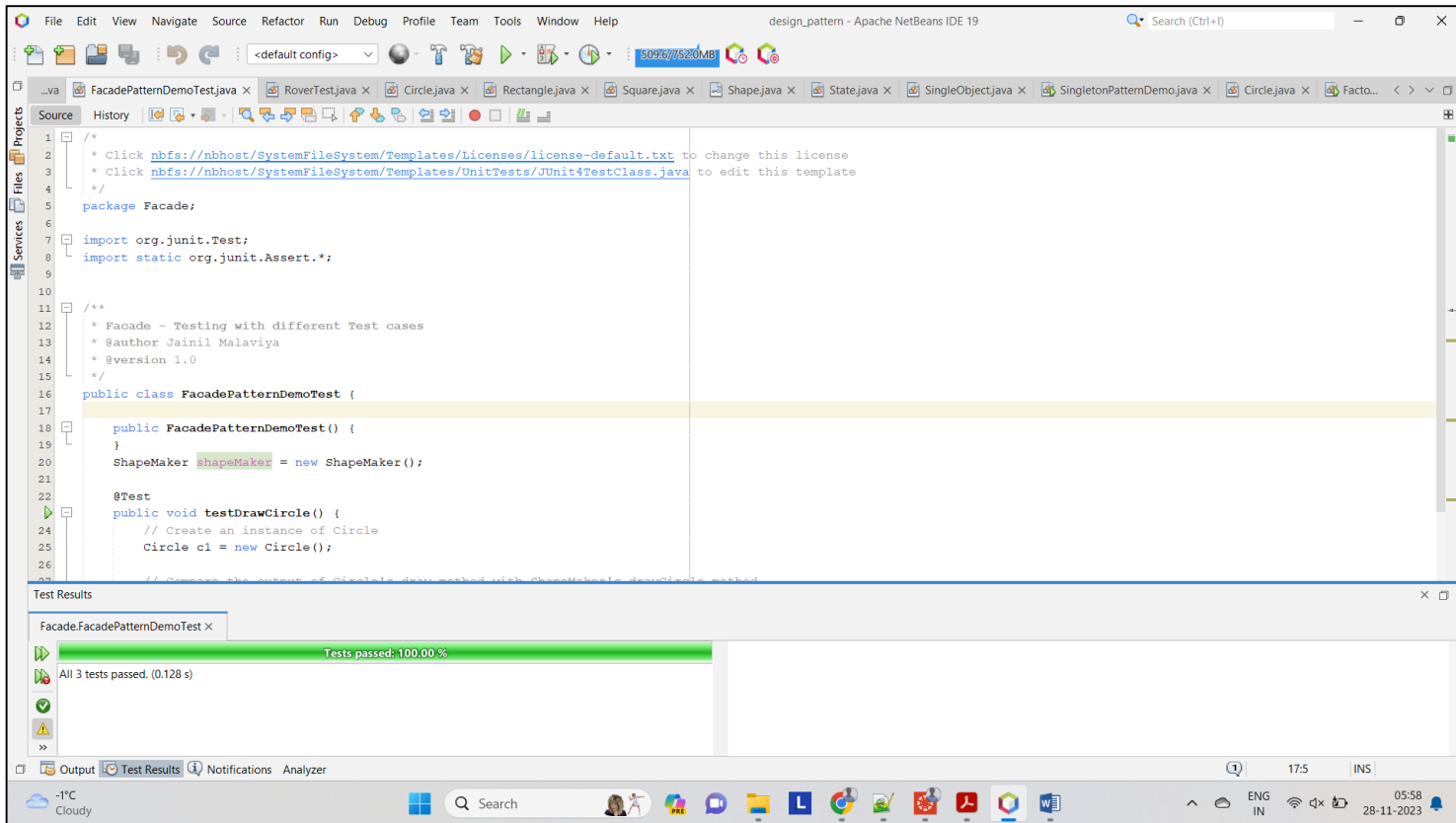


Assignment 4 - Design Patterns

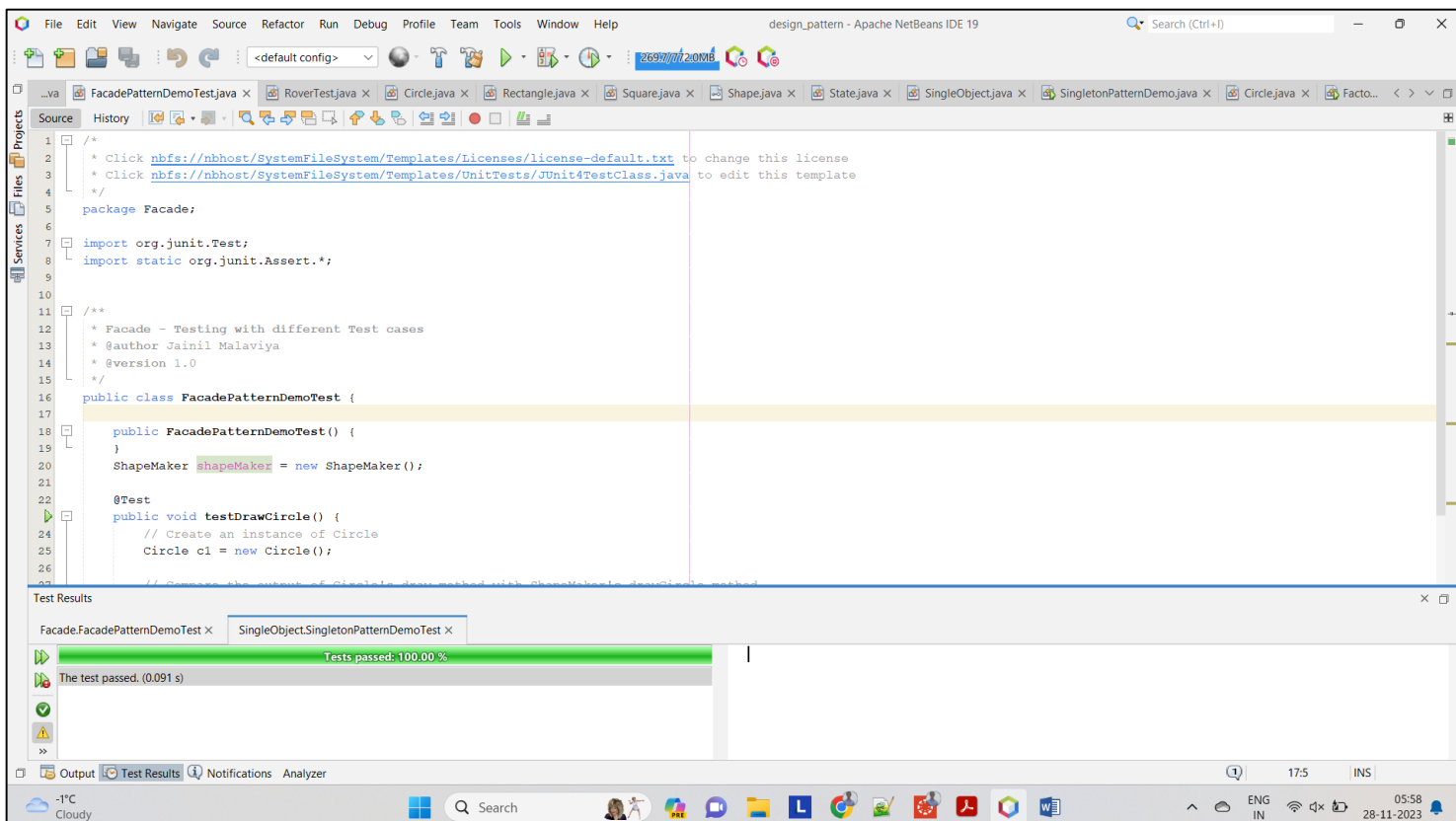
❖ Screenshots

■ Test Cases

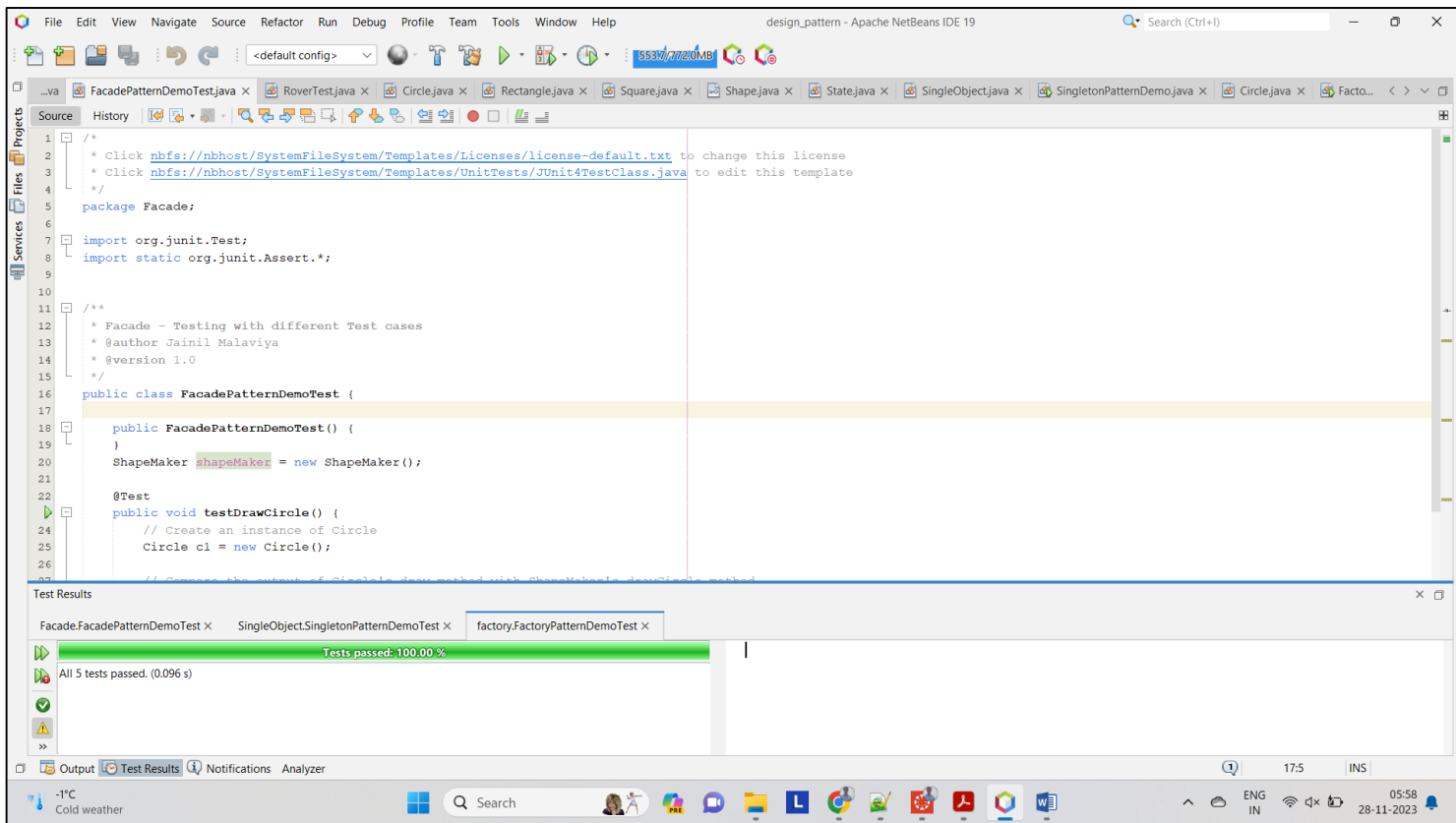
1. Façade



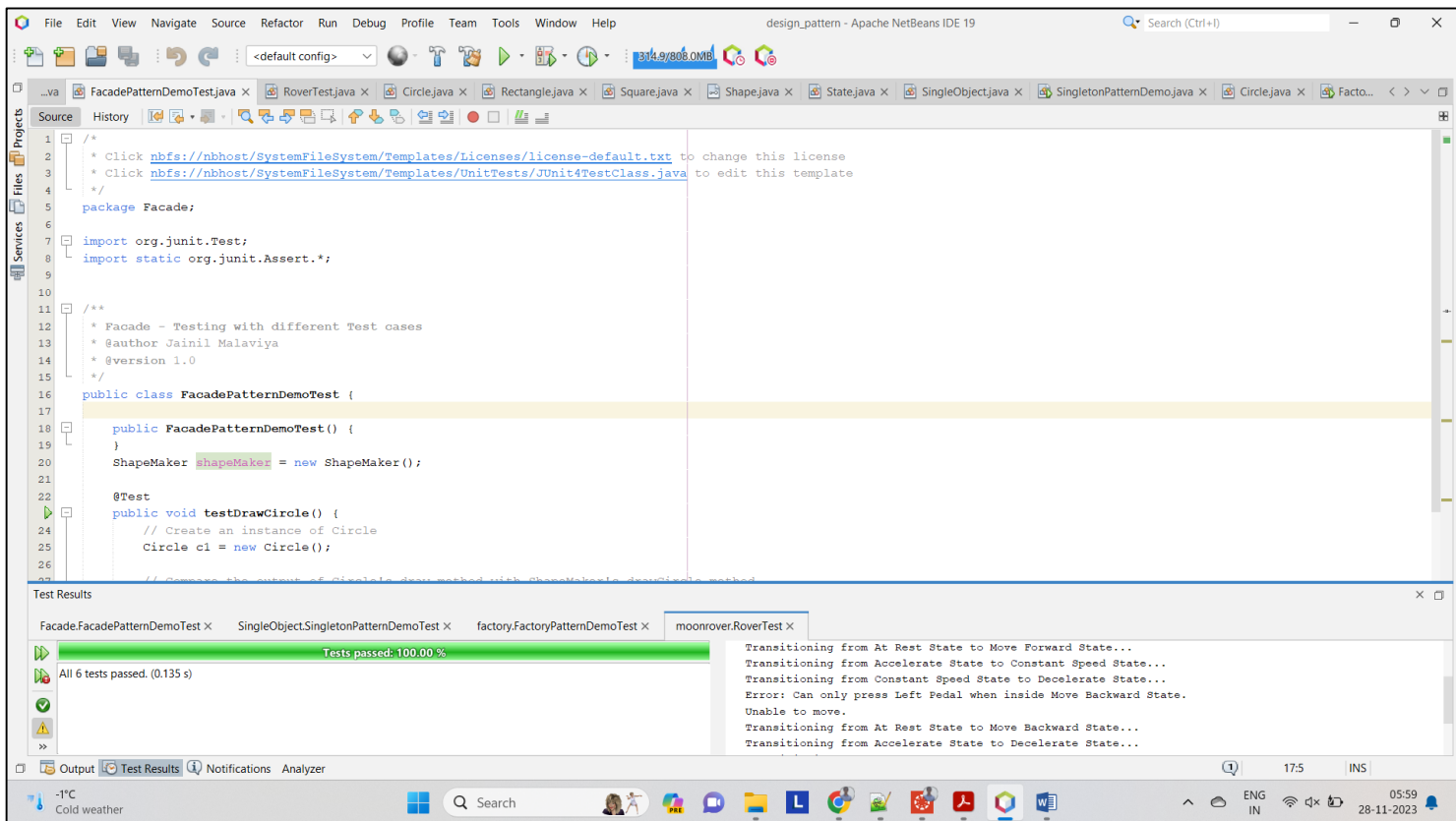
2. Single object



3. Factory



4. Moon Rover



Java Doc

1. Façade

The screenshot shows the JavaDoc package summary for the Facade design pattern. The browser address bar indicates the file path: D:/NEU/INFO5100/Assignment/4/design_pattern/dist/javadoc/Facade/package-summary.html. The page has a navigation bar with tabs: OVERVIEW, PACKAGE (selected), CLASS, USE, TREE, INDEX, and HELP. Below the navigation bar, there's a search bar and a link to the package description. The main content area is titled "Package Facade" and shows the package name "package Facade". There are three tabs: "All Classes and Interfaces" (selected), "Interfaces", and "Classes". A table lists the classes and interfaces in the package:

Class	Description
Circle	Circle class which implements Shape interface.
FacadePatternDemo	Facade design pattern example.
Rectangle	Rectangle class which implements Shape interface.
Shape	Shape interface that provides draw method for different shapes.
ShapeMaker	ShapeMaker class that encapsulates different shapes.
Square	Square class which implements Shape interface.

2. Single object

The screenshot shows the JavaDoc package summary for the SingleObject design pattern. The browser address bar indicates the file path: D:/NEU/INFO5100/Assignment/4/design_pattern/dist/javadoc/SingleObject/package-summary.html. The page has a navigation bar with tabs: OVERVIEW, PACKAGE (selected), CLASS, USE, TREE, INDEX, and HELP. Below the navigation bar, there's a search bar and a link to the package description. The main content area is titled "Package SingleObject" and shows the package name "package SingleObject". There is one tab: "Classes". A table lists the classes in the package:

Class	Description
SingleObject	The Singleton Design Pattern is a creational pattern that ensures a class has only one instance and provides a global point of access to that instance.
SingletonPatternDemo	SingletonPatternDemo is a class that demonstrates the Factory design pattern in action.

3. Factory

factory

D:\NEU\INFO5100\Assignment\4\design_pattern\dist\javadoc\factory\package-summary.html

Lenovo Pearson VUE - Home 22 How to Score 79+... Gmail YouTube Maps HOW TO PY4E - Python for E... 17 Free Amazing Re... Fake IT - Fake the... Random Credit Car... Other favorites

OVERVIEW PACKAGE CLASS USE TREE INDEX HELP

PACKAGE: DESCRIPTION | RELATED PACKAGES | CLASSES AND INTERFACES

SEARCH Search

Package factory

package factory

All Classes and Interfaces Interfaces Classes

Class	Description
Circle	Circle class which implements Shape interface.
FactoryPatternDemo	ShapeFactoryDemo is a class that demonstrates the Factory design pattern in action.
Rectangle	Rectangle class which implements Shape interface.
Shape	Shape interface that provides draw method for different shapes.
ShapeFactory	FactoryPattern is a class that demonstrates the Factory design pattern in action.
Square	Shape interface that provides draw method for different shapes.

-1°C Cloudy

Search

ENG IN 05:56 28-11-2023

4. Moon Rover

moonrover

D:\NEU\INFO5100\Assignment\4\design_pattern\dist\javadoc\moonrover\package-summary.html

Lenovo Pearson VUE - Home 22 How to Score 79+... Gmail YouTube Maps HOW TO PY4E - Python for E... 17 Free Amazing Re... Fake IT - Fake the... Random Credit Car... Other favorites

OVERVIEW PACKAGE CLASS USE TREE INDEX HELP

PACKAGE: DESCRIPTION | RELATED PACKAGES | CLASSES AND INTERFACES

SEARCH Search

Package moonrover

package moonrover

Classes

Class	Description
AtRest	Represents the At Rest state of the Rover
MoveBackward	Represents the Move Backward state of the Rover.
MoveForward	Represents to move forward Rest state of the Rover.
Rover	Rover class - represents a rover.
State	State class - the description of this class

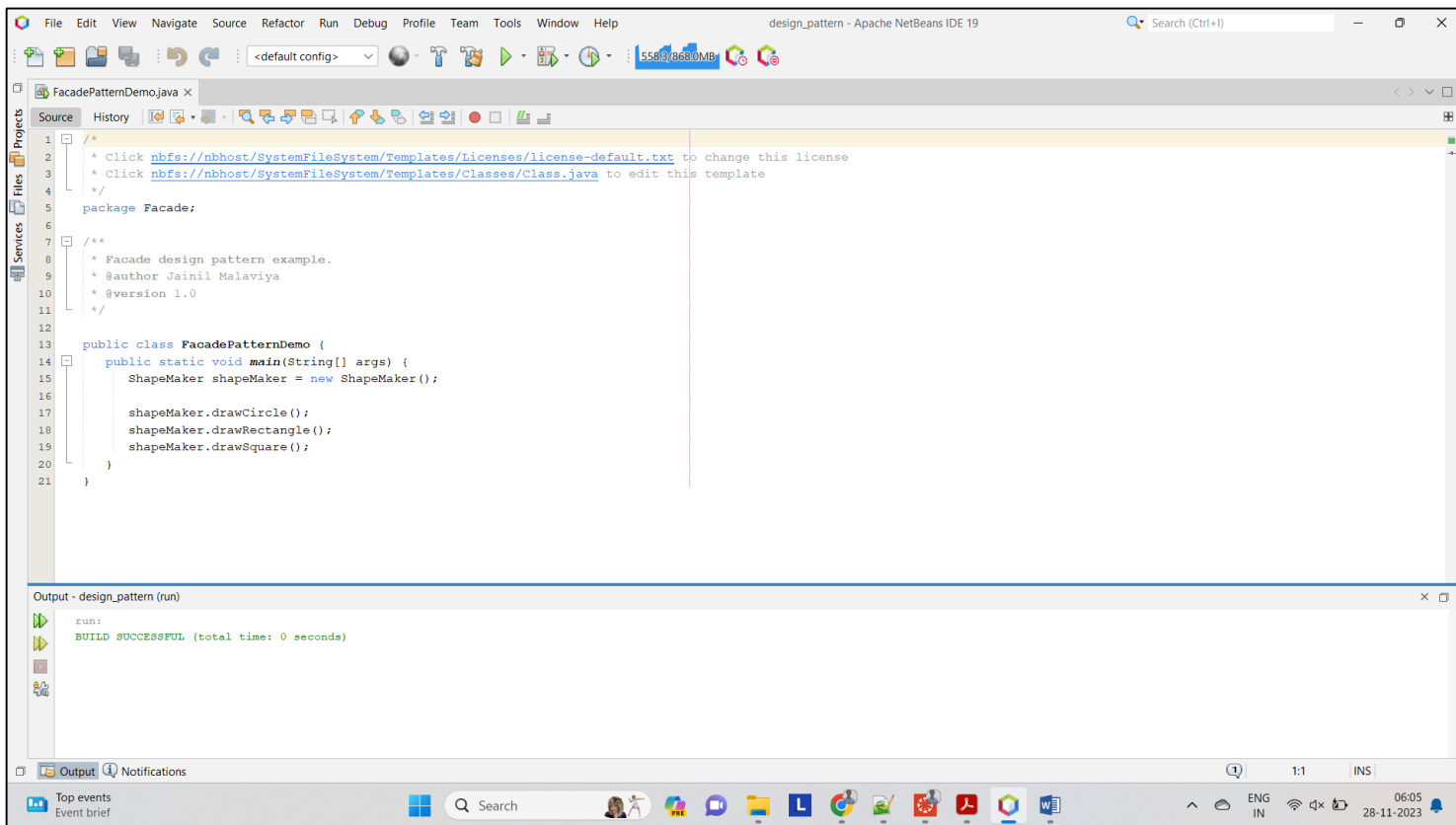
-1°C Cloudy

Search

ENG IN 05:57 28-11-2023

■ Sample Run

1. Façade



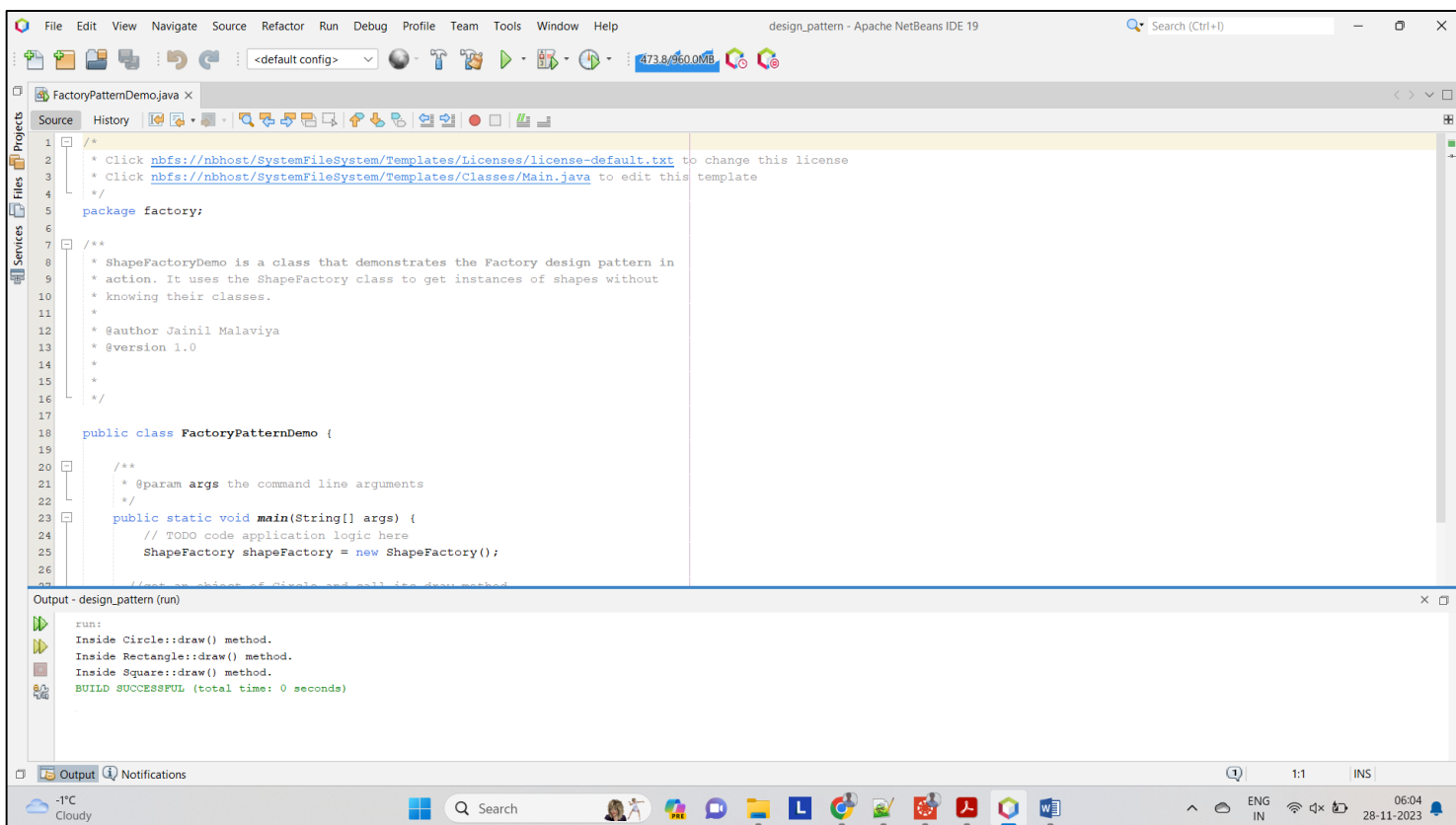
The screenshot shows the Apache NetBeans IDE with the 'FacadePatternDemo.java' file open. The code defines a 'Facade' package and a 'FacadePatternDemo' class. The 'main' method in 'FacadePatternDemo' creates a 'ShapeMaker' object and calls its 'drawCircle()', 'drawRectangle()', and 'drawSquare()' methods. The 'Output' window at the bottom shows the successful execution of the program.

```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package Facade;
6
7
8  /**
9   * Facade design pattern example.
10  * @author Jainil Malaviya
11  * @version 1.0
12  */
13
14  public class FacadePatternDemo {
15      public static void main(String[] args) {
16          ShapeMaker shapeMaker = new ShapeMaker();
17
18          shapeMaker.drawCircle();
19          shapeMaker.drawRectangle();
20          shapeMaker.drawSquare();
21      }
22  }
```

Output - design_pattern (run)

```
run:
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Factory



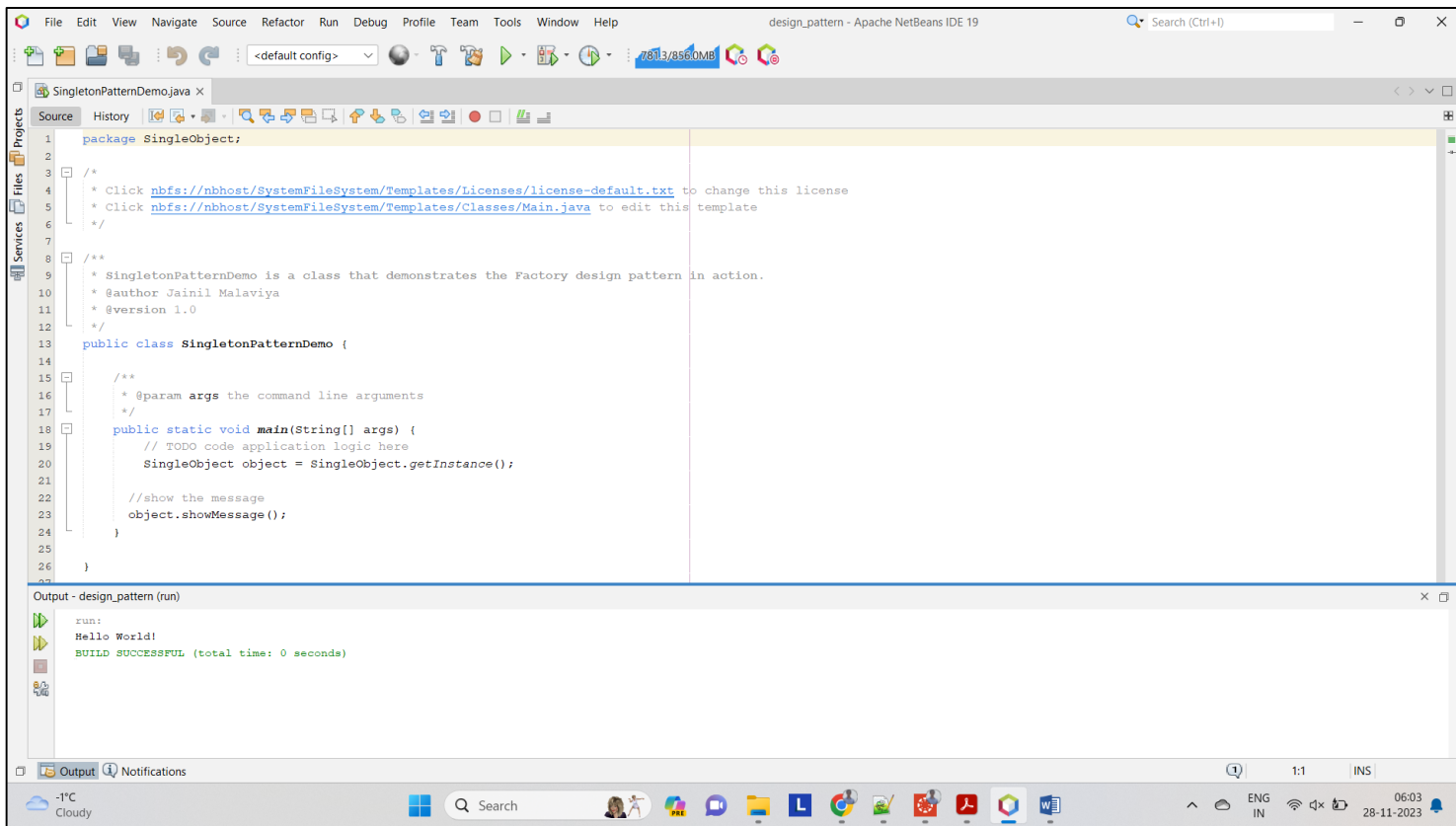
The screenshot shows the Apache NetBeans IDE with the 'FactoryPatternDemo.java' file open. The code defines a 'factory' package and a 'FactoryPatternDemo' class. The 'main' method in 'FactoryPatternDemo' creates a 'ShapeFactory' object and calls its 'drawCircle()', 'drawRectangle()', and 'drawSquare()' methods. The 'Output' window at the bottom shows the successful execution of the program, including the output of the 'draw' methods.

```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
4  */
5  package factory;
6
7
8  /**
9   * ShapeFactoryDemo is a class that demonstrates the Factory design pattern in
10  * action. It uses the ShapeFactory class to get instances of shapes without
11  * knowing their classes.
12  * @author Jainil Malaviya
13  * @version 1.0
14  */
15
16  /**
17   *
18   */
19
20  public class FactoryPatternDemo {
21
22      /**
23       * @param args the command line arguments
24       */
25      public static void main(String[] args) {
26          // TODO code application logic here
27          ShapeFactory shapeFactory = new ShapeFactory();
28
29          //get an object of Circle and call its draw method
30      }
31  }
```

Output - design_pattern (run)

```
run:
Inside Circle::draw() method.
Inside Rectangle::draw() method.
Inside Square::draw() method.
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. Single Object



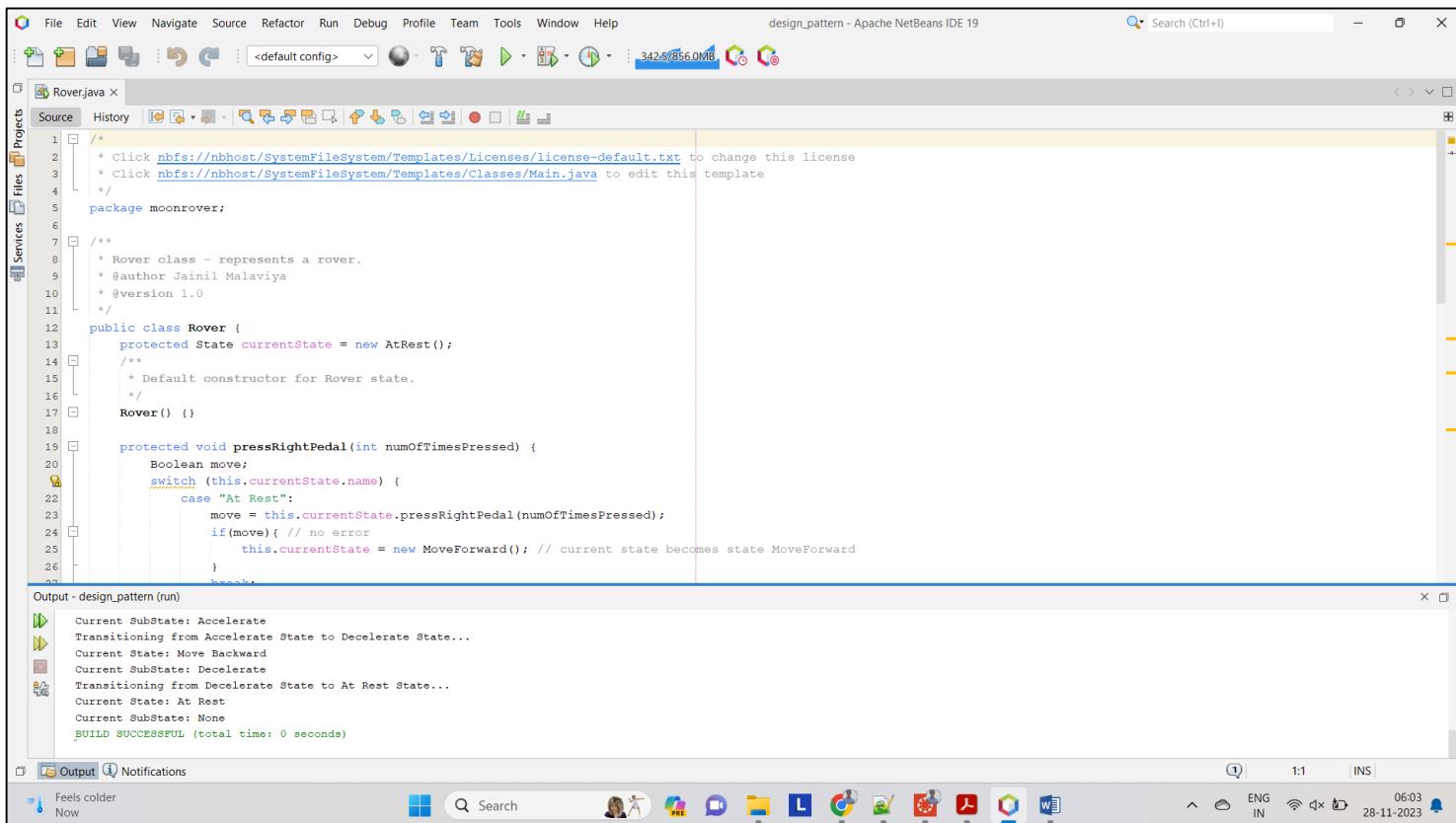
The screenshot shows the NetBeans IDE with the file `SingletonPatternDemo.java` open. The code is as follows:

```
1 package SingletonObject;
2
3 /**
4  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
5  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
6  */
7
8 /**
9  * SingletonPatternDemo is a class that demonstrates the Factory design pattern in action.
10  * @author Jainil Malaviya
11  * @version 1.0
12  */
13 public class SingletonPatternDemo {
14
15     /**
16      * @param args the command line arguments
17      */
18     public static void main(String[] args) {
19         // TODO code application logic here
20         SingletonObject object = SingletonObject.getInstance();
21
22         //show the message
23         object.showMessage();
24     }
25
26 }
```

The output window shows the following text:

```
run:
Hello World!
BUILD SUCCESSFUL (total time: 0 seconds)
```

4. Moon Rover



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

```
1 package moonrover;
2
3 /**
4  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
5  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
6  */
7
8 /**
9  * Rover class - represents a rover.
10  * @author Jainil Malaviya
11  * @version 1.0
12  */
13 public class Rover {
14     protected State currentState = new AtRest();
15     /**
16      * Default constructor for Rover state.
17      */
18     Rover() {}
19
20     protected void pressRightPedal(int numOfTimesPressed) {
21         Boolean move;
22         switch (this.currentState.name) {
23             case "At Rest":
24                 move = this.currentState.pressRightPedal(numOfTimesPressed);
25                 if (move) { // no error
26                     this.currentState = new MoveForward(); // current state becomes state MoveForward
27                 }
28                 break;
29         }
30     }
31 }
```

The output window shows the following text:

```
Current SubState: Accelerate
Transitioning from Accelerate State to Decelerate State...
Current State: Move Backward
Current SubState: Decelerate
Transitioning from Decelerate State to At Rest State...
Current State: At Rest
Current SubState: None
BUILD SUCCESSFUL (total time: 0 seconds)
```