

ASSIGNMENT_4

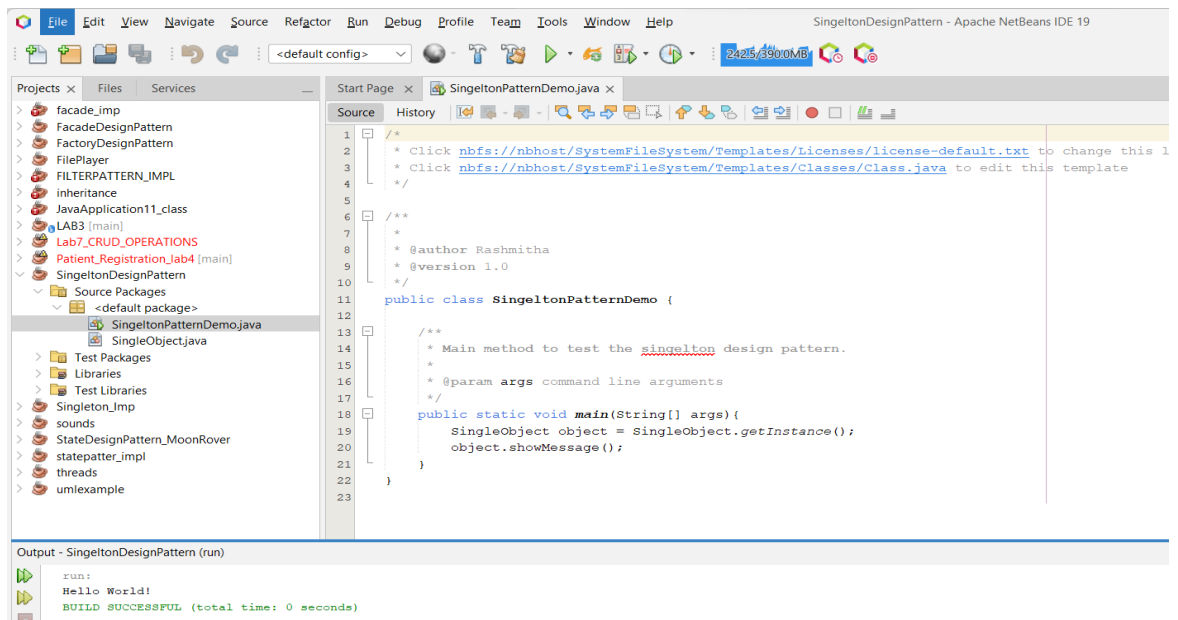
DESIGN PATTERNS

NUID: 002837410

SNAPSHOTS:

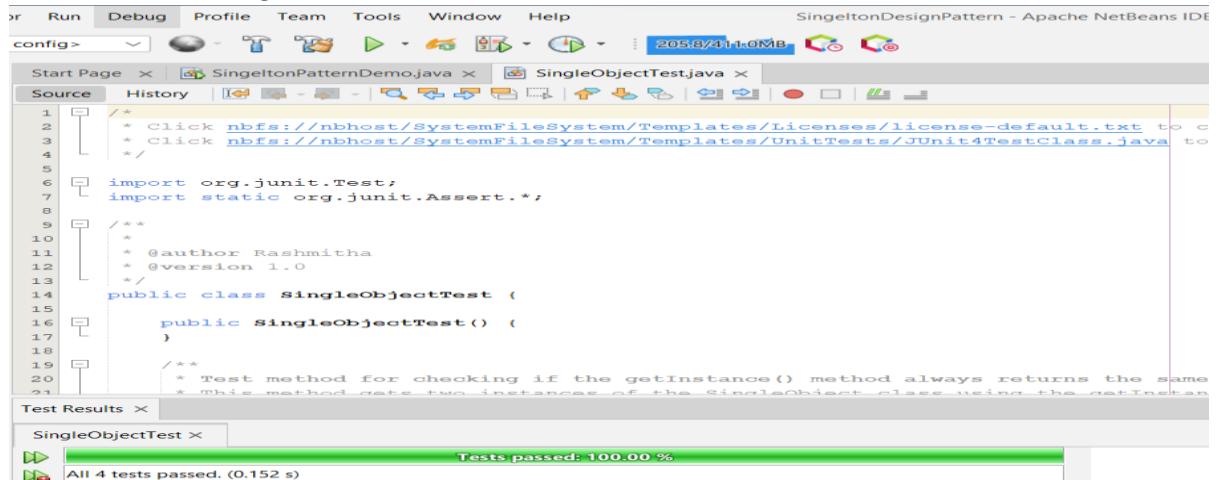
1. Singleton pattern

Sample Run:



TEST RUNS:

All Test cases running:



Test_Case:1

The screenshot shows an IDE with the following components:

- Source Editor:** Displays the code for `SingletonObjectTest.java`. The code includes two test methods: `testGetInstance()` and `testShowMessage()`. The `testGetInstance()` method is currently selected and highlighted in yellow. It contains the following code:

```
21  * This method gets two instances of the SingletonObject class using the getInstance() method,  
22  * and finally, it checks if both instances are equal, meaning that they refer to the same object in memory.  
23  */  
24  @Test  
25  public void testGetInstance() {  
26      SingletonObject obj1=SingletonObject.getInstance();  
27      SingletonObject obj2=SingletonObject.getInstance();  
28      assertTrue(condition:obj1.equals(obj2));  
29  }  
30  
31  /**  
32   * Test of showMessage method, of class SingletonObject.  
33   */  
34  @Test  
35  public void testShowMessage() {  
36  }  
37  
38  /**  
39   * Test method for getting the identifier value of the Singleton object.  
40   * This method sets the identifier value of the Singleton object using the setIdentifier() method,  
41   * then it gets the identifier value of the Singleton object using the getIdentifier() method,
```
- Test Results:** A tab titled "Test Results" is active, showing the results for `SingletonObjectTest.testGetInstance()`. It displays a green bar indicating "Tests passed: 100.00 %". Below this, it states "The test passed. (0.15 s)".

Test_Case:2

The screenshot shows the same IDE as in Test_Case:1, but with the following changes:

- Source Editor:** The code is the same, but the `testShowMessage()` method is now selected and highlighted in yellow. It contains the following code:

```
34  @Test  
35  public void testShowMessage() {  
36  }  
37  
38  /**  
39   * Test method for getting the identifier value of the Singleton object.  
40   * This method sets the identifier value of the Singleton object using the setIdentifier() method,  
41   * then it gets the identifier value of the Singleton object using the getIdentifier() method,
```
- Test Results:** The "Test Results" tab now shows the results for `SingletonObjectTest.testShowMessage()`. It displays a green bar indicating "Tests passed: 100.00 %". Below this, it states "The test passed. (0.127 s)".

Test case:3

SingletonDesignPattern - Apache NetBeans IDE 19

```
public void testShowMessage() {  
}  
  
/**  
 * Test method for getting the identifier value of the Singleton object.  
 * This method sets the identifier value of the Singleton object using the setIdentifier() method,  
 * then it gets the identifier value of the Singleton object using the getIdentifier() method,  
 * and finally, it checks if the retrieved identifier value matches the set identifier value.  
 */  
@Test  
public void testGetIdentifier() {  
    Singleton obj1=Singleton.getInstance();  
    obj1.setIdentifier(identifier: "value 1");  
    Singleton obj2=Singleton.getInstance();  
    assertEquals(expected: "value 1", actual=obj2.getIdentifier());  
}  
  
/**  
 * Test of setIdentifier method, of class Singleton.  
 */  
@Test  
public void testSetIdentifier() {  
}
```

Test Results

SingleObjectTest.testGetIdentifier ×

Tests passed: 100.00 %

The test passed. (0.125 s)

Test case:4

290.0/325.0M

```
public void testShowMessage() {  
}  
  
/**  
 * Test method for getting the identifier valu  
 * This method sets the identifier value of th  
 * then it gets the identifier value of the Si  
 * and finally, it checks if the retrieved ide  
 */  
@Test  
public void testGetIdentifier() {  
    Singleton obj1=Singleton.getInstance  
    obj1.setIdentifier(identifier: "value 1");  
    Singleton obj2=Singleton.getInstance  
    assertEquals(expected: "value 1", actual=obj2.g  
}  
  
/**  
 * Test of setIdentifier method, of class Sing  
 */  
@Test  
public void testSetIdentifier() {  
}
```

Test Results

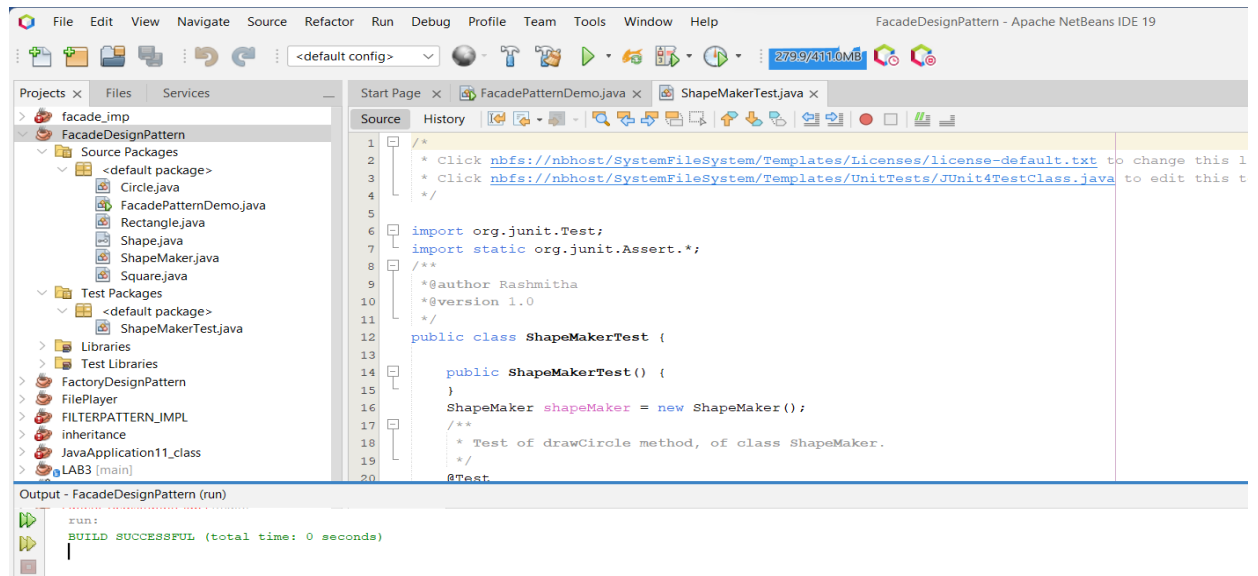
SingleObjectTest.testGetIdentifier × SingleObjectTest.testSetIdentifier ×

Tests passed: 100.00 %

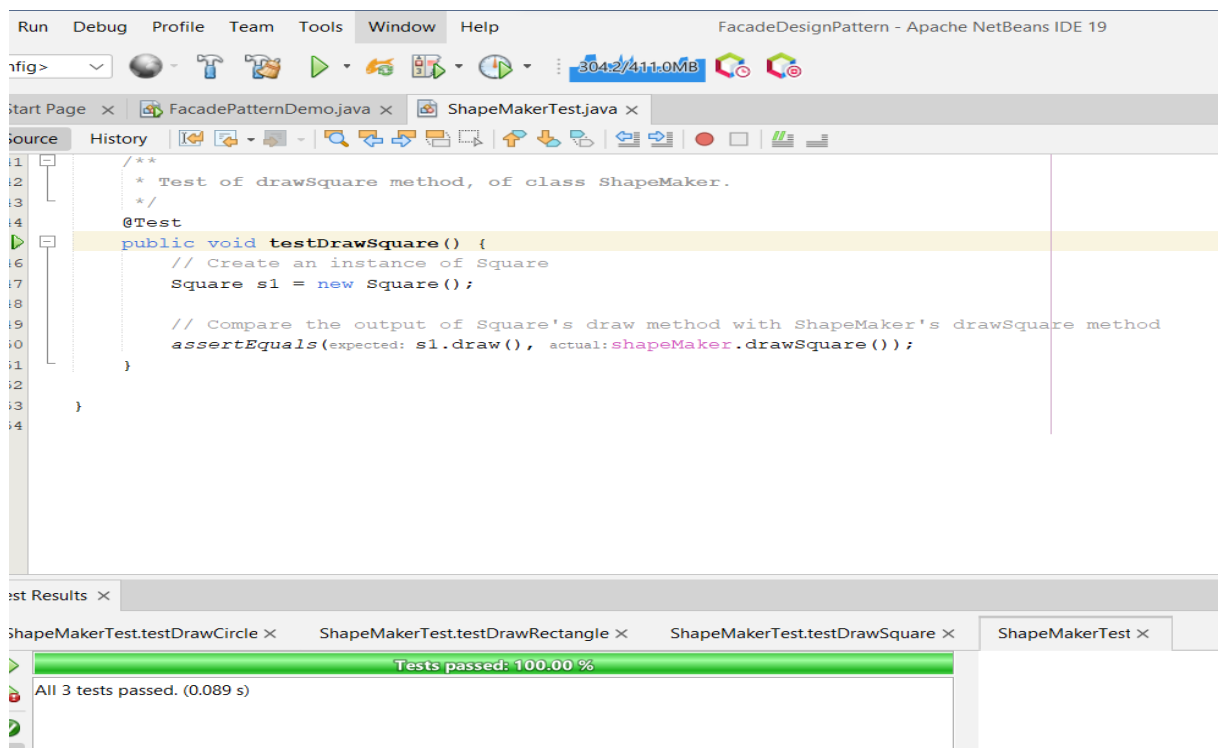
The test passed. (0.12 s)

2. Facade design pattern

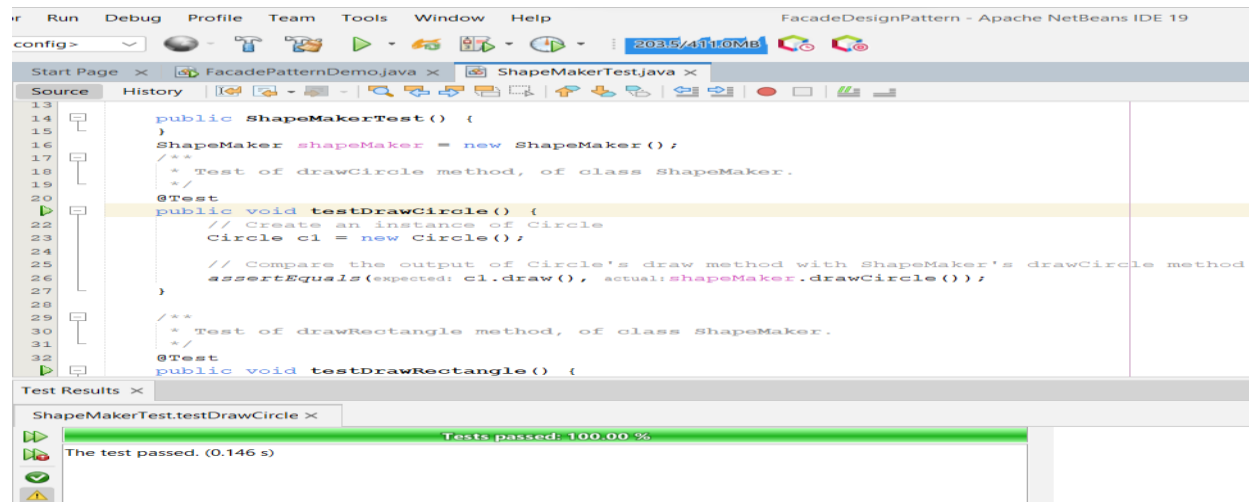
Sample Run:



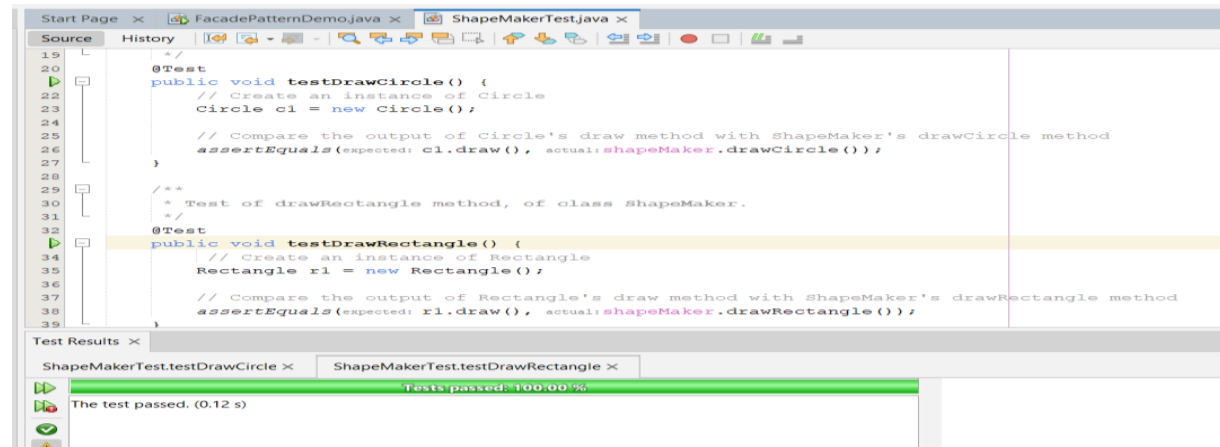
All Test cases running:



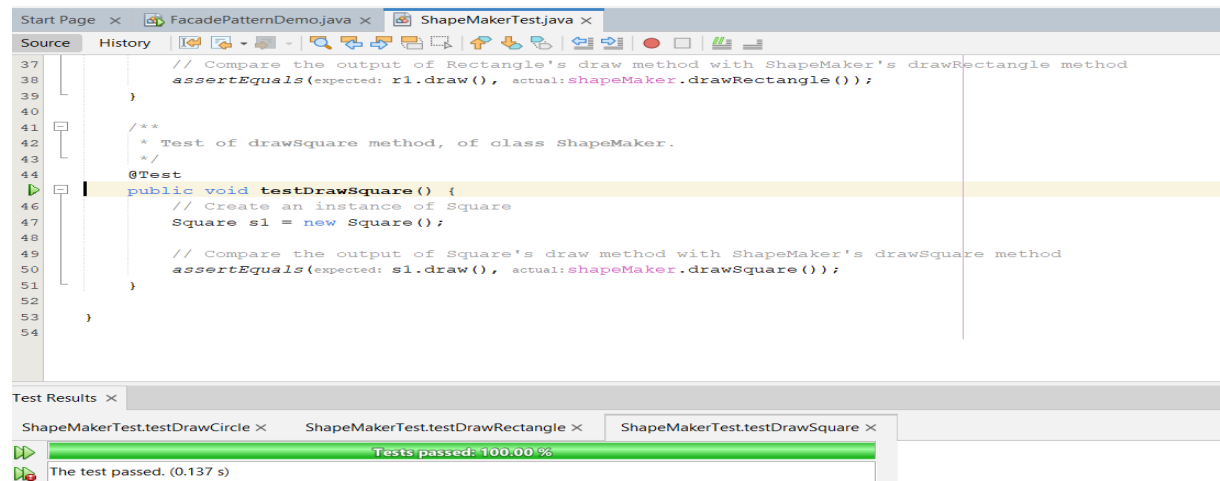
Test Case:1



Test Case:2

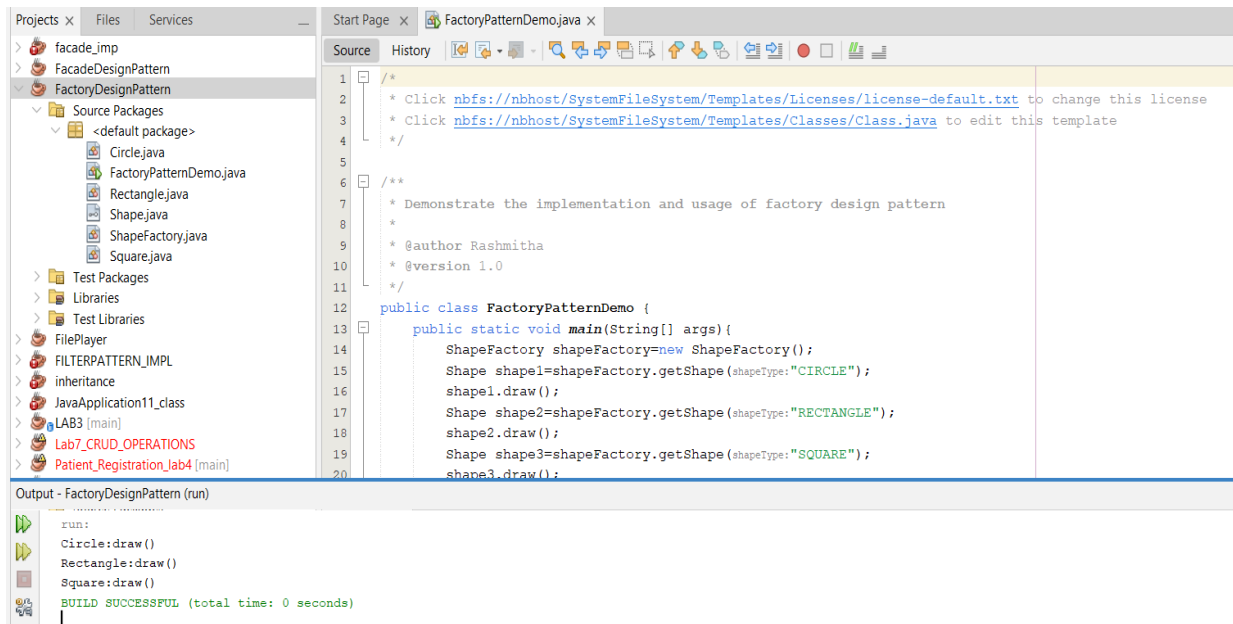


Test Case: 3



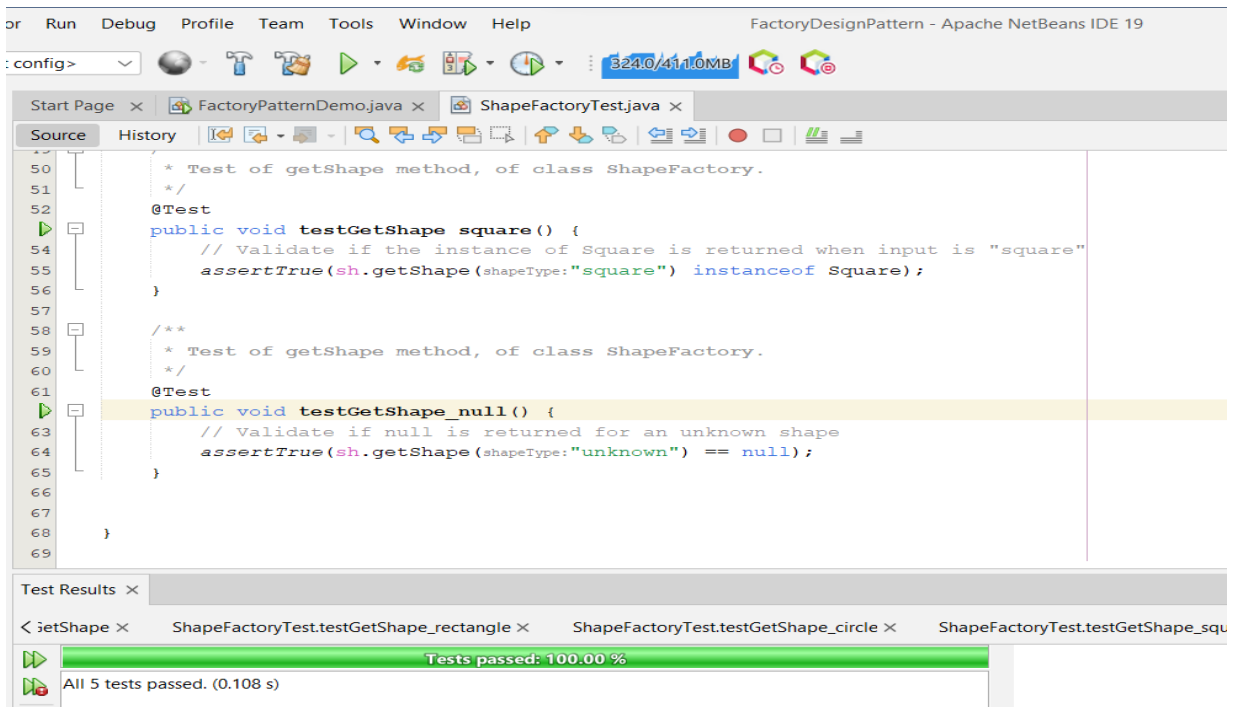
3. Factory Design Pattern

Sample Run:

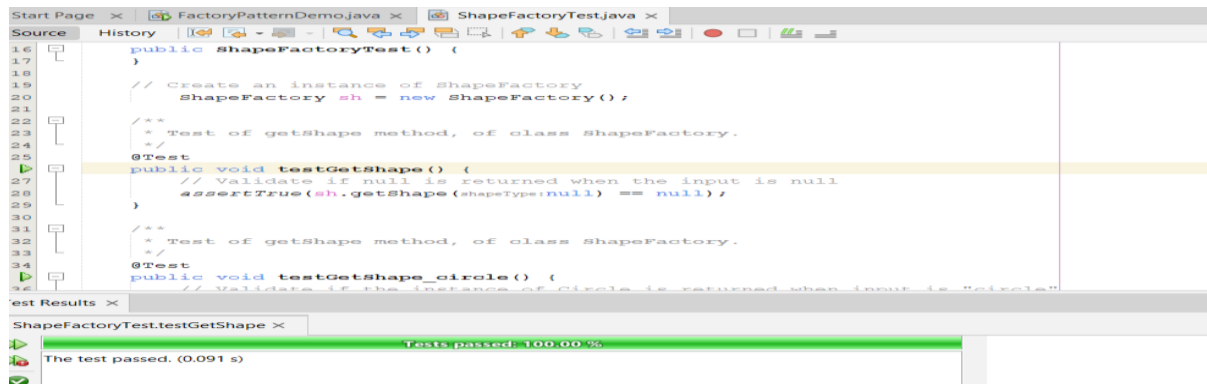


Test Runs:

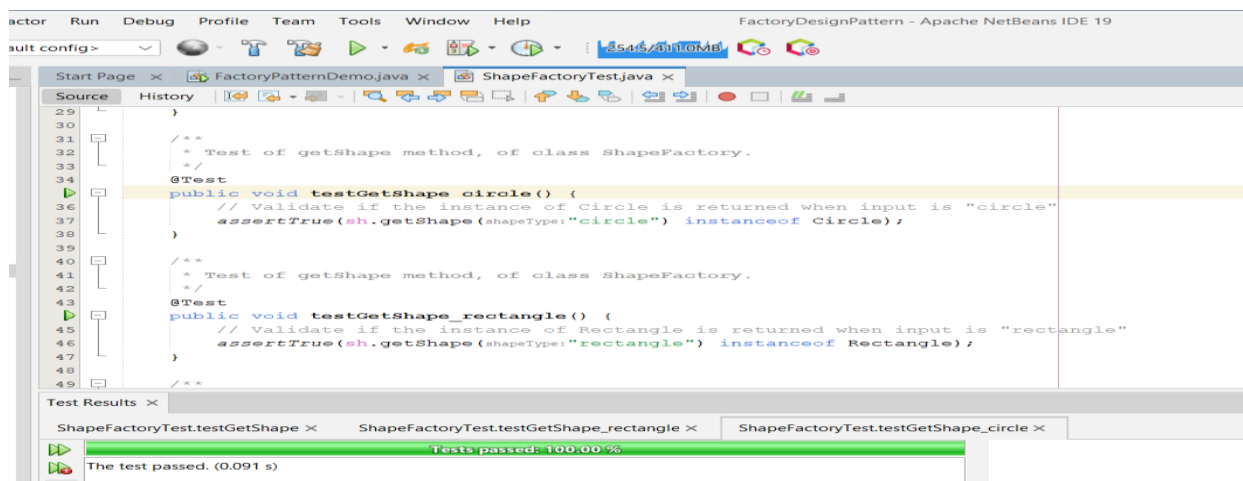
All test cases running:



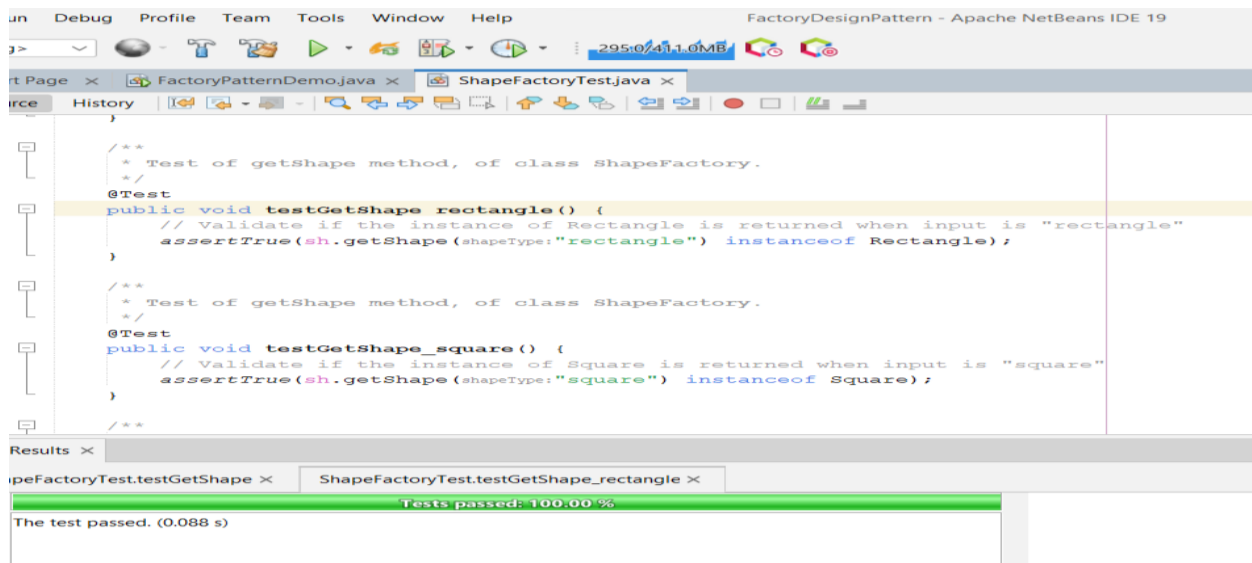
Test Case:1



Test Case:2



Test Case:3



Test case:4

The screenshot shows the NetBeans IDE with the `ShapeFactoryTest.java` file open. The source code is as follows:

```
50      * Test of getShape method, of class ShapeFactory.
51      */
52      @Test
53      public void testGetShape_square() {
54          // Validate if the instance of Square is returned when input is "square"
55          assertTrue(sh.getShape(shapeType:"square") instanceof Square);
56      }
57
58      /**
59      * Test of getShape method, of class ShapeFactory.
60      */
61      @Test
62      public void testGetShape_null() {
63          // Validate if null is returned for an unknown shape
64          assertTrue(sh.getShape(shapeType:"unknown") == null);
65      }
66
67
68  }
69
```

The Test Results window shows the following results:

- ShapeFactoryTest.testGetShape ×
- ShapeFactoryTest.testGetShape_rectangle ×
- ShapeFactoryTest.testGetShape_circle ×
- ShapeFactoryTest.testGetShape_square ×

Tests passed: 100.00 %

The test passed. (0.165 s)

Test case:5

The screenshot shows the NetBeans IDE with the `ShapeFactoryTest.java` file open. The source code is the same as in Test case 4:

```
50      * Test of getShape method, of class ShapeFactory.
51      */
52      @Test
53      public void testGetShape_square() {
54          // Validate if the instance of Square is returned when input is "square"
55          assertTrue(sh.getShape(shapeType:"square") instanceof Square);
56      }
57
58      /**
59      * Test of getShape method, of class ShapeFactory.
60      */
61      @Test
62      public void testGetShape_null() {
63          // Validate if null is returned for an unknown shape
64          assertTrue(sh.getShape(shapeType:"unknown") == null);
65      }
66
67
68  }
69
```

The Test Results window shows the following results:

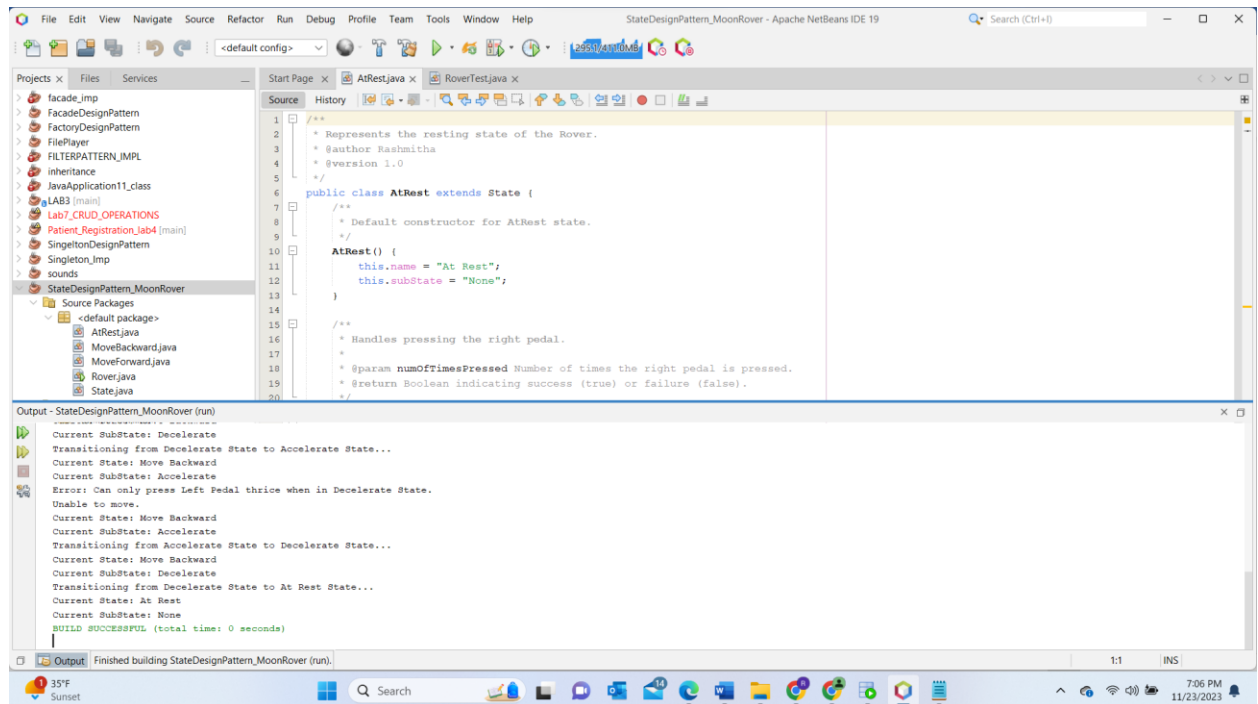
- ShapeFactoryTest.testGetShape ×
- ShapeFactoryTest.testGetShape_rectangle ×
- ShapeFactoryTest.testGetShape_circle ×
- ShapeFactoryTest.testGetShape_square ×

Tests passed: 100.00 %

The test passed. (0.111 s)

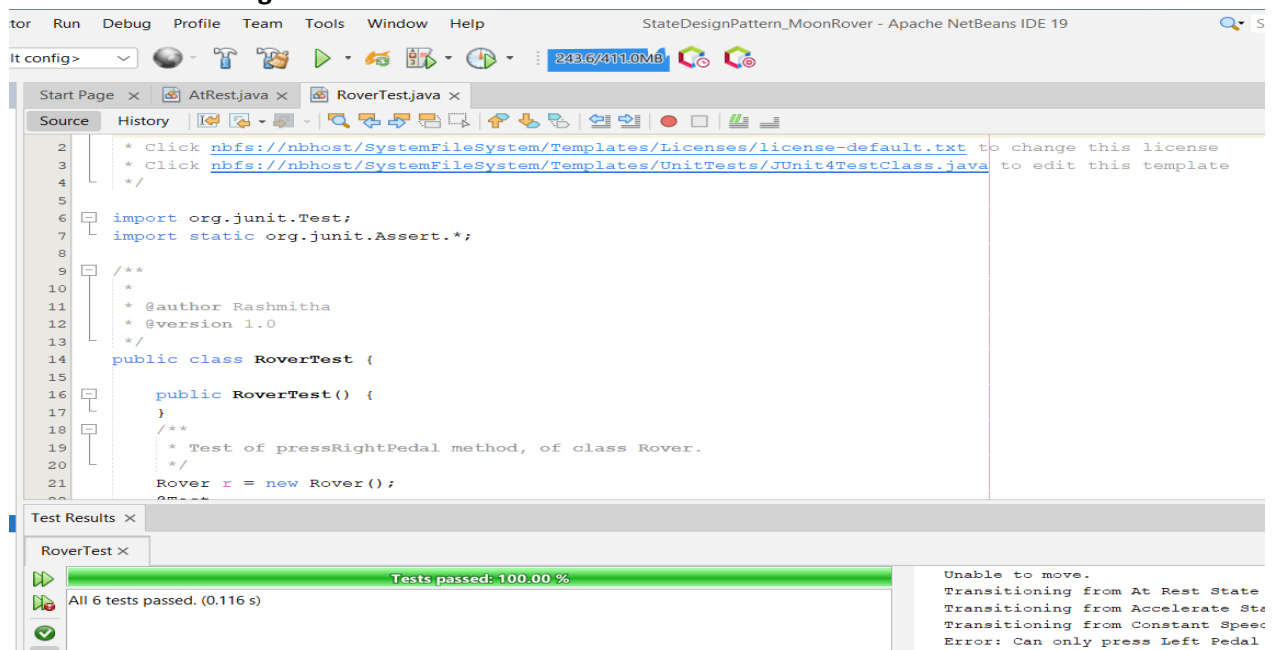
4. StateDesignPattern Moon Rover

Sample_Run:



Test Runs:

All Test cases running:



Test_Case:1

NetBeans IDE interface showing the execution of Test_Case:1. The source code for `RoverTest.java` is displayed, showing a test method `testPressRightPedal()` that tests the `pressRightPedal` method of the `Rover` class. The test results window shows that the test passed successfully with 100.00% success rate.

```
public class RoverTest {  
    public RoverTest() {  
        /**  
         * Test of pressRightPedal method, of class Rover.  
         */  
        Rover r = new Rover();  
        @Test  
        public void testPressRightPedal() {  
            //1. Rover should always start in "At Rest" state  
            assertEquals(expected: "At Rest", actual: r.currentState.name);  
            //2. When at rest, pressing right pedal twice should not move rover  
            r.pressRightPedal(numOfTimesPressed: 2);  
            assertEquals(expected: "At Rest", actual: r.currentState.name);  
            //3. When at rest, pressing right pedal once should move rover  
        }  
    }  
}
```

Test Results: RoverTest.testPressRightPedal ×
Tests passed: 100.00 %
The test passed. (0.206 s)

Test_Case:2

NetBeans IDE interface showing the execution of Test_Case:2. The source code for `RoverTest.java` is displayed, showing a test method `testPressRightPedalForTime()` that tests the `pressRightPedalForTime` method of the `Rover` class. The test results window shows that the test passed successfully with 100.00% success rate.

```
public class RoverTest {  
    public RoverTest() {  
        /**  
         * Test of pressRightPedalForTime method, of class Rover.  
         */  
        @Test  
        public void testPressRightPedalForTime() {  
            //1. Can only press Right Pedal for Time when inside Move Forward Sta  
            r.pressRightPedalForTime(numOfSecondsPressed: 3);  
            assertEquals(expected: "At Rest", actual: r.currentState.name);  
            //2. When at rest, pressing right pedal twice should not move rover  
            r.pressRightPedal(numOfTimesPressed: 2);  
            assertEquals(expected: "At Rest", actual: r.currentState.name);  
            //3. When at rest, pressing right pedal once should move rover  
            r.pressRightPedal(numOfTimesPressed: 1);  
            assertEquals(expected: "Move Forward", actual: r.currentState.name);  
        }  
    }  
}
```

Test Results: RoverTest.testPressRightPedal × RoverTest.testPressRightPedalForTime ×
Tests passed: 100.00 %
The test passed. (0.09 s)

Test_Case:3

NetBeans IDE interface showing the execution of Test_Case:3. The source code for `RoverTest.java` is displayed, showing a test method `testPressLeftPedal()` that tests the `pressLeftPedal` method of the `Rover` class. The test results window shows that the test passed successfully with 100.00% success rate.

```
public class RoverTest {  
    public RoverTest() {  
        /**  
         * Test of pressLeftPedal method, of class Rover.  
         */  
        @Test  
        public void testPressLeftPedal() {  
            //1. Left pedal can only be used when in Move Backward state  
            r.pressLeftPedal(numOfTimesPressed: 1);  
            assertEquals(expected: "At Rest", actual: r.currentState.name);  
            //2. Transition to Move Backward state to access left pedal by pressing  
            r.pressLeftPedalForTime(numOfSecondsPressed: 5);  
            assertEquals(expected: "Move Backward", actual: r.currentState.name);  
            //3. Left pedal when pressed in Move Backward state, rover transitions  
            r.pressLeftPedal(numOfTimesPressed: 2);  
            assertEquals(expected: "Decelerate", actual: r.currentState.subState);  
            //4. Left pedal when pressed in Decelrate subState, rover transitions  
            r.pressLeftPedal(numOfTimesPressed: 1);  
            assertEquals(expected: "Constant Speed", actual: r.currentState.subState);  
        }  
    }  
}
```

Test Results: RoverTest.testPressRightPedal × RoverTest.testPressRightPedalForTime × RoverTest.testPressLeftPedal ×
Tests passed: 100.00 %
The test passed. (0.093 s)

Test_Case:4

```
87
88
89  /**
90   * Test of pressLeftPedalForTime method, of class Rover.
91   */
92  @Test
93  public void testPressLeftPedalForTime() {
94      //1. Rover should always start in "At Rest" state
95      assertEquals(expected: "At Rest", actual:r.currentState.name);
96
97      //2. When at rest, pressing left pedal for 5 sec
98      r.pressLeftPedalForTime(numOfSecondsPressed:5);
99      assertEquals(expected: "Move Backward", actual:r.currentState.name);
100
101      //3. When at Move Backward state, pressing left pedal for 3 sec to move r
102      r.pressLeftPedalForTime(numOfSecondsPressed:3);
103      assertEquals(expected: "Constant Speed", actual:r.currentState.subState);
104
105      //3. When at Move Backward state, pressing left pedal for 3 sec to accele
106      r.pressLeftPedalForTime(numOfSecondsPressed:3);
107      assertEquals(expected: "Accelerate", actual:r.currentState.subState);
108  }
```

Test Results

RoverTest.testPressRightPedal × RoverTest.testPressRightPedalForTime × RoverTest.testPressLeftPedal × RoverTest.test

Tests passed: 100.00 %

The test passed. (0.105 s)

Test_Case:5

```
107
108
109
110
111  /**
112   * Test of printStateAndSubState method, of class Rover.
113   */
114  @Test
115  public void testPrintStateAndSubState() {
116  }
117
118  /**
119   * Test of main method, of class Rover.
120   */
121  @Test
122  public void testMain() {
123  }
124
125  }
```

Test Results

RoverTest.testPrintStateAndSubState ×

Tests passed: 100.00 %

The test passed. (0.118 s)

Test_Case:6

```
107
108
109
110
111  /**
112   * Test of printStateAndSubState method, of class Rover.
113   */
114  @Test
115  public void testPrintStateAndSubState() {
116  }
117
118  /**
119   * Test of main method, of class Rover.
120   */
121  @Test
122  public void testMain() {
123  }
124
125  }
```

Test Results

RoverTest.testPrintStateAndSubState × RoverTest.testMain ×

Tests passed: 100.00 %

The test passed. (0.125 s)

JAVA DOC:

Singleton pattern

PACKAGE

CLASS

USE

TREE

INDEX

HELP

PACKAGE: DESCRIPTION | RELATED PACKAGES | CLASSES AND INTERFACES

SEARCH

Search

X

Unnamed Package

Classes

Class	Description
SingletonPatternDemo	Ensures only a single instance of the class exists within the application.
SingleObject	Implements the Singleton pattern ensuring a single instance of the class.

Facade design pattern

PACKAGE

CLASS

USE

TREE

INDEX

HELP

PACKAGE: DESCRIPTION | RELATED PACKAGES | CLASSES AND INTERFACES

SEARCH

Search

X

Unnamed Package

All Classes and Interfaces

Interfaces

Classes

Class	Description
Circle	Represents a Circle implementing the Shape interface.
FacadePatternDemo	Demonstrate the implementation and usage of facade design pattern.
Rectangle	Represents a Rectangle implementing the Shape interface.
Shape	This interface that provides draw method for different shapes.
ShapeMaker	Acts as a Facade for creating and interacting with different shape objects.
Square	Represents a Square implementing the Shape interface.

Factory Design Pattern

PACKAGE

CLASS

USE

TREE

INDEX

HELP

PACKAGE: DESCRIPTION | RELATED PACKAGES | CLASSES AND INTERFACES

SEARCH

Unnamed Package

All Classes and Interfaces

Interfaces

Classes

Class	Description
Circle	Represents a Circle implementing the Shape interface.
FactoryPatternDemo	Demonstrate the implementation and usage of factory design pattern
Rectangle	Represents a Rectangle implementing the Shape interface.
Shape	This interface that provides draw method for different shapes.
ShapeFactory	ShapeFactory acts as a Factory for creating and interacting with different shape objects.
Square	Represents a Square implementing the Shape interface.

StateDesignPattern Moon Rover

PACKAGE

CLASS

USE

TREE

INDEX

HELP

PACKAGE: DESCRIPTION | RELATED PACKAGES | CLASSES AND INTERFACES

SEARCH

Unnamed Package

Classes

Class	Description
AtRest	Represents the resting state of the Rover.
MoveBackward	Represents the state of the Rover while moving backward.
MoveForward	Represents the state of the Rover while moving forward.
Rover	Represents a rover entity with various states and behaviors.
State	Represents the abstract concept of a state for the Rover.