6CS002

Unit Testing

Dr. Kevan Buckley

Lecture Outcomes

- To be aware of the features of a unit testing framework
- To be able to perform unit tests with JUnit
- To be able to implement your own testing framework

Testing a Simple Class

```
public class Rectangle {
  private double width;
  private double height;

  public Rectangle(double width, double height) {
    this.width = width;
    this.height = height;
  }

  public double getWidth() {
    return width;
  }

  public void setWidth(double width) {
    this.width = width;
  }
```

Testing a Simple Class

```
  public double getHeight() {
    return height;
  }

  public void setHeight(double height) {
    this.height = height;
  }

  public double getArea() {
    return width*height;
  }

  public double getDiagonalLength(){
    return Math.sqrt(width*width+height*height);
  }
}
```

File Edit Source Refactor Navigate Search Project Run Window Help

- New
  - Java Project
  - Project...
  - Package
  - Class
  - Interface
  - Enum
  - Annotation
  - Source Folder
  - Java Working Set
  - Folder
  - File
  - Untitled Text File
  - JUnit Test Case
  - Other...
- Open File...
- Close
- Close All
- Save
- Save As...
- Save All
- Revert
- Move...
- Rename...
- Refresh
- Convert Line Delimiters To
- Print...
- Switch Workspace
- Restart
- Import...
- Export...
- Properties
- 1 Simple.java [reflection/src/kunitDemo1]
- 2 TestSimple.java [reflection/src/...]
- 3 KUnit.java [reflection/src/kunit2]
- 4 Main.java [reflection/src/kunitDemo2]
- Exit

JUnit 4

imple.java

```
package kunitDemo1;

public class Simple {

  public int a = 10;
  private int b = 20;

  public Simple() {
  }

  public Simple(int a, int b) {
    this.a = a;
    this.b = b;
  }

  public void squareA(){
    this.a *= this.a;
  }

  private void squareB(){
    this.b *= this.b;
  }

  public int getA() {
    return a;
  }

  private void setA(int a) {
    this.a = a;
  }

  public int getB() {
    return b;
  }
```

New JUnit Test Case

JUnit Test Case

⚠ The use of the default package is discouraged.

○ New JUnit 3 test ◉ New JUnit 4 test

Source folder: reflection/src [Browse...]

Package: (default) [Browse...]

Name: TestRectangle

Superclass: java.lang.Object [Browse...]

Which method stubs would you like to create?

☐ setUpBeforeClass() ☐ tearDownAfterClass()

☐ setUp() ☐ tearDown()

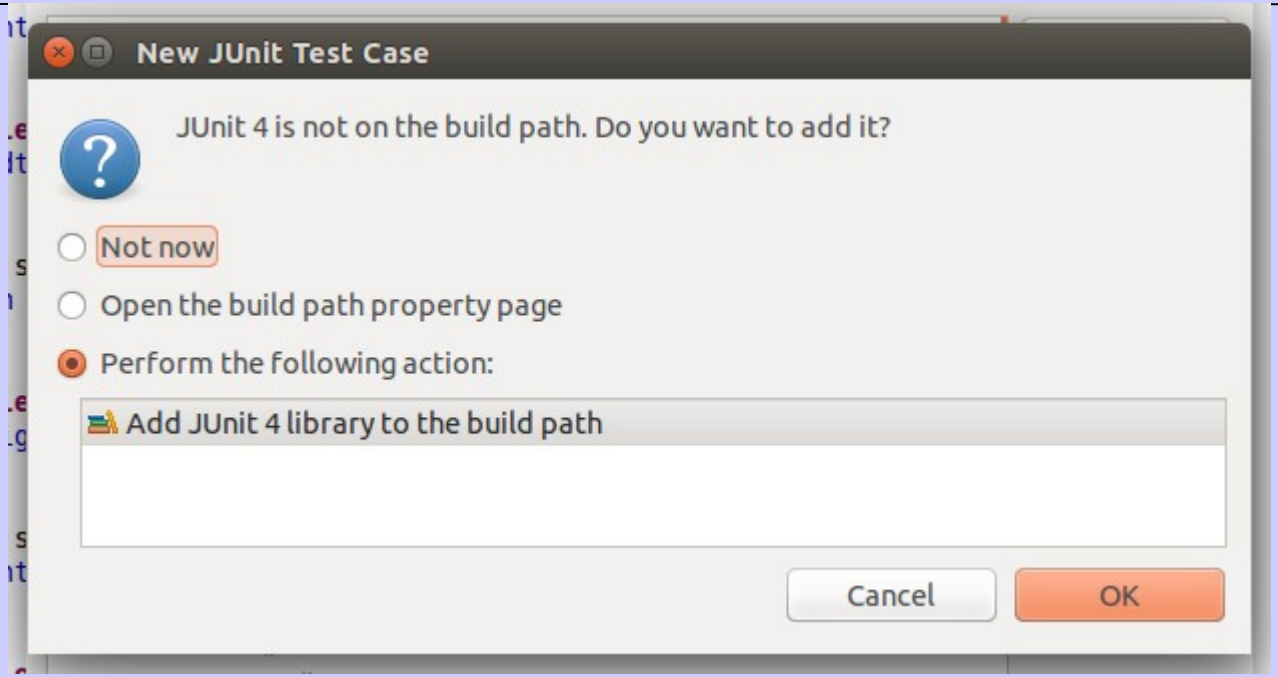
☐ constructor

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

Class under test: lecture2.Rectangle [Browse...]

? < Back | Next > | Cancel | Finish


New JUnit Test Case
JUnit 4 is not on the build path. Do you want to add it?
Not now
Open the build path property page
Perform the following action:
Add JUnit 4 library to the build path
Cancel
OK

Testing a Simple Class

```
import static org.junit.Assert.*;
import org.junit.Test;

public class RectangleTest {

  @Test
  public void testConstructionAndAccess() {
    Rectangle r = new Rectangle(1.3, 2.5);
    assertEquals(1.3, r.getWidth(), 0.001);
    assertEquals(2.5, r.getHeight(), 0.001);
    r.setWidth(5.6);
    r.setHeight(6.7);
    assertEquals(5.6, r.getWidth(), 0.001);
    assertEquals(6.7, r.getHeight(), 0.001);
  }
```

Indicates that the next method is a JUnit test

Tolerance

Actual

Expected

Testing a Simple Class

```
  @Test
  public void testGetArea() {
    Rectangle r = new Rectangle(1.3, 2.5);
    assertEquals(3.25, r.getArea(), 0.001);
  }

  @Test
  public void testGetDiagonalLength() {
    Rectangle r = new Rectangle(3, 4);
    assertEquals(5, r.getDiagonalLength(), 0.001);
  }

}
```

Assertions

- assertEquals
  - Expected, Actual, Tolerance(only for real numbers)
- assertTrue/assertFalse
  - Use these with booleans or classes that have an *equals* method
    - e.g. use this to compare strings:

```
String expected = "Kevan";
String actual = person.getName();
assertTrue(expected.equals(actual));
```

- fail
  - Use this to indicate tests that have not been implemented yet

Package Explorer | JUnit

Finished after 0.012 seconds

Runs: 3/3 Errors: 0 Failures: 0

lecture2.TestRectangle [Runner: JUnit 4] (0.004 s)

Failure Trace

Rectangle.java | TestRectangle.java

```java
package lecture2;
import static org.junit.Assert.*;


public class TestRectangle {

  @Test
  public void testConstructionAndAccess() {
    Rectangle r = new Rectangle(1.3, 2.5);
    assertEquals(1.3, r.getWidth(), 0.001);
    assertEquals(2.5, r.getHeight(), 0.001);
    r.setWidth(5.6);
    r.setHeight(6.7);
    assertEquals(5.6, r.getWidth(), 0.001);
    assertEquals(6.7, r.getHeight(), 0.001);
  }

  @Test
  public void testGetArea() {
    Rectangle r = new Rectangle(1.3, 2.5);
    assertEquals(3.25, r.getArea(), 0.001);
  }

  @Test
  public void testGetDiagonalLength() {
    Rectangle r = new Rectangle(3, 4);
    assertEquals(5, r.getDiagonalLength(), 0.001);
  }

}
```

A Very Simple Unit Test Framework

KUnit

KUnit

```java
public class KUnit {
  private static List<String> checks;
  private static int checksMade = 0;
  private static int passedChecks = 0;
  private static int failedChecks = 0;

  private static void addToReport(String txt) {
    if (checks == null) {
      checks = new LinkedList<String>();
    }
    checks.add(String.format("%04d: %s", checksMade++, txt));
  }

  public static void checkEquals(int value1, int value2) {
    if (value1 == value2) {
      addToReport(String.format("  %d == %d", value1, value2));
      passedChecks++;
    } else {
      addToReport(String.format("* %d == %d", value1, value2));
      failedChecks++;
    }
  }
```

KUnit

```
  public static void checkNotEquals(int value1, int value2) {
    if (value1 != value2) {
      addToReport(String.format("  %d != %d", value1, value2));
      passedChecks++;
    } else {
      addToReport(String.format("* %d != %d", value1, value2));
      failedChecks++;
    }
  }

  public static void report() {
    System.out.printf("%d checks passed\n", passedChecks);
    System.out.printf("%d checks failed\n", failedChecks);
    System.out.println();

    for (String check : checks) {
      System.out.println(check);
    }
  }
}
```

Using KUnit

```java
public class TestSimple {

  void checkConstructorAndAccess(){
    Simple s = new Simple(3, 4);
    checkEquals(s.getA(), 4);
    checkEquals(s.getB(), 4);
    checkNotEquals(s.getB(), 4);
    checkNotEquals(s.getB(), 5);
  }

  void checkSquareA(){
    Simple s = new Simple(3, 4);
    s.squareA();
    checkEquals(s.getA(), 9);
  }

  public static void main(String[] args) {
    TestSimple ts = new TestSimple();
    ts.checkConstructorAndAccess();
    ts.checkSquareA();
    report();
  }
}
```

```
3 checks passed
2 checks failed

0000: * 3 == 4
0001:   4 == 4
0002: * 4 != 4
0003:   4 != 5
0004:   9 == 9
```

A ~~Very~~ Simple Unit Test Framework

KUnit2

KUnit2

Note no calls to check methods now

```
package kunitDemo2;
import static kunit2.KUnit.*;

public class TestSimple {

  public void checkConstructorAndAccess(){
    Simple s = new Simple(3, 4);
    checkEquals(s.getA(), 4);
    checkEquals(s.getB(), 4);
    checkNotEquals(s.getB(), 4);
    checkNotEquals(s.getB(), 5);
  }

  public void checkSquareA(){
    Simple s = new Simple(3, 4);
    s.squareA();
    checkEquals(s.getA(), 9);
  }
}
```

KUnit2

There is a built in launcher for all check methods

- Work out how to do this with reflection

```
public class Main {

  public static void main(String[] args) {
    TestSimple ts = new TestSimple();
    runChecks(ts);
    report();
  }
}
```

Features of A Unit Test Framework

- Assertions
  - Methods that can be called to check for something
  - Static imports simplify syntax
  - Problems using the word “assert” in your own code so use something else
- Logging and reporting
  - Results of assertions are recorded
  - Report is produced to indicate level of success
  - Often reports can be filtered to only include failures
- Test launcher
  - A means of launching all tests
  - Testing methods can be given specific names or annotations (@test)
- Resilience to exceptions
  - A test causing an exceptions should not stop further tests running