

**I BCA
II SEMESTER
OBJECT
ORIENTED
ANALYSIS
AND
DESIGN**

UNIT I:

The Object Model-The Evolution of the Object Model: The generations of programming languages, The topology of Programming languages. Foundations of the Object Model: Object Oriented Analysis, Object Oriented design, Object Oriented Programming. Elements of the Object Model: Programming Paradigm(programming style), The Major and Minor Elements of the Object Models, Abstraction, Encapsulation, Modularity, Hierarchy(single inheritance, multiple inheritance, Aggregation), Static and Dynamic Typing, Concurrency, Persistence.

UNIT II:

Classes and Objects-The Nature of an Object: What is and what is not an Object, State, Behavior, and Identity. Relationships among Objects: Links, Aggregation. The Nature of a Class: Interface and Implementation, Class Lifecycle. Relationships among Classes: Association: Semantic Dependencies, Multiplicity, Inheritance, Polymorphism, Aggregation, Dependencies. The Interplay of Classes and Objects: Relationship between Classes and Objects, On Building Quality Classes and Objects: Measuring the Quality of an Abstraction

UNIT III:

Classification-The Importance of Proper Classification: The Difficulty of Classification, The Incremental and Iterative Nature of Classification. Identifying classes and Objects: Classical and Modern Approaches. Object Oriented Analysis: Classical Approaches, Behavior Analysis, Domain Analysis, Use Case Analysis, CRC Cards, Informal English Description, Structured Analysis. Key Abstractions and Mechanisms: Identifying Key Abstractions: Refining Key Abstractions, Naming Key Abstractions. Identifying Mechanisms.

UNIT IV:

The Unified Modeling Language: Diagram Taxonomy: Structure Diagrams, Behavior Diagrams. The Use of Diagrams in Practice: Conceptual, Logical and Physical Models, The Role of Tools. The Syntax and Semantics of the UML: The Package Diagrams, Component Diagrams, Deployment Diagrams, Use Case Diagrams.

UNIT V:

The Syntax and Semantics of the UML: Activity Diagrams, Class Diagrams, Sequence Diagrams, Interaction Diagrams, Composite Structure Diagrams, State Machine Diagrams, Timing Diagrams, Object Diagrams, Communication Diagrams.

TEXT BOOK:

1. Object-Oriented Analysis and Design with Applications, 3rd Edition, By: Robert A. Maksimchuk, Bobbi J. Young, Grady Booch, Jim Conallen, Michael W. Engel, Kelli A. Houston,
Pearson education.

REFERENCE BOOKS:

1. James Rumbaugh, Jacobson and Booch, Unified Modeling Language reference

UNIT I The Object Model

Q) Object Model:

- Object model will imagine the software application elements in terms of objects.
- The object model identifies the classes in the system and their relationship, as well as their attributes and operations.
- It represents the static structure of the system.
- The object model is represented graphically by a class diagram.

Step for object modeling

1. Read carefully, the problem statement.
2. Locate the object classes by underlining nouns.
3. Remove unnecessary and incorrect classes.
4. Prepare a data dictionary.
5. Locate associations between object classes.
6. Remove unnecessary and incorrect attributes.
7. Use inheritance to share common structure.
8. Traverse access paths to identify deficiency.
9. Remove unnecessary and incorrect associations.
10. Locate attributes of the object classes.

Q) The Evolution of the Object Model:

- The Evolution of the Object Model in software engineering focused from programming-in-the-small to programming-in-the-large.
- The OOP (Object Oriented Programming) approach is most commonly used approach now a days.
- OOP is being used for designing large and complex applications. Before OOP many programming approaches existed which had many drawbacks.
- Initially for designing small and simple programs, the machine language was used.
- Next came the Assembly Language which was used for designing larger programs
- Both machine and Assembly languages are machine dependent.
- Next came Procedural Programming Approach which enabled us to write larger and hundred lines of code.
- Then in 1970, a new programming approach called Structured Programming Approach was developed for designing medium sized programs.

- In 1980's the size of programs kept increasing so a new approach known as OOP was invented.

- 1) Monolithic Programming Approach
- 2) Procedural Programming Approach
- 3) Structured Programming Approach
- 4) Object Oriented Programming Approach

Monolithic Programming Approach:

- In this approach, the program consists of sequence of statements that modify data. All the statements of the program are Global throughout the whole program
- The program control is achieved through the use of jumps i.e. goto statements.
- In this approach, code is duplicated each time because there is no support for the function. Data is not fully protected as it can be accessed from any portion of the program.
- So this approach is useful for designing small and simple programs. The programming languages like ASSEMBLY and BASIC follow this approach.

Procedural Programming Approach:

- This approach is top down approach. In this approach, a program is divided into functions that perform a specific task.
- Data is global and all the functions can access the global data. Program flow control is achieved through function calls and goto statements.
- This approach avoids repetition of code which is the main drawback of Monolithic Approach.
- The basic drawback of Procedural Programming Approach is that data is not secured because data is global and can be accessed by any function.
- This approach is mainly used for medium sized applications. The programming languages: FORTRAN and COBOL follow this approach.

Structured Programming Approach:

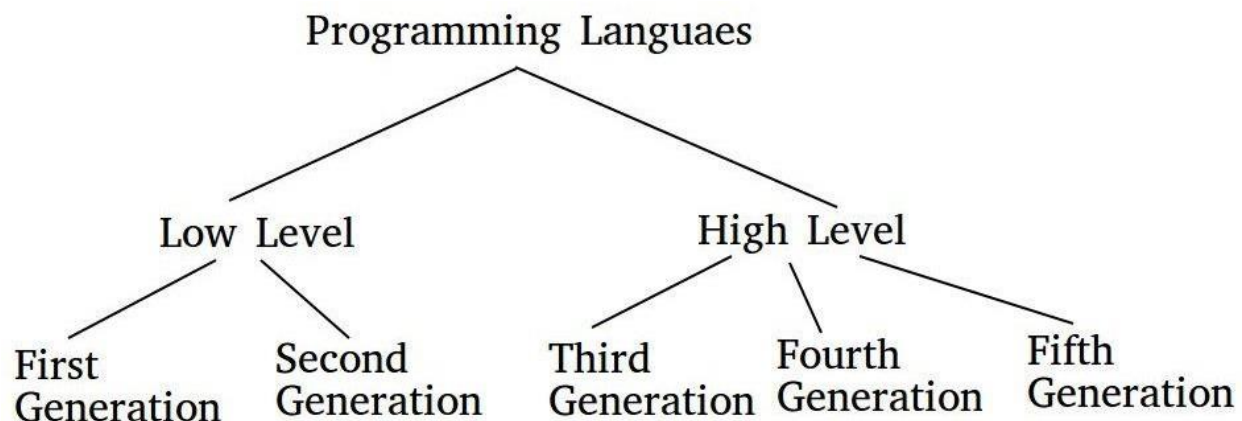
- The basic principal of structured programming approach is to divide a program in functions and modules
- The use of modules and functions makes the program more comprehensible (understandable).
- It helps to write cleaner code and helps to maintain control over each function. This approach gives importance to functions rather than data.
- It focuses on the development of large software applications. The programming languages: PASCAL and C follow this approach.

Object Oriented Programming Approach:

- The OOP approach came into existence to remove the drawback of conventional approaches.
- The basic principal of the OOP approach is to combine both data and functions so that both can operate into a single unit. Such a unit is called an Object.
- This approach secures data also. Now a days this approach is used mostly in applications. The programming languages: C++ and JAVA follow this approach. Using this approach we can write any lengthy code.

Q) Generations of programming language:

- Programming languages have been developed over the year in a phased manner.
 - Each phase of developed has made the programming language more user-friendly, easier to use and more powerful.
 - Each phase of improved made in the development of the programming languages can be referred to as a generation.
 - The programming language in terms of their performance reliability and robustness can be grouped into five **different generations**,
1. First generation languages (1GL)
 2. Second generation languages (2GL)
 3. Third generation languages (3GL)
 4. Fourth generation languages (4GL)
 5. Fifth generation languages (5GL)



1. First Generation Language (Machine language)

- A first-generation programming language (1GL) is a machine-level programming language.
- machine language is a collection of binary digits or bits that the computer reads and interprets.
- Machine language is the only language a computer is capable of understanding.
- In the machine language, a programmer only deals with a binary number.
- **Ex:** 1,0

Advantages of first generation language

- It had high speed
- Translators were not used

Disadvantages:

- Machine dependent
- Complex
- Error prone(was full of errors)
- Tedious(was time taking)

2. Second Generation language (Assembly Language)

- The second generation programming language also belongs to the category of low-level-programming language.
- In the second generation assembly language was used and developed in 1950s and its main developer was ibm.
- The assembly language contains some human-readable commands such as mov, add, sub, etc. mnemonic codes
- Since assembly language instructions are written in English words like mov, add, sub, so it is easier to write and understand.
- A computer can only understand the machine-level instructions, so we require a translator that converts the assembly code into machine code. The translator used for translating the code is known as an assembler.

Ex: ADD 12,8

Advantages of second generation language

- Easy to use and more understandable than machine language
- Less error prone than machine language
- More control on hardware
- Efficient than machine language

Disadvantages:

- It was machine dependent
- Harder to learn
- Slow development time
- No support for modern software technology

3. Third Generation languages (High-Level Languages)

- High-level languages allow programmers to write instructions in a language that is easier to understand than low-level languages.
- The high-level languages are considered as high-level because they are closer to human languages than machine-level languages.

- When writing a program in a high-level language, then the whole attention needs to be paid to the logic of the problem.
- A compiler is required to translate a high-level language into a low-level language.
- **Examples:** FORTRAN, ALGOL, COBOL, C++, C

Advantages

- Readability
- Easy debugging
- Easier to maintain
- Low development cost

Disadvantages:

- Poor control on hardware
- Less efficient

4. Fourth generation language (Very High-level Languages)

- 4GL or fourth-generation language is designed to be closer to natural language than a 3GL language.
- These are languages that consist of statements that are similar to statements in the human language. These are used mainly in database programming and scripting.
- Languages for accessing databases are often described as 4GLs.
- The fourth generation programming languages were designed and developed to reduce the time, cost and effort needed to develop different types of software applications.
- **Examples:** *Perl, Python, Ruby, SQL, MatLab*

Advantages:

1. High level languages are programmer friendly. They are easy to write, debug and maintain.
2. It provide higher level of abstraction from machine languages.
3. It is machine independent language.
4. Easy to learn.
5. Less error prone, easy to find and debug errors.

Disadvantages:

1. It takes additional translation times to translate the source to machine code.
2. High level programs are comparatively slower than low level programs.
3. Compared to low level programs, they are generally less memory efficient.
4. Cannot communicate directly with the hardware.

5. Fifth generation language (Artificial Intelligence Language)

- These are the programming languages that have visual tools to develop a program.
- **5GL** or fifth-generation language is programming that uses a visual or graphical development interface to create source language that is usually compiled with a 3GL or 4GL language compiler.
- The major fields in which the fifth-generation programming language are employed are Artificial Intelligence and Artificial Neural Networks.
- **Examples:** mercury, prolog, OPS5

Advantages of fifth generation languages

- These languages can be used to query the database in a fast and efficient manner.
- In this generation of language, the user can communicate with the computer system in a simple and an easy manner.
- Decision making machines can be developed
- System automation, which can reduce the efforts of programmer

Disadvantages:

- Programs continue to get more complicated and slowly loose relation with the original algorithm
- More resources required

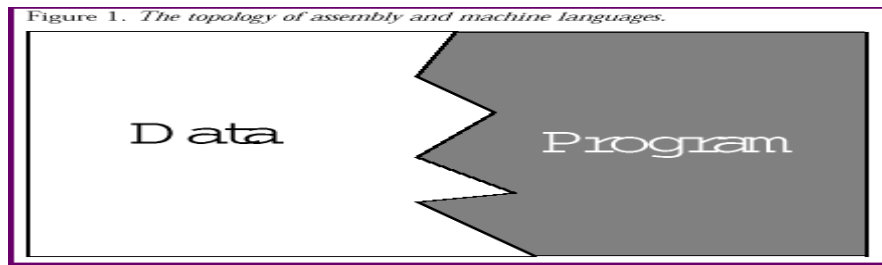
Q) The topology of Programming languages:

- Topology means basic physical building blocks of the language & how those parts can be connected.
- In this diagrams Arrows indicate dependency of subprograms on various data.
- In this Error in one part of program effect across the rest of system.

Topology of Assembly and Machine Languages:

Early languages (pre-FORTRAN and pre-COBOL) had little distinction between programs and data .

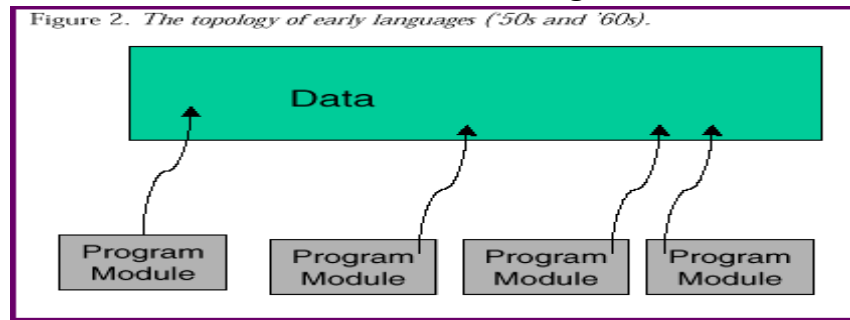
- The data and program co-existed.
- In this the programs were complex, difficult to debug, and almost impossible to modify.



Topology of Early Languages

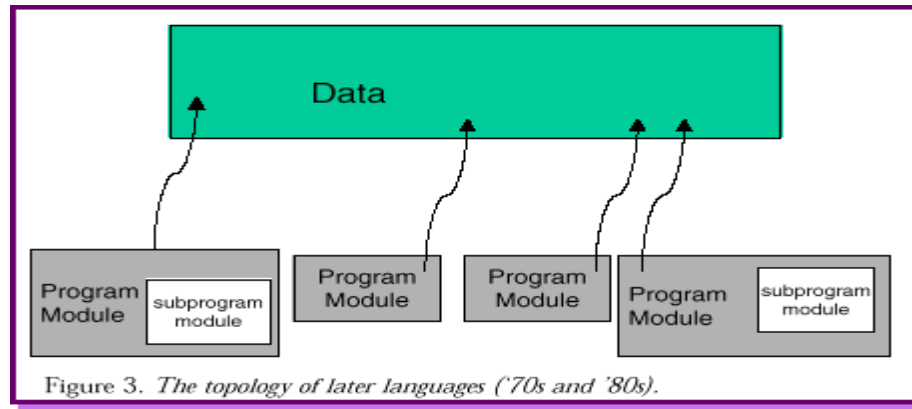
The first programming languages widely used, such as the early version of FORTRAN and COBOL, had a clear separation between the data and the program.

- These languages had a global data structure, but permitted modularization of program structure.
- Typically, there was only single-level modularization of the program (see Figure 2).
- While this separation of data and program was a good thing, all program segments were at a single level, and typically referenced each other in very complex ways.
- In this each module had unlimited access to all data because the data was global to all modules.
 - Global data is bad — it makes maintenance extremely difficult, since it is hard to determine which module is using the data.



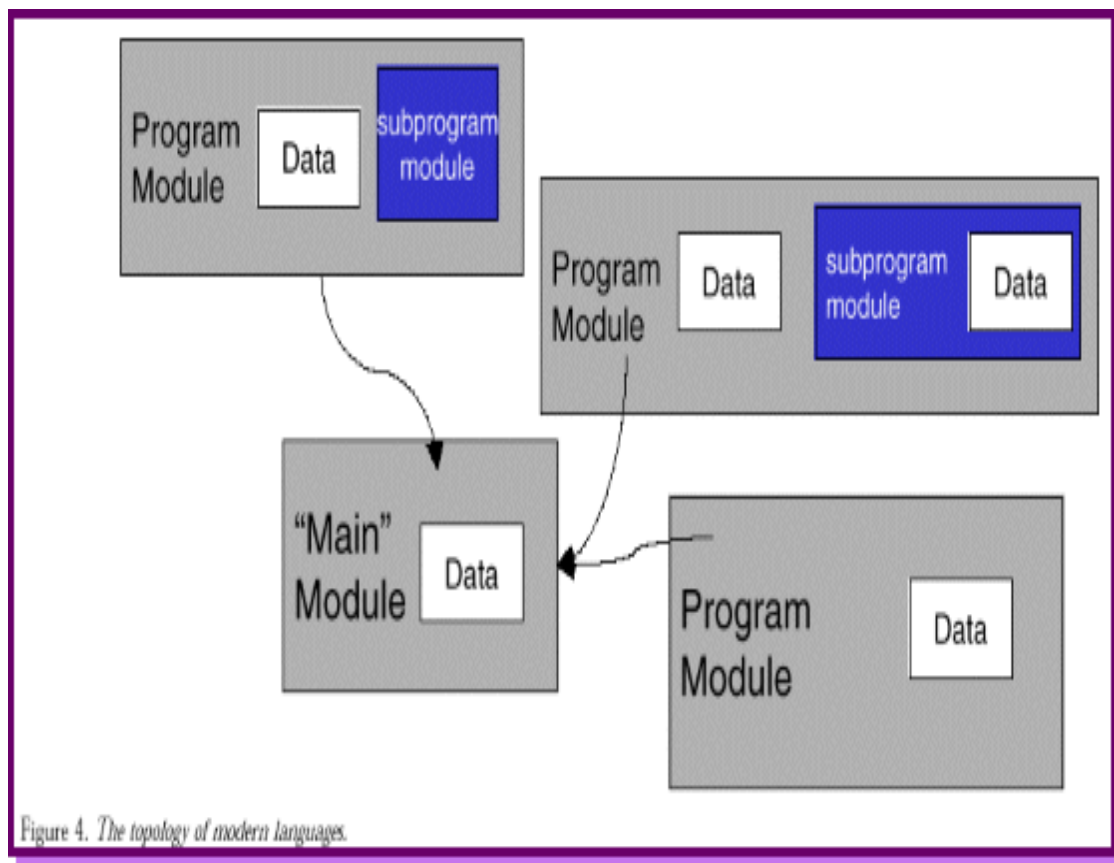
Topology of Later-Generation Languages:

- In this modules appeared at this time. The modules include data and functions.
- The functions are grouped by modules, and the data in each module does not interfere with each other.
- Despite the concept of modules, modules are rarely regarded as an important abstract mechanism. In practice, they are only used to group subprograms that are most likely to change at the same time.



Topology of Modern Languages:

- In this Languages it permit abstractions in both the data and the program units.
- In C++ and Java, classes can define method access, such as public and private.
- These modifications to the way we access data allow us to create powerful abstractions, and then control the way the data is accessed.
- We can even control who can access the data, and limit access to certain program modules.



Q) What is OOAD

Object-oriented analysis and design (OOAD) is a technical approach for analyzing and designing an application, system, or business by applying object-oriented programming, as well as using visual modeling throughout the software development process to guide stakeholder communication and product quality.

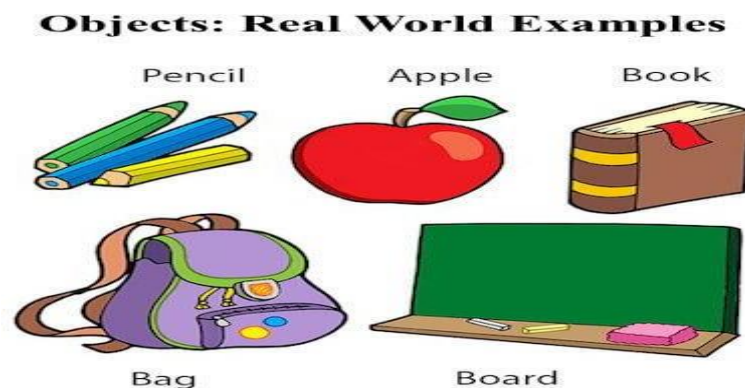
Or

Object-oriented analysis and design (OOAD) is a software engineering approach used to developing software system based on a group of interacting objects.

OOAD in modern software engineering is typically conducted in an iterative and incremental way. The outputs of OOAD activities are analysis models (for OOA) and design models (for OOD) respectively.

Object:

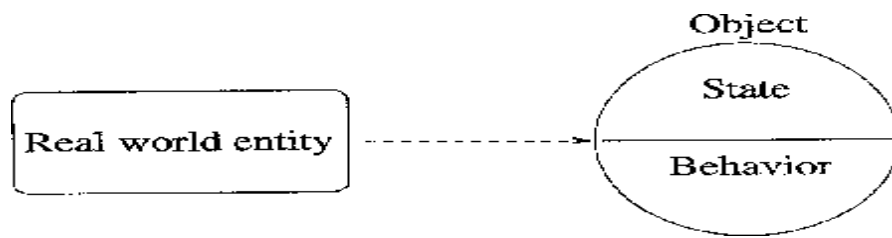
- Object is a real world entity that has state and behavior is known as an object
e.g. chair, bike, marker, pen, table, car, etc.



It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

- Software objects are modeled after real-world objects in that they too have state and behavior.
- A software object maintains its state in one or more variables ♦ A variable is an item of data named by an identifier.
- A software object implements its behavior with methods ♦ A method is a function (subroutine) associated with an object.
- An object is a single unit having both data variables and the functions that operate on that data. For example, in object oriented programming language like C++, the data and functions are bundled together as a self contained unit called an object.

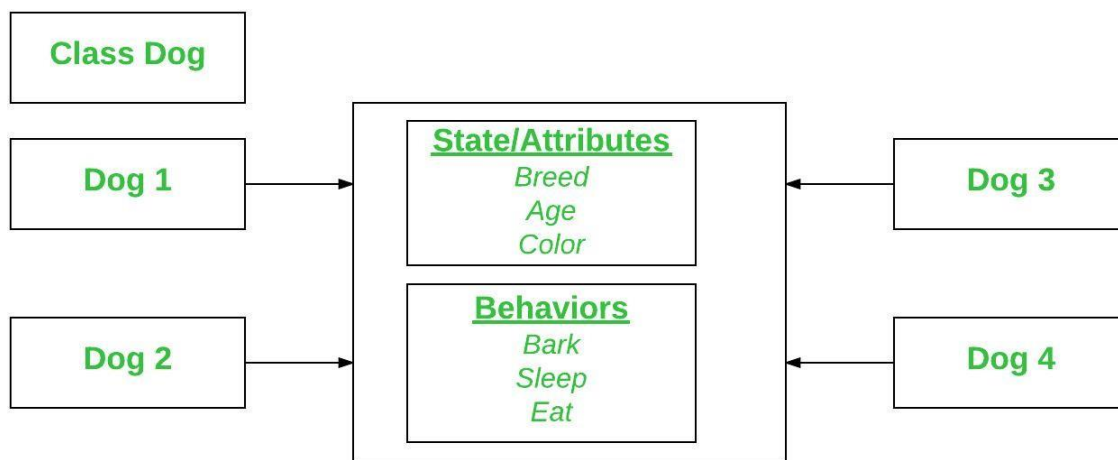
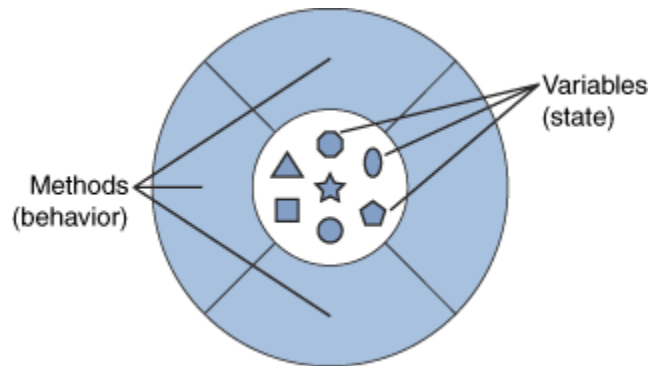
All the objects have a state, behavior and identity.



State: *what the objects have*, Student have a first name, last name, age, etc

Behavior: *what the objects do*, Student Can read(),write(),listen() etc

Identity: *what makes them unique*, Student have Student-ID-number, or an email which is unique.



Object Definitions:

- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.

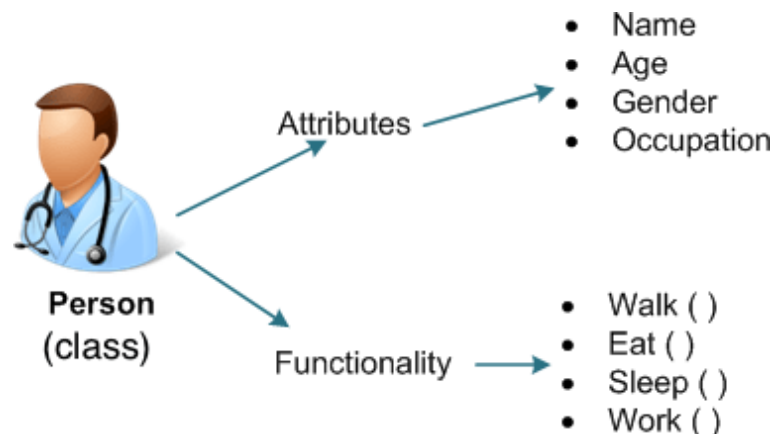
Ex: For example Bicycles have state (current gear, two wheels, number of gears) and behavior (braking, accelerating, slowing down, changing gears).

The main purpose of using objects are following.

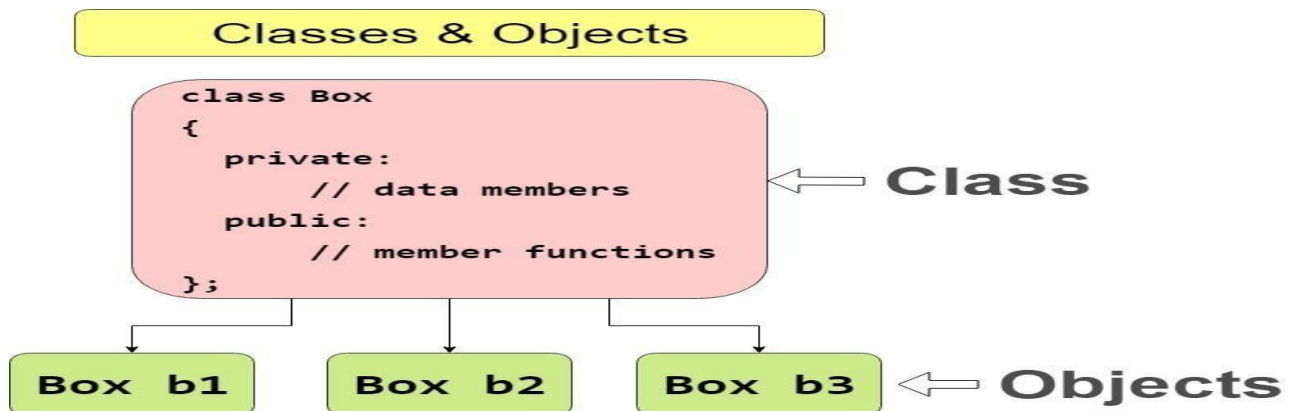
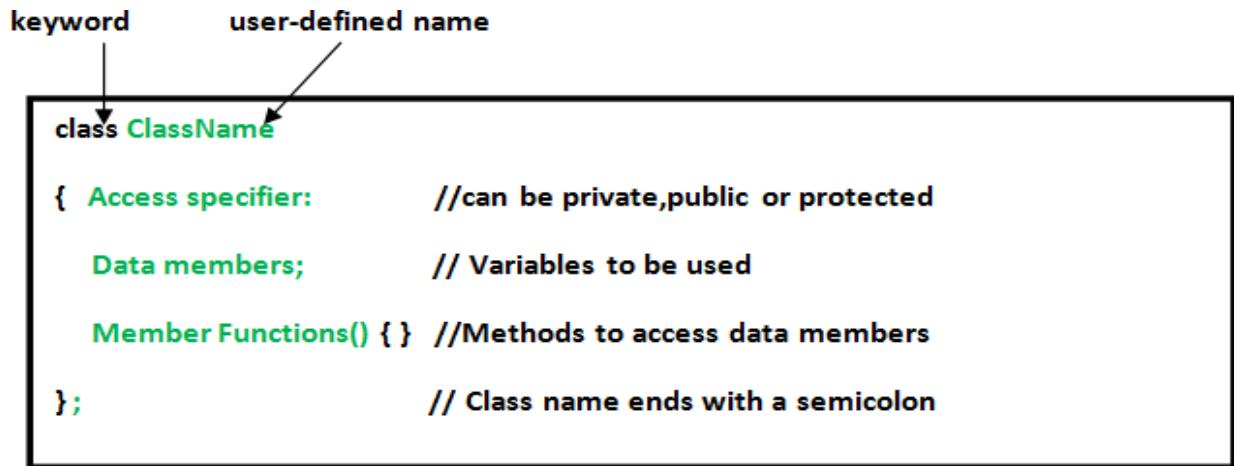
- They correspond to the real life entities.
- They provide interactions with the real world.
- They provide practical approach for the implementation of the solution.

Classes :

- In object-oriented programming, a **class** is a blueprint for creating **objects**.
- A class represents a collection of objects having same characteristics and common behavior.
- A **class** is a blueprint or prototype that defines the variables and the methods (functions) common to all **object**.
- A Class is a user-defined data-type which has data members and member functions.
- Data members are the data variables and member functions are the functions
- So, a class is a template for objects, and an object is an instance of a class.
- When the individual objects are created, they inherit all the variables and methods from the class.



Syntax:



EX:

```
#include <iostream>
#include <conio.h>
class Student
{
    private:                                //Access - Specifier
        char name[20];
        int rno, sub1, sub2, sub3;
        float total, avg;
    public:                                  //Access - Specifier
                                                //Member Functions read() and print()
    void read()
    {
```

```

//Get Input Values For Object Variables

cout << "Enter Name :";
cin >> name;

cout << "Enter Registration Number :";
cin >> rno;
cout << "Enter Marks for Subject 1,2 and 3 :";
cin >> sub1 >> sub2>> sub3;
}
void sum()
{
    total = sub1 + sub2 + sub3;
    avg = total / 3;
}

void print() {
//Show the Output

    cout << "Name :" << name << endl;
    cout << "Registration Number :" << rno << endl;
    cout << "Marks :" << sub1 << " , " << sub2 << " , " << sub3 << endl;
    cout << "Total :" << total << endl;
    cout << "Average :" << avg << endl;
}
};
int main() {
// Object Creation For Class

    Student s1, s2;

```

```
cout << "\n Student 1" << endl;
s1.read();
s1.sum();
s1.print();
cout << "\n Student 2" << endl;
s2.read();
s2.sum();
s2.print();
getch();
return 0;
}
```

Sample Output

Student 1

Enter Name : Naveen

Enter Registration Number :10001

Enter Marks for Subject 1,2 and 3 :90

80

65

Name : NAVEEN

Registration Number :10001

Marks :90 , 80 , 65

Total :235

Average :78.3333

Student 2

Enter Name :LOHITH

Enter Registration Number :10002

Enter Marks for Subject 1,2 and 3 :95

85

70

Name :LOHITH

Registration Number :10002

Marks :95 , 85 , 70

Total :250

Average :83.3333

Q) Object Oriented Analysis (OOA):

- Object Oriented Analysis (OOA) is process of understanding, finding and describing concepts in the problem domain.

- In OOA requirements are organized as objects. It integrates all the process and data.
- In OOA some advance models are used. The common models used in OOA are: Use cases, Object models.
- Use cases describe pictures or overview for standard domain functions that the system must achieved. Object models describe the names, class relations, operations, and properties of the main objects.

The three analysis techniques that are used ifor object-oriented analysis.They are

- **Object Modelling**
- **Dynamic Modelling**
- **Functional Modelling**

Object Modelling

Object modelling develops the static structure of the software system in terms of objects. It identifies the objects, the classes into which the objects can be grouped into and the relationships between the objects. It also identifies the main attributes and functions that are used in each class.

The process of object modelling can be visualized in the following steps –

- Identify objects and group into classes
- Identify the relationships among classes
- Create user object model diagram
- Define user object attributes
- Define the operations that should be performed on the classes
- Review glossary

Dynamic Modelling

After the static behavior of the system is analyzed, its behavior with respect to time and external changes needs to be examined. This is the purpose of dynamic modelling.

Dynamic Modelling can be defined as “a way of describing how an individual object responds to events, either internal events triggered by other objects, or external events triggered by the outside world”.

The process of dynamic modelling can be visualized in the following steps –

- Identify states of each object
- Identify events and analyze the applicability of actions
- Construct dynamic model diagram, comprising of state transition diagrams
- Express each state in terms of object attributes
- Validate the state–transition diagrams drawn

Functional Modelling

Functional Modelling is the final component of object-oriented analysis. The functional model shows the processes that are performed within an object and how the data changes as it moves between methods. It specifies the meaning of the operations of object modelling and the actions of dynamic modelling. The functional model corresponds to the data flow diagram of traditional structured analysis.

The process of functional modelling can be visualized in the following steps –

- Identify all the inputs and outputs
- Construct data flow diagrams showing functional dependencies

- State the purpose of each function
- Identify constraints
- Specify optimization criteria

Q) Object-Oriented Design

- Object-oriented design (OOD) is the process of using an object-oriented methodology to design a computing system or application.
- In the object-oriented design method, the system is viewed as a collection of objects .
- The state is distributed among the objects, and each object handles its state data.
- For example, in a Library Automation Software, each library representative may be a separate object with its data and functions to operate on these data.
- Objects have their internal data which represent their state.
- Similar objects create a class. In other words, each object is a member of some class. Classes may inherit features from the superclass.

The stages for object-oriented design can be identified as –

- Definition of the context of the system
- Designing system architecture
- Identification of the objects in the system
- Construction of design models
- Specification of object interfaces

Object-oriented design includes two main stages

1. System Design
2. Object Design

System Design

Object-oriented system design involves defining the context of a system followed by designing the architecture of the system.

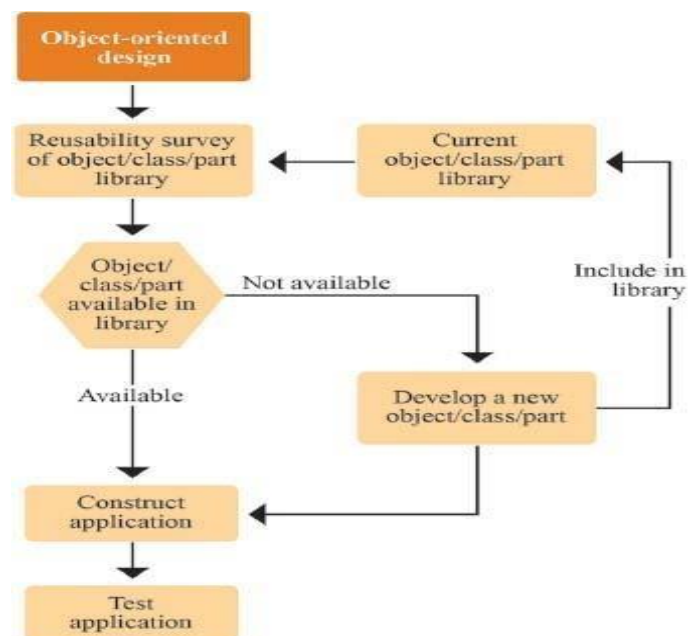
- **Context** – The context of a system has a static and a dynamic part. The static context of the system is designed using a simple block diagram of the whole system which is expanded into a hierarchy of subsystems. The subsystem model is represented by UML packages. The dynamic context describes how the system interacts with its environment. It is modelled using **use case diagrams**.
- **System Architecture** – The system architecture is designed on the basis of the context of the system in accordance with the principles of architectural design as well as domain knowledge. Typically, a system is partitioned into layers and each layer is decomposed to form the subsystems.

Object Design

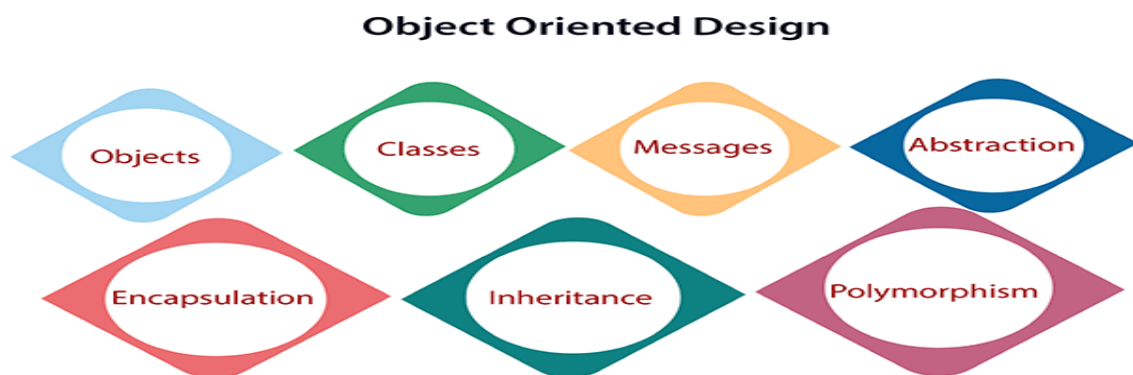
The object design phase determines the full definitions of the classes and associations used in the implementation, as well as the interfaces and algorithms of the methods used to implement operations. The object design phase adds internal objects for implementation and optimizes data structures and algorithms.

Object design includes the following phases –

- Find and define the objects.
- Organize the objects.
- Describe how the objects interact with one another.
- Define the external behavior of the objects.
- Define the internal behavior of the objects.



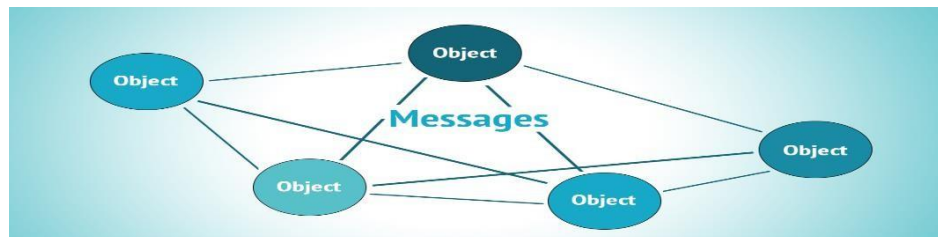
The different terms related to object design are:



1. **Objects:** Object is a real world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc. An object has attributes, behaviour, and Identity. Object is a **physical** entity. It contains member functions, variables that we have defined in the class.
2. **Classes:** Class is a **blueprint or template** from which objects are created. Inside a class, we define variables, constants, member functions, and other functionality. Class is a **group of similar objects**. Class is a **logical** entity.



3. **Messages:** Objects communicate with one another by sending and receiving information to each other. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

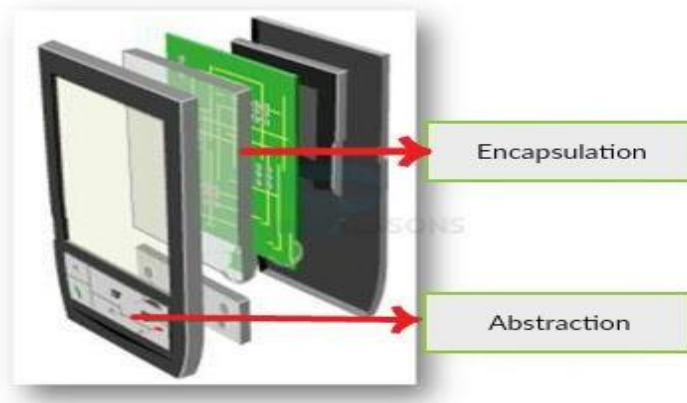


4. **Abstraction :**

Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

For example:

A database system details how a database is created while hiding how data is stored and maintained. The user only sees the database results on the screen.



5. Encapsulation:

- **Encapsulation** is a process of wrapping up of data and functions together as a single unit.
- It refers to the bundling of data with the methods that operate on that data to prevent them from being accessed by other classes.

```

class
{
    data members
    +
    methods (behavior)
}
    
```

ENCAPSULATION

6. Inheritance:

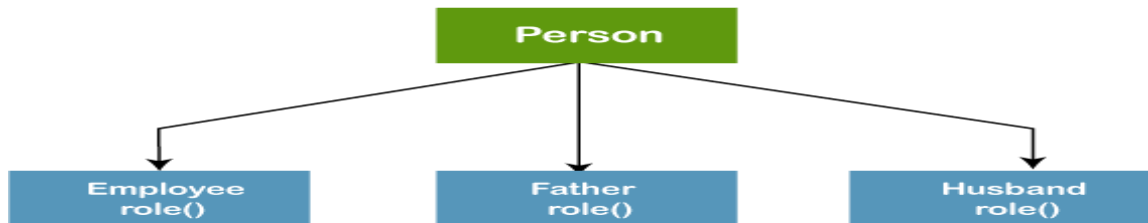
It's a technique of creating a new class from an existing one. A sub-class derives features from the parent class(base class). It provides code reusability.

There are various types of inheritance in OOP:

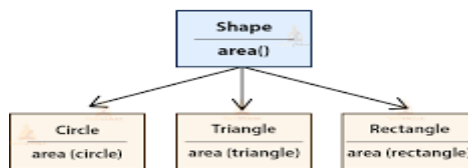
1. Single Inheritance
2. Multiple Inheritance
3. Hierarchical Inheritance
4. Multilevel Inheritance
5. Hybrid Inheritance.

5. Polymorphism:

The word **polymorphism** is derived from the two words i.e. **ploy** and **morphs**. Poly means many and morphs means forms. It allows us to create methods with the same name but different method signatures. It allows the developer to create clean, sensible, readable, and resilient code.



Example of Polymorphism in Java



Q) The Major and Minor Elements of the Object Models:

The object oriented model is defined by four major elements (abstraction, encapsulation, modularity and hierarchy) and by three minor ones (typecasting, simultaneity and persistence).

Major Elements

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

Minor Elements

- Typing
- Concurrency
- Persistence

Abstraction

- Abstraction means displaying only essential information and hiding the implementation details.

- Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.
- The main purpose of abstraction is hiding the unnecessary details from the users.
- Abstraction is selecting data from a larger pool to show only relevant details of the object to the user.

Ex: Consider a real life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car.

Encapsulation:

- **Encapsulation** is a process of wrapping up of data and functions together as a single unit.
- It refers to the bundling of data with the methods that operate on that data to prevents them from being accessed by other classes.
- In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class.
- A class is an example of encapsulation it consists of data and methods that have been bundled into a single unit.

Ex:

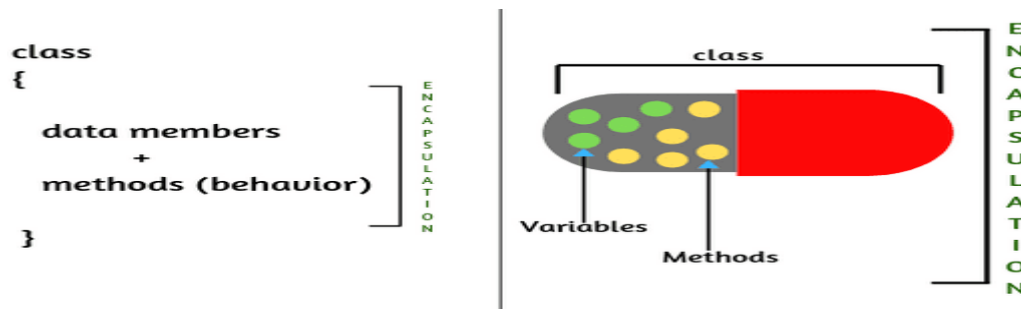


Fig: Encapsulation

Ex: Consider a real life example of encapsulation, in a company there are different sections like the accounts section, finance section, sales section etc. The finance section handles all the financial transactions and keep records of all the data related to finance. Similarly the sales section handles all the sales related activities and keep records of all the sales. Now there may arise a situation when for some reason an official from finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data. Here the data of sales section and the employees that can manipulate them are wrapped under a single name “sales section”.

Modularity

- Modularity is the process of decomposing a problem (program) into a set of modules so as to reduce the overall complexity of the problem.
- Modularity refers to the concept of making multiple modules first and then linking and combining them to form a complete system.
- Modularity enables re-usability and minimizes duplication.

Example: a banking program



Hierarchy:

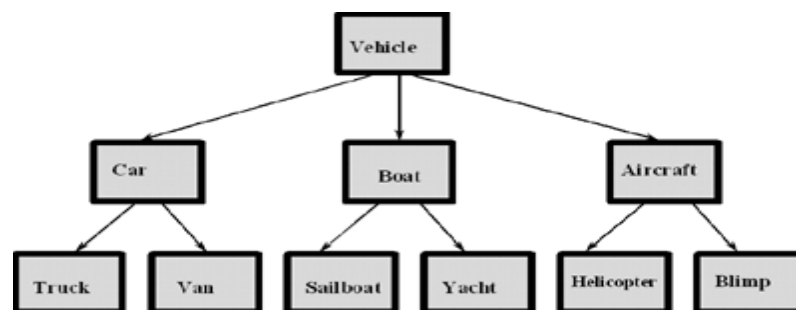
- A hierarchy is an organizational structure in which items are ranked according to levels of importance.
- Through hierarchy, a system can be made up of interrelated subsystems, which can have their own subsystems and so on until the smallest level components are reached.
- It uses the principle of “divide and conquer”.
- Hierarchy allows code reusability.

Two types of hierarchies in OOA are:

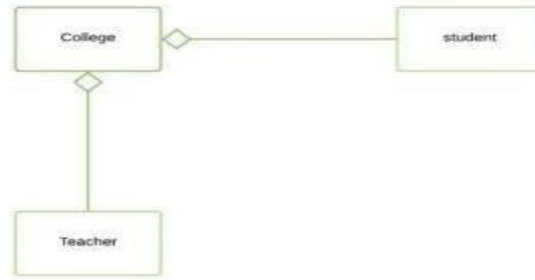
- “IS-A” hierarchy
- “PART-OF” hierarchy

“IS-A” hierarchy –

- In this from a super-class, a number of subclasses formed and which may again have subclasses and so on.
- For example, if we derive a class Rose from a class Flower, we can say that a rose “is-a” flower.



“PART–OF” hierarchy – It defines the hierarchical relationship by which a class may be composed of other classes.



Static and Dynamic Typing:

- Typing of a class prevents objects of different types from being interchanged. Binding is the association of the variables or classes to their types.
- The programming language can be categorized as strongly typed or weakly typed language.
- In strongly typed languages, the violation of type conformance can be detected at the compile time.
- This type of binding variables to their types called static binding or early binding
- In weakly typed languages, violation of typing conformance cannot be detected until runtime. Here variable are bound to their type at only execution time. This is called as dynamic binding or late binding. Static bind is fast but rigid while dynamic binding is slow but flexible.

Concurrency:

Concurrency in operating systems allows performing multiple tasks or processes simultaneously. When a single process exists in a system, it is said that there is a single thread of control. However, most systems have multiple threads, some active, some waiting for CPU, some suspended, and some terminated. Systems with multiple CPUs inherently permit concurrent threads of control; but systems running on a single CPU use appropriate algorithms to give equitable CPU time to the threads so as to enable concurrency.

Persistence

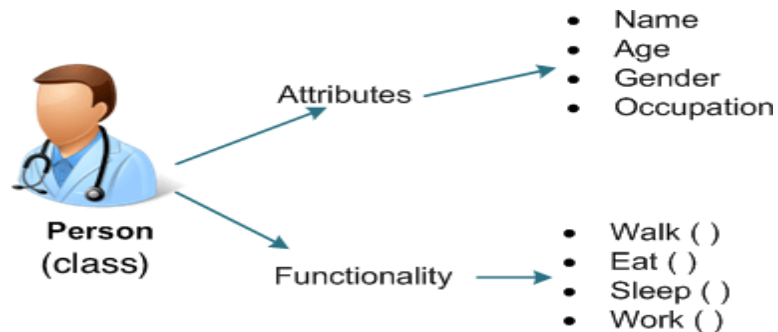
An object occupies a memory space and exists for a particular period of time. In traditional programming, the lifespan of an object was typically the lifespan of the execution of the program that created it. In files or databases, the object lifespan is longer than the duration of the process creating the object. This property by which an object continues to exist even after its creator ceases to exist is known as persistence.

UNIT-2

Classes and objects in ooad

Q) Classes :

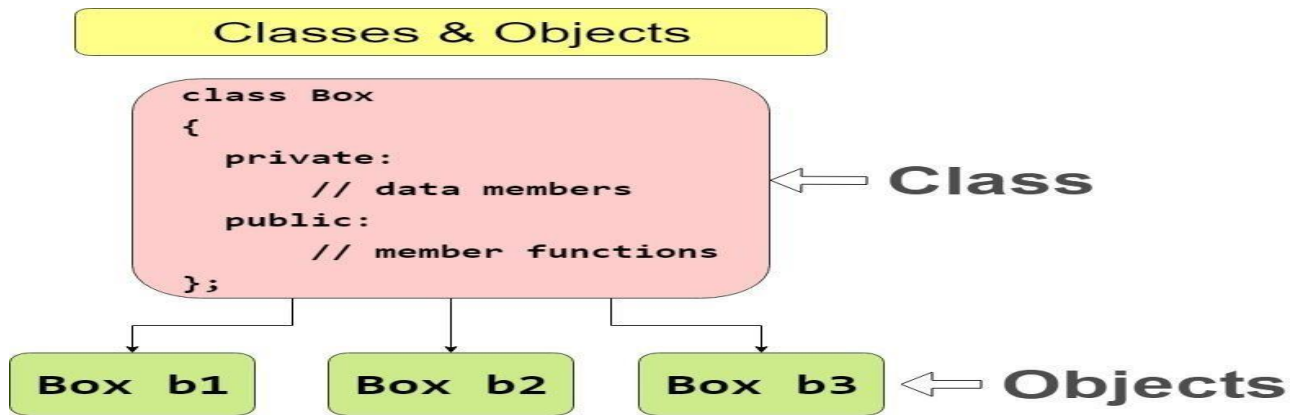
- In object-oriented programming, a **class** is a blueprint for creating **objects**.
- A class represents a collection of objects having same characteristics and common behavior.
- A **class** is a blueprint or prototype that defines the variables and the methods (functions) common to all **object**.
- Creation of an object as a member of a class is called instantiation. Thus, object is an instance of a class.
- A Class is a user-defined data-type which has data members and member functions.
- Data members are the data variables and member functions are the functions
- When the individual objects are created, they inherit all the variables and methods from the class.



Syntax:

keyword user-defined name

```
class ClassName
{
    Access specifier:           //can be private,public or protected
    Data members;              // Variables to be used
    Member Functions() { }     //Methods to access data members
};                             // Class name ends with a semicolon
```



Ex:

EX:

```
#include <iostream>
#include <conio.h>
class Student
{
    private:                                //Access - Specifier
        char name[20];
        int rno, sub1, sub2, sub3;
        float total, avg;
    public:                                  //Access - Specifier
                                                //Member Functions read() and print()

    void read()
    {
```

//Get Input Values For Object Variables

```
cout << "Enter Name :";  
cin >> name;
```

```
cout << "Enter Registration Number :";  
cin >> rno;  
cout << "Enter Marks for Subject 1,2 and 3 :";  
cin >> sub1 >> sub2 >> sub3;
```

```
}
```

```
void sum()
```

```
{
```

```
    total = sub1 + sub2 + sub3;  
    avg = total / 3;
```

```
}
```

```
void print() {
```

//Show the Output

```
    cout << "Name :" << name << endl;  
    cout << "Registration Number :" << rno << endl;  
    cout << "Marks :" << sub1 << " , " << sub2 << " , " << sub3 << endl;  
    cout << "Total :" << total << endl;  
    cout << "Average :" << avg << endl;
```

```
}
```

```
};
```

```
int main() {
```

// Object Creation For Class

```
    Student s1, s2;
```

```

    cout << "\n Student 1" << endl;
    s1.read();
    s1.sum();
    s1.print();
    cout << "\n Student 2" << endl;
    s2.read();
    s2.sum();
    s2.print();
    getch();
    return 0;
}

```

Sample Output

Student 1

Enter Name : Naveen

Enter Registration Number :10001

Enter Marks for Subject 1,2 and 3 :90

80

65

Name : NAVEEN

Registration Number :10001

Marks :90 , 80 , 65

Total :235

Average :78.3333

Student 2

Enter Name :LOHITH

Enter Registration Number :10002

Enter Marks for Subject 1,2 and 3 :95

85

70

Name :LOHITH

Registration Number :10002

Marks :95 , 85 , 70

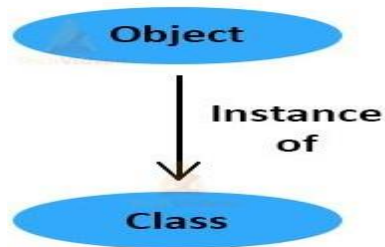
Total :250

Average :83.3333

Q) Objects in OOAD:

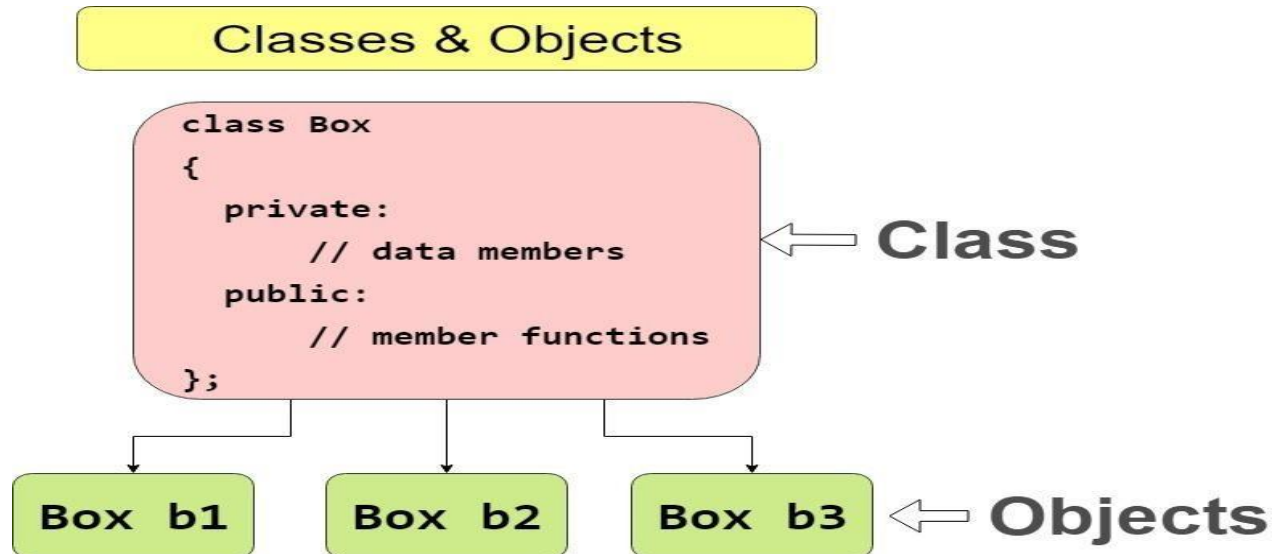
- Object is a real-world entity that has state and behavior and identity.
- Object is a real-world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.

- An object is *a runtime entity*.
- Object is created **many times** as per requirement.
- Object **allocates memory when it is created**.



Syntax: `ClassName ObjectName;`

Ex: `Student stu1;`



Q) Nature of the Object:

- An object has state, behavior, and identity. the structure and behavior of similar objects are defined in their common class.
- The terms instance and object are interchangeable.
- Object can be physical or logical (tangible and intangible).
- **Tangible is something which a person can see, feel or touch and thus they have the physical existence.**
- **whereas, the intangible is something which a person cannot see, feel or touch and thus do not have any of the physical existence.**

An object has three characteristics:

State: The state of the object contains all the properties of the object and current values of each of these properties.

Behaviour: It represents the behaviour (functionality) of an object. It is represented by methods of an object.

Types of Operations

Modifier: An operation that alters the state of an object

Selector: An operation that accesses the state of an object, but does not alter the state

Iterator: An operation that permits all parts of an object to be accessed in some well-defined order.

Identity: An object identity is typically implemented via a unique ID. It gives a unique name to an object and enables one object to interact with other objects.

Example of an object

Object-Car

Identity-

Object name- car

Attributes-

Brand

Model

Color

Behavior-

Start

stop

Speed



EX:

Q) The Nature Of The Class:

- An object is real world entity that exists in time and space.
- A class represents a collection of objects having same characteristics and common behavior.
- A single object is the instance of a class.

Interface and implementation:

- The interface of a class provides its outside view and hiding its structure and its behavior.
- This interface primarily consists of the declarations of all the operations applicable to instances of this class, but it may also include the declaration of other classes, constants, variables, and exceptions as needed to complete the abstraction.
- The implementation of a class is its inside view, which contains the secrets of its behavior.
- The implementation of a class primarily consists of the implementation of all of the operations defined in the interface of the class.

We can further divide the interface of a class into four parts:

1. **Public:** a declaration that is accessible to all clients
2. **Protected:** a declaration that is accessible only to the class itself and its sub-classes
3. **Private:** a declaration that is accessible only to the class itself
4. **Package:** a declaration that is accessible only by classes in the same package

Q) Relationship Between Objects:

- The relationship between objects defines how these objects will interact or collaborate to perform an operation in an application.
- In any application, objects of user interface classes interact with the business layer objects in order to perform an operation

Types of Relationships:

- **Links**
- **Aggregation**

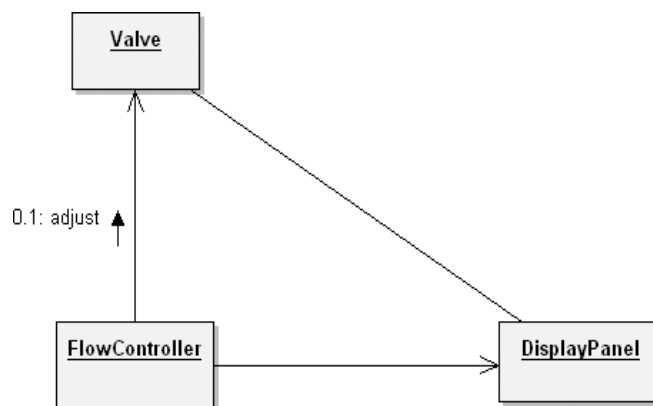
Links:

- The link is a physical or conceptual connection between objects.
- A link denotes the specific association through which one object (the client) applies (request) the services of another object (the supplier) .
- for example, a student, Ravi study in university.
- Through a link, one object may invoke the methods or navigate through another object.
- A link describes the relationship between two or more objects.
- Message passing between two objects is typically unidirectional, although it may occasionally be bidirectional.
- message passing is initiated by the client and is directed toward the supplier, data may flow in either direction across a link.

As a participant in a link, an object may play one of three roles.

- 1. Controller:** This object can operate on other objects but is not operated on by other objects.
In some contexts, the terms active object and controller are interchangeable.
- 2. Server:** This object doesn't operate on other objects; it is only operated on by other objects.
- 3. Proxy:** This object can both operate on other objects and be operated on by other objects. A proxy is usually created to represent a real-world object in the domain of the application.

In the following example FlowController acts as a controller object, DisplayPanel acts as a server object, and Valve acts as a proxy



Aggregation:

- **Aggregation is a** type of relationship i.e an object can be composed of one or more objects in the form of its properties
- An **aggregation** is a collection, or the gathering of things together.
- **Aggregation is** a relationship among classes by which a class can be made up of any combination of objects of other classes.
- It is a binary association, i.e., it only involves two classes. It is a kind of relationship in which the child is independent of its parent.
- It represents **Has-A** relationship.
- It describes a part-whole or part-of relationship.

- The symbol of aggregation is



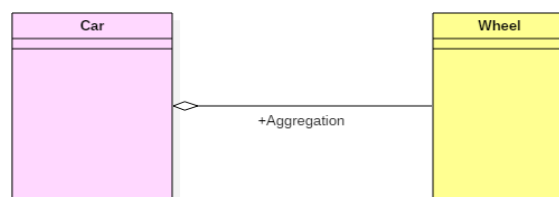
Example

In the relationship, “a car has-a motor”, car is the whole object or the aggregate, and the motor is a “part-of” the car. Aggregation may denote –

- **Physical containment** – Example, a computer is composed of monitor, CPU, mouse, keyboard, and so on.
- **Conceptual containment** – Example, shareholder has-a share.

Aggregation Example:

- Let us consider an example of a car and a wheel.
- A car needs a wheel to function correctly, but a wheel doesn't always need a car. It can also be used with the bike, bicycle, or any other vehicles but not a particular car. Here, the wheel object is meaningful even without the car object. Such type of relationship is called UML Aggregation relation.



Q) Relationships among Classes:

- Association is used to represent the relation between the classes.
- Classes are interrelated to each other in specific ways.
- The following are the relationships used in classes
 1. **Association**
 2. **inheritance**
 3. **Aggregation**

4. Polymorphism

5. Dependency

1. Association:

- Association is relation between two separate classes which establishes through their Objects.
- Association is a structural relationship that represents how two entities are linked or connected to each other within a system
- Association can be one-to-one, one-to-many, many-to-one, many-to-many.
- In Object-Oriented programming, an Object communicates to other Object to use functionality and services provided by that object. Composition and Aggregation are the two forms of association.

The symbol for association is



Degree of an Association

Degree of an association denotes the number of classes involved in a connection. Degree may be unary, binary, or ternary.

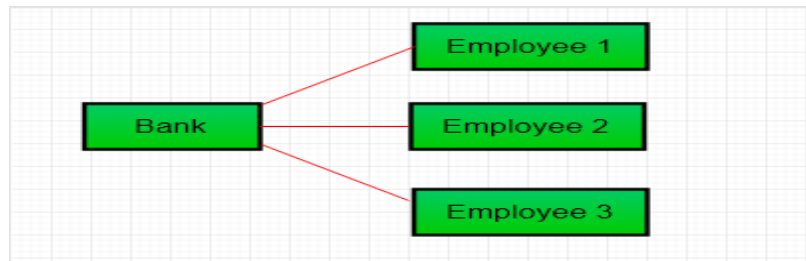
- A **unary relationship** connects objects of the same class.
- A **binary relationship** connects objects of two classes.
- A **ternary relationship** connects objects of three or more classes.

Cardinality Ratios of Associations

Cardinality of a binary association denotes the number of instances participating in an association. There are three types of cardinality ratios, namely –

- **One-to-One** – A single object of class A is associated with a single object of class B.
- **One-to-Many** – A single object of class A is associated with many objects of class B.
- **Many-to-Many** – An object of class A may be associated with many objects of class B and conversely an object of class B may be associated with many objects of class A.

In above example two separate classes Bank and Employee are associated through their Objects. Bank can have many employees, So it is a one-to-many relationship.

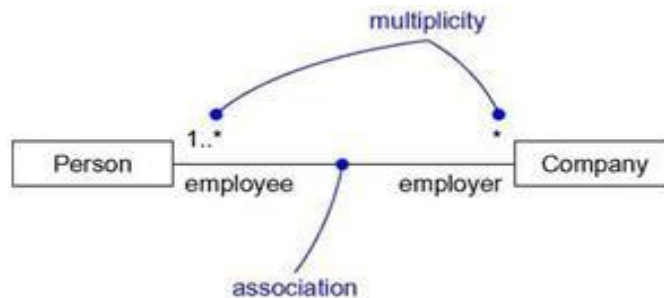


Multiplicity:

- Multiplicity in an association specifies how many objects participate in a relationship. Multiplicity decides the number of related objects.
- Multiplicity is generally explained as “one” or “many,” but in general it is a subset of the non-negative integers.

Table 1: Multiplicity Indicators.

Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only n (where $n > 1$)
0..n	Zero to n (where $n > 1$)
1..n	One to n (where $n > 1$)



Inheritance

- Inheritance is the process of creating new classes from the existing classes.
- The existing classes are called the base classes/parent classes/super-classes, and the new classes are called the derived classes/child classes/subclasses.
- The subclass can inherit or derive the attributes and methods of the super-class.
- The subclass may add its own attributes and methods and may modify any of the super-class methods. I
- Inheritance defines an “is – a” relationship.

Types of Inheritance

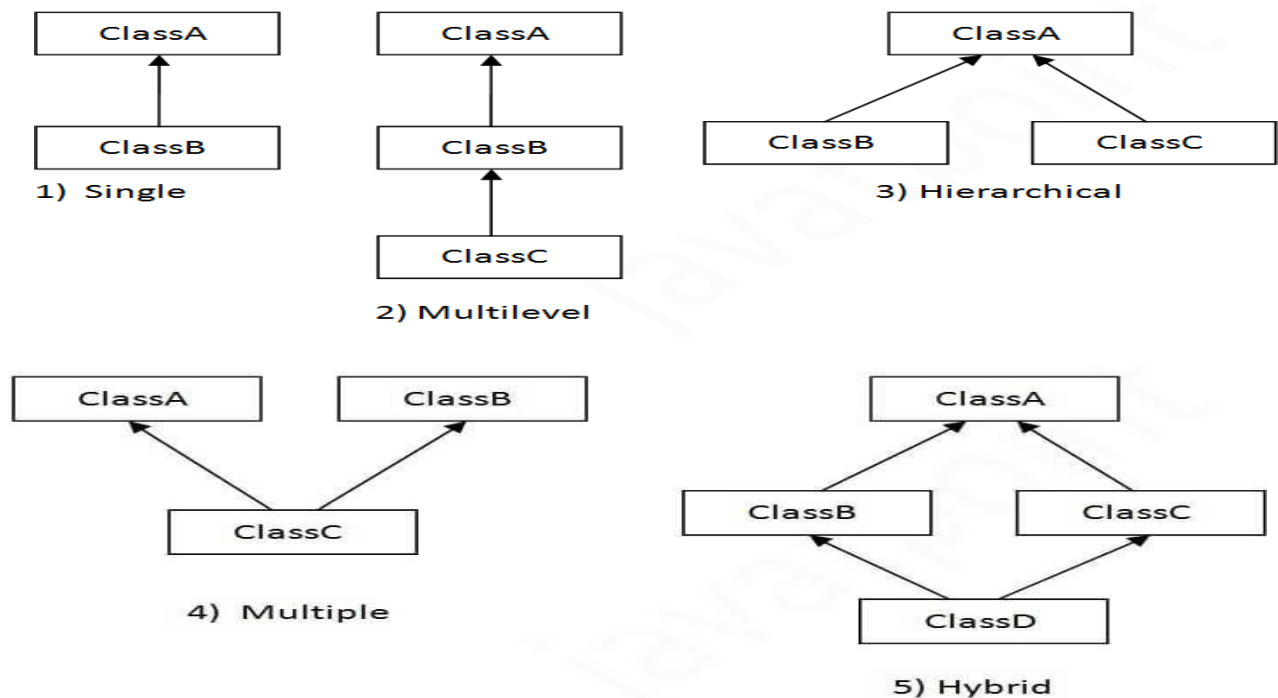
Single Inheritance – A subclass derives from a single super-class.

Multiple Inheritance – A subclass derives from more than one super-classes.


Multilevel Inheritance – A subclass derives the another subclass that is already derived from the super class.

Hierarchical Inheritance – When two or more classes inherits a single class, it is known as *hierarchical inheritance* .it continuing for a number of levels, so as to form a tree structure.

Hybrid Inheritance – A combination of multiple and multilevel inheritance is called Hybrid inheritance.



Aggregation:

- **Aggregation is** a relationship among classes by which a class can be made up of any combination of objects of other classes.
- An **aggregation** is a collection, or the gathering of things together.
- It represents **Has-A** relationship.
- It describes a part-whole or part-of relationship. It is a binary association, i.e., it only involves two classes.
- The symbol of aggregation is 

Example

In the relationship, “a car has–a motor”, car is the whole object or the aggregate, and the motor is a “part–of” the car. Aggregation may denote –

- **Physical containment** – Example, a computer is composed of monitor, CPU, mouse, keyboard, and so on.
- **Conceptual containment** – Example, shareholder has–a share.

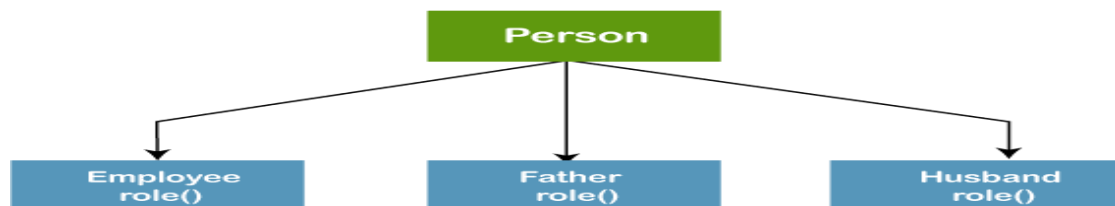
Aggregation Example:

- Let us consider an example of a car and a wheel.
- A car needs a wheel to function correctly, but a wheel doesn't always need a car. It can also be used with the bike, bicycle, or any other vehicles but not a particular car. Here, the wheel object is meaningful even without the car object. Such type of relationship is called UML Aggregation relation.

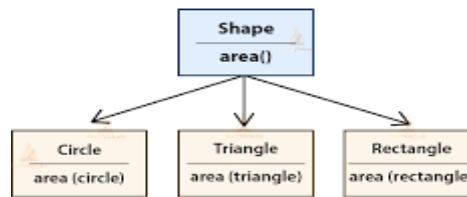


Polymorphism:

- The word **polymorphism** is derived from the two words i.e. **ploy** and **morphs**.
- Poly means many and morphs means forms.
- It allows us to create methods with the same name but different method signatures.
- It allows the developer to create clean, sensible, readable, and resilient code.



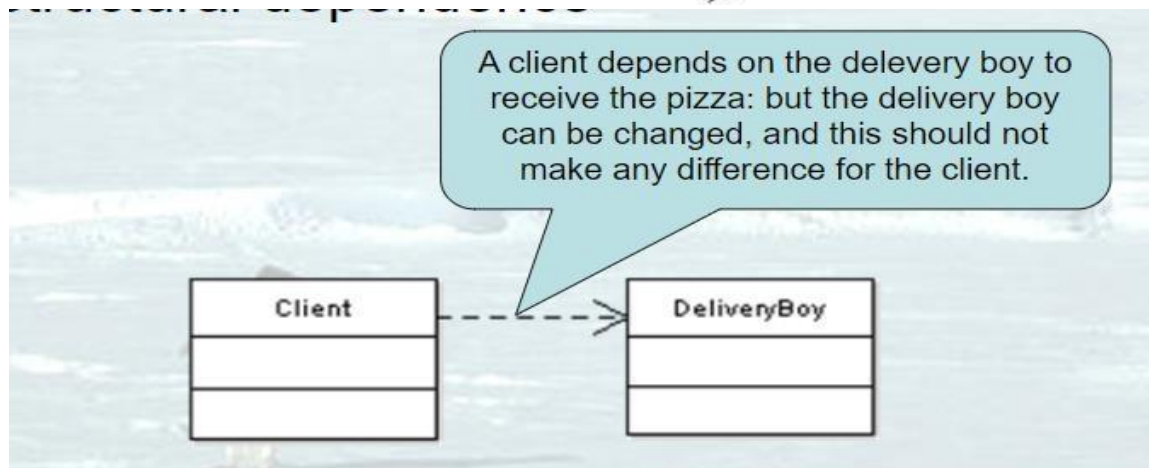
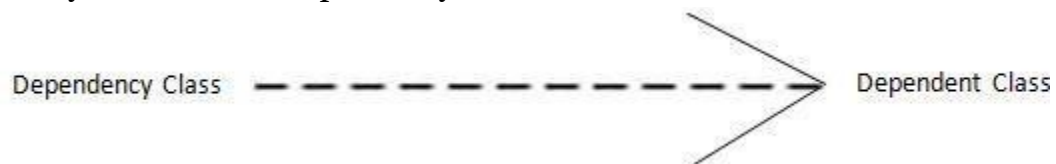
Example of Polymorphism in Java



Dependency:

Dependency is defined as a relation between two classes, where one class depends on another class but another class may or not may depend on the first class.

- So any change in one of the classes may affect the functionality of the other class, that depends on the first one.
- The symbol used for dependency is



Q) The Interplay of Classes and Objects

Relationship between Classes and Objects

- Classes and objects are closely related concepts
- Every object is an instance of some class.
- Every class has zero or more instances.
- Classes are usually static.
- Their definition is fixed prior to program execution.
- The class of most objects is static.
- Once created, an object doesn't change its class.

- Objects are created and destroyed frequently as a program runs.
- During analysis, the developer identifies the classes and objects (key abstractions) in the problem domain.
- During early design, the developer invents structures (mechanisms) so objects can work together to satisfy system requirements.

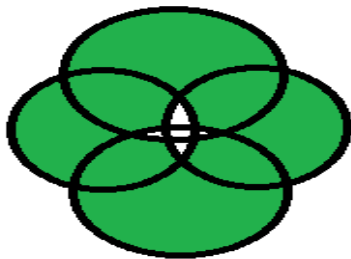
Q) Measuring the Quality of an Abstraction

- The design of classes and objects is incremental and iterative.
- The quality of design can be measured with the following metrics

1. **Coupling**
2. **Cohesion**
3. **Sufficiency**
4. **Completeness**
5. **Primitiveness**

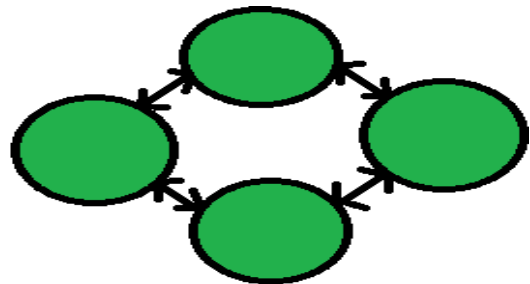
1. Coupling:

- Coupling is the degree of interaction between two classes between classes.
- Two classes that are tightly coupled are strongly dependent on each other.
- However, two classes that are loosely coupled are not dependent on each other.
- **Uncoupled classes** have no interdependence at all within them.



Tight coupling:

1. **More Interdependency**
2. **More coordination**
3. **More information flow**

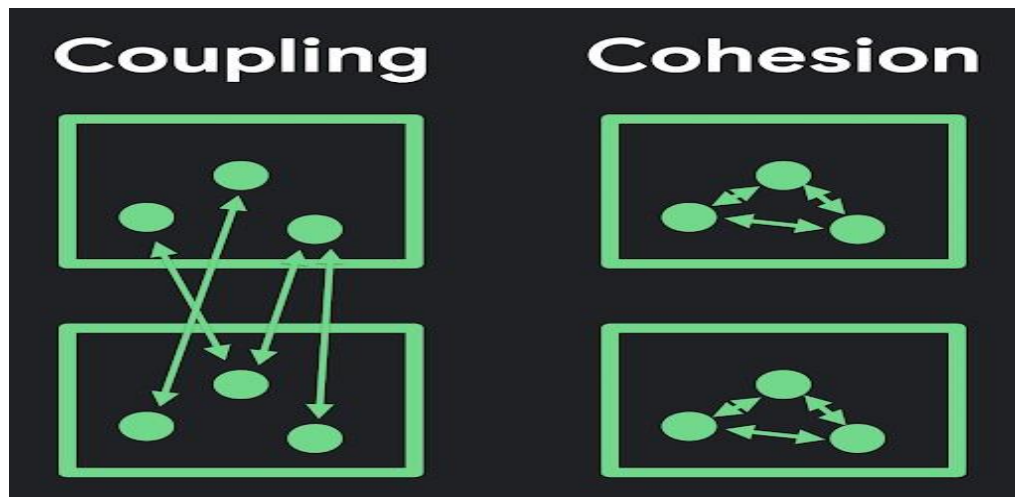


Loose coupling:

1. **Less Interdependency**
2. **Less coordination**
3. **Less information flow**

2. Cohesion:

- cohesion focuses on how single module/class is designed.
- Coupling is all about how classes interact with each other, on the other hand cohesion focuses on how single class is designed.
- Higher the cohesiveness of the class, better is the OO design. Highly cohesive classes are much easier to maintain and less frequently changed.



Sufficiency:

- Sufficiency means that you should include only those attributes and operations that are needed for the system you are modeling.
- You must create classes that contain all the attributes and operations necessary to perform the tasks you require of them.
- So, if you create a Car class, you must include an accelerate operation – otherwise no car object will be able to change speed.
- If you want to create a class that models a typical car, you need to specify things like its speed and model.

Completeness

- Completeness determines whether the interface of the class captures all the behavior of the class.
- The attributes and operations you give to a class must be general enough to allow other classes to communicate with it.

Primitiveness

- Primitiveness means that you shouldn't make a class unnecessarily complicated. It is a principle that lets you control a complex behavior with just a few operations.
- For any class, you should choose operations and attributes that are very basic.
- To create more complex class states, you can simply change the values of the attributes you have chosen Or you can combine simple operations to create complex ones.