

Exercise: JUnit Framework

Exercise Overview

The JUnit framework is an open source and industry proven Java unit-testing component. This framework can be integrated into the WSAD environment as an additional plug-in and used to test our applications. In this exercise, we will lead students through the installation process, the creation of a simple test case, its execution and examine the JUnit view .

This exercise is designed to reinforce the concepts of the JUnit framework application issues discussed in the student notebook.

Exercise Duration

This exercise will require approximately 30 minutes.

Exercise Objectives

Upon successful completion of this lab exercise, you should be able to:

- Understand the role of JUnit
- Illustrate the installation of JUnit
- Create a JUnit test case
- Executing test cases

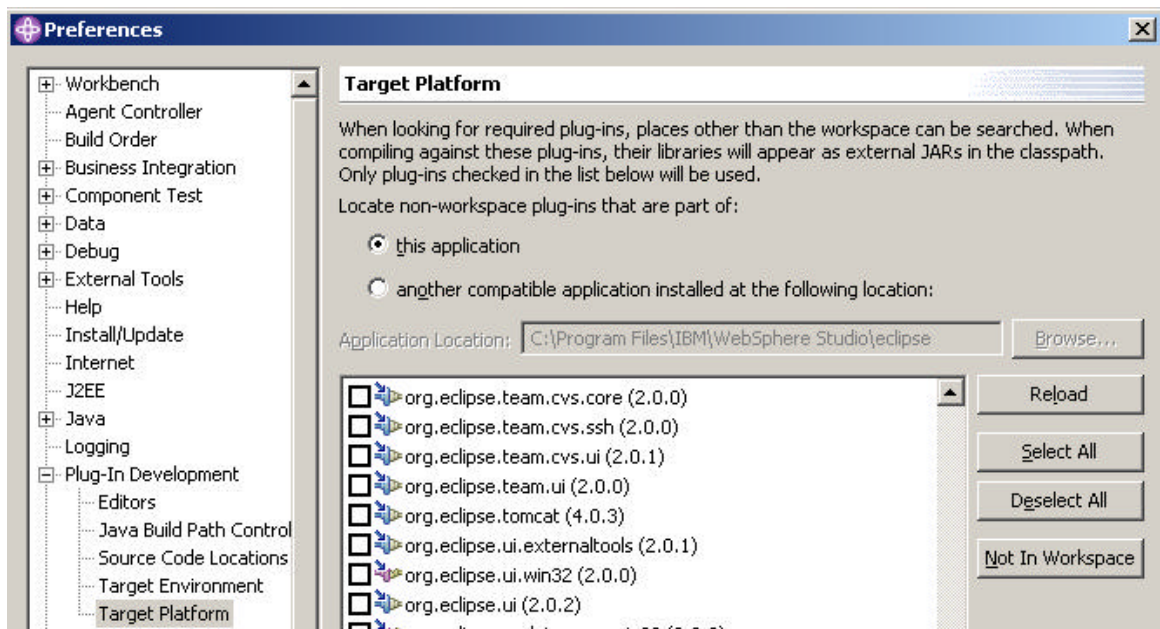
Exercise: JUnit Framework

Exercise Instructions

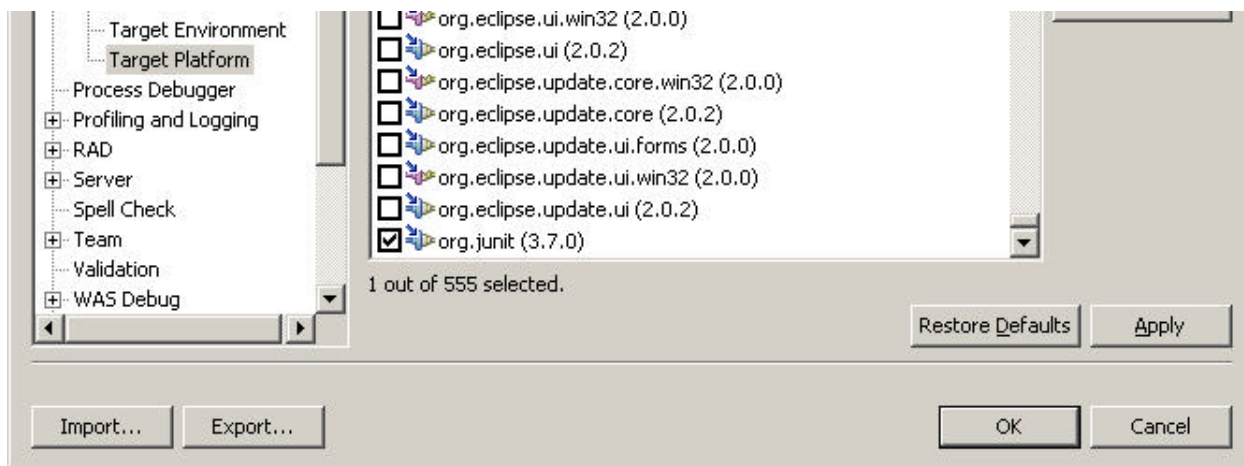
Step 1: Integrate JUnit with WSAD

Before we can utilize the capabilities of the JUnit framework, we must first implement the plug-in in the WSAD environment.

- 1) Open the Preferences panel by selecting Windows → Preferences from the main menu. Expand the Plug-in Development folder:

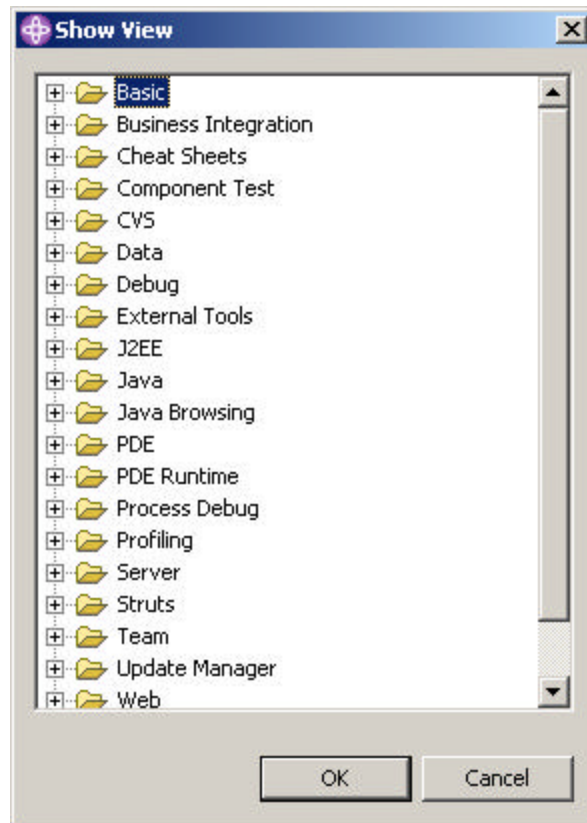


- 2) Click on the **Target Platform** entry and the panel will display all of the pluggable components that can be implemented in our WSAD environment. Scroll down to the last entry in the list:

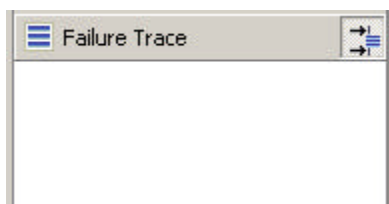


3) Select the `org.junit` entry, click the **Apply** button and then click the **Reload** button. To complete the installation you will need to exit and restart the WSAD tool.

4) After restarting WSAD, open the Java perspective via **Window → Open Perspective → Java**. Now we need to open the JUnit view. Go to **Window → Show View → Other**, the following will be displayed:



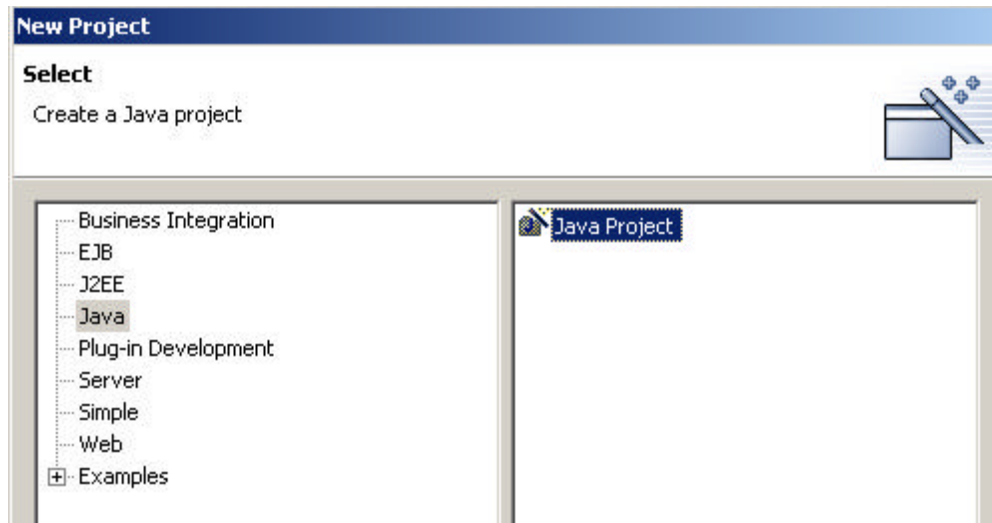
5) Expand the Java folder and select the JUnit view, then click the OK button. The Java perspective will be updated with these panels:



Step 2: Create a Test Case

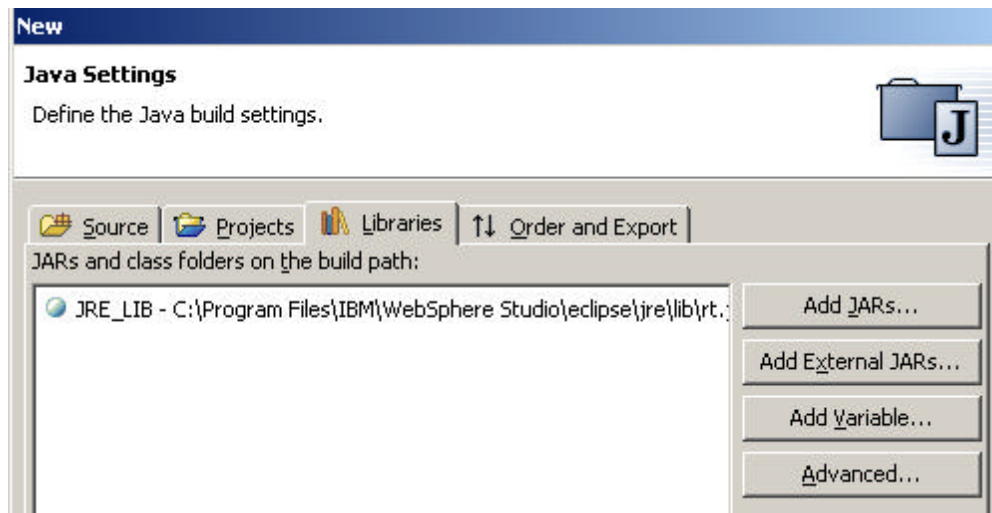
Now that JUnit has been installed, we can build a project, implement some test cases and test them in the WSAD tool.

- 1) First, return to the Java perspective. Create a new Java project by selecting **File** → **New** → **Project** and selecting a Java Project:



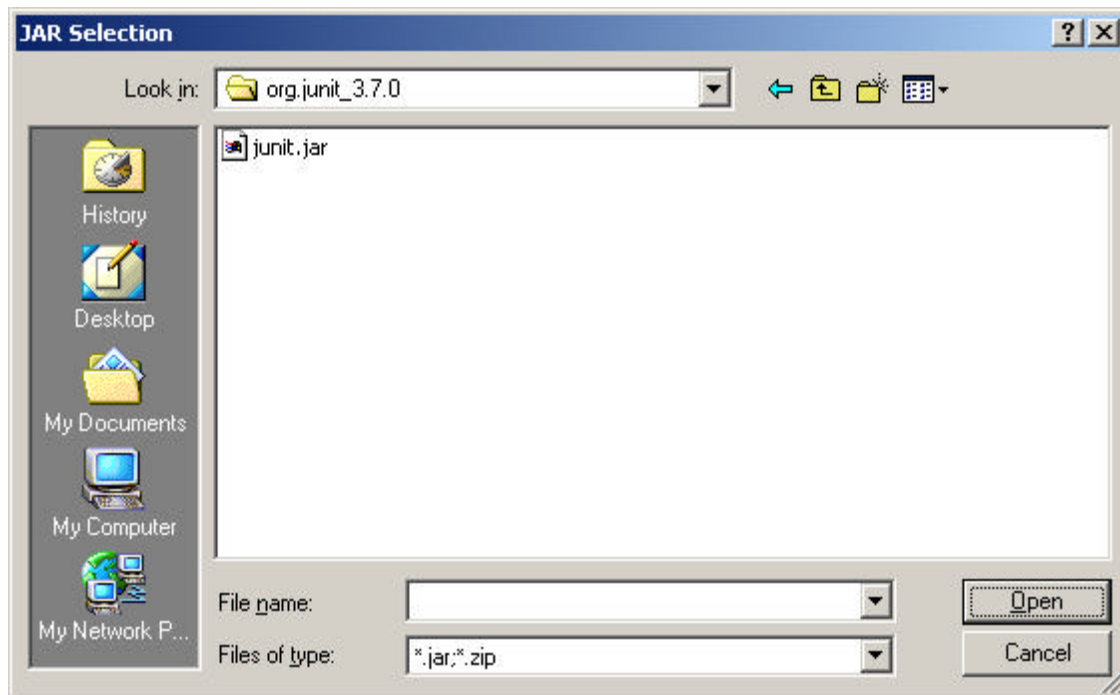
Click **Next** button.

- 2) Set the project name to SampleJavaProject and click the **Next** button. Then select the **Libraries** tab:



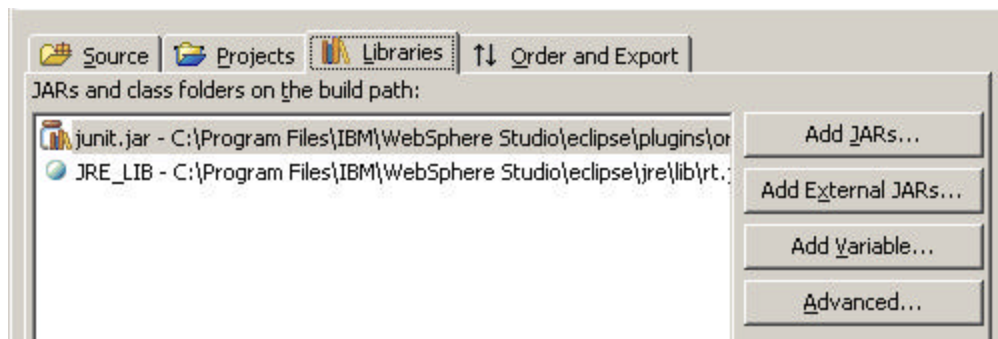
- 3) We will need to add the JAR files associated with JUnit to our sample Java project. Click on the **Add External JARs** button.

4) Navigate to the following path that will contain the necessary JAR files for our JUnit implementation: `<%Install_Path%/eclipse/plugins/org.junit_3.7.0`. You should locate the necessary JAR files:



Click **OK** button.

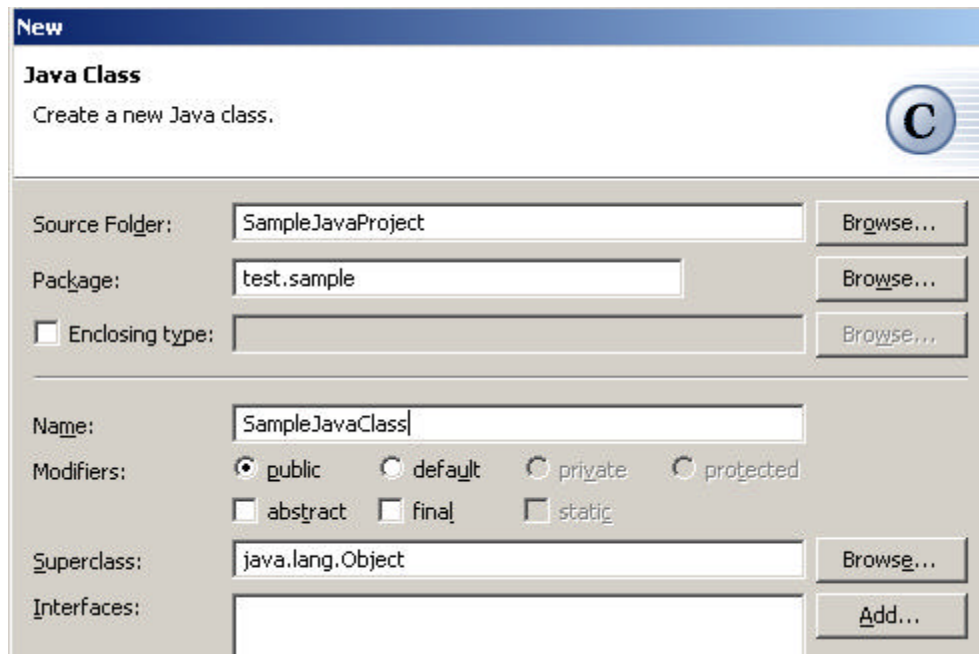
5) Our updated JAR file listing on the Libraries tab should now show the **junit.jar** file entry as shown below:



Click the **Finish** button.

6) Now we need to create a new Java class for our test case. In the Java perspective, click on the **Package Explorer** tab and highlight the **SampleJavaProject** project. We will need to add a new package for our project, right click the mouse and select **New → Package** from the menu. Enter `test.sample` as the packagename and click the **Finish** button.

7) To create our new sample class, highlight the `test.sample` package, right click the mouse and select **New → Class** from the menu. Enter **SampleJavaClass** as the Classname.



The screenshot shows the 'New Java Class' dialog box. The 'Source Folder' is 'SampleJavaProject', 'Package' is 'test.sample', and 'Name' is 'SampleJavaClass'. The 'Modifiers' section has 'public' selected. The 'Superclass' is 'java.lang.Object'. There are 'Browse...', 'Add...', and 'Finish' buttons.

Click **Finish** button.

8) The generated code for the **SampleJavaClass** should now be displayed in the Java Editor.

Step 3: Build Test Case Code

We have constructed the class for our sample test case. Now in this section we will add the code that we wish to test.

1) Return to the Java editor that contains the **SampleJavaClass**.

2) Add the following code to the **SampleJavaClass** for a new method called `add` that will simply add two different numbers together:

```
public int add(int a, int b) {  
    return a + b;  
}
```

```
}
```

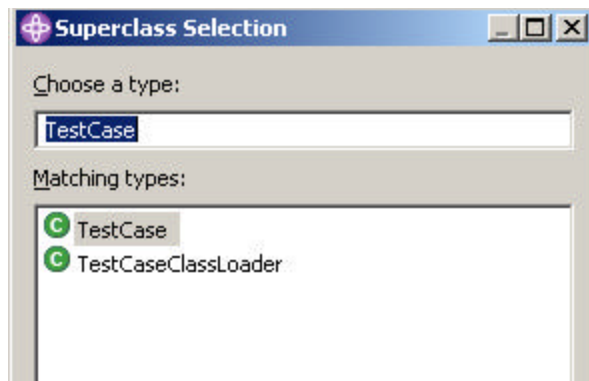
3) Save your changes with Ctrl-S.

Step 4: Build Sample Test Case

We now need to construct a Java class that we can use as a Factory class that will build an instance of our `JavaSampleClass` and invoke the `add` method.

1) Create a new package in our project called `test.testcase`.

2) Then, create your new sample test class, highlight the `test.testcase` package, right click the mouse and select **New** → **Class** from the menu. Enter `SampleJavaTestCase` as the Classname and select the option to build our default constructor. Click on the **Browse** button for the Superclass and select `TestCase` from the `junit.framework` package:



Click **OK** button. Click the **Finish** button.

3) Add the following import statements to the `SampleJavaTestCase` class:

```
import test.sample.SampleJavaClass;
```

4) Add the following method to test the `add` method in the `SampleJavaClass`:

```
/**
 * Test the add method in the SampleJavaClass
 */
public void testAdd() {
    SampleJavaClass sample = new SampleJavaClass();
    int c = sample.add(5,10);
    assertEquals(c,15);
}
```

5) Add the following method to test the `add` method again:


```
/**
 * Test the add method in the SampleJavaClass
 * This test supplies an invalid value in the assert statement
 */
public void testAddAgain( ) {

    SampleJavaClass sample = new SampleJavaClass();
    int c = sample.add(5,10);
    assertEquals(c,20);
}
```

Notice the use of the `assertEquals` method. This statement is used to verify the results returned from the method with the expected output. If the actual output and the expected output do not match, then an entry will appear under the failures area of JUnit.

6) Finally, we need to add the following main method to our test class that will utilize JUnit to execute our test case:

```
public static void main(String [] args) {

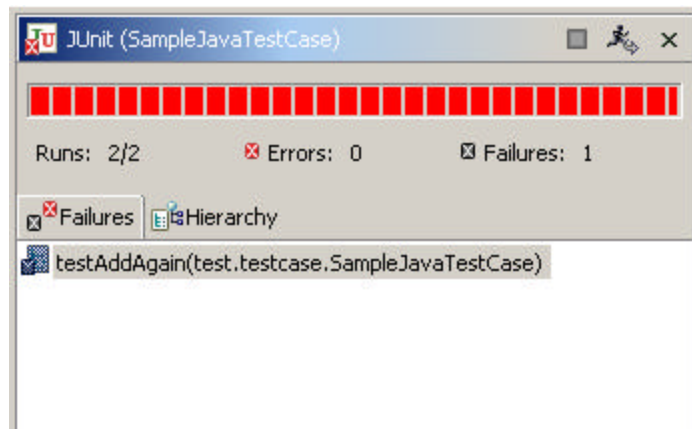
    junit.textui.TestRunner.run(new SampleJavaTestCase(""));
}
```

7) Save your changes using Ctrl-S.

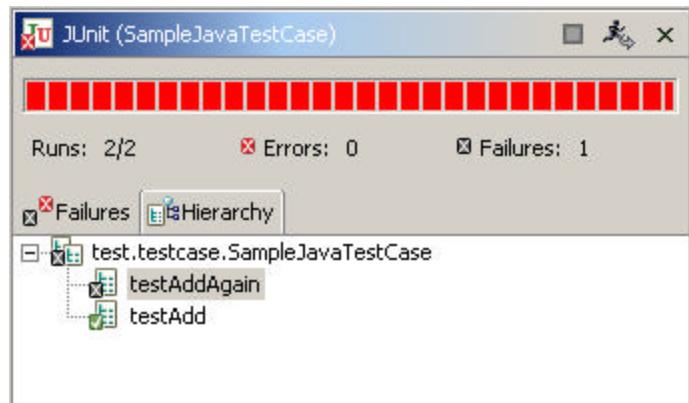
Step 5: Execute Sample Test Case

With both our sample application and our test case successfully created and compiled, we are ready to begin the process of testing our components using JUnit.

1) Highlight the `SampleJavaTestCase` class and from the toolbar select **Run** → **Runas** → **JUnit tests**. The JUnit view will appear and the Failure tab will show:

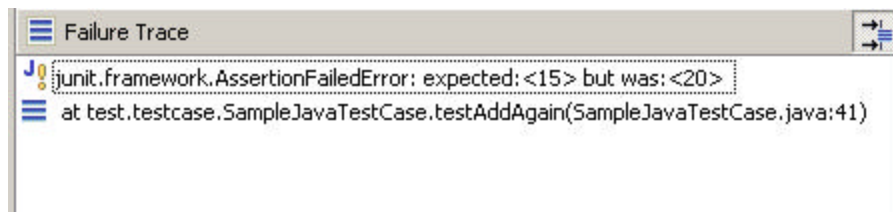


2) Click on the **Hierarchy** tab:



We had two different test cases. A green check mark (like on `testAdd`) denotes a successful test and an "X" (like for `testAddAgain`) will depict an unsuccessful test.

3) At the bottom of the JUnit view is the Failure Trace area. This will show us exactly where a failure has occurred in our application. Presently, it appears as:



4) The upper right-hand corner of the panel contains a single icon for filtering our output (the print stack trace information). Presently, the filter is turned on, click on it to turn it off:

