Welcome to the Class

Department of Computing and Information System

Design pattern

Md. Selim Hossain
Senior Lecturer
Department of Computing and Information System
Daffodil International University (DIU),Dhaka, Bangladesh

Design Pattern

Design patterns represent the best practices used by experienced object-oriented software developers. Design patterns are solutions to general problems that software developers faced during software development. These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time.

What is Gang of Four (GOF)?

In 1994, four authors Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides published a book titled **Design Patterns - Elements of Reusable Object-Oriented Software** which initiated the concept of Design Pattern in Software development.

These authors are collectively known as **Gang of Four (GOF)**. According to these authors design patterns are primarily based on the following principles of object orientated design.

- Program to an interface not an implementation
- Favor object composition over inheritance

Usage of Design Pattern

Design Patterns have two main usages in software development.

>Common platform for developers

Design patterns provide a standard terminology and are specific to particular scenario. For example, a singleton design pattern signifies use of single object so all developers familiar with single design pattern will make use of single object and they can tell each other that program is following a singleton pattern.

>Best Practices

Design patterns have been evolved over a long period of time and they provide best solutions to certain problems faced during software development. Learning these patterns helps unexperienced developers to learn software design in an easy and faster way.

Types of Design Patterns

As per the design pattern reference book **Design Patterns - Elements of Reusable Object-Oriented Software** , there are 23 design patterns which can be classified in three categories: Creational, Structural and Behavioral patterns.

- Design Patterns - Singleton Pattern
- Design Patterns - Builder Pattern
- Design Patterns - Prototype Pattern
- Design Patterns - Adapter Pattern
- Design Patterns - Bridge Pattern
- Design Patterns - Filter Pattern
- Design Patterns - Composite Pattern
- Design Patterns - Decorator Pattern
- Design Patterns - Facade Pattern
- Design Patterns - Flyweight Pattern
- Design Patterns - Proxy Pattern
- Chain of Responsibility Pattern
- Design Patterns - Command Pattern
- Design Patterns - Interpreter Pattern
- Design Patterns - Iterator Pattern
- Design Patterns - Mediator Pattern
- Design Patterns - Memento Pattern
- Design Patterns - Observer Pattern
- Design Patterns - State Pattern

Creational Patterns

These design patterns provide a way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator. This gives program more flexibility in deciding which objects need to be created for a given use case.

Structural Patterns

These design patterns concern class and object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities.

J2EE Patterns

These design patterns are specifically concerned with the presentation tier. These patterns are identified by Sun Java Center.

J2EE-Java 2 Platform Enterprise Edition

Factory pattern is one of the most used design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

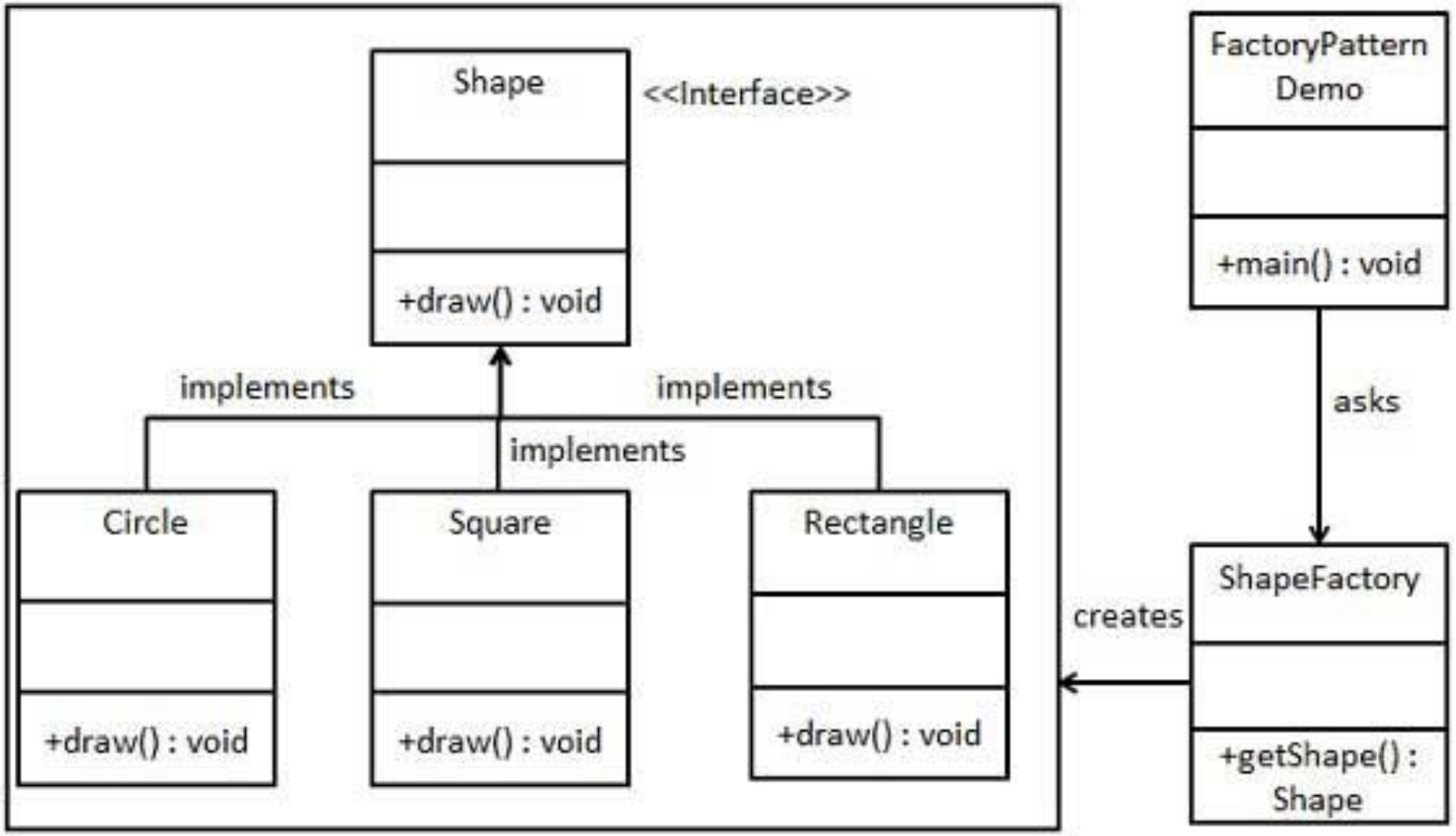
In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

Implementation

We're going to create a *Shape* interface and concrete classes implementing the *Shape* interface. A factory class *ShapeFactory* is defined as a next step.

FactoryPatternDemo, our demo class will use *ShapeFactory* to get a *Shape* object. It will pass information (*CIRCLE / RECTANGLE / SQUARE*) to *ShapeFactory* to get the type of object it needs.

Graphical Example of Design pattern



Design Pattern

- Well known patterns
  - **Model View Controller (MVC)**
  - **Facade pattern**

Model-View-Controller

- **Model** – data model
- **View** – presentation of the model
- **Controller** – controls the flow / interactions of the view and model

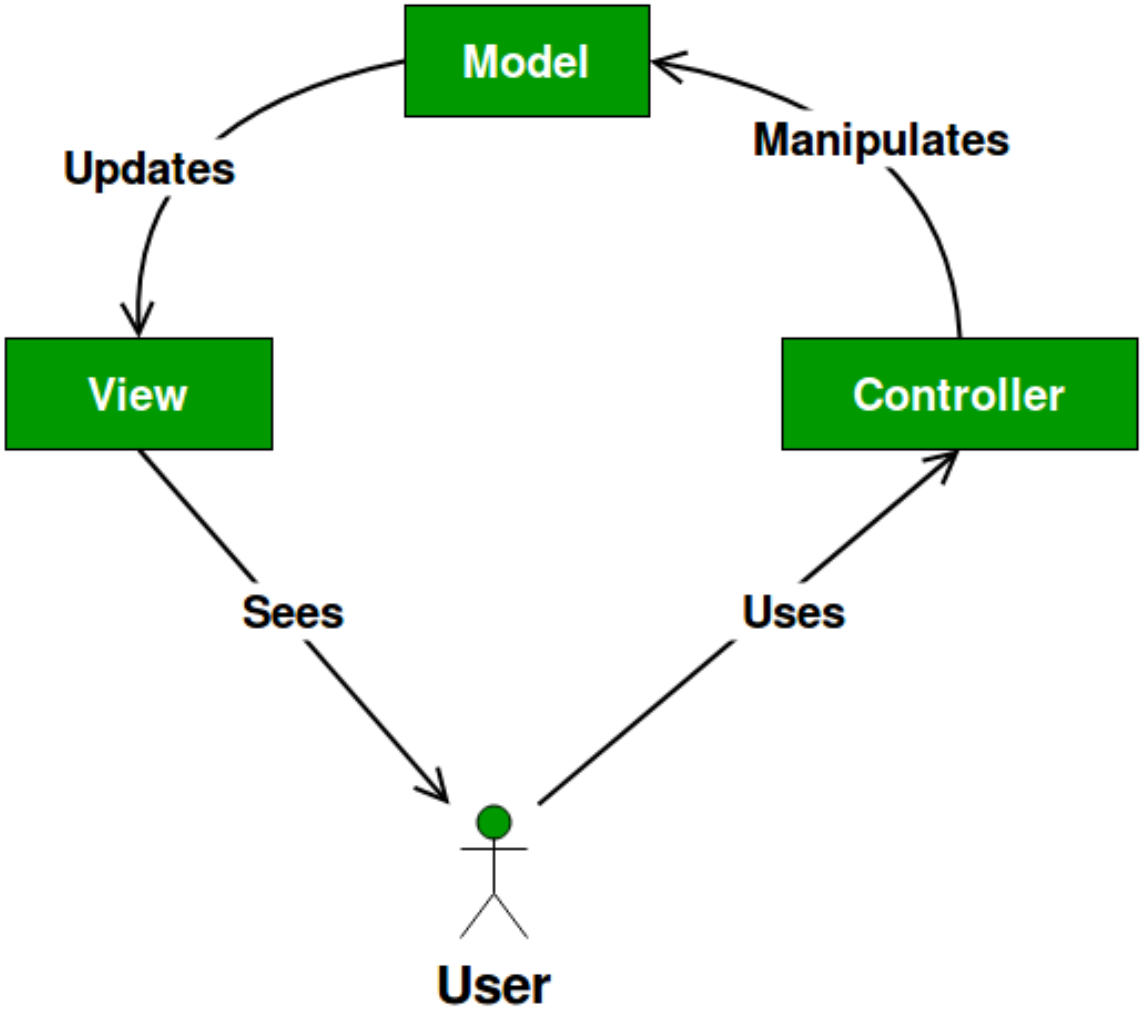
Model-View-Controller

- The model-view-controller (MVC) design pattern specifies that an application consist of a data model, presentation information, and control information.
- The pattern requires that each of these be separated into different objects.

Model-View-Controller

- The ***model*** (for example, the data information) contains only the pure application data; it contains no logic describing how to present the data to a user.
- The ***view*** (for example, the presentation information) presents the model's data to the user. The view knows how to access the model's data, but it does not know what this data means or what the user can do to manipulate it.
- Finally, the ***controller*** (for example, the control information) exists between the view and the model. It listens to events triggered by the view and executes the appropriate reaction to these events. In most cases, the reaction is to call a method on the model.

Model-View-Controller



Thanks to All