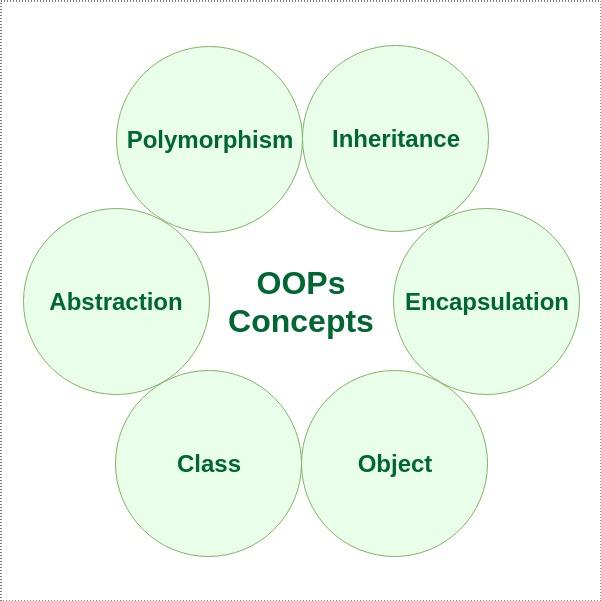
**Module II**

**Imperative Paradigm: Data Abstraction in Object Orientation**

* Grouping of data and Operations- Encapsulation,
* Overloading,
* Polymorphism,
* Inheritance,
* Initialization and Finalization
* Dynamic Binding

**Object Orientation** Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes or data members; and code, in the form of procedures, often known as methods or member functions.

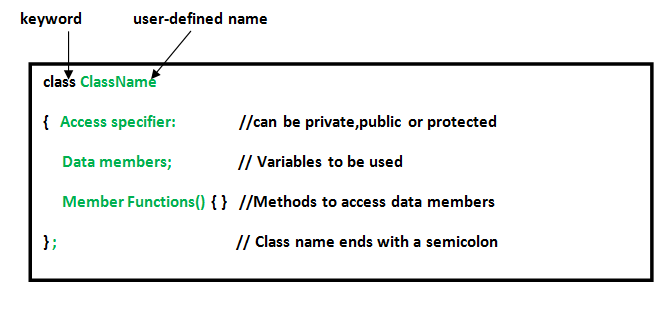
The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function



**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.

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.

A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.



**Accessing data members and member functions**: The data members and member functions of class can be accessed using the dot(‘.’) operator with the object. For example if the name of object is *obj* and you want to access the member function with the name *printName()* then you will have to write *obj.printName()* .

**Accessing Data Members**

The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object. Accessing a data member depends solely on the access control of that data member.

This access control is given by [Access modifiers in C++](https://www.geeksforgeeks.org/access-modifiers-in-c/). There are three access modifiers : **public, private and protected**.

**ClassName ObjectName;**

**Example-**

#include<iostream.h>

#include<conio.h>

class Test

{ //data members

public:

int data1;

float data2;

//member functions

public:

void func\_A()

{ data1=10;

cout<<data2;

cout<<"\nI m in func\_A";

func\_B();

}

void func\_B()

{

cout<<"\nI m in func\_B";

}

};

int main()

{

clrscr();

Test o1;//object of type Test - class

o1.data1=10; //private

// o1.data2=20; //public

o1.func\_A(); //public

// o1.func\_B(); //private

getch();

return 0;

}

**Example for data hiding**

#include<iostream.h>

#include<conio.h>

class Rectangle

{ //data members

private:

int l,b,Area,Peri;

//member functions

public:

void accept\_Data()

{

cout<<"\nEnter the values for length and breadth";

cin>>l>>b;

cal\_Area(l,b);

cal\_Peri(l,b);

}

private:

void cal\_Area(int l,int b)

{

Area = l\*b;

print\_Output(Area);

}

void cal\_Peri(int l,int b)

{

Peri = 2\*(l+b);

print\_Output(Peri);

}

void print\_Output(int out)

{

cout<<"\nOutput="<<out;

}

};

int main()

{

clrscr();

Rectangle obj;//object of type Test - class

obj.accept\_Data(); //private

// o1.data2=20; //public

// o1.func\_A(); //public

// o1.func\_B(); //private

getch();

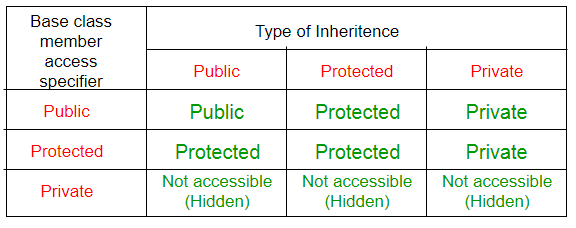
return 0;

}

**Inheritance The capability of a class to derive properties and characteristics from another class is called Inheritance.**

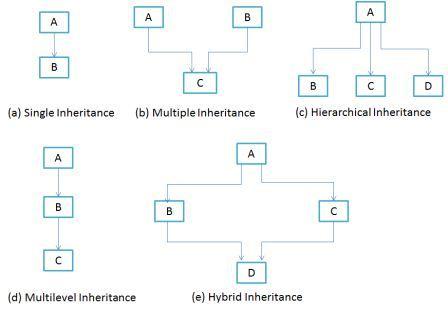
**Modes of Inheritance**

1. **Public mode**: If we derive a sub class from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class.
2. **Protected mode**: If we derive a sub class from a Protected base class. Then both public member and protected members of the base class will become protected in derived class.
3. **Private mode**: If we derive a sub class from a Private base class. Then both public member and protected members of the base class will become Private in derived class.



**Types of Inheritance In C++**

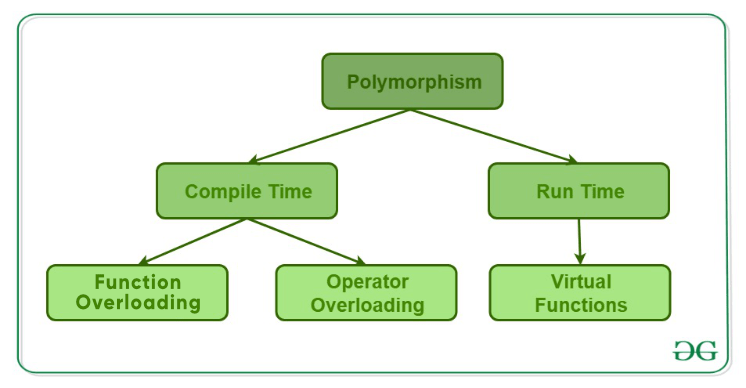
* Single Inheritance.
* **Multiple Inheritance**.(Not supported in java)
* **Multilevel Inheritance**.
* Hierarchical Inheritance.
* **Hybrid** Inheritance.



**Polymorphism:** The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form

**In C++ polymorphism is mainly divided into two types:**

* Compile time Polymorphism
* Runtime Polymorphism



**1)Compile time polymorphism**: This type of polymorphism is achieved by function overloading or operator overloading.

[**Function Overloading**](https://www.geeksforgeeks.org/function-overloading-c/): When there are multiple functions with same name but different parameters then these functions are said to be **overloaded**. Functions can be overloaded by **change in number of arguments** or/and **change in type of arguments**.

#include <iostream>

using namespace std;

void print(int i) {

cout << " Here is int " << i << endl;

}

void print(double f) {

cout << " Here is float " << f << endl;

}

void print(char const \*c) {

cout << " Here is char\* " << c << endl;

}

int main() {

print(10);

print(10.10);

print("ten");

return 0;

}

[**Operator Overloading**](https://www.geeksforgeeks.org/operator-overloading-c/): C++ also provide option to overload operators. For example, we can make the operator (‘+’) for string class to concatenate two strings.

1+2=3

[**Runtime polymorphism**](https://www.geeksforgeeks.org/virtual-functions-and-runtime-polymorphism-in-c-set-1-introduction/): This type of polymorphism is achieved by Function Overriding.

[**Function overriding**](https://www.geeksforgeeks.org/override-keyword-c/) on the other hand occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be **overridden**.

**// C++ program for function overriding**

**#include <stdc++.h>**

**using namespace std;**

**class base**

**{**

**public:**

**virtual void print ()**

**{ cout<< "print base class" <<endl; }**

**void show ()**

**{ cout<< "show base class" <<endl; }**

**};**

**class derived:public base**

**{**

**public:**

**void print ()**

**{ cout<< "print derived class" <<endl; }**

**void show ()**

**{ cout<< "show derived class" <<endl; }**

**};**

**//main function**

**int main()**

**{**

**base \*bptr;**

**derived d;**

**bptr = &d;**

**//virtual function, binded at runtime (Runtime polymorphism)**

**bptr->print();**

**// Non-virtual function, binded at compile time**

**bptr->show();**

**return 0;**

**}**

**Initialization and Finalization**

**Definition**: A *constructor* is a special class method that is called automatically to initialize an object at the beginning of its lifetime.

**Definition**: A *destructor* is a special class method that is called automatically to finalize an object at the end of its lifetime.

We shall address the following four issues related to constructors and destructors.

1. Choosing a constructor.
2. References and values.
3. Execution order.
4. Garbage Collection.

### **Choosing a Constructor**

Most OO languages permit a class to have multiple constructors. There are two principle methods used to select a specific constructor for an object.

* **Overloading**. In C++, Java, and C#, constructors act like overloaded methods. The constructor chosen is the one whose signature (number and types of parameters) matches the signature of the object declaration.
* **Named constructors**. In Smalltalk and Eiffel different constructors have different names and the object creator names the constructor that is to be used.

