coding75.com

# OOPs Notes

(Object Oriented Programming)

Available here: coding75.com

Join coding75 Pro 🚀

# Introduction to Object Oriented Programming

## What is Object oriented programming??

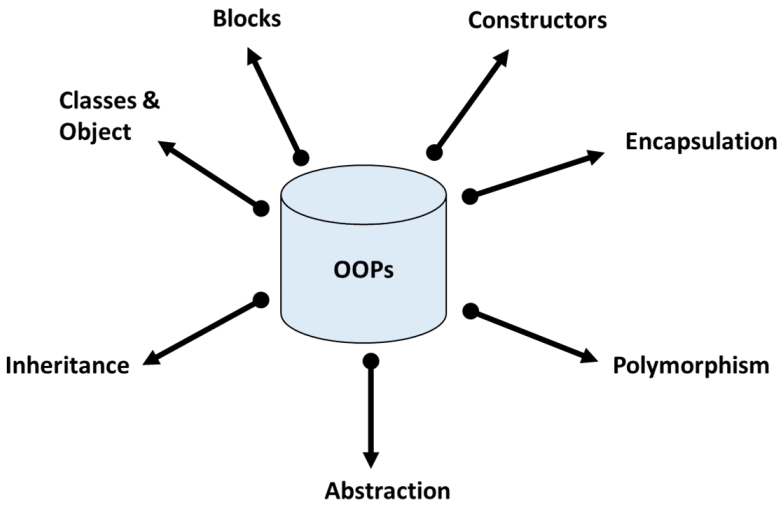
**Object-Oriented Programming (OOP)** is a **methodology or paradigm** for designing software through the utilization of **classes** and **objects.** It streamlines both software development and maintenance. OOPs consists of-

- **Class:** It is a blueprint or template for creating objects. It defines the attributes (data) and behaviors (methods) that objects of that class will have. Think of a class as a cookie cutter, and objects as the cookies made from that cutter.

  A class typically includes **attributes, methods, constructor** and **destructor.** (these are covered in detail below.)

- **Object:** an object is an instance of a class. It represents a specific realization of the attributes and behaviors defined by its class. Objects are created during run-time.

  An object includes **state, behavior** and its **identity (this keyword).**



Let's now see above mentioned parts of **classes** and **objects**:

## Constructor

A constructor is a unique method that is automatically called during the creation of an object. Its primary purpose is to initialize the data members of newly created objects. In C++, the constructor shares the same name as the class or structure it belongs to.

Constructor can be of three types-

1. **Default constructor:** A constructor without any arguments is referred to as a default constructor. It is automatically called during the creation of an object.
2. **Parameterized constructor:** A constructor with parameters is termed as a parameterized constructor. It is utilized to supply different values to individual objects.
3. **Copy constructor (Important):** A copy constructor is an overloaded constructor employed to declare and initialize an object based on another object. It comes in two forms: the default copy constructor provided by the language and the user-defined copy constructor.

### Learn more about constructors (examples using code):

C++: https://www.prepbytes.com/blog/cpp-programming/type-of-constructor-in-cpp/

Java: https://www.prepbytes.com/blog/cpp-programming/type-of-constructor-in-cpp/

Python: https://wiingy.com/learn/python/constructors-in-python/

## Destructor

A destructor in object-oriented programming (OOP) is a special member function of a class that is automatically called when an object is destroyed or goes out of scope. Its primary purpose is to perform cleanup tasks, such as

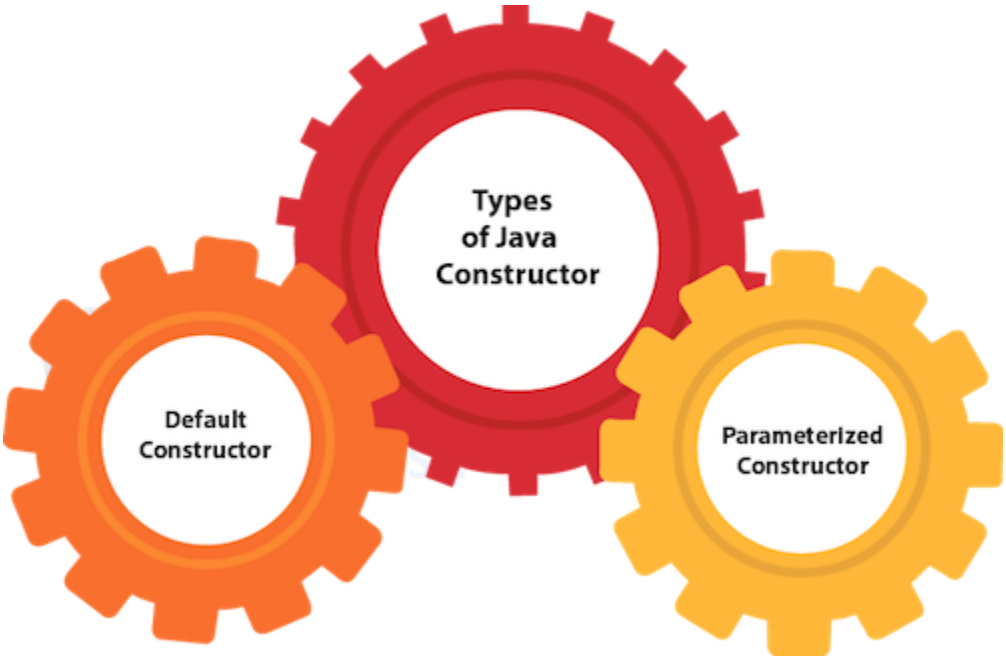
releasing resources (memory, file handles, database connections, etc.), that were allocated during the object's lifetime.

### Learn more about destructors (examples using code):

C++: https://www.algbly.com/Tutorials/Cpp-programming/Cpp-destructor.html

Java: https://www.codingninjas.com/studio/library/java-destructor

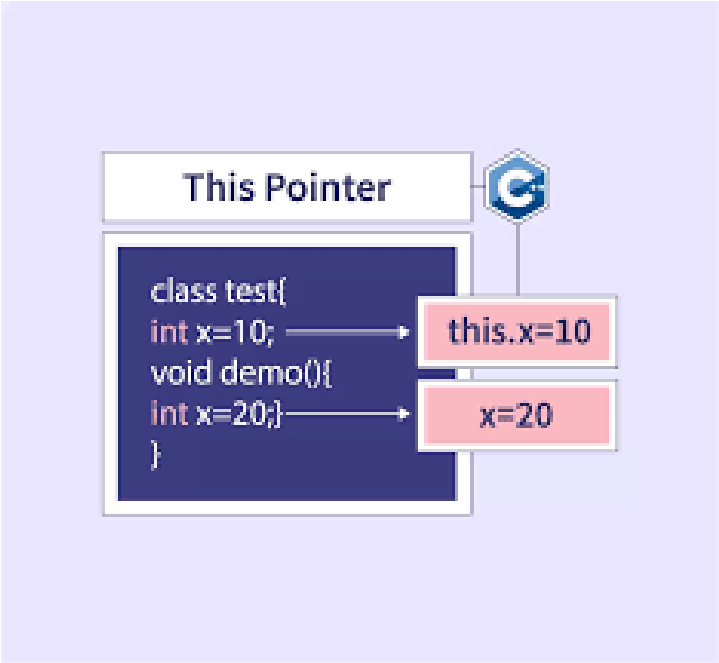
Python: https://www.scaler.com/topics/destructor-in-python/




## “This” keyword/pointer

The keyword "this" refers to the current instance of the class. It serves three main purposes:

1. It can pass the current object as a parameter to another method.
2. It can refer to the current class instance variable.
3. It can declare indexers.



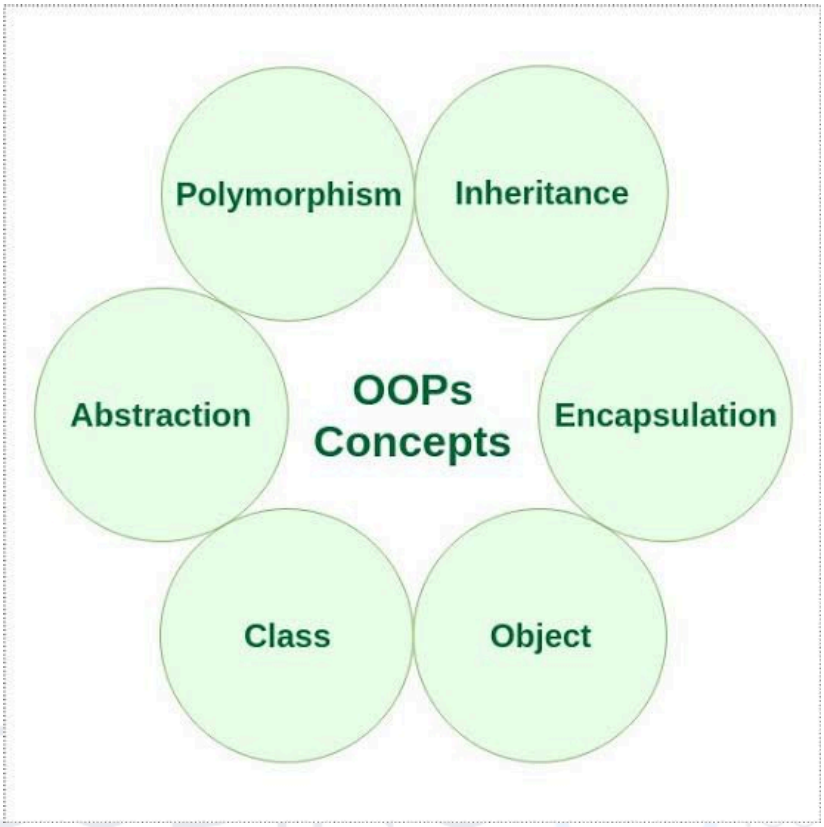

## Learn more about “this” pointer (examples using code):

C++: https://www.scaler.com/topics/cpp/this-pointer-in-cpp/

Java: https://www.programiz.com/java-programming/this-keyword

Python: In python “this” pointer is called “self”. The purpose served by “this” pointer in C++/Java is served by “self” keyword in python.

https://www.programiz.com/article/python-self-why




## Key concepts of OOPs

There are 4 main concepts / components / pillars of OOPs.

## Inheritance

Inheritance is a core feature of Object-Oriented Programming where a class can acquire properties and behaviors from another class. It's a vital aspect of OOP, enabling the creation of new classes based on existing ones.

When inheritance occurs, a new class, known as the "derived class" or "child class," is formed from an existing class, referred to as the "base class" or "parent class." This derived class inherits all the properties of the base class without altering them, and it can introduce additional features unique to

itself. These modifications in the derived class do not impact the base class, allowing for specialization.

In essence, the derived class is specialized based on the properties and behaviors inherited from the base class.

The terms "Subclass" or "Derived Class" refer to the class inheriting properties, while "Superclass" or "Base Class" denotes the class whose properties are inherited by the subclass.

## Access Modifiers

Also called visibility modes, dictate the accessibility of members (attributes and methods) of a class in inheritance relationships. There are typically three visibility modes:

1. **Public:**

   Public members are accessible from outside the class as well as by subclasses.

   Inheritance does not alter the accessibility of public members. They remain accessible in subclasses.

2. **Protected:**

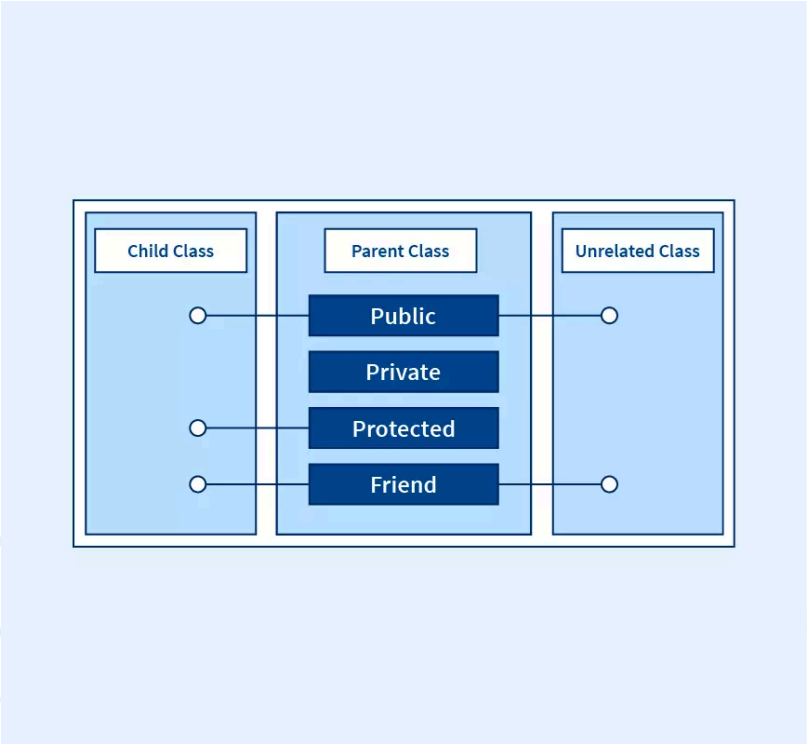
   Protected members are accessible within the class itself and by its subclasses.

   Subclasses can access protected members inherited from the superclass, but they are not accessible outside the class hierarchy.

3. **Private:**

   Private members are only accessible within the class where they are defined.

Inheritance does not grant access to private members in subclasses. They cannot be accessed directly by subclasses.




## Access modifiers Reference (For declaration and Examples using code):

C++: https://www.programiz.com/cpp-programming/access-modifiers

Java: https://www.programiz.com/java-programming/access-modifiers

Python: https://www.studytonight.com/python/access-modifier-python

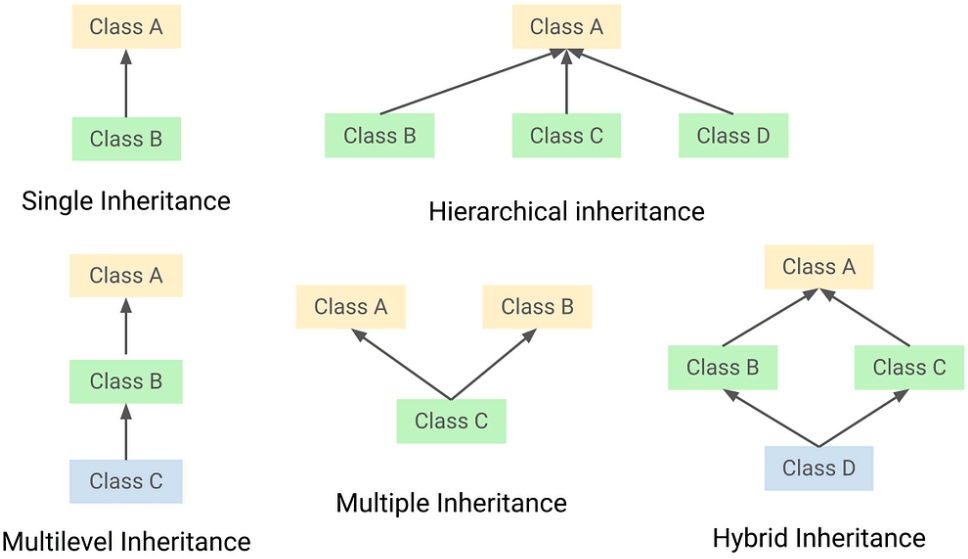
## Types of Inheritance

- **Single Inheritance:** a class inherits properties and behaviors from only one parent class, this means that a subclass can extend and specialize

the functionalities of a single parent class, forming a hierarchical relationship between the two classes.

- **Multiple Inheritance:** a class can inherit attributes and methods from more than one parent class. This means that a subclass can inherit functionalities from multiple superclasses, forming a more complex inheritance hierarchy. This type of inheritance is not provided by **Java**.
- **Multilevel Inheritance:** a class inherits from another class, and then another class inherits from that derived class. In other words, it involves chaining together multiple levels of inheritance, forming a hierarchical relationship among classes.
- **Hierarchical Inheritance:** refers to the process of creating multiple classes that inherit from a single base class.
- **Hybrid inheritance:** inheritance mechanism that blends together the features of simple inheritance, multiple inheritance, and hierarchical inheritance.





### Inheritance Reference (For declaration and Examples using code):

C++: https://www.algbly.com/Tutorials/Cpp-programming/Cpp-inheritance-types.html

Java: https://www.javatpoint.com/types-of-inheritance-in-java

Python: https://www.boardinfinity.com/blog/types-of-inheritance-in-python/

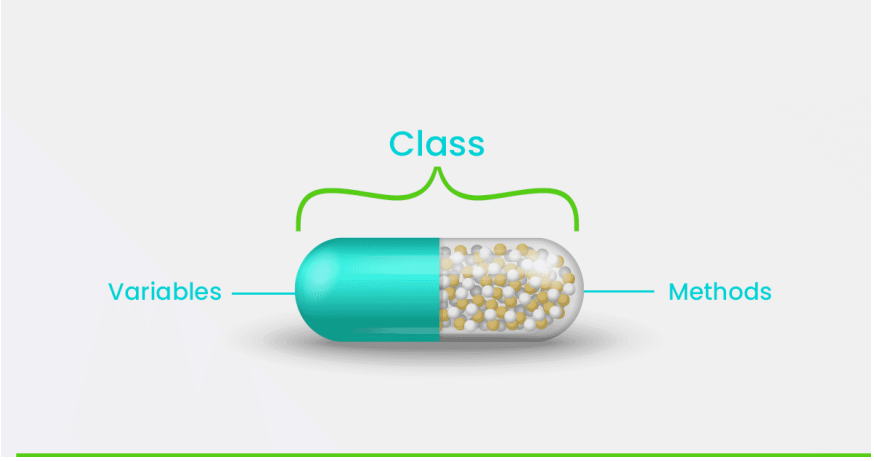
## Encapsulation

It refers to the **bundling of data** (attributes or properties) and **methods** (functions or procedures) that operate on that **data into a single unit**, typically a class. This unit restricts access to the data from outside interference and misuse, and only exposes the necessary functionalities through well-defined interfaces. It means that all the data and methods are bound together and all the unnecessary details are hidden from the user.



Two features integral to encapsulation-

1. **Data hiding:** means restricting access to any member of an object.
2. **Data binding:** binding data members and methods together as a whole, as a class.



### Encapsulation Reference (For declaration and Examples using code):

It is done using access modifiers in C++, Java and Python. Refer to Page 4 for references.

## Polymorphism

Polymorphism, in simple terms, means "many forms." In object-oriented programming, it refers to the ability of different objects to respond to the same message or method call in different ways. This allows objects of different classes to be treated as objects of a common superclass, promoting flexibility and extensibility in code.

There are two types of polymorphism-

1. **Compile-time polymorphism:**

   Compile-time polymorphism, also called Static Polymorphism, occurs during the compilation phase of a program. In this type of polymorphism, the compiler determines the appropriate behavior or type for an entity based on its context in the code.

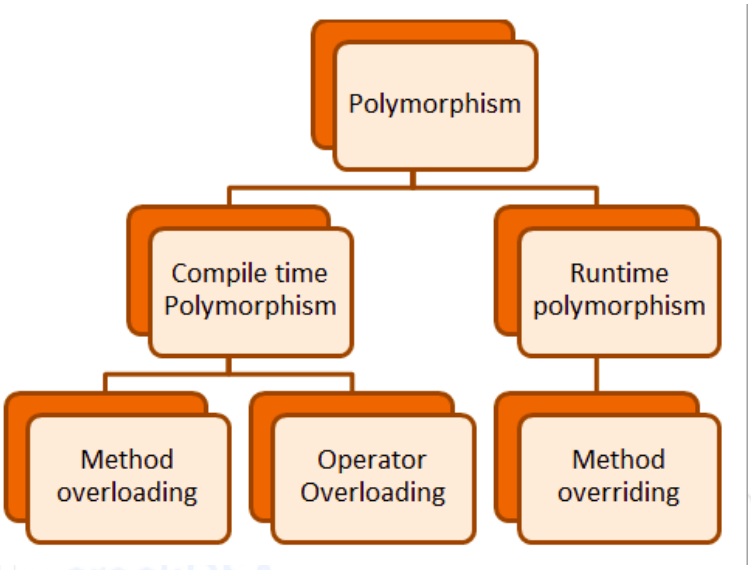
   By using method overloading and operator overloading, you can achieve compile-time polymorphism in your programs. The compiler determines which version of the method or operator to use based on the types and number of arguments provided, allowing for flexibility and code reuse.

2. **Run-time polymorphism:**

   Run-time polymorphism, also known as dynamic polymorphism, occurs when the appropriate method or behavior to execute is determined at run-time based on the actual type of the object being referred to. This is typically achieved through method overriding, where a subclass

provides a specific implementation of a method that is already defined in its superclass.

Run-time polymorphism, also known as dynamic polymorphism, is achieved through method overriding in object-oriented programming languages.




**Polymorphism Reference (For declaration and Examples using code):**

C++: https://www.scaler.com/topics/cpp/polymorphism-in-cpp/

Java: https://www.scaler.com/topics/java/polymorphism-in-java/

Python: https://www.toppr.com/guides/python-guide/tutorials/python-oops/polymorphism-in-python-with-examples/

## Abstraction

We aim to abstract real-life problems by simplifying them and focusing on essential aspects while omitting unnecessary details. By defining the properties of these problems, including the affected data and identified operations, we create a standardized model that can serve as a solution

template for similar problems. This approach proves efficient as many real-life problems share common properties, allowing for the reuse of abstracted models to solve various instances of similar problems.

Different types of abstraction exist in object-oriented programming (OOP), with two primary types being data abstraction and process abstraction.

- **Data Abstraction:**

  Data abstraction involves abstracting or encapsulating data entities, focusing on their essential characteristics while hiding unnecessary details.

  It allows for the representation of real-world entities in a simplified and manageable form within the program.

  Examples of data abstraction include classes, objects, and abstract data types (ADTs), which define data structures and their associated behaviors without revealing their internal implementation details.

- **Process Abstraction:**

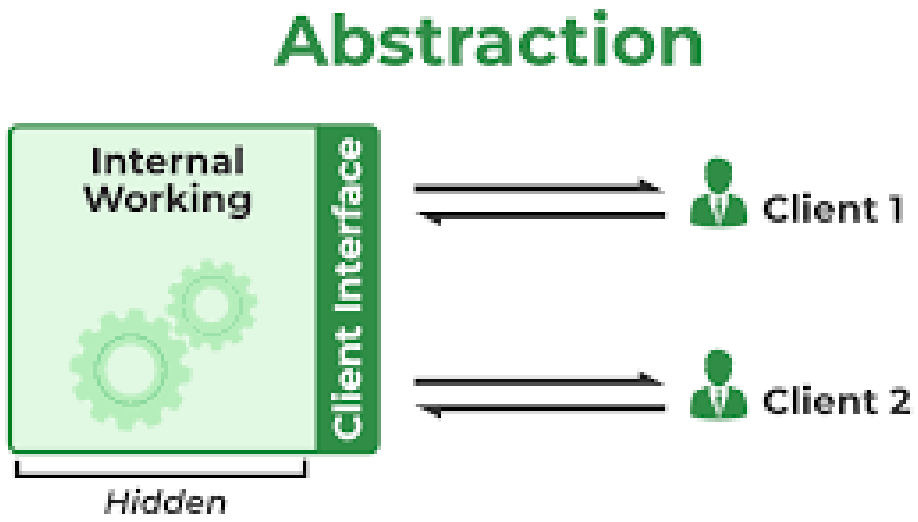
  Process abstraction, on the other hand, hides the underlying implementation details of a process or algorithm, focusing on its high-level functionality or behavior.

  It enables the separation of concerns by abstracting away the internal workings of a process, allowing users to interact with it at a higher level of abstraction.

  Examples of process abstraction include interfaces, which define a contract for implementing classes without specifying the implementation details, and function prototypes or method signatures, which declare the input/output parameters and return type of a function/method without providing its implementation.

Examples of process abstraction include interfaces, which define a contract for implementing classes without specifying the implementation details, and function prototypes or method signatures, which declare the input/output parameters and return type of a function/method without providing its implementation.

These two types of abstraction play a crucial role in OOP by promoting modularity, encapsulation, and code reusability, ultimately leading to more maintainable and scalable software systems.




## Abstraction Reference (For declaration and Examples using code):

C++: https://www.scaler.com/topics/cpp/abstraction-in-cpp/

Java: Majorly done using abstract classes and interfaces.

https://www.upgrad.com/blog/abstraction-in-java/

Python: https://www.scaler.com/topics/python/data-abstraction-in-python/

## Interface in OOPs

An interface in object-oriented programming (OOP) is a blueprint of a class that defines a set of methods that must be implemented by any class that implements the interface. It specifies a contract that classes must adhere to, outlining the methods they must provide without specifying the implementation details.

C++ does not provide the exact equivalent of Java interface. In C++, overriding a virtual function can only be done in a derived class of the class with the virtual function declaration, whereas in Java, the overrider for a method in an interface can be declared in a base class.

### Interface Reference (For declaration and Examples using code):

C++: Interfaces in C++ are also called abstract classes.

https://www.javatpoint.com/interfaces-in-cpp

Java: https://www.programiz.com/java-programming/interfaces

Python: https://www.tutorialspoint.com/python/python_interfaces.htm

## Abstract Classes

Abstract classes in object-oriented programming (OOP) are classes that cannot be instantiated directly and are intended to serve as base classes for other classes. They may contain abstract methods, which are methods without a body, and/or concrete methods, which have an implementation. Abstract classes provide a way to define a common interface or set of behaviors that subclasses must implement, while also allowing for code reuse and polymorphism.

Key characteristics of abstract classes:

1. Abstract classes cannot be directly instantiated; they serve as templates for other classes.
2. Abstract classes can have abstract methods, which must be implemented by concrete subclasses.
3. They may include concrete methods with implementations that subclasses can inherit or override.
4. Abstract classes can have member variables like regular classes.
5. They define common behaviors for subclasses, promoting code organization and reuse.

### Abstract Classes Reference (For declaration and Examples using code):

Java: https://www.programiz.com/java-programming/abstract-classes-methods

https://www.baeldung.com/java-interface-vs-abstract-class

Python: https://www.scaler.com/topics/abstract-class-in-python/

## Virtual Functions (IMPORTANT for C++)

In object-oriented programming, a virtual function or method is one that can be redefined in a derived class, providing the ability to customize its behavior. This enables polymorphic behavior, where the appropriate function implementation is determined dynamically at runtime based on the object's actual type.

### Virtual Functions Reference (For declaration and Examples using code):

C++: https://www.programiz.com/cpp-programming/virtual-functions

Java: In java all public variables and functions are virtual by default and it does not have any “virtual” keyword which exists in C++. Being virtual is a property of variables and methods. You may read about this property and its presence in Java, but from an interview point of view this is not important if Java is the focus.

https://www.educba.com/virtual-function-in-java/

Python: In python all methods are virtual by default, here also no “virtual” keyword exists.

## Pure virtual functions

Pure virtual functions, also known as abstract methods, are virtual functions in C++ that are declared in a base class but have no implementation. They are used to define a common interface for a class hierarchy, specifying methods that subclasses must implement. Pure virtual functions serve as placeholders for methods that are intended to be overridden by derived classes.

In Java pure virtual functions are abstract classes of Java. Same for python we have abstract classes and methods that can be used to serve the purpose of pure virtual functions.

### Pure virtual functions Reference (For declaration and Examples using code):

C++: https://www.scaler.com/topics/pure-virtual-function-in-cpp/

## Exceptions

Exceptions in object-oriented programming (OOP) are a mechanism for handling unexpected or exceptional conditions that occur during the execution of a program. Exceptions allow programmers to gracefully handle errors or exceptional situations without abruptly terminating the program.

### Exceptions and its types Reference (Examples using code):

C++: https://www.programiz.com/cpp-programming/exception-handling

Java: https://www.educative.io/answers/what-are-different-types-of-exceptions-in-java

Python: https://www.programiz.com/python-programming/exceptions

## Exception Handling

Exception handling refers to the practice of managing potential errors or unexpected situations in a program to prevent it from abruptly halting. It involves identifying undesirable program states and defining appropriate actions to ensure the program's smooth execution. The try-catch construct is commonly utilized for handling exceptions, allowing developers to enclose error-prone code within a try block and specify how to handle exceptions in catch blocks. This approach enables programmers to anticipate and address potential issues, enhancing the reliability and stability of their software applications.

### Exception Handling Reference (Examples using code):

C++: https://www.javatpoint.com/cpp-exception-handling

Java: https://www.digitalocean.com/community/tutorials/exception-handling-in-java

Python: https://www.programiz.com/python-programming/exception-handling

# Thank You!

## Join Coding75 Pro 🚀

- Live DSA Classes
- Project Building Session
- 1:1 Mock Interviews
- Resume Review
- CS Fundamental Classes
- And much more...



**Join Now**