**Object Oriented Programming**

The programming paradigm where everything is represented as an object is known as truly object-oriented programming language

**Smalltalk** is considered as the first truly object-oriented programming language.

OOPs refers to languages that use objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc. in programming.

The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

**Advantage of OOPs over Procedure-oriented programming language**

|  |  |
| --- | --- |
| **OOPs** | **Procedure-oriented programming language** |
| makes development, maintenance easier | it is not easy to manage if code grows as project size grows. |
| provide data hiding | language a global data can be accessed from anywhere. |

* OOPs provide the ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

**reasons why OOPs is mostly preferred, but the most important among them are:**

1. OOPs helps users to understand the software easily, although they don’t know the actual implementation.
2. With OOPs, the readability, understandability, and maintainability of the code increase multifold.
3. Even very big software can be easily written and managed easily using OOPs.
4. OOP helps to keep the C++ code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
5. OOP makes it possible to create full reusable applications with less code and shorter development time

**Advantages of OOP**

**1. Re-usability:** This is done with the use of a class. (reusing some facilities rather than building them again)

**2. Data Redundancy:** This is a condition created at the place of data storage (Databases)where the same piece of data is held in two separate places. **Data redundancy is one of the greatest advantages of OOP**. If a user wants a similar functionality in multiple classes, he/she can go ahead by writing common class definitions for similar functionalities and inherit them.

**3. Code Maintenance:** It is always easy and timesaving to maintain and modify the existing codes by incorporating new changes into them.

**4. Security:** With the use of data hiding and abstraction mechanism, we are filtering out limited data to exposure, which means we are maintaining security and providing necessary data to view.

**5. Better productivity:** Having more inbuilt features, and easier reading, writing and maintaining. An OOP programmer cans stitch new software objects to make completely new programs. A good number of libraries with useful functions in abundance make it possible.

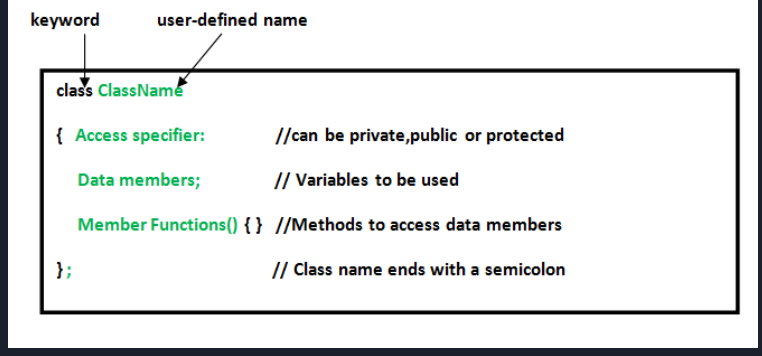
**6. Easy troubleshooting:** Working with OOP language, you will know where to look for. This is the advantage of using encapsulation in OOP; all the objects are self-constrained. With this modality behavior, the IT teams get a lot of work benefits as they are now capable of working on multiple projects simultaneously with an advantage that there is no possibility of code duplicity.

**7. Polymorphism Flexibility:** You behave in a different way if the place or surrounding changes. A person will behave like a customer if he is in a market, the same person will behave like a student if he is in a school and as a son/daughter if put in a house. Here we can see that the same person shows different behavior every time the surroundings are changed. This means polymorphism is flexible and helps developers in a number of ways- **It’s simplicity & Extensibility**

**8. Problems solving:** Decomposing a complex problem into smaller chunks or discrete components is a good practice. OOP is specialized in this behavior, as it breaks down your software code into bite-sized – one object at a time. The broken components can be reused in solutions to different other problems (both less and more complex), or they can be replaced by the future modules that relate to the same interface with implementations details.

**What is a Class?**

* A class in C++ is the building block that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. **C++ class is like a blueprint for an object.**
* **For Example:** Consider the Class of Cars. There may be many cars **with different names** and brands but all of them **will share some common properties** like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, Car is the class and wheels, speed limits, mileage are their properties**.**
* **A Class is a user defined datatype which has data members and member functions.**
* Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions define the properties and behavior of the objects in a Class.
* In the above example of class Car, the data member will be speed limit, mileage etc. and member functions can be apply brakes, increase speed etc.
* **An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.**

****

**Access Specifiers: -** used to implement an important aspect of Object-Oriented Programming known as Data Hiding. 3 types of access modifiers available in C++:

Public Private Protected

**1. Public**: All the class members declared under the public specifier will be available to everyone (classes and functions too). Using the direct member access operator (.) with the object of that class.

**2. Private:** The class members declared as private can be accessed only by the member functions inside the class. They are not allowed to be accessed directly by any object or function outside the class. Only the member functions or the **friend functions** are allowed to access the private data members of a class.

**3. Protected:** Protected access modifier is similar to private access modifier in the sense that it can’t be accessed outside of its class unless with the help of **friend class**, the difference is that the class members declared as Protected can be accessed by any subclass (derived class) of that class as well.



**Member Function: -**

A member function of a class is a function that has its definition or its prototype within the class definition like any other variable. It operates on any object of the class of which it is a member and has access to all the members of a class for that object.

**Data Members: -**

Data members include members that are declared with any of the fundamental types, as well as other types, including pointer, reference, array types, bit fields, and user-defined types. You can declare a data member the same way as a variable, except that explicit initializers are not allowed inside the class definition. However, a const static data member of integral or enumeration type may have an explicit initializer.

If an array is declared as a non-static class member, you must specify all of the dimensions of the array.

A class can have members that are of a class type or are pointers or references to a class type. Members that are of a class type must be of a class type that has been previously declared. An incomplete class type can be used in a member declaration if the size of the class is not needed. For example, a member can be declared that is a pointer to an incomplete class type.

**A class X cannot have a member that is of type X, but it can contain pointers to X, references to X, and static objects of X. Member functions of X can take arguments of type X and have a return type of X.**

**Friend Class & Friend Functions: -**

Friend Class A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class. **E.g**. a **LinkedList** class may be allowed to access private members of Node. Friend Function Like friend class, a friend function can be given a special grant to access private and protected members. A friend function can be:

a) A member of another class

b) A global function Following are some important points about friend functions and classes:

1. Friends should be used only for limited purpose. too many functions or external classes are declared as friends of a class with protected or private data, it lessens the value of encapsulation of separate classes in object-oriented programming.
2. Friendship is not mutual. If class A is a friend of B, then B doesn’t become a friend of A automatically.
3. Friendship is not inherited.
4. The concept of friends is not there in Java.

**Object:** object is an entity that has state and behavior. Here, state means data and behavior means functionality. - Object is a runtime entity, it is created at runtime. - Object is an instance of a class. All the members of the class can be accessed through object.

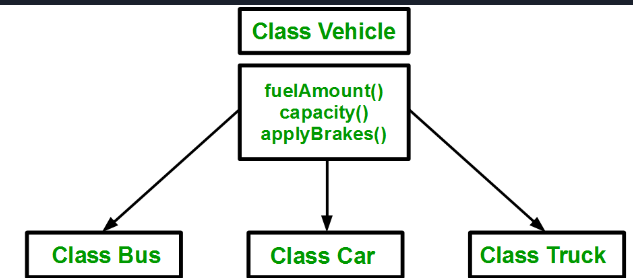
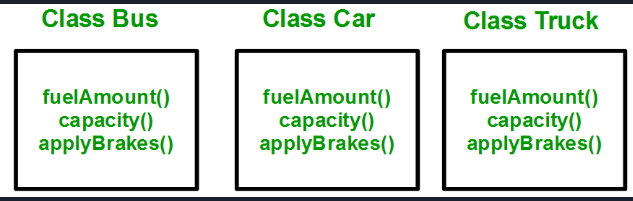
**Inheritance**

In C++, inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. In such way, you can reuse, extend or modify the attributes and behaviors which are defined in other class. Sub Class: The class that inherits properties from another class is called Sub class or Derived Class. The derived class is the specialized class for the base class. Super Class: The class whose properties are inherited by sub class is called Base Class or Super class.

**Advantage of C++ Inheritance**

**Code reusability:** Now you can reuse the members of your parent class. So, there is no need to define the member again. So, less code is required in the class.

**Why and when to use inheritance?**

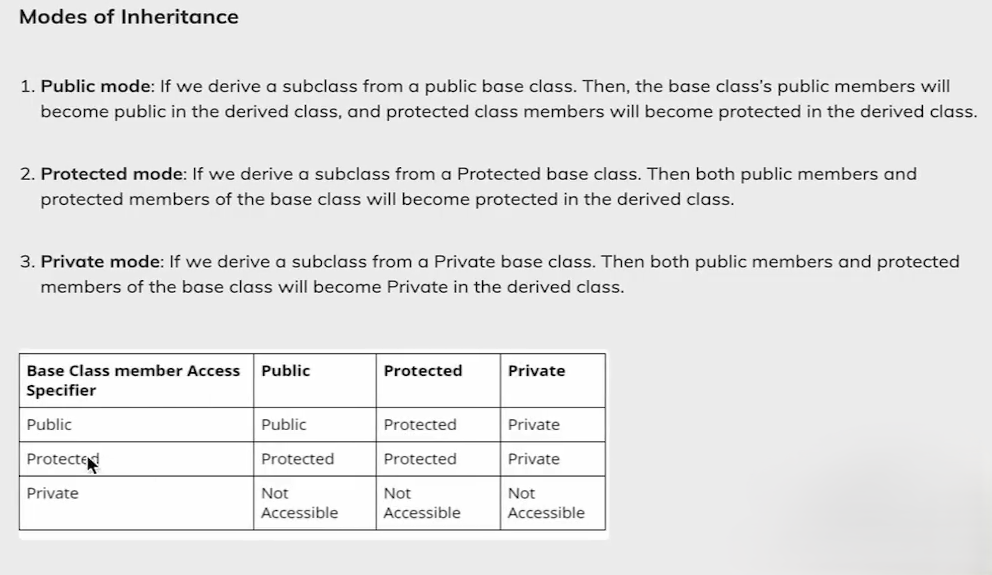
****

Duplication increases the chances of error and data redundancy. To avoid this type of situation, inheritance is used, it simply avoid the duplication of data and increase re-usability.

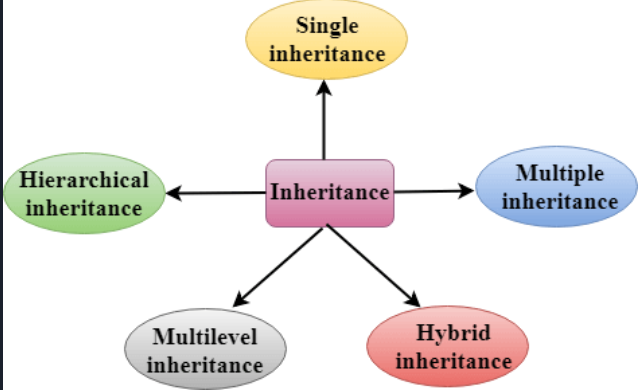
class subclass\_name: access\_mode base\_class\_name {

// body of subclass

};



**Types Of Inheritance:**

****

**Encapsulation:** It involves the bundling of data members and functions inside a single class.

Bundling similar data members and functions inside a class together also helps in data hiding.

encapsulation is a process of wrapping similar code in one place/ in a single unit.

**Why Encapsulation?**

* Encapsulation helps us keep related data & functions together, making code cleaner & easy to read.
* It helps to control the modification of our data members.
* The getter and setter functions provide read-only or write-only access to our class members. E.g.
* Helps to decouple components of a system. E.g. we can encapsulate code into multiple bundles.

**Polymorphism:**

The term "Polymorphism" is the combination of "poly" + "morphs" which means many forms.

There are two types of polymorphism in C++:

**Compile time polymorphism:** The overloaded functions are invoked by matching the type and number of arguments. This information is available at the compile time and, therefore, compiler selects the appropriate function at the compile time. It is achieved by function overloading and operator overloading which is also known as static binding or early binding. Now, let's consider the case where function name and prototype are same.

**Run time polymorphism:** Run time polymorphism is achieved when the object's method is invoked at the run time instead of compile time. It is achieved by method overriding which is also known as dynamic binding or late binding.

| **COMPILE TIME POLYMORPHISM** | **RUN TIME POLYMORPHISM** |
| --- | --- |
| The function to be invoked is known at the compile time. | The function to be invoked is known at the run time. |
| Also, k/a as overloading, early binding and static binding. | Also, k/a overriding, Dynamic binding and late binding. |
| Overloading is a compile time polymorphism where more than one method is having the same name but with the different number of parameters or the type of the parameters. | Overriding is a run time polymorphism where more than one method is having the same name, number of parameters and the type of the parameters. |
| Achieved by function overloading & operator overloading. | It is achieved by virtual functions and pointers. |
| It provides fast execution as it is known at the compile time. | It provides slow execution as it is known at the run time. |
| It is less flexible as mainly all the things execute at the compile time. | It is more flexible as all the things execute at the run time. |

**Data Abstraction in C++:**

Data Abstraction is a process of providing only the essential details to the outside world and hiding the internal details, i.e., representing only the essential details in the program.

Data Abstraction is a programming technique that depends on the separation of the interface and implementation details of the program.

In C++ program if we implement class with private and public members then it is an example of data abstraction.

Data Abstraction can be achieved in two ways:

**Abstraction using classes:** An abstraction can be achieved using classes. A class is used to group all the data members and member functions into a single unit by using the access specifiers. A class has the responsibility to determine which data member is to be visible outside and which is not.

**Abstraction in header files:** Another type of abstraction is header file. For example, pow () function available is used to calculate the power of a number without knowing which algorithm function uses to calculate the power.

**Advantages Of Abstraction:**

* Implementation details of the class are protected from the inadvertent user level errors.
* A programmer does not need to write the low-level code.
* Data Abstraction avoids the code duplication, i.e., programmer does not have to undergo the same tasks every time to perform the similar operation.
* The main aim of the data abstraction is to reuse the code and the proper partitioning of the code across the classes.
* Internal implementation can be changed without affecting the user level code.

**Access Specifiers Implement Abstraction:**



**Imperative Programming Paradigm:**

Imperative programming focuses on HOW to execute program logic and defines control flow as statements that change a program state. Imperative programming (from Latin imperare = command) is the oldest programming paradigm. A program based on this paradigm is made up of a clearly defined sequence of instructions to a computer.

Therefore, the source code for imperative languages is a series of commands, which specify what the computer must do – and when – in order to achieve a desired result. Values used in variables are changed at program runtime. To control the commands, **control structures such as loops or branches are integrated into the code.**

Imperative programming languages are very specific, and operation is system oriented. On the one hand, the code is easy to understand; on the other hand, many lines of source text are required to describe what can be achieved with a fraction of the commands using declarative programming languages.

E.g. Fortran, Java, Pascal, ALGOL, C, C#, C+, Assembler, BASIC, COBOL, Python, Ruby

The different imperative programming languages can, in turn, be assigned to three further subordinate programming styles – structured, procedural, and modular.

The **structured programming style** extends the basic imperative principle with specific control structures: sequences, selection, and iteration. This approach is based on a desire to limit or completely avoid jump statements that make imperatively designed code unnecessarily complicated.

The **procedural approach** divides the task a program is supposed to perform into smaller sub-tasks, which are individually described in the code. This results in programming modules which can also be used in other programs. The modular programming model goes one step further by designing, developing, and testing the individual program components independently of one another. The individual modules are then combined to create the actual software. This can be further classified as:

**a) Procedural Programming Paradigm:** Procedural programming specifies the steps a program must take to reach the desired state, usually read in order from top to bottom.

**b) Object-Oriented Programming or OOP:** Object-oriented programming (OOP) organizes programs as objects, that contain some data and have some behavior.

**c) Parallel Programming:** Parallel programming paradigm breaks a task into subtasks and focuses on executing them simultaneously at the same time.

**Declarative Programming Paradigm**

Declarative programming focuses on WHAT to execute and defines program logic, but not a detailed control flow.

There is no one specific definition of the paradigm, but all definitions agree on one thing: A characteristic feature of declarative programming languages is that they always describe the desired end result rather than outlining all the intermediate work steps. In declarative programming, the solution path to reach the goal is determined automatically.

This works well, provided the specifications of the final state are clearly defined and an appropriate implementation procedure exists. If both of these conditions are met, declarative programming is very efficient.

Since declarative programming does not specifically describe the “how” but works at a very high level of abstraction, the programming paradigm also leaves room for optimization.

If a better implementation procedure is developed, the integrated algorithm can identify and use it. This makes the paradigm futureproof. The procedure for how the result is to be achieved does not have to be set in stone when writing the code.

E.g. Prolog, Lisp, Haskell, Miranda, Erlang, SQL (in the broadest sense)

Declarative paradigm can be further classified into:

**a) Logical Programming Paradigm**: Logical programming paradigm is based on formal logic, which refers to a set of sentences expressing facts and rules about how to solve a problem

**b) Functional Programming Paradigm:** Functional programming is a programming paradigm where programs are constructed by applying and composing functions.

**c) Database Programming Paradigm**: Database programming model is used to manage data and information structured as fields, records, and files.