object Oriented System Design

KCS - 054

UNIT - 1

Introduction

Object Orientation

In an object oriented system, all data is represented as discrete objects with which the user and other objects may interact. Each object contains data as well as information about the executable file needed to interpret that data.

Object identity

Object identity is a fundamental object orientation concept. With object identity, objects can contain or refer to other objects. Identity is a property of an object that distinguishes the object from all other objects in the application.

Encapsulation

Encapsulation refers to the bundling of data, along with the methods that operate on that data, into a single unit. Many programming languages use encapsulation frequently in the form of classes.

Information Hiding

Information Hiding is the primary criteria of system modularization and should be concerned with hiding the critical decisions of OOPs designing. Information hiding isolates the end users from the

requirement of intimating knowledge of the design for the usage of a module.

Polymorphism

Polymorphism is one of the core concepts of object oriented programming and describes situations in which something occurs in several different forms.

Generosity

It is a process that allows a function or a class to work with different data types.

Importance of Modelling

Modeling gives graphical representation of system to be built. Modeling contributes to a successful software organization. Modeling is a proven and well accepted engineering technique.

Principles of Modelling

→ The choice of model is important.
→ Every model may be expressed at different levels of precision.
→ The best models are connected to reality.
→ No single model is sufficient.

Object Oriented Modelling

OOM is the construction of objects using a collection of objects that contain stored values of the instance variables found within an object. Unlike models that are record oriented, object oriented values are solely objects.

Advantages of OOPs Over Procedural Programming

A programming paradigm that is completely based on objects is known as object oriented programming language. object oriented programming is more secure than procedural programming languages, because of the level of abstraction or we can say data hiding property.

→ OOPs makes development and maintenance easier.

→ OOPs provides data hiding.

→ OOPs provide ability to stimulate real world event much more effectively.

→ OOPs is faster and easier to execute.

→ OOPs provides a clear structure for the programs.

→ OOPs helps to keep the Code DRY "Don't Repeat Yourself".

→ OOPs makes it possible to create full reusable applications with less Code and shorter development time.

→ Modularity for easier troubleshooting.

→ OOPs gives flexibility through Polymorphism.

→ OOPs gives better productivity.

class and object

The very first and fundamental thing of the OOPs Concept is class and object. Everything in the world as an object and those objects are comes from a class.

Examples :-

→ People are object and they belong from HUMAN class.

→ Dogs and Cats are objects they belong to the ANIMAL class.

→ Samsung, Nokia mobiles are objects they belong to MOBILE class.

→ BMW, Audi Cars are objects they belong from CAR class.

So, we can say in one word that class is the blueprint or the design of objects or instances.

All Basic Concepts of OOPs

→ class
→ objects
→ Encapsulation
→ Polymorphism
→ Inheritance
→ Abstraction

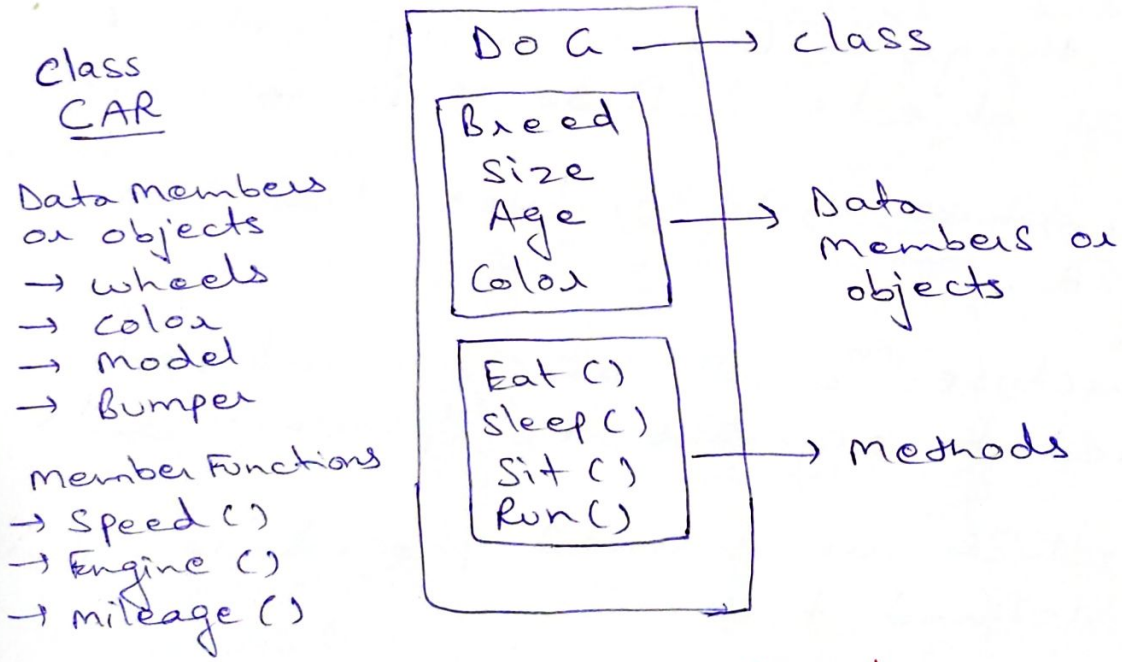
Object → An object can be defined as a data field that has unique attributes and behaviour.

class → class is a blueprint or a set of instructi-ons to build a specific type of object. It is a basic concept of object oriented programming which revolve around the real life entities.

Syntax of class

```
ClassName ObjectName;
```

object → An object in OOPs, ~~is~~ a object is an instance of a class. An object in OOPs is nothing but a self contained component which consists of methods and properties to make a particular type of data useful.



characteristics of an object

All individual objects possess three basic characteristics → identity, state and behaviour.

1) Identity means that each object has its own object identifier and can be differentiated from all other objects. Each object's name, or identity, is unique and distinct from other objects.

2) State refers to the properties of an object For example, values of variables in the object contain data that can be added, changed or deleted.

3) Behavior refers to actions that the object can take. For example, one object can respond to another object to carry out Software functions.

Some of the things in Programming that can be defined as objects include the following:-

→ variables, which hold values that can be changed.

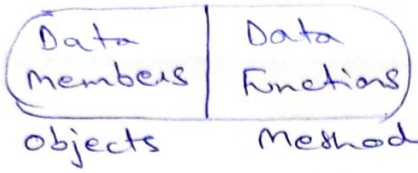
→ Data structures which are specialized format used to organise and process data.

→ Functions which are named procedures that perform a defined task.

→ Methods which are programmed procedures that are defined as components of a parent class and are included in any instance of that class.

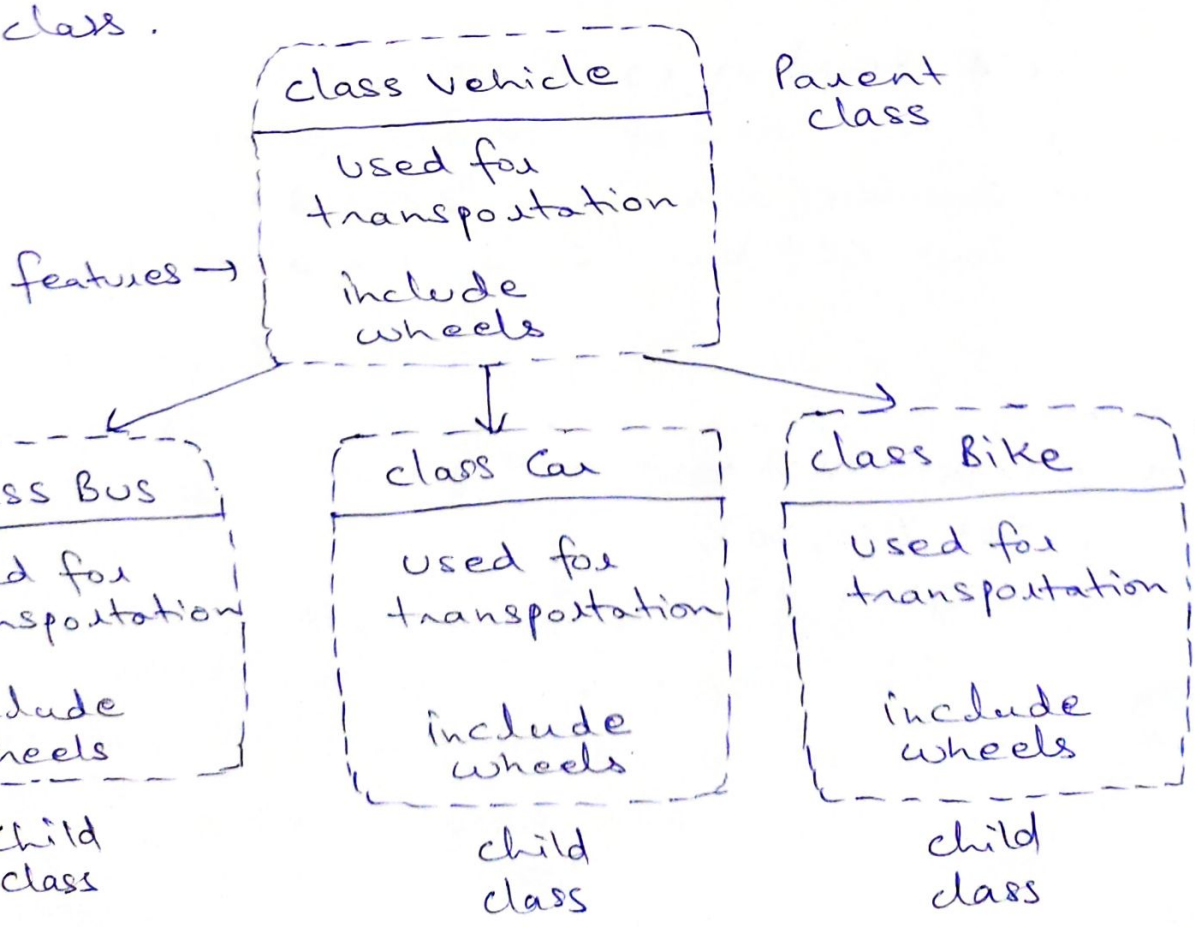
Encapsulation

The wrapping up of data and functions together in a single unit is known as encapsulation. It can be achieved by making the data members scope private and the member functions (methods) scope public to access these data members. Encapsulation makes the data non-accessible to the outside world.



Inheritance

Inheritance is the process in which two classes have an is-a-relationship among each other and the objects of one class acquire properties and features of the other class. The class which inherits the features is known as the child class, and the class whose features it inherited is called the parent class.



Abstraction

Data abstraction is one of the most essential and important feature of object oriented programming. Abstraction means displaying only essential information and hiding the implementation details or background details.

Program

```
# include <iostream>
using namespace std;
class implementAbstraction
{
   private
      int a, b;
   Public
     // method to set values of private members
     void set (int x, int y)
     {
        a = x;
        b = y;
     }
     void display ()
     {
       Cout << "a = " << a << endl;
       Cout << " b = " << b << endl;
     }
main
implementAbstraction obj;
obj. Set (10, 20);
obj. display ();
return 0;
```

Output

```
a = 10
b = 20
```

You can see in the above program we are not allowed to access the variables a and b directly, however one can call the function Set() to set the values in a and b and the function display() to display the values of a and b.

Advantages of Abstraction

→ Helps the user to avoid writing the low level code.

→ Avoids code duplication and increases reusability.

→ Can change internal implementation of class independently without affecting the user.

→ Helps to increase security of an application or program as only important details are provided to the user.

Abstraction Using access Specifiers

Access Specifiers are the main pillars of implementing abstraction in C++. We can use access specifiers to enforce restrictions on class members. For example :-

→ members declared as public in a class, can be accessed from anywhere in the program.

→ members declared as private in a class, can be accessed only from within the class. They are not allowed to be accessed from any part of code outside the class.

Polymorphism

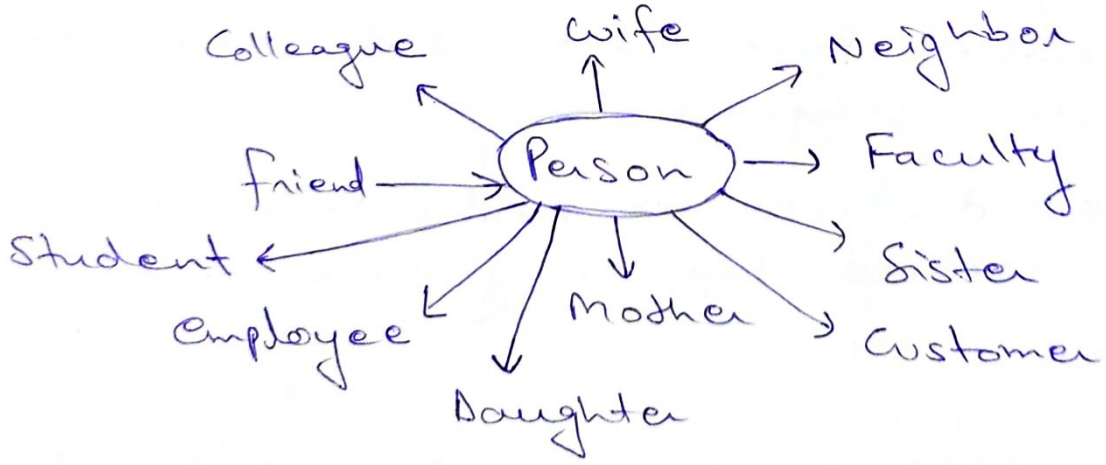
Polymorphism is the ability of any data to be processed in more than one form. Poly means many and morphism means types. Polymorphism is one of the most important concept of OOPs. The most common use of Polymorphism in OOPs occurs when a parent class reference is used to refer to a child class object. It is the ability of an object or reference to take many forms in different instances.

Polymorphism is the combination of two greek words. one is poly means many and other is morphism means forms.

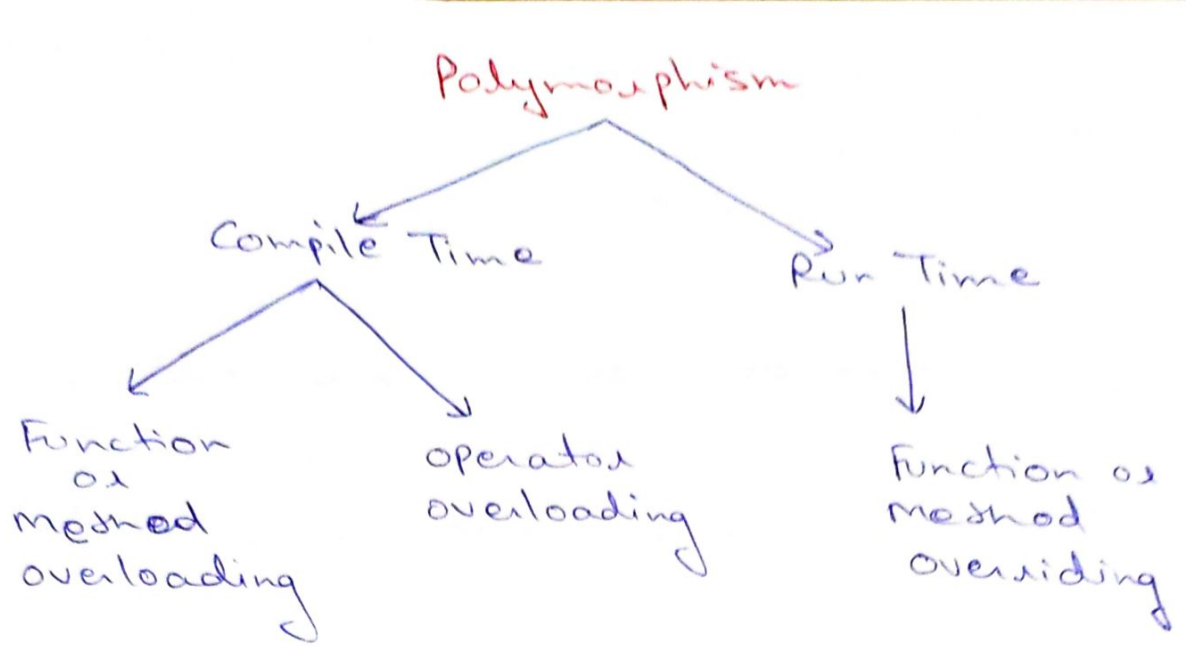
Poly + morphism
↓ ↓
many forms

Something occurs in several different forms

→ whose meaning is same object having different behaviours.



Void person (Teacher)
Void person (customer)
Void person (student)
Void Person (friend)



Compile Time Polymorphism

A Polymorphism which is exist at the time of Compilation is Called Compile time or early binding or static polymorphism.

Function overloading

When there are multiple functions with the same name but different parameters, then the functions are said to be overloaded. Functions can be overloaded by changing the number of arguments and or changing the type of arguments.

C++ Program of function overloading

```
#include<bits/stdc++.h>
using namespace std;
class Geeks
{
  public:
// function with 1 int parameter
void function(int x)
  {  cout<<"value of x is "<<x<<endl;
  }
```

```
// function with same name but 1 double
   Parameter
void function (double x)
{
 cout << " value of x is " << x << end1;
}
// function with same name and 2 int parameter
void function (int x, int y)
{
  cout << " value of x and ~~valu~~ y is " << x <<",
          << y << end1;
}
};
int main () {
Geeks obj1;
// which function is called will depend on the
   Parameters passed.
   The first function is called
   obj1. function (7);

// The second function is called
   obj1. function (9.132);

// The third function is called
   obj1. function (85, 64);

   return 0;
}
```

output

value of x is 7
value of y is 9.132
value of x and y
is 85, 64

operator overloading

C++ also provides the option to overload operators. for example, we can make use of the addition operator (+) for string class to concatenate two strings. we know that the task of this operator is to add two operands. So a single operator '+', when placed between integer operands, adds them and when placed between string operands, concatenates them.

C++ Program for operator overloading

```
#include <iostream>
using namespace std;
class complex {
private:
   int real, imag;

public:
   complex (int r = 0, int i = 0) { real = r; imag=i;}

// This is automatically called when '+' is used
// with between two complex objects

Complex operator + (Complex const & obj) {
        Complex res;
        res.real = real + obj.real;
        res.imag = imag + obj.imag;
        return res; }

void print() { cout << real << " + i" << imag <<
                                        end1; }

};

int main()
```

```
Complex C1 (10,5), C2 (2,4);
Complex C3 = C1 + C2; // An example Call to
                      " operator + "

C3.print ();
}
```

output

12 + 19

In the above example, the operator '+' is overloaded. Usually this operator is used to add two numbers (integers or floating point numbers), but here the operator is made to perform the addition of two imaginary or Complex numbers.

Run Time Polymorphism

A Polymorphism which exists at the time of execution of program is called run time Polymorphism.

Function overriding

→ whenever we writing function or method in Super and sub classes in such a way that method name and parameter must be Same Called method overriding.

→ It occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.

C++ Program for Function overriding

```
#include <bits/stdc++.h>
using namespace std;

class base
{
public:
    virtual void print ()
    { cout << "print base class" << endl; }

    void show ()
    { cout << "show base class" << endl; }
};

class derived : public base
{
public:
void print () // print() is already virtual function
                in derived class, we could also
                declared as virtual void print()
                explicitly
  { cout << "print derived class" << endl; }

void show ()
  { cout << "show derived class" << endl; }

};
// main function
int main ()
{
  base * bptr;
  derived d;
  bptr = &d;
// virtual function, binded at runtime (Runtime
                                    Polymorphism)
  bptr->print ();
```

```
// Non virtual function, binded at compile tim
bptx->show();
return 0;
}
```

output

```
Print desired class
Show base class
```

object oriented Technology

object oriented technology is a software design model in which objects contain both data and the instructions that work on the data. It is increasingly deployed in distributed Computing.

Pros and Cons of object oriented Technology

Pros :-

1) It allows for parallel development.
2) The modular classes are often reusable.
3) The coding is easier to maintain.
4) It is easy to maintain and modify.
5) It maintains the security of data.
6) Low cost development.

Cons :-

1) It can be inefficient.
2) It can be too scalable.
3) It can cause duplication.
4) It is slower than other programs.
5) It is not suitable for some sorts of problems.
6) It takes time to get used to it.

Introduction to UML

UML is a general purpose modelling language. The main aim of UML is to define a standard way to visualize the way a system has been designed. It is quite similar to blueprints used in other fields of engineering.

Conceptual Model of UML

A conceptual model can be defined as a model which is made of concepts and their relation ships. A conceptual model is the first step before drawing a UML diagram. It helps to understand the entities in the real world and how they interact with each other.

Architecture of the UML

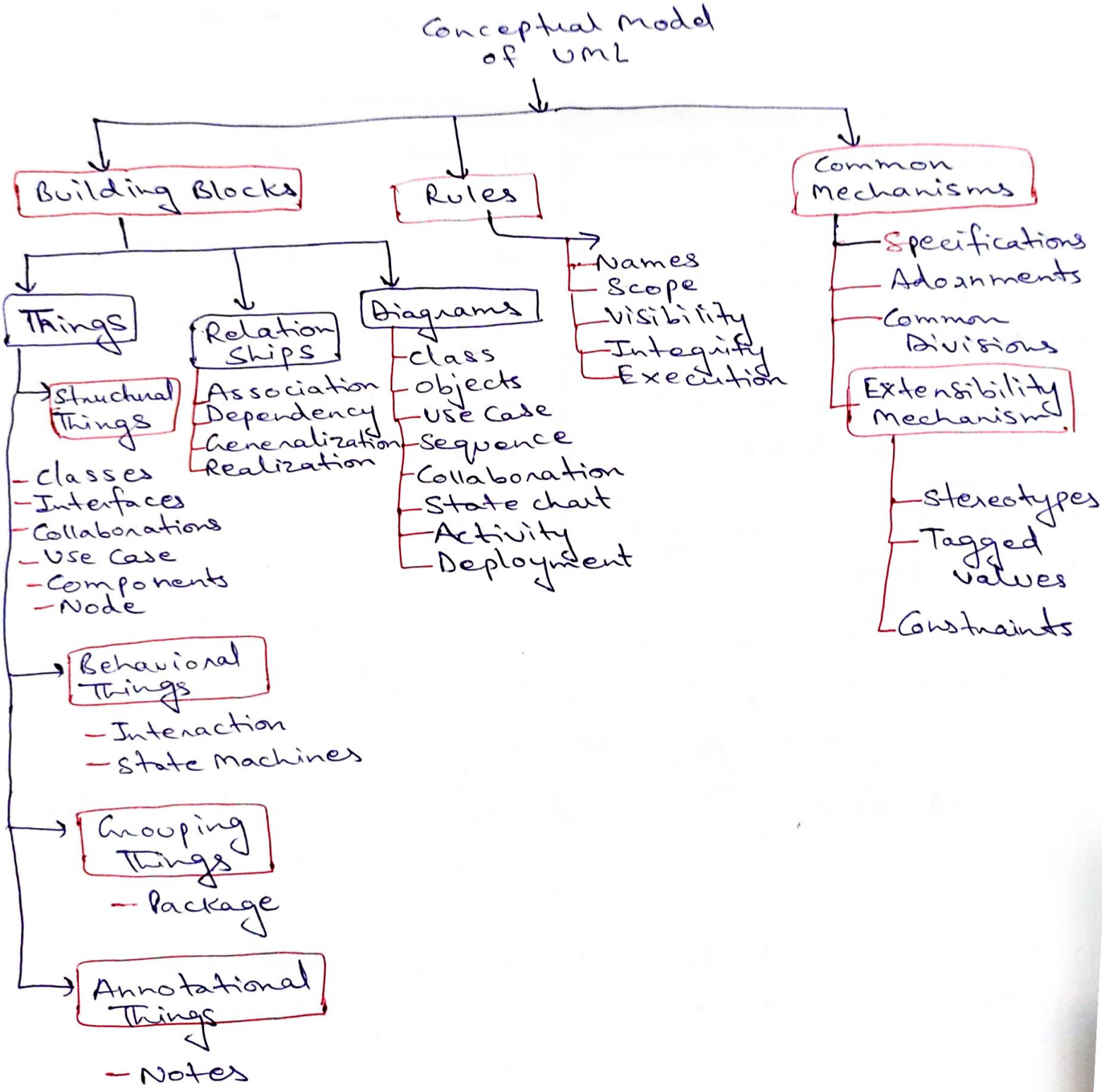
UML provides class diagram, object diagram to support this. Implementation defines the compone -nts assembled together to make a complete physical system. UML component diagram is used to support the implementation perspective. Process defines the flow of the system.

Features of object oriented Paradigms

- → Inheritance
- → Encapsulation
- → Abstraction
- → Polymorphism
- → Method overriding
- → method overloading
- → objects
- → classes
- → Constructors
- → Destructors

Conceptual Model of UML

A Conceptual model can be defined as a model which is made of concepts and their relationships. A Conceptual model is the first step before drawing a UML diagram. It helps to understand the entities in the real world and how they interact with each other.



Difference Between Arrays and Linked List

Arrays	Linked List
1) An array is a collection of elements of a similar data type.	1) linked list is an ordered collection of elements of the same type in which each element is connected to the next using pointers.
2) Arrays are stored in contiguous location.	2) linked lists are not stored in contiguous location.
3) Fixed in size.	3) Dynamic in size.
4) memory is allocated at compile time.	4) memory is allocated at run time.
5) Uses less memory than linked lists.	5) Uses more memory because it stores both data and the address of next node.
6) Elements can be accessed easily.	6) Elements accessing requires the traversal of whole linked list.
7) Insertion and deletion operation takes time.	7) Insertion and deletion operation is faster.

Unified Modeling language

UML is a standard visual language for describing and modelling software blueprints. The UML is more than just a graphical language. Stated formally, the UML is for visualizing, specifying, constructing and documenting.
The artifacts of a software intensive system.
The three aspects of UML :

1) language :-
→ It enables us to communicate about a subject which includes the requirements and the system.
→ It is difficult to communicate and collaborate for a team to successfully develop a system without a language.

2) Model
→ It is a representation of a system.
→ It captures a set of ideas (known as abstractions) about its subject.

3) Unified
→ To bring together the information systems and technology industry's best engineering practices.
→ these practices involve applying techniques that allow us to successfully develop systems.