

TEST DRIVEN DEVELOPMENT IN JAVA

Lab Guide

NIIT

R201171400011-SABHYA

Test Driven Development in Java

Lab Guide

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution and decompilation. Please view "Mark of Authenticity" label to establish proper licensed usage. No part of this document may be copied, reproduced, printed, distributed, modified, removed, amended in any form by any means whether electronic, mechanical, digital, optical, photographic or otherwise without prior written authorization of NIIT and its authorized licensors, if any.

Information in this document is subject to change by NIIT without notice. The names of companies, products, people, characters, and/or data mentioned herein are fictitious and are in no way intended to represent any real individual, company, entity, services, product or event, unless otherwise noted.

All products are registered trademarks of their respective organizations.
All software is used for educational purposes only.

Disclaimer: Every effort has been made to avoid errors or omissions in this publication. In spite of this, errors may creep in. Any mistake, error or discrepancy noted may be brought to our notice which shall be taken care of in the next edition. The documents and graphics in this courseware could include technical inaccuracies or typographical errors/translation errors. Changes are periodically added to the information herein. NIIT makes no representations about the accuracy of the information contained in the courseware and graphics in this courseware for any purpose. All documents and graphics are provided "as is". NIIT hereby disclaims all warranties and conditions with regard to this information, including all implied warranties and conditions of merchantability, fitness for any particular purpose, title and non-infringement. It is notified that NIIT will not be responsible for any consequential damage or loss of action to any one, of any kind, in any manner, therefrom. No part of this book may be reproduced or copied in any form or by any means [graphic, electronic or mechanical, including photocopying, recording, taping or information retrieval systems] or reproduced on any disc, tape, perforated media or other information storage device, etc., without the written permission of NIIT. Breach of this condition is liable for legal action.

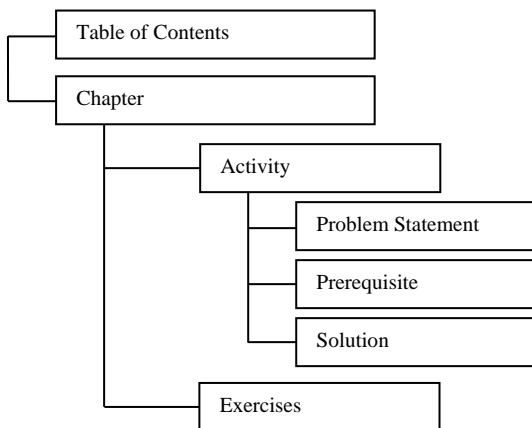
For binding mistake, misprints or for missing pages, etc., the publisher's liability is limited to replacement within one month of purchase by similar edition. All expenses in this connection are to borne by the purchaser.

Due to the dynamic nature of the internet, the URLs and web references mentioned in this document may be (are) subject to changes, for which NIIT shall not hold any responsibility.

All disputes are subject to Delhi jurisdiction only.

Test Driven Development in Java/LG/18_M08_V01
Copyright ©NIIT. All rights reserved.

COURSE DESIGN - LAB GUIDE



R201171400011-SABHYA

Table of Contents

Chapter 1 – Getting Started with Software Testing

Activity 1.1: Creating Test Scenarios	1.2
Problem Statement.....	1.2
Solution	1.2
Activity 1.2: Executing Test Cases in JUnit	1.4
Problem Statement.....	1.4
Solution	1.4
Exercises	1.7

Chapter 2 – Implementing Test Cases

Activity 2.1: Executing JUnit Test on Command Prompt	2.2
Problem Statement.....	2.2
Solution	2.2
Activity 2.2: Writing Test Cases in JUnit	2.4
Problem Statement.....	2.4
Solution	2.4
Exercises	2.8

Chapter 3 – Structuring Test Cases

Activity 3.1: Creating Parameterized Tests	3.2
Problem Statement.....	3.2
Solution	3.2
Exercises	3.4

The background of the entire page is a light blue network diagram. It consists of numerous small circular nodes connected by thin, light blue lines, forming a complex web-like structure. The nodes are distributed across the page, with some clusters being denser than others. The overall effect is a technical, digital aesthetic.

NIIT

R201171400011-SABHYA

Getting Started with Software Testing

CHAPTER 1



Activity 1.1: Creating Test Scenarios

Problem Statement

Sandra has created the following code snippet to find the greatest of three numbers:

```
public class Greatest {  
    public static int greatest(int a, int b, int c)  
    {  
        if(a>b && a>c)  
            return a;  
        else if(b>a && b>c)  
            return b;  
        else  
            return c;  
    }  
}
```

Now, you need to test this class considering the following conditions:

- Pass the integer values to the method to make the test pass.
- Pass the negative values to the method to check its outcome.
- Pass the decimal values to the method to make the test fail.
- Pass the string values to the method to make the test fail.

Create four test scenarios for the preceding conditions.

Solution

To create test scenarios for testing the `Greatest` class, you need to perform the following tasks:

1. Create a test scenario with integer values.
2. Create a test scenario with negative values.
3. Create a test scenario with decimal values.
4. Create a test scenario with string values.

Task 1: Creating a Test Scenario with Integer Values

To pass the integer values to the `greatest()` method, you need to perform the following steps:

1. Input three valid integer values, such as 5, 8, and 2, to the method.
2. Logically decide the outcome of the method. As a result, the expected output should be 8.
3. Process the logic within the method using these values, which fetches 8 as the actual result.
4. Test passes because the actual result matches the expected result.

Task 2: Creating a Test Scenario with Negative Values

To pass the negative values to the `greatest()` method, you need to perform the following steps:

1. Input three valid negative values, such as -2, -5, and -10, to the method.
2. Logically decide the outcome of the method. As a result, the expected output should be -2.
3. Process the logic within the method using these values, which fetches -2 as the actual result.
4. Test passes because the actual result matches the expected result.

Task 3: Creating a Test Scenario with Decimal Values

To pass the decimal values to the `greatest()` method, you need to perform the following steps:

1. Input three valid decimal values, such as 2.5, 5.5, and 10.5, to the method.
2. Logically decide the outcome of the method. As a result, the expected output should be an error because the data-type specified for the input parameters is `int`, which does not cover decimal values. Therefore, an error message for passing decimal values should be displayed.
3. Process the logic within the method using these values, which displays the error for passing the decimal values instead of integer values.
4. Test fails because the error is displayed.

Task 4: Creating a Test Scenario with String Values

To pass the string values to the `greatest()` method, you need to perform the following steps:

1. Input three string values, such as '10', '20', and '30', to the method.
2. Logically decide the outcome of the method. As a result, the expected output should be an error because the data-type specified for the input parameters is `int`, which does not cover string values. Therefore, an error message for passing string values should be displayed.
3. Process the logic within the method using these values, which displays the error for passing the string values instead of integer values.
4. Test fails because the error is displayed.

All these test scenarios can be collated together, as shown in the following table.

<i>Test Case No.</i>	<i>Input Values</i>	<i>Expected Values</i>
<i>Test case 1</i>	(5, 8, 2)	8
<i>Test case 2</i>	(-2,-5,-10)	-2
<i>Test case 3</i>	(2.5, 5.5, 10.5)	Error message is displayed for decimal values
<i>Test case 4</i>	('10','20','30')	Error message is displayed for string values

Test Scenarios for Testing the Greatest Class



Activity 1.2: Executing Test Cases in JUnit

Problem Statement

Explore the testing environment in NetBeans IDE using the JUnit framework.

Prerequisite: To perform this activity, you need to use the **Activity1_2.zip** file.

Solution

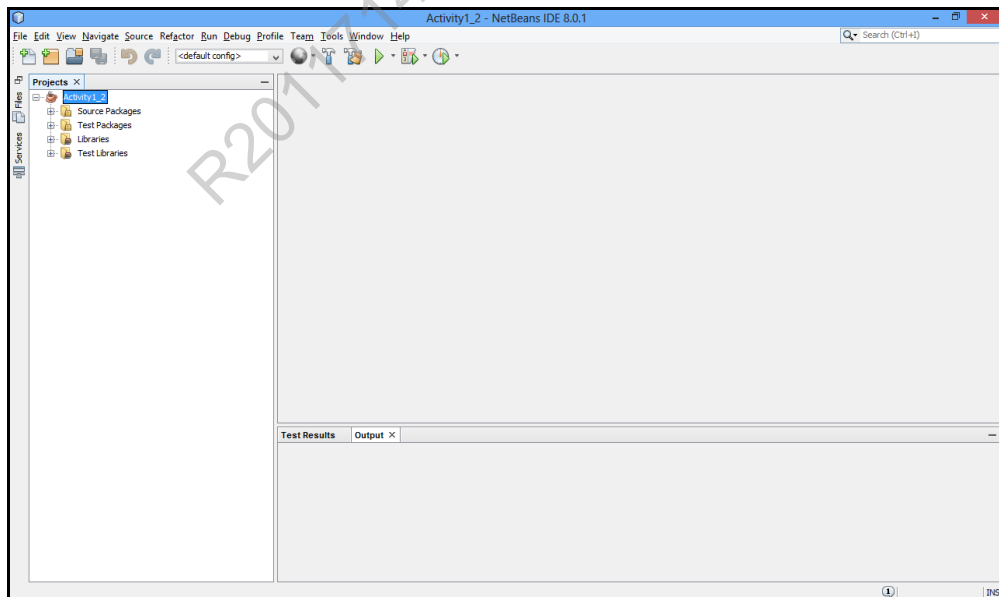
To explore the testing environment in NetBeans IDE using the JUnit framework, you need to perform the following tasks:

1. Open the **Activity1_2** project.
2. Run the test file to execute the test.

Task 1: Opening the Activity1_2 Project

To open the **Activity1_2** project, you need to perform the following steps:

1. Open **NetBeans IDE 8.0.1**. The **NetBeans IDE 8.0.1** window appears.
2. Select **File→Open Project**. The **Open Project** dialog box appears.
3. Browse to the location where the **Activity1_2** project is saved.
4. Click the **Open Project** button. The **Activity1_2** project opens in the **Projects** window of NetBeans IDE, as shown in the following figure.

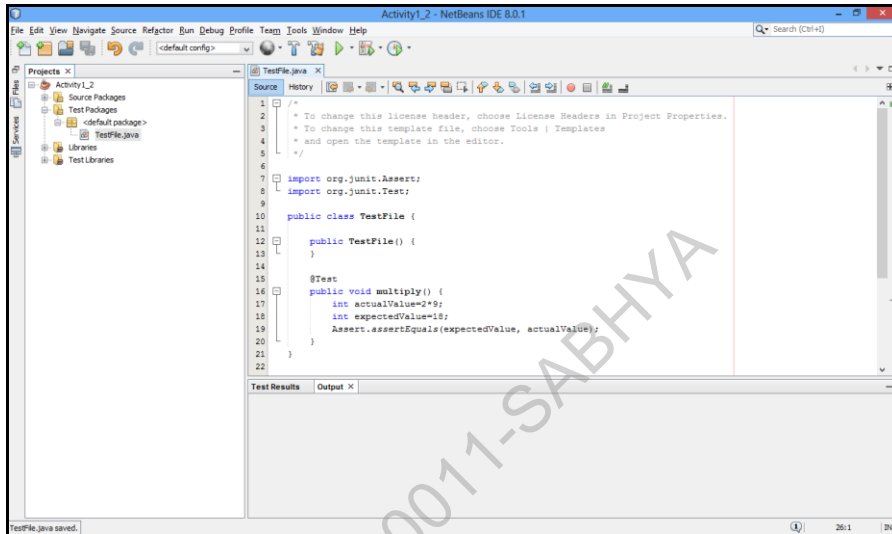


The Activity1_2 Project in the Projects Window

Task 2: Running the Test File to Execute the Test

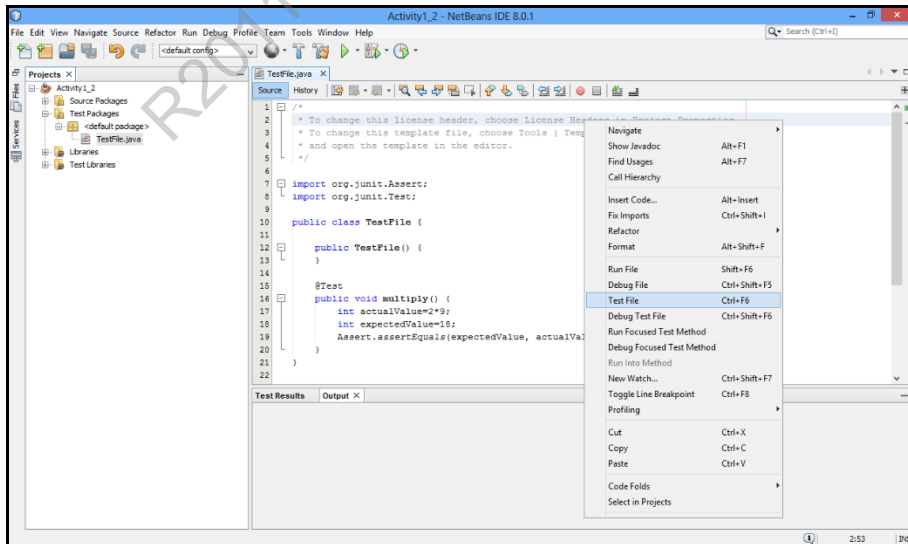
In order to run the test file and verify whether the test passes or fails, you need to perform the following steps:

1. Expand the **Test Packages**→<default package> folder within the **Activity1_2** project.
2. Double-click the **TestFile.java** file within it. The test file opens in the right-pane of NetBeans IDE, as shown in the following figure.



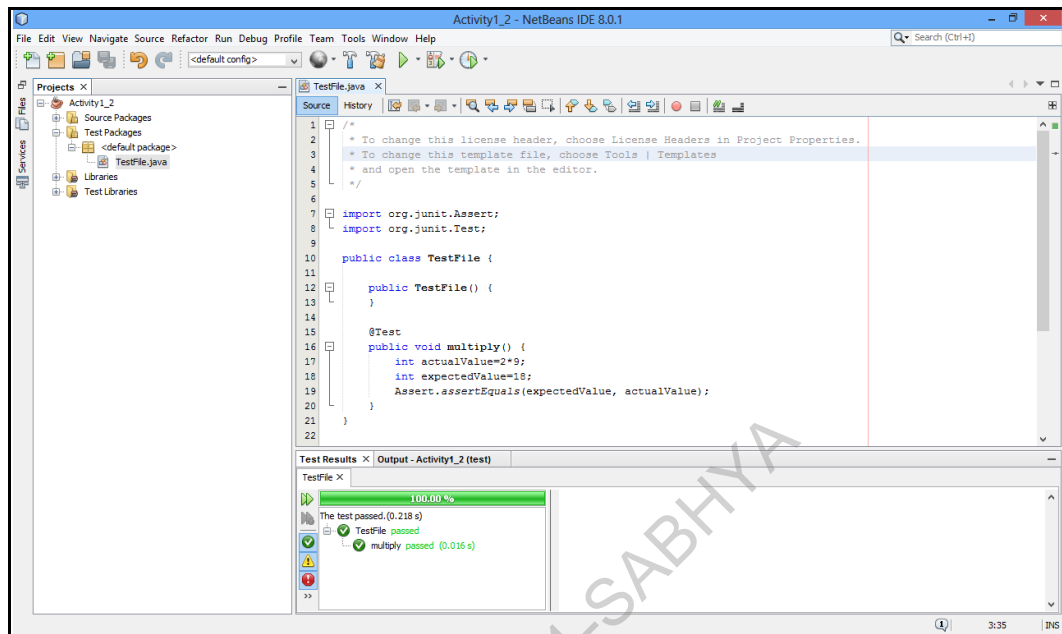
The TestFile.java File

3. Right-click the **TestFile.java** file in the right-pane and select the **Test File** option from the pop-up menu, as shown in the following figure.



The Test File Option

The test is executed and displays a green bar in the **Test Results** window, as shown in the following figure.




The Test Results Window

The green bar signifies that test has passed.

Note

*The time displayed in the **Test Results** window may vary on different systems because it is the time taken by the particular system to execute the test.*

Note

*If you do not get the detailed output of the test, showing the details of the test passed, then you need to click the **Show Passed**  button on the left bar of the **Test Results** window.*

Exercises

Exercise 1

Fred has written the following code snippet to find the factorial of a number given by a user:

```
public static long fact(long num) {  
    if (num < 0)  
    {  
        System.out.println("Invalid number:");  
        return num;  
    }  
    else  
    {  
        if (num <= 1)  
            return 1;  
        else  
            return num * fact(num - 1);  
    }  
}
```

You need to test this method considering the following values are passed to it as a parameter:

1. A natural number that makes the test pass
2. Another natural number that returns a large value of its factorial
3. The value, 0, to check for the boundary values
4. A negative number, which is an invalid number for factorial
5. A decimal number that makes the test fail

Create the required test scenarios for the method.

Exercise 2

Consider the following code snippet:

```
public class Calculator {
    public int add (int num1, int num2)
    {
        return num1 + num2;
    }
    public int subtract (int num1, int num2)
    {
        return num1 - num2;
    }
    public int multiply (int num1, int num2)
    {
        return num1 * num2;
    }
    public int divide (int num1, int num2)
    {
        return num1 / num2;
    }
}
```

You have to test the preceding code snippet thoroughly considering the following situations:

1. Pass integer values to the four methods to ensure that the tests pass.
2. Pass the combination of integer and decimal values to the `add()`, `subtract()`, and `multiply()` methods to make the test fail.
3. Divide a number by zero to make the test generate an exception.

Create the test scenarios based on the preceding situations.

Exercise 3

You have been assigned a task of creating an application that enables the employees of an organization to mark their attendance and apply for leaves. You divide the application into three modules: login, attendance, and leaves.

1. These modules have been developed by individual developers. Now, each of these modules needs to be tested before they are integrated together in the application. Which testing level will you use to test the modules individually? Justify your answer.
2. Next, you need to integrate these modules and test the complete application. Which testing level and approach will you use to test the application? Justify your answer.



NIIT

R201171400011-SABHYA

Implementing Test Cases

CHAPTER 2



Activity 2.1: Executing JUnit Test on Command Prompt

Problem Statement

You have developed a Java application that multiplies two numbers. Now, you need to create a test case and verify this application. You need to display the test results on the command prompt window. For this, you have been provided with the `TestRunner.java` file. If the test cases pass, the message, `true`, is displayed on the command prompt window. How will you accomplish this task?

Prerequisite: To perform this activity, you need to use the **Multiply.zip** file. Extract the **Multiply** project folder and save it at an appropriate location on your system.

Solution

To perform this activity, you need to perform the following steps:

1. Open the `TestJUnit.java` file.
2. Type the highlighted portion of the following code snippet in the `TestJUnit.java` file:

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;
public class TestJUnit
{
    public int multiply(int n1, int n2)
    {
        return n1*n2;
    }
    @Test
    public void testPrintMessage()
    {
        TestJUnit t=new TestJUnit();
        assertEquals(12,t.multiply(4,3));
    }
}
```

3. Open the **Command Prompt** window and navigate to the project folder where the `TestJUnit.java` and `jar` files are located.
4. Type the following command on the **Command Prompt** window to compile the java files:

```
javac TestJUnit.java TestRunner.java
```

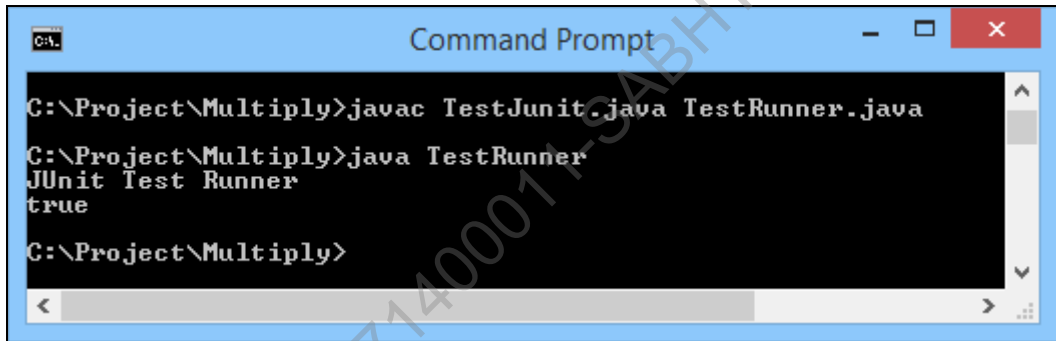
Note

*In order to execute the preceding command, you need to set the environment variables: **Path**, **JAVA_HOME**, **JUNIT_HOME**, and **CLASSPATH**. To set the environment variables, refer Chapter 1 of Student Guide.*

5. Press the **Enter** key. The `TestJUnit.java` and `TestRunner.java` files are compiled and the prompt reappears.
6. Type the following command on the **Command Prompt** window to execute the `TestRunner.java` file:

```
java TestRunner
```

7. Press the **Enter** key. The result is displayed on the **Command Prompt** window, as shown in the following figure.



```
C:\Project\Multiply>javac TestJUnit.java TestRunner.java
C:\Project\Multiply>java TestRunner
JUnit Test Runner
true
C:\Project\Multiply>
```

The Command Prompt Window Showing Test Result



Activity 2.2: Writing Test Cases in JUnit

Problem Statement

Sam has developed a Java application to calculate the area of a circle, rectangle, and square. You have been assigned the task to test this application. While testing, you need to ensure that your test cases should cover all the methods of the application. How will you accomplish this task?

Prerequisite: To perform this activity, you need to use the **Area.zip** file.

Solution

To perform this activity, you need to perform the following steps:

1. Open the **Area** project in **NetBeans IDE 8.0.1**.
2. Right-click the **Area** project in the **Projects** window, and then select **New→Other**. The **New File** dialog box is displayed.
3. Select the **Unit Tests** option from the **Categories** section.
4. Ensure that the **JUnit Test** option is selected in the **Files Types** section.

5. Click the **Next** button. The **New JUnit Test** dialog box appears, as shown in the following figure.

New JUnit Test

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Class Name:

Project:

Location:

Package:

Created File:

Generated Code

- ☒ Test Initializer
- ☒ Test Finalizer
- ☒ Test Class Initializer
- ☒ Test Class Finalizer

Generated Comments

- ☒ Source Code Hints

Warning: It is highly recommended that you do not place Java classes in the default package.

< Back Next > **Finish** Cancel Help

The New JUnit Test Dialog Box

6. Type **AreaTest** in the **Class Name** text box.
7. Type **TestArea** in the **Package** text box.
8. Click the **Finish** button. The **Select JUnit Version** dialog box appears, as shown in the following figure.

Select JUnit Version

Select JUnit version for which the created test skeletons should be created:

☐ JUnit 3.x

☒ JUnit 4.x

Select Cancel Help

The Select JUnit Version

9. Ensure that the **JUnit 4.x** option is selected.
10. Click the **Select** button. The `AreaTest.java` file is created.
11. Replace the code in the `AreaTest.java` file with the following code:

```
package TestArea;

import org.junit.After;
import org.junit.AfterClass;
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import area.Rectangle;
import area.Area;
import area.Circle;
import area.Square;

public class AreaTest {

    public AreaTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

    @Test
    public void testRectangle() {
        System.out.println("Testing Rectangle");
        Area instance = new Area();
        double expectedResult = 800;
        Rectangle rectangle = new Rectangle(20,40);
        double result = instance.calculateArea(rectangle);
        assertEquals(expResult, result,0.0);
    }

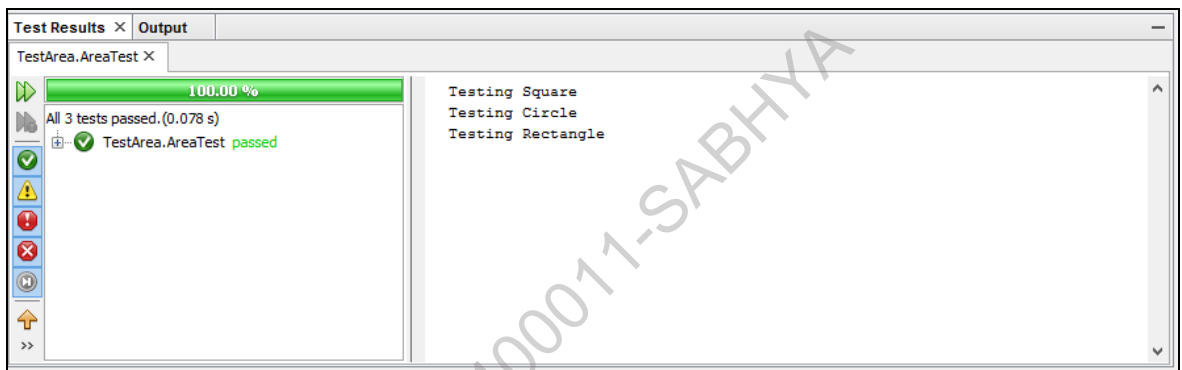
    @Test
    public void testCircle() {
        System.out.println("Testing Circle");
        Area instance = new Area();
        double expectedResult = 706.5;
        Circle circle = new Circle(15);
        double result = instance.calculateArea(circle);
    }
}
```

```

        assertEquals(expResult, result, 2);
    }
    @Test
    public void testSquare() {
        System.out.println("Testing Square");
        Area instance = new Area();
        double expResult = 1225;
        Square square = new Square(35);
        double result = instance.calculateArea(square);
        assertEquals(expResult, result, 0.0);
    }
}

```

12. Right-click the `AreaTest.java` file in the **Projects** window, and then select **Test File**. The test result is displayed in the **Test Results** window, as shown in the following figure.



The Test Results Window

Note

If the **Test Results** window is not displayed, select **Window → IDE Tools → Test Results**. In addition, the sequence of output statements displayed in the preceding figure may vary because the test methods execute in random order.

Exercises

Exercise 1

Joseph, a software developer, has developed a Java application to calculate the speed of an object. The method takes distance and time of an object as parameters and accordingly calculates the speed of the object. Now, he has been assigned the task to create test cases for this application on the following scenarios:

- A test case that takes integer values as input
- A test case that takes decimal values as input

Help Joseph accomplish this task.

Prerequisite: To perform this exercise, you need to use the **Speed.zip** file.

Exercise 2

You have developed a Java application to calculate the simple interest. The method takes principal amount, interest, and time as its parameters. Now, you have been assigned the task to create test cases on the following scenarios:

- A test case that takes integer values as input
- A test case that takes decimal values as input

How will you accomplish this task?

Prerequisite: To perform this exercise, you need to use the **SimpleInterest.zip** file.

Exercise 3

You have developed a Java application to convert the temperature entered in Fahrenheit to Celsius. Now, you have been assigned the task to create test cases on the following scenarios:

- A test case that takes an integer value as an input
- A test case that takes a decimal value as an input
- A test case that fails in case the user enters a value above or below the Fahrenheit range (-50 – 200)

How will you accomplish this task?

Prerequisite: To perform this exercise, you need to use the **Temperature.zip** file.

The background of the entire page is a light blue network diagram. It consists of numerous circular nodes connected by thin, dark blue lines, forming a complex web of connections. The nodes are of varying sizes, and the lines are of varying thicknesses, creating a sense of depth and connectivity.

NIIT

R201171400011-SABHYA

Structuring Test Cases

CHAPTER 3



Activity 3.1: Creating Parameterized Tests

Problem Statement

James has been asked to create a password generator application for a bank. The application generates a unique password by combining the first three letters of a customer's name with some random characters and numbers. For example, if the name of the customer is Jamie Hodge, the password can be Jam#\$\$, Jam56\$, Jam#\$\$645, or Jam#r\$5. For this, James has developed an application that contains a method to generate the unique password. This method takes two strings as its parameters and concatenates them. The first parameter accepts the first three letters of the customer's name, and the second parameter accepts the random characters and numbers. James needs to test the consistency of this application by creating a test case that accepts the following types of values:

- Combination of characters and numbers
- Combination of characters and special characters
- Combination of characters, special characters, and numbers

However, you have to execute a single test case for the preceding sets of values. Help James accomplish this task.

Prerequisite: To perform this activity, you need to use the **ComStr.zip** file.

Solution

To perform this activity, you need to perform the following steps:

1. Open the **ComStr** project in **NetBeans IDE 8.0.1**.
2. Right-click the **ComStr** project in the **Projects** window, and then select **New→JUnit Test**. The **New JUnit Test** dialog box appears.
3. Type **TestComStr** in the **Class Name** text box.
4. Type **Test** in the **Package** text box.
5. Click the **Finish** button. The **TestComStr.java** file is created.
6. Replace the code in the **TestComStr.java** file with the following code:

```
package Test;

import comstr.ComStr;
import java.util.Arrays;
import java.util.Collection;
import static org.junit.Assert.*;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;
```

```

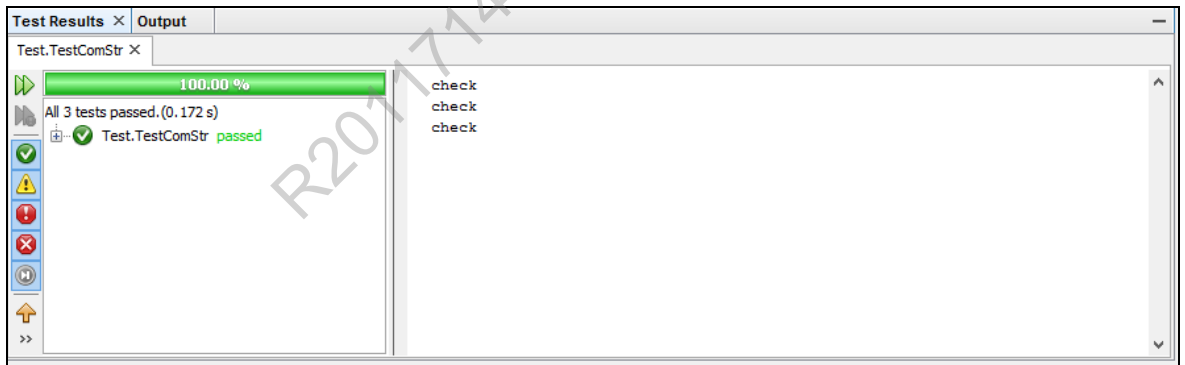
@RunWith(Parameterized.class)
public class TestComStr {
    private String expected;
    private String Char1;
    private String Char2;
    public TestComStr(String expectedresult, String Character1, String
Character2) {
        this.expected=expectedresult;
        this.Char1=Character1;
        this.Char2=Character2;
    }

    @Parameters
    public static Collection<String[]> checkednumbers() {
        return Arrays.asList(new String[][] {
            {"Jam345", "Jam", "345"}, {"Wod$$$%", "Wod", "$#$%"}, {"Sam34ed#$$%", "Sam", "34ed#$$%",
            "}});
    }

    @Test
    public void testCheck() {
        System.out.println("check");
        ComStr p=new ComStr();
        assertEquals(expected,p.combine(Char1,Char2));
    }
}

```

7. Right-click the `TestComStr.java` file in the **Projects** window, and then select **Test File**. The test result appears in the **Test Results** window, as shown in the following figure.



The Test Results Window in NetBeans

Exercises

Exercise 1

Harry has developed an application to check whether an entered character is a vowel or a consonant. The function takes a character as its input and returns vowel, if the character is a vowel, otherwise, consonant. Now, he has been assigned the task to test this application by creating test cases for all the possible scenarios, such as checking for a vowel and checking for a consonant. If a test case fails, an appropriate human-readable error message should be displayed.

Help Harry accomplish this task.

Prerequisite: To perform this exercise, you need to use the **VowelConsonant.zip** file.

R201171400011-SABHYA

R201171400011-SABHYA

NIIT