Test Java components

This tutorial will teach you about the process of unit testing Java components.

Time Required

To try out the tutorial using your own Java components, you will need approximately **30 minutes**. If you decide to explore other facets of component testing while working on the tutorial, it could take longer to finish.

Prerequisites

In order to complete this tutorial end to end, you should be familiar with the following:

- Basic principles of Java software development
- How to use the perspectives and views of IBM Rational Application Developer
- The JUnit testing framework

You will also need some Java components to test.

Learning Objectives

This tutorial is divided into several sections that should be followed in sequence. You will learn how to:

- Create a test project
- Create a Java component test
- Define test data
- Edit the Java code
- Create stubs
- Add stub data
- Run the test and analyze the test results

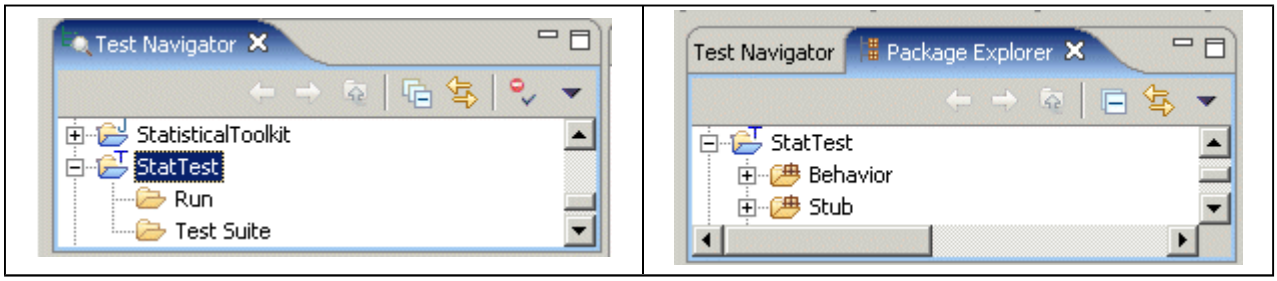
When you are ready, begin Exercise 1.1: Creating a test project.

Exercise 1.1: Creating a test project

Component test projects serve as a grouping mechanism for the test artifacts that you create using the automated component testing features. The test project is linked to one or several Java projects that contain the components you want to test.

1. From any perspective, click **File > New > Project > Component Test > Component Test Project** and click **Next**.
2. In the Create a Component Test Project wizard page, supply a name for the project and click **Next** to accept the default storage location or uncheck the **Use default** check box, specify a desired location, and click **Next**.
3. In the Define the Scope of Component Test Project page, select the components that can be used to create tests or stubs within this project and click **Finish**.

Test projects contain both execution-oriented artifacts (test runs and test suites) and code-oriented artifacts (test behavior scripts and stubs). The execution-oriented artifacts are viewed in the Test Navigator, while the code-oriented artifacts are viewed in the Package Explorer.



Before moving on to the next exercise, lets make sure you understand what these artifacts are.

A **run** is an artifact that gets created when you run a test. A run is the consolidated results of one test execution and may incorporate several test suites, test cases, and individual tests.

A **test suite** is a grouping mechanism that is used to organize the artifacts that get created when you create test projects and tests. A test suite contains test cases, stubs, and test deployment data.

A **behavior** is the actual Java source code that is created at the time of test generation, whereas a **stub** is a class that is used as a replacement for a class that interacts with the class under test.

To simplify your work with component testing, you can add the Package Explorer view to the Test perspective. To do this, go to the Test perspective; then click **Window > Show View > Other > Java > Package Explorer**. Now you are ready to begin Exercise 1.2: Creating a Java component test.

Exercise 1.2: Creating a Java component test

Before you begin, you must complete Exercise 1.1: Creating a test project..

After you create a test project, you can use the Create Java Component Test wizard to create Java component tests and stubs. If you need to test a Java interface, abstract class, or superclass, you can also create a special kind of component test called an abstract test.

In this example, the wizard takes you through the following steps:

1. Starting the wizard and selecting a test project
2. Using the test guidance to decide what to test
3. Selecting a test pattern
4. Defining the test scenario

Starting the wizard

To start the wizard:

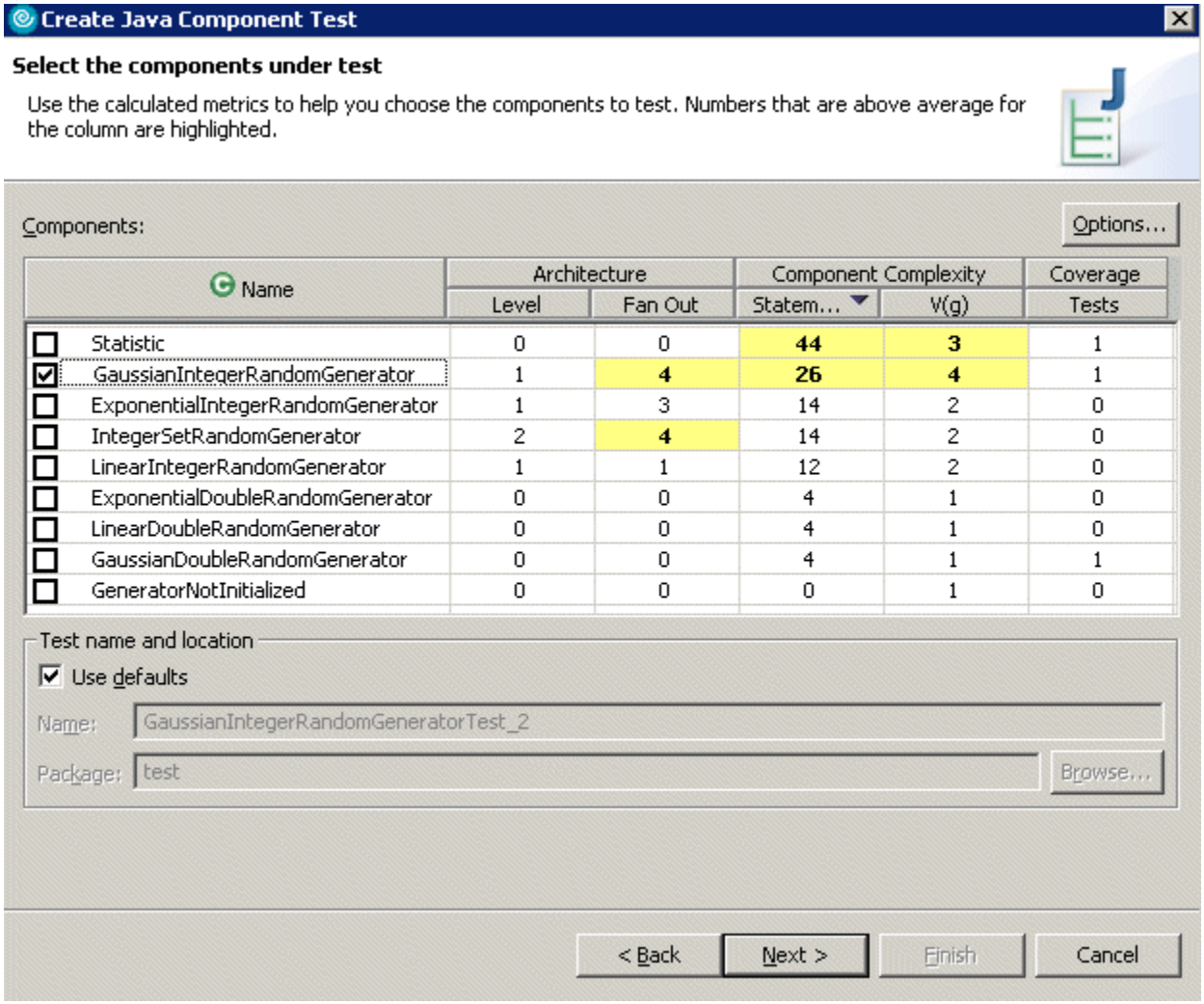
1. Click **File > New > Other > Component Test > Java > Java Component Test**.
2. Select the test project that will contain the test or click **New**to create a new project and click **Next**.

Using test guidance to help you decide what to test

After you select a test project, a static analysis is performed on the Java source files associated with the test project. These files were selected during the creation of the test project and serve to define the scope of the test. The list of files in the project can be updated by modifying the test project's Test Scope properties.

When the analysis is complete, you will see a list of components in a table format and sorted according to the computed metrics. You can use the guidance that these metrics provide to help you decide which classes or components are most important for you to test. Components with highlighted values or high numerical values are considered high-priority test candidates.

In the following figure, for example, GaussianIntegerRandomGenerator would be one place to start, so to test this class, simply click the check box next to GaussianIntegerRandomGenerator and click **Next**. (You can also click **Options** to modify which metrics are displayed and to change the sort order.)



Selecting a test pattern

After you select the classes you are going to test, you need to select a test pattern. Test patterns provide a sort of template for different kinds of Java component tests. The available test patterns for Java components include:

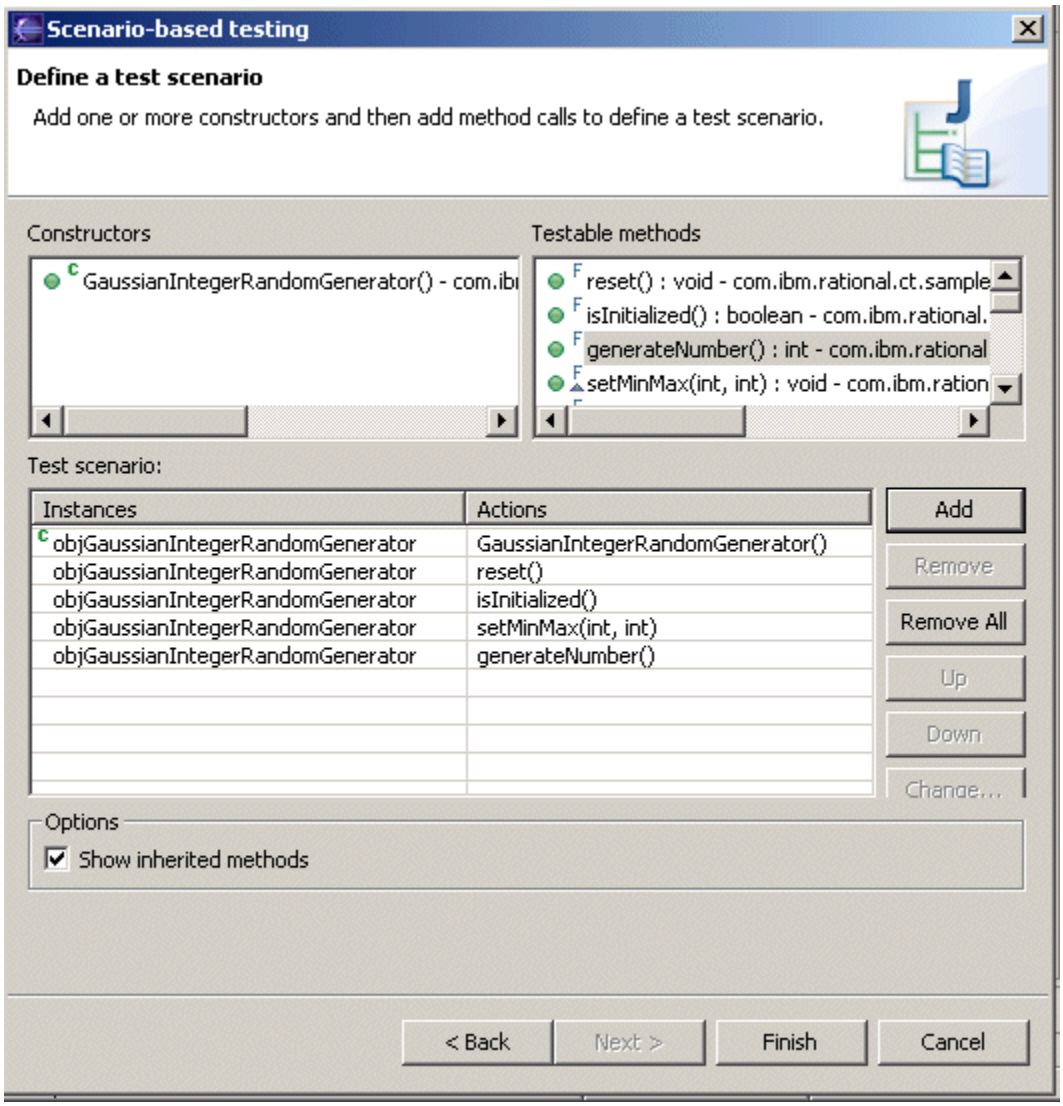
- Method-level pattern. Use this test pattern to test one or more individual Java methods.
- Scenario-based pattern. Use this test pattern to create a test scenario of methods and constructors from one or several classes. (This is the pattern used in this exercise.)
- Abstract test pattern. Use this test pattern to test a Java interface, an abstract class, or a superclass.

For your first test, select the scenario-based test pattern and click **Next**.

Defining the test scenario

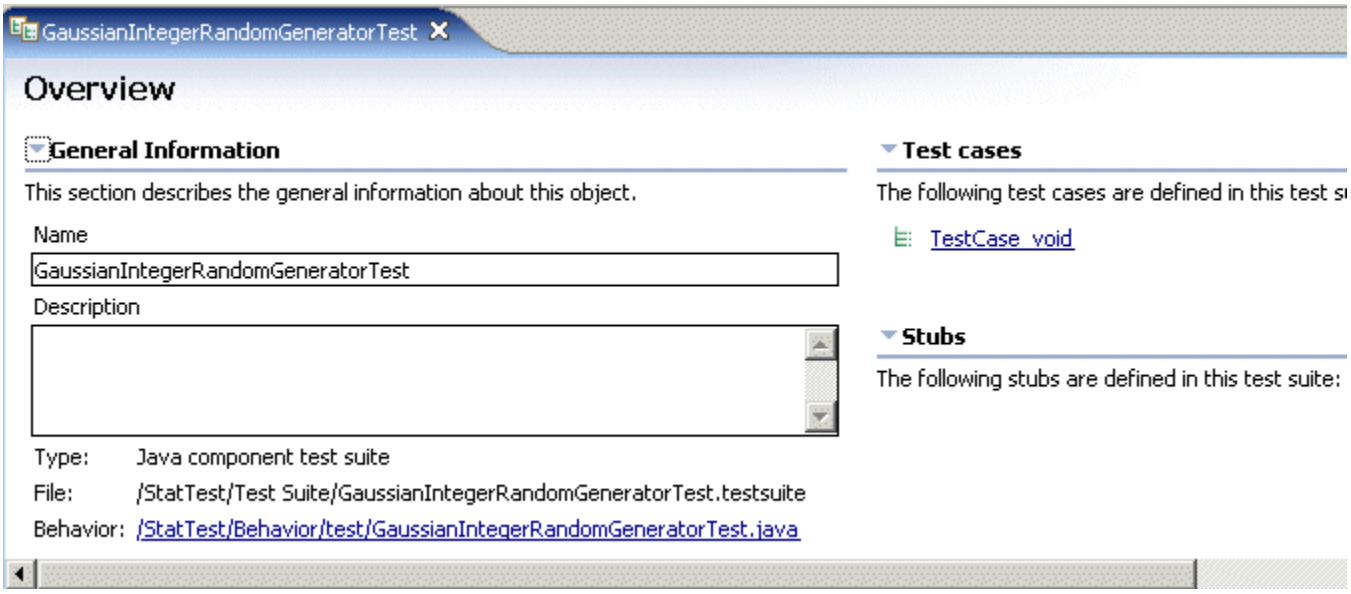
After you select the test pattern, you will see a wizard page that lets you define the test scenario. The process is as follows:

1. Add an instance of the class-under-test by selecting a constructor from the list and clicking **Add**. (You can also double-click a constructor to add it to the scenario.)
2. If you want to assign a more meaningful name to the class instance, select the instance from the test scenario, click **Rename**, enter the new name, and click **OK**.
3. Click **Show inherited methods** to display all of the methods that can be included in the scenario.
4. Double-click each method to be included in the scenario. (A sample scenario is shown below. The scenario has one constructor and four methods.)
5. When you are finished building the scenario, click **Finish**.

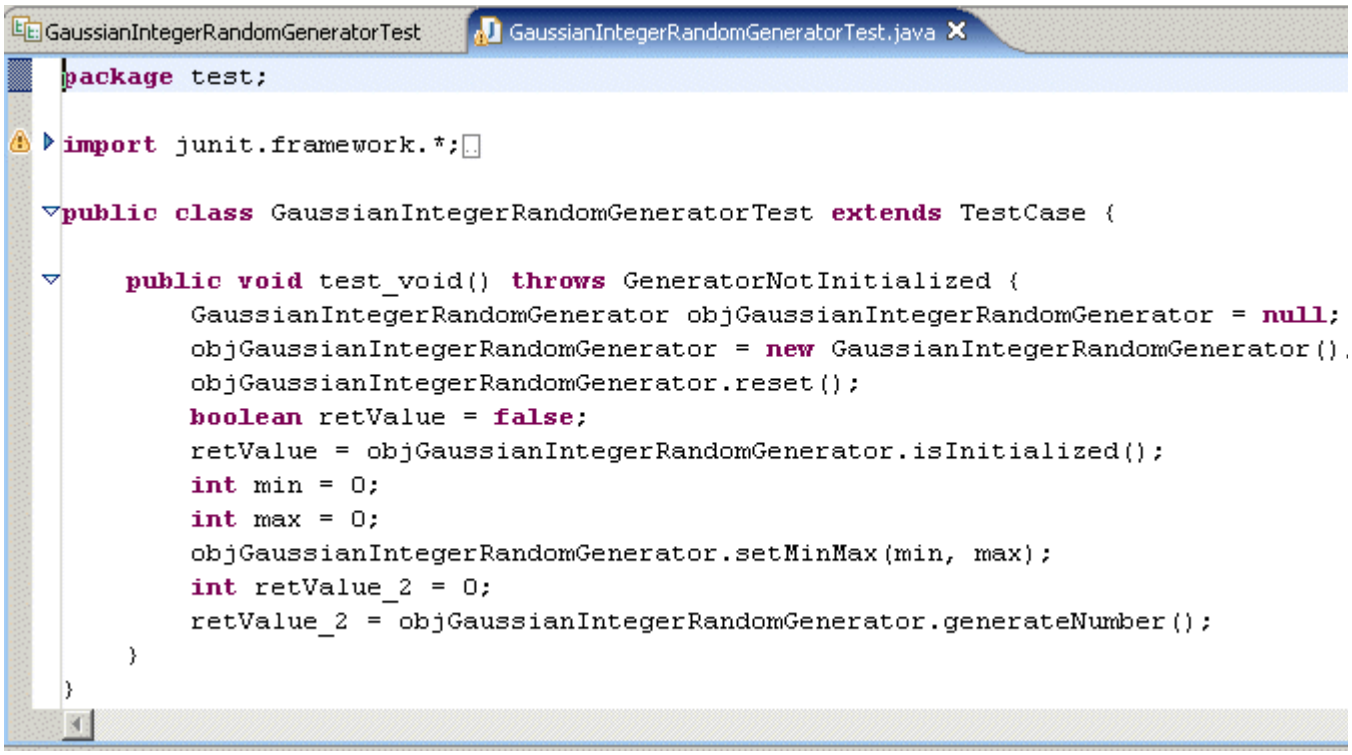


Viewing the results

When you create a scenario-based test, a single test case is created within the test suite. In the test behavior code, the test case is implemented as a single JUnit test method. After you finish creating the test, the Overview page of the Test Suite editor is displayed automatically, as shown below:



From this page, you can edit the name of the test, add a description of the test, and open the test behavior code in the Java editor. To view the code, click the Behavior (/StatTest/Behavior/test/GaussianIntegerRandomGeneratorTest.java in the example above).



Now you are ready to begin Exercise 1.3: Editing the test.

Exercise 1.3: Editing the test

Before you begin, you must complete Exercise 1.2: Creating a Java Component Test

After creating a test, you can edit it to make it do exactly what you want. For example, you can:

- Use test data tables to define test data specific to your test
- Use the Java editor to edit the test behavior code
- Create stubs for classes that the code you are testing interacts with

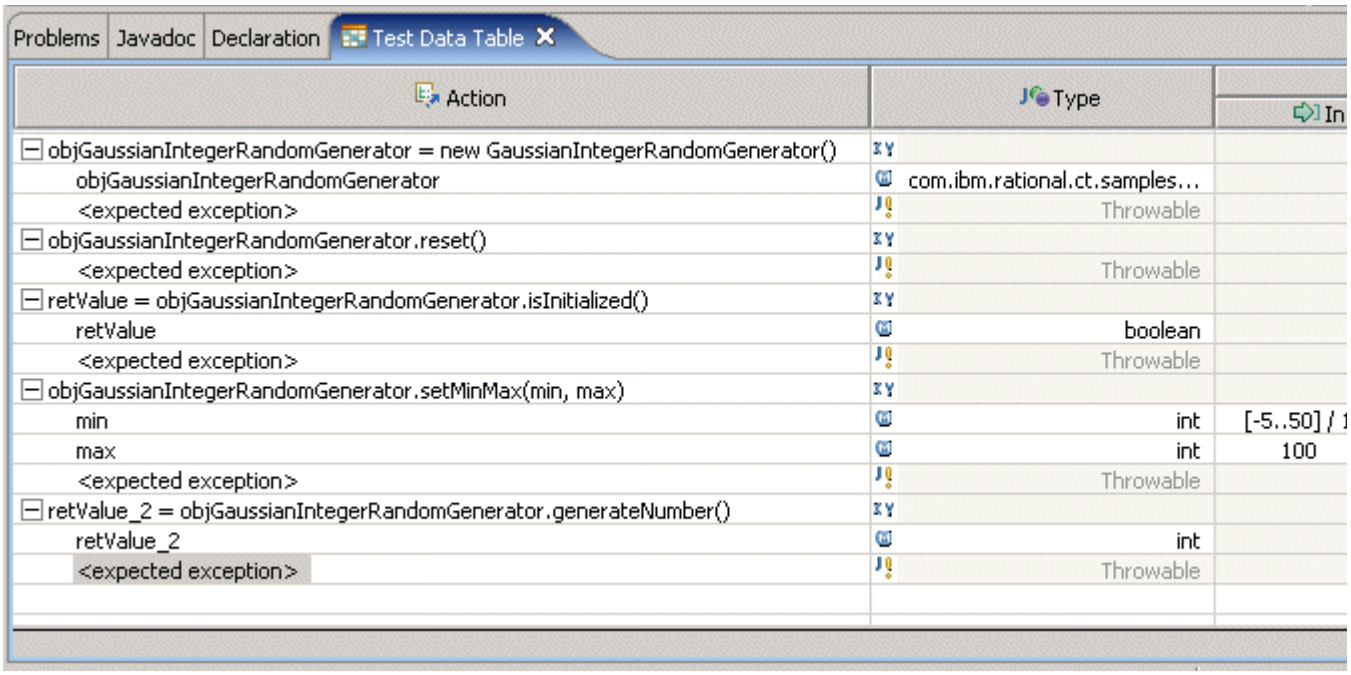
Defining test data

In addition to the test behavior code, test suite, and one or several test cases, a test contains test data in the form of one or more test data tables. When you create a test, one test data table is created for each test case. You can use the test data tables to perform data-driven testing.

Each row in a test data table represents an object or expression in your code, and each column pair represents a data set (or equivalence class, as it is sometimes called). Each data set column is divided in half, one half for the input values, and the other half for the expected output values. Expected outputs include return values, exceptions, and any parameters that are specifically marked for output. Return values and exceptions are mutually exclusive.

Any syntactically valid expression that can appear on the right side of an assignment statement can be entered in a test data table cell. Thus, all of the following are considered to be valid: primitive values, strings (enclosed in double quotes), variable references, and constructor and method calls. Expressions can also include variables, constants, and logical operators.

The following sample test data table shows a cell with a range of values, another cell with a single integer value, and another cell with an expected exception.



To view a test data table:

1. Open the test behavior code that is generated when you create a test.
2. Click any test method.

Here is a short list of some of the actions you can perform with a test data table:

- Define a range of values. To get started, right-click in any table cell that can accept a numerical data type and click **Define Range**.
- Define a set of values. To get started, right-click in any table cell that can accept a set of values and click **Define Set**.
- Insert a new data set. Right-click in the column header of a test data table and click **Insert Data Set**.
- Supply test data for the attributes of object data types
- Define the elements in an array

Editing the test behavior code

In addition to editing the code by hand, you can simply right-click in the Java editor and click **Component Test** to display a menu of elements you can add to the code. The test behavior code is synchronized with the test data tables, so modifications that you make to the code are reflected in the test data table and vice versa.

Creating stubs and stub data

For testing purposes, you might find it useful to stub out classes that the code you are testing interacts with. Stubs are typically used for the following purposes:

- To isolate the testing of the component-under-test (CUT) from other classes or components that the CUT interacts with
- To implement unavailable classes or components that the CUT needs to interact with
- To verify the interactions of the CUT with external components
- To verify the behavior of the CUT when one of its service providers generates exceptions or non-typical values

When you create a stub, you can reuse it in multiple tests.

To create a stub, you use a wizard similar to the one you use when you create a test. To stub a class:

1. Click **File > New > Other > Component Test > Java > Java Component Stub**.
2. Select the test project that will contain the stub and click **Next** or click **New**to create a new project and click **Next**.
3. Select the Java source files or libraries that you wish to stub and click **Finish**. This displays the Test Suite editor.
4. In the Test Suite editor, click the **Stubs** tab.
5. Click **Add**, select the stub you want to add to the test suite, and click **Finish**.
6. Still in the Test Suite editor, click the **Overview** tab and under **Stubs**, click the stub that is defined in the test suite. This displays the Stub page of the Test Suite editor.
7. Under General Information, click **Edit** next to the name of the stub's Java source file. This displays the stub user code.
8. Click on a method in the stub code to display the stub data table.
9. Enter your stub data.

Note that stub data tables work somewhat differently than test data tables. With the stub data table, you simulate the stubbed class by specifying the actual input and return values for each stubbed method.

To replace the stub with the real class at any time, simply remove the stub from the test suite, using the Test Suite editor.

Now you are ready to begin Exercise 1.4: Running the test and analyzing the result.

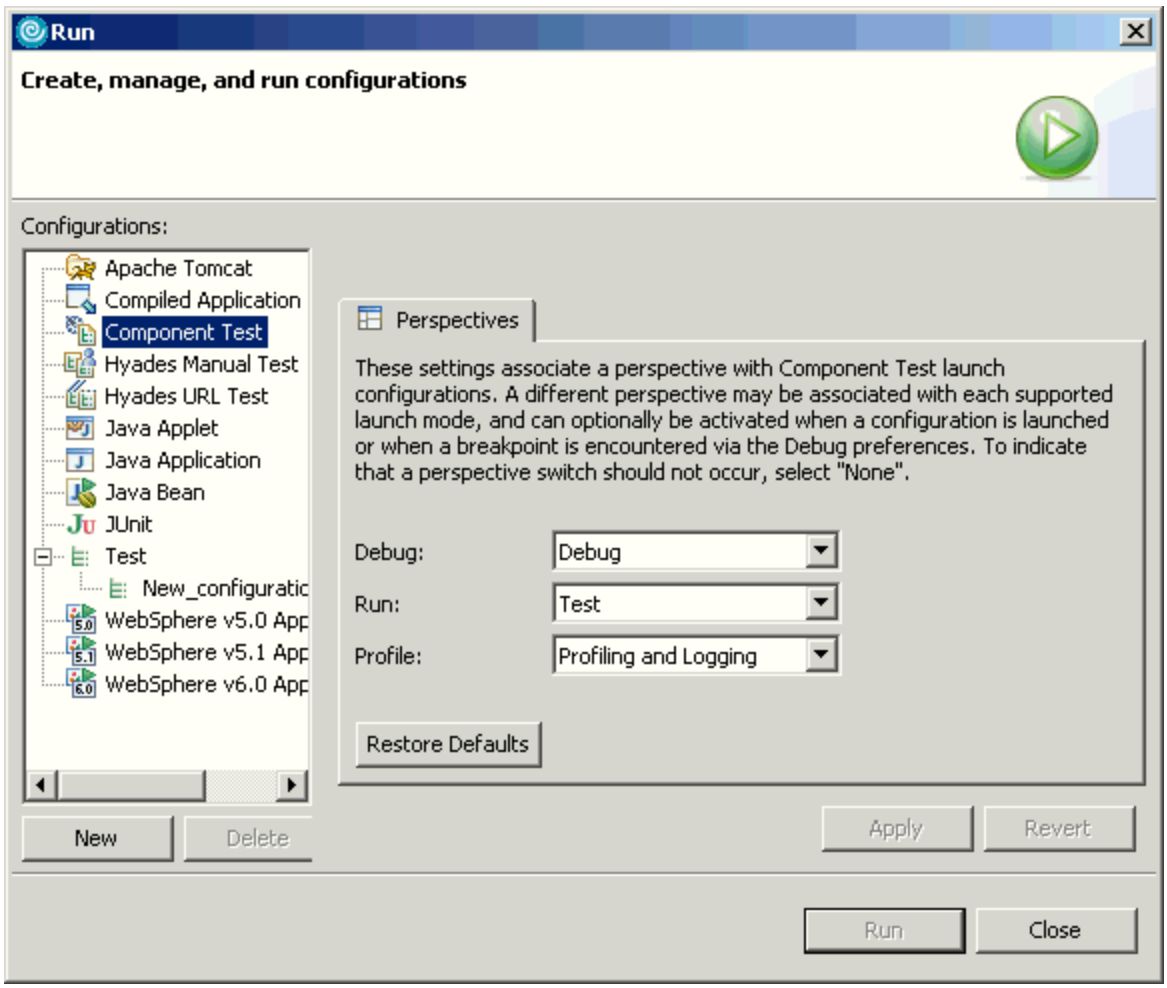
Exercise 1.4: Running the test and analyzing the results

Before you begin, you must complete Exercise 1.3: Editing the test.

After you edit your test, you can run it and view the test results. A test runs with the data you have supplied for it in a test data table. If you supply sets or ranges of values in the test data table, running a single test results in the running of many individual tests. For example, if you run a test for a method that has two arguments, and you supply five values for arg1 and six values for arg2, running the test results in 30 individual tests.

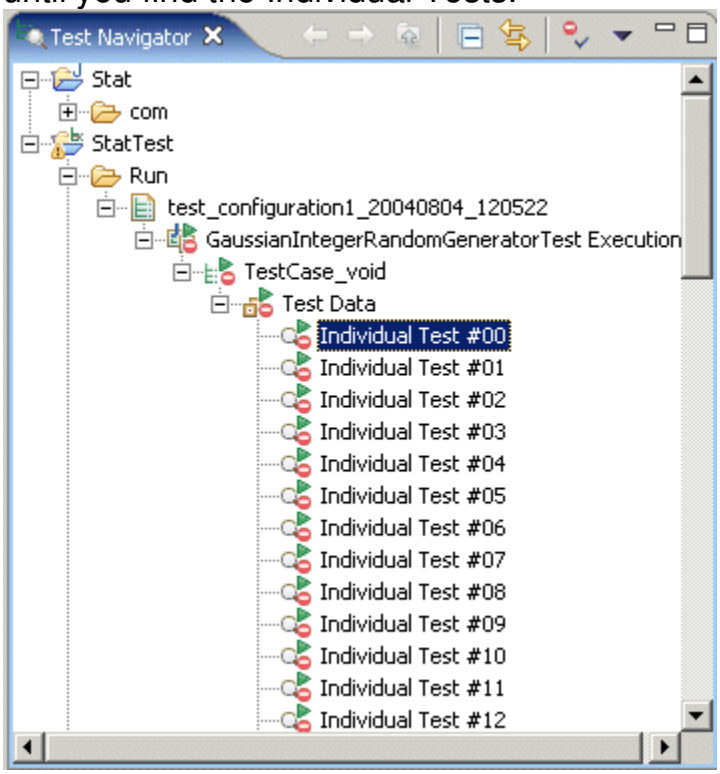
To run the test:

1. Right-click a test suite you wish to run in the Test Navigator and click **Run > Run** to start the Run wizard.

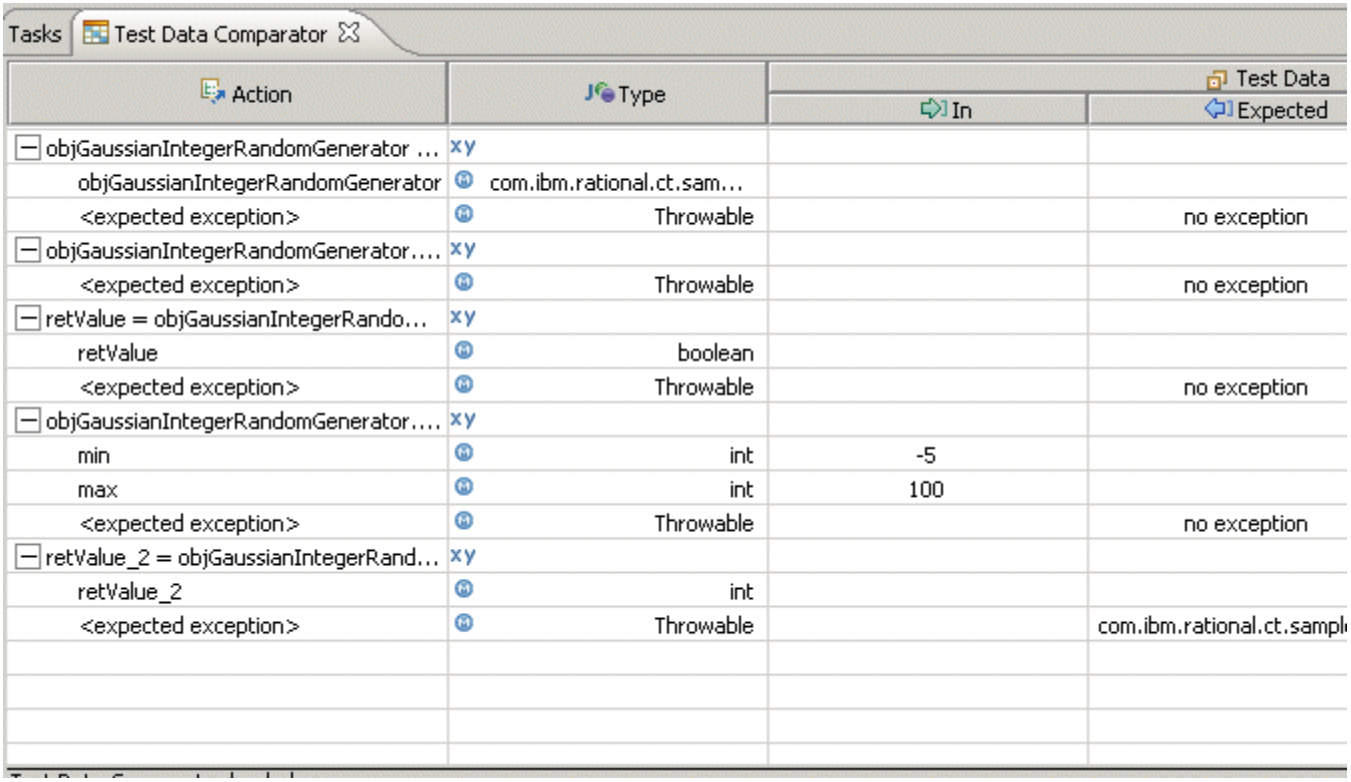


2. Under Configurations, click **Component Test**, click **New**, and enter a configuration name. (The next time you run a test, you can use the same configuration or create a new one.)

3. Select the test to be run.
4. Click the **Execution Results** tab, specify a location where you want the tests results to be stored, and click **Run**.
5. After the test run is finished, go to the Test Navigator and expand the Run folder until you find the Individual Tests.



6. Double-click an Individual Test to display the test results in the Test Data Comparator, as shown below:



The Test Data Comparator has three columns: one for the input data, one for the expected outputs, and one for the actual results. The actual results column appears in green when the actual results match the expected results and in red when there are discrepancies.

Finish your tutorial by reviewing the materials in the Summary.

Test Java components summary

This tutorial has taught you the basic steps that are involved in creating Java component tests.

Completed Learning Objectives

If you have completed all of the exercises, you should now be able to do the following:

- Create a test project
- Create a Java component test
- Edit the test by adding test data, editing the Java code, and adding stub components
- Add stub data
- Run the test and analyze the test results

More information

If you want to learn more about the topics covered in this tutorial, please refer to the help topics in the information center for the Rational Developer products.