Module-5

Design Patterns

A design patterns are **well-proved solution** for solving the specific problem/task.

Now, a question will be arising in your mind what kind of specific problem? Let me explain by taking an example.

Problem:

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 of 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.

Design patterns are programming language independent strategies for solving the common object-oriented design problems. That means, a design pattern represents an idea, not a particular implementation.

By using the design patterns you can make your 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.

Advantage of design pattern:

1. They are reusable in multiple projects.
2. They provide the solutions that help to define the system architecture.
3. They capture the software engineering experiences.
4. They provide transparency to the design of an application.
5. They are well-proved and testified solutions since they have been built upon the knowledge and experience of expert software developers.

6. 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.

When should we use the design patterns?

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.

There are mainly three types of design patterns, which are further divided into their sub-parts:

- Creational Design Pattern
- Structural Design Pattern
- Behavioral Design Pattern

These Design Patterns are called as **GoF Design Patterns** or **Gangs of Four Design Patterns.** ***Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides*** **have been named as "The Gang of Four".**

Gang of Four Design Patterns

Creational Design Patterns

- Abstract Factory. Allows the creation of objects without specifying their concrete type.
- Builder. Uses to create complex objects.
- Factory Method. Creates objects without specifying the exact class to create.
- Prototype. Creates a new object from an existing object.
- Singleton. Ensures only one instance of an object is created.

Structural Design Patterns

- Adapter. Allows for two incompatible classes to work together by wrapping an interface around one of the existing classes.
- Bridge. Decouples an abstraction so two classes can vary independently.
- Composite. Takes a group of objects into a single object.
- Decorator. Allows for an object’s behavior to be extended dynamically at run time.
- Facade. Provides a simple interface to a more complex underlying object.
- Flyweight. Reduces the cost of complex object models.
- Proxy. Provides a placeholder interface to an underlying object to control access, reduce cost, or reduce complexity.

Behavior Design Patterns

- Chain of Responsibility. Delegates commands to a chain of processing objects.
- Command. Creates objects which encapsulate actions and parameters.
- Interpreter. Implements a specialized language.
- Iterator. Accesses the elements of an object sequentially without exposing its underlying representation.
- Mediator. Allows loose coupling between classes by being the only class that has detailed knowledge of their methods.
- Memento. Provides the ability to restore an object to its previous state.
- Observer. Is a publish/subscribe pattern which allows a number of observer objects to see an event.
- State. Allows an object to alter its behavior when its internal state changes.
- Strategy. Allows one of a family of algorithms to be selected on-the-fly at run-time.
- Template Method. Defines the skeleton of an algorithm as an abstract class, allowing its sub-classes to provide concrete behavior.
- Visitor. Separates an algorithm from an object structure by moving the hierarchy of methods into one object.

Creational design patterns

Creational design patterns are concerned with **the way of creating objects.** These design patterns are used when a decision must be made at the time of instantiation of a class (i.e. creating an object of a class).

There are following 6 types of creational design patterns.

1. Factory Method Pattern

2. Abstract Factory Pattern
3. Singleton Pattern
4. Prototype Pattern
5. Builder Pattern
6. Object Pool Pattern

Factory Method Pattern

subclasses are responsible to create the instance of the class.

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, 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.

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 sub-classes.**That means Abstract Factory lets a class returns a factory of classes. So, this is the reason that 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 object 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.

Singleton pattern

Singleton Pattern says that just"define a class that has only one instance and provides a global point of access to it".

In other words, a 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:** creation of instance at load time.
- **Lazy Instantiation:** creation of instance when required.

Advantage of Singleton design pattern

- Saves memory because object is not created at each request. Only single instance is reused again and again.

How to create Singleton design pattern?

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

- **Static member:** It gets memory only once because of static, itcontains the instance of the Singleton class.
- **Private constructor:** It will prevent to instantiate 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.

Usage of Singleton design pattern

- Singleton pattern is mostly used in multi-threaded and database applications. It is used in logging, caching, thread pools, configuration settings etc.

Prototype Design Pattern

Prototype Pattern says that **cloning of an existing object instead of creating new one and can also be customized as per the requirement**.

This pattern should be followed, if the cost of creating a new object is expensive and resource intensive.

Advantage of Prototype Pattern

The main advantages of prototype pattern are as follows:

- It reduces the need of sub-classing.
- It hides complexities of creating objects.
- The clients can get new objects without knowing which type of object it will be.
- It lets you add or remove objects at runtime.

Usage of Prototype Pattern

- When the classes are instantiated at runtime.
- When the cost of creating an object is expensive or complicated.
- When you want to keep the number of classes in an application minimum.
- When the client application needs to be unaware of object creation and representation.

Builder Design Pattern

Builder Pattern says that **"construct a complex object from simple objects using step-by-step approach"**

It is mostly used when object can't be created in single step like in the de-serialization of a complex object.

Advantage of Builder Design Pattern

The main advantages of Builder Pattern are as follows:

- It provides clear separation between the construction and representation of an object.
- It provides better control over construction process.
- It supports to change the internal representation of objects.

Object Pool Pattern

Mostly, performance is the key issue during the software development and the object creation, which may be a costly step.

Object Pool Pattern says that **" to reuse the object that are expensive to create".**

Basically, an Object pool is a container which contains a specified amount of objects. When an object is taken from the pool, it is not available in the pool until it is put back. **Objects in the pool have a lifecycle: creation, validation and destroy.**

A pool helps to manage available resources in a better way. There are many using examples: especially in application servers there are data source pools, thread pools etc.

Advantage of Object Pool design pattern

- It boosts the performance of the application significantly.
- It is most effective in a situation where the rate of initializing a class instance is high.
- It manages the connections and provides a way to reuse and share them.
- It can also provide the limit for the maximum number of objects that can be created.

Usage:

- When an application requires objects which are expensive to create. Eg: there is a need of opening too many connections for the database then it takes too longer to create a new one and the database server will be overloaded.
- When there are several clients who need the same resource at different times.

NOTE: Object pool design pattern is essentially used in Web Container of the server for creating thread pools and data source pools to process the requests.

Structural design patterns

Structural design patterns are 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 other classes.

Types of structural design patterns

There are following 7 types of structural design patterns.

1. Adapter Pattern

   Adapting an interface into another according to client expectation.

2. Bridge Pattern

   Separating abstraction (interface) from implementation.

3. Composite Pattern

   Allowing clients to operate on hierarchy of objects.

4. Decorator Pattern

   Adding functionality to an object dynamically.

5. Facade Pattern

   Providing an interface to a set of interfaces.

6. Flyweight Pattern

   Reusing an object by sharing it.

7. proxy Pattern

   Representing another object.

Adapter Pattern

An Adapter Pattern says that just **"converts the interface of a class into another interface that a 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.

Bridge Pattern

A Bridge Pattern says that just **"decouple the functional abstraction from the implementation so that the two can vary independently".**

The Bridge Pattern is also known as **Handle or Body.**

Advantage of Bridge Pattern

- It enables the separation of implementation from the interface.
- It improves the extensibility.
- It allows the hiding of implementation details from the client.

Usage of Bridge Pattern

- When you don't want a permanent binding between the functional abstraction and its implementation.
- When both the functional abstraction and its implementation need to extended using sub-classes.
- It is mostly used in those places where changes are made in the implementation does not affect the clients.

Composite Pattern

A Composite Pattern says that just **"allow clients to operate in generic manner on objects that may or may not represent a hierarchy of objects".**

Advantage of Composite Design Pattern

- It defines class hierarchies that contain primitive and complex objects.
- It makes easier to you to add new kinds of components.
- It provides flexibility of structure with manageable class or interface.

Usage of Composite Pattern

It is used:

- When you want to represent a full or partial hierarchy of objects.
- When the responsibilities are needed to be added dynamically to the individual objects without affecting other objects. Where the responsibility of object may vary from time to time.

Decorator Pattern

A Decorator Pattern says that just **"attach a flexible additional responsibilities to an object dynamically".**

In other words, The Decorator Pattern uses composition instead of inheritance to extend the functionality of an object at runtime.

The Decorator Pattern is 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 sub-classing is no longer practical.

Facade Pattern

A Facade Pattern says that just **"just provide a unified and simplified interface to a set of interfaces in a subsystem, therefore it hides the complexities of the subsystem from the client".**

In other words, Facade Pattern describes a higher-level interface that makes the sub-system easier to use.

Practically, **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 simple interface to a complex sub-system.
- When several dependencies exist between clients and the implementation classes of an abstraction.

Flyweight Pattern

A Flyweight Pattern says that just **"to reuse already existing similar kind of objects by storing them and create new object when no matching object is found"**.

Advantage of Flyweight Pattern

- It reduces the number of objects.
- It reduces the amount of memory and storage devices required if the objects are persisted

Usage of Flyweight Pattern

- When an application uses number of objects
- When the storage cost is high because of the quantity of objects.
- When the application does not depend on object identity.

Proxy Pattern

Simply, proxy means an object representing another object.

According to GoF, a Proxy Pattern **"provides the control for accessing the original object".**

So, we can perform many operations like hiding the information of original object, on demand loading etc.

Proxy pattern is also known as **Surrogate or Placeholder.**

Advantage of Proxy Pattern

- It provides the protection to the original object from the outside world.

Usage of Proxy Pattern:

It is used:

- It can be used in **Virtual Proxy** scenario---Consider a situation where there is multiple database call to extract huge size image. Since this is an expensive operation so here we can use the proxy pattern which would create multiple proxies and point to the huge size memory consuming object for further processing. The real object gets created only when a client first requests/accesses the object and after that we can just refer to the proxy to reuse the object. This avoids duplication of the object and hence saving memory.
- It can be used in **Protective Proxy** scenario---It acts as an authorization layer to verify that whether the actual user has access the appropriate content or not. For example, a proxy server which provides restriction on internet access in office. Only the websites and contents which are valid will be allowed and the remaining ones will be blocked.
- It can be used in **Remote Proxy** scenario---A remote proxy can be thought about the stub in the RPC call. The remote proxy provides a local representation of the object which is present in the different address location. Another example can be providing interface for remote resources such as web service or REST resources.
- It can be used in **Smart Proxy** scenario---A smart proxy provides additional layer of security by interposing specific actions when the object is accessed. For example, to check whether the real object is locked or not before accessing it so that no other objects can change it.

Behavioral Design Patterns

Behavioral design patterns are concerned with **the 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 be loosely coupled.**

That means the implementation and the client should be loosely coupled in order to avoid hard coding and dependencies.

There are 12 types of behavioral design patterns:

1. Chain of Responsibility Pattern

2. Command Pattern
3. Interpreter Pattern
4. Iterator Pattern
5. Mediator Pattern
6. Memento Pattern
7. Observer Pattern
8. State Pattern
9. Strategy Pattern
10. Template Pattern
11. Visitor Pattern
12. Null Object

Chain Of Responsibility Pattern

In chain of responsibility, sender sends a request to a chain of objects. The request can be handled by any object in the chain.

A Chain of Responsibility Pattern says that just **"avoid coupling the sender of a request to its receiver by giving multiple objects a chance to handle the request".** For example, an ATM uses the Chain of Responsibility design pattern in money giving process.

In other words, we can say that normally each receiver contains reference of another receiver. If one object cannot handle the request then it passes the same to the next receiver and so on.

Advantage of Chain of Responsibility Pattern

- It reduces the coupling.
- It adds flexibility while assigning the responsibilities to objects.
- It allows a set of classes to act as one; events produced in one class can be sent to other handler classes with the help of composition.

Usage of Chain of Responsibility Pattern:

It is used:

- When more than one object can handle a request and the handler is unknown.
- When the group of objects that can handle the request must be specified in dynamic way.

Command Pattern

A Command Pattern says that "*encapsulate a request under an object as a command and pass it to invoker object. Invoker object looks for the appropriate object which can handle this command and pass the command to the corresponding object and that object executes the command*".

It is also known as **Action or Transaction.**

Advantage of command pattern

- It separates the object that invokes the operation from the object that actually performs the operation.
- It makes easy to add new commands, because existing classes remain unchanged.

Usage of command pattern:

It is used:

- When you need parameterize objects according to an action perform.
- When you need to create and execute requests at different times.
- When you need to support rollback, logging or transaction functionality.

Interpreter Pattern

An Interpreter Pattern says that **"to define a representation of grammar of a given language, along with an interpreter that uses this representation to interpret sentences in the language".**

Basically the Interpreter pattern has limited area where it can be applied. We can discuss the Interpreter pattern only in terms of formal grammars but in this area there are better solutions that is why it is not frequently used.

This pattern can applied for parsing the expressions defined in simple grammars and sometimes in simple rule engines.

SQL Parsing uses interpreter design pattern.

Advantage of Interpreter Pattern

- It is easier to change and extend the grammar.
- Implementing the grammar is straightforward.

Usage of Interpreter pattern:

It is used:

- When the grammar of the language is not complicated.
- When the efficiency is not a priority.

Iterator Pattern

According to GoF, Iterator Pattern is used **"to access the elements of an aggregate object sequentially without exposing its underlying implementation".**

The Iterator pattern is also known as **Cursor.**

In collection framework, we are now using Iterator that is preferred over Enumeration.

ava.util.Iterator interface uses Iterator Design Pattern.

Advantage of Iterator Pattern

- It supports variations in the traversal of a collection.
- It simplifies the interface to the collection.

Usage of Iterator Pattern:

It is used:

- When you want to access a collection of objects without exposing its internal representation.
- When there are multiple traversals of objects need to be supported in the collection.

Mediator Pattern

A Mediator Pattern says that "to define an object that encapsulates how a set of objects interact".

When we begin with development, we have a few classes and these classes interact with each other producing results. Now, consider slowly, the logic becomes more complex when functionality increases. Then what happens? We add more classes and they still interact with each other but it gets really difficult to maintain this code now. So, Mediator pattern takes care of this problem.

Mediator pattern is used to reduce communication complexity between multiple objects or classes. This pattern provides a mediator class which normally handles all the communications between different classes and supports easy maintainability of the code by loose coupling.

Benefits:

- It decouples the number of classes.
- It simplifies object protocols.

- It centralizes the control.
- The individual components become simpler and much easier to deal with because they don't need to pass messages to one another.

  The components don't need to contain logic to deal with their intercommunication and therefore, they are more generic.

Usage:

- It is commonly used in message-based systems likewise chat applications.
- When the set of objects communicate in complex but in well-defined ways.

Memento Pattern

A Memento Pattern says that "to restore the state of an object to its previous state". But it must do this without violating Encapsulation. Such case is useful in case of error or failure.

The Memento pattern is also known as **Token**.

Undo or backspace or ctrl+z is one of the most used operation in an editor. Memento design pattern is used to implement the undo operation. This is done by saving the current state of the object as it changes state.

Benefits:

- It preserves encapsulation boundaries.
- It simplifies the originator.

Usage:

- It is used in Undo and Redo operations in most software.
- It is also used in database transactions.

Observer Pattern

An Observer Pattern says that "just define a one-to-one dependency so that when one object changes state, all its dependents are notified and updated automatically".

The observer pattern is also known as Dependents or Publish-Subscribe.

Benefits:

- It describes the coupling between the objects and the observer.
- It provides the support for broadcast-type communication.

Usage:

- When the change of a state in one object must be reflected in another object without keeping the objects tight coupled.
- When the framework we writes and needs to be enhanced in future with new observers with minimal chamges.

State Pattern

A State Pattern says that "the class behavior changes based on its state". In State Pattern, we create objects which represent various states and a context object whose behavior varies as its state object changes.

The State Pattern is also known as Objects for States.

Benefits:

- It keeps the state-specific behavior.
- It makes any state transitions explicit.

Usage:

- When the behavior of object depends on its state and it must be able to change its behavior at runtime according to the new state.
- It is used when the operations have large, multipart conditional statements that depend on the state of an object.

Strategy Pattern

A Strategy Pattern says that "defines a family of functionality, encapsulate each one, and make them interchangeable".

The Strategy Pattern is 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.

Template Pattern

A Template Pattern says that "just define the skeleton of a function in an operation, deferring some steps to its subclasses".

Benefits:

- It is very common technique for reusing the code.This is only the main benefit of it.

Usage:

- It is used when the common behavior among sub-classes should be moved to a single common class by avoiding the duplication.

Visitor Pattern

In Visitor pattern, we use a visitor class which changes the executing algorithm of an element class. By this way, execution algorithm of element can vary as and when visitor varies. This pattern comes under behavior pattern category. As per the pattern, element object has to accept the visitor object so that visitor object handles the operation on the element object.

Null Object Pattern

In Null Object pattern, a null object replaces check of NULL object instance. Instead of putting if check for a null value, Null Object reflects a do nothing relationship. Such Null object can also be used to provide default behavior in case data is not available.

In Null Object pattern, we create an abstract class specifying various operations to be done, concrete classes extending this class and a null object class providing do nothing implementation of this class and will be used seamlessly where we need to check null value.

GRASP Patterns

- GRASP stands for General Responsibility Assignment Software Patterns
- guides in assigning responsibilities to collaborating objects.
- 9 GRASP patterns

  Creator

  Information

  Expert

  Low Coupling

  Controller

  High Cohesion

  Indirection

  Polymorphism

  Protected Variations

  Pure Fabrication

Responsibility:

Responsibility can be:

– accomplished by a single object.

– or a group of object collaboratively accomplish a responsibility

- GRASP helps us in deciding which responsibility should be assigned to which object/class.
- Identify the objects and responsibilities from the problem domain, and also identify how objects interact with each other.
- Define blue print for those objects – i.e. class with methods implementing those responsibilities.

Creator

- Creator decides who creates an Object ? Or who should createa new instance of some class?
- “Container” object creates “contained” objects.
- Decide who can be creator based on theobjects association and their interaction.

Consider VideoStore and Video in that store.

- VideoStore has an aggregation association with Video. I.e., VideoStore is the container and the Video is the contained object.
- So, we can instantiate video object in VideoStore class

Expert

- Given an object O, which responsibilities canbe assigned to O?
- Expert principle says – assign those responsibilities to o for which o has the information to fulfill that responsibility.
- They have all the information needed to perform operations, or in some cases they collaborate with others to fulfill their responsibilities.

Assume we need to get all the videos of aVideoStore.

- Since VideoStore knows about all the videos,we can assign this responsibility of giving all the videos can be assigned to VideoStore class.

- VideoStore is the information expert.

Low Coupling

- How strongly the objects are connected to each other? i.e. Coupling
- Coupling – object depending on other object.
- When depended upon element changes, it affects the dependantalso.
- Low Coupling – How can we reduce the impact of change independed upon elements on dependant elements.
- Prefer low coupling – assign responsibilities so that coupling remainlow.
- Minimizes the dependency hence making system maintainable,efficient and code reusable

Two elements are coupled, if

  - One element has aggregation/compositionassociation with another element.
  - One element implements/extends other element.

Controller

- Deals with how to delegate the request from the UI (User Interface) layerobjects to domain layer objects.
- when a request comes from UI layer object, Controller pattern helps us in determining what is that first object that receive themessage from the UI layer objects.
- This object is called controller object which receives requestfrom UI layer object and then controls/coordinates with otherobject of the domain layer to fulfill the request.
- It delegates the work to other class and coordinates theoverall activity.

We can make an object as Controller, if

- Object represents the overall system (facade controller)
- Object represent a use case, handling a sequence of operations(session controller).

Benefits

- can reuse this controller class.
- Can use to maintain the state of the use case.
- Can control the sequence of the activities

Bloated Controllers

Controller class is called bloated, if

- The class is overloaded with too many responsibilities

  Solution –Add more controllers
- Controller class also performing many tasksinstead of delegating to other class.

  Solution – controller class has to delegate thingsto others.

High Cohesion

- How are the operations of any element are functionallyrelated? i.e. Cohesion
- Related responsibilities in to one manageable unit.
- Prefer high cohesion
- Clearly defines the purpose of the element
- Benefits
  - Easily understandable and maintainable.

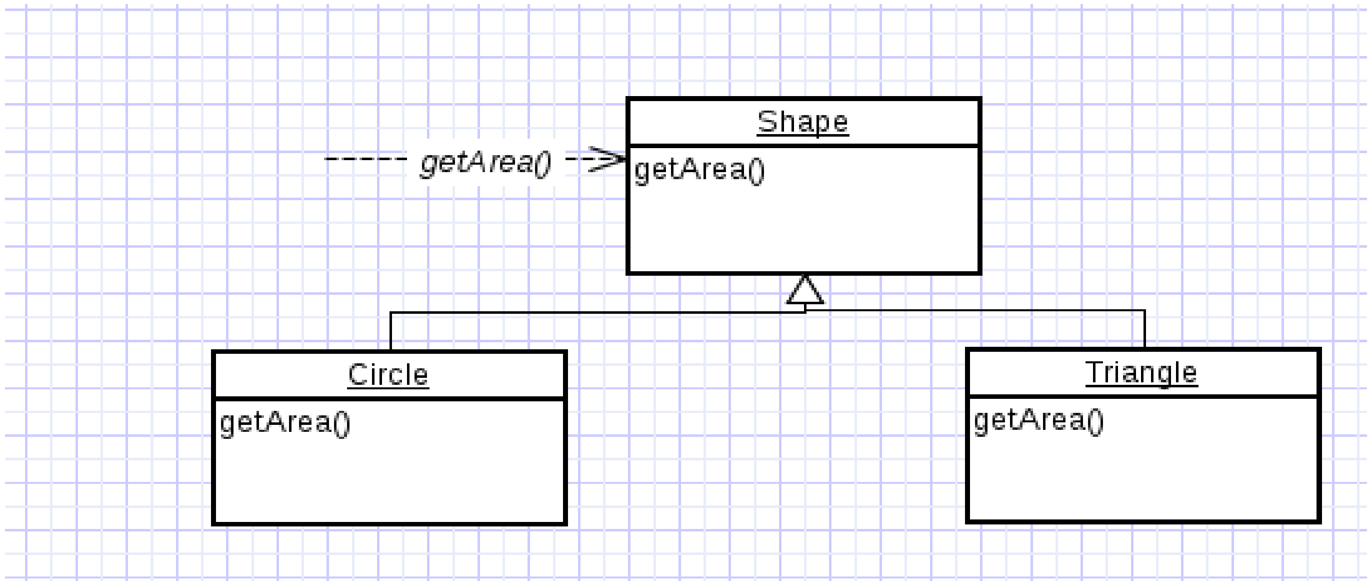
- Code reuse
- Low coupling

Polymorphism

- Polymoprphism allows to handle related but varying elements based on element type.
- Polymorphism guides us in deciding whichobject is responsible for handling those varying elements.
- Benefits: handling new variations will becomeeasy.

Example for Polymorphism

The getArea() varies by the type of shape, so we assign thatresponsibility to the subclasses.



By sending message to the Shape object, a call will be madeto the corresponding sub class object – Circle or Triangle.

Pure Fabrication

- Fabricated class/ artificial class – assign set ofrelated responsibilities that doesn't represent any domain object.
- Provides a highly cohesive set of activities.
- Behavioral decomposed – implements somealgorithm.
- Examples:Adapter, Strategy
- Benefits: High cohesion, low coupling and canreuse this class.

Example

- Suppose we Shape class, if we must store the shapedata in a database.
- If we put this responsibility in Shape class, there will bemany database related operations thus making Shape incohesive.
- So, create a fabricated class DBStore which is responsible to perform all database operations.
- Similarly logInterface which is responsible for logging information is also a good example for Pure Fabrication.

Indirection

- To avoid a direct coupling betweentwo or more elements is known as Indirection.
- Indirection introduces an intermediate unit to communicate between the other units, so that the other units are not directly coupled.
- Benefits: low coupling
- Example:Adapter, Facade, Obserever

Protected Variation

- It allows to avoid impact of variations of someelements on the other elements.
- It provides a well defined interface so that thethere will be no affect on other units.
- Provides flexibility and protection fromvariations.
- Provides more structured design.
- Example: polymorphism, data encapsulation,interfaces

Model View Controller (MVC) Pattern

MVC is known as an architectural pattern, which embodies three parts Model, View and Controller, or to be more exact it divides the application into three logical parts: the model part, the view and the controller.

Model designs based on MVC architecture follow the MVC design pattern and they separate the application logic from the user interface when designing software. As the name implies MVC pattern has three layers, which are:

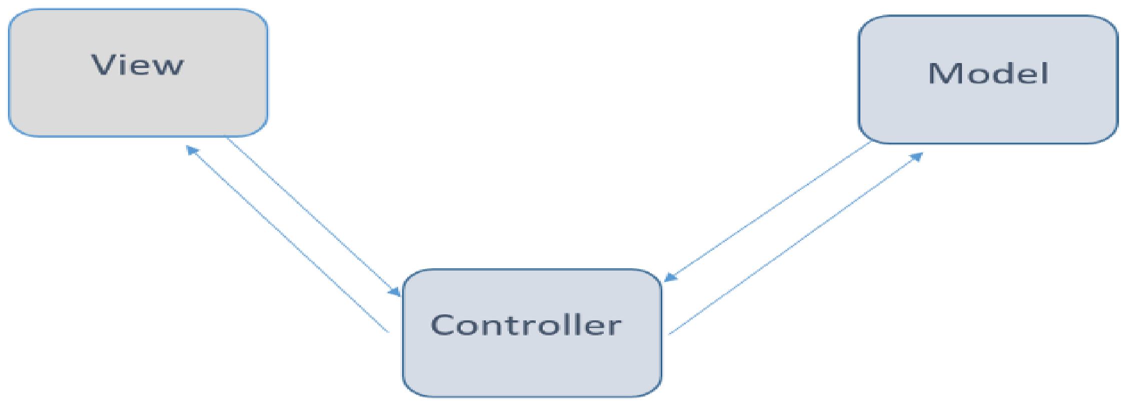
- Model: Data (Represents the business layer of the application)
- View: An interface to view and modify data (Defines the presentation of the application)
- Controller: Operations that can be performed on the data (Manages the flow of the application)

The model represents the data, and does nothing else. The model does NOT depend on the controller or the view.

The view displays the model data, and sends user actions (e.g. button clicks) to the controller. The view can be independent of both the model and the controller

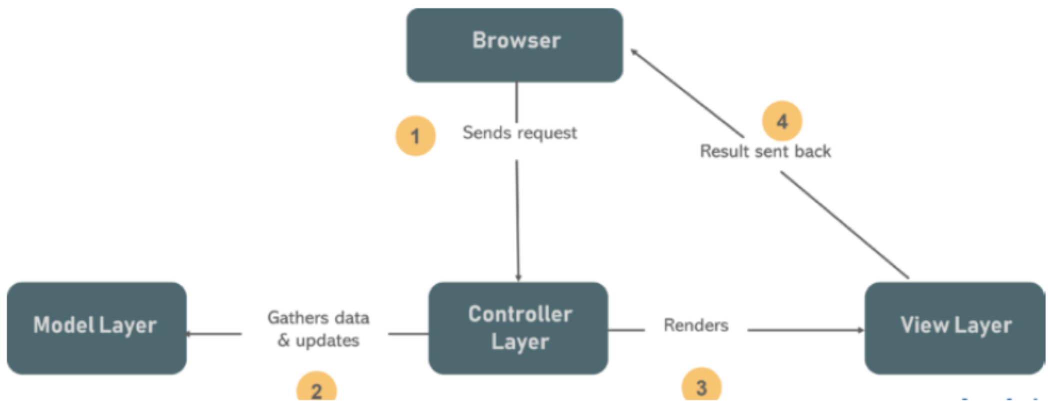
The controller provides model data to the view, and interprets user actions such as button clicks. The controller depends on the view and the model.

The main reasons why MVC is used are: First, it doesn't allow us to repeat ourselves and second, it helps to create a solid structure of our web applications.



In Java Programming context, the Model consists of simple Java classes, the View displays the data and the Controller consists of servlets. This separation results in user requests being processed as follows:

1. The browser on the client sends a request for a page to the controller present on the server
2. The controller performs the action of invoking the model, thereby, retrieving the data it needs in response to the request
3. The controller then gives the retrieved data to the view
4. The view is rendered and sent back to the client for the browser to display



Advantages of MVC Architecture in Java

MVC architecture offers a lot of advantages for a programmer when developing applications, which include:

- Multiple developers can work with the three layers (Model, View, and Controller) simultaneously
- Offers improved *scalability*, that supplements the ability of the application to grow

- As components have a low dependency on each other, they are easy to maintain
- A model can be reused by multiple views which provides reusability of code
- Adoption of MVC makes an application more expressive and easy to understand
- Extending and testing of the application becomes easy