



TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY



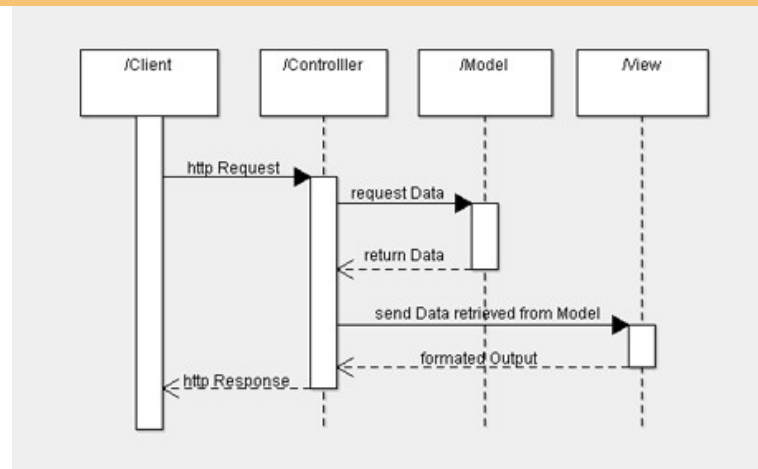
# Phân tích thiết kế hệ thống

## Một số mẫu thiết kế



# Nội dung

- Mẫu thiết kế
- Mẫu Factory
- Mẫu Observer
- Mẫu MVC



# Mẫu thiết kế

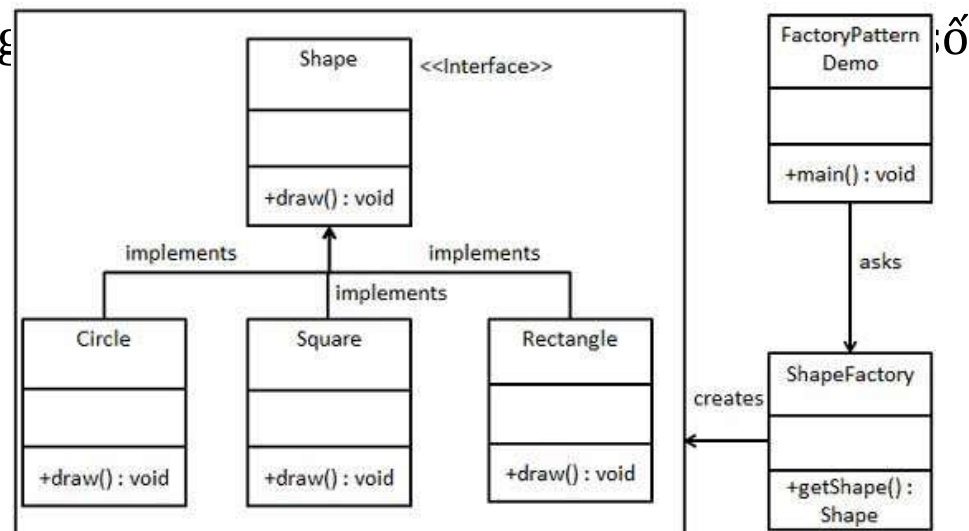
- Mẫu thiết kế (design pattern) là giải pháp thiết kế của các cá nhân/hãng phần mềm đưa ra cho các vấn đề thường xuyên gặp phải khi phát triển phần mềm.
- Mỗi mẫu thiết kế thường bao gồm:
  - Tên mẫu
  - Vấn đề cần giải quyết
  - Giải pháp
  - Kết quả

## Mẫu thiết kế

- Có nhiều mẫu, các mẫu cũng liên tục được cải tiến. Có thể nhóm các mẫu vào 3 nhóm:
  - Nhóm mẫu liên quan tới việc tạo đối tượng
  - Nhóm mẫu liên quan tới cấu trúc hệ thống
  - Nhóm mẫu liên quan tới hành vi

# Mẫu Factory

- Tên: Factory Pattern
- Vấn đề cần giải quyết: Tạo đối tượng của lớp phụ thuộc vào dữ liệu đầu vào (ẩn dấu cách tạo)
- Giải pháp: Tạo lớp Factory
- Thí dụ: tạo đối tượng



# Mẫu Factory

```
public interface Shape {  
    void draw();  
}
```

```
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.");  
    }  
}
```

# Mẫu Factory

```
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")){  
            return new Circle();  
        } else if(shapeType.equalsIgnoreCase("RECTANGLE")){  
            return new Rectangle();  
        } else if(shapeType.equalsIgnoreCase("SQUARE")){  
            return new Square();  
        }  
        return null;  
    }  
}
```

# Mẫu Factory

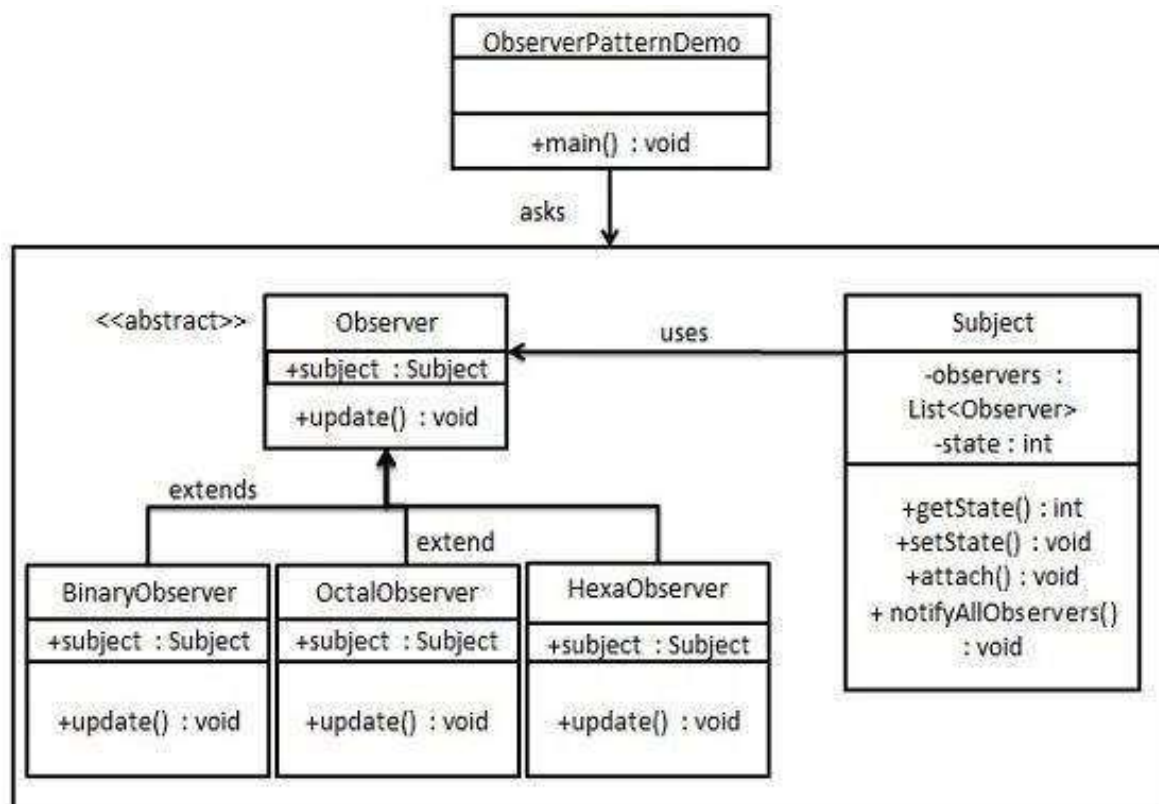
```
public class FactoryPatternDemo {  
    public static void main(String[] args) {  
        ShapeFactory shapeFactory = new ShapeFactory();  
        //get an object of Circle and call its draw method.  
        Shape shape1 = shapeFactory.getShape("CIRCLE");  
  
        //call draw method of Circle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

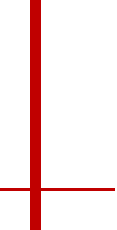


# Mẫu Observer

- Tên: Observer Pattern
- Vấn đề: Nhiều đối tượng phụ thuộc vào một đối tượng khác và khi đối tượng khác này thay đổi, các đối tượng phụ thuộc nó cần được thông báo (cập nhật)
- Giải pháp:

# Mẫu Observer

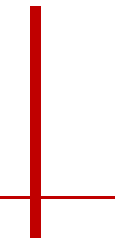




```
import java.util.ArrayList;
import java.util.List;
public class Subject {
    private List<Observer> observers = new ArrayList<Observer>();
    private int state;

    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();
        }
    }
}
```



```
public abstract class Observer {  
    protected Subject subject;  
    public abstract void update();  
}
```

```
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() ) );  
    }  
}
```



```
public class OctalObserver extends Observer{
```

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

```
    @Override
```

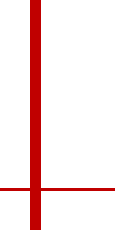
```
    public void update() {  
        System.out.println( "Octal String: " + Integer.toOctalString( subject.getState() ) );  
    }  
}
```

```
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() );  
    }  
}
```

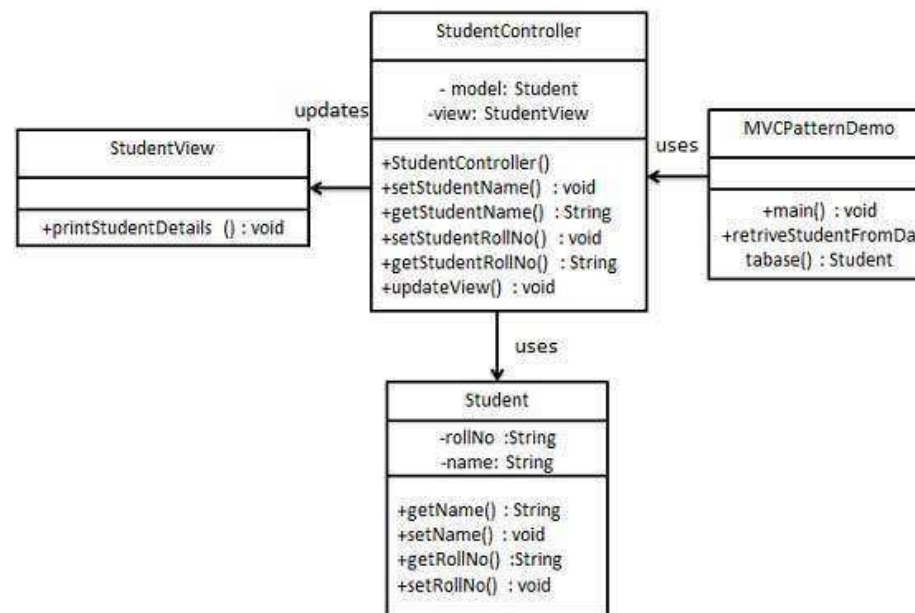


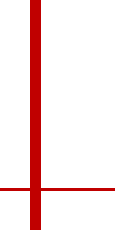
```
public class ObserverPatternDemo {  
    public static void main(String[] args) {  
        Subject subject = new Subject();  
  
        new HexaObserver(subject);  
        new OctalObserver(subject);  
        new BinaryObserver(subject);  
  
        System.out.println("First state change: 15");  
        subject.setState(15);  
        System.out.println("Second state change: 10");  
        subject.setState(10);  
    }  
}
```

```
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
```

# Mẫu MVC

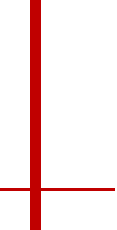
- Tên MVC pattern
- Vấn đề: Phân tách cấu trúc ứng dụng.
- Giải pháp



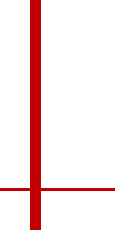


```
public class Student {  
    private String rollNo;  
    private String name;  
  
    public String getRollNo() {  
        return rollNo;  
    }  
  
    public void setRollNo(String rollNo) {  
        this.rollNo = rollNo;  
    }  
  
    public String getName() {  
        return name;  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

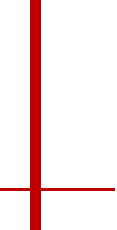




```
public class StudentView {  
    public void printStudentDetails(String studentName, String studentRollNo){  
        System.out.println("Student: ");  
        System.out.println("Name: " + studentName);  
        System.out.println("Roll No: " + studentRollNo);  
    }  
}
```



```
public class StudentController {
    private Student model;
    private StudentView view;
    public StudentController(Student model, StudentView view){
        this.model = model;
        this.view = view;
    }
    public void setStudentName(String name){
        model.setName(name);
    }
    public String getStudentName(){
        return model.getName();
    }
    public void setStudentRollNo(String rollNo){
        model.setRollNo(rollNo);
    }
    public String getStudentRollNo(){
        return model.getRollNo();
    }
    public void updateView(){
        view.printStudentDetails(model.getName(), model.getRollNo());
    }
}
```



```
public class MVCPatternDemo {
    public static void main(String[] args) {
        //fetch student record based on his roll no from the database
        Student model = retrieveStudentFromDatabase();

        //Create a view : to write student details on console
        StudentView view = new StudentView();
        StudentController controller = new StudentController(model, view);

        controller.updateView();

        //update model data
        controller.setStudentName("John");
        controller.updateView();
    }

    private static Student retrieveStudentFromDatabase(){
        Student student = new Student();
        student.setName("Robert");
        student.setRollNo("10");
        return student;
    }
}
```

```
Student:
Name: Robert
Roll No: 10
Student:
Name: John
Roll No: 10
```

# MVC in Laravel

