**OOPs**

**What is meant by the term OOPs?**

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic.

**What is the need for OOPs?**

Solving a problem with the help of real world objects this approach leads us to Object Oriented Programming.

Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

OOP is faster and easier to execute. It provides a clear structure for the programs. OOP helps to keep the C++ code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug. OOP makes it possible to create full reusable applications with less code and shorter development time.

**What is Structured Programming?**

It is a programming style which aims to improve the quality clarity development time to make use of block structure. Better way to program as it involve systematic organization of program.

It is basically a subset of procedural programs.

In this, programmers are allowed to code a program simply by dividing the program into modules or smaller units.

It does not use GOTO to control the flow of execution. Instead, it uses loops

The GOTO statement gives the power to jump to any part of a program but, makes the logic of the program complex and tangled.

In modern programming, the goto statement is considered a harmful construct and a bad programming practice.

The GOTO statement can be replaced in most of C++ program with the use of break and continue statements.

**What are the main features of OOPs?**

* Encapsulation
* Inheritance
* Abstraction
* Polymorphism

**What is a class?**

A class is a blueprint of an object It is a template where we define an object. A class has a scope where we define functions.

**What is an object?**

Object is an instance of a class. It has its own state behavior and identity.

**Explain encapsulation with example?**

// Program to calculate the area of a rectangle

#include <iostream>

using namespace std;

class Rectangle {

public:

// Variables required for area calculation

int length;

int breadth;

// Constructor to initialize variables

Rectangle(int len, int brth) : length(len), breadth(brth) {}

// Function to calculate area

int getArea() {

return length \* breadth;

}

};

int main() {

// Create object of Rectangle class

Rectangle rect(8, 6);

// Call getArea() function

cout << "Area = " << rect.getArea();

return 0;

}

**Explain Polymorphism with example?**

Polymorphism is nothing but assigning some values to the sub class to something that is already declare in the main class.

**Function Overloading**

// C++ program to overload sum() function

#include <iostream>

using namespace std;

// Function with 2 int parameters

int sum(int num1, int num2) {

return num1 + num2;

}

// Function with 2 double parameters

double sum(double num1, double num2) {

return num1 + num2;

}

// Function with 3 int parameters

int sum(int num1, int num2, int num3) {

return num1 + num2 + num3;

}

int main() {

// Call function with 2 int parameters

cout << "Sum 1 = " << sum(5, 6) << endl;

// Call function with 2 double parameters

cout << "Sum 2 = " << sum(5.5, 6.6) << endl;

// Call function with 3 int parameters

cout << "Sum 3 = " << sum(5, 6, 7) << endl;

return 0;

}

**Operator Overloading**

We cannot use operator overloading for basic types such as int, double, etc.

Operator overloading is basically function overloading, where different operator functions have the same symbol but different operands.

And, depending on the operands, different operator functions are executed. For example,

// C++ program to overload ++ when used as prefix

#include <iostream>

using namespace std;

class Count {

private:

int value;

public:

// Constructor to initialize count to 5

Count() : value(5) {}

// Overload ++ when used as prefix

void operator ++() {

value = value + 1;

}

void display() {

cout << "Count: " << value << endl;

}

};

int main() {

Count count1;

// Call the "void operator ++()" function

++count1;

count1.display();

return 0;

}

Here, we have overloaded the ++ operator, which operates on objects of Count class (object count1 in this case).

We have used this overloaded operator to directly increment the value variable of count1 object by 1.

This is also a compile-time polymorphism.

**Virtual Functions**

In C++, we may not be able to override functions if we use a pointer of the base class to point to an object of the derived class.

Using virtual functions in the base class ensures that the function can be overridden in these cases.

Thus, virtual functions actually fall under function overriding. For example,

#include <iostream>

using namespace std;

class Base {

public c:

virtual void print() {

cout << "Base Function" << endl;

}

};

class Derived : public Base {

public:

void print() {

cout << "Derived Function" << endl;

}

};

int main() {

Derived derived1;

// pointer of Base type that points to derived1

Base\* base1 = &derived1;

// calls member function of Derived class

base1->print();

return 0;

}

This is also a Run Time Polymorphism.

**Function Overriding**

When we call the function using an object of the derived class, the function of the derived class is executed instead of the one in the base class.

So, different functions are executed depending on the object calling the function.

This is known as function overriding in C++. For example,

// function overriding

#include <iostream>

using namespace std;

class Base {

public:

virtual void print() {

cout << "Base Function" << endl;

}

};

class Derived : public Base {

public:

void print() {

cout << "Derived Function" << endl;

}

};

int main() {

Derived derived1;

// Call print() function of Derived class

derived1.print();

return 0;

}

It's a runtime polymorphism because the function call is not resolved by the compiler, but it is resolved in the runtime instead.

**Explain Inheritance with example?**

Inheritance is a mechanism that allows one class to gain the properties of another class, in the same way, that a child inherits some attributes from each of their parents. Inheritance allows programmers to create a new class that reuses the data members and methods of an existing class.

// C++ program to demonstrate inheritance

#include <iostream>

using namespace std;

// base class

class Animal {

public:

void eat() {

cout << "I can eat!" << endl;

}

void sleep() {

cout << "I can sleep!" << endl;

}

};

// derived class

class Dog : public Animal {

public:

void bark() {

cout << "I can bark! Woof woof!!" << endl;

}

};

int main() {

// Create object of the Dog class

Dog dog1;

// Calling members of the base class

dog1.eat();

dog1.sleep();

// Calling member of the derived class

dog1.bark();

return 0;

}

**Does multiple inheritance support by PHP?**

PHP support single inherence means a child class can only be inherit from a single parent class. PHP doesn’t support multiple inherence but we can use this functionality by using of interface & by using of traits instead of classes we can use it. Traits are used to declare methods that can be used in multiple classes

**Are there any limitations of Inheritance?**

Main disadvantage of using inheritance is that two classes are tightly coupled so that they cannot be used independently.

**Explain Abstraction?**

It is a process of hiding implementation details and only showing the functionality to the user. Abstraction focus on what the object does instead of how it does. It is achieved by using Abstract class and Interface.

**S**ome **D**ifferences **b**etween **A**bstraction & **E**ncapsulation

Abstraction in OOP solves the issue at design level while Encapsulation solve it at implementation level. Abstraction in Programming is about hiding unwanted details while showing most essential information. Encapsulation means binding the code and data into a single unit.

**Explain constructor?**

Constructor is a method that is invoked automatically at the time of creation of object. There are three types of constructor. The name of the constructed is the same as the name of the class. There is no return type of a constructor.

* Default constructor
* Parameterized Constructor
* Copy constructor
* Private constructor

**Explain destructor?**

Destructor is a method that is invoked automatically at the time of deletion of an object. The name of the destructor is same as the name of the class but there is a titled nimble.

**What are the various types of inheritance?**

* Single inheritance
* Multiple Inheritance
* Multi-Level Inheritance
* Hierarchical Inheritance.
* Hybrid Inheritance.
* Multipath inheritance.

**What is a subclass?**

A subclass is a child the class that inherits from another class

**What is a superclass?**

A superclass is a parent class the class being inherited from

final class Vehicle {

    ...

  }

  class Car extends Vehicle {

    ...

  }

**What is an interface?**

The interface can have only abstract methods. It supports multiple inheritances.   
It cannot provide the implementation of the abstract class. An interface can have only have public abstract methods. The interface can only have a public static final variable.

**Difference between overloading and overriding explain with example?**

**Method Overloading**

Method overloading is providing two separate methods in a class with the same name but different arguments, while the method return type may or may not be different, which allows us to reuse the same method name.

**Method Overriding**

Overriding means same method name and same parameter, occur in different class that has inheritance relationship.

we use method overriding to implement specific functionality to the method.

1) Method Overriding occurs between two classes superclass and subclass

2) Since method overriding occurs between superclass & subclass inheritance is involved.

3) In overriding return type must be same.

4) Parameters must be same.

**What is an abstract class?**

Abstraction occurs when a programmer hides any irrelevant data about an object or an instantiated class to reduce complexity and help users interact with a program more efficiently.

Think about the interface of your mobile phone. Whether you use an Android operating system or iOS, you don't interact directly with the code that allows your phone to connect to the internet, send a text message or play a video game. Instead, you interact with the code through a user interface that is designed to streamline your experience and make it easy to access the functions and methods you need to complete a task. In this case, the interface is abstracted away from the actual implementation of the code.

**How is an abstract class different from an interface explain with example?**

**Abstract Class**

1. Abstract class can have abstract and non-abstract methods.
2. Abstract class doesn't support multiple inheritance.
3. Abstract class can have final, non-final, static and non-static variables.
4. Abstract class can provide the implementation of interface.
5. The abstract keyword is used to declare abstract class.

**Interface**

1. Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2. Interface supports multiple inheritance.
3. Interface has only static and final variables.
4. Interface can't provide the implementation of abstract class.
5. The interface keyword is used to declare interface.

**What is meant by Garbage Collection in OOPs world?**

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.

It is automatically done by the garbage collector(a part of JVM) so we don't need to make extra efforts.