Design Pattern

By Bishwa Karn

- A **design patterns** are **well-proved** solution for solving the specific problem/task.
- Every design pattern has some specification or set of rules for solving the problems.
- Design patterns are **programming language independent strategies** for solving the common object-oriented design problems. i.e. a design pattern represents an idea, not a particular implementation.
- By using the design patterns, we can make our code more flexible, reusable and maintainable.
- It is the most important part because java internally follows design patterns.
- To become a professional software developer, you must know at least some popular solutions (i.e. design patterns) to the coding problems.
- We must use the design patterns during the analysis and requirement phase of SDLC(Software Development Life Cycle).
- Design patterns ease the analysis and requirement phase of SDLC by providing information based on prior hands-on experiences.

Design patterns are categorized into two parts:

- Core Java (or JSE) Design Patterns.
- JEE Design Patterns.

Core Java Design Patterns

In core java, there are mainly three types of design patterns, which are further divided into their sub-parts:

1. **Creational Design Pattern**
   - Factory Pattern
   - Abstract Factory Pattern
   - Singleton Pattern
   - Prototype Pattern
   - Builder Pattern.
2. **Structural Design Pattern**
   - Adapter Pattern
   - Bridge Pattern
   - Composite Pattern
   - Decorator Pattern
   - Facade Pattern
   - Flyweight Pattern
   - Proxy Pattern
3. **Behavioral Design Pattern**
   - Chain Of Responsibility Pattern
   - Command Pattern
   - Interpreter Pattern
   - Iterator Pattern
   - Mediator Pattern
   - Memento Pattern
   - Observer Pattern
   - State Pattern
   - Strategy Pattern
   - Template Pattern
   - Visitor Pattern

JEE or J2EE Design Patterns

J2EE design patterns are built for the developing the Enterprise Web-based Applications.
In J2EE , there are mainly three types of design patterns, which are further divided into their sub-parts:

1. **Presentation Layer Design Pattern**
   - Intercepting Filter Pattern
   - Front Controller Pattern
   - View Helper Pattern
   - Composite View Pattern
2. **Business Layer Design Pattern**
   - Business Delegate Pattern
   - Service Locator Pattern
   - Session Facade Pattern
   - Transfer Object Pattern
3. **Integration Layer Design Pattern**
   - Data Access Object Pattern
   - Web Service Broker Pattern

Advantage of design pattern

- They are reusable in multiple projects.
- They provide the solutions that help to define the system architecture.
- They capture the software engineering experiences.
- They provide transparency to the design of an application.
- They are well-proved and testified solutions since they have been built upon the knowledge and experience of expert software developers.
- Design patterns don’t guarantee an absolute solution to a problem. They provide clarity to the system architecture and the possibility of building a better system.

MVC Pattern

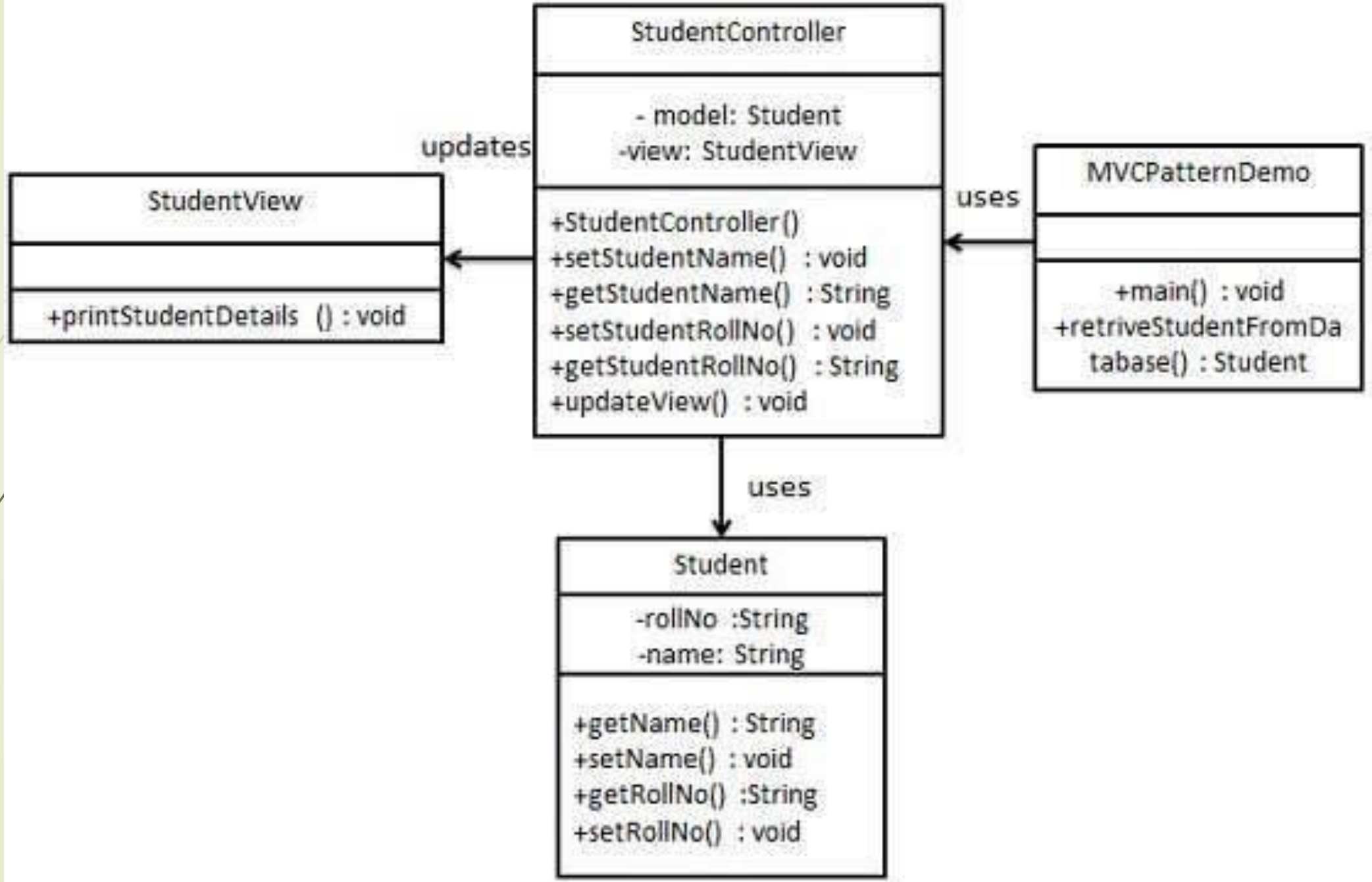
- MVC Pattern stands for **Model-View-Controller** Pattern. This pattern is used to separate application's concerns.

1. **Model** - Model represents an object or JAVA POJO (Plain Old Java Object) carrying data. It can also have logic to update controller if its data changes.
2. **View** - View represents the visualization of the data that model contains.
3. **Controller** - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

Example

Let’s create:

- **Student** object acting as a model.
- **StudentView** will be a view class which can print student details on console.
- **StudentController** is the controller class responsible to store data in **Student** object and update view **StudentView** accordingly.
- **MVCPatternDemo** is our demo class, and it will use **StudentController** to demonstrate use of MVC pattern.


StudentController
- model: Student
-view: StudentView
+StudentController()
+setStudentName() : void
+getStudentName() : String
+setStudentRollNo() : void
+getStudentRollNo() : String
+updateView() : void
updates
StudentView
+printStudentDetails () : void
uses
MVCPatternDemo
+main() : void
+retriveStudentFromDatabase() : Student
uses
Student
-rollNo :String
-name: String
+getName() : String
+setName() : void
+getRollNo() :String
+setRollNo() : void

Creating Button

```
import javax.swing.*;
public class ButtonExample {
    public static void main(String[] args) {
        JFrame f = new JFrame("Button Example");
        JButton btn = new JButton("Click Here");
        btn.setBounds(50,100,95,30);
        f.add(btn);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```