
- Let us first fix say(), then create another method -> goodBye() that prints “goodbye” but we intentionally check only “goodbi”.
- When we run Junit, it will report an error indicating the incorrect method. The bar will only be green after all methods are corrected.

5/31/2004 Vishnu Kotrajaras, PhD 23



```
Runs: 2/2          Errors: 0          Failures: 1
testgoodBye(TestHelloWorld)
```

Indicates that only one method is incorrect

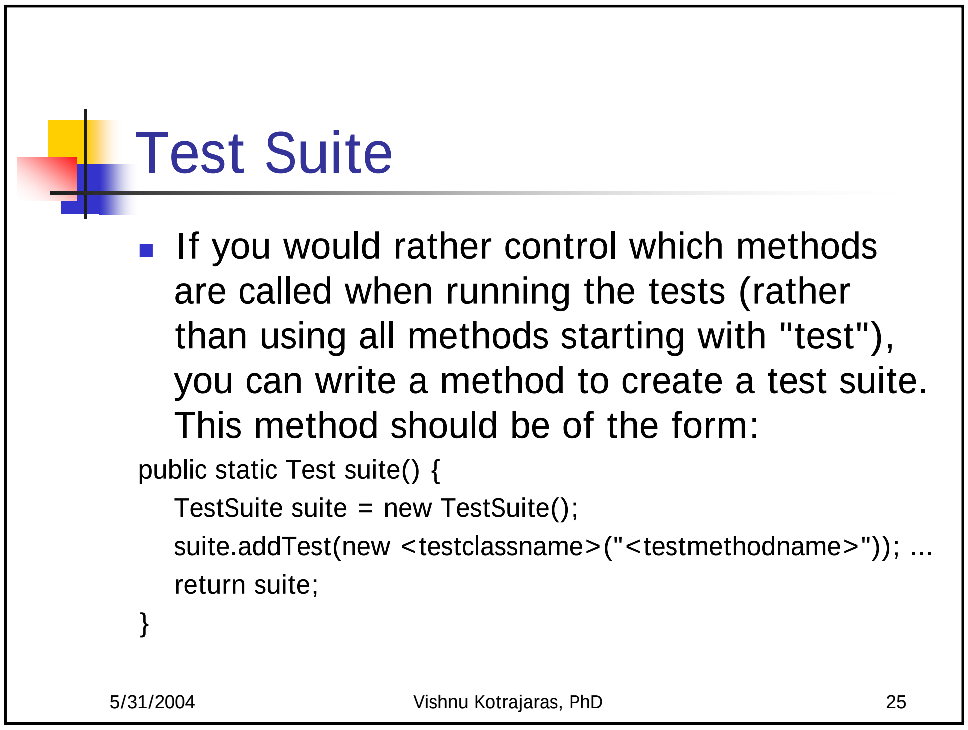
```
import junit.framework.TestCase;
public class TestHelloWorld extends TestCase {
    public TestHelloWorld( String name) {
        super(name);
    }
    public void testSay() {
        HelloWorld hi = new HelloWorld();
        assertEquals("Hello World!", hi.say());
    }

                                () {
                                 new HelloWorld();
                              odbi", hi.goodBye());
        }
    public static void main(String[] args) {
        //junit.textui.TestRunner.run( TestThatWeGetHelloWorldPr
    }
}
```

Failure Trace

```
junit.framework.ComparisonFailure: expected:<...i> but was:<...ye>
 at TestHelloWorld.testgoodBye(TestHelloWorld.java:13)
 at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
 at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java
 at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorI
 at junit.framework.TestResult$1.protect(TestResult.java:106)
```

Tasks (14 items)

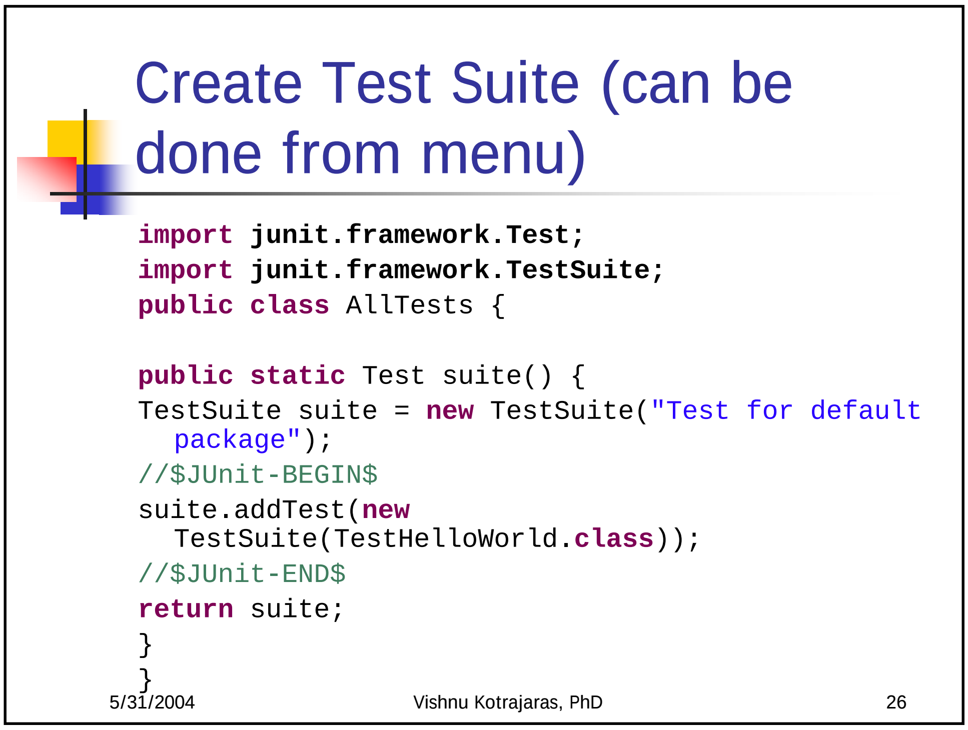
	✓	!	Description	Re
⊗			The public type QAnimation must be defined in its own file	QA
⊗			The project was not built due to classpath errors (incomplete or involved in cycle).	fin
⊗			The project was not built due to classpath errors (incomplete or involved in cycle).	fin
⊗			The project was not built due to classpath errors (incomplete or involved in cycle).	Lab
⊗			The project was not built due to classpath errors (incomplete or involved in cycle).	Pe
⚠			The import javax.swing is never used	QA
⚠			The import java.util is never used	QA
⚠			The import java.awt is never used	QA
⚠			The import java.awt.event is never used	QA
⚠			The import java.applet.Applet is never used	qu



Test Suite

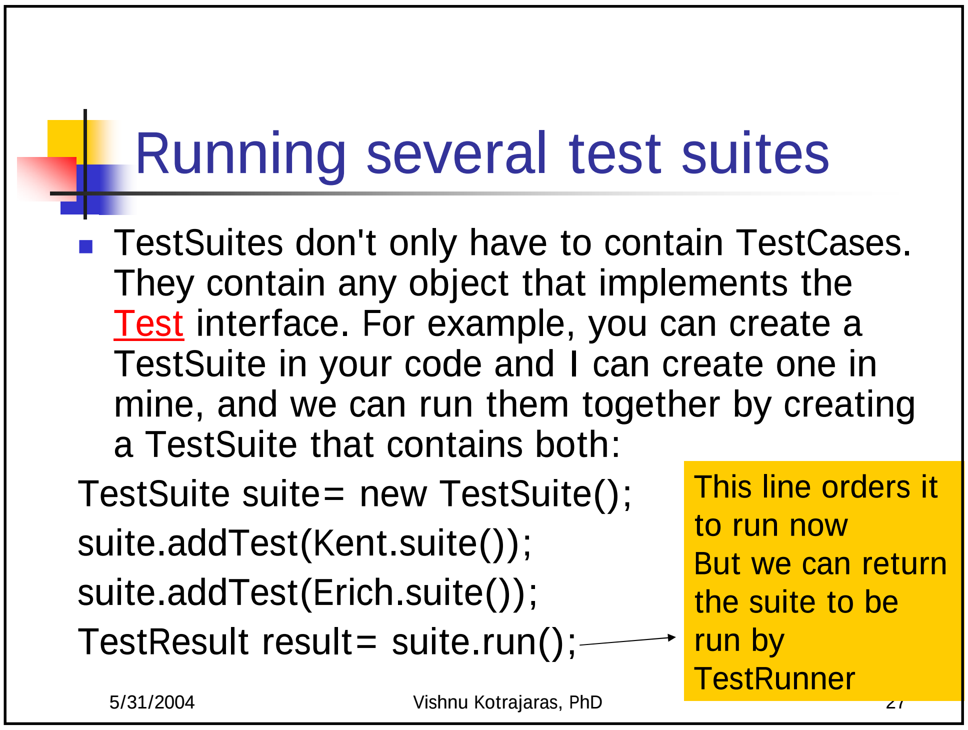
- If you would rather control which methods are called when running the tests (rather than using all methods starting with "test"), you can write a method to create a test suite. This method should be of the form:

```
public static Test suite() {
    TestSuite suite = new TestSuite();
    suite.addTest(new <testclassname>("<testmethodname>")); ...
    return suite;
}
```



Create Test Suite (can be done from menu)

```
import junit.framework.Test;
import junit.framework.TestSuite;
public class AllTests {

public static Test suite() {
TestSuite suite = new TestSuite("Test for default
  package");
//$JUnit-BEGIN$
suite.addTest(new
  TestSuite(TestHelloWorld.class));
//$JUnit-END$
return suite;
}
}
```



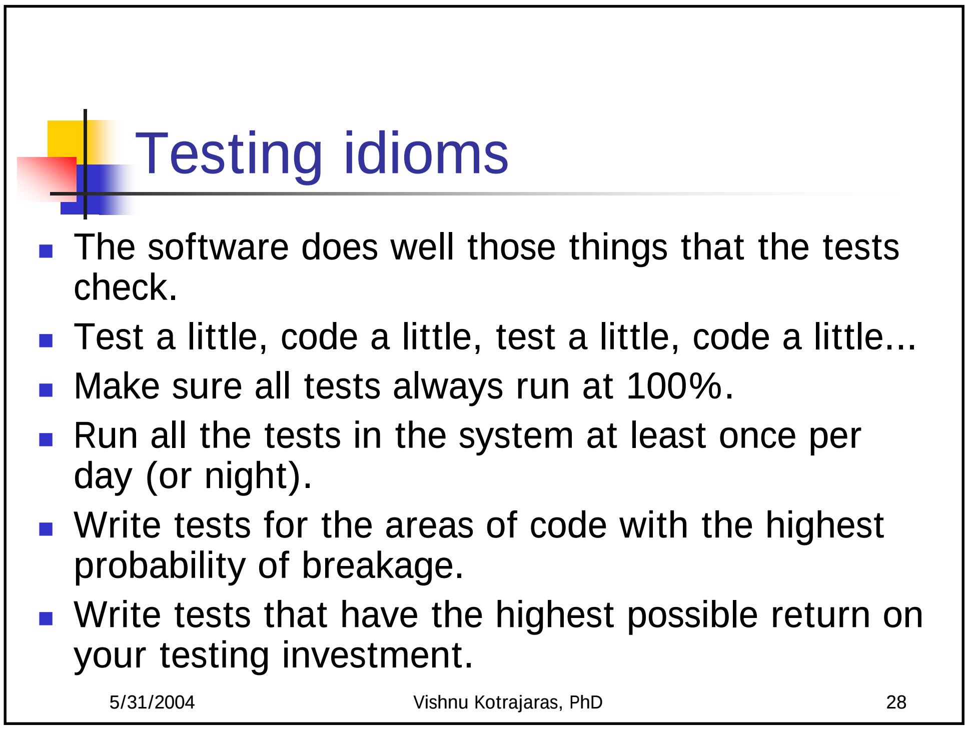
Running several test suites

- TestSuites don't only have to contain TestCases. They contain any object that implements the Test interface. For example, you can create a TestSuite in your code and I can create one in mine, and we can run them together by creating a TestSuite that contains both:

```
TestSuite suite= new TestSuite();
suite.addTest(Kent.suite());
suite.addTest(Erich.suite());
TestResult result= suite.run();
```

This line orders it to run now
But we can return the suite to be run by TestRunner

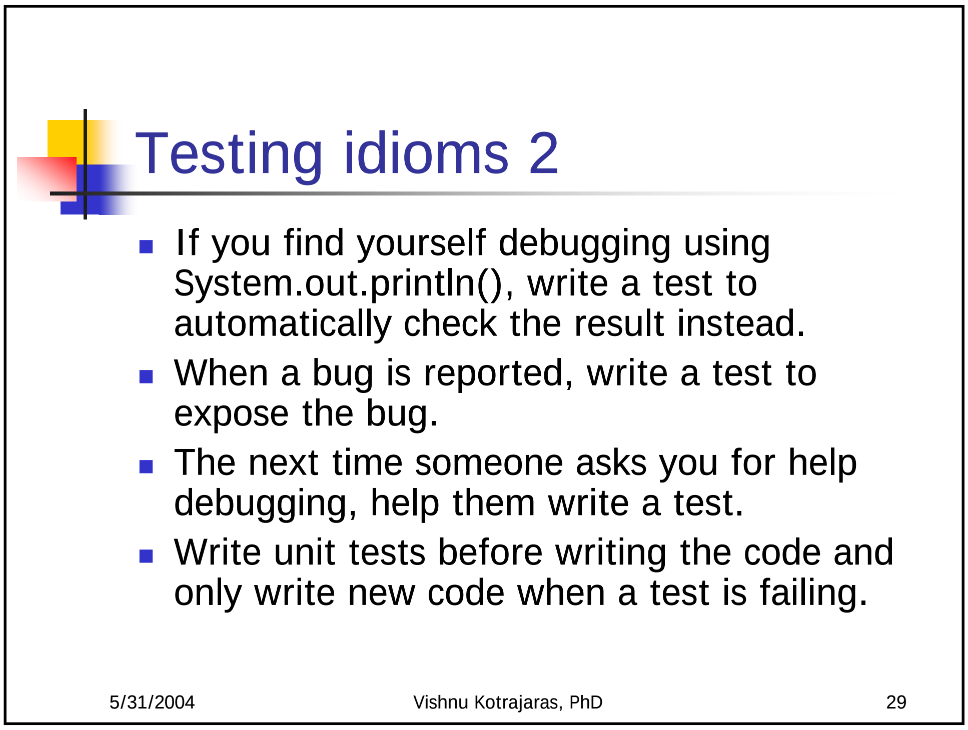
5/31/2004 Vishnu Kotrajaras, PhD 27



Testing idioms

- The software does well those things that the tests check.
- Test a little, code a little, test a little, code a little...
- Make sure all tests always run at 100%.
- Run all the tests in the system at least once per day (or night).
- Write tests for the areas of code with the highest probability of breakage.
- Write tests that have the highest possible return on your testing investment.

5/31/2004 Vishnu Kotrajaras, PhD 28


Testing idioms 2

- If you find yourself debugging using System.out.println(), write a test to automatically check the result instead.
- When a bug is reported, write a test to expose the bug.
- The next time someone asks you for help debugging, help them write a test.
- Write unit tests before writing the code and only write new code when a test is failing.

5/31/2004 Vishnu Kotrajaras, PhD 29