Design Patterns

A design patterns are well-proven solution for solving specific problem and task

Ex:

Problem Given:
Suppose you want to create a class for which only a single instance (or object) should be created and that single object can be used by all other classes.

Solution:
Singleton design pattern is the best solution to the above specific problem. So, every design pattern has **some specification or set of rules** for solving the problems. What are those specifications, you will see later in the types of design patterns.

Note :

- Design patterns are programming language independent strategies for solving common object oriented design problems
- Must use design patterns during the analysis and requirement phases

Advantages:

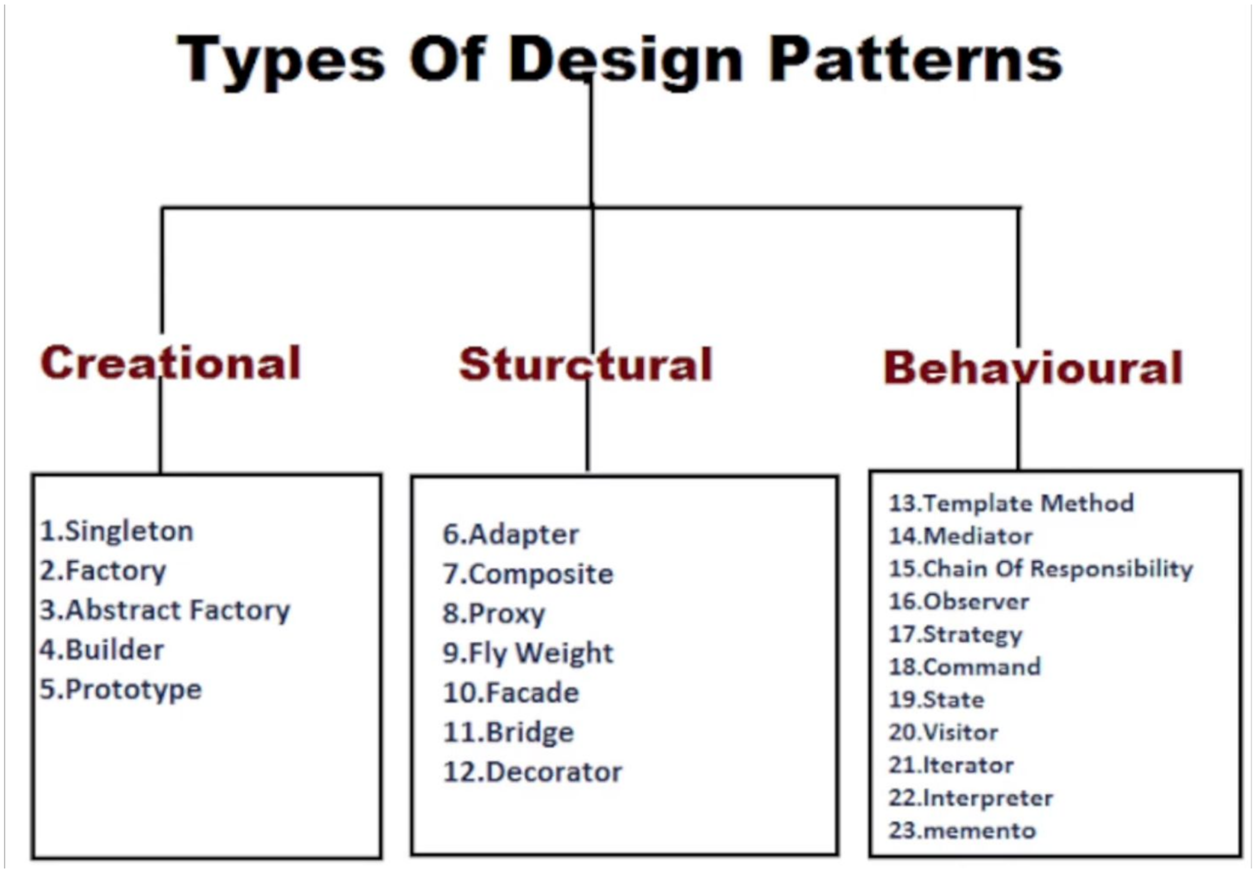
- Code more flexible
- Reusable and maintainable
- Provide solutions that help to define system architecture
- Captured the software engineering experiences
- Provide transparency to the design of an application
- Don't guarantee an absolute solution to a problem ,they provide clarity to the system architecture and the possibility of building a better system

Categorization of Design patterns

- Core Java(or JSE)
- J2EE

Core Java Design Patterns

In core Java there are mainly three types of design patterns,which are further divided in to their sub-parts

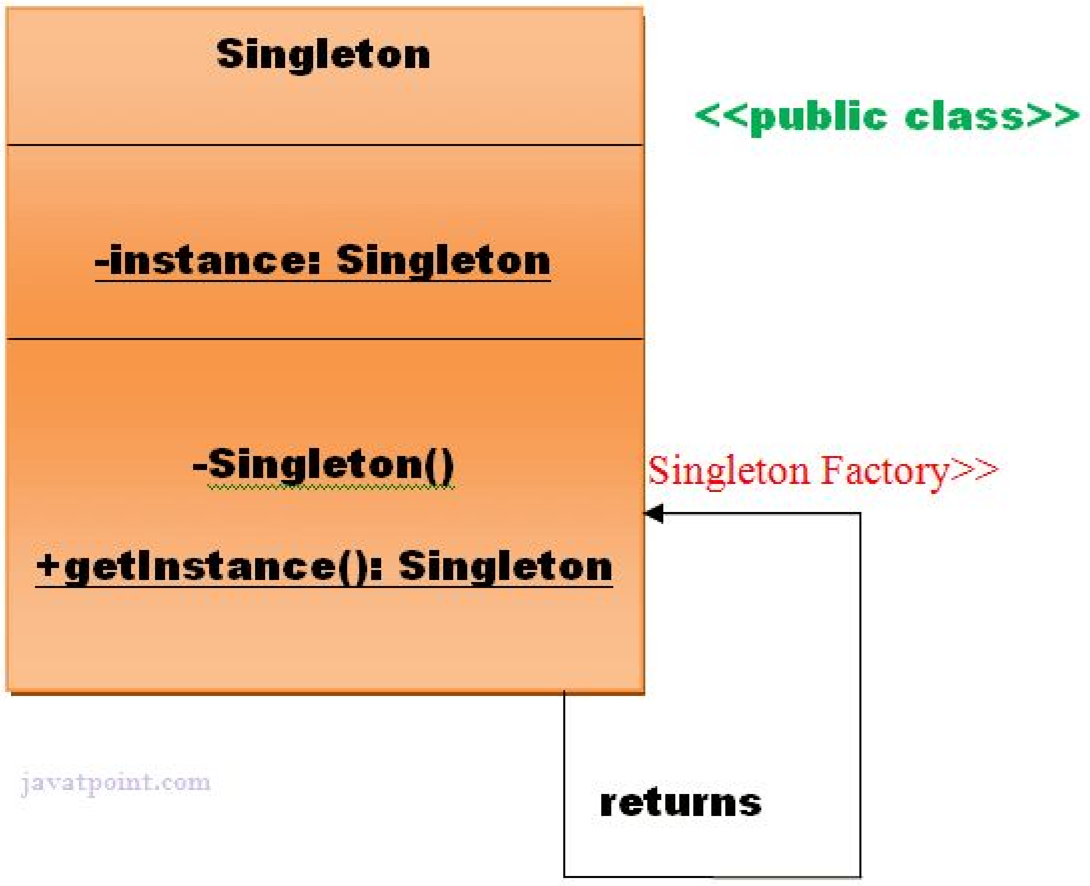


1.Creational Design Pattern

Concerned with way of creating objects .These design patterns are used when decision must be made at the time of instantiation of class

1.1 Singleton Design Pattern

Define a class that has only one instance and provides a global point of access to it
Class must ensure that only single instance should be created and single object can be used by all other classes



There are two forms of singleton design pattern

Early instantiation

Lazy instantiation

Advantage

Save memory because object is not created at each request.only single instance is reuse again and again

Usage

- Multithreaded and database applications

- logging,caching ,thread pools ,configuration settings etc.

To create the singleton class, we need to have a static member of class, private constructor and static factory method.

- **Static member:** It gets memory only once because of static, it contains the instance of the Singleton class.
- **Private constructor:** It will prevent instantiating the Singleton class from outside the class.
- **Static factory method:** This provides the global point of access to the Singleton object and returns the instance to the caller.

1.2.Factory Method Pattern

A Factory Pattern or Factory Method Pattern says that just **define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate.** In other words, subclasses are responsible for creating the instance of the Class.

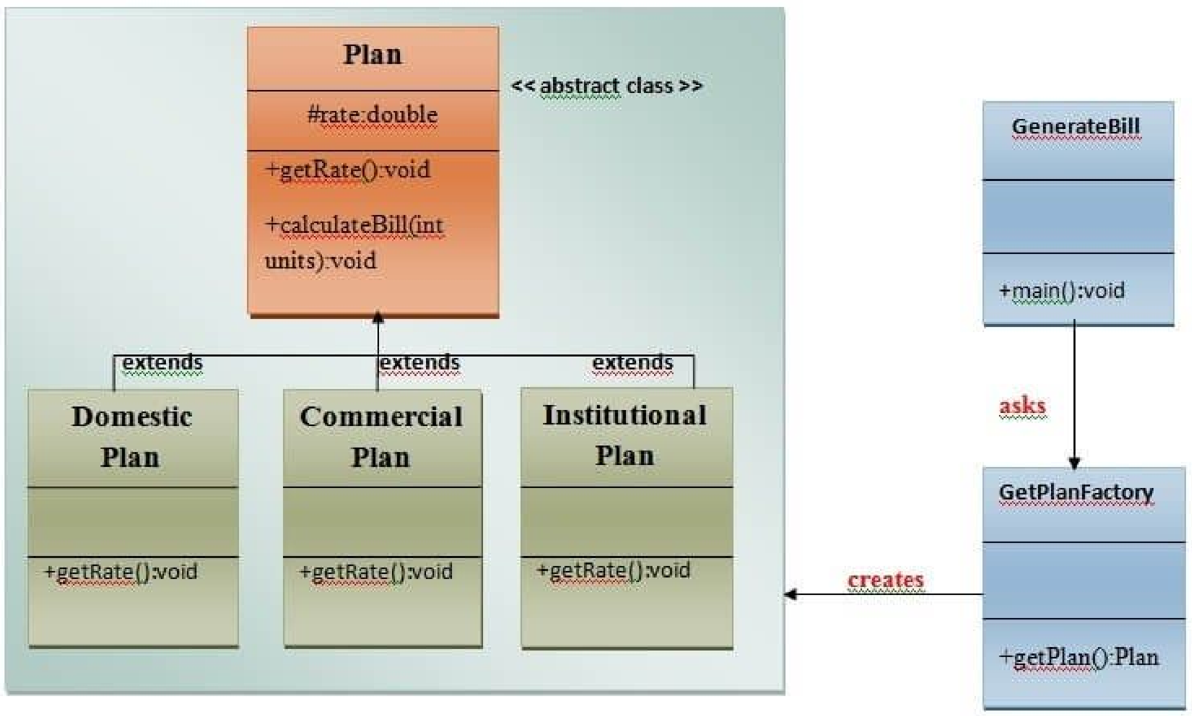
The Factory Method Pattern is also known as **Virtual Constructor.**

Advantage of Factory Design Pattern

- Factory Method Pattern allows the sub-classes to choose the type of objects to create.
- It promotes the loose-coupling by eliminating the need to bind application-specific classes into the code. That means the code interacts solely with the resultant interface or abstract class, so that it will work with any classes that implement that interface or that extends that abstract class.

Usage of Factory Design Pattern

- When a class doesn't know what sub-classes will be required to create
- When a class wants that its sub-classes specify the objects to be created.
- When the parent classes choose the creation of objects to its sub-classes.



1.3.Abstract Factory Pattern

Abstract Factory Pattern says that just **define an interface or abstract class for creating families of related (or dependent) objects but without specifying their concrete subclasses.That** means Abstract Factory lets a class return a factory of classes. So, this is the reason that the Abstract Factory Pattern is one level higher than the Factory Pattern.

An Abstract Factory Pattern is also known as **Kit.**

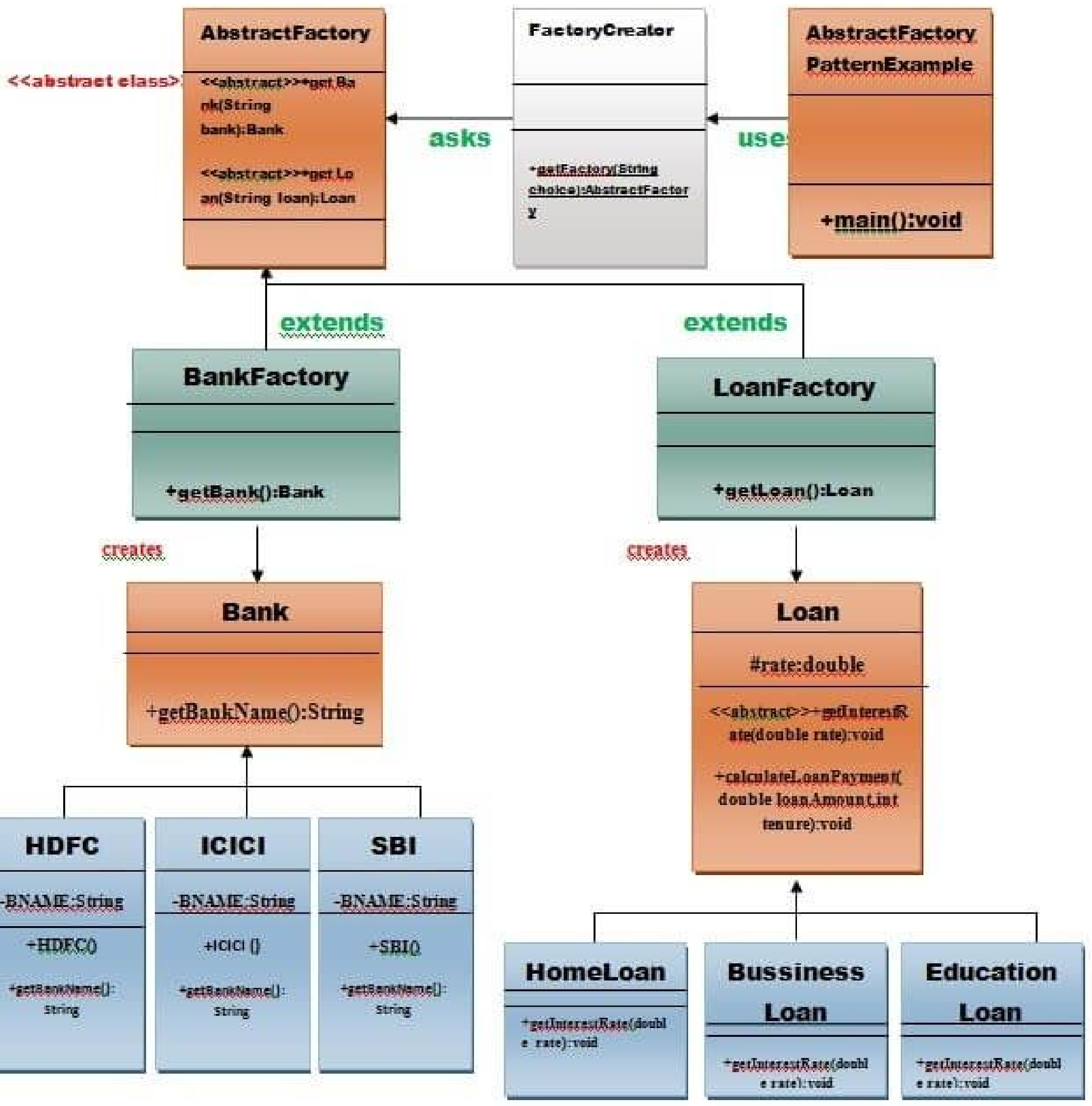
Advantage of Abstract Factory Pattern

- Abstract Factory Pattern isolates the client code from concrete (implementation) classes.
- It eases the exchanging of object families.
- It promotes consistency among objects.

Usage of Abstract Factory Pattern

- When the system needs to be independent of how its objects are created, composed, and represented.
- When the family of related objects has to be used together, then this constraint needs to be enforced.

- When you want to provide a library of objects that does not show implementations and only reveals interfaces.
- When the system needs to be configured with one of a multiple family of objects.



2.Structural Design Pattern

- Concerned with how classes and objects can be composed to form larger structures
- The structural design patterns simplifies the structure by identifying the relationships
- These patterns focus on how the classes inherit from each other and how they are composed from the other classes

2.1.Decorator Pattern

Attach a flexible additional responsibilities to an object dynamically
Decorator pattern uses **composition** instead of inheritance to extend functionality of an object at run time
Also known as wrapper

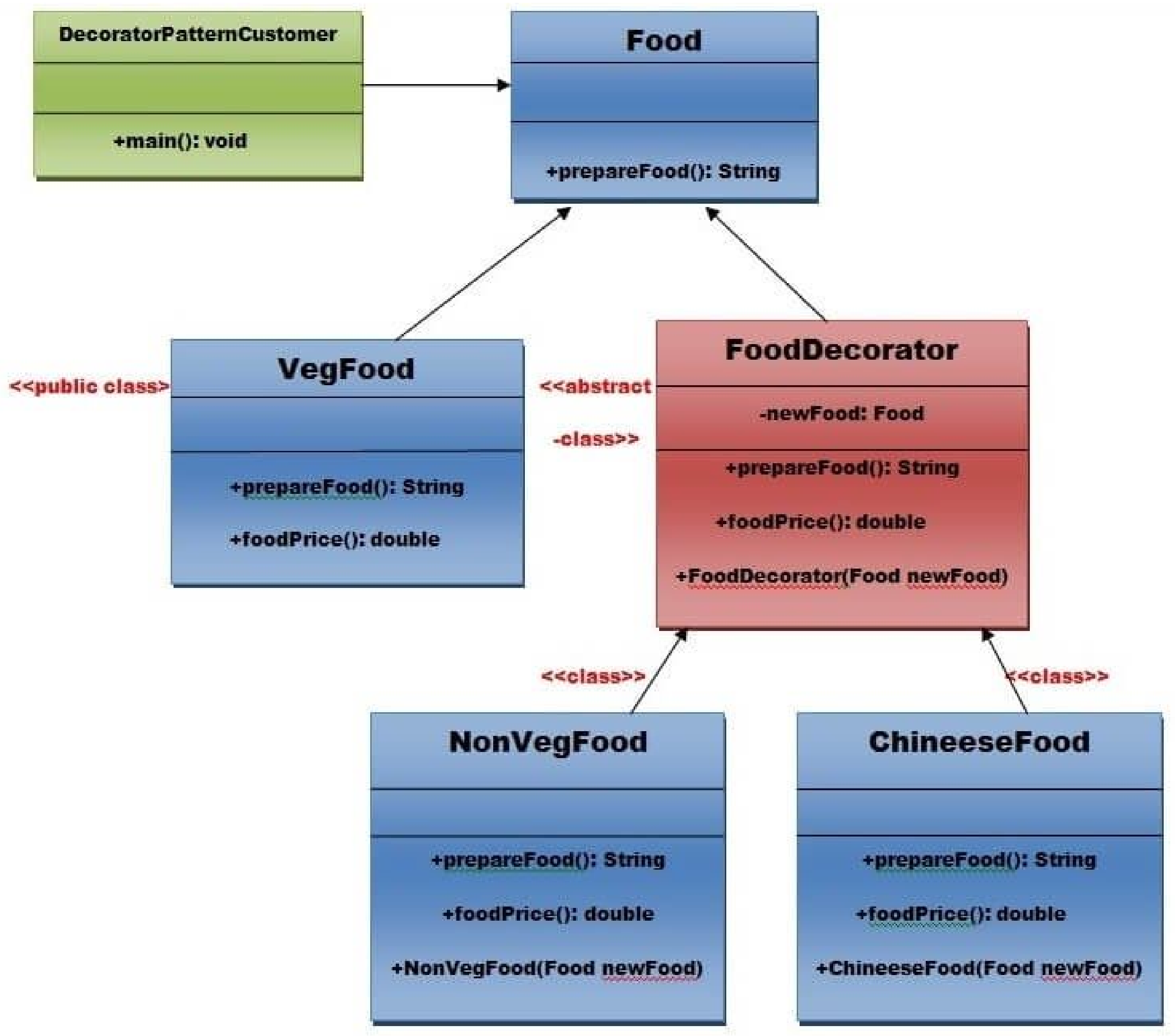
Advantage of Decorator Pattern

- It provides greater flexibility than static inheritance.
- It enhances the extensibility of the object, because changes are made by coding new classes.
- It simplifies the coding by allowing you to develop a series of functionality from targeted classes instead of coding all of the behavior into the object.

Usage of Decorator Pattern

It is used:

- When you want to transparently and dynamically add responsibilities to objects without affecting other objects.
- When you want to add responsibilities to an object that you may want to change in future.
- Extending functionality by subclassing is no longer practical.



2.2.Facade Pattern

- Provide unified and simplified interface to a set of interfaces in a subsystem ,therefore it hides complexities of the subsystem from the client
- Higher level interface that make sub system easier to use
- Every abstract factory is a type of facade

Advantage of Facade Pattern

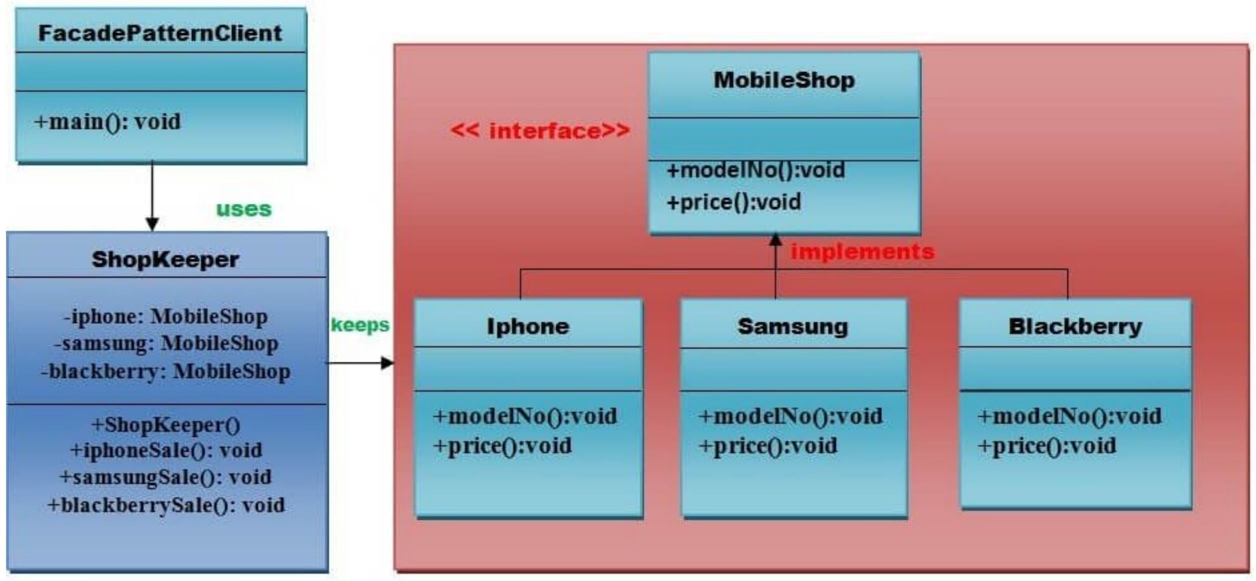
- It shields the clients from the complexities of the sub-system components.

- It promotes loose coupling between subsystems and its clients.

Usage of Facade Pattern:

It is used:

- When you want to provide a simple interface to a complex sub-system.
- When several dependencies exist between clients and the implementation classes of an abstraction.



2.3 .Adapter Pattern

- Convert the interface of a class into another interface that client wants
- In other words, to provide the interface according to client requirement while using the services of a class with a different interface.
- The Adapter Pattern is also known as **Wrapper.**

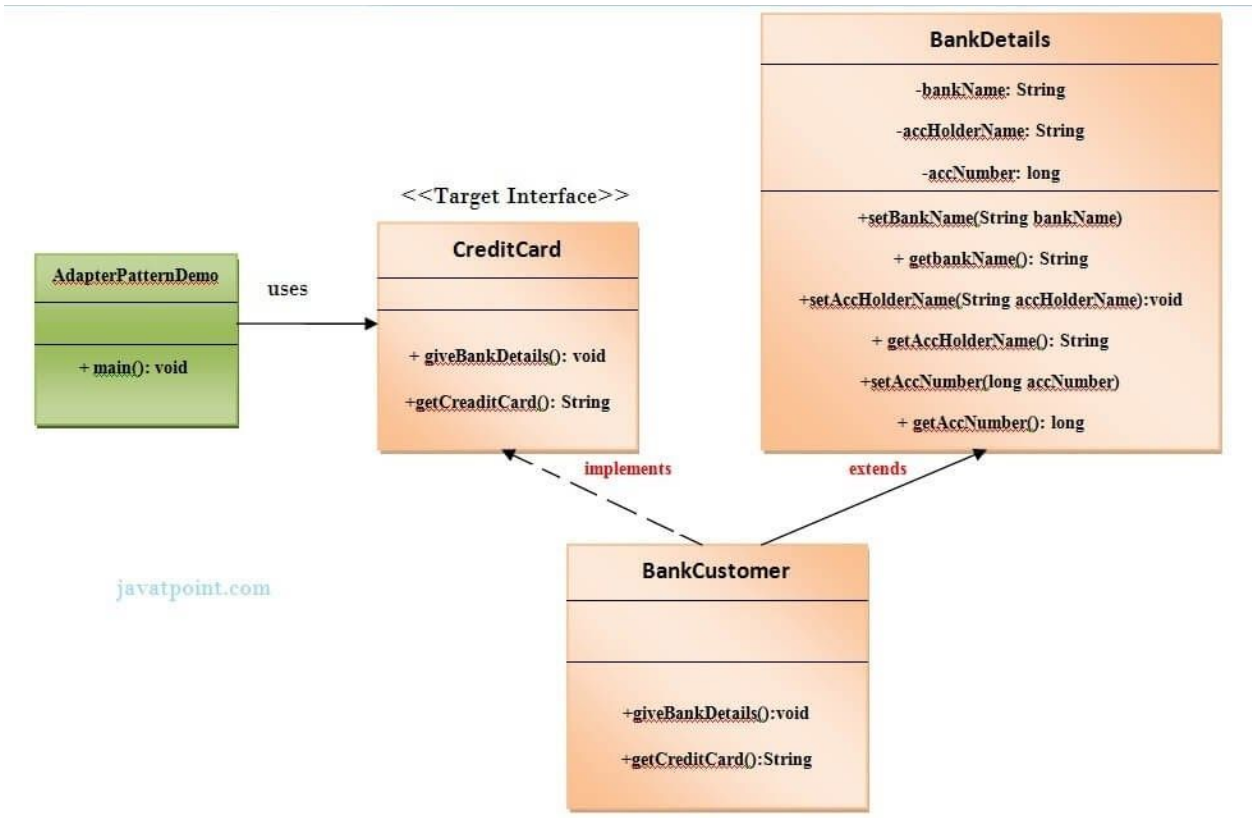
Advantage of Adapter Pattern

- It allows two or more previously incompatible objects to interact.
- It allows reusability of existing functionality.

Usage of Adapter pattern:

It is used:

- When an object needs to utilize an existing class with an incompatible interface.
- When you want to create a reusable class that cooperates with classes which don't have compatible interfaces.
- When you want to create a reusable class that cooperates with classes which don't have compatible interfaces.



3.Behavioural Pattern

- Concerned with interaction and responsibility of objects
- In these design patterns , the interaction between the objects should be in such a way that they can easily talk to each other and still should loosely coupled
- That means the implementation and the client should be loosely coupled in the order to avoid hard coding and dependencies

3.1.Strategy Pattern

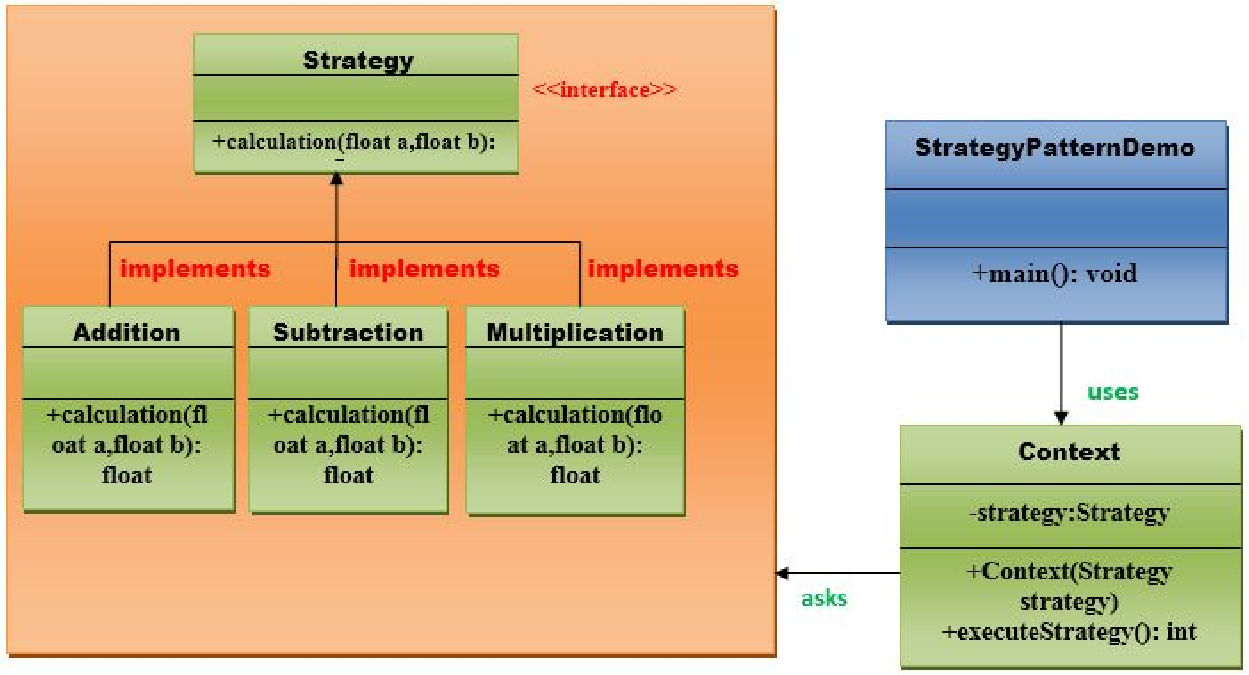
Define a family functionality encapsulate each one and make them interchangeable
Also known as Policy

Benefits:

- It provides a substitute to subclassing.
- It defines each behavior within its own class, eliminating the need for conditional statements.
- It makes it easier to extend and incorporate new behavior without changing the application.

Usage:

- When the multiple classes differ only in their behaviors.e.g. Servlet API.
- It is used when you need different variations of an algorithm.



3.2.Observer Pattern

The Observer Pattern defines a one to many dependency between objects so that one object changes state, all of its dependents are notified and updated automatically.

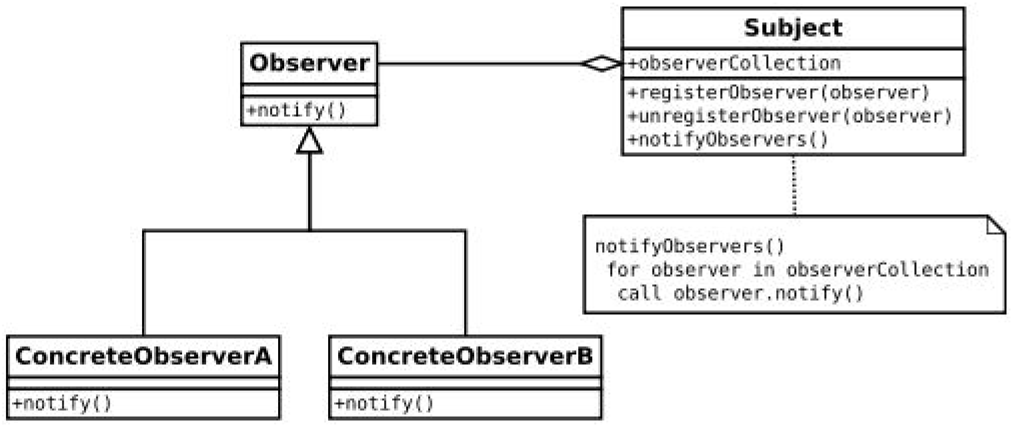
Ex:

To understand observer pattern, first you need to understand the subject and observer objects.

The relation between subject and observer can easily be understood as an analogy to magazine subscription.

- A magazine publisher(subject) is in the business and publishes magazines (data).
- If you(user of data/observer) are interested in the magazine you subscribe(register), and if a new edition is published it gets delivered to you.
- If you unsubscribe(unregister) you stop getting new editions.
- Publisher doesn’t know who you are and how you use the magazine, it just delivers it to you because you are a subscriber(loose coupling).

- One to many dependency is between Subject(One) and Observer(Many).
- There is dependency as Observers themselves don’t have access to data. They are dependent on Subject to provide them data.



Here Observer and Subject are interfaces(can be any abstract super type not necessarily java interface).

- All observers who need the data need to implement an observer interface.
- notify() method in the observer interface defines the action to be taken when the subject provides it data.
- The subject maintains an observerCollection which is simply the list of currently registered(subscribed) observers.
- registerObserver(observer) and unregisterObserver(observer) are methods to add and remove observers respectively.
- notifyObservers() is called when the data is changed and the observers need to be supplied with new data.

Advantages:

Provides a loosely coupled design between objects that interact. Loosely coupled objects are flexible with changing requirements. Here loose coupling means that the interacting objects should have less information about each other.

Observer pattern provides this loose coupling as:

- Subject only knows that observer implements the Observer interface.Nothing more.
- There is no need to modify Subject to add or remove observers.

- We can reuse subject and observer classes independently of each other.

Disadvantages:

- Memory leaks caused by Lapsed listener problem because of explicit register and unregistering of observers.

When to use this pattern?

You should consider using this pattern in your application when multiple objects are dependent on the state of one object as it provides a neat and well tested design for the same.

Real Life Uses:

- It is heavily used in GUI toolkits and event listeners. In java the button(subject) and onClickListener(observer) are modelled with observer pattern.
- Social media, RSS feeds, email subscription in which you have the option to follow or subscribe and you receive the latest notification.
- All users of an app on play store get notified if there is an update.