JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2002**

Review of JBuilder 6 Enterprise Version

Dave Neuendorf and Richard Wiener

JBuilder 6 Enterprise Version from Borland is a powerful Java software development platform that provides an integrated development environment for GUI development, servers and tools that support J2EE web development, tools for unit testing, re-factoring and UML visualization. We have examined most of these features during our review.

J2EE DEVELOPMENT

JBuilder offers lots of support for J2EE development. The Enterprise version comes bundled with developer licenses for the Borland Enterprise Server (an EJB application server), Borland InterBase (a relational database) and Borland DataStore (a Java object-relational database).

For servlet and jsp development it has integrated Tomcat 3.2 and 4.0. It also comes with InternetBeans Express (a collection of components and corresponding tag library) and integrated support for the Cocoon jsp framework.

To handle the project housekeeping of all this, JBuilder provides wizards that can create servlets, jsp's and EJB's. It also has a tool called the EJB Designer, which allows visual class design for the various types of enterprise beans. During the design process, the tool keeps boilerplate code and deployment descriptors in sync. For the latter, JBuilder provides a deployment descriptor editor.

For databases, JBuilder's DataExpress is a set of classes that can be used to create a proxy between a database and an application. It provides caching and other services for database clients, without tying a client to a particular database. The data-aware Swing components that come with JBuilder also use these classes. Tools are provided for visually building the necessary connections.

A wonderfully useful tool for database development is JBuilder's DatabasePilot. Using DatabasePilot made it a pleasure to create and administer relational databases and their tables.

Related technologies for which JBuilder provides significant help include JMS (Java Message Service), RMI (Remote Method Invocation), JNDI (Java Naming and Directory Interface) and XML. Support for JMS includes creation of EJB 2.0 message-driven beans, a bundled JMS implementation called SonicMQ, and a JMS wizard.

The documentation provides tutorials with parallel architecture discussion chapters for most of these tools. Most of the tutorials work as expected, though we had trouble adding a database to a JBuilder data module when we followed the tutorial instructions. Where there are not tutorials, there are sample projects to study.

There are tutorials for jsp and servlet development, including accessing databases directly. Unfortunately, we could not find one that showed the steps of efficiently developing a web application that used both jsp's and EJB's. The best we could do is to copy one of the sample applications and modify it.

The InternetBeans Express bundle is Borland's attempt to make jsp development cleaner and easier. The components can work with the DataExpress classes to hook up the jsp's to databases. Integrated Cocoon support is another approach to separating presentation from logic. These features are commendable, but we believe it would have been even more useful if Borland had integrated the Apache Struts framework into JBuilder. Struts cries out for a tool to manage development, and this would have been a great opportunity for Borland to provide that tool.

JBuilder has built-in support for the WebLogic, WebSphere and iPlanet EJB application servers, in addition to their own Borland Enterprise Server. It is also possible to develop for a generic application server, but specific support for popular open-source servers like JBoss would have been even more helpful.

Tools are provided for including non-supported databases in projects. To test these tools, we set up JBuilder to use MySQL, an open source RDBMS. This was a simple process that took only a few minutes. It involved pointing JBuilder to the jdbc driver and making it available in projects. It was equally simple to set up to work with MySQL in DatabasePilot. Once this setup was complete, we were able to use MySQL as the database for the various tutorials.

UNIT TESTING

JBuilder 6 Enterprise provides support for the *JUnit* framework, a compelling approach to testing developed as part of the Extreme Programming paradigm. A set of five wizards are available that provide the framework for constructing unit tests of each class that is constructed in an application. The construction of a test harness is encouraged even before the class that is to be tested is built.

Using two of the wizards, the following skeletal framework is produced by JBuilder:

```
import junit.framework.*;
public class FractionTest extends TestCase {
  public FractionTest(String s) {
    super(s);
  }

  protected void setUp() {
  }

  protected void tearDown() {
  }

}


import junit.framework.*;

public class AllTests extends TestCase {

    public AllTests (String s) {
        super(s);
    }

    public static Test suite () {
        TestSuite suite = new TestSuite();
        suite.addTestSuite(FractionTest.class);
        return suite;
    }
}
```

Much of the tedious mechanics and overhead of proceeding with unit testing is automated by the framework integrated into JBuilder. A *Fraction* class was developed as part of the review of the JUnit testing. A small portion of the process is reproduced to demonstrate the power of the JBuilder JUnit testing framework.

The class *Fraction* for which the test harness is being developed is shown after only the earliest stage of development.

```
public class Fraction  {
    // Fields
    private int numerator, denominator = 1;

    // Constructor
    public Fraction (int numerator) {
        this.numerator = numerator;
    }

    public Fraction (int numerator, int denominator) {
```

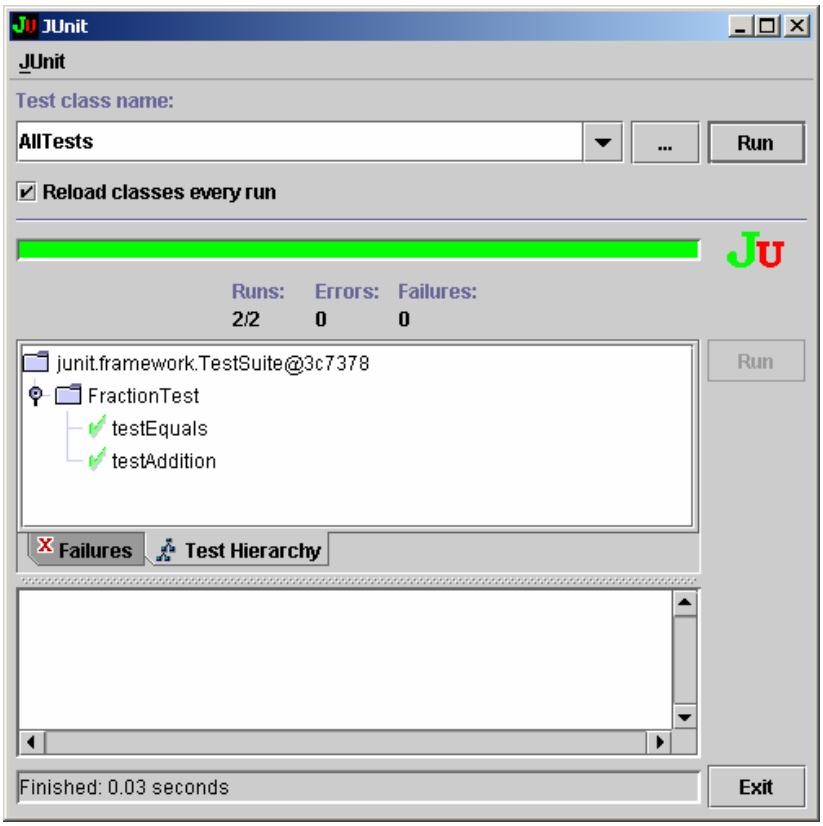
```
        this.numerator = numerator;
        this.denominator = denominator;
    }

    // Queries
    public Fraction add (Fraction f) {
        Fraction result =
            new Fraction(numerator * f.denominator +
                         f.numerator  *  denominator,  denominator  *
                         f.denominator);
        result.simplify();
        return result;
    }

    public boolean equals (Object obj) {
        if (obj instanceof Fraction) {
            Fraction f = (Fraction) obj;
            return  numerator  *  f.denominator  ==  denominator  *
                   f.numerator;
        }
        return false;
    }

    private void simplify () {
        // Details not shown
    }
}
```

Two quick tests are embedded in the *FractionTest* class as shown:

```
import junit.framework.*;

public class FractionTest extends TestCase {

    // Fields
    private Fraction f1;
    private Fraction f2;

    public FractionTest (String s) {
        super(s);
    }

    public void testEquals () {
        Assert.assertEquals(f1, f1);
        Assert.assertTrue(!f1.equals(null));
           // Additional tests not shown
    }

    public void testAddition () {
        Fraction expectedResult = new Fraction(17, 12);
        Assert.assertTrue(expectedResult.equals(f1.add(f2)));
    }
```

```
        protected void setUp () {
            f1 = new Fraction(2, 3);
            f2 = new Fraction(3, 4);
        }

        protected void tearDown () {
        }
    }
```

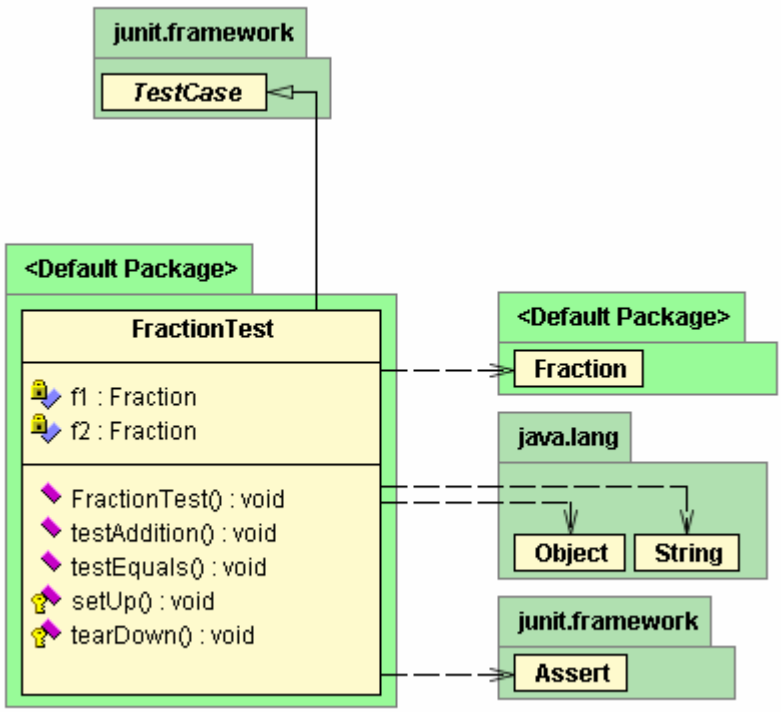
When the application is run, the following GUI appears with a green light indicating that the two tests have passed. In the event of a test failure, the precise location of the failure is displayed in the error pane window.



As more functionality is added to the *Fraction* class, the test harness is expanded. As changes are made over time to class *Fraction*, each change can be quickly tested using the test harness.

UML VISULIZATION

JBuilder allows a suite of classes within a project to be viewed through a UML class diagram. This quick reverse engineering feature is quite useful in obtaining an overview of a system under development. The UML overview for the JUnit testing system discussed above is shown below.



1 REFACTORING

Martin Fowler's book, *Refactoring*, has focused the attention of developers on the need for systematic, incremental improvement of existing code. Fowler names and describes a multitude of refactorings for this purpose.

JBuilder 6 Enterprise makes a start at providing tool support for such refactoring. The product has tools for renaming packages, classes, methods, fields, local variables and properties; and moving a class or interface from one package to another. Changing the package structure, or the names of program symbols, is a highly error-prone activity when performed using an editor's search/replace tools. We have been involved in projects where changing the package structure would have involved so much effort, and introduced so many bugs, that project management decided against doing it.

JBuilder's refactoring tools can take most of the pain and potential for error out of renaming symbols or moving classes between packages. They intelligently rename the

program symbols wherever they occur: at the point of declaration, and at every reference. For methods, it is even possible to automatically add a forwarding method to handle the situation where external callers don't know about the renaming.

SUPPORT

Borland provides 30 days of free installation support for JBuilder. All other support is sold through various incident package plans. JBuilder is a complex product, and it is likely that users will require some support beyond the manuals and on-line help. Buyers should factor that support into the cost of owning the product.

CONCLUSIONS

Based on our extensive examination of JBuilder 6, we believe that Borland has lived up to its reputation in putting together a useful and important integrated development environment and we highly recommend this product.