Notes for OOP

Object-Oriented Programming (OOP)

▼ OOP focuses on objects and their interactions.

Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects.

- **Key principles:**

1.Encapsulation (Privacy block of code within a class)

2.Abstruction (Hiding complex things behind the procedure to make things look simple)

3.Inheritance(PARENT and CHILD class, Similarities and Differences)

4.Polymorphism(many forms, same function names but performing diff tasks within a class through parameters types, num of parameters & return types)

Class & Object Terminology

▼ Class: Blueprint for creating objects.

Class is a user-defined data type which defines its properties and its functions. Class is the only logical representation of the data. For example, Human being is a class. The body parts of a human being are its properties, and the actions performed by the body parts are known as functions. The class does not occupy any memory space till the time an object is instantiated.

▼ Object: An instance of a class.

Object is a run-time entity. It is an instance of the class. An object can represent a person, place or any other item. An object can operate on both data members and member functions.

- Methods:Functions within a class

Types, Overloading, and References

- Data types: int, float, char, etc.
- Function overloading: Same name, different parameters.
- References: Alias for a variable.

Dynamic Memory

- `new` and `delete` operators allocate and deallocate memory.

▼ Helps manage memory at runtime.

When an object is created **using** a new keyword, then space is allocated for the variable in a heap, and the starting address is stored in the stack memory.

Access Modifiers

The C++ language supports three modifiers for granting access to the members of class:

- `Private`: Bars all access
- `Protected`: Limits access to derived classes only
- `Public`: Unlimited access

Construction and Destruction

- Constructors initialize objects.(no return type, same name as class, needs to be public)
- Default constructor, Parameterized constructor, Copy constructor >> shallow, deep copy
- Destructors clean up resources.

Constructor : Constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new objects generally.

- Constructors have the same name as class or structure.
- Constructors don’t have a return type. (Not even void)
- Constructors are only called once, at object creation.
- There can be **three types** of constructors.

- 1. Non-Parameterized constructor : A constructor which has no argument is known as non-parameterized constructor(or no-argument constructor). It is invoked at the time of creating an object. If we don’t create one then it is created by default.
- 2. Parameterized constructor : Constructor which has parameters is called a parameterized constructor. It is used to provide

different values to distinct objects.

- 3. Copy Constructor : A Copy constructor is an **overloaded**

constructor used to declare and initialize an object from another object. There is only a user defined copy constructor in Java(C++ has a default one too).

(Setter & Getter)

- S**etter Methods:** These methods are used to set the values of private members of a class. They allow controlled modification of the object's state.
- **Getter Methods:** These methods are used to retrieve the values of private members. They provide controlled access to the internal state of the object.

Encapsulation

- Encapsulation: Bundling data and methods that operate on the data.
- Hiding implementation details.
- Promotes information hiding.

Encapsulation

- Encapsulation is the process of combining data and functions into a single unit called class. In Encapsulation, the data is not accessed directly; it is accessed through the functions present inside the class. In simpler words, attributes of the class are kept private and public getter and setter methods are provided to manipulate these attributes. Thus, encapsulation makes the concept of data hiding possible.(**Data hiding**: a language feature to restrict access to members of an object, reducing the negative effect due to dependencies.

Member Operators

- Member access operators: `.` and `>`.

- Allow access to class members.

The Current Object

- `this` pointer refers to the current object.
- Used for disambiguating member variables.
- **'this' keyword :** 'this' keyword in that refers to the current instance of the class. In OOPS it is used to:

1. pass the current object as a parameter to another method
2. refer to the current class instance variable

Classes and Resources

- Classes group data and functions.
- Resource management within classes.

Helper Functions

- Functions assisting class operations.
- Can be friend functions or utility functions.

Input and Output Operators

- Overloading `>>` and `<<` for input and output.
- Customizing I/O for user-defined types.

Inheritance and Derived Classes

- Creating new classes from existing ones.
- Derived classes inherit properties.

Inheritance

Inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. In such a way, you can **reuse, extend or modify** the attributes and behaviors which are defined in other classes.

Functions in a Hierarchy

- Inherited functions in a class hierarchy.
- Overriding functions in derived classes.

Polymorphism

Overview

- Ability to take multiple forms.
- Dynamic polymorphism through virtual functions.
- **Polymorphism**

Polymorphism is the ability to present the same interface for differing underlying forms (data types). With polymorphism, each of these classes will have different underlying data. Precisely, Poly means ‘many’ and morphism means ‘forms’.

Types of Polymorphism IMP

1. Compile Time Polymorphism (Static)(Overloading)

2. Runtime Polymorphism (Dynamic)(Overridding)

- Method Overloading : Method overloading is a technique which allows you to have more than one function with the same function name but with different functionality.

Virtual Functions

- Functions declared in a base class.
- Overridden in derived classes for polymorphism.

Abstract

Base Classes (Hiding complex things behind the procedure to make things look simple)

- Classes with one or more pure virtual functions.
- Cannot be instantiated.
- it is hiding the unnecessary details & showing only the essential parts/functionalities to the user.

Templates

- Generic programming with templates.
- Function and class templates.

Input and Output Refinements

- Improving I/O for user-defined types.
- Overloading stream insertion and extraction operators.

Derived Classes and Resources Refinements

- Managing resources in derived classes.
- Refinements in resource handling.

Language Standards

- C++ evolves with standards (e.g., C++11, C++14).
- New features and improvements.

Appendices

- Standard Library Functions
- ASCII Collating Sequence
- Operator Precedence
- Relation between C++ and C

Additions

Base Class Copy Constructor:

- Default behavior is to perform a shallow copy.

Derived Class Copy Assignment Operator:

- Calls the base class's copy assignment operator.

Derived Class Copy Constructor:

- Calls the default constructor of the base class.

Shallow Copy Utility:

- Shallow copy is particularly useful for processing without altering the original source.

Operator Assignment or Copy Constructor Absence:

- Unexpected behavior may occur in classes without these.

Base Class Constructor Argument:

- In the call from the derived class, the argument is the derived class object.

Destructor of Derived Class:

- Automatically calls the destructor of the base class.

Child Class Constructor and Base Class Constructor:

- Each child class constructor calls the default constructor of the base class.

Copy Constructor in Base and Derived Classes:

- If the base class has a copy constructor but the derived class doesn't, it may lead to compilation errors.

Function Templates:

- While the statement "All function templates begin with the keyword class" is false, it's important to note that function templates typically begin with the keyword `template` followed by the template parameter list.
- The correct statement is that every formal type parameter is preceded by either the keyword `typename` or `class`.