

Objects, Classes and Inheritance in C++

M.P.Mishra

Associate Professor

SOCIS,IGNOU

email: mpmishra@ignou.ac.in

C++ Programming

- Object
- Class
- Constructor
- Function Overloading
- Inheritance

An Object

Every object has an outside view and an inside view.

Example: student, account, item etc.

Message : An object sends a message to another object when it wants some service from the second object. That is , objects interact with each other by sending messages to perform certain functions.

Operations(Types of Methods)

The operations, which form the interface of an object may perform different kinds of functions.

Manager : Perform initialization and clean up of the instance of the class objects.

Modifier : Alter the state of an object.

Selector : Allow to access the state of an object.

Class

- Classification is a process by which we can **group objects together**.
- A class consists of objects, which have the:
 - 1. **Same interface and**
 - 2. **Same attributes.**

A class has two parts:

- i. An interface
- ii. An implementation

Class Declaration

General syntax of class construct:

```
class class_name
{
  private:
    data_type member_data;
    return type member_function( argument list...);
  public:
    data_type member_data;
    return type member_function( argument list...);
  protected:
    data_type member_data;
    return type member_function( argument list...);
};
```

Access Specifiers

Private: Private implies, that the member of the class are hidden, and can be accessed within the class.

Protected: Protected members are accessible within the class and to the derived class.

Public: These members are visible outside the class, and can be accessed from any where within the program.

Instance Variables

- The variables representing **the state of an object** are called instance variables. The variable can be of atomic or may reference other object.
- When a variable references other object then the object is a complex object .

Example:

object: Point

instance variables: x,y

operations: Display_Point()

Some Facts about Objects

- Every object has a state and a specified behavior.
- State of an object is defined by the **values in its data members**.
- Each object must maintain its own copy of the space required by the data members.
- The **behaviors which is specified** by the member functions is the **same for all the objects of the class**. Thus, only one copy of the member functions for all the objects of the class is needed.
- **Example :**
 1. All objects of Account Class will same set of attributes and behaviors.
 2. All objects of Account Class will same set of attributes and behaviors.

Another example- Point class with constructor

```
#include <iostream>
using namespace std;
class Point
{
private:
    int x;
    int y;
public: Point() //default constructor
{
    x = 0;
    y = 0;
    cout<<"Point created is at
:"<<"("<<x<<","<<y<<")"<<endl;
}
public: Point(int a , int b ) // constructor
{
    x = a;
    y = b;
    cout<<"Point created is at
:"<<"("<<x<<","<<y<<")"<<endl;
}
};
public: void display_point( )
{
    cout<<"Point is at
:"<<"("<<x<<","<<y<<")"<<endl;
}
};
int main( )
{
    Point p1;
    Point p2(3,5) ;
    p1.display_point() ;
    p2.display_point();
}
```

Point class with Constructor

The image shows a screenshot of the Dev-C++ IDE. The main window displays the source code for a C++ program. The code defines a `Point` class with two constructors: a default constructor and a parameterized constructor. The `main` function creates two `Point` objects, `p1` and `p2`, and calls their `display_point` method.

```
1 #include <iostream>
2 using namespace std;
3 class Point
4 {
5     private:
6         int x;
7         int y;
8     public: Point() //default constructor
9     {
10         x = 0;
11         y = 0;
12         cout<<"Point created is at :"<<"("<<x<<","<<y<<")<<endl;
13     }
14     public: Point(int a , int b ) // constructor
15     {
16         x = a;
17         y = b;
18         cout<<"Point created is at :"<<"("<<x<<","<<y<<")<<endl;
19     }
20     public: void display_point( )
21     {
22         cout<<"Point is at :"<<"("<<x<<","<<y<<")<<endl;
23     }
24 };
25 int main( )
26 {
27     Point p1;
28     Point p2(3,5) ;
29     p1.display_point() ;
30     p2.display_point();
31 }
```

The output window shows the execution results:

```
C:\Users\DELL\Desktop\CPP Programs\Point.exe
Point created is at :(0,0)
Point created is at :(3,5)
Point is at :(0,0)
Point is at :(3,5)
-----
Process exited after 0.06754 seconds with return value 0
Press any key to continue . . .
```

The status bar at the bottom indicates the current line is 18, column 27, and the total length of the file is 595 characters.

Constructor

- Used to **initialize object data members**.
- A class object can only be defined without specifying a set of arguments provided it either declares no constructor or it declares a default constructor.
- **Default constructor** is invoked **without user specified arguments**.

Constructor

- Whether you have written an explicit constructor or not, the compiler generates the code necessary to create the object, allocate memory for it, and perform other kinds of **behind-the-scenes initialization**.
- This happens automatically, just as it does for variables of basic types such as int, char etc.

Example: Date class with constructor

```
#include <iostream>
using namespace std;
class Date
{
private:
    int date;
    int month;
    int year;
public: Date( int i, int j, int k)
    {
        cout<< "Constructor is running..."
        <<endl;
        date= i;
        month =j;
        year=k;
    }
    public: Display_Date()
    {
        cout<<"The Date is::"<<date<<"/"
        <<month<<"/"<<year ;
    }
};
int main( )
{
    Date Today( 20, 4,2020);
    // creating object
    Today.Display_Date();
    return 0;
}
```

Output

The screenshot displays the Dev-C++ IDE interface. The main window shows the source code for a C++ program named 'Date.cpp'. The code defines a 'Date' class with private attributes 'date', 'month', and 'year', and public methods 'Date(int i, int j, int k)', 'Display_Date()', and 'main()'. The 'main()' function creates a 'Date' object and calls 'Display_Date()'. The output window on the right shows the execution results: 'Constructor is running...' followed by 'The Date is::20/4/2020'. Below this, it indicates the process exited after 0.1208 seconds with a return value of 0.

```
1 #include <iostream>
2 using namespace std;
3 class Date
4 {
5     private:
6         int date;
7         int month;
8         int year;
9     public: Date( int i, int j, int k)
10    {
11        cout<<"Constructor is running..."<<endl;
12        date= i;
13        month =j;
14        year=k;
15    }
16    public: Display_Date()
17    {
18        cout<<"The Date is: "<<this->date<<"/"<< this->month<<"/"<<this->year ;
19    }
20 }
21 };
22 int main( )
23 {
24     Date Today( 20, 4,2020); // creating object
25     Today.Display_Date();
26     return 0;
27 }
28
```

```
C:\Users\DELL\Desktop\CPP Programs\p1.exe
Constructor is running...
The Date is::20/4/2020
-----
Process exited after 0.1208 seconds with return value 0
Press any key to continue . . .
```

Inheritance

- ❖ It allows to derive a new class from the existing class.
- ❖ The derived class inherits the features of the base class (existing class)
- ❖ . Means of deriving new class from existing classes, called base classes (parent class)
- ❖ Reuses existing code
- ❖ Derived class are developed from base classes by adding or altering code of base classes

Inheritance

Inheritance in the object oriented programming is a means of defining one class in terms of another.

For example, a PG-Student is a type of Student. There are certain characteristics that are true for all Students, yet there are specific characteristics for PG-Students.

Many more examples of Inheritance can be thought of...

Account – SavingAccount

Book – StudyMaterial

Inheritance

Class derived from existing class

```
class classname: (public|protected|private) Operator basename
{
    member declarations
};
```

Public Inheritance

When deriving a class from a **public** base class:

- **public** members of the base class become **public** members of the derived class
- **protected** members of the base class become **protected** members of the derived class.
- **private** members can never be accessed directly from a derived class, it can be accessed through calls to the **public** and **protected** members of the base class.

Accessibility of Inherited Members

Base class visibility Derived class visibility

	Public	Private	Protected
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Protected	Private	Protected
Public	Public	Private	Protected

Accessibility in Public Inheritance

Accessibility	private variables	protected variables	public variables
Accessible from Base Class	yes	yes	yes
Accessible from Derived class	(not inherited)	yes	yes
Accessible from 2nd level Derived class	(not inherited)	yes	yes

Accessibility in Protected Inheritance

Accessibility	private variables	protected variables	public variables
Accessible from Base class	yes	yes	yes
Accessible from Derived class	(not inherited)	yes	yes (inherited as protected variables)
Accessible from 2nd level Derived class	(not inherited)	yes	yes

Accessibility in Private Inheritance

Accessibility	private variables	protected variables	public variables
Accessible from Base class	yes	yes	yes
Accessible from Derived class	(not inherited)	yes (inherited as private variables)	yes (inherited as private variables)
Accessible from 2nd level Derived class	(not inherited)	(not inherited)	(not inherited)

Access Control and Inheritance

Access	public	Protected	Private
Same class	Yes	Yes	Yes
Derived classes	Yes	Yes	No
Outside classes	Yes	No	No

Single Inheritance

```
#include <cstdlib>
#include <iostream>
#include <conio.h>
using namespace std;
/*Base Class*/
class A_Base
{
public:
    void ABodyTest(void);
};
void A_Base::ABodyTest(void)
{
    cout <<"I am the body of ABodyTest()inside
    A_Base"<< endl;
}
class B_Derive : public A_Base
{
public:
    void BBodyTest(void);
};
void B_Derive::BBodyTest(void)
{
    cout <<"Iam the body of BBodyTest()inside
    B_Derived"<< endl;
}
int main()
{
    //create object of derived class - class B
    B_Derive objB;
    objB.ABodyTest();
    objB.BBodyTest();
    getch();
    return 0;
}
```

Output

The image shows a screenshot of the Dev-C++ 4.9.9.2 IDE. The main window displays a C++ source file named `simpleInheritance.cpp`. The code includes headers for `<cstdlib>`, `<iostream>`, and `<conio.h>`, and uses the `std` namespace. It contains a comment: `/*C++ program to demonstrate example of simple inheritance*/` and another comment: `/*Base Class*/`. The code defines a `class A_Base`. Below the code editor, a console window titled `C:\Dev-Cpp\vid1-test.exe` shows the program's output: `I am the body of ABodyTest() inside A_Base` and `I am the body of BBodyTest() inside B_Derived`. The Windows taskbar at the bottom shows the Start button, several open applications (Dev-C++, Video EMPC, inheritance tu..., Microsoft Po..., 2 Firefox), and the system clock displaying 11:50 AM on 27:31.

```
#include <cstdlib>
#include <iostream>
#include <conio.h>
using namespace std;

/*C++ program to demonstrate example of simple inheritance*/

/*Base Class*/
class A_Base
```

```
C:\Dev-Cpp\vid1-test.exe
I am the body of ABodyTest() inside A_Base
I am the body of BBodyTest() inside B_Derived
```

Constructor Call in Inheritance

```
#include <cstdlib>
#include <iostream>
#include <conio.h>
using namespace std;
class Base
{
    int i;
public:
    Base()
    {
        cout << "Base default constructor call" << endl;
    }
};
class Derived : public Base {
    int j;
public:
    Derived()
    {
        cout << "Derived class default constructor
            call" << endl;
    }
};
Derived(int x)
{
    j=x;
    cout << "Derived parameterized constructor
        call" << endl;
};
int main()
{
    Base B;
    Derived D1;
    Derived D2(50);
    getch();
}
```

Output

The screenshot shows the Dev-C++ IDE with a project named 'Vid1_inheritanceconstructor.dev'. The output window displays the following text:

```
Base default constructor call  
Base default constructor call  
Derived class default constructor call  
Base default constructor call  
Derived parameterized constructor call
```

The source code window shows the following code:

```
{  
    cout << "Derived class default constructor call"<<endl;  
}  
Derived(int x)  
{  
    j=x;  
    cout << "Derived parameterized constructor call"<<endl;  
}
```

The status bar at the bottom indicates '14: 1' and '35 Lines in file'. The Windows taskbar at the bottom shows the Start button and several open applications: Dev-C++, Video EMPC C++, inheritance tut..., Microsoft Powe..., 2 Firefox, and C:\Dev-Cpp\Wid... The system clock shows 1:12 PM.

Function Overloading and Overriding

- Overloading
 - allows different methods to have the same name, but different signatures.
- Overriding
 - When both base class and derived class have a member function with same name and arguments (number and type of arguments).
 - When an object of the derived class call the member function which exists in both classes (base and derived), the member function of the derived class is invoked and the function of the base class is ignored.

Function Overloading: An Example

```
#include<iostream>
#include<conio.h>

//Single line comment
using namespace std;
class SUM
{
int num4;
// Function to find sum of three number
public: void sum(int num1, int num2, int num3)
{
    num4 = num1+num2+num3;
}
public: void sum(int num1, int num2)
{
    num4 = num1+num2;
}

public: void display()
{
    cout<<"The Sum is :"<<num4<<endl;
}
}; //This is where the execution of program
    begins
int main()
{
    SUM S1;
    cout<<"Welcome to the program"<<endl;
    S1.sum(4,5);
    S1.display();
    S1.sum(4,5,6);
    S1.display();
    return 0;
}
```

Function Overloading: An Example

The screenshot displays the Dev-C++ IDE with a C++ program demonstrating function overloading. The code defines a class `SUM` with two `sum` methods: one for three numbers and one for two numbers. The `main` function creates a `SUM` object `S1` and calls both methods, demonstrating how the same function name can be used with different parameter lists.

```
1 #include<iostream>
2 #include<conio.h>
3
4
5 //Single line comment
6 using namespace std;
7 class SUM
8 {
9     int num4;
10
11 // Function to find sum of three number
12 public: void sum(int num1, int num2, int num3)
13 {
14     num4 = num1+num2+num3;
15 }
16 public: void sum(int num1, int num2)
17 {
18     num4 = num1+num2;
19 }
20 public: void display()
21 {
22     cout<<"The Sum is :"<<num4<<endl;
23 }
24 };//This is where the execution of program begins
25 int main()
26 {
27     SUM S1;
28     cout<<"Welcome to the program"<<endl;
29     S1.sum(4,5);
30     S1.display();
31     S1.sum(4,5,6);
32     S1.display();
33     return 0;
34 }
```

The execution output shows the program's runtime behavior:

```
C:\Users\DELL\Desktop\CPP Programs\SUM.exe
Welcome to the program
The Sum is :9
The Sum is :15
-----
Process exited after 0.07147 seconds with return value 0
Press any key to continue . . .
```

Thank You