At a glance SDA

UNIT 1 Introduction to Software Design

Good Software is determined on the basis of

1. Quality of product
    a. Correctness
    b. Reliability
    c. Usability
    d. Testability
    e. Maintainability
2. Quality of process
    a. Code reviews
    b. Requirements understanding
    c. Testing
3. Quality in the context of business environment
    a. Return on investment
    b. Express in effort
        i. Schedule
        ii. Productivity
        iii. Customer

System Approach

```
    System is collection of entities and activities plus description of
relationships between entities and activities
```

Design Methods

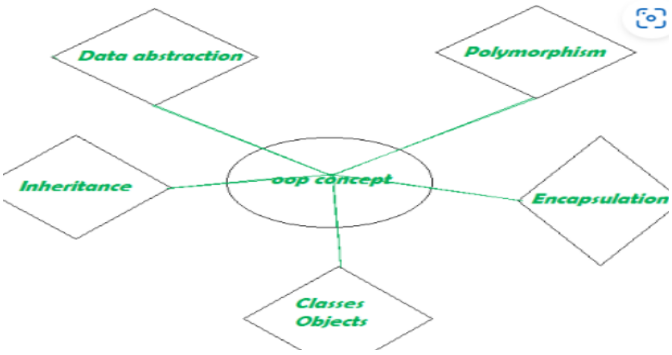
1. Procedural and structural design methods are two different approaches to software engineering and civil engineering respectively

2. Both methods aim to achieve efficiency, reliability, and safety in their respective domains

```
1. Procedural

    Procedural design is a technique that transforms structural
components into a procedural description of the software

2. Structural Design methods

   Structural design is a process that determines the shape, size, and
arrangement of structural members that can withstand external loads
```

Object Oriented design method

1. Object-oriented design method is an approach to software engineering that views the system as a collection of interacting objects
2. Here are some diagrams that illustrate some of the concepts and principles of object-oriented design



   a. **Encapsulation**: This diagram shows how an object can hide its data and methods from the outside world by using access modifiers such as public, private, protected, etc. The object can also provide a public interface that defines how other objects can interact with it.
   b. **Inheritance**: This diagram shows how a class can inherit attributes and methods from another class (called superclass or parent class) and also add its own attributes and methods (called subclass or child class). The subclass can also override some of the methods inherited from the superclass to provide different behavior
   c. **Polymorphism**: This diagram shows how different subclasses can share a common interface (defined by an abstract class or an interface) and provide different implementations of the same method. This allows objects of different types to be treated uniformly by using a reference of the common interface.
   d. **Adapter**: This diagram shows how an adapter class can act as a bridge between two incompatible interfaces. The adapter class implements one interface and

uses an instance of another interface to perform the required operations. This allows objects that use different interfaces to work together

e. **Composite**: This diagram shows how a composite class can represent a hierarchy of objects that share a common interface. The composite class can contain other objects (called components) that implement the same interface. The composite class can also implement methods that delegate operations to its components. This allows objects that have part-whole relationships to be treated uniformly

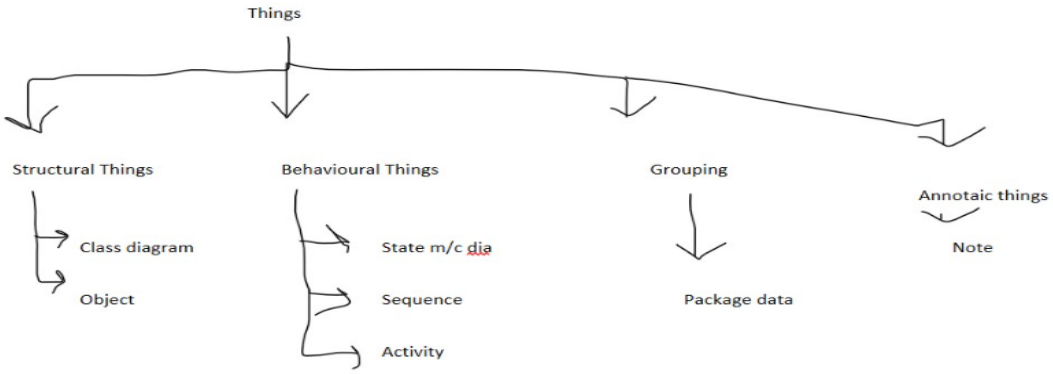
Unified Modeling Language

1. UML is generalized modeling language distinct from other languages like c,c++
2. There are a total of **9 UML Diagrams**
    a. Use case diagram
    b. Class diagram
    c. Object diagram
    d. State diagram
    e. Activity diagram
    f. Sequence diagram
    g. Collaboration diagram
    h. Component diagram
    i. Deployment diagram
3. They can be classified as

```
    A. Static: the structural aspect of the system, define what parts
the system is made up of.
    B. Dynamic: The behavioral features of a system; for example, the
ways a system behaves in response to certain events or actions are the
dynamic characteristics of a system.
    C. Implementation: The implementation characteristic of a system is
an entirely new feature that describes the different elements required for
deploying a system.
```

4. UML has two things
    a. Diagram
    b. Notation
5. UML Building block
    a. They can be classified things, relationships & diagrams each explained separately

Things: They can be classified as



b. Structural things
   - It includes class & object
     1. Classes
        a. In object-oriented programming, a class is a template or blueprint that defines the attributes and methods of a type of object. A class can be used to create multiple instances of objects that share the same characteristics and behavior.In a class diagram classes are represented
        b. In UML, a class is a graphical representation of an object-oriented concept that shows its name, attributes, and operations in a rectangular box
        c. Default definition: A Class is a set of identical things that outlines the functionality and Attributes of an object. It also represents the abstract class whose functionalities are not defined
        d. It looks like when represented

```
+------------+
| Name       |
+------------+
| Functions  |
+------------+
| Variables  |
+------------+
```

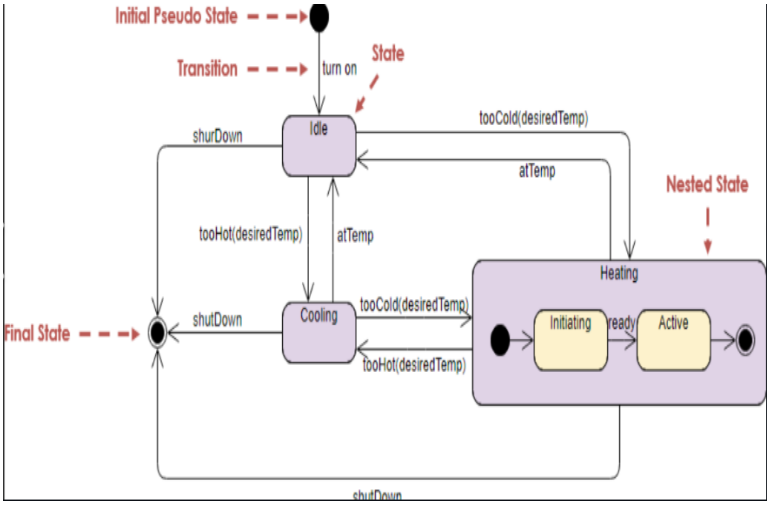
     2. Objects
        a. An individual that describes the behavior and the functions of a system
        b. In UML, an object is a graphical representation of an instance of a class that shows its name (optional), class

name (underlined), and attribute values (if any) in a rectangular box

c. An object can also show its relationships with other objects through different types of lines and symbols
d. The notation of the object is similar to that of the class

c. Behavioral Things
- They are the verbs that encompass the dynamic parts of a model
- It contains state machine & activity diagram

1. State machine
   a. It defines a sequence of states that an entity goes through in the software development lifecycle
   b. A state machine is a specification of the dynamic behavior of individual class objects, use cases, and entire systems.
   c. Basic components are mentioned in the state diagram given below



2. Activity Diagram
   a. It portrays all the activities accomplished by different entities of a system. It is represented the same as that of a state machine diagram
   b. Used to show the message flow from one activity to another activity
   c. It consists of an initial state, final state, a decision box, and an action notation
   d. Shows flow control within a system
   e. Has two types { **Action state**: It cannot be decomposed further & **Activity state**: Can be decomposed further }

d. Grouping Things

```
i.  A method that together binds the elements of the UML model.
```

```
    ii. In UML, the package is the only thing, which is used for grouping

    iii. Package: Package is the only thing that is available for grouping
behavioral and structural things
```

e. Annotating Things

```
    i. It is a mechanism that captures the remarks, descriptions, and
comments of UML model elements. In UML, a note is the only Annotational
thing

    ii. It is used to attach the constraints, comments, and rules to the
elements of the model. It is a kind of yellow sticky note.
```

Relationships in UML

a. Dependency

- Dependency is a kind of relationship in which a change in target element affects the source element, or simply we can say the source element is dependent on the target element
- It depicts the dependency from one entity to another
- It is denoted by a dotted line followed by an arrow at one side as shown below

```
          --->
```

b. Association

- It tells how many elements are actually taking part in forming that relationship
- Association is a structural relationship that represents how two entities are linked or connected to each other within a system
- It can form several types of associations, such as one-to-one, one-to-many, many-to-one, and many-to-many
- It has been categorized into four types of associations (Bi-directional, unidirectional, aggregation (composition aggregation), and reflexive)

```
Note:
```

```
    An aggregation is a special form of association and Composition is a special form of aggregation
```

c. Aggregation

- An aggregation is a special form of association
- It portrays a part-of relationship. It forms a binary relationship, which means it cannot include more than two classes. It is also known as Has-a relationship
- It forms a weak association
- Consider the following example

```
            Example - A doctor has patients when the doctor gets transfer to another hospital, the patients do not go to a new workplace
```

d. Composition

- In a composition relationship, the child depends on the parent. It forms a two-way relationship. It is a special case of aggregation
- It is known as Part-of relationship
- It forms a strong association
- Consider the example below

```
            Example - A hospital and its wards. If the hospital is destroyed, the wards also get destroyed
```

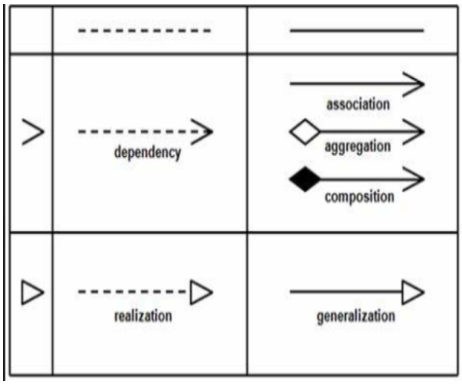
e. Generalization

- It portrays the relationship between a general thing (a parent class or superclass) and a specific kind of that thing (a child class or subclass)

f. Realization

- It is a semantic kind of relationship between two things, where one defines the behavior to be carried out, and the other one implements the mentioned behavior

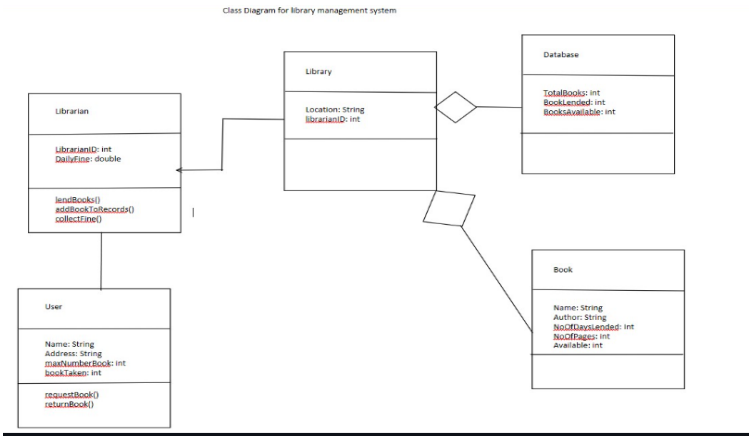
```
Note: Refer this for all kind of relationships
```



Diagram

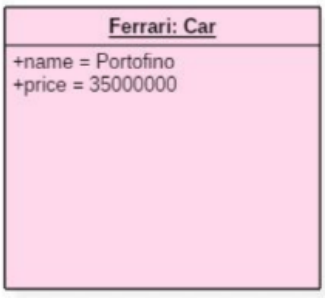
Class diagram

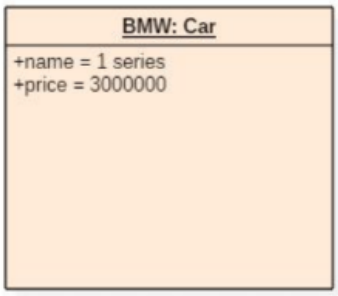
- Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application
- A collection of class diagrams represent the whole system



Object Diagram

- Object diagrams are dependent on the class diagram as they are derived from the class diagram





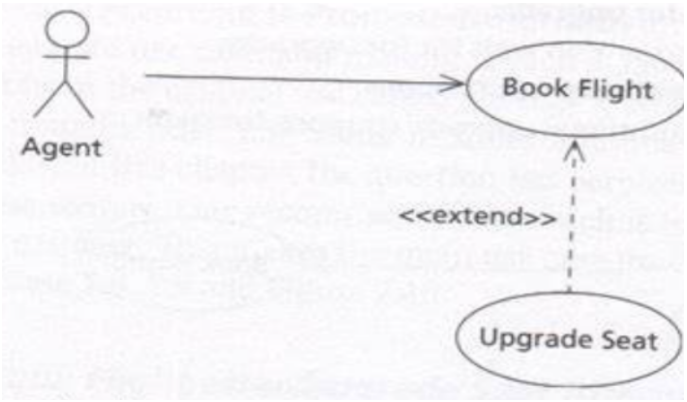
Use Case Diagram

- Actors represent roles, that is, a type of user of the system
- Use cases represent a sequence of interaction for a type of functionality
- The use case model is the set of all use cases. It is a complete description of the functionality of the system and its environment

Note:

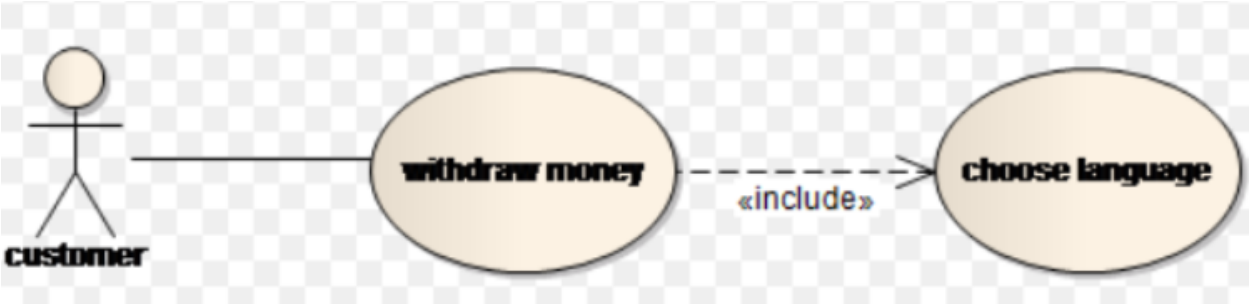
The <<extend>> Relationship

- <<extend>> relationships represent exceptional cases.
- The exceptional event flows are factored out of the main event flow for clarity.
- Use cases representing exceptional flows can extend more than one use case.
- The direction of a <<extend>> relationship is to the extended use case



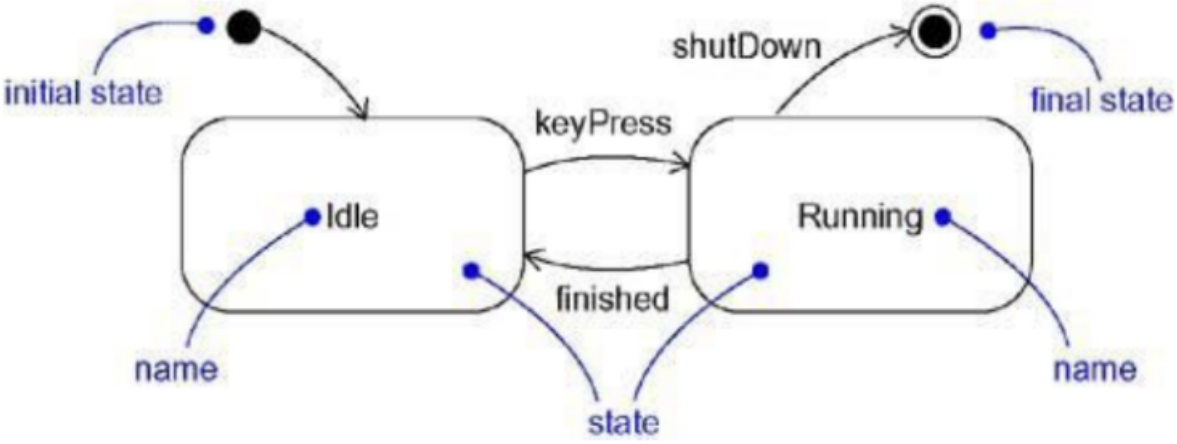
The <<include>> Relationship

- An <<include>> represents behavior that is factored out for reuse, not because it is an exception.
- The direction of a <<include>>relationship is to the using use case (unlike <<extend>> relationships).



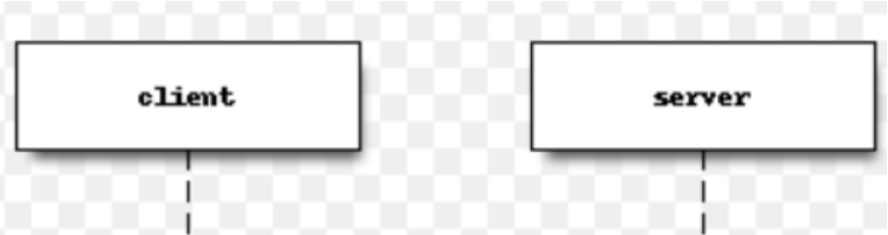
State Diagram

- To model dynamic aspect of the System.
- Define different state of an object during its lifetime.
- Described flow of control from one state to another state.
- These states are controlled by internal or external events.



Sequence Diagram

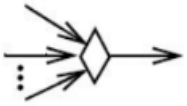
- A sequence diagram is a type of interaction diagram that shows how objects or parts of a system work together to perform a function or a use case
- It also shows the order or sequence of the interactions over time
- Sequence diagrams can help you understand the logic of complex processes, functions, or methods

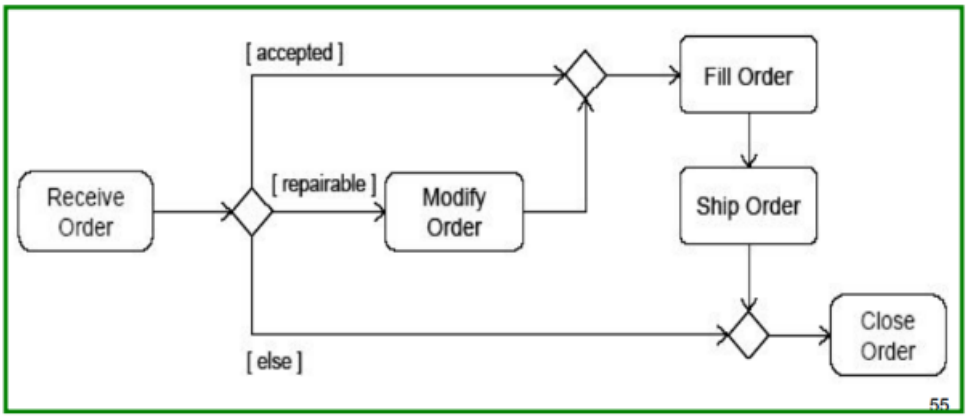


Note:

- Control nodes
  - **Merge Nodes**

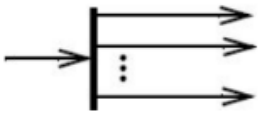
```
1. Bring together multiple alternate flows
2. All controls and data arriving at a merge node are immediately
   passed to the outgoing edge
3. There is no synchronization of flows or joining of tokens
```

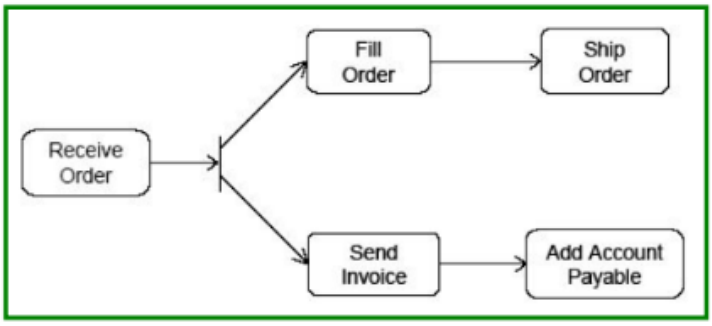




  - **Fork nodes**

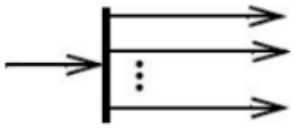
```
1. Fork nodes split flows into multiple concurrent flows (tokens are
   duplicated)
```

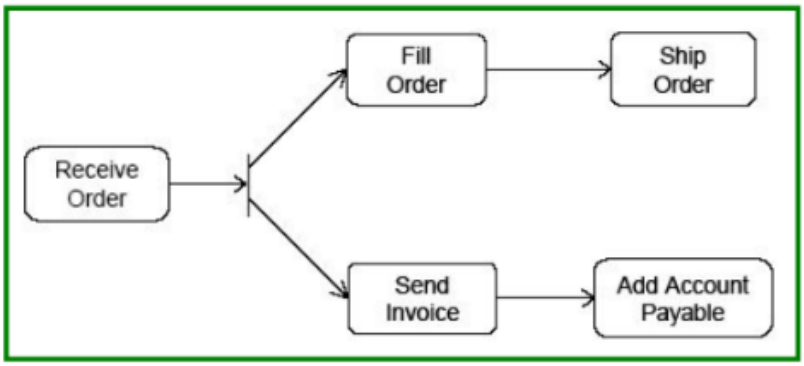




- **Join nodes**

```
1. Join nodes synchronize multiple flows
```





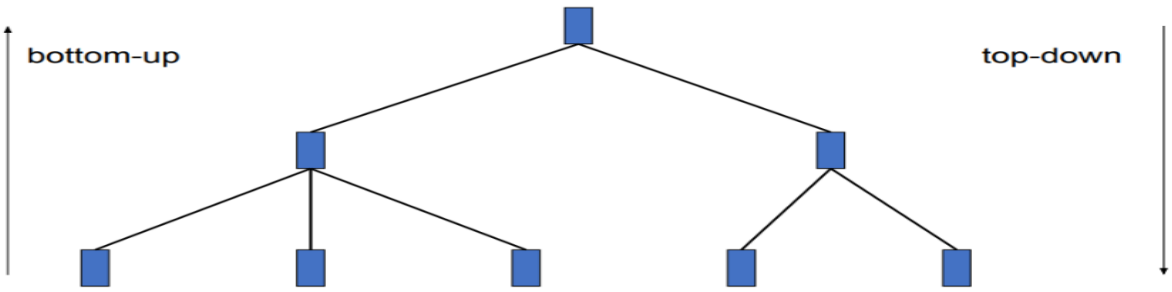
More Design Methods

```
• Design is a trial-and-error process
• The process is not the same as the outcome of that process
• There is an interaction between requirements engineering,
  architecting, and design
```

They are categorized into 4 Design methods

1. Functional decomposition

a. Functional decomposition is a method of breaking down a complex system or process into smaller, simpler components based on their specific functions
b. It helps to simplify the design, analysis, and implementation of complex systems by identifying the functions and relationships between them



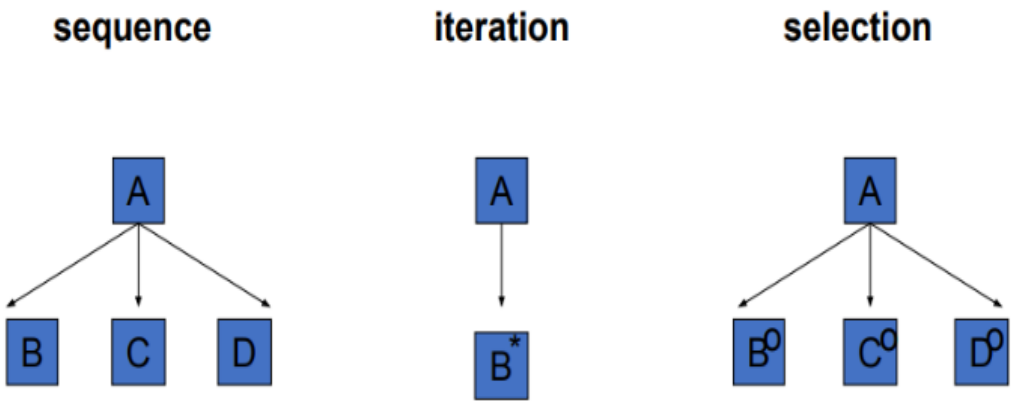
2. Data Flow Design (SA/SD)

- Result of SA: logical model, consisting of a set of DFD's, augmented by

minispecs, data dictionary, etc.

- Structured Design = transition from DFD's to structure charts
- Heuristics for this transition are based on notions of coupling and cohesion
- Major heuristic concerns choice for top-level structure chart, most often: transform-centered

3. Design based on Data Structures (JSD/JSP)
    - JSD (Jackson Structured Design)
        - **Used on large scale**
        - JSD has three stages
            - **Modeling stage**: description of real world problem in terms of entities and actions
            - **Network stage**: model system as a network of communicating processes
            - **Implementation stage**: transform network into a sequential design
        - JSD tries to fill in the gap of JSP which we will discuss further
        - This life cycle is depicted as a Process Structure Diagram (PSD); these resemble JSP's structure diagrams
    - JSP (Jackson Structured Programming)
        - Basic idea: good program reflects structure of its input and output
        - Program can be derived almost mechanically from a description of the input and output
        - Input and output are depicted in a structure diagram and/or in structured text/schematic logic (a kind of pseudocode)
        - Three basic compound forms: **sequence, iteration, and selection**



        - The problem: **how to obtain a mapping from the problem structure to the data structure?**
    - OO design method

- Booch: early, new and rich set of notations
- Fusion: more emphasis on process
- RUP: full life cycle model associated with UML

Classification of design methods

	problem-oriented	product-oriented
conceptual	I ER modeling Structured analysis	II Structured design
formal	III JSD VDM	IV Functional decomposition JSP

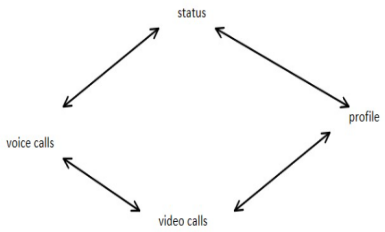
UNIT 2 Architectural Design

Architecture

```
Architecture is nothing but a blue print that defines a system
```

Software Architecture

a. It is blueprint for the system
b. Abstraction to manage system and establish communication and coordination among components



c. It refers to the fundamental structure of the software and the disciplines to create such software

Note: All software architecture are software design but not all software design are software architecture

d. Software architecture affects the quality of software. They are directly proportional
e. Why use software architecture?
    i. It helps in real world complex world
    ii. We can distribute work between team members
    iii. Helps in understanding the system
f. Users of software Architecture
    i. Project Managers
    ii. Software Dev
    iii. Tester
    iv. Anyone who wants to make an improvement by looking at architecture

Software Design

It is mechanism to transform user requirements into something useful

```
Objectives

    i. Correctness
    ii. Completeness
    iii. Efficient
```

Architectural Design vs Software Design

Software Design	Architectural Design
Sub modules are present	Only overall model is designed
Implementation is there somewhat	Implementation is hidden

Architectural Design

- **System structuring**
- **Control Modeling**
- **Modular decomposition**

Subsystem & modules

- Sub system: It is a system own its own **doesn't depend on other sub system**
- Module: It is a component which **provides services but it cannot be considered as sub systems**

Architectural Design Process

1. Understand the problem
2. Identify design elements
3. Evaluate the Design architecture
4. Transform the architecture

Types of Architectures

- **Business architecture**: Design the business strategy
- **Application architecture**: servers as a blueprint for individual application system
- **Information technology architecture**: Define hardware and software for that system

Architecture Modules

- It involves high level structure of software system abstraction by using decomposition
- Its parts are
    - UML
        - Pictorial language use to make blueprints
        - Divided into two sub categories
            - Structural (class diagram,object diagram,component diagram, Package diagram, Deployment diagram)

- Behavioural (Sequence diagram, State diagram, Activity diagram)

Architectural View Model (4+1 view model)

- It is used to describe the system from the viewpoint of different stakeholders
- The 4 + 1 view model was designed by philippe to describe the architecture of software
- It is a multiple view model

Types of views

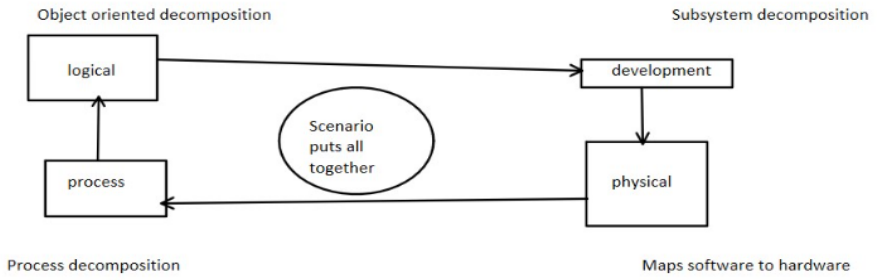
1. Logical View
    a. Shows the component (object) or system as well as their interaction
    b. Viewer are end-users, analyst
    c. Considers functional requirements
    d. UML diagram (class, state, object, communication diagram)

2. Process
    a. Shows process and how processes communicate
    b. Integrators & developers are viewers
    c. Considers non functional requirements
    d. Activity diagram is drawn

3. Development
    a. Gives a building block of system and describe static organization system modules

4. Physical
    a. Shows installation, configuration of software

5. Scenario
    a. Shows design is complete by performing validation



ADL (Architecture Description language)

- It provides syntax and semantics for defining architecture of software
- It defines interfaces between components
- An architecture description language is a formal specification language

Software Architecture Styles

a. Data Centered architecture

i. It has two distinct components
   1. A central data structure
   2. A collection of client software

ii. Main purpose is to achieve integrality of data

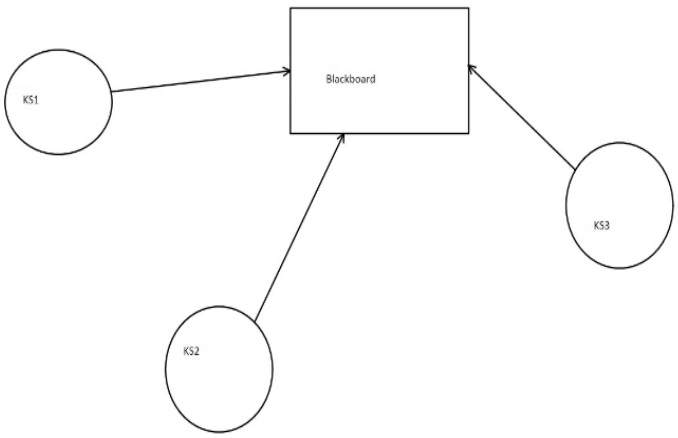
iii. Processes are individually executed by client components

iv. Stored data is accessed in order to perform operations on it

v. It adds scalability & supports modification

vi. Hospital management system is perfect example of it

- Blackboard Architecture Style
  - Provides Scalability which provides ease to update knowledge source
  - Supports reusability of knowledge source



- Repository Architecture Style
  - Consists of a central data store and a collection of independent components that operate on it

- The components can communicate only through the data store and do not have direct interaction with each other
- Repository architecture style is commonly used in database systems, library information systems, compilers, and computer-aided software engineering environments

b. Data flow architecture

```
        i. It is applied when input data is converted into a series of
manipulative components into output data
        ii. It's types
            I. Batch sequential (Provides simple division on subsystem
only problem is low throughput)
            II. Pipe and filter architecture (It transforms the data
stream, processes it and writes the transformed data stream over a pipe
for the next filter to process. Has two types passive and active filter)
            III. Process Control (The flow of data comes from a set of
variables, which controls the execution of process)
        iii. Some advantages of this architecture are
            I. It supports reusability & it is maintainable.
            II. It supports concurrent execution
```

c. Logical architecture

i. Logical architecture in software is a type of structural design that describes the organization and interaction of the software components without specifying the physical implementation details

ii. Logical architecture shows the logical structure of the system, such as subsystems, classes, layers, and interfaces

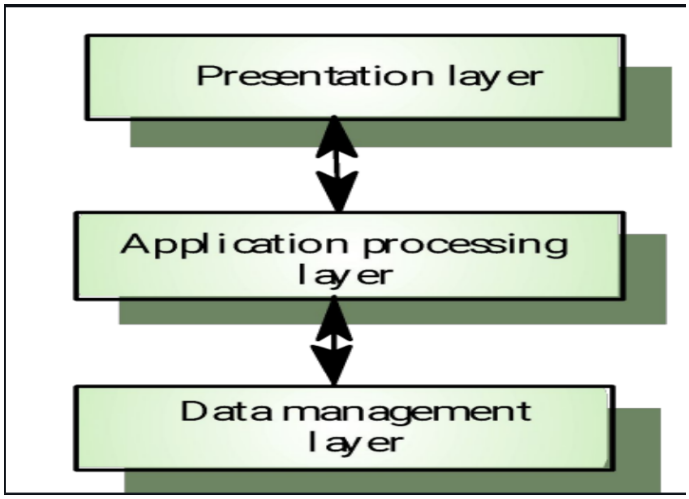
iii. Logical architecture helps to plan and communicate the system design before developing and deploying it

iv. Logical architecture can be represented by diagrams that use different symbols and notations to show the software components and their relationships

d. Layered Architecture

i. Organizes the system into layers with related functionality associated with each layer.
ii. It is an n-tier architecture pattern where the components are organized in horizontal layers
iii. A layer provides services to the layer above it.
iv. Components are interconnected but do not depend on each other
v. It has three types of layers
I. **Presentation layer** (Concerned with presenting the results of a computation to system users and with collecting user inputs)
II. **Application processing layer** (Concerned with providing application specific functionality e.g., in a banking system, banking functions such as open account, close account, etc.)
III. **Data management layer** (Concerned with managing the system database)

This is how the layer communicate amongst themselves



e. Hierarchical Architecture

i. Hierarchical architecture views the whole system as a hierarchy structure, in which the software system is decomposed into logical modules or subsystems at different levels in the hierarchy
ii. Hierarchical decomposition is a powerful abstraction mechanism
iii. It is divided into

I. **Main-subroutine:** This style divides a software system into subroutines by using top-down refinement according to desired functionality. The subroutines can be

reused and shared by multiple callers. Data can be passed as parameters to subroutines by value or by reference.

II. **Master-slave:** This style applies the 'divide and conquer' principle and supports fault computation and computational accuracy. It consists of a master component that distributes tasks to slave components that provide duplicate services. The master chooses a result among slaves by a certain selection strategy.

III. **Virtual machine:** This style simulates a physical machine or an abstract computing model by providing an interface that hides the implementation details. It consists of a virtual machine component that interprets instructions from an application component and executes them on a physical machine component.

f. Interaction-Oriented Architecture

1. Interaction-oriented architecture has two major styles

I. **Model-View-Controller (MVC)**

a. It is an independent user interface and captures the behavior of application problem domain

b. View can be used to represent any output of information in graphical form such as diagram or chart

II. **Presentation-Abstraction-Control (PAC)**

a. Each agent has three component (Presentation, abstraction & control)

b. The presentation component − Formats the visual and audio presentation of data

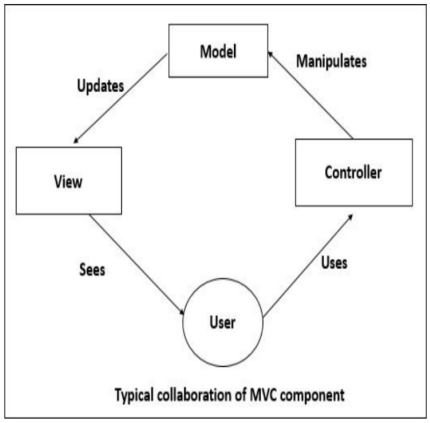
c. The abstraction component − Retrieves and processes the data

d. The control component − Handles the task such as the flow of control and communication between the other two components

e. Advantages: (Support for multi-tasking and multi-viewing, Support for agent reusability)

g. Disadvantages: (Overhead due to the control bridge between presentation and abstraction, Difficult to determine the right number of agents)

The Model-View-Controller diagram



Typical collaboration of MVC component

Distributed system architecture

```
Components are present on different platforms & several components can
cooperate with one another over a communication network in order to
achieve a specific objective objective or goal
```

Parallel & Distributed system

Parallel	Distributed
Shared memory	Separate memory
Communication via shared memory	Communication via messages

Client server Architecture

- Client: The one who requests

- Server: It receives the request & replies the client
- Middlewares are in between client
- There are different types of client models

1. Multiprocessor architectures

- Simplest distributed system model
- System composed of multiple processes which may (but need not) execute on different processors
- Architectural model of many large real-time systems

A. Fat client models

1. Server is the incharge of data

2. More processing is delegated to the client

3. Most suitable for new C/S systems where the capabilities of the client system are known in advance

4. More complex than a thin client model especially for management

B.Thin client models

1. Applications processing & data management is carried out by server

2. The client is simply responsible for running the presentation software.

3. Used when legacy systems are migrated to client server architectures.

4. A major disadvantage is that it places a heavy processing load on both the server and the network

2. DBMS Architecture

1st-tier

- The simple architecture is client server & database

2nd-tier

- Based on client server architecture application directly interact with database

3rd-tier

- Contains presentation layer, application layer & database
- It is more scalable

3. Three tier architecture

a. In a three-tier architecture, each of the application architecture layers may execute on a separate processor
b. Allows for better performance than a thin-client approach and is simpler to manage than a fat-client approach
c. A more scalable architecture - as demands increase, extra servers can be added

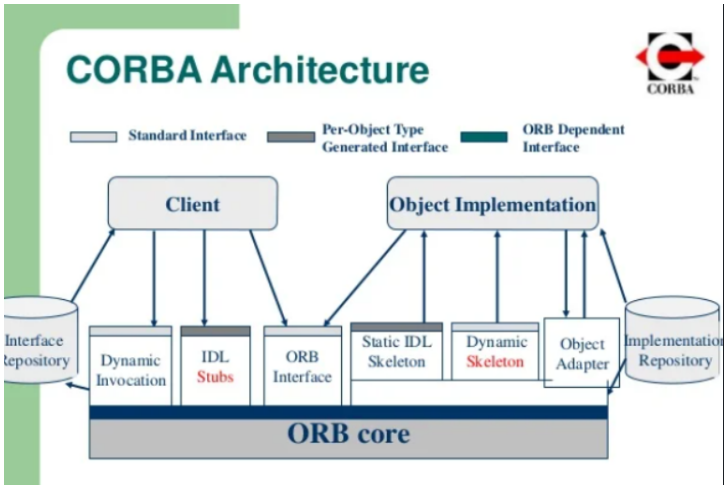
Distributed object architectures

- There is no distinction in a distributed object architectures between clients and servers
- Each distributable entity is an object that
- Provides services to other objects and receives services from other objects
- Object communication is through a middleware system called an object request broker (software bus)
- More complex to design than C/S systems
- Advantages
    - It allows the system designer to delay decisions on where and how services should be provided
    - It is a very open system architecture that allows new resources to be added to it as required
    - The system is flexible and scalable
    - It is possible to reconfigure the system dynamically

- Uses
    - As a logical model that allows you to structure and organise the system
    - As a flexible approach to the implementation of client-server systems

CORBA

- Distributed object based system
- Provides interoperability
- CORBA Architecture



- Communication takes place in IDL language here
- Communication is between client & server!

Broker

- It is responsible for coordinating request
- ORB provides the mechanism required for distributed object to communicate with one another, whether locally or on remote devices, written in different language, or at different locations on a network

Object Adapter

- Different kinds of object implementations.
- Object residing in their own process and requiring activation
- Others not requiring activation
- OA helps the ORB to operate with different type of object

Interface Repository

- Contains interface regarding the interface to ORB objects.
- Can be used by the ORB in 2 ways:
- To Provide type- checking of request signatures whether a request was issued through stub
- To check correctness of inheritance gaps

IDL

- In java we cannot separate a class definition from its implementation as we can in C++
- CORBA allows the separation of definition & implementation
- CORBA uses IDL for defining interface between clients and servers.
- ORB vendors provide specific IDL compiles for supported language

Component based concurrent & Real time software architecture with case studies

UNIT 3 Design Pattern

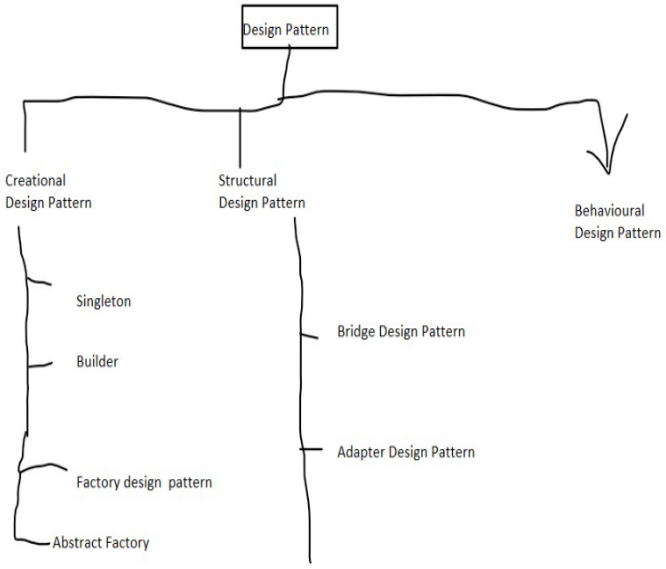
What is the design pattern?

1. General repeatable pattern
2. Not mandatory to always implement design pattern
3. They are generic, simple, well tested, reusable and object oriented in nature
4. We need to pick the right pattern for our project as there are 23 design patterns

5. It requires communicating object that includes name, abstract and key aspects and focuses on OOC Problems (Object Oriented Concepts)

The Gang of four (GOF)

- Four authors Erich Gamma, Richard Helm, Ralph Johnson and John V publish a book Design Pattern - Elements of reusable Object Oriented Software
- Their design patterns are categorized. These are known as gang of four.
- The Design pattern can be Categorized as follows:

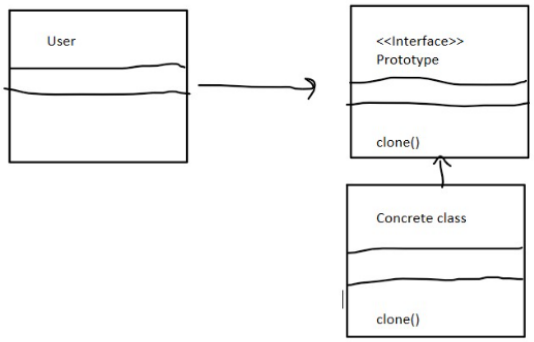


Description of Categories

1. Creational Design Patterns
   a. provides a way to create objects while hiding the creational logic rather than instantiating objects yourself directly
   b. It gives more flexibility
   c. It can be further categorized as singleton, builder, prototype, factory
2. Structural Design Patterns
   a. These design patterns concern class and object composition
   b. Those design patterns are about organizing classes and objects to a larger structure
   c. It uses Inheritance between classes
   d. Further divided as Bridge, Adapter
3. Behavioral Design Pattern
   a. Concerned with communication between objects
   b. Example chain of responsibility, Command, Interpreter, Iterator, Mediator, Memento etc

Creational Design Patterns Types

Prototype Design pattern



a. Here we are cloning the existing object instead of creating a new instance

b. We can customized it

c. We can add or remove objects at runtime

Builder Design

a. It helps us build complex java object in an easy and readable manner

b. It is used when we want to hide internal complex structure from external entities which use interface to create object

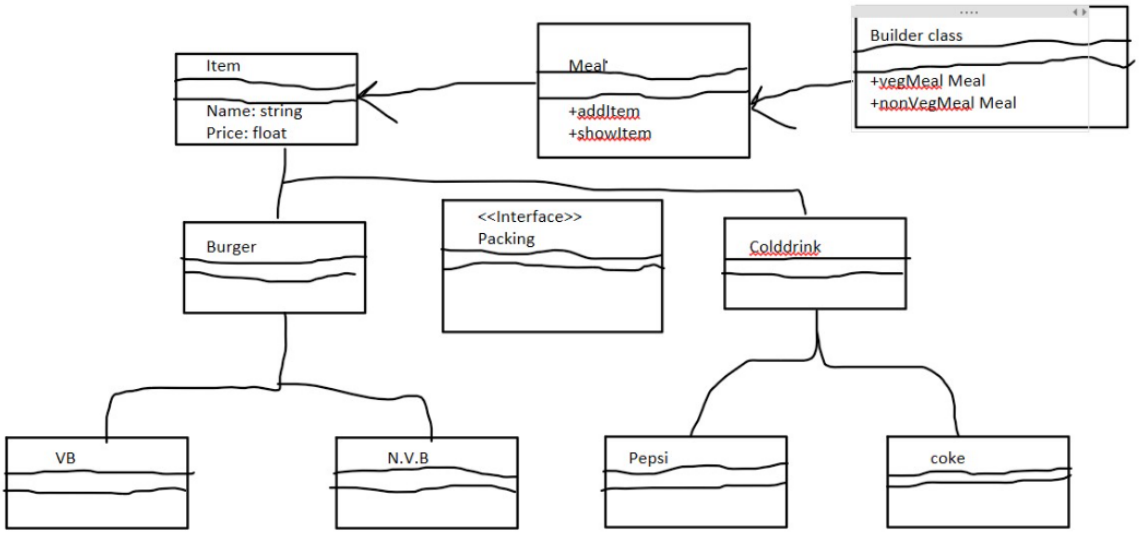
c. It separates object construction from its representation

d. It has 3 major parts

i. Product (This is the actual object that we are trying to construct)

ii. Builder (Builds complex part)

iii. Executor (Executes complex part)



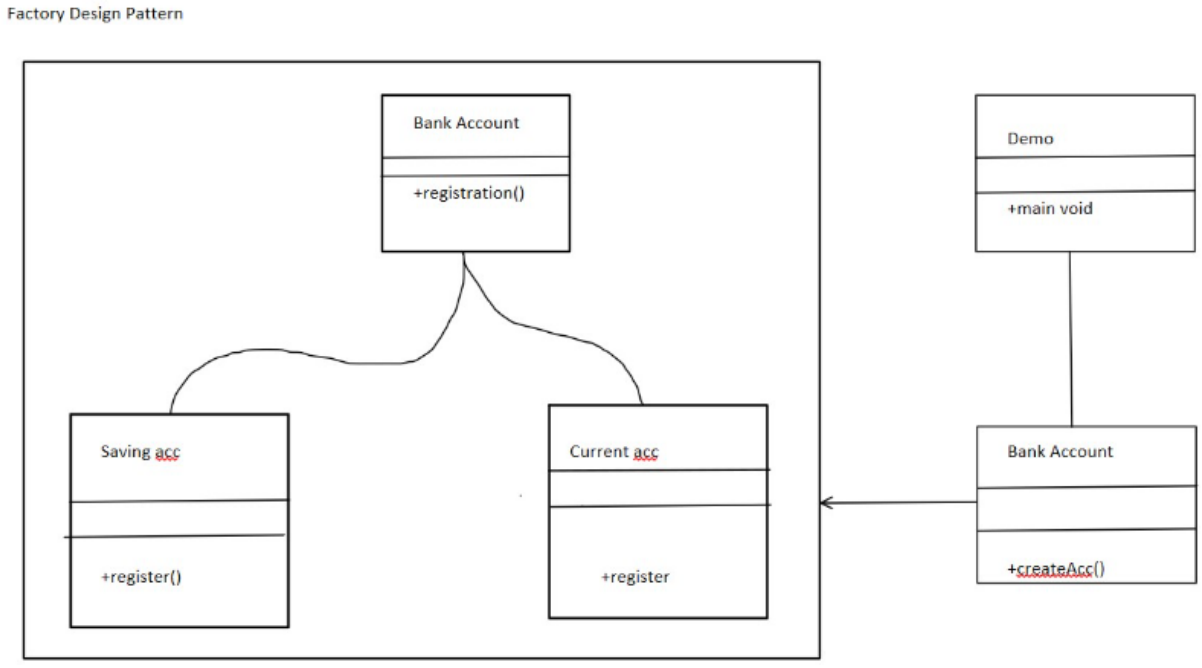
Factory Design Pattern

a. Define an interface or abs class for creating an object but let the subclasses decide which class to instantiate

b. The main purpose is to return objects at runtime

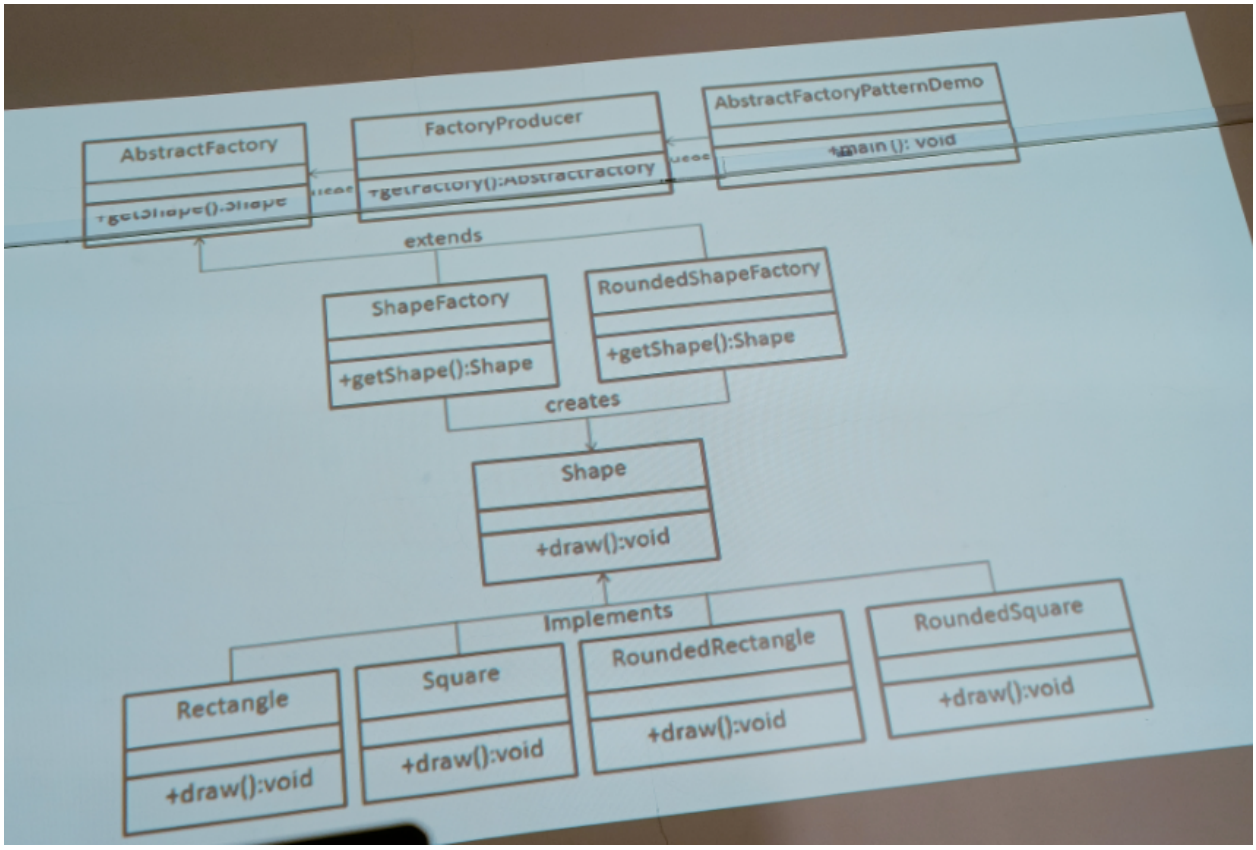
c. Helps us to separate the object creation code from the one which is using it.

d. Reuses the same memory



Abstract Factory Design Pattern

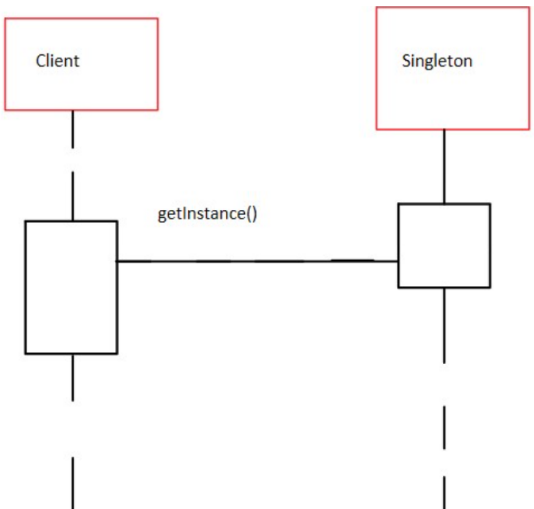
a. Similar to Factory Design Pattern
b. Add another level of abstraction layer to factory design pattern
c. It creates factory of factory i.e super factory



Singleton Design Pattern

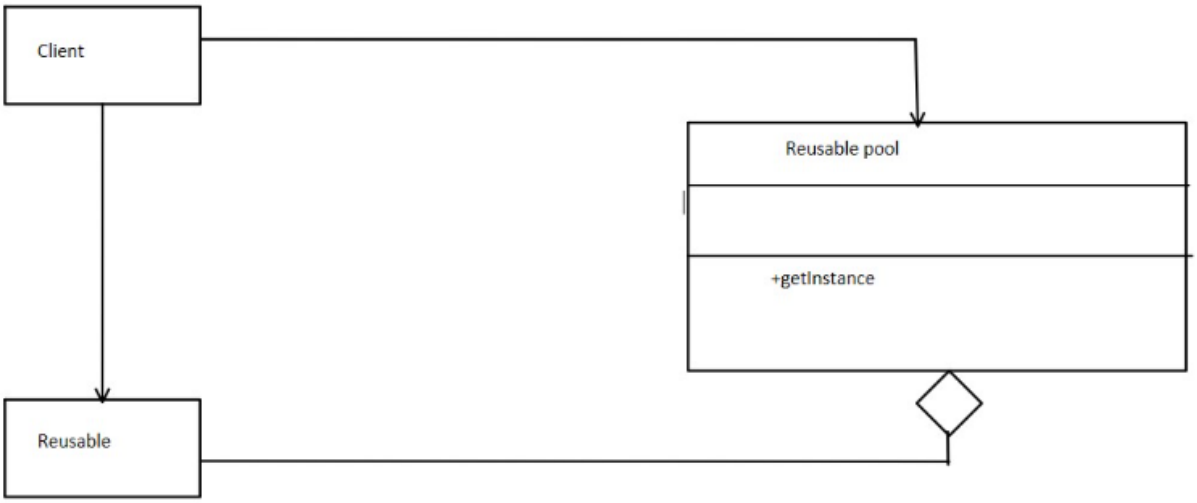
a. private constructor
b. Instance private
c. Global Method

```
        Public static Singleton
getInstance(){
            if(instance == null){
```



```
            instance = new Singleton();
            return instance;
            }
        }
```

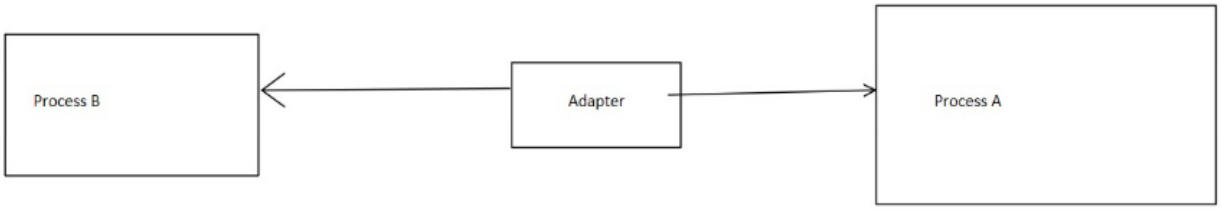
Object pool (Collection of objects)



The Structural Design Patterns

Adapter

a. Adapter is between two objects
b. Alternate form of class creation is possible
c. The pattern converts the incompatible interface of a class into another interface that client requires

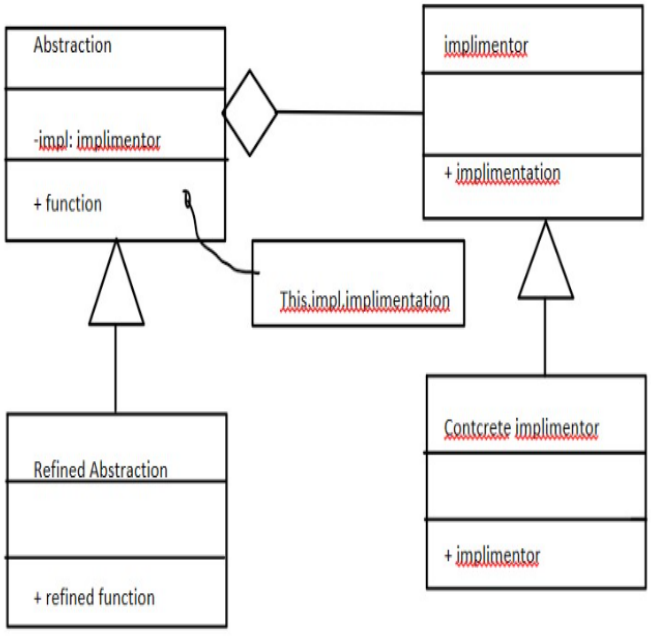


Decorator

a. Used to extend the functionality of the base class
b. It offers wrapper to existing class
c. Example In pizza class toppings can be added
d. It does not affects the behavior of other objects in the class

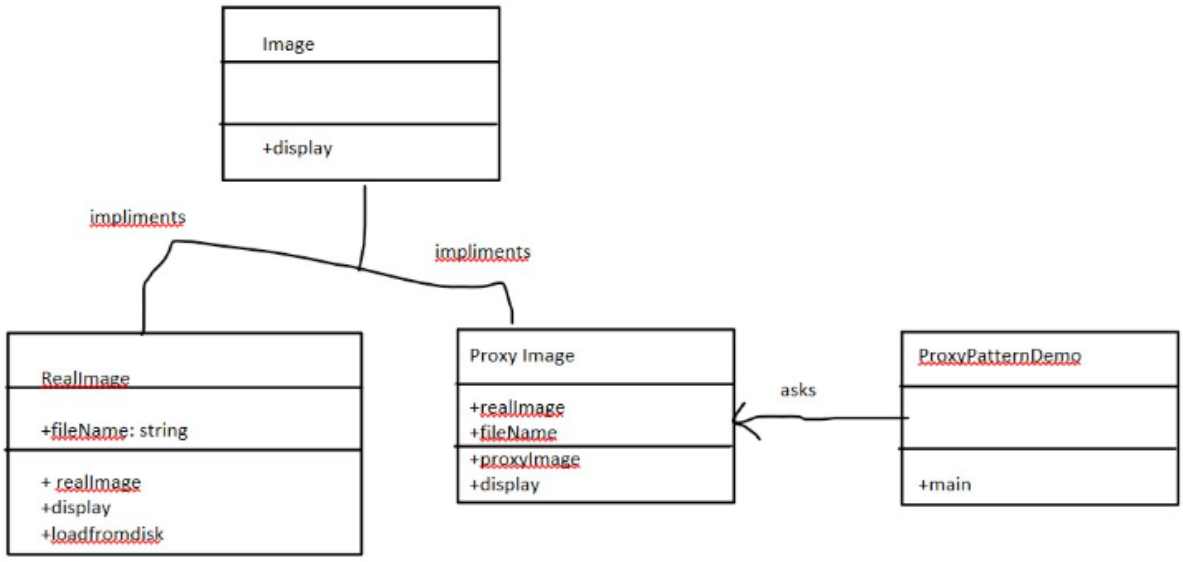
Bridge Design Pattern

a. It is used to split large class into two separate inheritance hierarchies
b. One is for implementations and one for abstraction
c. This pattern is known as bridge design pattern



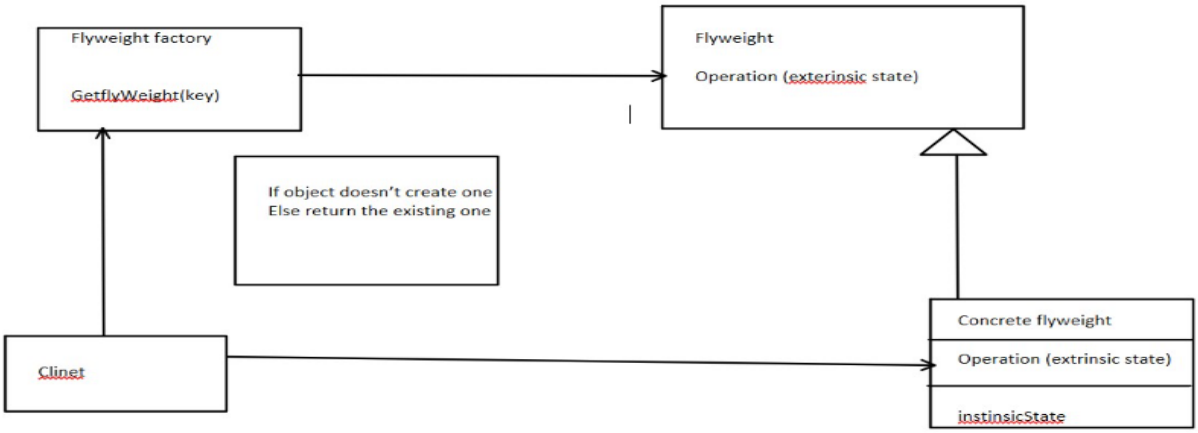
Proxy Design Pattern

a. In proxy pattern a class functionality of another class.
b. Here we create object having original object to interface its outer world functionality
c. According to GOF (Gang oF Four) it provide interface to access information of original object
d. Types
   i. Virtual proxy (Avoids duplication of memory)
   ii. Remote proxy (Provides local representation of object)
   iii. Protective proxy (Checks whether proper content is served)
   iv. Smart proxy (Provides additional layer of security)



Flyweight design pattern

a. It is used to reduce the number of objects created and improves performance
b. It reuses similar existing kind of objects
c. Provides good solution to reducing memory load by sharing objects
d. Has two states
   i. Intrinsic states (It is constant for over all state)
   ii. Extrinsic states (Changing part)
e. Example Car object
   i. Intrinsic (Color,Image)
   ii. Extrinsic (x,y,z coordinates)

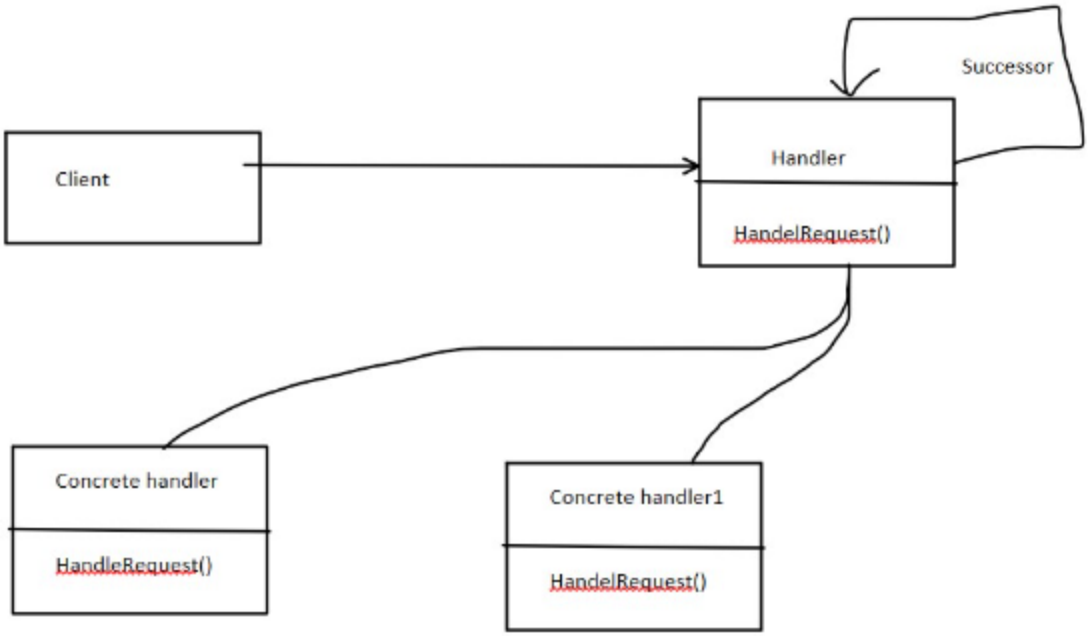


The Behavioral design pattern

Chain of responsibility pattern

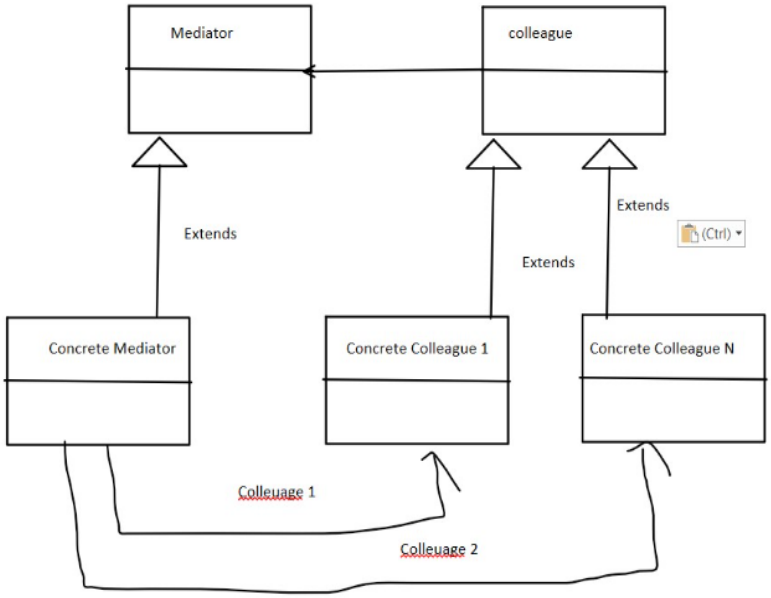
a. There is a handler that handles the responsibility of the objects which are in process

b. Request can be handled with one handler or many handlers
c. Example
   I. One full note of 200 rs can paid the sum of 200 rs (One handler)
   II. Multiple notes of 10,20,50 can pay the sum of 200 rs (Multiple handlers)
d. This design pattern is to be used when the sender doesn't have enough idea about it.



Mediator

a. It says that "to define an object that encapsulates how a set of object interacts"
b. Used to reduce communication complexity between multiple objects
c. It is more generic

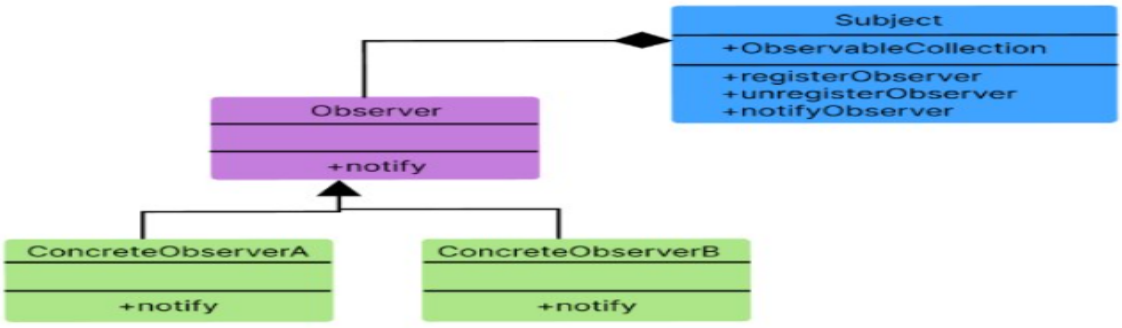


Observer

a. Observer is a behavioral design pattern. It specifies communication between objects: observables and observers. An observable is an object which notifies observers about the changes in its state

b. We can implement the code using two classes in java. One is Observer/Observable, Other is PCL (PropertyChangeListener)

c.Observer Pattern looks like this

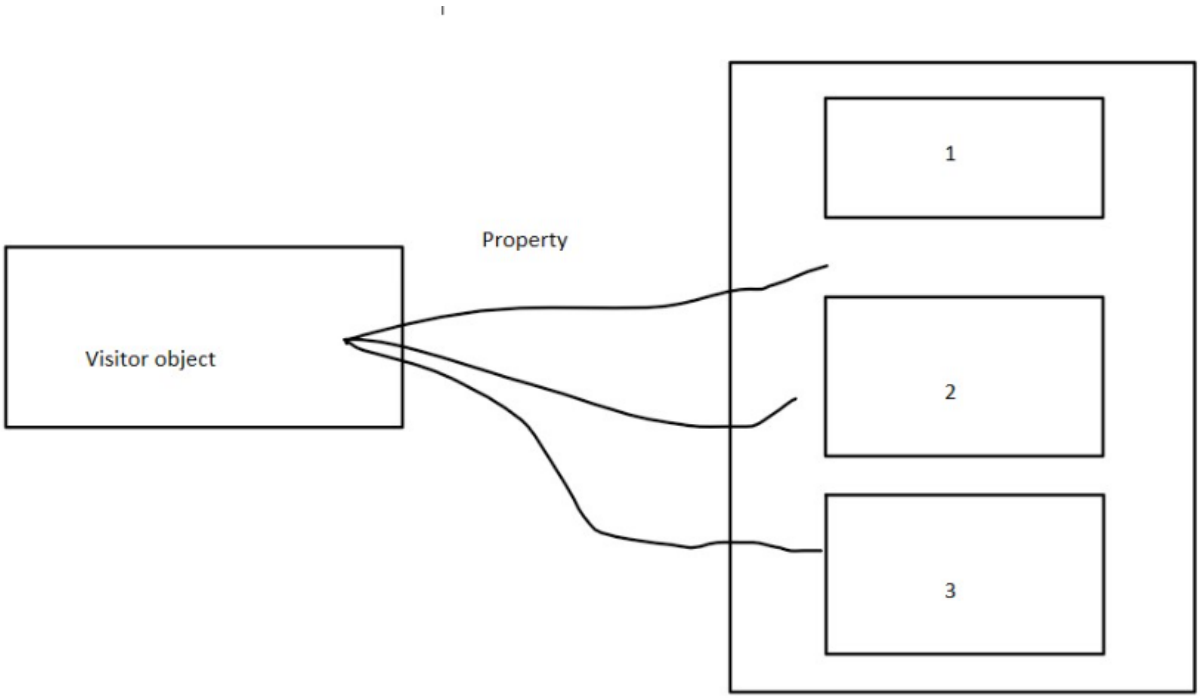


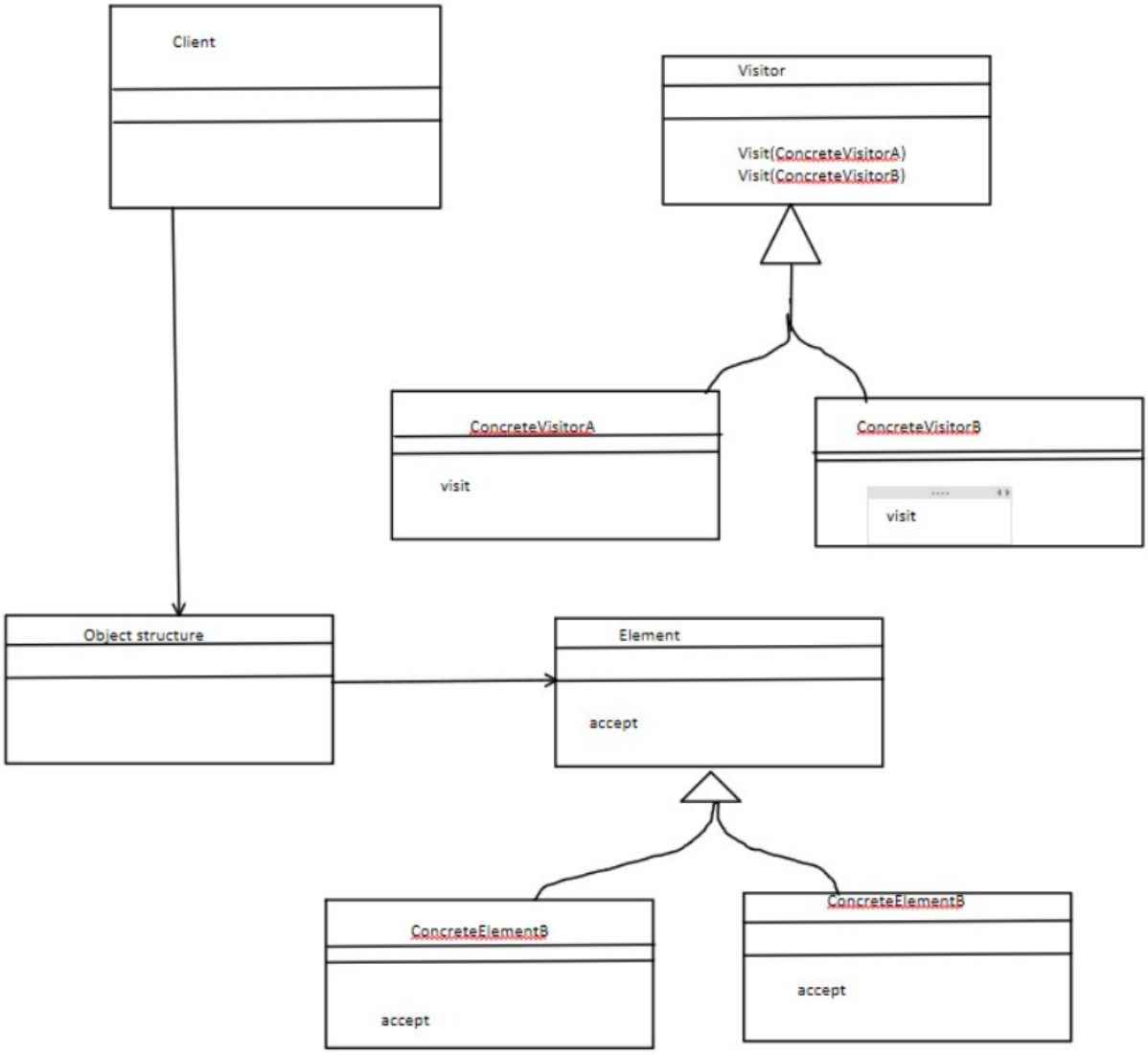
Strategy Design Pattern

a. Class Behavior or pattern can be changed at run time
b. Example while in sorting we can change the sorting algorithm according to the given array

Visitor Design Pattern

a. Represent an operation to be performed on the elements of object
b. Visitor let's you define a new operation without changing the class elements on which it operates
c. We use a visitor which changes the executing algorithm of an element object





State Design Pattern

a. It is a behavioral design pattern that lets an object alter its behaviour when its internal state changes. It appears as if the object changed its class

b. Here class Behavior changes based on its state. This type of design pattern comes under behavior pattern

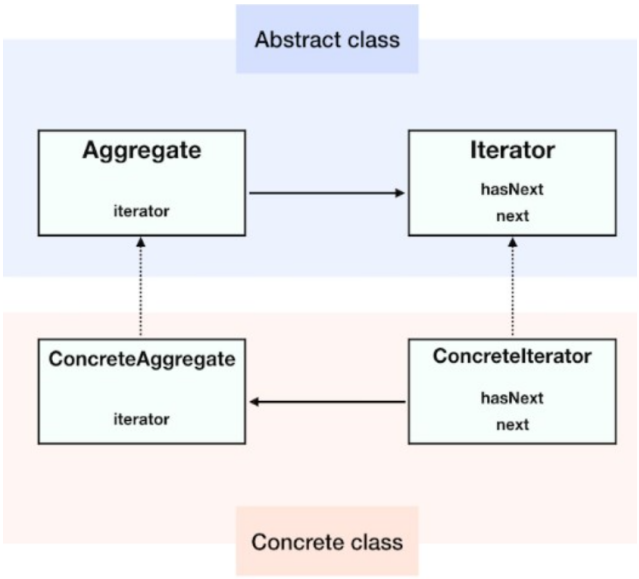
c. State pattern is know as object pattern

Iterator design pattern

a. It is used to access the elements of an aggregate object sequentially without exposing its underlying implementation

b. In OOP this pattern is used to traverse a container and access the containers elements

c. Used when there are multiple traversal of objects



Memento Design Pattern

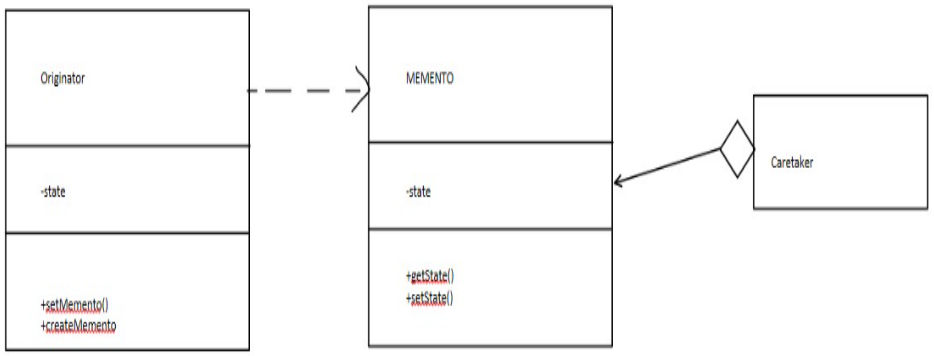
a. It is used to restore the state of an object to its previous state. Memento pattern falls under behavioral pattern

b. It uses three actors

a. State of an object that has to be restored

b. Originator creates and stores in memento objects and caretaker restores object state

c. It is used in undo/redo operations



CASE STUDY