

Session 11

Design Patterns

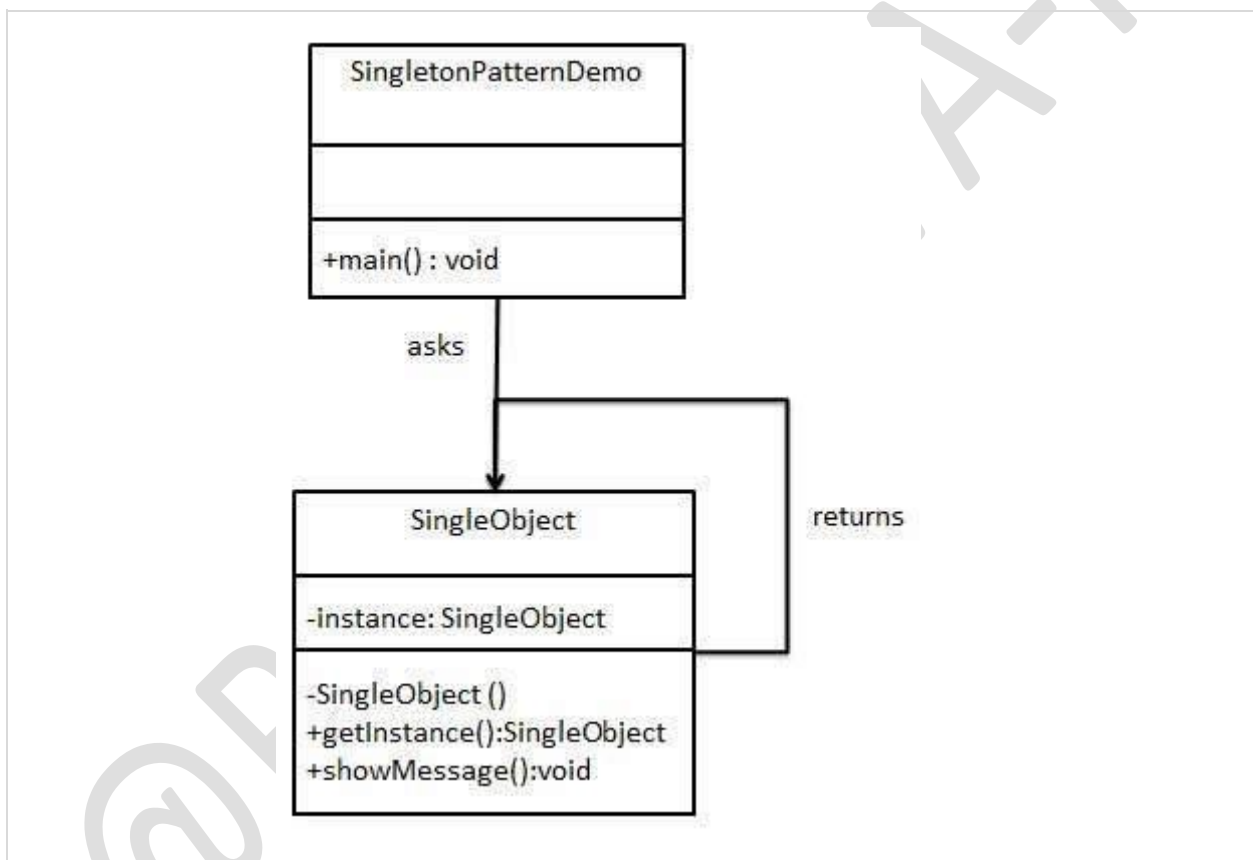
Phần I - Thực hiện trong 120 phút

1. Mục tiêu

- Hiểu biết về Design Pattern.
- Có thể xây dựng và sử dụng một số mẫu thiết kế phổ biến.

2. Thực hiện

Bài thực hành 1: Vận dụng kiến thức về thiết kế Singleton để tạo thực thể đối tượng đảm bảo luôn chỉ có duy nhất một thể hiện của nó được tạo tại bất kỳ thời điểm nào.



Bước 1: Viết class Singleton.java

```
package demo.jp2.lab08;
```

```
/**
```

```
*
```

```
* @authorminhvufc
```

```
*/
```

```
public class SingleObject {

    //create an object of SingleObject
    private static SingleObject instance = new SingleObject();

    //make the constructor private so that this class cannot be
    //instantiated
    private SingleObject() {
    }

    //Get the only object available
    public static SingleObject getInstance() {
        return instance;
    }

    public void showMessage() {
        System.out.println("Hello World!");
    }
}
```

Bước 2: Viết SingletonPatternDemo.java và lấy về đối tượng thực thể duy nhất.

```
package demo.jp2.lab08;

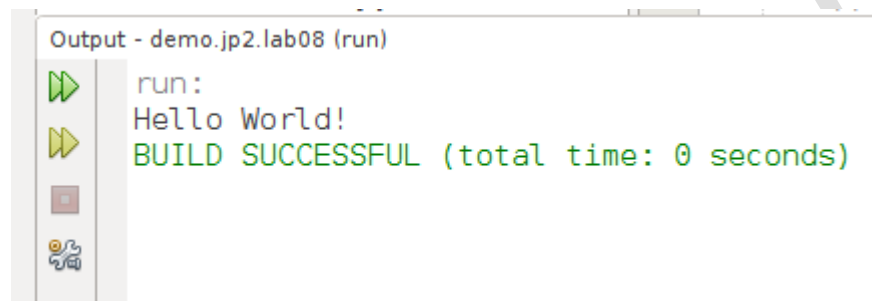
/**
 *
 * @authorminhvuvc
 */
public class SingletonPatternDemo {

    public static void main(String[] args) {
        //illegal construct
    }
}
```

```
//Compile Time Error: The constructor SingleObject() is not visible
//SingleObject object = new SingleObject();
//Get the only object available
SingleObject object = SingleObject.getInstance();

//show the message
object.showMessage();
}
}
```

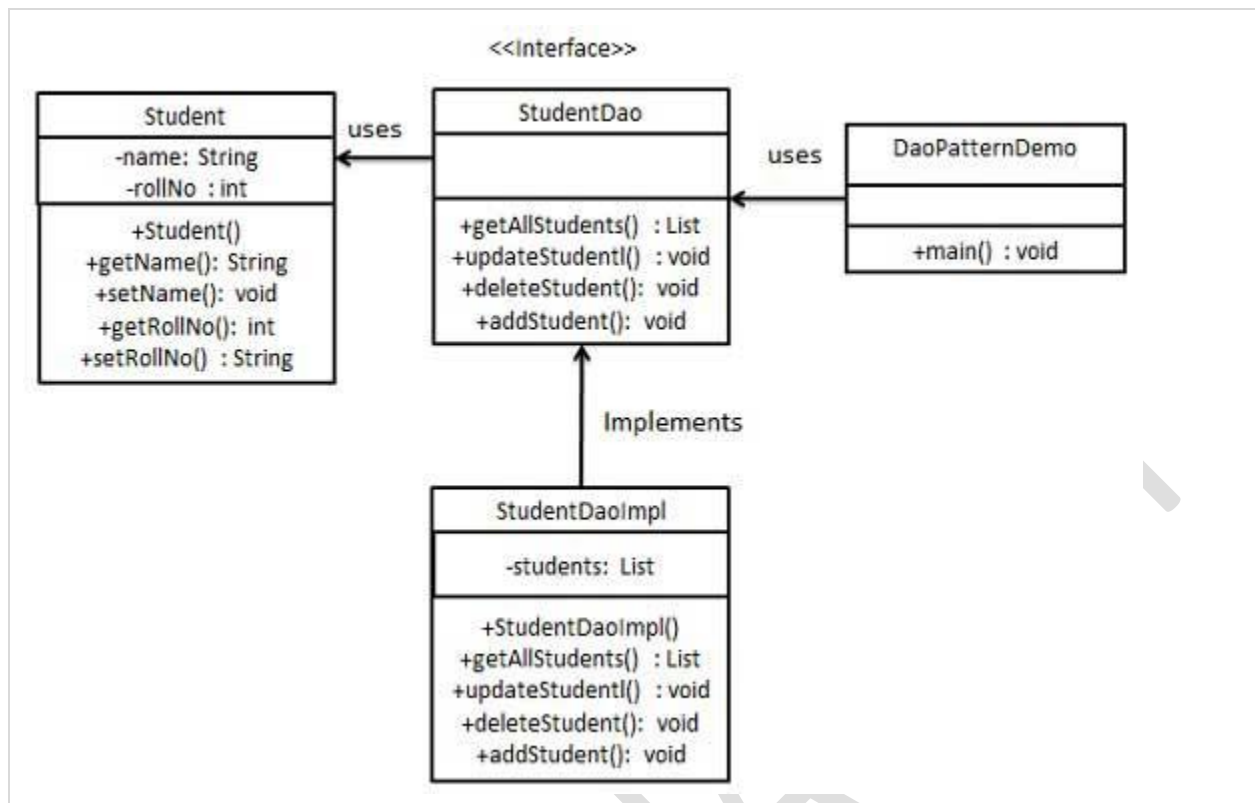
Bước 3: Xem kết quả



The screenshot shows an IDE's output console for a file named 'demo.jp2.lab08 (run)'. On the left, there are icons for running (a green play button), stepping through (a yellow play button), and debugging (a red square). The output text is as follows:

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

Bài thực hành 2: Vận dụng kiến thức về thiết kế DAO để thiết kế module phân tách phần xử lý dữ liệu cấp thấp hoặc nghiệp vụ xử lý nâng cao khỏi tầng giao diện người dùng.



Chú giải: trong thiết kế này, cần xây dựng các thành phần như sau

- Data Access Object Interface: là interface định nghĩa hoạt động tiêu chuẩn thực hiện trên đối tượng thực thể.
- Data Access Object concrete class: là class thực thi interface trên, class này chịu trách nhiệm lấy dữ liệu từ database hoặc thành phần chứa dữ liệu có sẵn khác.
- Model Object or Value Object: là một java class cơ bản chứa các hàm get/set để lưu trữ hoặc lấy dữ liệu sử dụng.

Bước 1: Viết class Student.java

```
package demo.jp2.lab08;
```

```
/**
```

```
*
```

```
* @authorminhvuvc
```

```
*/  
public class Student {  
  
    private String name;  
    private int rollNo;  
  
    Student(String name, int rollNo) {  
        this.name = name;  
        this.rollNo = rollNo;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public int getRollNo() {  
        return rollNo;  
    }  
  
    public void setRollNo(int rollNo) {  
        this.rollNo = rollNo;  
    }  
}
```

Bước 2: Viết interface Data Access Object (DAO)

```
package demo.jp2.lab08;  
  
import java.util.List;
```

```
/**
 *
 * @authorminhvuafc
 */
public interface StudentDao {

    public List<Student> getAllStudents();

    public Student getStudent(int rollNo);

    public void updateStudent(Student student);

    public void deleteStudent(Student student);
}
```

Bước 3: Viết StudentDaoImpl.java thực thi interface trên.

```
package demo.jp2.lab08;

import java.util.ArrayList;
import java.util.List;

/**
 *
 * @authorminhvuafc
 */
public class StudentDaoImpl implements StudentDao {

    //list is working as a database
    List<Student> students;
}
```

```
public StudentDaoImpl() {  
    students = new ArrayList<Student>();  
    Student student1 = new Student("Robert", 0);  
    Student student2 = new Student("John", 1);  
    students.add(student1);  
    students.add(student2);  
}  
  
@Override  
public void deleteStudent(Student student) {  
    students.remove(student.getRollNo());  
    System.out.println("Student: Roll No " + student.getRollNo() + ", deleted from  
database");  
}  
  
//retrive list of students from the database  
@Override  
public List<Student>getAllStudents() {  
    return students;  
}  
  
@Override  
public Student getStudent(int rollNo) {  
    return students.get(rollNo);  
}  
  
@Override  
public void updateStudent(Student student) {  
    students.get(student.getRollNo()).setName(student.getName());  
    System.out.println("Student: Roll No " + student.getRollNo() + ", updated in the  
database");  
}
```

```
}  
}
```

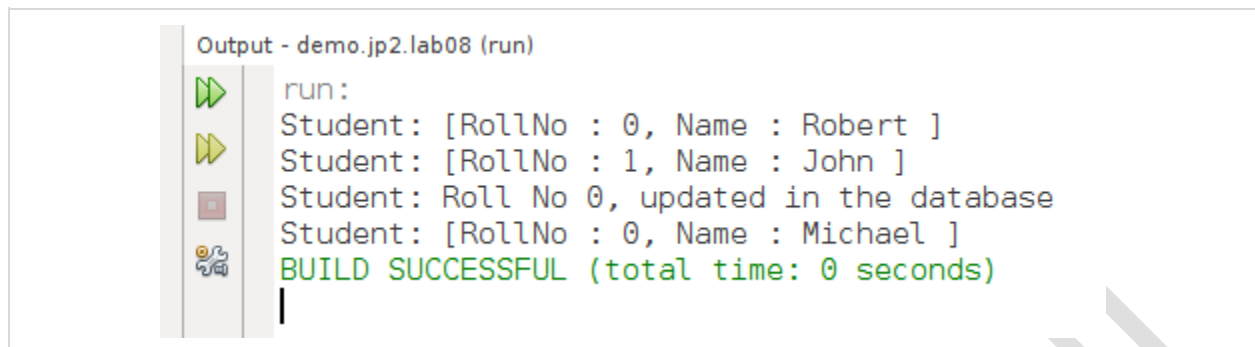
Bước 4: Viết class DaoPatternDemo.java gọi và sử dụng đối tượng DAO trên.

```
package demo.jp2.lab08;  
  
/**  
 *  
 * @authorminhvufc  
 */  
public class DaoPatternDemo {  
  
    public static void main(String[] args) {  
        StudentDao studentDao = new StudentDaoImpl();  
  
        //print all students  
        for (Student student : studentDao.getAllStudents()) {  
            System.out.println("Student: [RollNo : " + student.getRollNo() + ", Name : "  
+ student.getName() + " ]");  
        }  
  
        //update student  
        Student student = studentDao.getAllStudents().get(0);  
        student.setName("Michael");  
        studentDao.updateStudent(student);  
  
        //get the student  
        studentDao.getStudent(0);  
        System.out.println("Student: [RollNo : " + student.getRollNo() + ", Name : " +  
student.getName() + " ]");  
    }  
}
```



```
}
```

Bước 5: Xem kết quả

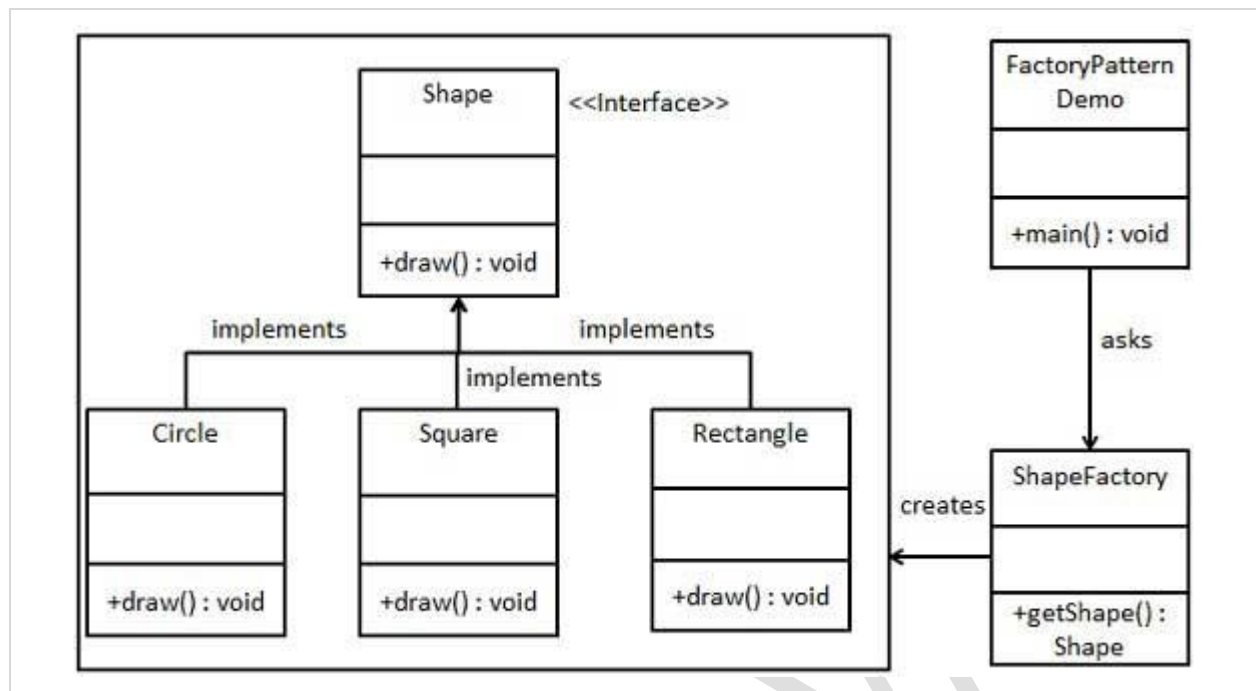


```
Output - demo.jp2.lab08 (run)
run:
Student: [RollNo : 0, Name : Robert ]
Student: [RollNo : 1, Name : John ]
Student: Roll No 0, updated in the database
Student: [RollNo : 0, Name : Michael ]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Chú giải: Mô hình thiết kế DAO được sử dụng khi mong muốn xây dựng một lớp đối tượng phụ trách việc quản lý một thực thể hoặc một nhóm thực thể hay nghiệp vụ phức tạp. Những thao tác xử lý dù đơn giản hay phức tạp được phân tách rõ ràng, tạo điều kiện nâng cấp hoặc sửa chữa phần mềm sau này trở lên dễ dàng.

Bài thực hành 3: Vận dụng kiến thức về mô hình Factory để thiết kế module người dùng truyền vào tham số hình vẽ thích hợp thì sẽ nhận về được giá trị mong muốn.

Chú giải: Mô hình thiết kế Factory (được hiểu như nhà máy). Với mô hình này, có thể thiết kế cho module mà cách thức xử lý được ẩn đi hoàn toàn, chỉ cần quan tâm "nguyên vật liệu" đầu vào là gì thì sản phẩm ra sẽ như mong muốn.



Bước 1: Tạo interface Shape.java

```
package demo.jp2.lab08;
```

```
/**
 *
 * @author minhvufc
 */
public interface Shape {

    void draw();
}
```

Bước 2: Tạo các class cụ thể hóa interface trên

```
public class Rectangle implements Shape {

    @Override
    public void draw() {
        System.out.println("Inside Rectangle::draw() method.");
    }
}
```

```

}

public class Square implements Shape {

    @Override
    public void draw() {
        System.out.println("Inside Square::draw() method.");
    }
}

public class Circle implements Shape {

    @Override
    public void draw() {
        System.out.println("Inside Circle::draw() method.");
    }
}

```

Bước 3: Tạo class Factory trả về hình vẽ (Shape) tương ứng với tham số truyền vào.

```

package demo.jp2.lab08;

/**
 *
 * @authorminhvufc
 */
public class ShapeFactory {

    //use getShape method to get object of type shape
    public Shape getShape(String shapeType) {
        if (shapeType == null) {
            return null;
        }
    }
}

```

```
}  
if (shapeType.equalsIgnoreCase("CIRCLE")) {  
    returnnew Circle();  
  
    } elseif (shapeType.equalsIgnoreCase("RECTANGLE")) {  
        returnnew Rectangle();  
  
    } elseif (shapeType.equalsIgnoreCase("SQUARE")) {  
        returnnew Square();  
    }  
  
    returnnull;  
}  
}
```

Bước 4: Xây dựng class FactoryPatternDemo.java, truyền các tham số vào đối tượng Factory để lấy về hình mong muốn.

```
package demo.jp2.lab08;  
  
/**  
 *  
 * @authorminhvufc  
 */  
public class FactoryPatternDemo {  
  
    public static void main(String[] args) {  
        ShapeFactory shapeFactory = new ShapeFactory();  
  
        //get an object of Circle and call its draw method.  
        Shape shape1 = shapeFactory.getShape("CIRCLE");  
    }  
}
```

```
//call draw method of Circle
shape1.draw();

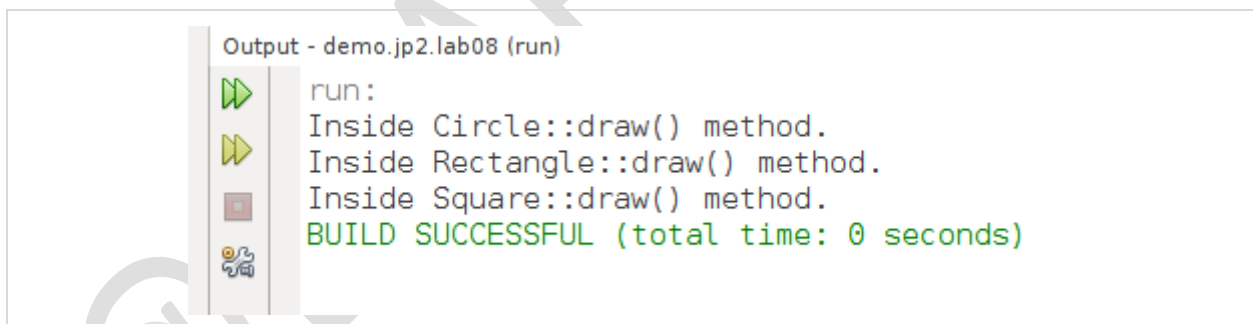
//get an object of Rectangle and call its draw method.
Shape shape2 = shapeFactory.getShape("RECTANGLE");

//call draw method of Rectangle
shape2.draw();

//get an object of Square and call its draw method.
Shape shape3 = shapeFactory.getShape("SQUARE");

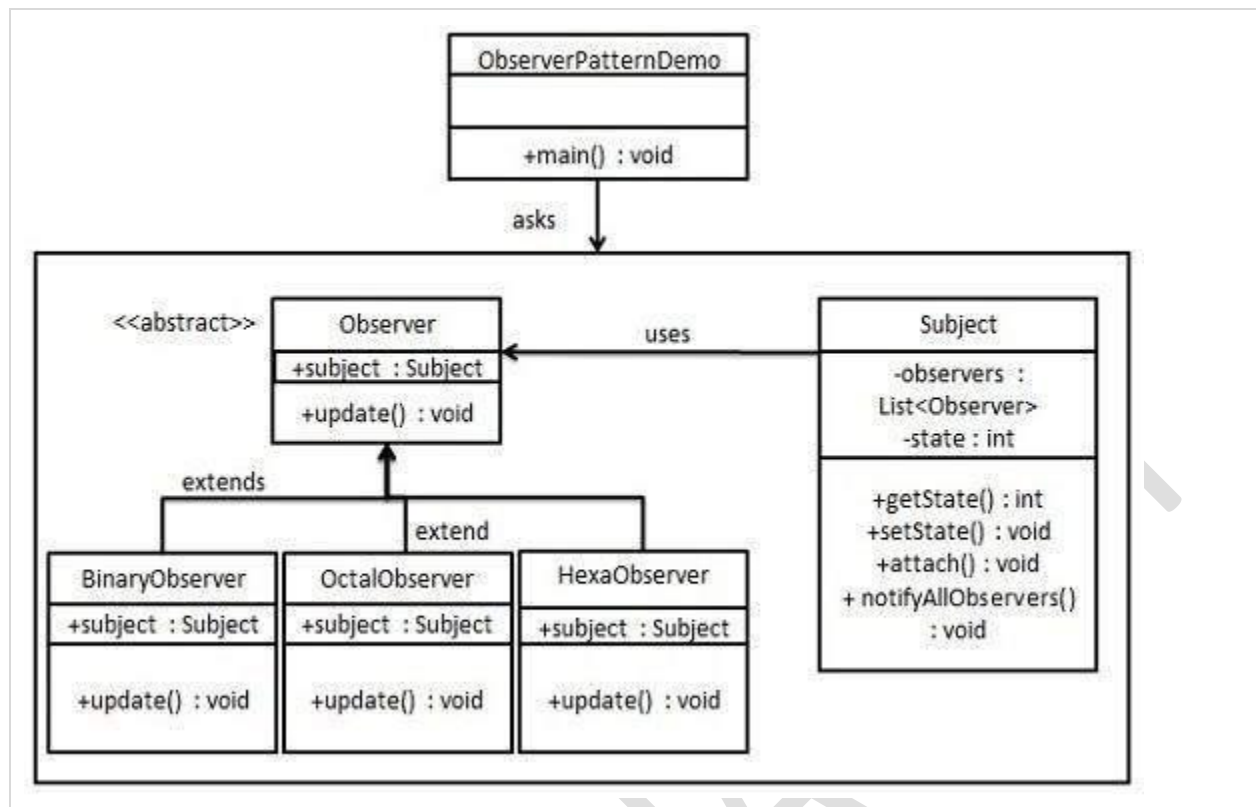
//call draw method of circle
shape3.draw();
}
}
```

Bước 5: Xem kết quả



Bài thực hành 4: Vận dụng hiểu biết về mẫu thiết kế Observer, viết module khi nhập 1 số nhị phân thì lập tức chuyển đổi sang bát phân và thập lục phân, tương tự, nhập bát phân hoặc thập lục phân thì chuyển đổi ra giá trị 2 hệ số còn lại.

Chú giải: Mô hình thiết kế Observer giải quyết bài toán nhiều đối tượng thực thể có liên quan tới nhau. Khi một thực thể xảy ra biến cố thay đổi, ngay lập tức thay đổi đó được thông báo tới những thực thể liên quan một cách tự động.



Bước 1: Tạo class Subject.java

```
public class Subject {

}
```

Bước 2: Tạo class Observer.java

```
public abstract class Observer {

    protected Subject subject;

    public abstract void update();

}
```

Bước 3: Viết chi tiết thân class Subject

```
public class Subject {

    private List<Observer> observers = new ArrayList<Observer>();

}
```

```
private int state;

public int getState() {
    return state;
}

public void setState(int state) {
    this.state = state;
    notifyAllObservers();
}

public void attach(Observer observer) {
    observers.add(observer);
}

public void notifyAllObservers() {
    for (Observer observer : observers) {
        observer.update();
    }
}
}
```

Chú giải:

- Trong Subject class khai báo một danh sách List các observer (đối tượng quan sát), hàm attach() có trách nhiệm thêm đối tượng quan sát và hàm notifyAllObservers() có trách nhiệm thông báo tới tất cả đối tượng quan sát nhận đăng ký.
- Trong class Observer có khai báo đối tượng Subject tham chiếu.

Bước 4: Tạo các lớp quan sát kế thừa Observer

```
package demo.jp2.lab08.observer;

/**
 *
 * @authorminhvuafc
 */
public class BinaryObserver extends Observer {

    public BinaryObserver(Subject subject) {
        this.subject = subject;
        this.subject.attach(this);
    }

    @Override
    public void update() {
        System.out.println("Binary String: " +
Integer.toBinaryString(subject.getState()));
    }
}

package demo.jp2.lab08.observer;

/**
 *
 * @authorminhvuafc
 */
public class OctalObserver extends Observer {

    public OctalObserver(Subject subject) {
        this.subject = subject;
        this.subject.attach(this);
    }
}
```



```

    }

    @Override
    public void update() {
        System.out.println("Octal String: " + Integer.toOctalString(subject.getState()));
    }
}

package demo.jp2.lab08.observer;

/**
 *
 * @authorminhvuvc
 */
public class HexaObserver extends Observer {

    public HexaObserver(Subject subject) {
        this.subject = subject;
        this.subject.attach(this);
    }

    @Override
    public void update() {
        System.out.println("Hex String: " +
Integer.toHexString(subject.getState()).toUpperCase());
    }
}

```

Bước 5: Tạo class ObserverPatternDemo.java

```

package demo.jp2.lab08.observer;

```

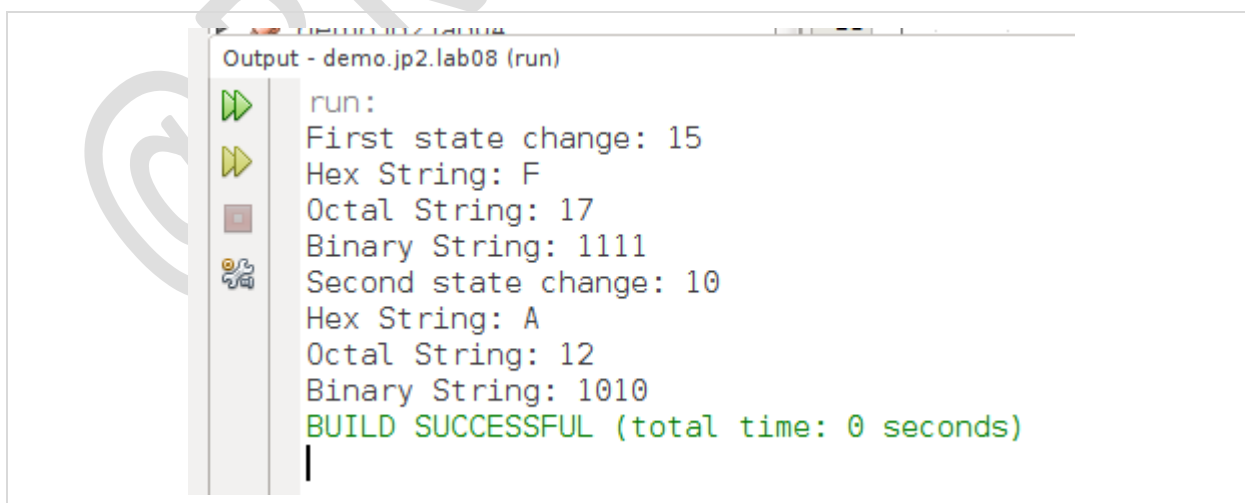
```
/**
 *
 * @authorminhvuvc
 */
public class ObserverPatternDemo {

    public static void main(String[] args) {
        Subject subject = new Subject();

        new HexaObserver(subject);
        new OctalObserver(subject);
        new BinaryObserver(subject);

        System.out.println("First state change: 15");
        subject.setState(15);
        System.out.println("Second state change: 10");
        subject.setState(10);
    }
}
```

Bước 6: Xem kết quả



```
Output - demo.jp2.lab08 (run)
run:
First state change: 15
Hex String: F
Octal String: 17
Binary String: 1111
Second state change: 10
Hex String: A
Octal String: 12
Binary String: 1010
BUILD SUCCESSFUL (total time: 0 seconds)
```

Phần II - Bài tập tự làm

1. Áp dụng mẫu **Signleton** (mẫu FA :D) để thiết kế lớp phụ trách kết nối thao tác tới CSDL.
2. Áp dụng mẫu **DAO (Data Access Object)** để giải bài toán xây dựng ứng dụng CRUD quản lý nhân viên. Gợi ý CSDL:

tblNhanVien
int id
nvarchar(64) name
int yearOfBirth
varchar(256) address
varchar(11) phone
int salary

3. Áp dụng mẫu **Factory** để thiết kế module xây dựng sản phẩm của nhà máy Honda Việt Nam. Nhà máy có các sản phẩm dòng xe
SH 150 - Mã **sh150**
SH Mode - Mã **shmod**
Lead 125 - Mã **lead125**
Vision 125 - Mã **vision**
Ví dụ khách hàng chọn mã shmod thì phải in ra màn hình thông số xe (bịa đại lên :))).
4. Áp dụng mẫu **Observer** để thiết kế module tính giá trị quy đổi tiền tệ từ VND sang các ngoại tệ khác. Ví dụ nhập 21.000 sẽ quy đổi ra 1 USD hay 0,8 Euro.... (bài này cũng thoải mái chém các loại ngoại tệ).
5. Vận dụng kiến thức mẫu **Strategy** để thiết kế máy tính đơn giản yêu cầu người dùng nhập 2 số và phép tính cần tính (Cộng, Trừ, Nhân, Chia).