Object Oriented Programming

- **Object:**

An object is an instance of a class. It represents a real-world entity with attributes (data) and behaviors (methods). For example, a "Car" object might have attributes like color and speed and methods like drive() or stop().

- **Class:**

A class is a blueprint or template for creating objects. It defines the attributes (data members) and methods (functions) that objects of the class will have.

- **Method:**

A method is a function defined within a class that operates on objects of that class. It represents the behavior of an object.

- **Encapsulation:**

Encapsulation is the concept of bundling data (attributes) and methods (functions) into a single unit, i.e., a class, and restricting access to some components using access specifiers (private, protected, public). It ensures data security and hides implementation details.

- **Data Abstraction:**

Data abstraction is the process of exposing only the essential features of an object while hiding the details of its implementation. It allows users to focus on what an object does instead of how it does it.

- **Inheritance:**

Inheritance is the mechanism in which one class (child/derived class) acquires the properties and behaviors of another class (parent/base class). It enables code reuse and the creation of hierarchical relationships between classes.

- **Polymorphism:**

Polymorphism means "many forms." In OOP, it allows objects of different classes to be treated as objects of a common base class. It is achieved through method overloading (compile-time polymorphism) and method overriding (runtime polymorphism).

- **Compile-Time Polymorphism**:

A type of polymorphism resolved at compile time, achieved through method overloading or operator overloading.

- **Runtime Polymorphism**:

A type of polymorphism resolved at runtime, achieved through method overriding using inheritance and virtual functions.

- **Data Binding:**

Data binding is the process of linking a function call to the method body. It determines which method to invoke based on the binding mechanism.

- **Static Binding:**

Static binding (also known as early binding) occurs at compile-time. It is used in cases like function overloading or accessing private methods. The compiler decides which method to call based on the type of reference.

- **Dynamic Binding:**

Dynamic binding (also known as late binding) occurs at runtime. It is used in cases like method overriding in inheritance. The decision of which method to call is made at runtime based on the object type.

● Overloading is static Binding, whereas Overriding is dynamic Binding. Overloading is nothing but the same method with different arguments, and it may or may not return the same value in the same class itself. Overriding is the same method name with the same arguments and return types associated with the class and its child class.

- **Constructor:**

A special member function in a class that is automatically called when an object of the class is created. Its purpose is to initialize the object. Constructors have the same name as the class and do not have a return type.

- **Destructor:**

A special member function in a class that is automatically called when an object goes out of scope or is explicitly deleted. It is used to release resources acquired by the object. Destructors have the same name as the class, preceded by a tilde (~), and they do not have a return type.

- **Access Modifiers**:

Keywords used in object-oriented programming to set access levels for class members. The common access modifiers are public, private, and protected.

- **Pointers:**

Variables that store the memory address of another variable. Pointers are used to dynamically allocate memory, pass arguments by reference, and create complex data structures like linked lists.

- **Operator Overloading:**

A feature in C++ that allows you to redefine the way operators work for user-defined types. It enables custom behavior for operators like +, -, *, etc., when used with objects of a class.

- **Function Overloading:**

A feature in C++ (and other programming languages) that allows multiple functions to have the same name but with different parameter lists. The functions must differ in the number, type, or order of parameters. Function overloading enables the same function name to perform different tasks based on the arguments passed.

- **Abstract Class:**

A class that cannot be instantiated on its own and is designed to be a base class for other classes. It typically contains at least one pure virtual function, which must be overridden by derived classes.

- **Virtual Function:**

A member function in a base class that can be overridden in derived classes. It enables runtime polymorphism, allowing the correct function to be called for an object, regardless of the type of reference (or pointer) used for the call.

- **Friend Class:**

A class that is granted access to the private and protected members of another class. This is specified using the friend keyword in the class declaration.

- **Exception Handling:**

A mechanism to handle runtime errors in a program using constructs like try, catch, and throw. It ensures that a program can respond to errors gracefully without crashing.

- **Static Variable (Local Scope):**

A variable declared with the static keyword inside a function retains its value between function calls. Unlike regular local variables, it is initialized only once and persists throughout the program's lifetime.

- **Static Variable (Class Scope):**

A static member variable in a class is shared among all objects of the class. It belongs to the class rather than any specific object.

- **Static Function (Class Scope):**

A static member function in a class can be called without creating an object of the class. It can only access static data members or other static member functions.

- **Method Overloading**:

A feature that allows multiple methods in the same class to have the same name but different parameter lists (number, type, or order).

- **This Pointer**:

A special pointer available within a class's non-static member functions that refers to the calling object itself.

- **Virtual Function**:

A member function in a base class that can be overridden in a derived class, enabling runtime polymorphism.

- **Virtual inheritance**

Virtual inheritance facilitates you to create only one copy of each object even if the object appears more than one in the hierarchy

❖ Difference Between POP and OOP.

Aspect	Procedural-Oriented Programming (POP)	Object-Oriented Programming (OOP)
Basic Concept	Program is divided into procedures or functions.	Program is divided into objects containing data and methods.
Focus	Focuses on functions and the sequence of tasks.	Focuses on data encapsulation and objects.
Data Handling	Data is globally accessible; less secure.	Data is encapsulated within objects; more secure.
Code Reusability	Limited reusability; functions are standalone.	High reusability through inheritance and polymorphism.
Examples of Languages	C, Fortran, Pascal.	C++, Java, Python, Kotlin.
Encapsulation	Not supported inherently; data and functions are separate.	Supported; combines data and methods into objects.
Inheritance	Not supported.	Supported; allows code reuse and hierarchical relationships.
Polymorphism	Not supported.	Supported; methods can behave differently based on context.

❖ Difference Between C and C++.

Aspect	C	C++
Programming Paradigm	Procedural programming language.	Object-oriented programming language (with procedural support).
Focus	Focuses on functions and procedures.	Focuses on objects and classes.
Encapsulation	Not supported; data and functions are separate.	Supported; combines data and methods into objects.
Inheritance	Not supported.	Supported; allows classes to inherit properties.
Polymorphism	Not supported.	Supported; allows method overriding and overloading.
Exception Handling	Not supported.	Supported through try, catch, and throw blocks.
Memory Management	Done manually using functions like malloc and free.	Done manually or using constructors, destructors, and smart pointers.
Standard Library	Limited standard library (mostly focused on functions).	Rich standard library (includes STL for data structures and algorithms).
Data Security	Less secure; no direct support for data hiding.	More secure; supports data hiding via access specifiers (private, protected, public).
Applications	Suitable for system-level programming (e.g., OS, embedded systems).	Suitable for system-level programming as well as application development.
Namespace	Not supported; risk of name collisions.	Supported through the namespace feature.
File Extension	.c for source files.	.cpp for source files.
Examples of Usage	Operating systems, embedded systems, compilers.	GUI applications, game development, simulations.

❖ Difference Between C++ vs Java.

Aspect	C++	Java
Programming Paradigm	Object-oriented programming with procedural support.	Purely object-oriented (except for primitive types).
Platform Dependency	Platform-dependent; compiled code runs only on the target OS.	Platform-independent due to the Java Virtual Machine (JVM).
Speed	Generally faster due to direct machine code execution.	Slower compared to C++ due to JVM overhead.
Memory Management	Manual, using malloc/free or new/delete.	Automatic, using garbage collection.
Pointers	Fully supported; allows direct memory manipulation.	Not supported; memory is managed indirectly through references.
Multiple Inheritance	Supported (with ambiguity resolution using virtual inheritance).	Not directly supported (achieved through interfaces).
Operator Overloading	Supported; developers can define custom behavior for operators.	Not supported.
Standard Library	Rich libraries but less uniform than Java.	Extensive standard library with uniform APIs (e.g., Java Standard API).
Security	Less secure; relies on developer practices for security.	More secure; JVM provides built-in security features (e.g., sandboxing).
Portability	Less portable; platform-specific implementations are common.	Highly portable; write once, run anywhere (with JVM).
File Extension	.cpp for source files.	.java for source files.
Examples of Usage	Game engines (e.g., Unreal), operating systems, high-performance apps.	Android development, web applications, enterprise software (e.g., Spring).

- ## Types of inheritance in C++:

Type of Inheritance	Description	Example
Single Inheritance	A class inherits from one base class.	class B : public A { };
Multiple Inheritance	A class inherits from more than one base class.	class C : public A, public B { };
Multilevel Inheritance	A class inherits from another derived class (forming a chain).	class B : public A { }; class C : public B { };
Hierarchical Inheritance	Multiple derived classes inherit from a single base class.	class B : public A { }; class C : public A { };
Hybrid Inheritance	A combination of two or more types of inheritance (e.g., multiple and hierarchical).	class B : public A { }; class C : public A { }; class D : public B, public C { };
Multipath Inheritance	A derived class indirectly inherits from a base class through multiple paths (handled using virtual inheritance).	class A { }; class B : virtual public A { }; class C : virtual public A { }; class D : public B, public C { };

❖ New operator in C++ other than C.

1. Scope Resolution Operator (::)
2. Member Access Through Pointer to Member (->*)
3. Member Access Through Object to Member (.*)
4. New Operator (new)
5. Delete Operator (delete)
6. Typeid Operator (typeid)
7. Dynamic Cast Operator (dynamic_cast)
8. Static Cast Operator (static_cast)
9. Reinterpret Cast Operator (reinterpret_cast)
10. Const Cast Operator (const_cast)
11. Overloaded Operators (Defined by the user, e.g., +, -, ==, etc.)