DAY 1

**CLASS**:

A class is a user-defined type. A class is a collection of class members, which can be:

member variables (also called "fields"),

member functions (also called "methods"),

member types or typedefs (e.g. "nested classes"),

member templates (of any kind: variable, function, class or alias template

ACCESS SPECIFIERS:

Public : Everyone has access

Protected : Only the class itself, derived classes and friends have access

Private : Only the class itself and friends have access

FRIEND KEYWORD:

The friend keyword is used to give other classes and functions access to private and protected members of the class, even though they are defined outside the class`s scope.

FINAL KEYWORD:

Deriving a class may be forbidden with final specifier.

NESTED CLASS:

A class can also contain another class definition inside itself, which is called a "nested class”, the containing class is referred to as the "enclosing class". The nested class definition is considered to be a member of the enclosing class

STATIC MEMBERS:

A class is also allowed to have static members, which can be either variables or functions. These are considered tobe in the class' scope, but aren't treated as normal members; they have static storage duration (they exist from the start of the program to the end), aren't tied to a particular instance of the class, and only one copy exists for the entire class.

OBJECT:

Object is an instance of class.

EXAMPLE:

class MyClass {

public:

int myNum;

string myString;

void print(){

cout<<myNum<<” “<<myString;

};

int main() {

MyClass myObj;

myObj.myNum = 15;

myObj.myString = “HI";

myobj.print();

return 0;

}

**ENCAPSULATION**:

Encapsulation is an Object Oriented Programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse. Data encapsulation led to the important OOP concept of data hiding.

Data encapsulation is a mechanism of bundling the data, and the functions that use them and data abstraction is a mechanism of exposing only the interfaces and hiding the implementation details from the user.

EXAMPLE:

#include <iostream>

using namespace std;

class Employee {

private:

int salary;

public:

//Setter

void setSalary(int s) {

salary = s;

}

// Getter

int getSalary() {

return salary;

}

};

int main() {

Employee myObj;

myObj.setSalary(50000);

cout << myObj.getSalary();

return 0;

}

**ABSTRACTON**:

Data abstraction refers to providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.

In C++, classes provides great level of data abstraction. They provide sufficient public methods to the outside world to play with the functionality of the object and to manipulate object data, i.e., state without actually knowing how class has been implemented internally.

For example, your program can make a call to the sort() function without knowing what algorithm the function actually uses to sort the given values. In fact, the underlying implementation of the sorting functionality could change between releases of the library, and as long as the interface stays the same, your function call will still work.

EXAMPLE:

#include <iostream>

using namespace std;

class Adder {

public:

// constructor

Adder(int i = 0) {

total = i;

}

// interface to outside world

void addNum(int number) {

total += number;

}

// interface to outside world

int getTotal() {

return total;

};

private:

// hidden data from outside world

int total;

};

int main() {

Adder a;

a.addNum(10);

a.addNum(20);

cout << "Total " << a.getTotal() <<endl;

return 0;

}

**INHERITANCE**:

The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important feature of Object Oriented Programming.

Sub Class: The class that inherits properties from another class is called Sub class or Derived Class.

Super Class: The class whose properties are inherited by sub class is called Base Class or Super class.

Access Control and Inheritance

|  |  |  |  |
| --- | --- | --- | --- |
| **Access** | **public** | **protected** | **private** |
| Same class | yes | yes | yes |
| Derived classes | yes | yes | no |
| Outside classes | yes | no | no |

Type of Inheritance

Public Inheritance − When deriving a class from a public base class, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected members of the base class.

Protected Inheritance − When deriving from a protected base class, public and protected members of the base class become protected members of the derived class.

Private Inheritance − When deriving from a private base class, public and protected members of the base class become private members of the derived class.

Single Inheritance: In single inheritance, a class is allowed to inherit from only one class. i.e. one sub class is inherited by one base class only.

Multiple Inheritance: Multiple Inheritance is a feature of C++ where a class can inherit from more than one classes. i.e one sub class is inherited from more than one base classes.

Multilevel Inheritance: In this type of inheritance, a derived class is created from another derived class.

Hierarchical Inheritance: In this type of inheritance, more than one sub class is inherited from a single base class. i.e. more than one derived class is created from a single base class.

Hybrid Inheritance: Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.

Avoiding ambiguity using scope resolution operator:

Using scope resolution operator we can manually specify the path from which data member a will be accessed

EXAMPLE:

#include <iostream>

using namespace std;

// Base class

class Shape {

public:

void setWidth(int w) {

width = w;

}

void setHeight(int h) {

height = h;

}

protected:

int width;

int height;

};

// Derived class

class Rectangle: public Shape {

public:

int getArea() {

return (width \* height);

}

};

int main(void) {

Rectangle Rect;

Rect.setWidth(5);

Rect.setHeight(7);

cout << "Total area: " << Rect.getArea() << endl;

return 0;

}

**POLYMORPHISM**:

The word polymorphism means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

Example:

#include <iostream>

using namespace std;

class Shape {

protected:

int width, height;

public:

Shape( int a = 0, int b = 0){

width = a;

height = b;

}

int area() {

cout << "Parent class area :" <<endl;

return 0;

}

};

class Rectangle: public Shape {

public:

Rectangle( int a = 0, int b = 0):Shape(a, b) { }

int area () {

cout << "Rectangle class area :" <<endl;

return (width \* height);

}

};

class Triangle: public Shape {

public:

Triangle( int a = 0, int b = 0):Shape(a, b) { }

int area () {

cout << "Triangle class area :" <<endl;

return (width \* height / 2);

}

};

int main() {

Shape \*shape;

Rectangle rec(10,7);

Triangle tri(10,5);

shape = &rec;

shape->area();

shape = &tri;

shape->area();

return 0;

}