# MIT World Peace University

Object Oriented Programming with Java and C++
Second Year B. Tech, Semester 1

---

# Theory Assignment

---

## Assignment No. 1

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A2, PA 20

October 9, 2022

# 1 Questions

## 1.1 Explain the Various Features of Object Oriented Programming, and also note down its applications in various domains.

1. **Encapsulation Enforces Modularity**
   Encapsulation refers to the creation of self-contained modules that bind processing functions to the data. These user-defined data types are called "classes," and one instance of a class is an "object." For example, in a payroll system, a class could be Manager, and Pat and Jan could be two instances (two objects) of the Manager class. Encapsulation ensures good code modularity, which keeps routines separate and less prone to conflict with each other.

2. **Inheritance Passes "Knowledge" Down**
   Classes are created in hierarchies, and inheritance allows the structure and methods in one class to be passed down the hierarchy. That means less programming is required when adding functions to complex systems. If a step is added at the bottom of a hierarchy, only the processing and data associated with that unique step needs to be added. Everything else is inherited. The ability to reuse existing objects is considered a major advantage of object technology.

3. **Polymorphism Takes any Shape**
   Object-oriented programming allows procedures about objects to be created whose exact type is not known until runtime. For example, a screen cursor may change its shape from an arrow to a line depending on the program mode. The routine to move the cursor on screen in response to mouse movement would be written for "cursor," and polymorphism allows that cursor to take on whatever shape is required at runtime. It also allows new shapes to be easily integrated.

4. **Data Abstraction Hides the Unneccessory**
   Abstraction refers to the user's interaction with just a subset of an object's characteristics and operations. To access a complicated item, abstraction uses simpler, high-level techniques.

   Simple items are used to show complexity. Keep complicated information hidden from the user. Simple classes are used to indicate complexity in abstraction. Encapsulation is an extension of abstraction.

**Some Applications of OOP Languages in Various Domains are given below**

1. **Real Time Systems** The term "real-time system" refers to any information processing system with hardware and software components that perform real-time application functions and can respond to events within predictable and specific time constraints. Using Object-oriented technology, we can develop real-time systems, this will offer adaptability, ease of modifications, reusability for the code. There is a lot of complexity involved in designing real-time systems, OOP techniques make it easier to handle those complexities.

2. **Client Server System**

   The client-server systems are those that involve a relationship between cooperating programs in an application. In general, the clients will initiate requests for services and the servers will provide that functionality. The client and server either reside in the same system or communicate with each other through a computer network or the internet.

3. **Object Oriented Database**

   Nowadays each and every data is being stored and processed, the traditional model of storing data i.e the relational model stores each and every piece of data in tables that consist of rows and columns. However as complexity grows, storing in the form of tables becomes quite cumbersome, here the need for storing in the form of real-world objects comes into the picture. These databases try to maintain a direct correspondence between the real-world and database objects in order to let the object retain its identity and integrity. They can then be identified and operated upon.

   A popular example of object-oriented databases is **MongoDB**.

4. **Neural Networks and Parallel Programming** A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature.

5. **Simulation and Modeling System**

   A good example of a simulation and modeling system is of automobiles such as cars, Once the model of the car structure is developed by the engineer's team, as and when they feel that the product is good to go they can release the product. OOP provides an appropriate approach for simplifying these complex models.

6. **CIM/CAD/CAM Systems** OOP can also be used in manufacturing and designing applications as it allows people to reduce the efforts involved. For instance, it can be used while designing blueprints and flowcharts. So it makes it possible to produce these flowcharts and blueprint accurately.

7. **Computer Aided Designs**

   As per Wikipedia, Computer-aided design (CAD) is the use of computers (or workstations) to aid in the creation, modification, analysis, or optimization of a design. In mechanical design, it is known as mechanical design automation (MDA), which includes the process of creating a technical drawing with the use of computer software.

   One of the good examples of computer-aided design is **MATLAB**. It is used by the programmers to solve difficult mathematical models, these models are then used in bigger system designs to check whether the system will function as expected or not

## 1.2 Explain difference between compile time polymorphism and run time polymorphism.

### Compile Time Polymorphism

In compile-time polymorphism, a function is called at the time of program compilation. We call this type of polymorphism as early binding or Static binding. Function overloading and operator overloading is the type of Compile time polymorphism. It can be implemented in 2 simple ways in c++

1. **Function Overloading** :
   Function overloading means one function can perform many tasks. In C++, a single function is used to perform many tasks with the same name and different types of arguments. In the

function overloading function will call at the time of program compilation. It is an example of compile-time polymorphism.

```
class Addition {
public:
  int ADD(int X,int Y)   // Function with parameter
  {
    return X+Y;     // this function is performing addition of  two Integer value
  }
  int ADD() {              // Function with same name but without parameter
    string a= "HELLO";
    string b="SAM";   // in this function concatenation is performed
  string c= a+b;
  cout<<c<<endl;

  }
};
int main(void) {
  Addition obj;   // Object is created
  cout<<obj.ADD(128, 15)<<endl; //first method is called
  obj.ADD();  // second method is called
  return 0;
}


```

2. **Operator Overloading**:
   Operator overloading means defining additional tasks to operators without changing its actual meaning. We do this by using operator function.
   ◦ The purpose of operator overloading is to provide a special meaning to the user-defined data types.
   The advantage of Operators overloading is to perform different operations on the same operand.

```
  #include <iostream>
  using namespace std;
  class A
  {

    string x;
    public:
    A(){}
    A(string i)
    {
    x=i;
    }
    void operator+(A);
    void display();
  };

  void A:: operator+(A a)
  {

    string m = x+a.x;
    cout<<"The result of the addition of two objects is : "<<m;

```

```
      }
      int main()
      {
        A a1("Welcome");
        A a2("back");
        a1+a2;
        return 0;
      }


```

**Run Time Polymorphism**

In Runtime polymorphism, functions are called at the time the program execution. Hence, it is known as late binding or dynamic binding.

Function overriding is a part of runtime polymorphism. In function overriding, more than one method has the same name with different types of the parameter list.

It is achieved by using virtual functions and pointers. It provides slow execution as it is known at the run time. Thus, It is more flexible as all the things executed at the run time.

In C++ it can be implemented using these 2 methods.

1. **Function Overriding:**
   In function overriding, we give the new definition to base class function in the derived class. At that time, we can say the base function has been overridden. It can be only possible in the 'derived class'. In function overriding, we have two definitions of the same function, one in the superclass and one in the derived class. The decision about which function definition requires calling happens at runtime. That is the reason we call it 'Runtime polymorphism'.

```

// Java program to demonstrate
// runtime polymorphism

// Implementing a class
class Test {

    // Implementing a method
    public void method()
    {
        System.out.println("Method 1");
    }
}

// Defining a child class
public class GFG extends Test {

    // Overriding the parent method
    public void method()
    {
        System.out.println("Method 2");
    }

    // Driver code
    public static void main(String args[])
    {
        Test test = new GFG();

```

```
        test.method();
    }
}

```

2. **Virtual Functions:**
   A virtual function is declared by keyword virtual. The return type of virtual function may be int, float, void.

   A virtual function is a member function in the base class. We can redefine it in a derived class. It is part of run time polymorphism. The declaration of the virtual function must be in the base class by using the keyword virtual. A virtual function is not static.

   The virtual function helps to tell the compiler to perform dynamic binding or late binding on the function.

   If it is necessary to use a single pointer to refer to all the different classes' objects. This is because we will have to create a pointer to the base class that refers to all the derived objects.

```
#include <iostream>
using namespace std;
class Base
{
public:
  virtual void show_val()
  {
    cout << "Class::Base";
  }
  };
  class Derived : public Base
  {
    public:
  void show_val()
  {
    cout << "Class::Derived";
    }
    };
    int main()
    {
      Base *b; // Base class pointer Derived d; //Derived class object b = &d;
    b->show_val();   //late Binding
}
```

### 1.3 Create a Java program to find the factorial of a number using default constructor and parameterized constructor.

```

// Code to find factorial by using default and parameterized constructor.
import java.util.*;

class Factorial {
    int n = 0;
    Scanner input = new Scanner(System.in);

    Factorial() {
        System.out.println("Enter a Number whose factorial you want to find: ");
        this.n = input.nextInt();
```

```
    }

    Factorial(int n) {
        this.n = n;
    }

    int factorial() {
        int res = 1, i;
        for (i = 2; i <= this.n; i++)
            res *= i;
        return res;
    }
}

class HelloWorld {
    // Driver Code
    public static void main(String args[]) {
        Factorial para = new Factorial(5);
        Factorial def = new Factorial();

        System.out.println("(Parameterized) Factorial : " +
                "  is " + para.factorial());
        System.out.println("\n(Default) Factorial :   " +
                "  is " + def.factorial());

    }
}
```

```
Enter a Number whose factorial you want to find: 6
(Parameterized) Factorial :  is 120
(Default) Factorial :   is 720
```

### 1.4 Create a C++ program that creates a class "Arithmatic" which contains integer data members. Overload all the four arthmetic operators so that they operate on the objects of "Arthmatic".

```
// Create a C++ program that creates a class "Arithmatic" which contains integer
   data members. Overload all the four arthmetic operators so that they operate
  on the objects of "Arthmatic".

#include <iostream>
using namespace std;

class Arithmatic
{
public:
  int x, y;
  Arithmatic(int xx = 0, int yy = 0)
  {
    x = xx;
    y = yy;
  }

  Arithmatic operator+(Arithmatic const &obj)
  {
    Arithmatic res;
```

```
      res.x = x + obj.x;
      res.y = y + obj.y;
      return res;
    }
    Arithmatic operator-(Arithmatic const &obj)
    {
      Arithmatic res;
      res.x = x - obj.x;
      res.y = y - obj.y;
      return res;
    }
    Arithmatic operator/(Arithmatic const &obj)
    {
      Arithmatic res;
      res.x = x / obj.x;
      res.y = y / obj.y;
      return res;
    }
    Arithmatic operator*(Arithmatic const &obj)
    {
      Arithmatic res;
      res.x = x * obj.x;
      res.y = y * obj.y;
      return res;
    }
    void print()
    {
      cout << x << endl;
      cout << y << endl;
    }
  };

  int main()
  {
    Arithmatic a(1, 2), b(2, 3);
    Arithmatic res;
    res = a + b;
    res.print();
    res = a - b;
    res.print();
    res = a / b;
    res.print();
    res = a * b;
    res.print();
    return 0;
  }

```

```
3
5
-1
-1
0
0
2
6
```