



Test First Development Using Eclipse

Wayne Beaton

Evangelist, Eclipse Foundation

Agenda

- What are you testing?
- Testing straight-up Java code
- Testing Eclipse plug-ins
- Testing Java EE applications

Expectation Management

- Exclusively concerned with unit testing
- Other forms of testing are also required
 - Integration testing
 - Functional testing
 - System testing
 - Acceptance testing
 - Performance testing

What Are You Testing?

- Your frameworks
- Your infrastructure
- Your code

What Aren't You Testing?

- Other people's frameworks
- Other people's infrastructure
- Other people's code

- If you must test other people's code, do it in isolation!

Test First Tools

- JUnit
- Red Xs and red squiggly lines
- CTRL-1
- CTRL-space

JUnit 3.8.1 (1/2)

- Test class extends `junit.framework.TestCase`
- Test methods start with 'test'
- Inherited `assert*`, fail methods for validating
- Fixtures managed with `setUp` and `tearDown`

JUnit 3.8.1 (2/2)

The screenshot shows the Eclipse IDE with a code completion menu open for the word "test". A yellow callout bubble with the text "ctrl-space" points to the menu. The menu lists several options, with "test - test method" selected. The background code shows a method signature: `public void testname() throws Exc`.

```
}  
test  
priv @ test - test method  
Test - org.junit  
Test - junit.framework  
Test - test method (JUnit 4)  
test() void - Method stub  
TestableObject - org.eclipse.ui.testing  
TestAcceptor1 - com.sun.corba.se.impl.orb.Pa  
priv TestAcceptor2 - com.sun.corba.se.impl.orb.Pa  
TestBadServerIdHandler - com.sun.corba.se.ii  
Press 'Ctrl+Space' to show Template Proposals
```

```
public void testname() throws Exc  
{  
}
```

JUnit 4 (1/2)

- Uses Java 5 Annotations
- Tests indicated with `@Test`
- `org.junit.Assert.assert*`, fail static methods
- Fixture management with `@Before`, `@After`, `@BeforeClass`, `@AfterClass`

```
import static junit.framework.Assert.*;
```

JUnit 4 (2/2)

```
assertEquals("Blue Dress", view.nameText.getText());
assertEquals("USD$24.99", view.priceText.getText());
}
```

ctrl-space

test

- test - test method
- @ Test - org.junit**
- Test - junit.framework
- Test - test method (JUnit 4)**
- test() void - Method stub
- TestableObject - org.eclipse.ui.testing
- TestAcceptor1 - com.sun.corba.se.impl.orb.Pa
- priv TestAcceptor2 - com.sun.corba.se.impl.orb.Pa**
- TestBadServerIdHandler - com.sun.corba.se.i

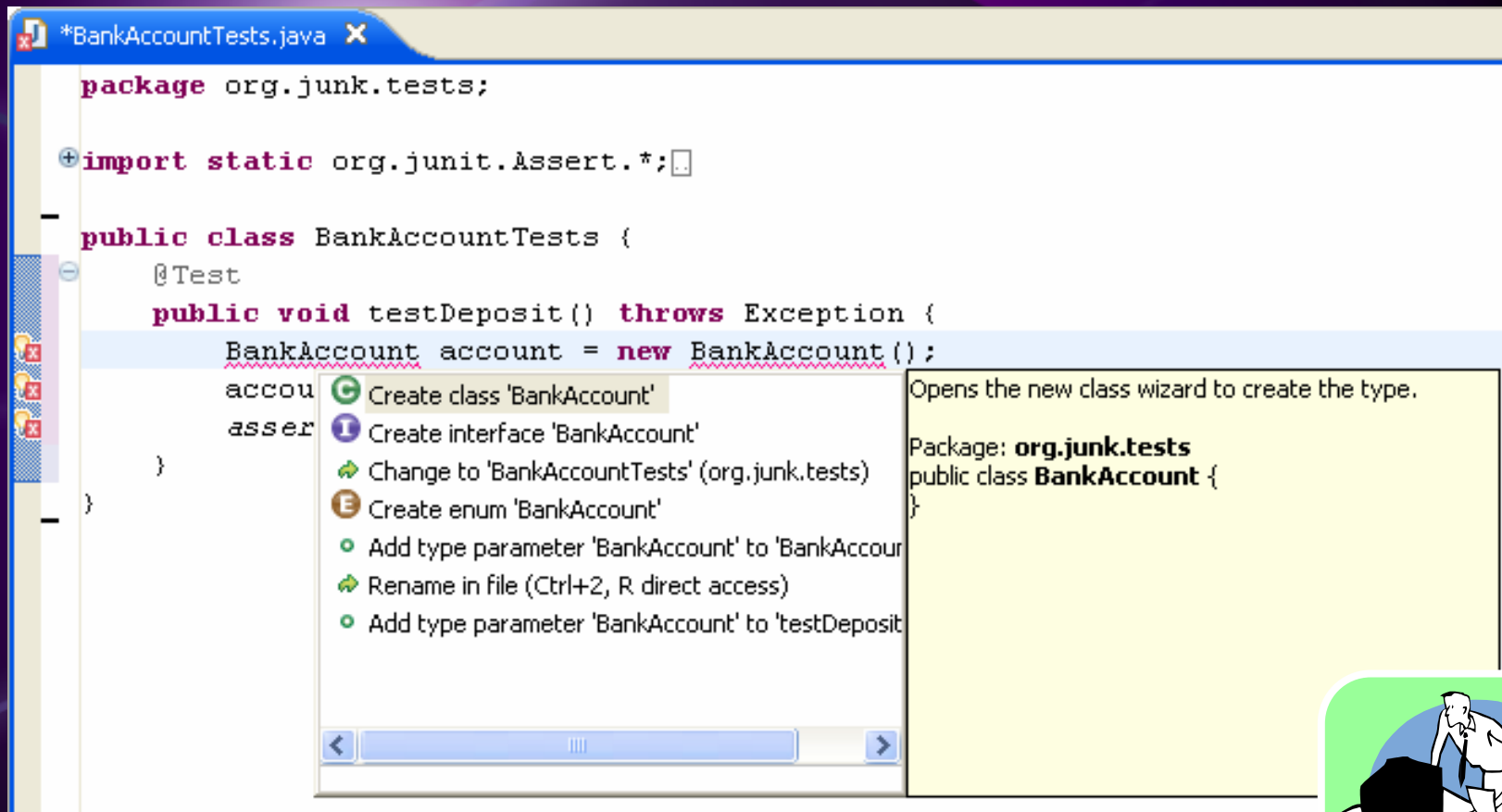
```
@Test
public void testname() throws Exc
}
```

Press 'Ctrl+Space' to show Template Proposals

Build Tests Before the Code Being Tested

- Red Xs are your friends!
- Write tests that show how you want to use the code
- You may not get it right the first time (and that's okay)

CTRL-1 Your Way Through the Errors



The screenshot shows an IDE window titled '*BankAccountTests.java'. The code contains a package declaration, an import statement for JUnit, and a public class 'BankAccountTests' with a test method 'testDeposit'. A red squiggly line under 'BankAccount' in the 'new BankAccount()' call indicates a compilation error. A context menu is open over this error, listing options: 'Create class 'BankAccount'', 'Create interface 'BankAccount'', 'Change to 'BankAccountTests' (org.junk.tests)', 'Create enum 'BankAccount'', 'Add type parameter 'BankAccount' to 'BankAccount'', 'Rename in file (Ctrl+2, R direct access)', and 'Add type parameter 'BankAccount' to 'testDeposit''. A tooltip on the right explains that the first option opens the new class wizard.

```
package org.junk.tests;

import static org.junit.Assert.*;

public class BankAccountTests {
    @Test
    public void testDeposit() throws Exception {
        BankAccount account = new BankAccount();
    }
}
```

Options in the context menu:

- Create class 'BankAccount'
- Create interface 'BankAccount'
- Change to 'BankAccountTests' (org.junk.tests)
- Create enum 'BankAccount'
- Add type parameter 'BankAccount' to 'BankAccount'
- Rename in file (Ctrl+2, R direct access)
- Add type parameter 'BankAccount' to 'testDeposit'

Tooltip text: Opens the new class wizard to create the type.

Package: **org.junk.tests**
public class **BankAccount** {
}



Working with Frameworks a Little Harder

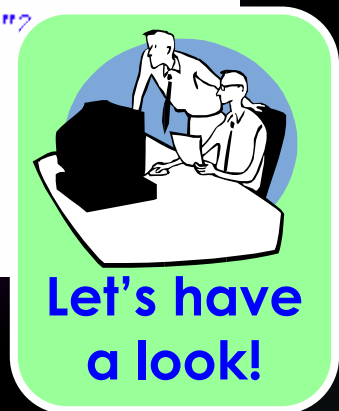
- Not often as simple as CTRL-1
- Layer isolation
 - Good for architecture anyway!
- Test Harness/Mock Objects

Testing Eclipse Plug-ins

- Remember that you're trying to test **your code**
- Use fragments to gain privileged access while keeping tests separate

```
@Test
public void testSelection() throws Exception {
    AuctionItemSummaryView view = (AuctionItemSummaryView) getActivePage()
        .showView("org.eclipse.samples.stuff.AuctionItemSummaryView");
    view.selectionHandler.selectionChanged(null, new StructuredSelection() {
        public boolean isEmpty() {
            return false;
        }
        public Object getFirstElement() {
            return new AuctionItem("Blue Dress", new BigDecimal(new BigInteger("2499")));
        }
    });

    assertEquals("Blue Dress", view.nameText.getText());
    assertEquals("USD$24.99", view.priceText.getText());
}
```



Testing Java EE Applications

- Remember that you're trying to test **your code**
- Test harness/mock objects instead of runtime

```
public void testGetAuctionInformation() throws Exception {  
    // mocks request URL http://host:port/root/servlet?id=42  
    HttpServletRequest request = new MockHttpServletRequest("GET");  
    request.setAttribute("id", "42");  
  
    HttpServletResponse response = new MockHttpServletResponse();  
  
    servlet.service(request, response);  
  
    assertEquals("Blue Dress", request.getAttribute("name"));  
    assertEquals("$42.00", request.getAttribute("price"));  
}
```



Testing Database Access (1/2)

```
@Override
protected void setUp() throws Exception {
    super.setUp();
    backend = new DerbyBackend(directory.getAbsolutePath());
    backend.initialize(OrganizerManager.instance);

    task = new Task();
    task.setSubject("Test Subject");

    connection = DriverManager.getConnection("jdbc:derby:" + directory.getAbsolutePath());
    statement = connection.createStatement();
}
```

Testing Database Access (2/2)

```
public void testAddSingleTask() throws Exception {
    Date rightNow = DateFormat.getDateInstance(DateFormat.SHORT, Locale.CANADA).parse("8/6/06");
    task.setDueDate(rightNow);

    backend.addTasks(Collections.singletonList(task));

    assertEquals(task, backend.getTasks().iterator().next());

    // Id should have been filled in by the addTasks call.
    assertNotNull(task.getId());

    // Query the database directly to ensure that the right things have been written.
    ResultSet results = statement.executeQuery("select subject, due_date from organizer.tasks");
    assertTrue(results.next()); // There should be exactly one row.
    assertEquals("Test Subject", results.getString(1));
    assertEquals(new java.sql.Date(rightNow.getTime()), results.getDate(2));
    assertFalse(results.next()); // There shouldn't be more than one row.
}
```

Other Tricks

- Override how your servlet/enterprise bean obtains the JNDI/servlet/bean context
- Consider existing testing frameworks
 - HttpUnit
- Eclipse test framework
- Remember what you're testing!

Summary

- Remember what you need to test
- Remember what you don't need to test
- Use stuff before you build it
 - Xs are your friends
- ctrl-1 and ctrl-space are your other friends
- Isolate layers