Chapter-1

Review of object oriented Analysis and Design

"owning a hammer doesnot make one an Architect' Knowing an object-oriented language (such as Java) is a necessary but insufficient first step to create object systems. Knowing how to 'think in objects' is also a critical

The most important skill in object-oriented Analysis and Design is assigning responsibilities to objects. That determines how objects interact and what classes should perform what operation.

Analysis:

Analysis emphasizes an investigation of the problem and requirements, rather than a solution. what the problem is all about and what the system must do. "Do the right thing"

Design

Design emphasizes a conceptual solution that fulfills the requirements, rather than its implementation "Do the thing right"

OOA:

OOA is a method of analysis that examines requirements from the perspective of classes and objects found in the vocabulary of problem domain.

OOD :

It is a method of design encompassing the process of object-oriented decomposition and notation for depicting both logical and physicall as well as static and dynamic models of the system under design - Grady Booch.

Software :

It is a generic term for organized collections of collections of computer data and instructions.

↳ System Software
↳ Application Software

- System software

is responsible for controlling, itegrating, and managing the individual hardware components of a computer system.

Eg: Operating system

- Application Software

is used to accomplish specific tasks other than just running the computer system.

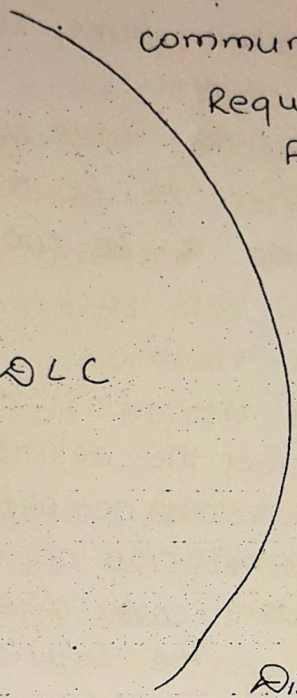
Eg: Microsoft office.

* Software Engineering

- is an engineering branch associated with development of software product using well-defined scientific principle, methods and procedures.
- is the process of solving customer's problem by the systematic development and evolution of large, high quality software systems within cost, time and other constraints.

Software Development Life cycle (SDLC)

- is a well defined, structured sequence of steps in software engineering to develop the intended software product.



SDLC

communication
Requirement Gathering
Feasibility study
system Analysis

Software Design
Coding
Testing
Integration
Implementation

Operation and Maintainenance
Disposition

Software Development Paradigm

- It helps developer to select a strategy to develop the software
- A software dev. paradigm has its own set of tools, method and procedures, which are expressed clearly and defines SDLC.

They are:

1. waterfall
2. Iterative
3. Spiral
4. V model
5. Big Bang

- Designing object oriented software is ...
and designing reusable OO software is even harder
- Experienced designers reuse solutions that ...
worked in the past.
- Well structured OO system have recurring
patterns of classes and objects.
- Knowledge of the patterns that have worked in
the past allows a designer to be more produc...
and the resulting designs to be more flexible
and reusable.

- Design Patterns describes the relationship
interactions of different class or objects or ...
- They do not give the final class or types that
will be used in software code directly ...
give an abstract view of the solution.
- Patterns show us how to build systems with
good object oriented design qualities by reusing
succesfull design and architectures.
- Expressing proven techniques speed up the
development process and make the design ...
more accessible to developers of new system

"The recurring aspects of designs are called
Design Patterns"

A pattern is the outline of a reusable solution
to a general solution to a general problem
encountered in a particular context.

- In software engineering, a design pattern is a general reusable solution to a commonly occuring problem in software design.
- A design Pattern is not a finished design that can be transformed directly into code.
- It is a description or template for 'how to solve a problem that can be used in many different situations'.
- Object oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved.

A 'GOOD PATTERN' should be:
- be as general as possible
- contain a solution that has been proven to efficitively solve the problem in the indicated context

Characters of design Patterns
- They give the developer a selection of tried and tested solutions to work with.
- They are language neutral. They can be any language that supports object orientation.
- They have a proven track record as they are already widely used and thus reduce technical risk to the project
- They are highly flexible and can be used in practically any type of application or domain

Studying patterns is an effective way to learn from experience of others.

The originator of Patterns:

- Each pattern describes a problem which occurs over and over again in our env, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over without ever doing it the same way twice - Christopher Alexander 1977

Pattern description

Each pattern is a three-part rule, which expresses a relation between a certain context, a problem and a solution.

1. Context: The general situation in which the pattern applies

2. Problem: A short sentence raising the main difficulty

3. Solution: The recommendation way to solve the problem in the context
   - to balance the forces

4. Forces: The issues or concerns to consider while solving the problem

5. References: Who developed or inspired the patterns.

6. Antipatterns: Solutions that are harmful or do not work with context [optional]

7. Related patterns: Patterns that are similar to this pattern.

They help us solve recurring design problems.
Note that: design patterns don't solve the problem themselves, they help us solve the problem.

- Design patters are often called time tested solutions for object oriented programing problems.

* The importance of design patterns are:
- Identifying the right pattern for the problem will help us to avoid costly changes, un-maintainable complex and in-efficient code as the system scales up.

→ Communication, Learning and Enhanced Insight
- Over the last decade, design patterns have become part of every developer's vocabulary.
= This really helps in communication. one can easily tell another developer on the team, "I've used Command patterns here" and another developer understands not just the design, but can also figure out the rationale behind it.
- This helps in providing developers with better insight about parts of the application or 3rd party framework they use.

→ Decomposing system into objects
- The hard part of OO is to find objects and decomposing a system. one has to think about encapsulation, granularity, dependencies, flexibility, performance, evolution, reusability and so on.
- Design pattern really helps identify less obvious abstraction

→ Determining object Granularity.
→ Specifying object Interfaces
→ Designing for change
→ Relating run-time and compile time structure
→ Ensuring right reuse mechanism
→ Specifying object Implementation.

Classification of Design Patterns:

Design patterns were originally classified into 3 types:
- Creational Patterns.
- Structural Patterns.
- Behavioural Patterns.

extra: - Concurrency Patterns [since later been added]

Creational Patterns:

- creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation.
- The basic form of object creation could result in design problems or added complexity to the design.

- Creational design patterns solve this problem by somehow controlling this object creation.

types of Creational Design patterns

1. Factory Method:
   Creates an instance of several derived class.
2. Singleton:
   A class of which only a single instance can exist.
3. Abstract Factory:
   Creates an instance of several families of class.
4. Prototype:
   A fully intialized instance to be copied or cloned.
5. Builder:
   separates object construction from its representation.

B. Structural Patterns:

- Structural design patterns are design patterns that ease the design by identifying a simple way to realise realationships between entities.
- These describe how objects and classes combine themselves to form a large structure.

types of structural patterns:

1. Facade:
   A single class that represents an entire subsystem
2. Decorator:
   Add responsibilities to objects dynamically.
3. Composite:
   A tree structure of simple and composite objects.
4. Adapter:
   Match interfaces of different classes.

5. Flyweight:
   A fine-grained instance used for efficient sharing.
6. Proxy:
   An object representing another object
7. Bridge:
   Separates an object's interface from its implementation.

C. Behavioural Patterns:

- Behavioural Design Patterns is a design Patterns that identify common communication patterns between objects and ~~rati~~ realize these patterns
- These patterns increase flexibility in carrying out this communication

~~types of~~ types of Behavioural design patterns

1. Interpreter:
   It handles the expressions in grammars.
2. Iterator
   visits member of a collection in a sequential fashion
3. Mediator
   captures behaviour among peer objects without building a dependency between the objects
4. Observer
   A way of notifying change to a number of class by a single object.
5. State
   Alter an object's behaviour when its state changes
6. Chain of Responsibility
   A way of passing a request between a chain of objects

7. Command
Captures function flexibility
8. Template:
captures basic algorithm, allowing variability
9. Momento:
captures and restore the object's internal state
10. Strategy:
Encapsulate an algorithm inside a class.
11. Visitor
Defines a new operation to a class without change.

* Gang of four (GOF)
In 1994, Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides published a book titled Design Patterns - Elements of Reusable object-oriented software which initiated the concept of Design Pattern in software Development.

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

* As per 'this' book, there are 23 design patterns which be classified in three categories: Creational, Structural and Behavioural patterns.

Types of Pattern
-Architectural pattern
-Design Patterns
-IDIOMS

Creational Patterns:

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

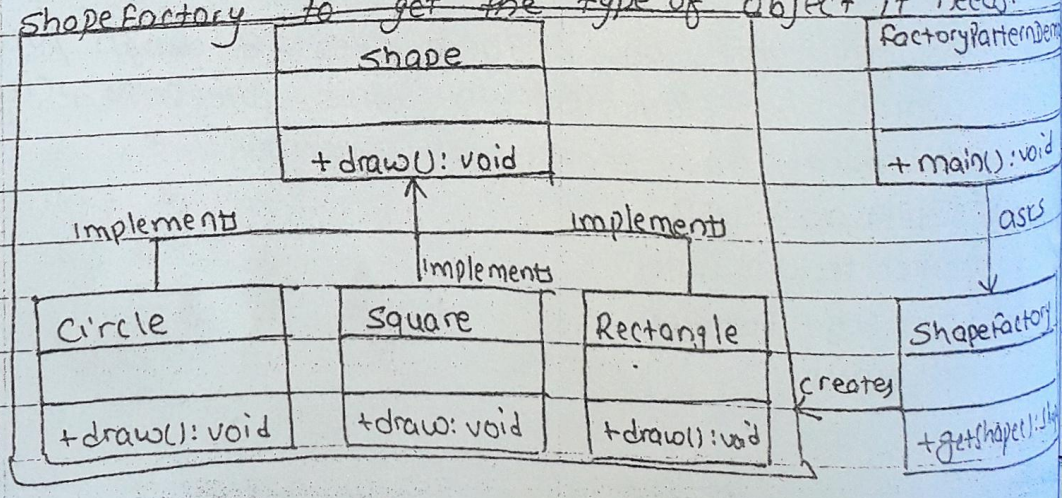
1. Factory Design Pattern

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

a. Implementation:

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

FactoryPatternDemo, our main class (demo class), will use ShapeFactory to get a shape object. It will pass information (circle, Rectangle, Square) to ShapeFactory to get the type of object it needs.



Step1: Create an interface

Shape.java

```
public interface Shape {
    void draw();
}
```

Step 2: Create concrete classes implementing same interface

Rectangle.java

```
public class Rectangle implements Shape {
    @Override
    public void draw() {
        sop("Inside Rectangle::draw() method.");
    }
}
```

Square.java

```
public class Square implements Shape {
    @Override
    public void draw() {
        sop("Inside Square::draw() method.");
    }
}
```

Circle.java

```
public class Circle implements Shape {
    @Override
    public void draw() {
        sop("Inside circle::draw() method.");
    }
}
```

Step3 : Create a factory to generate object of concrete class based on given information.

ShapeFactory.java

```
public class ShapeFactory {

   // use getShape method to get object of type shape
   public Shape getShape(String shapeType) {
      if(shapeType == null)
         return null;
      else if(shapeType.equalsIgnoreCase("CIRCLE")) {
         return new Circle();
      else if(shapeType.equalsIgnoreCase("RECTANGLE"))
         return new Rectangle();
      else if(shapeType.equalsIgnoreCase("Square"))
         return new Square();
      return null;
   }
}
```

Step4 : Use the factory to get objects of concrete class by passing an information such as type

FactoryPatternDemo.java

```
public class FactoryPatternDemo {

   public static void main(String[] args) {
      ShapeFactory sf = new ShapeFactory();

      // get an object of circle and call its draw meth
      Shape shape1 = sf.getShape("CIRCLE");
      shape1.draw();
```

```
//get an object of Rectangle and call its draw method
Shape shape2 = sf.getShape("RECTANGLE");
shape2.draw();

//get an object of Square and call its draw method
Shape shape3 = sf.getShape("SQUARE");
shape3.draw();
}
}
```

Step 5: Verify the output

```
Inside Circle :: draw() method.
Inside Rectangle :: draw() method.
Inside Square :: draw() method.
```

Abstract Factory Patterns

Abstract Factory Patterns work around a super factory which creates other factories. Abstract factory is also called factory of factories.

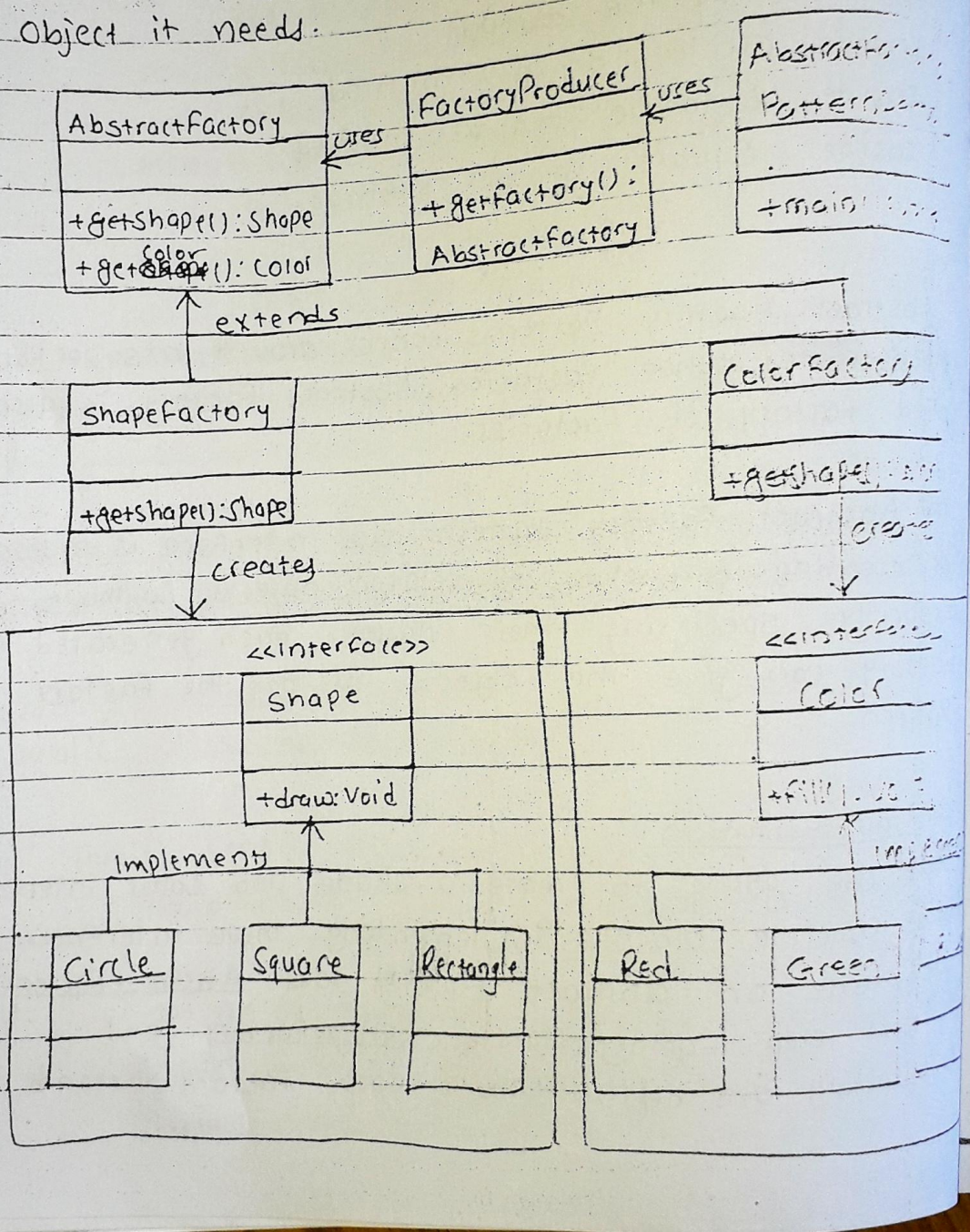
In Abstract factory pattern, an interface is responsible for creating a factory of related objects without explicitly specifying their classes. Each generated factory can give the objects as per the factory Pattern.

a. Implementation

We are going to create a shape and color interfaces and concrete classes implementing these interfaces. We create an abstract factory class AbstractFactory. As next step, Factory classes ShapeFactory and ColorFactory are defined where each factory extends

AbstractFactory: A factory creator
FactoryProducer is created.

AbstractFactoryPatternDemo, our demo class, uses
FactoryProducer to get an AbstractFactory object.
It will pass information (CIRCLE / RECTANGLE / SQUARE
for shape to AbstractFactory to get the type
object it needs. It also passes information (RED /
BLUE) for color to AbstractFactory to get the
object it needs.



1) 1. creates an interface for shapes

Shape.java

```
public interface Shape{
    public void draw();
}
```

2. create: concrete classes implementing the same interface

Rectangle.java

```
public class Rectangle implements Shape{
    @ public void draw{
        Sop(" Inside Rectangle :: draw(). ");
    }
}
```

Square.java

```
public class Square implements Shape{
    @Override
        public void draw(){ sop(" Inside Square :: draw(). "
    }
```

Circle.java

```
Public class Circle implements Shape{
    @override
        public void draw(){ sop(" Inside Circle: draw(). ");;
    }
```

3. Creates an Interface for Colors

Color.java

```
public interface color{
    public void fill();
}
```

4. Create Concrete classes implementing the same interface

Red.java

```
public class Red implements Color{
    @overrid public void fill(){ sop(" Red :: fill(). "); }
}
```

Green.java

```
public class Green implements Color{
  @Override public void fill() { sop(" Green::fill().");}
}
```

Blue.java

```
public class Blue implements Blue {
  @Override public void fill() { sop(" Blue::fill().");}
}
```

5. Create an Abstract class to get factories for color and shape objects.

AbstractFactory.java

```
Public abstract class AbstractFactory {
    abstract Shape getShape(String shapeType);
    abstract Color getColor( String color);
}
```

6. Create factory classes extending AbstractFactory to generate object of concrete class based on given information.

ShapeFactory.java

```
public class ShapeFactory extends AbstractFactory {
    @Override
    public Shape getShape( String shapeType) {
      if( shapeType == null ) return null;
      if( st.equals("RECTANGLE")) return new Rectangle();
      else if(st.equals("CIRCLE")) return new Circle();
      else if(st.equals("SQUARE")) return new Square();
      return null;
    }
}
```

```
@Override
public Color getColor(String color){
    return null;
}
}
```

ColorFactory.java

```
public class ColorFactory extends AbstractFactory {
    @Override
    public Shape getShape(String getShapeType) {
        return null;
    }

    @Override
    Color getColor(String color){
        if(color==null) return null
        if(color.equals("RED")) return new Red();
        else if(color.equals("GREEN")) return new Green();
        else if(color.equals("BLUE")) return new Blue();
        return null;
    }
}
```

7. Create a factory generator/producer class to get factori by passing an information such as Shape or Color

FactoryProducer.java

```
Public class FactoryProducer {
    public static AbstractFactory getFactory(String Choice){
        if(choice.equals("SHAPE")) return new ShapeFactory();
        else if(choice.equals("COLOR")) return new ColorFactory();
        return null;
```

8. Use the FactoryProducer to get AbstractFactory in order to get factories of concrete classes by passing an information such as type.

```
AbstractFactoryPatternDemo.java
public class AbstractFactoryPatternDemo {
   public static void main(String[] args) {

      //get shape factory
      AbstractFactory shapeFactory = FactoryProducer.
            getFactory("SHAPE");
      //get an object of shape Circle, rect, square & draw
      Shape s1 = shapeFactory.getShape("CIRCLE");
      s1.draw();

      Shape s2 = shapeFactory.getShape("RECTANGLE");
      s2.draw();
      Shape s3 = shapeFactory.getShape("SQUARE");
      s3.draw();

      //get Color factory
      AbstractFactory colorFactory = FactoryProducer.
            getFactory("COLOR");
      Color c1 = colorFactory.getColor("RED");
      c1.fill();
      Color c2 = colorFactory.getColor("GREEN");
      c2.fill();
      Color c3 = colorFactory.getColor("BLUE");
      c3.fill();
   }
}
```

3

g. verify the output

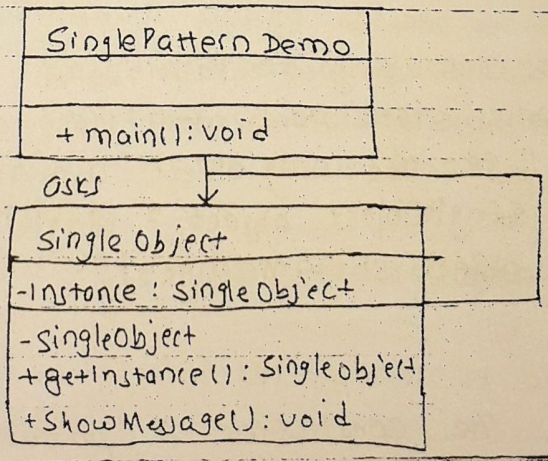
Inside circle:: draw().
Inside Rectangle:: draw().
Inside Square:: draw()
Red:: fill();
GREEN:: fill()
BLUE:: fill()

3. Singleton Pattern

Singleton Pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without instantiating the object of the class.

a. Implementation

- we are going to create a singleObject class which has its constructor as private and a static instance itself.
- SingleObject class provides a static method to get its static instance to outside world. ~~Singleton Pattern~~ SingletonPatternDemo, our demo class, will use SingleObject class to ge a singleObject's object.



Step 1: create a singleton class

SingleObject.java

```
public class SingleObject {
    //create an object of SingleObject
    private static SingleObject instance = new
        SingleObject();
    //make the constructor private so that this
    class cannot be instantiated
    private SingleObject() {

    }

    //Get the only object available
    public static SingleObject getInstance() {
        return instance;
    }

    public void showMessage() {
        System.out.println("Hello from SingleObject");
    }
}
```

```
Public class SingletonPatternDemo {
    public static void main(String args[]) {
        // get the only object available
        SingleObject object = SingleObject.getInstance();
        object.showMessage();
    }
}
```

Note: The constructor is private so we can not create object by new SingleObject();

5. Builder Pattern

11. Design pattern

A design pattern is general repeatable solution to commonly occuring problem in software design. A design pattern is not a finished design that can be a transformed directly into code. It is a description or template for how to solve a problem that can be used in many different solutions.

Uses of design Patterns :

- It speed up the development process by providing tested, proven development paradigm.
- It improves code readability for coders
- It provides general solutions, documented in a format that doesnot require specific tied of particular solution.

1. Creational patterns

In software engineering, creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation. The basic form of object creation could result in design issues or added complexity to the design. Creational design patterns solves this problem by somehow controlling this object creation.

- Abstract factory : creates an instance of several families of classes
- Builder : separates object construction from its representation.
- Factory method : creates an instance of several derived classes

Christopher Alexander 1970 - de...
Gang of Four GoF 1990s

- prototype: A fully intialized instance to be copied or cloned.
- Singleton: A class of which only a single instance can exist.

Context: कस्तो situation मा Patern use गर्ने ?
Problem: के के गर्दा design issues आएको ?
Forces: कुन चिजहरूलाई ध्यान गर्नुपर्छ, कस्तो समस्या समाधान गर्नाको निम्ति।
Solutions: कुन तरीकाले माथि आएको समस्या समाधान हुन्छ

Factory Design Patterns (creating instance of derived)

- factory is just something that produces the item such as cars, TVs.

- Factory method is used to construct objects such that they can be decoupled from implementing system.
- Define an Interface for creating an object, but let the subclasses decide which class to instantiate. The factory method lets a class defer instantiation to subclasses.
- Description

a. Context: A reusable framework that needs to create object as a part of it works: However, the class of created objects will depend on application
b. Problem: How do you enable a programmer to add new application-specific class into system built on such framework
c. Forces: we want to have the framework class create of application specific classes that the framework does not yet know about.
d. solution:

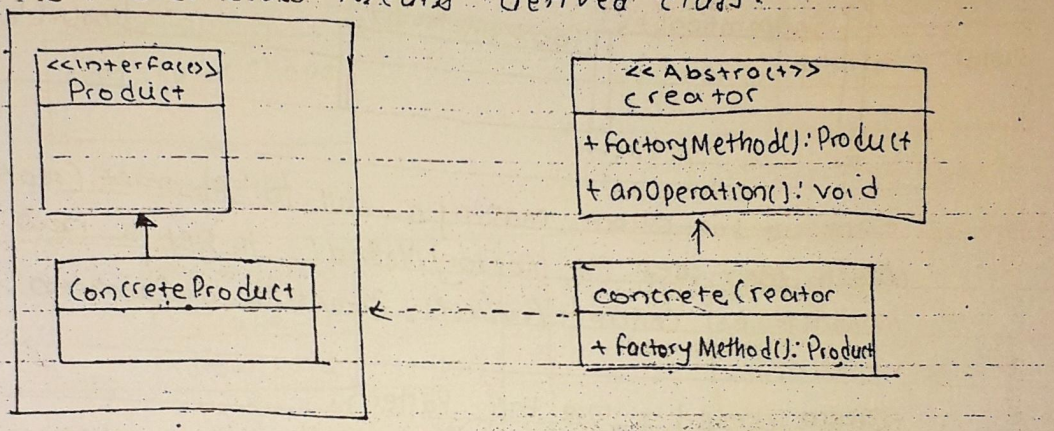
5

a. framework delegates the creation of instances of Application Specific classes to a specialized class, from the factory

b. factory is generic interfaces that defined framework.

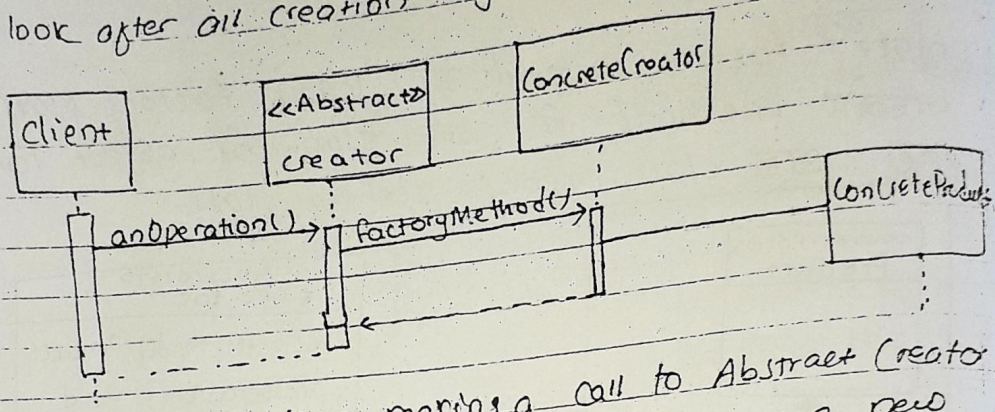
c. factory declares a method whose purpose is to create some subclass of ~~App~~ specific class of ~~class~~ genericClass.

Note: factory method में एक factory class बनाते हैं methods के अंदर class को object देते हैं अंदर class means derived class.




```
public interface product { }
public class concreteproduct implement Product { }
public class concreteCreator extends creator
    protected Product factoryMethod {
        Product pr = new ConcreteProduct();
        return pr;
    }
}
public class Creator {
    main() {
        Product P1 = new ConcreteCreator().factory
                                          Method();
    }
}
```

Now lets take look at the diagram definition of the factory method. The 'creator' hides the creation and instatiation of product from clients. This is a benefit to the client as they are now insulated from any future change - the creator will look after all creation logic, allowing to decoupling.



Here we see client making a call to Abstract Creator, which then uses the factoryMethod() to get a new instance of ConcreteProduct, completes anOperation().

Where would I use this Pattern.

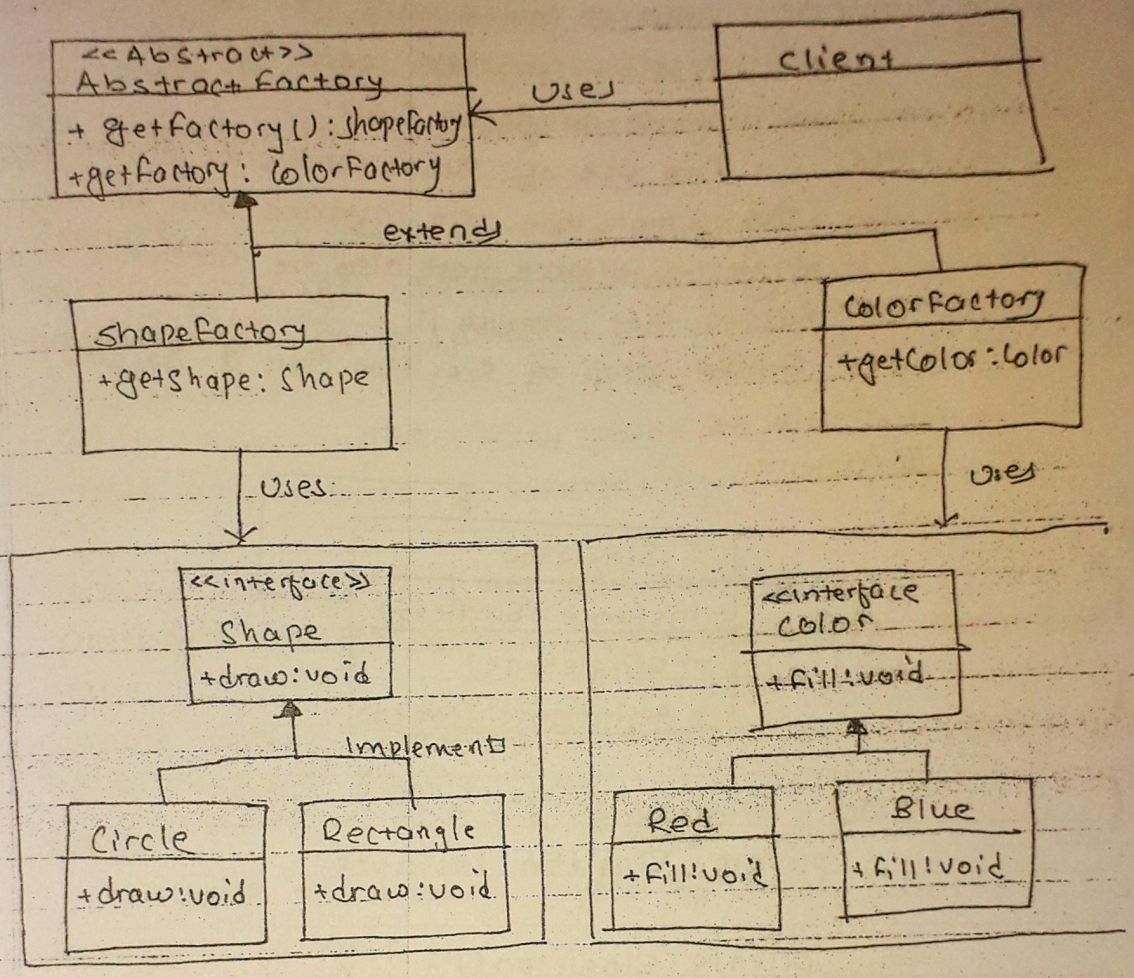
FactoryMethod pattern allows for the case where a client doesn't know the what concrete classes, it will be required to create at runtime, but just want to get a class that do the job. But lets the subclasses decide which implementation of concrete class to use.

→ Abstract Factory method are implemented with FactoryMethod can be implemented using Prototype.

→ Factory method → inheritance, Prototype : creation through delegation.

Abstract Factory Method : (creating Families)

provides an interface for creating families of related or dependent objects without specifying their concrete classes.



→ The main benefits of this pattern is that the client is totally decoupled from concrete classes. Although new product family could be easily added into system

→ provide an interface for creating families of related or dependent object without specifying their concrete classes

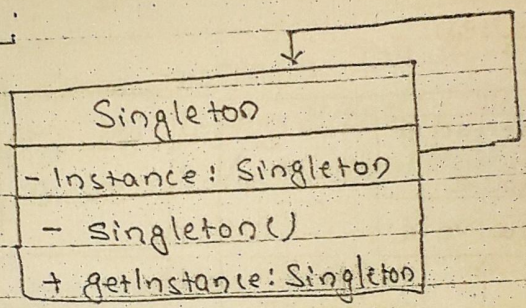
The Singleton Pattern:

Context: It is very common to find classes for which only one instance (exactly... creating one) should exist

Problem: How do you ensure that it is never possible to create more than one instance of a singleton class

Forces: - The use of public constructor cannot guarantee that no more than one instance will be created
- Singleton instance must also be accessible to all classes that require it
- Global access of Singleton class is necessary

Structure:




```
class Singleton {
    private static Singleton instance;

    private Singleton() {---}

    public static Singleton getInstance() {
        if(instance == null)
            instance = new Singleton();
        return instance;
    }
    public void doSomething() { }
}
```

The Singleton pattern defines a getInstance operation which exposes the unique instance accesible by clients: getInstance() ensures that only one way of instantiating.

1. Define a private static member variable of myClass
2. make the default constructor private
3. Define a public static method to access the member

Example: In a system, there should be only one window manager to close, minimize or maximize and accesible from all the system.

- It provides a global access point to that instance
- only one instance is created.

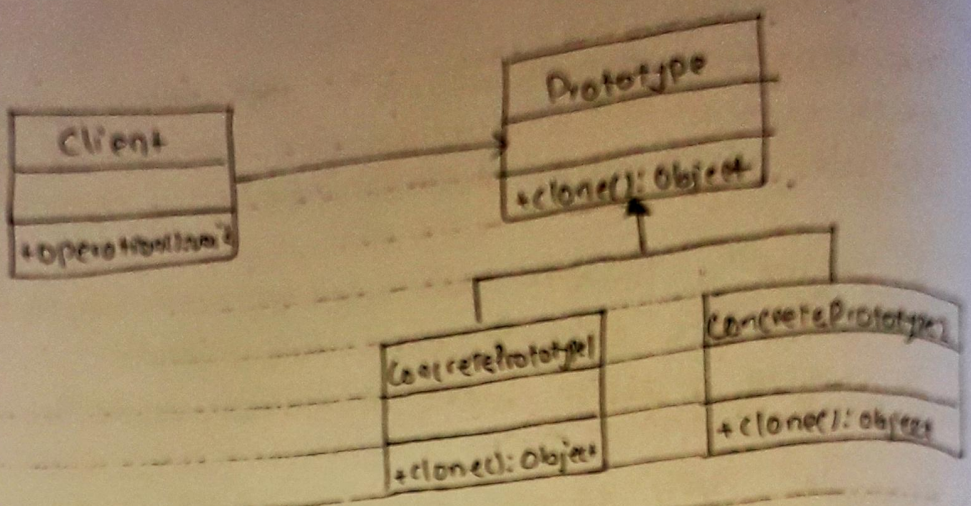
Prototype pattern (mix and match)

- prototype pattern involves copying something that already exist
- create objects based on a template of an existing code through cloning

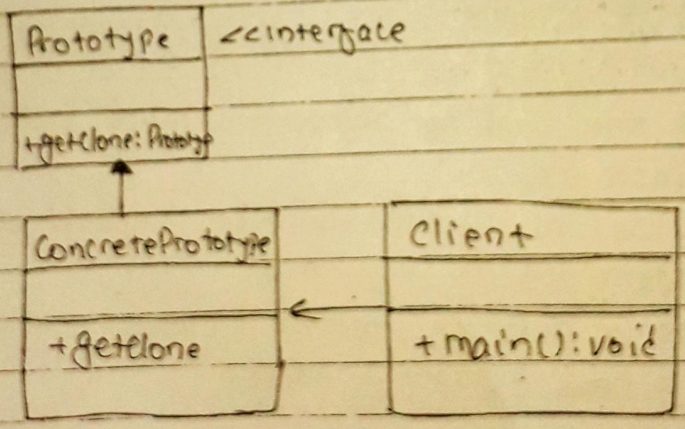
Intent:

- Specifying the kind of objects to create using a prototypical instance
- create a set of almost identical objects whose type is determined at runtime
- Assume that a prototype instance is known: clone it whenever a new instance is needed

The process of cloning starts with an initialized and instatiated class. The client ask for a new object of that type to prototype. class. A ConcretePrototype, depending of the type of object is needed, will handle the cloning through clone(), making a new instance of itself.



- Prototype pattern says that "cloning of an existing object instead of creating new one and can also be customized as per the requirement
- If the cost of creating object is expensive and resource intensive.
- Advantages
  - it reduces the need of sub-classing
  - hides complexities of creating object
  - it lets you remove/add objects at runtime.



prototype.java

```
interface prototype {
    public Prototype getClone();
}
```

ConcretePrototype.java

```
public class ConcretePrototype implements Prototype {
  @Override
    public Prototype getClone() {
      return new ConcretePrototype();
    }
}
```

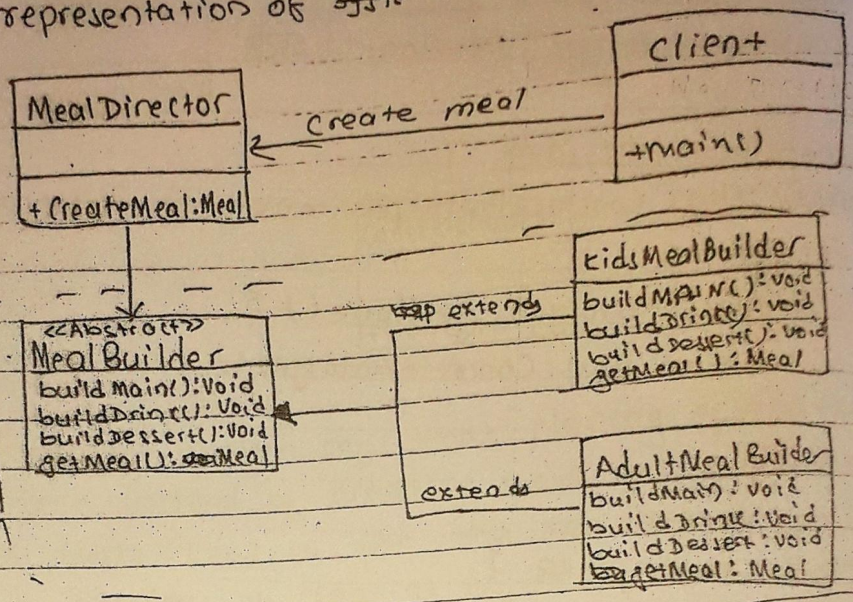
Client.java

```
class PrototypeDemo {
    main() {
      ConcretePrototype CP1 = new ConcretePrototype();
      CP1.doSomething();
      ConcretePrototype CP2 = CP1.clone();
      CP2.doSomething();
    }
}
```

Builder Design Pattern (Fast food order by kid)

- Builder pattern says that "construct a complex object from simple objects using step by step approach.
- Separate the construction of complex object from its representation so that same construction process can create different representations
- It provides clear separation bet^n construction and representation of an object.
- it supports to change the
- It is used when object creation algorithm should be decoupled from the system, and multiple

representation of ~~syste~~ creation algorithm required




```
public abstract MealBuilder {
    public abstract void buildMain();
public abstract void buildDrink();
public abstract void buildDessert();
public abstract Meal getMeal();
}
public class KidsMealBuilder extends MealBuilder{
}
public class AdultMealBuilder {
}
```

MealDirector

```
public class MealDirector {
    public Meal createMeal(MealBuilder builder){
        builder.buildDrink();
        builder.buildMain();
        builder.buildDessert();
        return builder.getMeal();
    }
}
```

GHJV: Gamma, Helm, Johnson, Vlissides (GoF) (1991-1994)
Design Patterns: Elements of Reusable Object-Oriented Software

Client.java

```
public class Client {
    main() {
        MealDirector director = new MealDirector();
        MealBuilder builder = null;
        if(isKid)
            builder = new KidsMealBuilder();
        else
            builder = new AdultMealBuilder();
        Meal meal = director.createMeal(builder);
    }
```

- In software Engineering, structural design patterns are DP that ease in design by ~~simply~~ identifying ~~the~~ a simple way to realize relationship between entities
- Structural dp are concerned with how classes to form larger structure
- It simplifies the structure by identifying the relationship
  a. Adapter pattern: Adapting an interface into another
  b. Bridge pattern: Separates an object's interface from it's implementation
  c. Composite: A tree structure of simple and composite objects.
  d. Decorator: Add responsibilities to objects dynamically.
  e. Facade: A single class that represents an entire subsystem
  f. Flyweight: Reusing an object by sharing it
  g. Proxy: An object representing another object

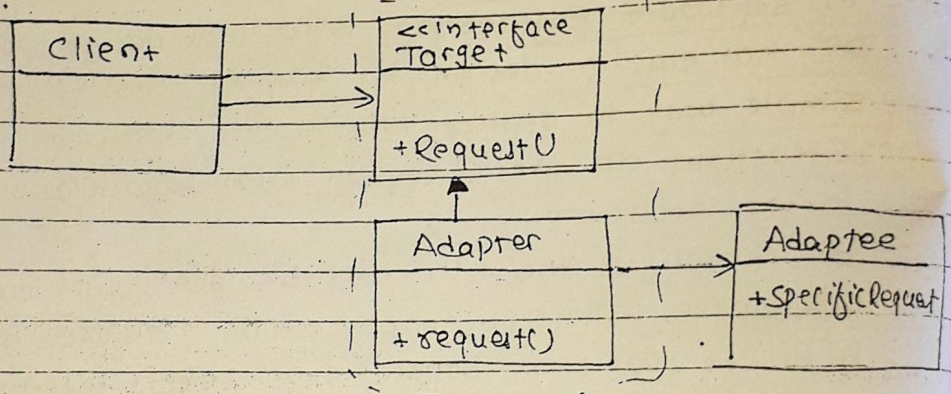
[wrapper]

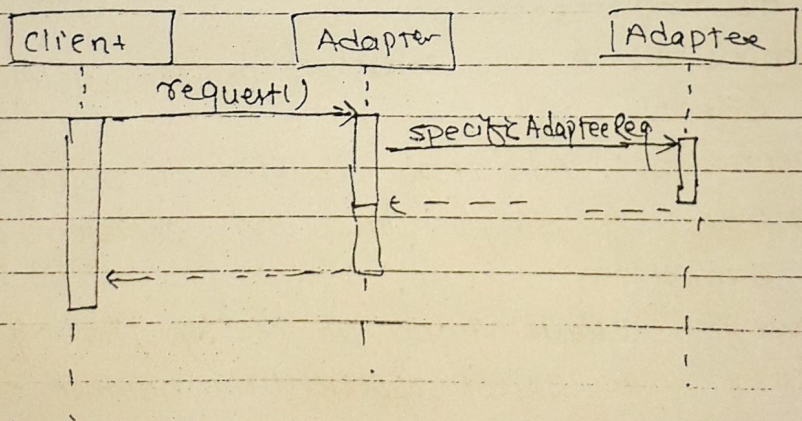
- Converts the interface of a class into another interface that clients wants.
- To provide the interface according to client requirement while using the services of a class with different interfaces.

Advantage

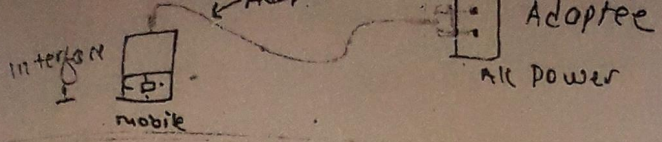
- It allows 2 or more incompatible objects to interact
- It allows reusability of existing functionality.
- It is used: when an objects need to utilize an existing class with an incompatible interface

following specification for Adapter pattern:





In background, The Adapter knows how to return any result, when Client request method to target interface.



-The main use of this pattern is when class that you need to use doesnot meet the requirements of an Interface

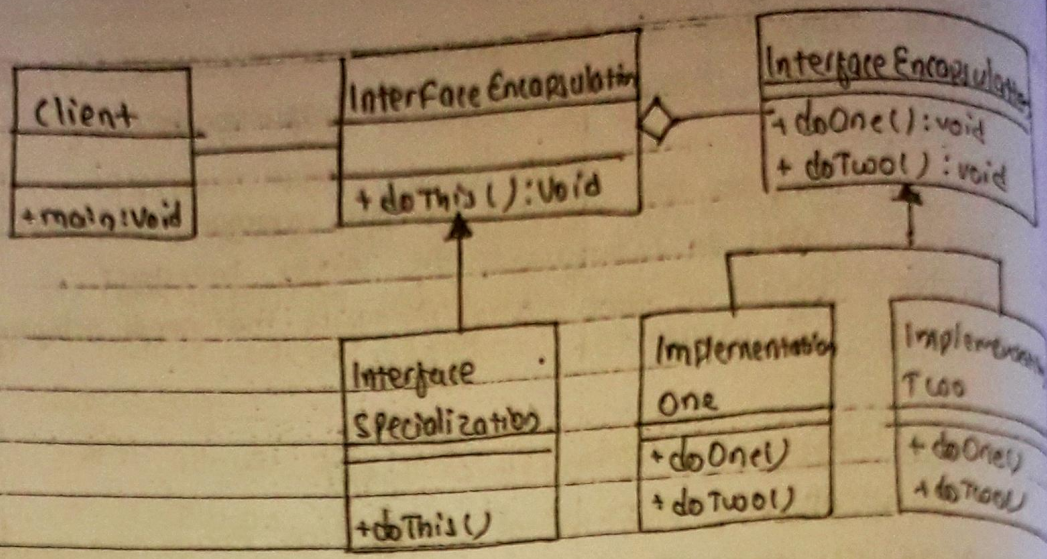
Target: defines the domain specific interface that client uses
Adapter: adapts an Interface to the Target Interface
Adaptee: defines an ~~ado~~ existing interface that needs adapting.
Ex:

2) Bridge Pattern [Handle or Body]

- Decouple the functional abstraction from the Implementation so that the two can vary independently.
- Advantage
- it enables the separation of Implementation from Interface.
- It improves the extensibility
- it allows the hiding complexities from client
- Understand: displaying different image formats on different OS: You might have different image abstraction for both jpeg and png images. The image structure is same across all OS, but it is viewed (implemented) is different on each OS.
- Decouple an abstraction from its implementation so that the two can vary independtly.

Use:

- desire of run time binding
- hiding details




```
in doThis { Implementation.doOne();
            Implementation.doTwo();
          }
```

Example : Bridge Pattern decouples an abstraction from its implementation, so that the two can vary independently. A household switch controlling lights, ceiling fans etc are examples of ~~real~~ Bridge. The purpose of switch is to turn on/off.

यो design pattern मा : यहुदा class लाई अको class ले implement गर्छ, अनि आफ्नो way मा modify गरेर काम लाउँछ, जस्तै Switch मा on/off हुन्छ, तर अब Switch-Implementation को switch लाई यसरी तानेर on/off गर्न बनायो।

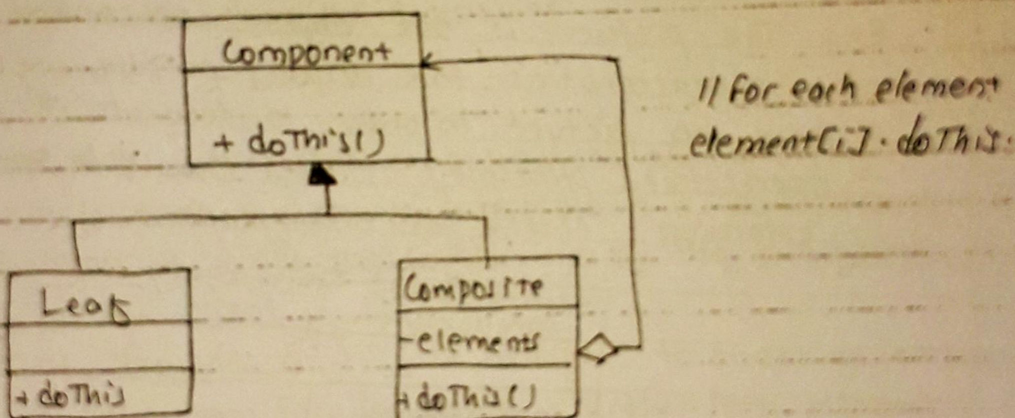
उक्त class के sub class consist जाते हैं ए subclass एक simple class हुन्छ or subclass को पनि अरू subclass हुन्छ

↑

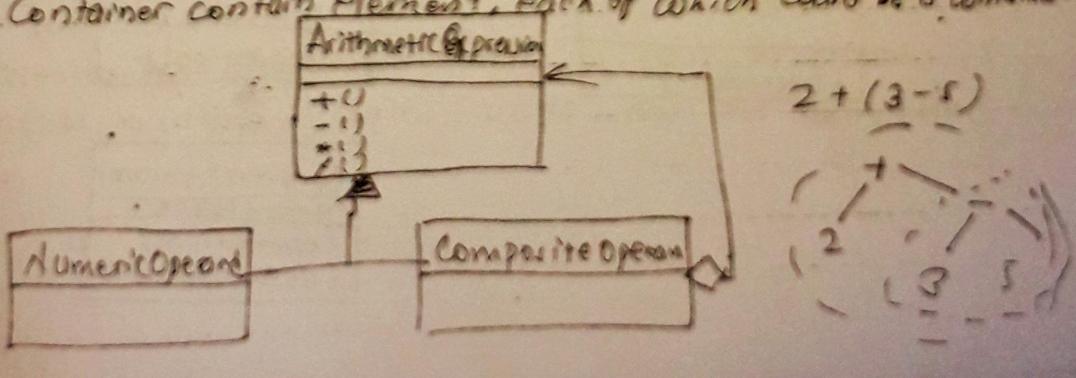
Composite Design Pattern

- Composite dp compose objects into tree structures to represent whole part hierarchies. Composite lets client treat individual objects and compositions of objects uniformly.
- Example: In hotel, In MenuBar has menu has many menu items, which themselves can have submenus. (Organization Chart, Employee hierarchy)

- Composite as a collection of objects, where any one of these objects could itself be a Composite or a simple objects.
- recursive Composition
- '1 to many' 'has-a' up the 'is-a' hierarchy.



- Directories that contain files, each of which could be directory.
- Container contain element, each of which could be a Container



Describe each :-

The intent of this pattern is to compose objects in tree structures to represent part-whole hierarchies

Decorator Pattern →

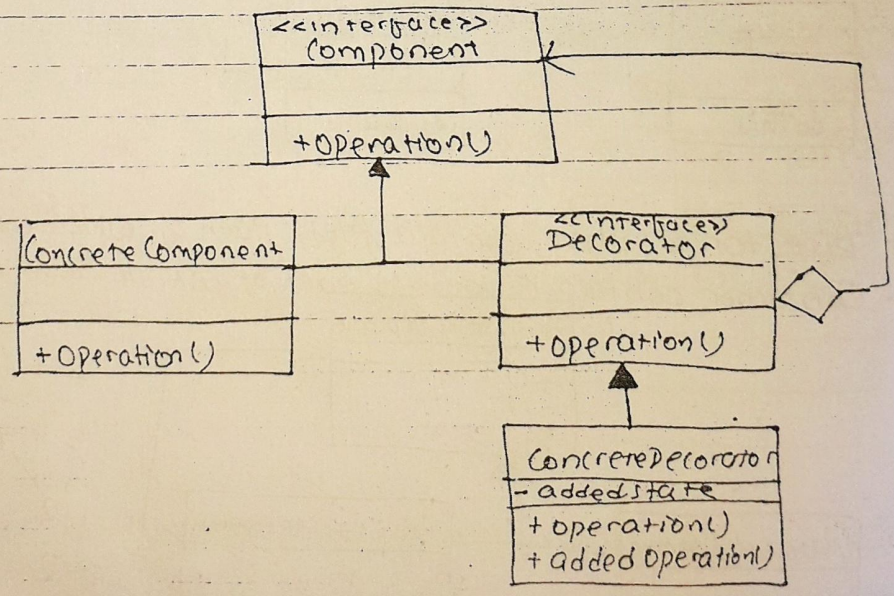
- It allows class Behaviour to be extended dynamically
- Attach additional responsibilities to an object dynamically
- Decorator provide a flexible alternative to subclassing for extending functionality.

Problem : You want to add behaviour to individual objects. Though Inheritance added applies to entire class and is static

The Concept of decorator is that it adds additional attributes / responsibilities to an object dynamically

Ex: The picture is our object which has its own characteristics. for display purpose, we add a frame to picture, In order to decorate it.

- open/closed principle : classes should be open for extension, closed for modification.



- The Component defines the interface for objects that can have responsibilities added dynamically.
- ConcreteComponent is simple implementation of this Interface
- Decorator has reference to Component, as Decorator is essentially wrapping the Component.
- ConcreteComponent just adds responsibilities to original Component.

§ Existing class लाई runtime मा यसको method हरू add गर्ने। जस्तो picture भन्ने class हुन्छ, runtime मा कि, ~~Class~~ picture show गर्यो भने, picture मा frame Add गर्यो (additional property / behaviour add)

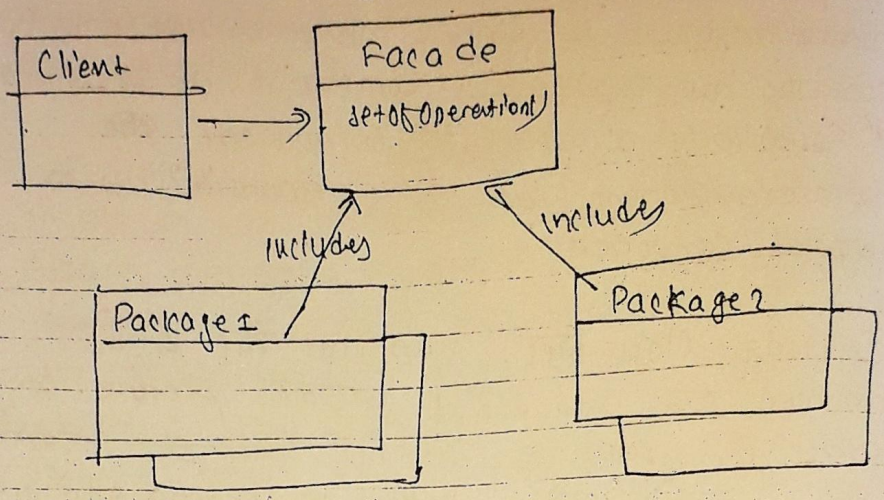
}

It can be done by subclassing

Facade Pattern

- It says "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
- GOF: Provides a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to ~~ease~~ use.
- Insulating client from subsystem
- Like in Factory Adapter pattern, the Facade can be used to hide the inner working of class. All client needs to interact with Facade, not the subsystem

- All Abstract Factory Method are Facade.

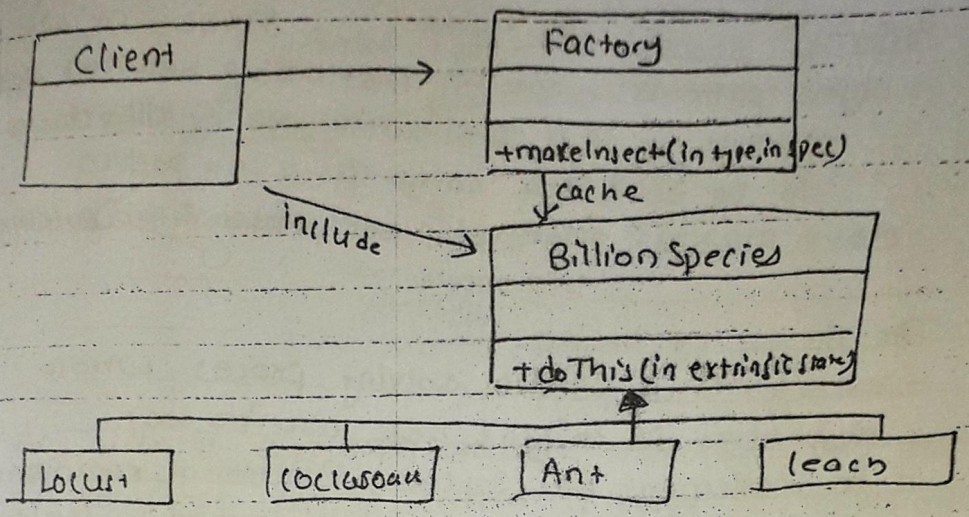


- It Simplify the interface
- Service oriented architecture make use of facade
- All other packages are hidden from client
- A segment of the client community needs a simplified interface to the overall functionality of complex subsystem.
- provide an interface to a package of classes
- modularizes designs by hiding complexity
- Webservices provided by a web site

Flyweight : Reusing the objects by sharing it

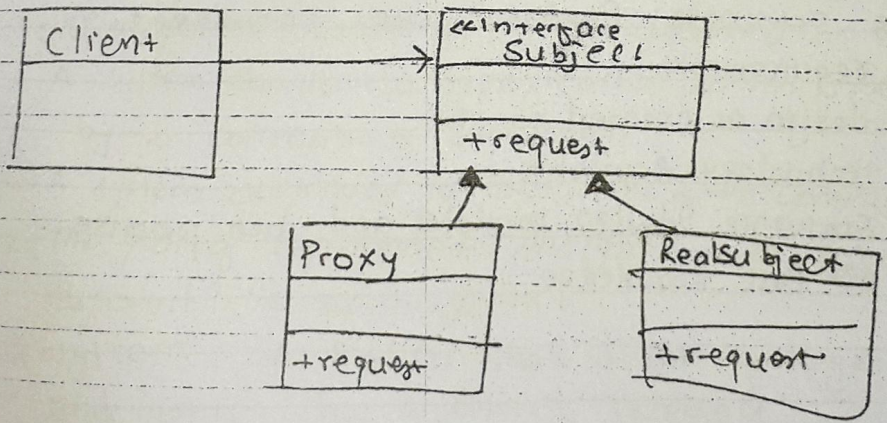
- Designing objects down to the lowest levels of system 'granularity' provides optimal flexibility
- The flyweight pattern describes how to share objects to allow their use at fine granularity without prohibitive cost.
- Many similar objects are used and storage cost is high
- Create a factory that can cache and reuse existing class instances
- It implements Composite to implement shared Leaf nodes

- It shields the Clients from complexities
- It promotes loose coupling betn subsystem and its clients
- It is used when an application uses no. of Object



Example: Modern Browser, it loads all image and save it to cache, when user scroll, it just display required image from cache.

Proxy : an obj repr another obj [surrogate]
- provides the control for accessing the original object
- provides the protection to the original obj. from outside world
- In real world, cheque or credit card is proxy for which our account ie credit card represent cash. provide means of accessing cache



Patterns scope:

- object patterns: specify relationship between objects
  - purpose of dp is to allow instance of diff classes to be used in a same place in a pattern
- class patterns: relationship bet^n classes & their subclasses

The process of Design

- Design is a problem solving proces whose objective is to find and describe way
- To implement the system's functional requirement
- while respecting the constraints imposed by quality, platform and proces requirement

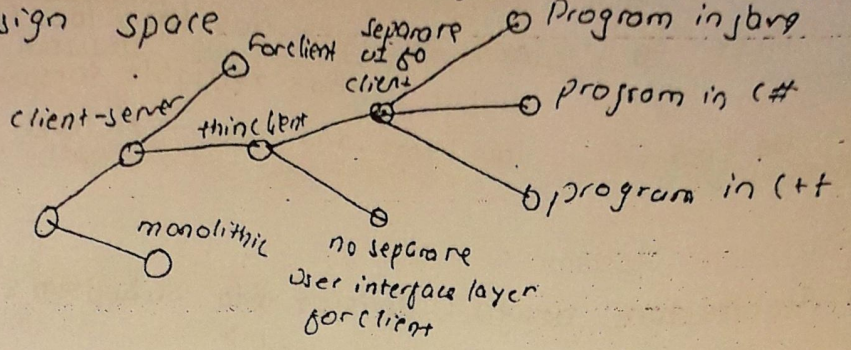
* A designer is faced with a series of design issues
  - These are sub problems of the overall design proble
  - Each issues has several alternative solution
  - Designer makes a design decision to resolve ea issues.
    - This process involves ~~the best~~ choosing the bes option from among the alternatives

Design Decision

The Software Engineer uses knowledge of

a. requirements
b. design as created so far
c. technology Available
d. Software design principles and best practise
e. Past knowledge

The space of possible designs that could be achieved by choosing different set of alternatives is often called design space



Component

Any piece of software/hardware that has a clear role
- A component can be isolated, allowing you to replace it with a different component that has equivalent functionality.
- Many components are design for reusable
- Special purpose functions (Goal meet)

Module

A component that is defined at the programing Language Level.

Ex: methods, classes and packages are models in java

System

- A logical entity, having set of definable responsibilities or objective, consisting of hardware or both
- A system can have a specification which is then implemented by a collection of component
- A system continued to exist, even if its component changes.
- The goal of requirement Analysis is to determine the responsibilities of a system

Subsystem : A system that is part of larger system and and which has a definite Interface

Top down design: to provide the system, a good structure
- First do high level construct and then low level
Bottom up design: useful so that reusable component can be created. low level and then high level.

Different Aspect of Design

A. Architecture Design: division into subsystem & components
: How these will be connected?
: How will they interact?
B. Class ~~Diagram~~ Design : various features of class
C. User Interface Design
D. Algorithm Design : computational mechanism
E. Protocol Design : communication

Design pattern VS Architecture

Design pattern
- Design is tactical
- well known patterns for solving technical problem in a proven way
- How classes, methods collaborate with each other
- pattern describes a problem which occurs over and over again

Architecture
- It mostly address problem of functionality, system partitioning, protocol, security.
- Architecture is strategic
- It compromises the frameworks, tool, programs
(1) Component rather than classes

Principles leading to Good design :

overall goals of good design are

- reducing cost and increasing revenue
- ensure that requirement are conform.
- Accelerating development.
- Usability, Efficiency, reliability, Maintainability, Reusability

1. Divide an Conquer
2. Increase cohesion where possible
3. Reduce coupling where possible
4. Keep the level of abstraction as high as possible
5. Increase Reusability where possible
6. Reuse existing designs
7. Design for flexibility
8. Anticipate obsolescence
9. Design for portability
10. Design for Testability
11. Design defensively

Techniques for making good design decision

1. List and describe the alternatives for the design decision
2. List advantage / disadvantage of each w.r.t our objective
3. determine if any alternative prevent us from meeting one or more objective
4. Choose the alternative that helps ~~you~~ us to best meet our objective
5. Adjust priorities

Evaluation of alternatives

- Security
- Maintainability
- CPU efficiency
- N/w bandwidth efficiency
- memory.

Software Architecture

- Software architecture is process of designing the global organization of a software system, including
  - Dividing software into subsystem
  - Deciding how these will interact.
  - Determining their interfaces.
- The architecture is the core of the design, so all
- The architecture will often constrain the overall effort reusability and maintainability of the system
- Software Architecture is the process of defining a structured solution that meets all of the technical and operational requirement

* Importance of Software Architecture

- better understand of the system
- allows people to work on individual pieces of the system in isolation
- To prepare for extension of system.
- To facilitate reuse and reusability
- It gives the right technical solutions to ensure your ~~access~~ success.
- If base of Soft is solid, -ve impacts on time to time market, costs and adaptability.
- It is Important because :

~~* What~~

a. A basis for Communication
b. Earliest Decision
c. Transferability of the model

→

When we talk about the architecture of a software, we talk about the plans that describes a set of aspects and decision that are important to a software.
- This implies taking into consideration all kinds of requirement (performance security), the org. of system, how the system parts communicate with each other, whether they are external dependencies what the guidelines and implementation technologies are, what risks to take into consideration

- Softwar Arch. should be strong and easy to maintain when we find bugs.
- domain concepts that ready all members will understand
- flexible, extensible, scalable, usable on long term
- Refactoring should be easy
- when adding features, performance shouldn't decrease.

- People believe that, by looking a final product, one can't say it have a good Software Architecture.
- Still, we can find signs of good one
- To be a good Software Architecture, Software should be:
  a. Software is user-friendly
  b. Solution is flexible, user adapt it quite easily
  c. If soft is scalable
  d. easy to test, Performance is fast.
  e. Strong & reliable

Architectural Model

- logical breakdown into subsystem
- Interfaces among the subsystem
- data will be shared among subsystem
- Components exists at run time;

Design Stable Architecture

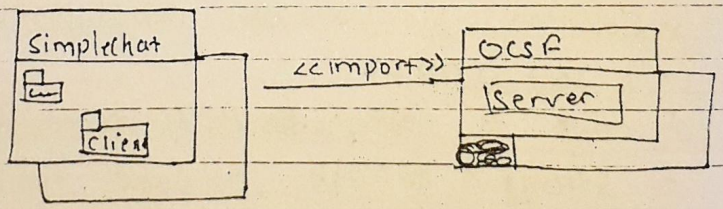
To ensure maintainability and readality of system, architectural model must be designed to be stable.
- Being stable means that the new features can be easily added with only small changes to the architecture

Design an architectural model

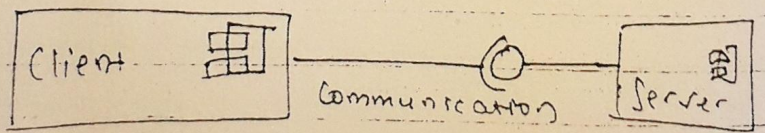
1. Start by sketching an outline of arch.
   - based on principle requirement and usecases
   - determine main component
   - Choose among various architectural
   - Refine the architecture
   - mature the architecture

Architecture using UML

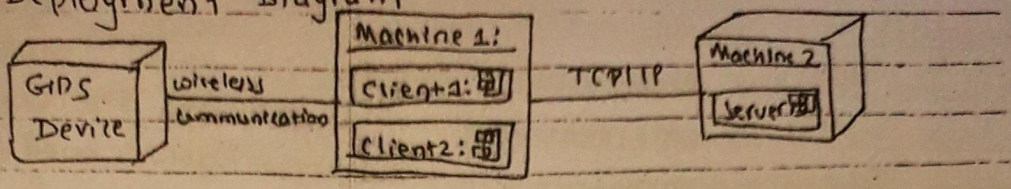
- Package Diagram



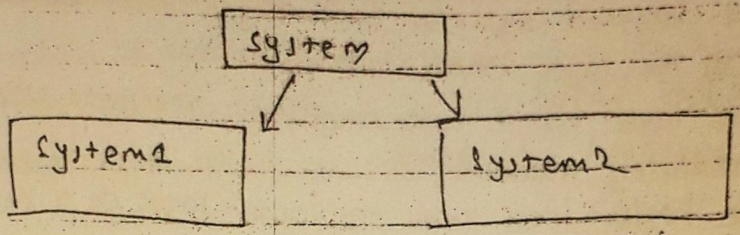
- Component Diagram



- Deployment diagram



- Subsystem Diagram



Architectural Patterns :

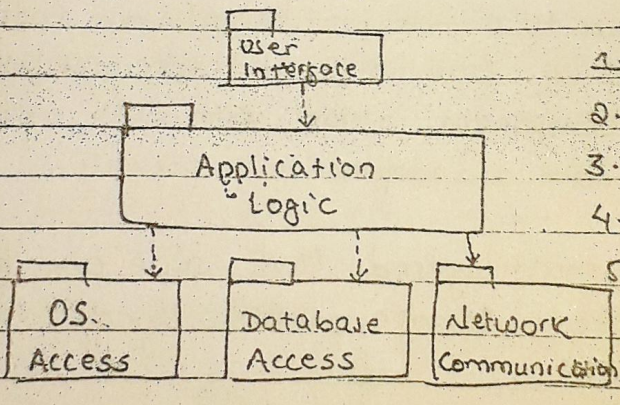
- The notion of patterns can be applied to software architecture.
- These are called architectural patterns or style.
- Each allows you to design flexible systems using component
- It tell us how to ~~architector~~ organise our code.
- highest level of granularity and specifies layer.
- how these component interact with each other
- Component are as independent as possible

Example:
Client server, multilayered, MVC, pipe and filter, message oriented, service oriented, Monolithic, Event Driven.

Architectural pattern is general, reusable solution to a commonly occuring problem in Soft Arch within given context.

A. MultiLayer Architectural Pattern:

- MLA is an arch. model that propose the org. of Software components into different layers. Each of these layers is implemented as a physically seperated containers of Software components
- In a Layered system, each layer communicates only with the layer immediately below it.
- Each layer has a well-defined interface used by layer immediately above.
- The higher level sees the lower layer as a set of Semi
- A complex System can be built by superposing layers at increasing level of abstraction.
- It is important to have a separate for UI
- Layers immediately below the UI Layer provide the application functions determined by the use case.
- Bottom layers provide general ~~event~~ services
  Eg: Nlw communication, database acces



a) Typical layers in an Application program

1. Divide and Conque
2. Increase Coherion
3. Reduce Coupling
4. ~~Dis~~Increase Abstracti
5. Increase Reusability
6. Reuse exist design
7. ~~Increase~~ Anticipate des
8. Increase Flexibilit
9. ~~Re~~design for Portabl
10. design for Testabl
11. design defensive

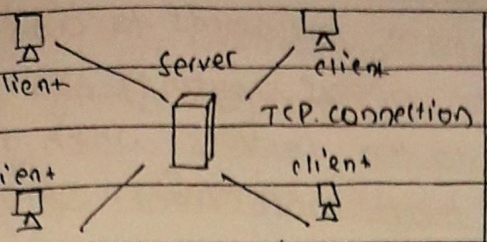


fig: client-server

Client server is a program relationship in which one progam (client) requests a service or resources from another program (the server).

- The client-server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers and service requesters, called client.
- clients and servers exchange messages in a request-response pattern. The client sends a request, server response
- clients/server example are: web browser, mail
- The client server architecture decreased traffic network by providing a query/response rather than a total file transfer.
- Remote procedure call (RPCs) or standard query language (SQ are used to communicate between client and server.
- The performance is depends on processor speeds, memory, bandwidth, capacity, disk speeds, input-output devices
- The ~~eth~~ system is scalable, client can be added.
- The environment is Heterogenous and multivendor ie the os, hlw is not same bet^n client and server.
- client and server process communicate through API and RPCs.

* Another type of nlw architecture is peer to peer (P2P), in which computer is both client and server.

- Client Server is more secured than peer to peer N/W because client server can have passwords to own individual profiles so that nobody can access anything what they can.
- All the data is stored onto the servers which generally have far greater security than most client. Server can control the access and resources better to guarantee that only those clients with appropriate permissions may access and change data.
- P2P is less secure because security is handled by individual computers, not on the network as a whole.
- P2P doesnot have central storage or authentication of client, all clients are conversly dedicated servers

Client Server	Peer to Peer
1. Specific server and specific clients.	1. each node act as client o server
2. client request for server responses.	2. each node can request for service and provide servic
3. Focus on sharing Information.	3. Focus on Connectivity.
4. The data is stored in a centralized server.	4. Each peer has its own data.
5. When too many client request, server can be bottlenecked	6. As service are provided by several servers, not bottlenecked.
6. Expensive	6. Cheap
7. Secured	7. less secured
8. stable / scalable	8. unstable / o

* UML is a standard langooge for specifying, visualizing, constructing and documenting the artifacts of the software systems.

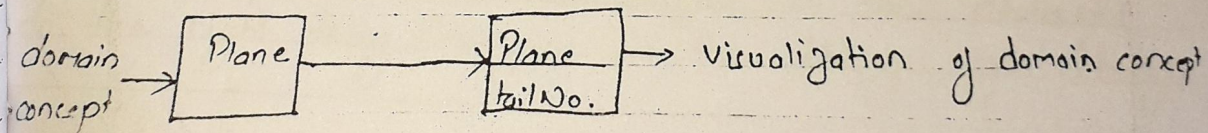
Object Oriented Design and Modelling

Q. What is object oriented analysis and design? How does object orientation emphasizes on representation of object. Describe with example.

⇒ During object-oriented analysis there is an emphasis on finding and describing the objects or concepts in the problem domain. for example, in the case of the flight information system, some of the concepts include plane, flight and pilot.

During object-oriented design there is an emphasis on defining software objects and how they collabrate to fulfill the requirements. for example a Plane software object may have a tailNumber attribute and a getFlightHistory Method.

Finally during implementation design objects are implemented such as a Plane class in Java.



representation in an object oriented programming language →

```
public class Plane
{
    private string tailNumber;
    public list getflightHistory();
}
```

Ans. Object Oriented emphasizes representation object.

* Analysis emphasizes in an investigation of the problem and requirements, rather than a solution. For example: if a new online trading system is desired, Analysis answers the following questions:

How will it be used?
What are it's functions?

Analysis is a broad term and it is reffered to as object oriented analysis.

Design emphasizes a conceptual solution (in software and hardware) that fulfills the requirement rather than it's implementation. Design idea often exclude low-level or obvious details obvious to intented customers. Ultimate design can be implemented and the implementation (such as code) express the true and complete design.

They have been summarized in the phrases do the right thing (Analysis) do the things right (design

Q. What is Unified Process (UP)?

A software development process describes an approach to building, deploying and possibly maintaining software. The UP has emerged as a popular iterative software development process for building object-oriented systems. It has 4 phases:

1. Inception: approximate vision, business case, scope, vague estimates.

2. Elaboration: refined vision, iterative implementation of the core architecture, resolution of high risks, identification of most requirements and scope, more realistic estimate

3. Construction: iterative implementation of the remaining lower risk and easier elements and preparation for deployment.

4. Transition:

It has 3 disciplines:

① Business Modelling: single app → domain object modelling large scale business, dynamic modelling across entire enterprise

② Requirements: writing use case and identifying non-functional requirements.

③ Design: All aspects of design, including the overall architecture, objects database, networking etc.

- The UP is an iterative process. Iterative development influences how to Introduce OOAD, and to understand how it is best practised
- UP practices provide an example structure for how to do and thus how to explain OOAD.
- UP is flexible, and can be applied in lightweight and agile approach that includes practices from other agile methods.

Workflows	inception phase	elaboration Phase	construction Phase	transition Phase
Business Modelling				
Requirement Analysis				
Design				
Implementation				

i) Sequence diagram:

The sequence diagram captures the time sequence of message flow from one object to another. It shows how objects communicates with each other over time. That is sequence diagrams are used to model object interactions arranged in time sequence and to distribute use case behaviour to classes. They can also be used to illustrate all the paths a particular use case can ultimately produce. It can be illustrate as:

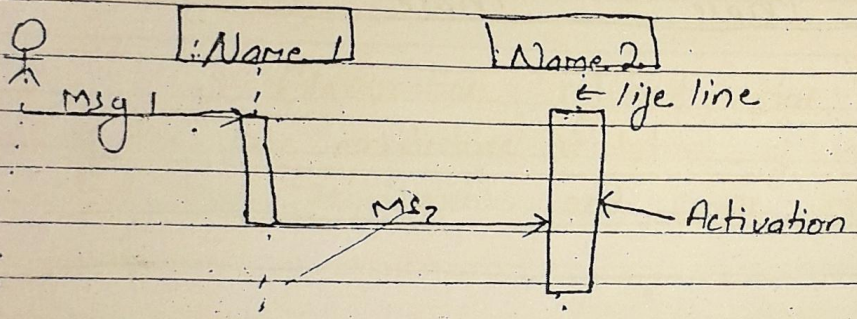


fig:- Sequence Diagram.

ii) Collaboration diagram:

Collaboration diagrams illustrate object interactions in a graph or network format, in which objects can be placed anywhere on the diagram. Collaboration diagrams show how objects interact and their roles. They represent the structural organization of object. They are best suited to the portrayal of simple interactions

among relatively small numbers of objects. It can be illustrat as:

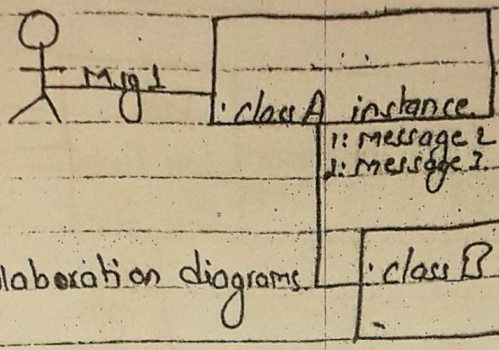


Fig:- Collaboration diagrams

They are similar as they are called interactic diagrams. Both of them display messages between objects. Both sequence and collaboration diagrams show synchronous messages.

Ex:- (#)

- Object Oriented Design
  - → Real Use case
  - → Design class Diagram (DCD)
  - → Interaction Diagram
    - → Sequence Diagram
    - → Collaboration Diagram
- Object Oriented Analysis
  - → Essential Use case
  - → Domain modelling
  - → System Sequence Diagram (SSD)

for Object Oriented Analysis:

(a)

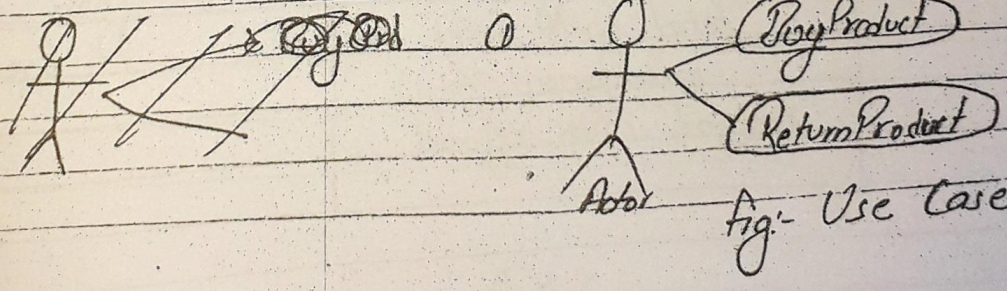


Fig:- Use Case.

(II)

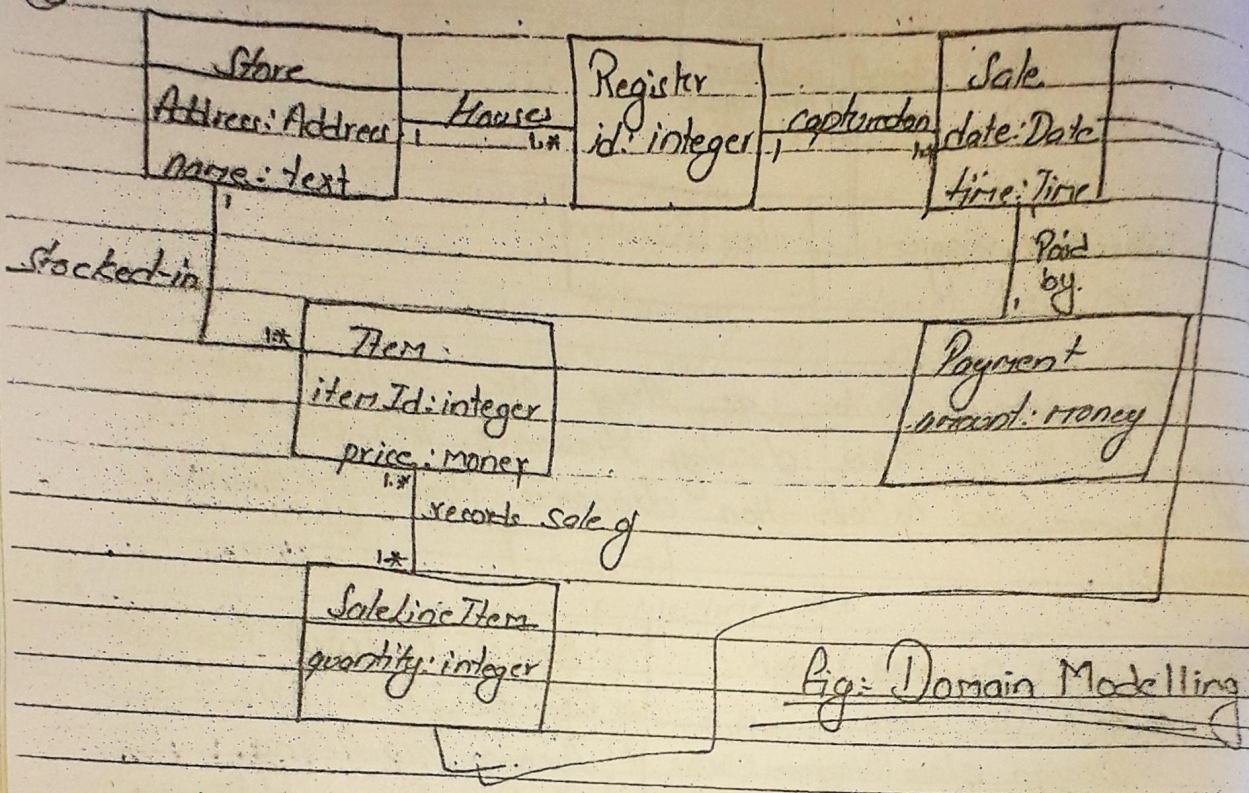
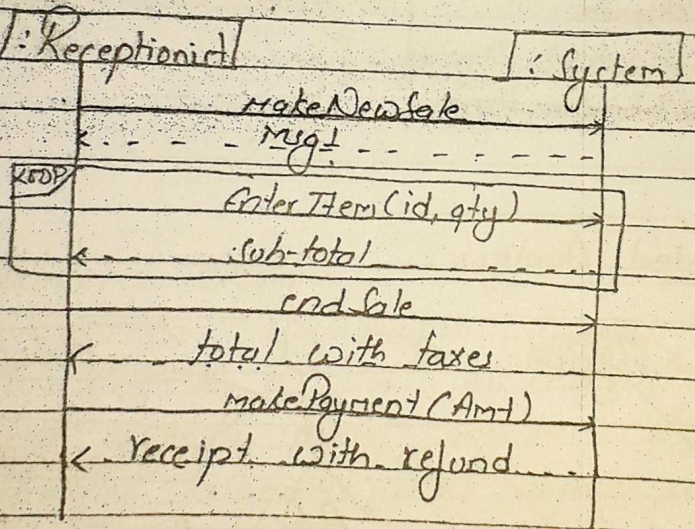


Fig: Domain Modelling

(III)



What is Iterative and Evolutionary Development?

⇒ In iterative approach, development is organized into a series of short, fixed-length mini projects called iteration. Outcome of each iteration is a tested, integrated and executable partial system. Each iteration include its own requirement analysis, design and implementation and testing activities. The system grows incrementally overtime, iteration by iteration, and thus this approach is also known as iterative and incremental development.

Iterative development is embrace change (Accept change). Each iteration involves choosing a small subset of the requirements and quickly designing, implementing and testing. In early iterations the choice of requirements and design may not be exactly what is ultimately desired. But the act of swiftly taking a small step, before all requirements are finalized, or the entire design is speculatively defined, leads to rapid feedback.

Q. What is GRASP? What are it's principles?

=> GRASP is an acronym for General Responsibility Assignment Software Patterns. GRASP provides guidance for assigning responsibilities to classes and, to a limited extent, determining the classes that will be in an object-oriented system.

To short GRASP stands for designing objects with responsibilities.

The GRASP principles or patterns are a learning aid to help you understand essential object design in a methodical, rational, explainable way. This approach is based on patterns of assigning responsibilities. GRASP defines 9 basic Object Oriented Design principles or basic building blocks in design.

- -> Information Expert
- -> Creator
- -> Controller
- -> low coupling
- -> High Cohesion
- -> Polymorphism
- -> Pure fabrication
- -> Indirection
- -> Protected variations.

~~GRASP Examples~~

Expert pattern deals with the problem of what is a general principle of assigning responsibilities to objects? And gives the solution as: Assign a responsibility to the information expert in the class that has the information necessary to fullfill the responsibility for eg:- Who should be responsible for knowing grand total of sale? And the answer is if there are relevant classes in Design Model, look there first and if not look in Domain Model and attempt to use its representation to inspire the creation of corresponding design classes.

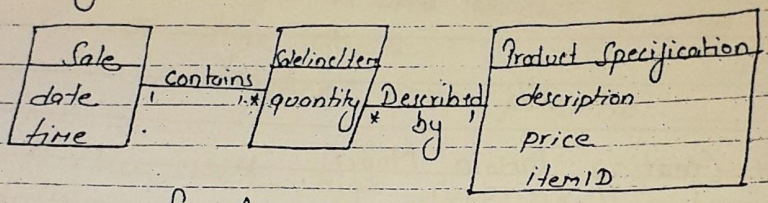
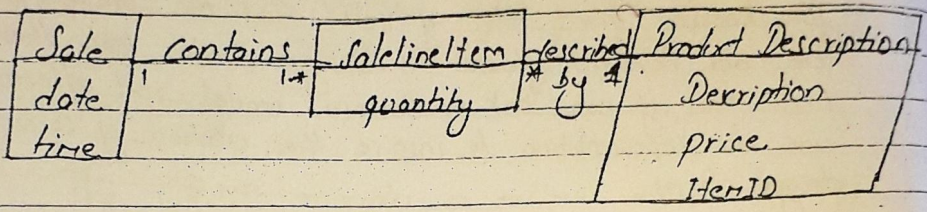


Fig:- Associations for Sale

Whereas creator pattern deals with the problem as who should be responsible for creating a new instance of some class? And gives solution as: Assign class B the responsibility to create an instance of class A if one or more of the following is true:

- B aggregrates A objects
- B contains A objects
- B records instances of A objects
- B closely uses A objects
- B has the initializing data that will be passed to A when it is created.

Here, B is creator of A objects. If one or more options apply, prefer a class B which contains class A. for eg:- Who should be responsible for creating a SaleLineItem objects, the creator pattern suggests to choose Sale as an instance of SaleLineItem.



④ Steps for Creating Domain Model

1. List the candidate conceptual classes using the Conceptual Class Category list and noun phrase identification technique related to the current requirements under consideration.

2. Draw them in a domain model

3. Add the associations necessary to record relationships for which there is a need to preserve some memory.

4. Add the attributes necessary to fulfill the information requirements.

(iii) Low Coupling: How to support low dependency, low change impact, and increased reuse?
=> Assign a responsibilities so that coupling remains low.

Coupling is a measure of how strongly one element is connected to, has knowledge of, or relies on other elements. Low coupling is a evaluative pattern that dictates how to assign responsibilities to support:

- lower dependency between the classes
- change in one class having lower impact on other classes.
- higher reuse potential.

A class with high coupling suffer from following undesirable problems:

(i) Change in related class force local change
(ii) Harder to understand in isolation.
(iii) Harder to reuse because its use requires the additional presence of the classes on which it is dependent.

Benefits are:-

(i) not affected by changes in other components
(ii) Simple to understand in isolation
(iii) Convinient to reuse.

High Cohesion:

Q-> How to keep complexity managable?

A-> Assign a responsibility so that cohesion remains high.

- Cohesion is a measure of how strongly related and focused the responsibilities of an an element are.
- An element with highly related responsibilities, and which does not do a tremedous amount of work, has high cohesion.
- A class with low cohesion does many unrelated things or does too much work. Such classes are undesirable; they suffer from following problems:
  - -> Hard to comprehend
  - -> Hard to reuse
  - -> Hard to maintain
  - -> delicate; constantly effected by change.

Controller Pattern.

Problem: Who should be responsible for handling an input system event?

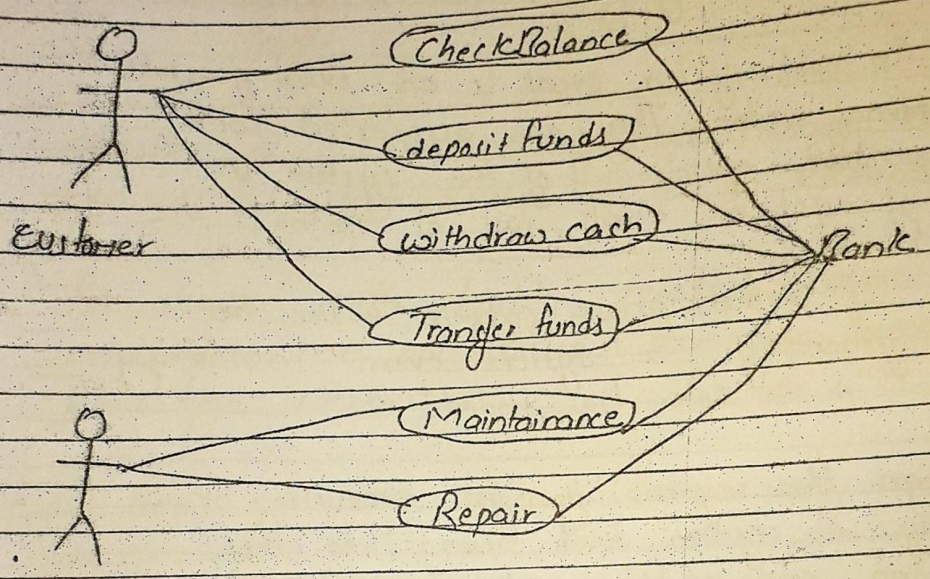
Solution: An input system event is an event generated by an external actor. They are associated with system operations. operations of the system in response of system events, just as methods are related.

For eg: If cashier presses "endsale" button he is generating a system event indicating "a sale has ended"

⇒ So, assign the responsibility for receiving or handling a system event message to class representing one of the following choices:

⇒ Represents the overall system, device or subsystem.
⇒ Use the same controller class for all system events in the same use case scenario.

2) Use case for ATM transaction:



Static
1. Class diagram
2. Object diagram
3. Package diagram
   ↳ Model diagram
4. Composite structure
   ↳ 4.1 internal
   ↳ 4.2 Collaboration

Dynamic
⑥ Use Case
⑪ Information
⑪ Activity

⑭ State Machine
↳ Behavioural sets
↳ problem

Q. How procedure Analysis and Design is different from OOAD, illustrate with examples.

	Structured	Object-oriented.
Phase		
- Method	SDLC	Iterative/Incremental
- Focus	Process	Objects
- Risk	High	low
- Reuse	low	High
- Maturity	Matured and widespread	Emerging.
- Suitable for	Well-defined projects with stable user requirement	Risky large projects with changing user requirements.
- Analysis	Structuring Requirements: - DFD's - Structured English - Decesion Table - ER Analysis	Requirement Engineering: - Use Case Model - Object Model •find Classes & class relatio •Object Interaction

Q. Explain different phases and discipline of Rational Unified Process?

Rational unified process is a complete software development process framework developed by Rational Corporation. Its an iterative development methodology based upon six industry proven best practices.

RUP is divided into four phases:

i) Inception phase

ii) Elaboration Phase
iii) Construction Phase
iv) Transition Phase

i) Inception phase:

The primary objective is to scope the system adequately as a basis for validating initial costing and budgets. To complement the busine case, a basic use case model, project plan, initial risk assesment and project description are generat It is the approximate vicion, business care, scope vouge estimates

ii) Elaboration Phase:

The Primary objective is to mitigate the key risk items identified by analysis up to the end of this phase. The elaboration phase is where the project starts to take shape. In this phase, the problem domain analysis is made and the architecture of the projecto gets it's basic form. It is the resolution of high risk identification of most requirements and scope, more realistic estimates.

iv) Construction Phase:

The primary objective is to build th software system. In this phase, the main focus is on the development of components and other

features of the system.

(iv) Transition Phase:

The primary objective is to transit the system from development into production, making it available to users and understood by the end user. The activities of this ~~beta testing~~ phase include training the end users and maintainers and beta testing the system to validate it against the end user's expectations.

RUP follows following disciplines:

(i) Business modelling:

Discovers all business use cases. Details a single set of business use case.

(ii) Requirements:

Discovers all requirement use cases.

(iii) Analysis and Design: Decides on technologies for the whole solution.

(iv) Implementation:

Owns the build plans that show what classes will integrate with one another.

(v) Test:

Ensures that the text is complete and

conducted for the right motivators

(vi) Deployment
Oversees deployment for all deployment units.

(vii) Project Management:
It makes go/no decisions

(viii) Environment:
Creates guidelines for using a specific tool.

(ix) Configure and Change Management:
Sets up policies and plans and establishes a change control process.

Year : 2016 (Fall)

1(a) What is object-oriented analysis and design? Support your answer with suitable example.

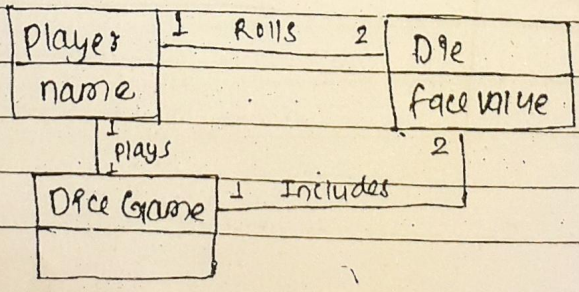
Sol) Object Oriented Analysis (OOA) is the procedure of identifying software engineering requirements and developing software specifications in terms of a software system's object model, which comprises of interacting objects. So, Object oriented analysis and design (OOAD) is a popular technical approach for analyzing and designing an application, system or business by applying object oriented programming, as well as using visual modelling throughout the development life cycles to foster better stake holder communication and product quality.

For example, consider a 'dice game' in which software simulates a player rolling two dice. If total is seven, he wins else lose.

Define use case.

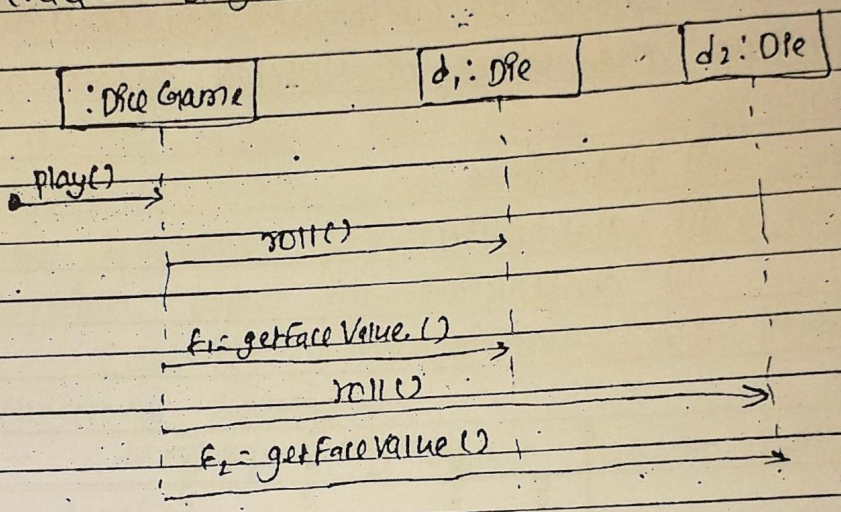
- Player requests to roll the dice. System presents results: If the dice face value totals seven, player wins; otherwise, player loses.

Define a Domain Model



So, domain model helps in visualization of the concepts or mental models of a real-world domain. So, it is also called conceptual object model.

Assign Object Responsibility and Draw Interaction diagram.



Define Design Class Diagram

DiceGame	1 — 2	Die
die1: Die		faceValue: int
die2: Die		getFaceValue: int
play()		roll()

Hence OO designs and languages can support a lower representational gap between the software components and our mental models of a domain.

1(b) What are phases of Unified process? Describe in short.

Soln → The Unified process is a popular iterative and incremental software development process framework. It is the best known and extensively documented refinement of the Unified process. It is not just only a process but rather an extensively used framework which should be customized for specific organizations or projects. The phases of unified process are enlisted below:

(i) Inception
(ii) Elaboration
(iii) construction
(iv) Transition

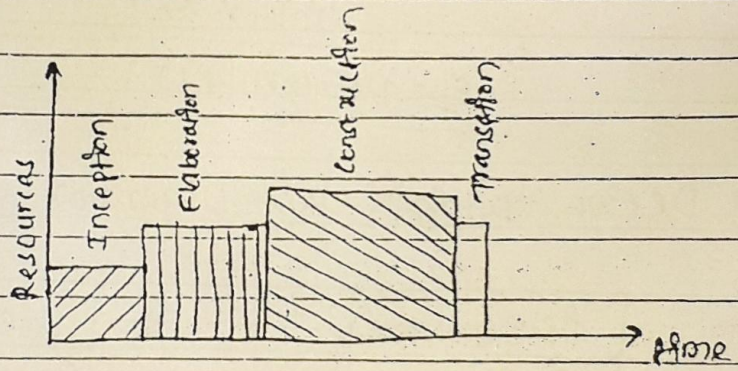


Figure: Graph showing time Vs resources in UP.

* Inception phase:

Inception is the smallest phase in project, and ideally is short phase. The following are

- prepare preliminary project schedule and cost estimate.
- Feasibility
  - Buy or develop it

The life cycle objective milestone marks the end of the inception phase.

Elaboration

The goal of elaboration phase is to establish the ability to build new system given the financial constraints, and other kinds of constraints that the development project faces. The task in elaboration phase includes:

- Capturing a healthy majority of remaining functional requirement.
- Addressing significant risk on ongoing basis.
- expanding the candidate architecture into full architectural baseline.

And major milestone associated with elaboration phase are:

- most of the functional requirements for new system ~~has~~ have been captured in the use case modes.
- architectural baseline is complete which will serve as solid foundation for ongoing development.

Construction:

The primary goal of construction phase is to build a system capable of operating successfully in beta customer environments. During construction, the team performs tasks that involve building the system iteratively and incrementally making sure that the viability of the system is always evident in executable form.

Transition:

The primary goal of transition phase is to roll out the fully functional system to customers. During transition, the project team focuses on correcting defects and modifying the system to correct previously unidentified problems.

2 (a) "System development is model development". Do you agree? Justify.

Sol'n → The system development is process of system engineering, information system and software engineering to describe a proress of planning, creating, testing and deploying an information system. Where as model development is the formulation of conceptual model as a result of system modeling that describes and represents systems.

As like in model development, first

Of all we specify domain for which the model is to be developed. Then, the model is developed as per the need, it is tested for it's output. Similarly a system is developed as in the same way. It goes through number of tests and improvements. As like a model of new component is needed then it is attached, like the same way a sub-system is added into system when the new component is needed. Hence, we can state that "system development is model development."

2(b) Discuss the strength and weakness of object-oriented and prioedural programming with the help of banking transaction example.

Sol) The strength of object-oriented programming are:

- Modularity for easier troubleshooting.
- Reuse of code through inheritance
- Flexibility through polymorphism.
- Effective problem solving.

The weakness of OOP are:

- programs are of large size.
- it requires a lot of work to create a project.

- Object oriented programs are slowes than other programs. programs demand more system resources, thus slowing the program down.

The strength of procedural programming are:
- Excellent for general purpose programming.
- No-need to reinvent the wheel as well tried and tested coding algorithms are available.
- Good level of control without having to know precise target CPU details.
- portable source code.

The weakness of procedural programming are:
- Lack of a cognitive structure for understanding the relationship between all the procedures and functions and data.
- programmers need to specialize in a specific procedural programming language.
- It is difficult to relate with the real world projects.
- It is difficult to maintain, if the code grows larger.
- lack of security.

For example, let us take banking transaction:

OO programming	procedural programming
main()	main()
{	{
Customer c1 = New Customer(acc-no);	Struct Customer c1;
/* some codes */	/* some codes */
Transaction t = New Transaction()	c1.accno = acc-no;
t.makeTransaction(c1, amount);	/* Some codes */
t.flush();	makeTrann(c1, amount);
/* Some codes */	/* some codes */
}	}

3(a) Describe work flow for capturing requirement as use cases, including the participating workers and their activities.

Sol:) Use case model allows developer and customer to agree on requirements, i.e. conditions and capabilities to which the system must conform a model of system containing actors and use cases and their relationships. Actor represents user type. Each type of users may be represented by one or more actors often corresponds to worker in a business. A role of worker

defines what a worker does in a particular business process. The roles can be used to derive corresponding actors will play. Each use case represents a way the actors use the system. A use case specifies a sequence of actions, including alternatives of the sequence; that the system can perform. Example: Withdraw money (granted, denied, different amount etc). Use case instance is the execution of a usecase. It is one path through the use case. An sequence of interactions between an actor instance and the use case. Each worker in a business is an actor. During such process of identifying actors, it should be possible to identify at least one user who can play the candidate actor- minimal overlap between roles.

Example: buyer, seller, accounting system

3(b) Design is four dimensional view of a system. Justify along with design concepts.

SOL) Software design is a process through which requirements are translated into a representation of software. From a project management point of view, software design can be conducted in two major steps:

(1) preliminary Design:

concerned with the transformation of requirements into data and software architecture.

(9) Detail Design:

Focuses on refining the architectural representation and lead to detailed data structure and algorithmic representation of software.

The three main design activities concerned in Design phase are: Data Design, Architectural Design and procedural Design. In addition, many modern applications have a distinct interface design activity.

- Data design is used to transform the information domain into data structures.
- Architectural Design is used to develop a modular program structure and represent the control relationships between modules.
- procedural design is used to transform structural components into a procedural description of the software.
- Interface Design establishes the layout and interaction mechanisms for human-machine interaction.

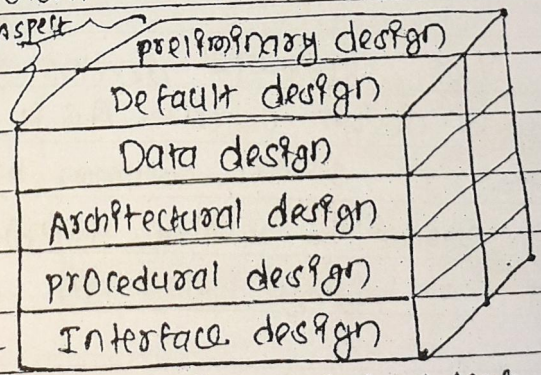


Figure: Relationship bet^n technical & Mgt. Aspect of design.

4(a) Define design pattern. How is design pattern important? Is software development possible without applying design pattern?

Sol? → A design pattern is a general repeatable solution to a commonly occuring problem in software design. A design pattern is not finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

The design pattern are important because:

- They can speed up development process by providing tested, proven development paradigms.
- Reusing it helps to prevent subtle issues that can cause major problems and improve code readability for coders.
- They provide general solutions, documented in a format that does not require specification to a particular problem.
- Allows developers to communicate using well-known well understood names for software interactions.

Yes, software development is possible without applying design pattern. As it is just a general solution to some common problem. But developers may choose to resolve using other methods. But using them highly helps in speeding up development.

4(b) Describe suitable design pattern for following problem. "What is the best way to represent related objects (occurrence) in a class diagram?"

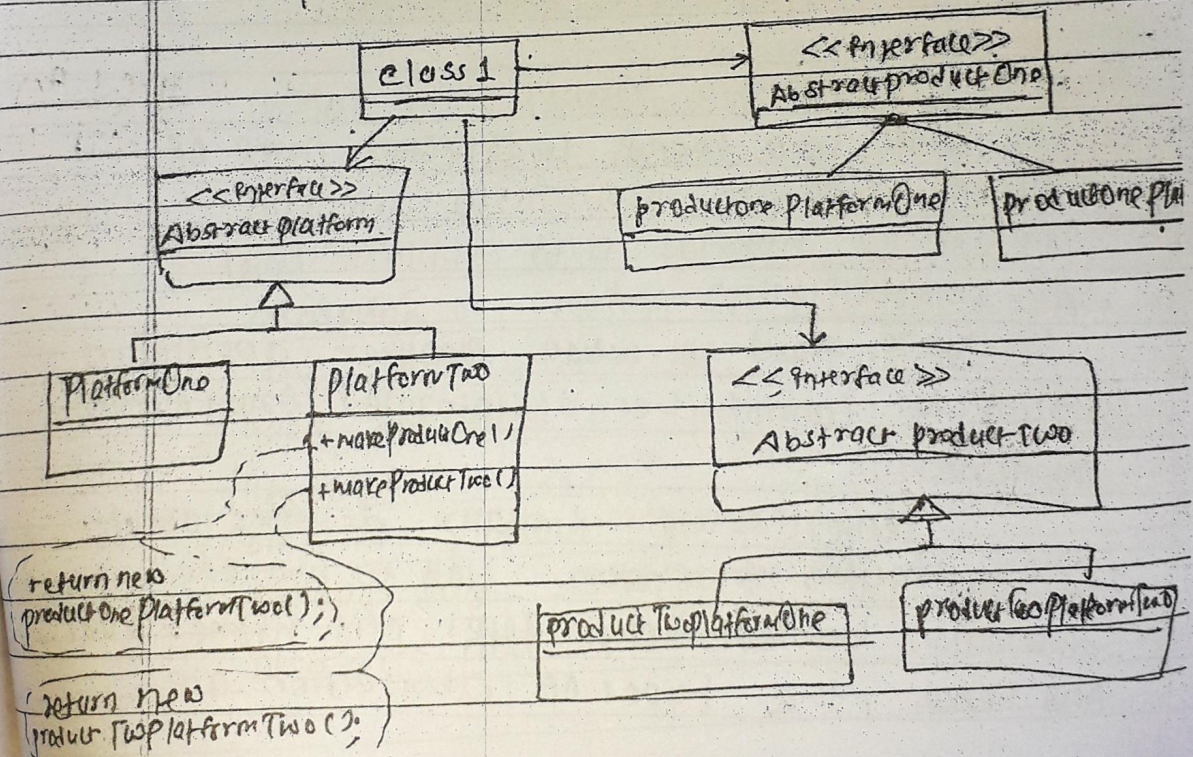
Sol: As looking at the problem, the best solution of design pattern is Abstract Factory. It is so because

→ It provides an interface for creating families of related or dependent objects without specifying their concrete classes.

→ A hierarchy that encapsulates: many possible "platforms and the construction of a suite of "products".

→ The new operator considered harmful.

So, the structure will be.



5(a). Define design principle, design concept and design pattern. What are the disadvantages of design patterns?

Sol? → Design principles represent a set of guidelines that help us to avoid having a bad design. The design principles are associated to Robert Martin who gathered them in "Agile Software Development: principles, patterns and practices". According to Robert martin there are 3 important characteristics of a bad design that should be avoided.

* Rigidity:
  It is hard to change because every change affects too many parts of the system.

* Fragility
  When you change unexpected parts of system break.

* Immobility
  It is hard to reuse another application because it cannot be disentangled from current application.

Design concepts usually describes about the abstraction in system. How to provide abstraction between lower level and higher level. And hence, the types of abstraction are

① procedural Abstraction
② Data Abstraction

Design pattern is a general repetable solution, to a commonly occuring problems in software design. It is description or template for how to solve a problem that can be used in many different situations.

The disadvantages of Design pattern are
- They do not lead to direct code reuse.
- They are complex in nature.
- They are deceptively simple
- They are validated by experience and discussion.

5(b) What is software architecture? why do we need it?

Sol^n → Software architecture refers to the high level structures of a software system, the discipline of creating such structures, and the documentation of these structures. These structures are needed to reason about the software system. Each structure comprises software elements, relations among them, and properties of both elements and relations.

Software architecture is about making fundamental structural choices which are costly to change once implemented. Software architecture choices include specific structural options from possibilities, in the design of software. For example, the system that control the space shuttle launch vechicle had the requirement of being very fast and very reliable. Therefore an appropriate real-time computing language would need to be choosen.

We need software architecture for

- high productivity
- Better code maintainability
- higher adaptability
- Hype agnostic

6 (a) Describe software oriented architecture and design principles it helps to adhere.

— A software-oriented architecture (SOA) is a style of software design where services are provided to the other components by application components, through a communication protocol over a network. The basic principles of service oriented architecture are independent of vendors, product and technologies. A service is a discrete unit of functionality that can be accessed remotely and acted upon

and updated independently, such as retrieving a credit card statement online. A service has four properties according to one of many definitions of SOA.

(i) It logically represents a business activity with specified outcome.

(ii) It is self-contained.

(iii) It is a black box for it's consumers.

(iv) It may consist of other underlying services.

The service oriented design principles may be broadly categorized as follows:

- Standardized service contract
- Service loose coupling
- service abstraction
- Service reusability
- Service autonomy
- Service statelessness
- Service discoverability

6(b) Discuss in short about MVC Architecture with suitable diagram.

→ Model-view-controller (MVC) architecture is software architectural pattern for implementing good software on computers. It divides a given application into three interconnected parts. This is done to separate internal

representation of information from the user. Information is presented to and accepted from the user the MVC design pattern decouples these major components allowing for efficient code reuse and parallel development.

Traditionally used for desktop GUIs, this architecture has become popular for designers of web applications and even mobile, desktop and other clients. Popular programming language like Java, C#, Ruby, PHP and others have popular MVC framework that are currently being used in web application development straight out of the box.

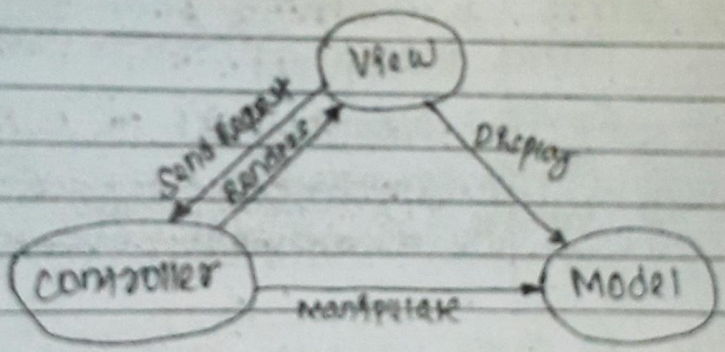


Fig: MVC Architecture

The model stores data that is retrieved according to commands from the controller and displayed in the view. A view generates new output to the user based on changes in the model. A

Controller can send commands to the model's state. It can send commands to it's associated view to change the view's presentation of the model.

7 (a) player role design pattern:

We know during development, we may need to assign different roles in different context of an application. An object may not need to play one or both roles simultaneously, so it is desirable to improve encapsulation keeping information related to only one role in one class, or making two classes to describe the same object is not good OO design so, let us view an example:

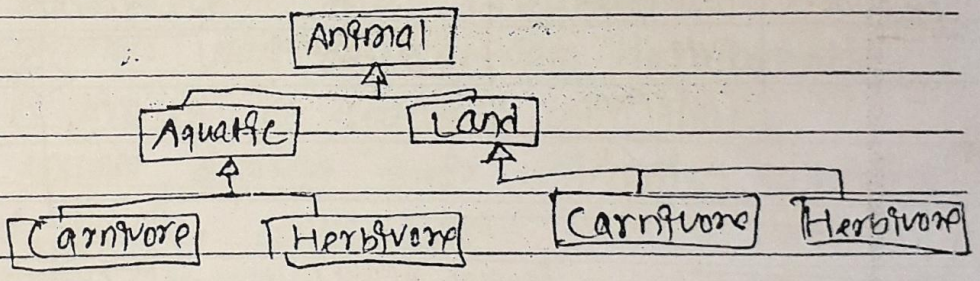


Figure: Badly designed classes.

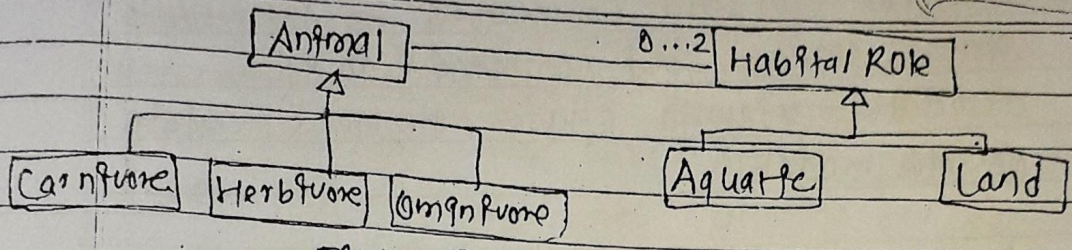
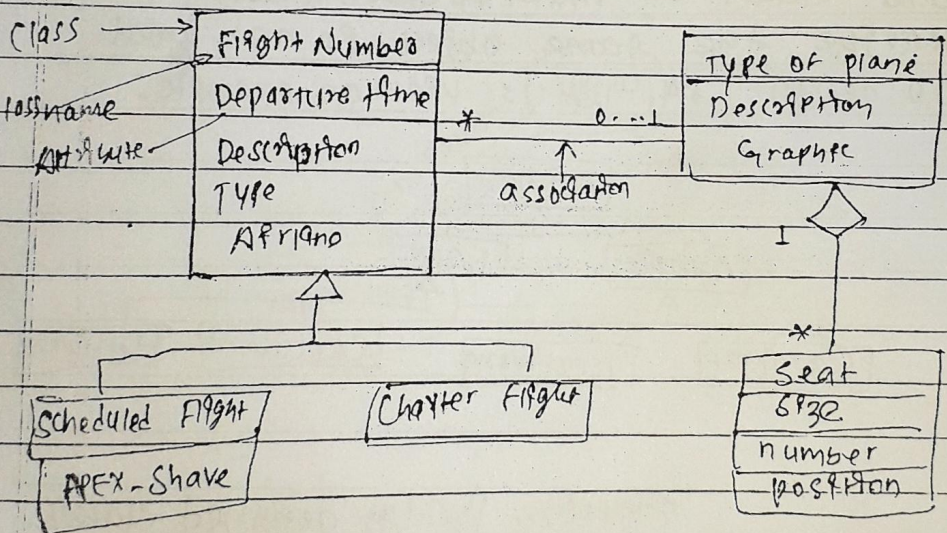


Fig: Finely designed classes.

7 (c)

Class Diagram

Class Diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations, and the relationships among objects.



A Class represents a relevant concept from the domain, a set of persons, objects, or ideas that are depicted on the IT system.

Year: 2015 (spring)

What is UML? Is UML an object oriented supported tool to design object oriented systems. Support your example with a suitable example.

Unified modeling language (UML) is a standardized modeling language enabling developers to specify, visualize, construct and document artifacts of a software system. Thus, UML makes these artifacts scalable, secure and robust in execution. So, UML is an important aspect involved in object oriented software development.

The UML is a tool for specifying s/w systems standardized diagrams types to help us describe and visually map a software system design and structure. Many article discuss great feature of UML, but just a few of them describe experimental results with UML in real-life projects. In order to quantitatively present the advantages of UML over common way of software development.

A group of students who took (attended the course during their university education on C++, SQL basic, Java basic and C were divided into two subgroup. Among them, group (A) took UML courses of presentation while the group (B) didn't attend any UML sessions. Now, two systems were to be designed. The non-UML group or group (B), having no knowledge on UML nor experiences in the object oriented

system design and development had to design the systems using previous knowledge and sense. so, they just made few tough descriptions of the system operation and some flow- diagrams on the flip side, UML group could use various tools of UML to understand and show the internal structure & architecture of the system. It created mostly use case sequence and activity diagrams and even class diagram that show classes, methods and attributes. Now, implementation as a final goal of every development process requires that many details got to be defined.

1(b) What are the activities of Unified process? Describe in short.

Soln: The unified process is a popular iterative and incremental software development process framework. It is best known and extensively documented refinement of the Unified process. It is not just only a process but rather an extensively used framework which should be customized for specific organizations or projects. The phases and activities of unified process are enlisted below:

(i) Inception
(ii) Elaboration
(iii) construction
(iv) Transition

Inception

Inception is the smallest phase in project and ideally is short phase. In inception phase idea is planned. The following are type or goals for the inception phase:

- Establish.
- prepare preliminary project schedule and cost estimate
- Feasibility
- Buy or develop it.

The life cycle objective milestone marks the end of the inception phase.

Elaboration

The goal of elaboration phase is to establish the ability to build new system given the financial constraints, and other kinds of constraints that the development project faces. The task in elaboration phase includes:

- Capturing a healthy majority of remaining functional requirement.
- Addressing significant risk on ongoing

basis.
- expanding the candidate architecture into full architectural baseline.

And major milestone associated with elaboration phase are:

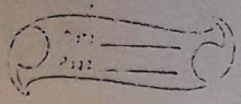
- most of the functional requirements for new system have been captured in the use case model.
- architectural baseline is complete which will serve as solid foundation for ongoing development.

Construction

The primary goal of construction phase is to build a system capable of operating successfully in beta customer environments. During construction, the team performs tasks that involve building the system iteratively and incrementally making sure that the viability of the system is always evident in executable form.

Transition

The primary goal of transition phase is to roll out the fully functional system to customers. During transition, the project team focuses on correcting defects and



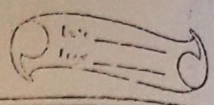
modifying the system to correct previously unidentified problems.

④ What is business modelling? What is the relationship between Business modeling requirement gathering in the development of a software? Take one example to prove your point.

↳ The process of Business model design is part of business strategy. Business model design and innovation refer to the way a firm (or a network of firms) defines its business logic at the strategic level. In contrast, firms implement their business much at the operational level, through their business operation.

For many projects, deciding what needs to be done is more difficult than getting it done. It does not matter how talented a development team is, if they are not sent out to accomplish the right goals, they can't be expected to find success. That's why proper requirements are essential to produce high quality software.

The requirement can be obvious or hidden or unknown, expected or unexpected from client point of view. Requirement Gathering is one of the steps of requirement engineering process which includes: Feasibility requirement study,



software requirement specification and software requirement validation. This implies firstly the feasibility study analyzes whether the software product can be practically materialized terms of implementation, contribution of project organization cost constraints and as per values and objectives of the organization.

Today's s/w development methodologies are equipped with a combination of methods and technologies for business process engineering and requirements. However experienced has shown that the methods and technique do often disappointing results when being applied independent of each other. There are two main obstacles to effective alignment of business process and the information system (IS) that support them:

(i) The IS are not developed with correct understanding of the business they are supposed to support.

(ii) Business process are not linked to the system requirements and thus envolve independent from the IS.

Hence, unless the two process. i.e. business modelling and requirement gathering go side-by-side or aligned, the development of a software is heavily affected after math.

(b) Discuss the strength and weakness of Object Oriented and procedural programming with the help of point of sale system.

→ Object oriented programming is problem solving technique which uses classes and objects to create models based on the real world environment. An OOP application may use a collection of objects which will pass message when called upon to request a specific service or information. The main advantage of OOP programming is it makes it easy to maintain and modify existing code as new objects are created inheriting characteristics from existing one. This makes adjusting program much simpler. But the major disadvantage of OOP is increase in size of program, effort speed.

Similarly, procedural programming which at time has been referred to as time programming takes more top-down approach to programming. They takes an applications by solving problems from the top. to the code down to the bottom. They are faster

and size efficient but developer must edit every line of code that corresponds to the original change in code. For example: Let us take a point of sales system which comprises network operated by a main computer and linked to several checkout terminals with pos system.

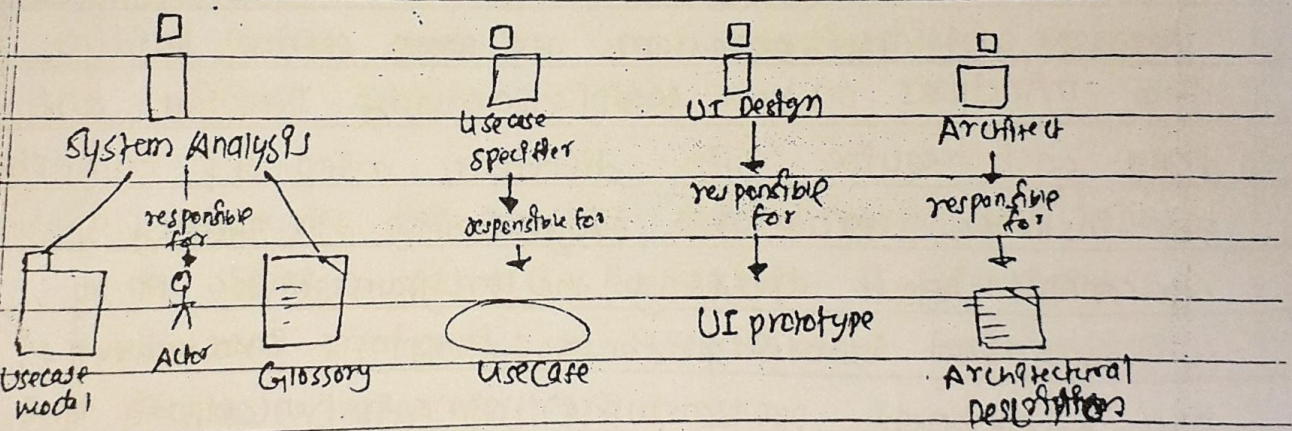
- One can analyze sales data, figure out how well the item on your shelves sell and adjust purchasing levels accordingly.
- One can maintain a sales history to help adjust buying decisions for seasonal purchasing trends.
- One can improve pricing accuracy by integrating bar-code scanner and credit card authorization ability with pos system.

Q 3(a) What are use cases? Describe workflow for capturing requirement as use cases, including the participating workers and their activities.

Sol) In software and system engineering, a use-case is a list of actions or events steps typically defining the interactions between a role (known in UML as actor) and a system to achieve a goal. The actor can be a human or other external

system within the unified process, five workflow come across the set of four phases: Requirements, analysis, Design, Implementation, and testing. Each workflow is a set of activities that varies project workers perform.

The primary activities of the requirement work are aimed at building the use case model, which captures the functional requirements of the system being defined - the model helps the project stake to reach agreement on the capabilities of the system and the conditions to which it must conform.



- declarative - which does not state the order in which operations execute.
- functional - which disallows side effect.
- Object oriented - which groups code together with state the code modifies.
- procedural - which groups code into functions and so on.

So design patterns are easier to understand, they are template of solution. some common design patterns are :-

- Factory method
- Abstract factory.

Factory method:

creates an object from a set of similar classes, on some parameter, usually a string. example is: creation of a message Object object in Java.

Q) What is the role of design pattern? How is design pattern important? Highlight the role of software development with respect to the design pattern?

Sol) In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem. A design pattern is not a finished design that can be transformed directly into code.

- declarative - which does not state the order in which operations execute.
- functional - which disallows side effect.
- Object oriented - which groups code together with state the code modifies.
- procedural - which groups code into functions and so on.

So design patterns are easier to understand, they are template of solution, some common design patterns are :-

- Factory method
- Abstract factory.

Factory method:

It creates an object from a set of similar classes, on some parameter, usually a string. Example is: creation of a Message Digest object in Java.

Q) What is the role of design pattern? How is design pattern important? Highlight the role of software development with respect to the design pattern?

Soln → In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem. A design pattern is not a finished design that can be transformed directly into code.

This is a common method of mathematical pattern analysis and such analysis is important for the following reasons:

- Understanding patterns allows examine to identify such pattern when they first appear.
- It follows someone to make educated guesses.
- Understanding patterns is in developing mental skill.

To understand the role of design patterns, lets face an example of an adaptor. Adaptor pattern is one among those 23 patterns described in the book "Design pattern" by gang of four. Adaptor provides a solution to the scenario in which a client and server need to interact with one another but cannot because their interfaces are incompatible.

To implement an adaptor, we create a custom class that has the interface provided by the server and defines the server operations in terms the client expect.

Hence, the role of design pattern in software development is very crucial to make timely and effective response to design issue and to build a better quality software. Design patterns have two major benefits:

- (a) They provide you with a way to solve issues related to software development

using a proven solution.

(b) They make communication between designers efficient. Software professionals can head when they refer the name of the pattern used to solve the particular issue when discussing system design.

4 (b) Write about structure and documentation of pattern.

Sol) Design pattern documentation is highly structure. The patterns are documented from template that identifies the information needed to understand the software problem and the solution in terms of the relationship between the classes and objects necessary to implement the solution. There's no uniformity in the design pattern template decision. i.e. the authors prefer different style for their pattern templates. Some want to be more expressive less structural while the rest prefer their pattern to be more precise and high gain instruction. An example of template from the authors of the first book "Design pattern" is as follow:

Term	Description
pattern-name	→ Describes the short expressive name
Intent	→ what the pattern does.
Also known as	→ List any synonym for the pattern
Applicability	→ list the situations where pattern is applicable.
Structure	→ set of diagrams of the classes and objects that depict pattern.
participants	→ the classes and object that participate in design pattern.
Collaborations	→ Describes how the participants collaborate to carry out their responsibility.

The documentation of design patterns describes about the context in which the pattern is used, therefore the faces within the context that the pattern seeks to resolve and the suggested solution.

5(a) Define design principle, design concept and design pattern. What are the dis-advantages of design patterns?

Sol'n → Software design principles represent a set of guidelines that help us to avoid having bad design. According to Robert Martin there are 3 important characteristics of a bad design that should be

avoided:

(i) Rigidity
(ii) Fragility
(iii) Immobility

Other major principles includes

* DRY (Don't Repeat Yourself)

⇒ Try to avoid duplicates. For instance, imagine you have copied and pasted blocks of code in different part in your system, then there are lots of same codes.

* Keep it simple.

Most systems works best if they are kept simple rather than making them complex, therefore, simplicity should be a key goal in design and unnecessary complexity should be avoided. Software design is the process of implementing software solutions to one of more sets of problems. One of the main components of a software design is the requirements analysis (SRA). SRA is a part of the software development process that lists specifications used in software engineering. The sets of fundamentals software design concepts includes the following:

(i) Abstraction

(ii) Architecture
(ii) patterns
(iv) Modularity
(v) Information hiding
(vi) Functional independence
(vii) Refinement
(viii) Refactoring

The desadvantages of design patterns are as follows:

- Design patterns may increase or decrease the understandability at a design and implementation. They can decrease by adding indirection or increasing the amount of code.
- It does not lead to direct code reuse.
- Complex in nature
- They are deceptively simple.
- They are validated by experience and decisions.
- Teams may suffer from pattern overload.
- Integrating patterns into a software development process is a human-intensive activity.

5 (b) What is software architecture? Why do we need it?

Sol:- Software architecture is the process of designing the global organization of a software system, including dividing software into subsystems, deciding how these will interact, and determining their interfaces. The term 'software architecture' is also applied to the documentation produced as a result of the process.

Software engineers discuss all aspect of systems design in terms of the architectural model. The architectural model will often consider the overall efficiency, reusability and maintainability of the system. There are four main reason why we need to develop an architectural model:

(i) To enable everyone to better understand the system.
(ii) To allow people to work on individual pieces of the system in isolation.
(iii) To prepare for extension of the system.
(iv) To facilitate reuse and reusability.

6 (a) Describe Service oriented architecture and design principles it helps to adhere.

Soln: A service-oriented architecture (SOA) is a style of software design where services are provided to the other components by application components through a communication protocol over a network. The basic principles of service-oriented architecture are independent of vendors, products and technologies. A service is a discrete unit of functionality that can be accessed remotely and acted upon and updated independently, such as retrieving a credit card statement online.

The principles of SOA are:

1. Explicit Boundaries
2. Shared contract and schema, not class.
3. policy-driven.
4. Autonomous
5. Wire formats, not programming languages APIs
6. Document-oriented.
7. Loosely couples
8. Standards-compliant
9. Vendor independent
10. Metadata-driven.

Date
Page

6(b). Discuss about client server and Distributed architecture. Are they similar? Justify. What is the role of distributed architecture in today's world of automation? Explain.

Sol? Client-server architecture is a network architecture in which each computer or process on the network is either a client or a server. Servers are powerful computers or processes dedicated to managing disk drives (file servers), printers (print server) or network traffic (network servers).

On the other hand, Distributed system architecture (DSA) is the ideal solution for integrating processes where those are multiple units, control rooms or geographically distributed locations. In distributed architecture components are hosted on different platform.

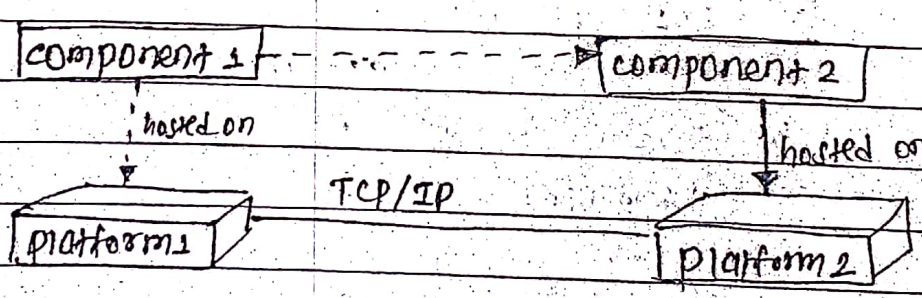


Figure: Distributed system Architecture

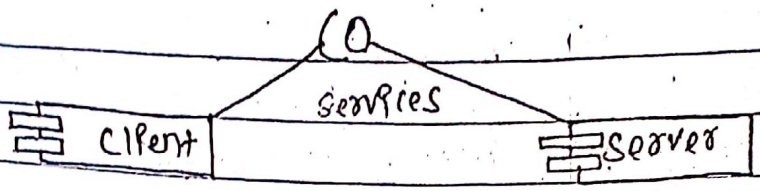


Figure: 1

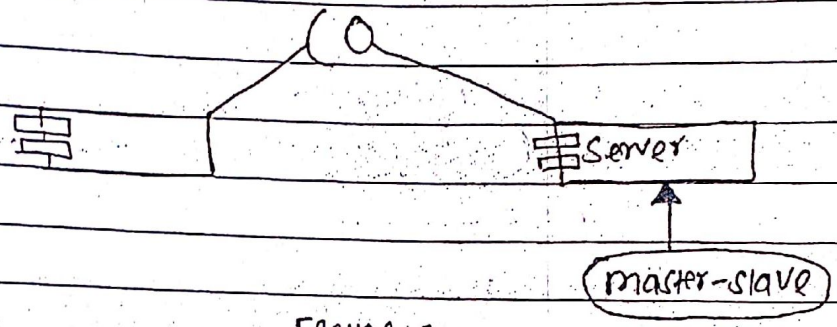


Figure: 2

7 (a) Player-Role design pattern

Sol) "Studying software design patterns is an effective way to learn from the experience of others".

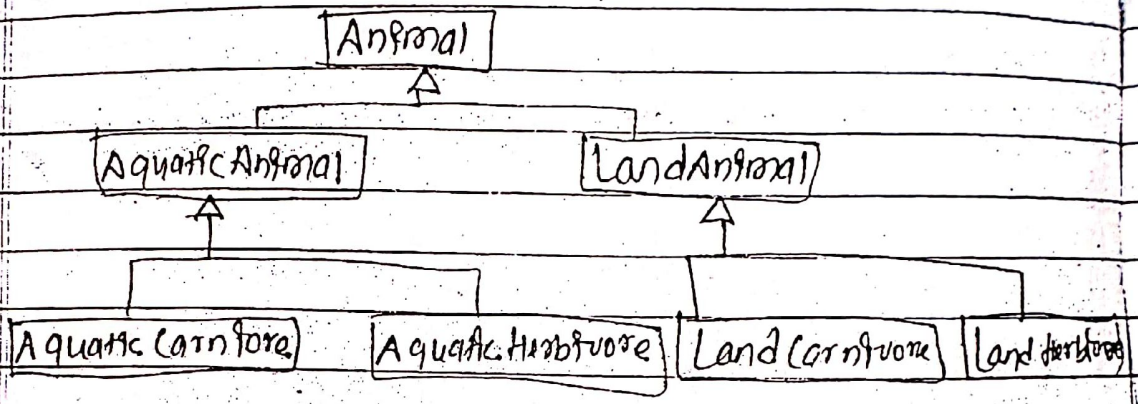
Context:

- an object may play different roles in different contexts of an application.
- an object may need to change roles throughout the course of the application.
- an object may not need to play one or both roles simultaneously.

Issues:

- It is desirable to improve encapsulation by keeping information related to only one role in one class.
- making two classes to describe the same object is not good OO design.

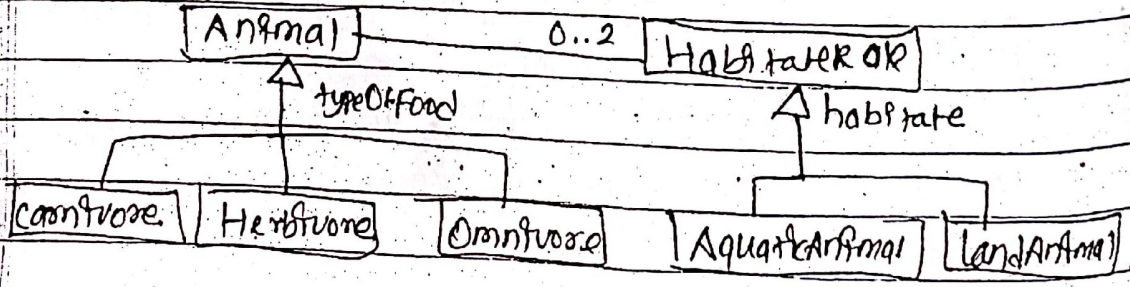
Example of class design of animals



problem:

Two or more objects are required if the animal needs to play different roles, like aquatic and Land Carnivore, which is not good OO design.

Player-role solution to Animals.



Solutions:

An animal can be:
- Carnivore
- Herbivore
- Omnivore

An animal can have:
- No habitat role
- an aquatic habit role
- a land habitat role
- an aquatic and land habitat roles.

7(b) Sol^n

Reuse Vs Reusability

Code reuse is called software reuse. Software reuse is the process of creating new software systems from existing software components. Reuse has an enormous impact on productivity. The following elements of software reused are, Software specifications, designs, tests cases, data, prototypes, plans, documentation, frameworks and templates.

Reusability is the degree to which the artifacts can be reused. The ability to use all or the greater part of the same programming code or system design in another application. Reusability is the segment of source code that can be used to add new functionalities with slight

Or localizes code modifications when a change in implementation is required.
Reusability is the extent to which a software component is able to be reused.
Reusable modules and classes increase the prior testing, reduce implementation time and use has to have appropriate design and coding standards. The types of software Reuse:

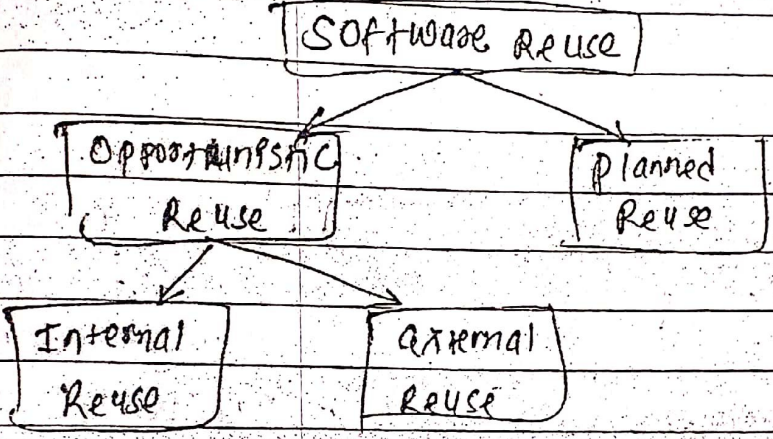


Fig: Types of Reuse.