Program : **B.Tech**

Subject Name: **Object Oriented Programming & Methodology**

Subject Code: **CS-305**

Semester: **3rd**

Class Notes

UNIT I

1. Object Oriented Thinking:-

Traditionally, a programming problem is attacked by coming up with some kinds of data representations, and procedures that operate on that data. Under this model, data is inert, passive, and helpless; it sits at the complete mercy of a large procedural body, which is active, logical, and all-powerful.
The problem with this approach is that programs are written by programmers, who are only human and can only keep so much detail clear in their heads at any one time. As a project gets larger, its procedural core grows to the point where it is difficult to remember how the whole thing works. Minor lapses of thinking and typographical errors become more likely to result in well-concealed bugs. Complex and unintended interactions begin to emerge within the procedural core, and maintaining it becomes like trying to carry around an angry squid without letting any tentacles touch your face. There are guidelines for programming that can help to minimize and localize bugs within this traditional paradigm, but there is a better solution that involves fundamentally changing the way we work.

1.1 What is abstract interaction?

If you want to change the television channel from your seat, you use a remote control. That remote control is an **object** with a number of **attributes** and **behaviors** hidden inside of it. Without an understanding of those hidden attributes—the microchips, wiring, etc.—you still know and expect that pressing a button will perform that particular function. You've interacted with the remote control in the abstract, skipping the steps the remote was designed to carry out. That's the beauty of OOP—the focus is on how the objects behave, not the code required to tell them how to behave.

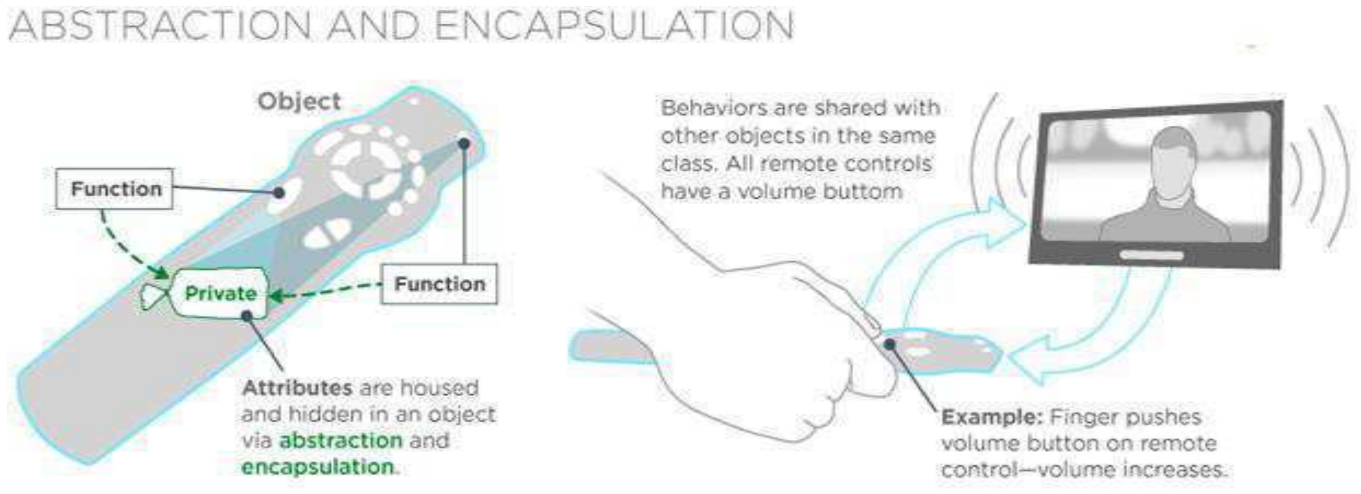
ABSTRACTION AND ENCAPSULATION
Object
Function
Private
Function
Attributes are housed and hidden in an object via abstraction and encapsulation.
Behaviors are shared with other objects in the same class. All remote controls have a volume buttom
Example: Finger pushes volume button on remote control—volume increases.

Figure1. Abstract interaction

2. What Are Objects?

A car is an example of a complex object, with many attributes. We don't need to understand all of its internal mechanics, what kind of engine it has, how the gas makes it run, or even where the gas came from in order to know how to interact with it. The car's behaviors have been made simple for us through object-oriented logic: put the key in the ignition, and the car turns on and gets us where we need to go. The attributes that make this possible—all of the car's parts, electronics, and engineering—are a "package" we don't need to break down in order to understand.

Apply this to software building, and it allows developers to break down big, complicated projects into compartmentalized objects, program them to have attributes and behaviors, then essentially set them aside and focus on programming how the objects interact—a higher level of thinking that makes writing code less linear and more efficient. Modern, high-level languages like Python and Ruby are perfect examples of OOP. The fact that they're able to be so streamlined gets right to the heart of OOP logic.

2 .Object-Oriented Programming & Back-End Development

What is object-oriented programming in terms of how a site is built? OOP defines most modern server-side scripting languages, which are the languages back-end developers use to write software and database technology. This behind-the-scenes, server-side technology tells a website or web application how to behave, and also builds the architecture for a site to interact with its database. That scaffolding is how data is delivered and processed, effectively making it the brain of a website. And that's where object-oriented logic comes into play.

If a website's brain uses object-oriented logic, it's designed to think of data as objects**.** It affects how a site is built from the ground up, how data is organized, how later growth and maintenance of the site will occur, and more.

2.1 Benefits of object-oriented technology include:

- Ease of software design
- Productivity
- Easy testing, debugging, and maintenance
- It's reusable
- More thorough data analysis, less development time, and more accurate coding, thanks to OOP's inheritance method
- Data is safe and secure, with less data corruption, thanks to hiding and abstraction.
- It's sharable (classes are reusable and can be distributed to other networks).

3. Difference Between Procedure Oriented Programming (Pop) &Object-Oriented Programming (Oop)

Object Oriented Programming	Procedure Oriented Programming	Points
In OOP, program is divided into parts called objects.	In POP, program is divided into small parts called functions.	Divided Into

In OOP, Importance is given to the data rather than procedures or functions because it works as a real world.	In POP, Importance is not given to data but to functions as well as sequence of actions to be done.	Importance
OOP follows Bottom Up approach.	POP follows Top Down approach.	Approach
OOP has access specifies named Public, Private, Protected, etc.	POP does not have any access specified.	Access Specifies
In OOP, objects can move and communicate with each other through member functions.	In POP, Data can move freely from function to function in the system.	Data Moving
OOP provides an easy way to add new data and function.	To add new data and function in POP is not so easy.	Expansion
In OOP, data cannot move easily from function to function, it can be kept public or private so we can control the access of data.	In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	Data Access
OOP provides Data Hiding so provides more security.	POP does not have any proper way for hiding data so it is less secure.	Data Hiding
In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.	In POP, Overloading is not possible.	Overloading
Example of OOP is: C++, JAVA, VB.NET, C#.NET.	Example of POP is: C, VB, FORTRAN, and Pascal.	Examples

4 Principles Of Object-Oriented Systems

The conceptual framework of object–oriented systems is based upon the object model. There are two categories of elements in an object-oriented system −

Major Elements − By major, it is meant that if a model does not have any one of these elements, it ceases to be object oriented. The four major elements are −

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

Minor Elements − By minor, it is meant that these elements are useful, but not indispensable part of the object model. The three minor elements are −

- Typing
- Concurrency
- Persistence

1) Abstraction

Abstraction means to focus on the essential features of an element or object in OOP, ignoring its extraneous or accidental properties. The essential features are relative to the context in which the object is being used

.

Grady Booch has defined abstraction as follows −

"An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer."

Example − When a class Student is designed, the attributes enrolment number, name, course, and address are included while characteristics like pulse_rate and size_of_shoe are eliminated, since they are irrelevant in the perspective of the educational institution.

2) Encapsulation

Encapsulation is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside. The class has methods that provide user interfaces by which the services provided by the class may be used.

3) 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. Booch has defined modularity as −

"Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules."

Modularity is intrinsically linked with encapsulation. Modularity can be visualized as a way of mapping encapsulated abstractions into real, physical modules having high cohesion within the modules and their inter–module interaction or coupling is low.

4) Hierarchy

In Grady Booch's words, "Hierarchy is the ranking or ordering of abstraction". 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.

The two types of hierarchies in OOA are −

- **"IS–A" hierarchy** − It defines the hierarchical relationship in inheritance, whereby from a super-class, a number of subclasses may be derived 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 in aggregation by which a class may be composed of other classes. For example, a flower is composed of sepals, petals, stamens, and carpel. It can be said that a petal is a "part–of" flower.

5) Typing

According to the theories of abstract data type, a type is a characterization of a set of elements. In OOP, a class is visualized as a type having properties distinct from any other types. Typing is the enforcement of the notion that an object is an instance of a single class or type. It also enforces that objects of different types may not be generally interchanged; and can be interchanged only in a very restricted manner if absolutely required to do so.

The two types of typing are −

- **Strong Typing** − Here, the operation on an object is checked at the time of compilation, as in the programming language Eiffel.
- **Weak Typing** − Here, messages may be sent to any class. The operation is checked only at the time of execution, as in the programming language Smalltalk.

6) 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.

In an object-oriented environment, there are active and inactive objects. The active objects have independent threads of control that can execute concurrently with threads of other objects. The active objects synchronize with one another as well as with purely sequential objects.

7) 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.

5 Structured Analysis vs. Object Oriented Analysis

The Structured Analysis/Structured Design (SASD) approach is the traditional approach of software development based upon the waterfall model. The phases of development of a system using SASD are −

- Feasibility Study
- Requirement Analysis and Specification
- System Design
- Implementation
- Post-implementation Review

Now, we will look at the relative advantages and disadvantages of structured analysis approach and object-oriented analysis approach.

5.1 Advantages/Disadvantages of Object Oriented Analysis

Advantages	**Disadvantages**
Focuses on data rather than the procedures as in Structured Analysis.	Functionality is restricted within objects. This may pose a problem for systems which are intrinsically procedural or computational in nature.
The principles of encapsulation and data hiding help the developer to develop systems that cannot be	It cannot identify which objects would generate an optimal system design.

tampered by other parts of the system.	
The principles of encapsulation and data hiding help the developer to develop systems that cannot be tampered by other parts of the system.	The object-oriented models do not easily show the communications between the objects in the system.
It allows effective management of software complexity by the virtue of modularity.	All the interfaces between the objects cannot be represented in a single diagram.
It can be upgraded from small to large systems at a greater ease than in systems following structured analysis.	

5.2 Advantages/Disadvantages of Structured Analysis

Advantages	**Disadvantages**
As it follows a top-down approach in contrast to bottom-up approach of object-oriented analysis, it can be more easily comprehended than OOA.	In traditional structured analysis models, one phase should be completed before the next phase. This poses a problem in design, particularly if errors crop up or requirements change.
It is based upon functionality. The overall purpose is identified and then functional decomposition is done for developing the software. The emphasis not only gives a better understanding of the system but also generates more complete systems.	The initial cost of constructing the system is high, since the whole system needs to be designed at once leaving very little option to add functionality later.
The specifications in it are written in simple English language, and hence can be more easily analyzed by non-technical personnel.	It does not support reusability of code. So, the time and cost of development is inherently high.

6 . Advantages And Disadvantages Of Object-Oriented Programming (Oop)

This reading discusses advantages and disadvantages of object-oriented programming, which is a well-adopted programming style that uses interacting objects to model and solve complex programming tasks. Two examples of popular object-oriented programming languages are Java and C++. Some other well-known object-oriented programming languages include Objective C, Perl, Python, JavaScript, Simula, Modula, Ada, Smalltalk, and the Common Lisp Object Standard.

6.1 Some of the advantages of object-oriented programming include:

1. Improved software-development productivity: Object-oriented programming is modular, as it provides separation of duties in object-based program development. It is also extensible, as objects can be extended to include new attributes and behaviors. Objects can also be reused within an across

applications. Because of these three factors – modularity, extensibility, and reusability – object-oriented programming provides improved software-development productivity over traditional procedure-based programming techniques.

2. Improved software maintainability: For the reasons mentioned above, object- oriented software is also easier to maintain. Since the design is modular, part ofthe system can be updated in case of issues without a need to make large-scale changes.

3. Faster development: Reuse enables faster development. Object-oriented programming languages come with rich libraries of objects, and code developed during projects is also reusable in future projects.

4. Lower cost of development: The reuse of software also lowers the cost of development. Typically, more effort is put into the object-oriented analysis and design, which lowers the overall cost of development.

5. Higher-quality software: Faster development of software and lower cost of development allows more time and resources to be used in the verification of the software. Although quality is dependent upon the experience of the teams, object- oriented programming tends to result in higher-quality software.

6.2 Some of the disadvantages of object-oriented programming include:

1. Steep learning curve: The thought process involved in object-oriented programming may not be natural for some people, and it can take time to get used to it. It is complex to create programs based on interaction of objects. Some of the key programming techniques, such as inheritance and polymorphism, can be challenging to comprehend initially.

2. Larger program size: Object-oriented programs typically involve more lines of code than procedural programs.
3. Slower programs: Object-oriented programs are typically slower than procedure- based programs, as they typically require more instructions to be executed.

4. Not suitable for all types of problems: There are problems that lend themselves well to functional-programming style, logic-programming style, or procedure-based programming style, and applying object-oriented programming in those situations will not result in efficient programs.

The core of the pure object-oriented programming is to create an object, in code, that has certain properties and methods. While designing C++ modules, we try to see the whole world in the form of objects. For example, a car is an object which has certain properties such as color, the number of doors, and the like. It also has certain methods such as accelerate, brake, and so on.

7 . Characteristics Of Object-Oriented Programming:

1) **Object**
This is the basic unit of object oriented programming. That is both data and function that operate on data are bundled as a unit called as an object.

2) **Class**

When you define a class, you define a blueprint for an object. This doesn't define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

3) **Abstraction**
   Data abstraction refers to, providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.
   For example, a database system hides certain details of how data is stored and created and maintained. Similar way, C++ classes provide different methods to the outside world without giving internal detail about those methods and data.

4) **Encapsulation**
   Encapsulation is placing the data and the functions that work on that data in the same place. While working with procedural languages, it is not always clear which functions work on which variables but object-oriented programming provides you a framework to place the data and the relevant functions together in the same object.
5) **Inheritance**
   One of the most useful aspects of object-oriented programming is code reusability. As the name suggests Inheritance is the process of forming a new class from an existing class that is from the existing class called as a base class, a new class is formed called as derived class.
   This is a very important concept of object-oriented programming since this feature helps to reduce the code size.
6) **Polymorphism**
   The ability to use an operator or function in different ways in other words giving different meaning or functions to the operators or functions is called polymorphism. Poly refers to many. That is a single function or an operator functioning in many ways different upon the usage is called polymorphism.
7) **Overloading**
   The concept of overloading is also a branch of polymorphism. When the exiting operator or function is made to operate on new data type, it is said to be overloaded.

<u>8 Object Model</u>

An object model is a logical interface, software or system that is modeled through the use of object-oriented techniques. It enables the creation of an architectural software or system model prior to development or programming.

An object model is part of the object-oriented programming (OOP) lifecycle.

An object model helps describe or define a software/system in terms of objects and classes. It defines the interfaces or interactions between different models, inheritance, encapsulation and other object-oriented interfaces and features.

<u>8.1 Object model examples include</u>:

- **Document Object Model (DOM):** A set of objects that provides a modeled representation of dynamic HTML and XHTML-based Web pages

- **Component Object Model (COM)**: A proprietary Microsoft software architecture used to create software components

9 Primitive Built-In Types

C++ offers the programmer a rich assortment of built-in as well as user defined data types. Following table lists down seven basic C++ data types:

Keyword	Type
Bool	Boolean
Char	Character
Int	Integer
Float	Floating point
Double	Double floating point
Void	Valueless
wchar_t	Wide character

9.1 Types of Modifier:-

Several of the basic types can be modified using one or more of these type modifiers:

- signed
- unsigned
- short
- long

9.2 Data type and with range:-

The following table shows the variable type, how much memory it takes to store the value in memory and what is a maximum and minimum value which can be stored in such type of variables.

Typical Range	Typical Bit Width	Type
-128 to 127 or 0 to 255	1byte	char
0 to 255	1byte	unsigned char
-128 to 127	1byte	signed char
-2147483648 to 2147483647	4bytes	int
0 to 4294967295	4bytes	unsigned int

-2147483648 to 2147483647	4bytes	signed int
-32768 to 32767	2bytes	short int
0 to 65,535	2bytes	unsigned short int
-32768 to 32767	2bytes	signed short int
-2,147,483,648 to 2,147,483,647	8bytes	long int
-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8bytes	signed long int
0 to 18,446,744,073,709,551,615	8bytes	unsigned long int
+/- 3.4e +/- 38 (~7 digits)	4bytes	float
+/- 1.7e +/- 308 (~15 digits)	8bytes	double
+/- 1.7e +/- 308 (~15 digits)	8bytes	long double
1 wide character	2 or 4 bytes	wchar_t

The sizes of variables might be different from those shown in the above table, depending on the compiler and the computer you are using.

9.3 Variable Declaration in C++

A variable declaration provides assurance to the compiler that there is one variable existing with the given type and name so that compiler proceed for further compilation without needing complete detail about the variable. A variable declaration has its meaning at the time of compilation only, compiler needs actual variable definition at the time of linking of the program.

Example

Try the following example where a variable has been declared at the top, but it has been defined inside the main function:

```
#include <iostream>
using namespace std;

// Variable declaration:
externint a, b;
externint c;
extern float f;

int main () {
  // Variable definition:
```

```
int a, b;
int c;
float f;

  // actual initialization
  a = 10;
  b = 20;
  c = a + b;

cout<< c <<endl ;

  f = 70.0/3.0;
cout<< f <<endl ;

return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
30
23.3333
```

We will learn what a function is and it's parameter in subsequent chapters. Here let us explain what local and global variables are.

1. **Local Variables**

Variables that are declared inside a function or block are local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. Following is the example using local variables:

```
#include <iostream>
using namespace std;

int main () {
  // Local variable declaration:
int a, b;
int c;

  // actual initialization
  a = 10;
  b = 20;
  c = a + b;

cout<< c;
return 0;
}
Output
30
```

2. Global Variables

Global variables are defined outside of all the functions, usually on top of the program. The global variables will hold their value throughout the life-time of your program.
A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. Following is the example using global and local variables:

```
#include <iostream>
using namespace std;

// Global variable declaration:
int g;

int main () {
  // Local variable declaration:
int a, b;

  // actual initialization
  a = 10;
  b = 20;
  g = a + b;

cout<< g;

return 0;
}
Output
30
```

10 C++ Arrays

In programming, one of the frequently arising problem is to handle numerous data of same type.
Consider this situation, you are taking a survey of 100 people and you have to store their age. To solve this problem in C++, you can create an integer array having 100 elements.
An array is a collection of data that holds fixed number of values of same type. For example:

```
int age[100];
```

Here, the *age* array can hold maximum of 100 elements of integer type.
The size and type of arrays cannot be changed after its declaration.

DataTypearrayName [arraySize];

float mark[5];

C++ program to store and calculate the sum of 5 numbers entered by the user using arrays.

```
#include <iostream>
using namespace std;

int main()
{
   int numbers[5], sum = 0;
cout<< "Enter 5 numbers: ";

   //  Storing 5 number entered by user in an array
   //  Finding the sum of numbers entered
   for (int i = 0; i< 5; ++i)
   {
cin>> numbers[i];
      sum += numbers[i];
   }

cout<< "Sum = " << sum <<endl;

   return 0;
}
```

11 C++ Structures

Structure is a collection of variables of different data types under a single name. It is similar to a class in that, both holds a collection of data of different data types.

For example: You want to store some information about a person: his/her name, citizenship number and salary. You can easily create different variables *name, citNo, salary* to store these information separately.
However, in the future, you would want to store information about multiple persons. Now, you'd need to create different variables for each information per person: *name1, citNo1, salary1, name2, citNo2, salary2*
You can easily visualize how big and messy the code would look. Also, since no relation between the variables (information) would exist, it's going to be a daunting task.
The struct keyword defines a structure type followed by an identifier (name of the structure).

struct Person

```
{
   char name[50];
   int age;
   float salary;
};
```

C++ Program to assign data to members of a structure variable and display it.

```
#include <iostream>
using namespace std;

struct Person
{
   char name[50];
   int age;
   float salary;
};

int main()
{
   Person p1;

cout<< "Enter Full name: ";
cin.get(p1.name, 50);
cout<< "Enter age: ";
cin>> p1.age;
cout<< "Enter salary: ";
cin>> p1.salary;

cout<< "\nDisplaying Information." <<endl;
cout<< "Name: " << p1.name <<endl;
cout<<"Age: " << p1.age <<endl;
cout<< "Salary: " << p1.salary;

   return 0;
}
```

……………………………………………….End of Unit 1………………………………………………………..