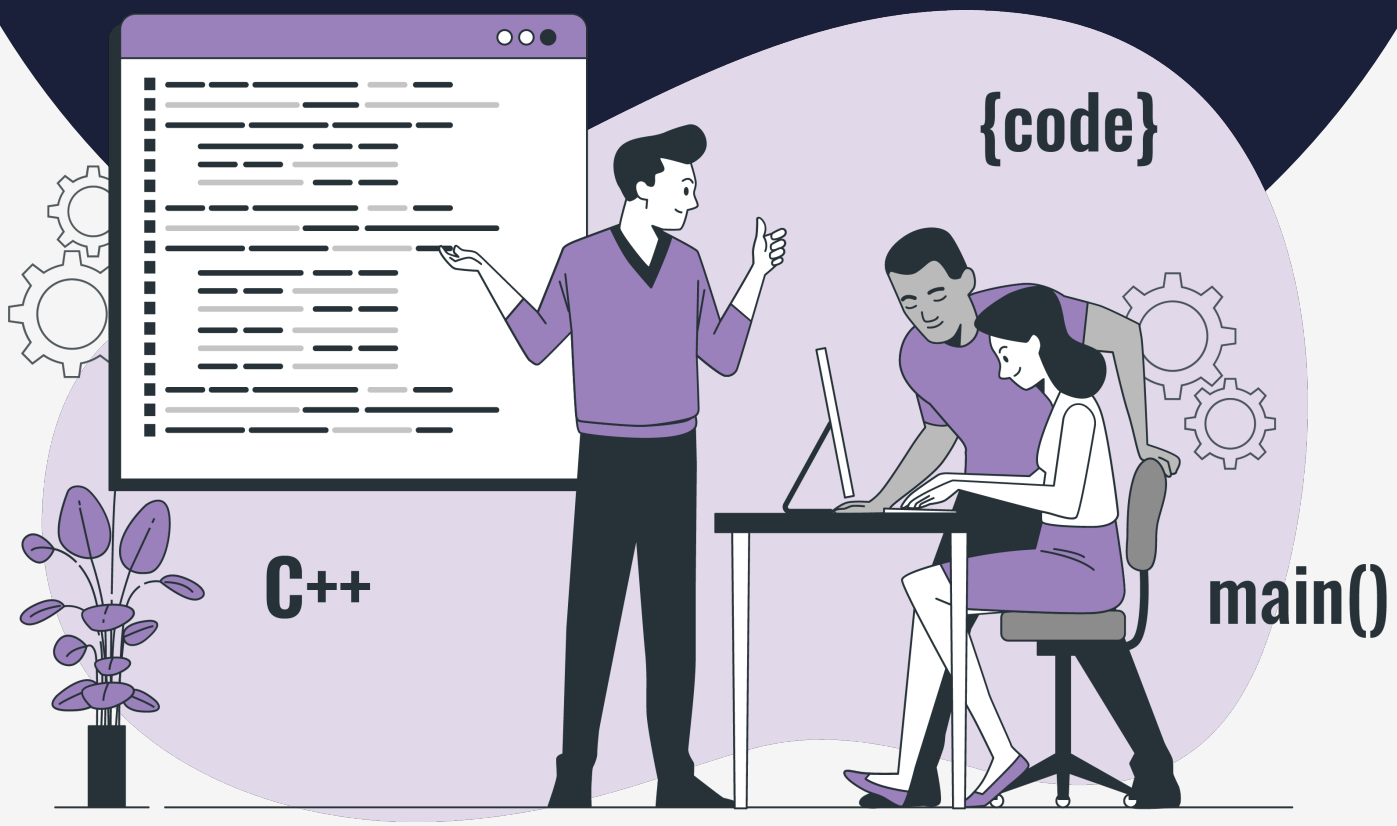
Lesson:

OOPs

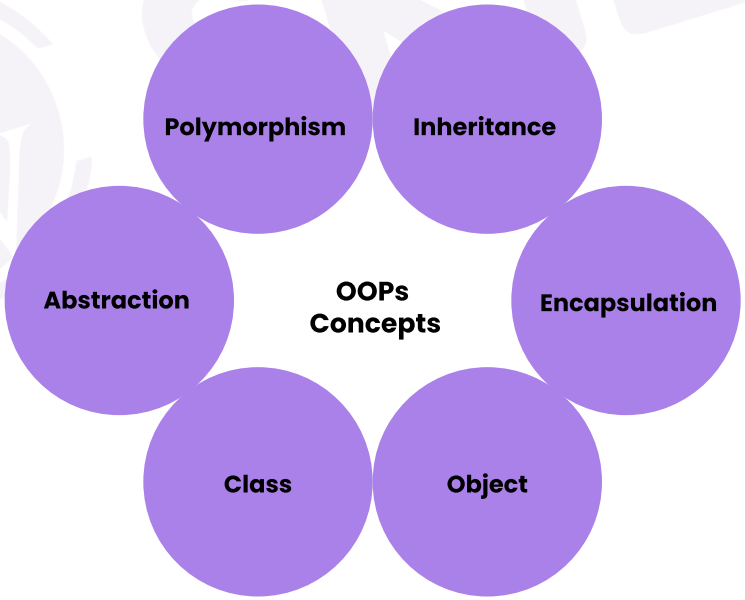


Concepts involved

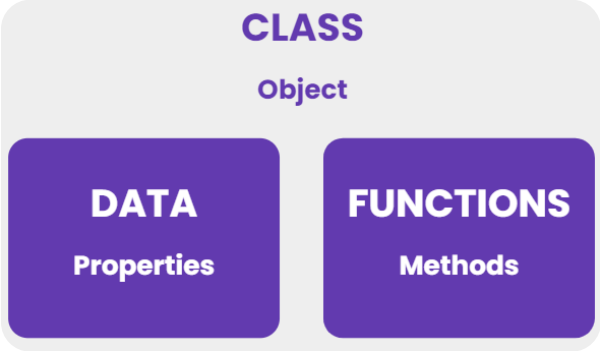
- What is Object Oriented Programming ?
- What is Class ?
- What is an Object ?
- How Class and Object are represented through code ?
- Constructor
- Encapsulation
- Abstraction
- Access Specifiers
- What is inheritance
- Types of Inheritance

What is Object-Oriented Programming ?

Programming languages that use objects as a key source to implement what is to happen in the code are known as "object-oriented programming," or OOPs. It consists of classes and objects which simplifies the software development process.The primary goal of OOP is to bind together the data and the functions that use them such that only that function and no other section of the code may access the data.Implementing real-world concepts like inheritance, encapsulation, polymorphism, etc. in programming is the goal of object-oriented programming.



Class in C++ is the fundamental unit of Object-Oriented programming. It is a user-defined data type that may be utilized by creating an instance of the class in order to access and use its data members and member functions. Class can be understood with the help of the following example:
Consider a class of students, there might be students with different names,roll number, class, section etc but all of them will have these properties i.e. there will be no student who has no name, roll number, class or section. Here student is the class and the properties are attributes of the class student.



Code: https://pastebin.com/wW4uQWyb
Explanation: We have created a class Student that has two data members rollNumber. and a string name which is used to store the name of the student.

Object is an instance of the class. An identified entity with certain traits and behaviors is called an object. In the above example, different students are different objects of the class.
There are a couple of ways to create an object of class. Consider a class named parent, we can create its objects using:

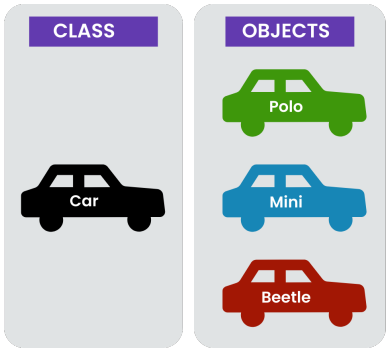
- Parent p1;
- Parent *p2=new Parent();

The second method creates an object pointer which points at the object p2.

Code: https://pastebin.com/DkikatGb

Explanation: We have a class Student which has three data members rollNumber, standard and section to store the following details of the student. In main() we have created an object of the class Student named s1.

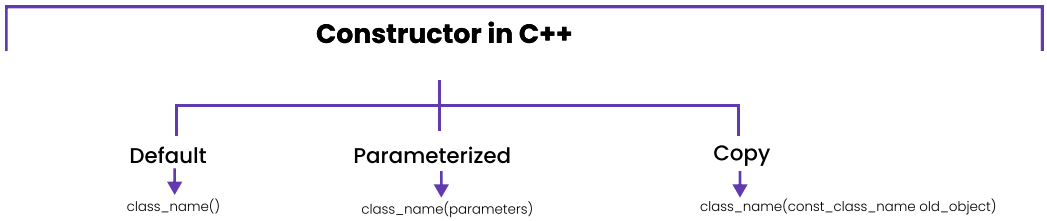
Example 1
Let's see an example of a Car. You all know that Car in general is a vague term. Why? Because cars can be of many types/models. So, we can call car as a class and different models of cars as objects. Car is a blueprint on which different models of the car have been made up of. Cars don't exist physically in general but exist as different models in real life.



Example 2
Similarly, Take another example of Animal, if you will see an animal then after seeing it you will know which type of animal it is. It can be a dog, cat, lion, goat etc. but in general terms it is an animal. So, Animal can be called as a Class and different types of animals like cat, dog etc. will be termed as objects. Different objects have different behaviors. Dog barks. Lion roars. They have different colors.
Code: https://pastebin.com/6AaNHmSA

We have a class Animal which has two properties i.e. name and sound and we have created an object of the class to assign properties to different animals.

Constructor



When we create an object of a class then a special method known as constructor is automatically invoked which is used to instantiate the object and has no return type. The name of the constructor is the same as that of the class and it is invoked at the time of object creation. Even if we do not define any constructor , the compiler will automatically provide a default constructor to the class. This constructor is of three types:

Default Constructor: This type of constructor takes in no parameters or arguments.

Code Link:https://pastebin.com/WNbNipE8

Explanation: We have created a class DefaultConstruct and have created a constructor for that class which takes no parameters but assigns values to the data members of the class.

Parameterized Constructor: This is a constructor in which arguments are passed in the constructor which are then used to initialize the arguments of the class. When we define this type of constructor then the compiler will not call the default constructor.

Code Link: https://pastebin.com/eaWUCKQB

Explanation: We have created a class ParameterizedConstruct with a constructor which takes two arguments which are further assigned to x and y data members of the function. When an object of this class is created we pass the two arguments at that time in the object name which assigns those values to the x and y for that object i.e. x becomes 3 and y becomes 4.

Copy Constructor: This constructor is basically a member function which initializes an object using another object of the same class. It basically takes a reference to the object of that class as an argument.

Code Link: https://pastebin.com/zeM6LZpS

Explanation: We have created a copy constructor of class Shape which takes an object of that class as an argument and assigns the values of that object to the newly created object. In the main() we first create an object s1 with values 5 and 20, we then create an object s2 of that class which is assigned with s1 using copy constructor and thus s2.x becomes equal to 5 and s2.y becomes equal to 20.

```
s1.x = 5, s1.y = 20
s2.x = 5, s2.y = 20

...Program finished with exit code 0
Press ENTER to exit console.
```

Destructor: Destructor is a member function which basically destroys the object that has been created by the constructor. It has the same name as that of the class but it has a tilde(~) symbol before the name. A class can only have one destructor i.e. the destructor cannot be overloaded unlike a constructor which we will study further in polymorphism.

Code link: https://pastebin.com/2n9uFpMX

Explanation: We have created a class Student and have created a default constructor as well as a destructor for that class. As soon as the object is created the constructor will be called first and once all the functions of the class are executed, the destructor will be called for that object.

```
Constructor is working
Main is executed
Destructor is working

...Program finished with exit code 0
Press ENTER to exit console.
```

Encapsulation

Encapsulation in C++ is basically referred to as binding of methods and variables together that are present in a single class. The data stored in the variables of the class is not accessed directly, instead it is done with the help of methods present in the class.
Consider a practical illustration of encapsulation: at a company, there are various divisions, such as the accounts division, the administrative division, the tech division, etc. The administrative department manages all admin work and maintains records of all admin data. In a similar vein, the tech section manages all actions linked to tech and keeps track of all tech stacks. Now a scenario could occur when an administrative official requires all the tech information for a specific month for some reason. He is not permitted to view the tech section's data directly in this instance. To request the specific information, he must first speak with another officer in the tech department. This is what encapsulation is.

Encapsulation also leads to data abstraction or hiding which basically restricts the user from accessing the data members of the object. In the above example, the data of any of the sections like administrative, tech or accounts are hidden from any other section. This is done with the help of access specifiers which we will be studying in a while.

Code example: https://pastebin.com/89mBTDih

```
10

...Program finished with exit code 0
Press ENTER to exit console.
```

Explanation: The following code shows a class Encapsulation which has a variable data which is private and it can be accessed only using get() and set() functions as shown in the code. Where the get() function returns the variable data and set() function assigns the value to variable data.

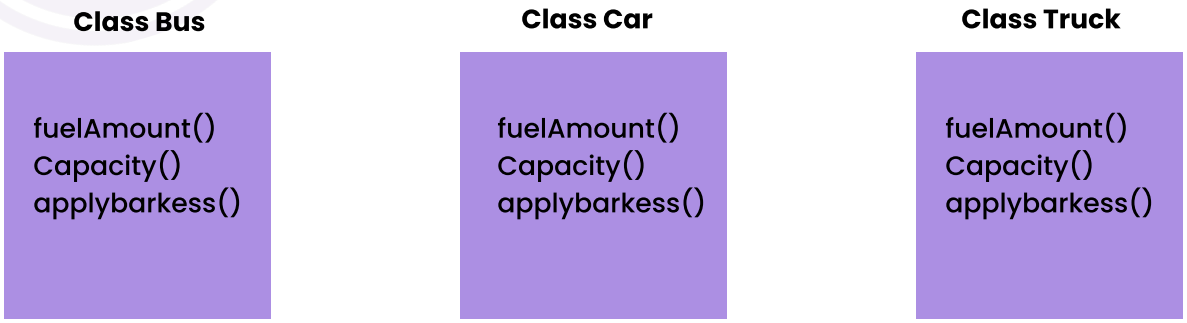
Abstraction

Abstraction is a very important and useful feature in OOPs which enables us to showcase or display only essential information and hide the details related to that information. It is the process of exposing to the outside world only the information that is absolutely necessary while obscuring implementation or background information.
Consider the example of pow() function in C++ library math.h . When we use this function we just pass two numbers (n,m) as arguments and the function returns us mth power of n. We use the function without knowing how it is working internally so this is abstraction.

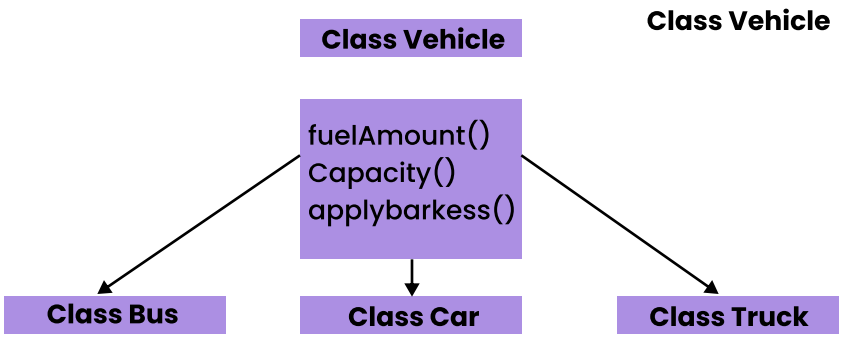
Now we will study a very useful feature of Object-Oriented Programming and this is Inheritance.

What is inheritance and where is it used?

Inheritance is a very useful feature which enables a class to derive the properties of some other class. The class whose properties are inherited is known as the parent class or the super class. The class which has inherited properties is called the child class or the sub class. Inheritance is useful since it provides us with the feature of reusability i.e. if class A inherits class B, then class A can reuse the attributes of class B without those attributes being defined in class A.



All these classes have the same properties so instead of writing them again and again we can just write these properties in one class and derive the other classes from that class.



Before we move ahead with types of inheritance we need to understand another important concept which is access specifiers:
Access Specifiers basically assign accessibility to the members of a class which allows them to be accessed by some other class or by the user. There are 3 types of access specifiers:

- **Public:** The data members and functions that are declared as public can be accessed by other classes and functions. They can be accessed anywhere in the code.
  https://pastebin.com/kU5QVhwS

  Explanation: All the data members of class Car are public which means they can be accessed by other classes and functions in the code.

- **Protected:** The data members and functions declared protected can be accessed only by the parent class and its member functions and its derived child classes. It cannot be accessed anywhere else.
  https://pastebin.com/5awEhes1

  Explanation: The data member side of class Square is declared as protected and can be accessed only by the base class and its derived classes whereas the other members/functions are declared public.

- **Private:** The data members and functions declared private can only be accessed within that class. They can not be accessed by any other class derived from that class.
  https://pastebin.com/SHYMtGve

  Explanation: The data member side of class Square is declared as private which means it can only be accessed within the class whereas the other function is declared public and can be accessed from anywhere in the code.

How are these access specifiers used in Inheritance?

When a class is derived from another class then we specify the kind of access the derived class has i.e. if it can access the members of the parent class or not and this is done with the help of these access specifiers.There are three modes

- Public: If a subclass is derived from a public base class. The protected members of the base class will then become protected in the derived class,the public members of the base will become public in the derived class and the private members of the base class will become private in the derived class.
- Private: If a subclass is derived from a Private base class. The base class's protected and public members will then both change to Private in the derived class.

- Protected: If a Protected base class is used to create a subclass. The base class's protected and public members will then both become protected in the derived class.

The syntax for the same is:
derived-class-name — name of the base class

Example:

```
-> class A : private B          //private derivation
        {          }
->class A : public B            //public derivation
        {          }
-> class A : protected B        //protected derivation
        {          }
-> class ABC: XYZ               //private derivation by default
         {         }
```

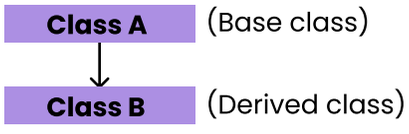
Code link: https://pastebin.com/CFidJEMM

Explanation: We have a class parent which has three data members a,b and c which are public, protected and private respectively. We have created a class child1 which inherits class parent as public which means that access of a,b and c will remain the same for class child1 as that of class parent. Class child2 is inherited from class parent as private which means that the public and protected members of class parent i.e. a and b will be private for class child2.
Class child3 is inherited from parent class as protected which means a and b will be protected for child3 and c will remain private.

Types of Inheritance

- Single Inheritance
- Multilevel Inheritance
- Multiple Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance
-

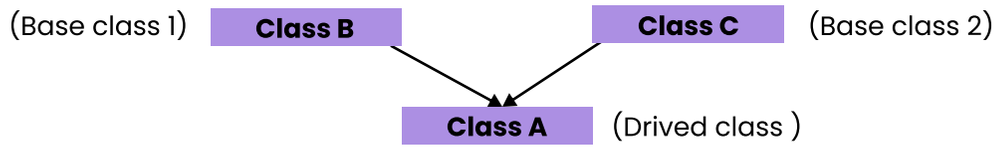
1) Single Inheritance: When a class inherits from a single class i.e. the derived class is inherited by a single base class.



Code Link: https://pastebin.com/88ykJdPB

Explanation: We have a class Shape which is our base class and a class Rectangle which is derived from the class Shape and this is single inheritance.
In the main function we have created an object of derived class Rectangle which will call the constructor of it's parent class and print "This is a shape"

2) Multiple Inheritance: In multilevel inheritance, a class can inherit from more than one class i.e one derived class or subclass is inherited from more than one base



Code link: https://pastebin.com/VRf0maaM

Explanation: We have a class Vehicle and another class TwoWheeler. We have created a class Bike which inherits from class Vehicle and class TwoWheeler thus representing the inheritance as shown in the above diagram.

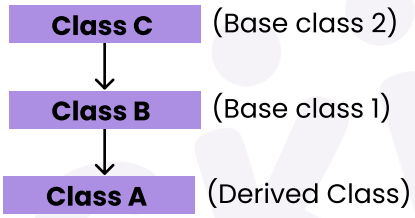
Output:

```
This is class Vehicle
This is class TwoWheeler
This is class Bike

...Program finished with exit code 0
Press ENTER to exit console.
```

Explanation: The class Bike inherits class Vehicle and class TwoWheeler, when we create an object of class Bike, the constructor of base classes will be executed first. The class Vehicle is the first base class for the derived class Bike so the constructor of Vehicle will be called followed by the constructor of TwoWheeler and then the constructor of class Bike will be called.

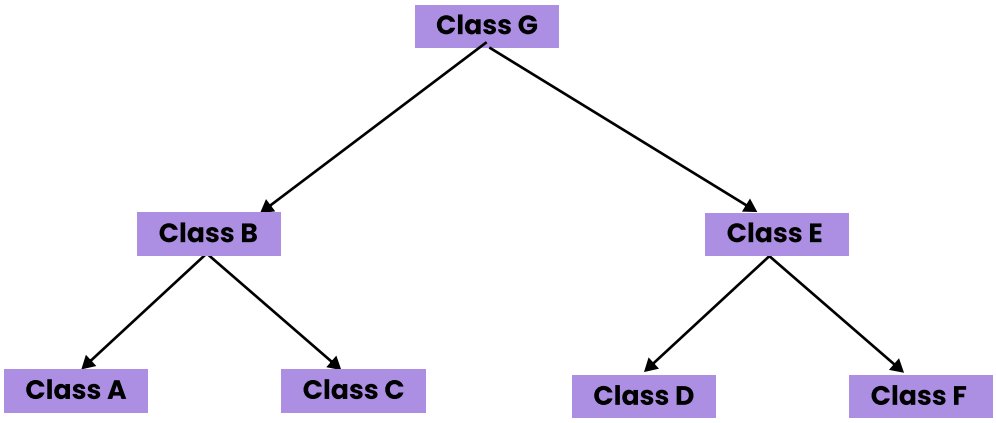
3) Multilevel Inheritance: When a base class is itself derived from some other class it is known as multilevel inheritance i.e. the parent class is itself a child class of some other parent class.



Code link: https://pastebin.com/YPWKxRqr

Explanation : We have a class Shape and a class Quadrilateral is derived from that class which is followed by a class Square which is derived from class Quadrilateral. Thus showing the inheritance as depicted in the figure above.

4) Hierarchical Inheritance: More than one subclass is inherited from a single base class in this type of inheritance i.e. a single base class is used to create multiple derived classes.



Code link: https://pastebin.com/UCNucUPG

Explanation: We have a class Shape which is our base class, and two classes Rectangle and Square are derived from that class which shows hierarchical inheritance.

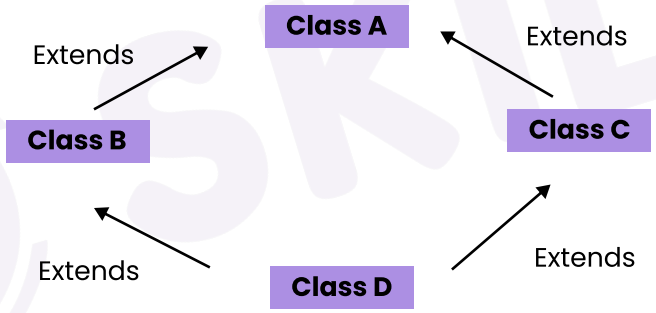
5) Hybrid (Virtual) Inheritance: It is a combination of more than one type of inheritance. For example: Hierarchical inheritance and Multiple Inheritance are combined together and this is called hybrid inheritance.

Code link: https://pastebin.com/jzRbXqyA

Explanation: We have two base classes Vehicle and Fare. A class Car is derived from the class Vehicle and a class Bus is derived from Vehicle and Fare. If we look at the diagram above then the class F and G are the classes Vehicle and Fare respectively, and the class B is class Car and class E is class Bus.

Diamond Problem in Inheritance ?

The "diamond problem" is a term used to describe a situation that can occur in a class hierarchy when a subclass extends multiple superclasses and those superclasses have a common ancestor. This can lead to ambiguity because the subclass has multiple inherited methods with the same name, which can cause the subclass to behave unexpectedly.



Code: https://pastebin.com/02hFfRWc

Explanation: In the above code, we have a base class Vehicle and two classes FourWheeler and TurboEngine which are derived from the class Vehicle. We have another class Car which is derived from both the FourWheeler and TurboEngine class. Thus, on creating an object of class Car, the constructor of class Vehicle is called twice as both classes FourWheeler and TurboEngine are derived from class Vehicle. Thus the constructor of class Vehicle is called twice and this is a diamond problem.

Output:

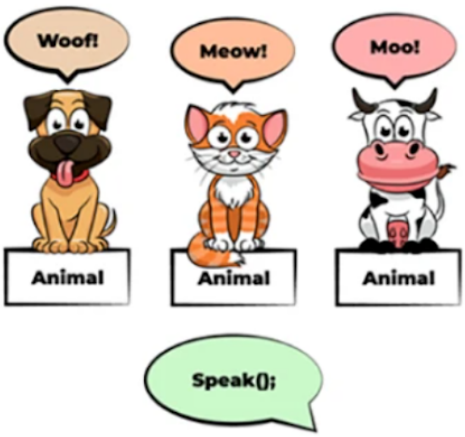
```
Vehicle called
FourWheeler called
Vehicle called
TurboEngine called
Car called


...Program finished with exit code 0
Press ENTER to exit console.
```

Polymorphism

Polymorphism is the ability of an entity, such as an object or method, to take on multiple forms. It can be thought of as the ability of a message to be expressed in multiple ways.

For example, a person may exhibit the roles of a father, husband, and employee in different situations and contexts, each with its own set of characteristics and behaviors. Similarly, in object-oriented programming, an object can take on multiple forms and exhibit different behaviors depending on the context in which it is used.



Polymorphism is a powerful feature of object-oriented programming that allows for greater flexibility and code reuse. It enables a single interface to be used to invoke methods defined in multiple classes, making it easier to add new functionality to a program without requiring extensive modifications to existing code.
Here is an example of polymorphism in C++.

Compile-time Polymorphism

Compile-time polymorphism, also known as static polymorphism, is a type of polymorphism that is resolved during the compilation of a program. It is achieved through function overloading.It basically has a number of functions with the same name but different number or type of arguments.

Function Overloading

Function overloading is a type of compile-time polymorphism in which multiple methods in a class have the same name but different parameter lists. When a function is called, the C++ compiler determines which version of the method to execute based on the number and type of the arguments passed to the method.

Here is an example of method overloading in C++:

Code link: https://pastebin.com/W3SxLgyz

Output:

```
helper function with integer argument is executed
value of x is 1
helper function with double argument is executed
value of x is 1.332
helper function with two integer arguments is executed
value of x and y is 1, 2


...Program finished with exit code 0
Press ENTER to exit console.
```

In the example above, the Poly class has three overloaded versions of the helper functions, one that takes one int parameter, another that takes one double parameter, and another that takes two int parameters.

Operator Overloading
Operator overloading gives special meaning and functioning to different operators. For ex: the '+' operator can be used to add two complex numbers, we will define the function of the operator in such a way that our desired operation is achieved.

Code link: https://pastebin.com/ardeuss3

Explanation: In the above code we have a class complex which represents complex numbers, here we have overloaded the '+' operator and changed its functioning. The '+' operator here is used to add two complex numbers as mentioned in line number 17. It adds the real parts together and the imaginary parts together and returns the resulting complex number, ans.real stores the sum of real parts of the numbers and ans.imaginary stores the sum of imaginary part of the number.

```
4 + i6


...Program finished with exit code 0
Press ENTER to exit console.
```

Run-time Polymorphism

Run-time polymorphism, also known as dynamic polymorphism, is a type of polymorphism that is resolved at runtime. It is achieved through function overriding in C++, which is the ability of a subclass to override the methods of its superclass.
Function Overriding occurs when a derived class has a definition for one of the member functions of the base class.

Code Link: https://pastebin.com/KjGKur54

Output:

```
derived class
base class


...Program finished with exit code 0
Press ENTER to exit console.
```

In the above code, the print function is declared virtual in the parent class and overriden in the child class i.e. the print function that will be executed is decided on runtime. The virtual keyword is thus used to execute runtime polymorphism.

Function Overloading	Function Overriding
Occurs within a single class	Occurs in two related classes, a superclass and subclass
Different methods have the same name but different parameters	Same method name, parameters and return type in the superclass and subclass
Signature of functions are different	Signature of functions should be same
Can have different access modifiers	Must have the same or a more restrictive access modifier

Differences between Compile time Polymorphism and Run time Polymorphism

Compile-time Polymorphism	Run-time Polymorphism
Occurs at compile time	Occurs at runtime
Achieved through function overloading and operator overloading	Achieved through function overriding
function names should be the same but parameter list must be different.	function signature (name and parameter list) must be the same
Faster execution time	Slower execution time
More memory efficient	Less memory efficient

Friend Function

Friend function is a non-member function which is listed in the class and is used to access the private members of the class to perform operations using these private members of the class. The friend function cannot access the private members directly, instead it uses a specific syntax i.e. object name and the dot operator.

Code link: https://pastebin.com/XbTqJaeu

Explanation: In this code as we can see that we have created a friend function named add which takes in object of class addition as a parameter and returns the sum of private data members a and b and returns their sum. This is the use of the friend function.

Output:

```
22


...Program finished with exit code 0
Press ENTER to exit console.
```

Upcoming Class Teasers

- LinkedList