ORIGINAL RESEARCH PAPER

Volume-8 | Issue-6 | June-2019 | PRINT ISSN No. 2277 - 8179

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH

CONCEPTS OF OBJECT ORIENTED PROGRAMMING

Engineering

Prof. Chetashri S. Bhusari Lecturer , Vidyalankar Polytechnic Wadala, Mumbai

Prof. Shonal A Vaz* Lecturer, Vidyalankar Polytechnic Wadala, Mumbai *Corresponding Author

Prof. Supriya Angne Vidyalankar Polytechnic Wadala, Mumbai

ABSTRACT

Object-Oriented Programming (OOP) uses "objects" to model realworld objects. Object-Oriented Programming (OOP) consist of some important concepts namely Encapsulation, Polymorphism, Inheritance and Abstraction. These features are generally referred to as the OOPS concepts. If you are new to object oriented approach for software development, an object in OOP has some state and behavior. In Java, the state is the set of values of an object's variables at any particular time and the behaviour of an object is implemented as methods. Class can be considered as the blueprint or a template for an object and describes the properties and behavior of that object, but without any actual existence. An object is a particular instance of a class which has actual existence and there can be many objects (or instances) for a class. Static variables and methods are not purely object oriented because they are not specific to instances (objects) but common to all instances.

KEYWORDS

I. INTRODUCTION

Object oriented programming – As the name suggests uses objects in programming. Object oriented programming aims to implement real world entities like inheritance, hiding, polymorphism in programming. The main aim of OOP is to bind together the data and the functions that operates on them so that no other part of code can access this data except that function.

Object Oriented programming is a programming style that is associated with the concept of Class, Objects and various other concepts revolving around these two, like Inheritance, Polymorphism, Abstraction, Encapsulation etc.

II. CLASSES AND OBJECTS

Class: The building block of C++ that leads to Object Oriented programming is a **Class**. 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. A class is like a blueprint for an object. For Example: Consider the Class of Cars. and member functions defines 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 applying 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.

Defining Class and Declaring Objects

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.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 end.

Declaring Objects: When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

Syntax:
ClassName ObjectName;
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++. There are three access modifiers: **public, private and protected**.

Member Functions in Classes

There are 2 ways to define a member function:

- Inside class definition
- Outside class definition

To define a member function outside the class definition we have to use the scope resolution operator (: :) along with class name and function name.

III. DATA ABSTRACTION AND 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. **Encapsulation** is defined as wrapping up of data and information under a single unit. In Object Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulates them. Consider a real life example of encapsulation, in a company there are different sections like the accounts section, finance section, sales section etc. The finance section handles all the financial transactions and keep records of all the data related to finance.

IV. INHERITANCE

Inheritance in Object Oriented Programming can be described as a process of creating new classes from existing classes. New classes inherit some of the properties and behavior of the existing classes. An existing class that is "parent" of a new class is called a base class. Inheritance is a technique of code reuse. The most frequent use of inheritance is for deriving classes using existing classes, which provides reusability. The existing classes remain unchanged.When a class is derived from more than one class, all the derived classes have similar properties to those of base classes.

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the baseclass, and the new class is referred to as the derived class.

Base and Derived Classes

A class can be derived from more than one classes, which means it can inherit data and functions from multiple base classes. To define a derived class, we use a class derivation list to specify the base class(es). A class derivation list names one or more base classes and has the form

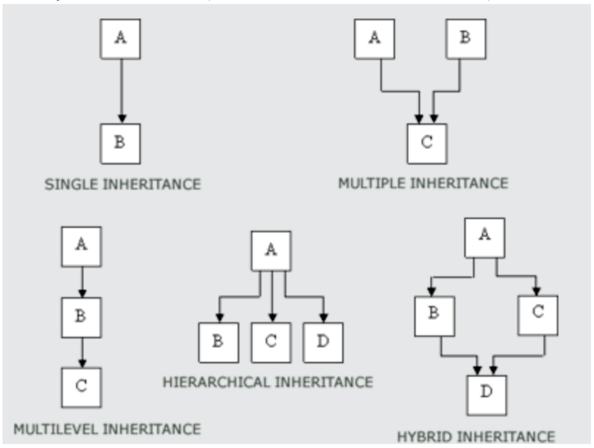
class derived-class: access-specifier base-class

Where access-specifier is one of **public, protected,** or **private**, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.

Types of Inheritance :

In C++, we have 5 different types of Inheritance. Namely,

1. Single Inheritance 2. Multiple Inheritance 3.Hierarchical Inheritance 4. Multilevel Inheritance
2. Hybrid Inheritance (also known as Virtual Inheritance)



1.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.

Syntax:

```
class subclass_name : access_mode base_class
{
 //body of subclass
};
```

2. 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**.

Syntax:

```
class subclass_name : access_mode base_class1, access_mode
base_class2, ....
{
 //body of subclass
};
```

Here, the number of base classes will be separated by a comma (', ') and access mode for every base class must be specified.

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

4.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.

5. Hybrid (Virtual) Inheritance: Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance. Below image shows the combination of hierarchical and multiple inheritance:

V. POLYMORPHISM IN C++

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. Real life example of polymorphism, a person at a same time can have different characteristic. Like a man at a same time is a father, a husband, a employee. So a same person posses have different behavior in different situations. This is called polymorphism. Polymorphism is considered as one of the important features of Object Oriented Programming.

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**: 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**.

Operator Overloading

- C++ also provide option to overload operators. For example, we can make the operator ('+') for string class to concatenate two strings. We know that this is the addition operator whose task is to add to operands. So a single operator '+' when placed between integer operands , adds them and when placed between string operands, concatenates them.
- **Function overriding** 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**.

Polymorphism is a feature of OOP that allows the object to behave differently in different conditions. In C++ we have two types of polymorphism:

1) Compile time Polymorphism – This is also known as static (or early) binding.
2) Runtime Polymorphism – This is also known as dynamic (or late) binding.
1) Compile time Polymorphism

Function overloading and Operator overloading are perfect example of Compile time polymorphism.

1) Compile time Polymorphism Example

In this example, we have two functions with same name but different number of arguments. Based on how many parameters we pass during function call determines which function is to be called, this is why it is considered as an example of polymorphism because in different conditions the output is different. Since, the call is determined during compile time thats why it is called compile time polymorphism.

VI. CONSTRUCTOR AND DESTRUCTOR

A class **constructor** is a special member function of a class that is executed whenever we create new objects of that class. A constructor will have exact same name as the class and it does not have any return type at all, not even void. Constructors can be very useful for setting initial values for certain member variables.

A special method of the class that will be automatically invoked when an instance of the class is created is called a constructor.

The main use of constructors is to initialize private fields of the class while creating an instance for the class. When you have not created a constructor in the class, the compiler will automatically create a default constructor in the class.

The default constructor initializes all numeric fields in the class to zero and all string and object fields to null.

Constructors can be divided into 5 types:

- Default Constructor.
- Parametrized Constructor.
- Copy Constructor.
- Overloaded Constructor
- Multiple Constructor

Default Constructor :

```
#include <iostream.h>
class Line
{
 public:
void setLength( double len );
double getLength( void );
Line();  // This is the constructor
private:
double length;
};

// Member functions definitions including constructor
Line::Line(void) {
cout << "Object is being created" << endl;
}
void Line::setLength( double len ) {
length = len;
}
double Line::getLength( void ) {
return length;
}
// Main function for the program
int main()
{
Line line;
// set line length
line.setLength(6.0);
cout << "Length of line : " << line.getLength() <<endl;
return 0;
}
```

Parameterized Constructor:

A default constructor does not have any parameter, but if you need, a constructor can have parameters. This helps you to assign initial value to an object at the time of its creation as shown in the following example −

```
#include <iostream>
using namespace std;
class Line {
public:
void setLength( double len );
double getLength( void );
Line(double len);  // This is the constructor

private:
double length;
};
// Member functions definitions including constructor
Line::Line( double len)
 {
cout << "Object is being created, length = " << len << endl;
length = len;
}
void Line::setLength( double len ) {
length = len;
}
double Line::getLength( void ) {
return length;
}
// Main function for the program
int main()
{
Line line(10.0);
// get initially set length.
cout << "Length of line : " << line.getLength() <<endl;
// set line length again
line.setLength(6.0);
cout << "Length of line : " << line.getLength() <<endl;
return 0;
}
```

Copy Constructor :A copy constructor is a member function which initializes an object using another object of the same class. A copy constructor has the following general function prototype:

ClassName (const ClassName &old_obj);

Following is a simple example of copy constructor.

```
#include<iostream>
using namespace std;
class Point
{
private:
int x, y;
public:
Point(int x1, int y1) { x = x1; y = y1; }
// Copy constructor
Point(const Point &p2) {x = p2.x; y = p2.y; }
int getX()          {  return x; }
int getY()          {  return y; }
};
int main()
{
Point p1(10, 15); // Normal constructor is called here
Point p2 = p1; // Copy constructor is called here
// Let us access values assigned by constructors
cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
cout << "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY();
}
```

Overloaded Constructor :

Constructor can be overloaded in a similar way as overloading. Overloaded constructors have the same name (name of the class) but different number of arguments. Depending upon the number and type of arguments passed, specific constructor is called. Since, there are multiple constructors present, argument to the constructor should also be passed while creating an object.

Program :

```
#include <iostream>
using namespace std;
class Area
{
private:
int length;
int breadth;
public:
// Constructor with no arguments
Area(): length(5), breadth(2) { }
// Constructor with two arguments
Area(int l, int b): length(l), breadth(b){ }
void GetLength()
{
cout << "Enter length and breadth respectively: ";
cin >> length >> breadth;
}
int AreaCalculation() {  return length * breadth;  }
void DisplayArea(int temp)
{
cout << "Area: " << temp << endl;
}
};
int main()
{
Area A1, A2(2, 1);
int temp;
cout << "Default Area when no argument is passed." ;       temp =
A1.AreaCalculation();
A1.DisplayArea(temp);
cout << "Area when (2,1) is passed as argument.";
temp = A2.AreaCalculation();
A2.DisplayArea(temp);
}
```

For object *A1*, no argument is passed while creating the object.

Thus, the constructor with no argument is invoked which initializes length to 5 and breadth to 2. Hence, area of the object *A1* will be 10.

For object *A2*, 2 and 1 are passed as arguments while creating the object.

Thus, the constructor with two arguments is invoked which initializes *length* to *l* (2 in this case) and *breadth* to *b* (1 in this case). Hence, area of the object *A2* will be 2.

VII. POINTERS IN C++

A pointer is a variable whose value is the address of another variable. Like any variable or constant, you must declare a pointer before you can work with it. The general form of a pointer variable declaration is −
type *var-name;

Here, type is the pointer's base type; it must be a valid C++ type and var-name is the name of the pointer variable. The asterisk you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Following are the valid pointer declaration.

```
int    *ip;    // pointer to an integer
double *dp;    // pointer to a double
float  *fp;    // pointer to a float
char   *ch     // pointer to character
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address.

There are few important operations, which we will do with the pointers very frequently. **(a)** We define a pointer variable. **(b)** Assign the address of a variable to a pointer. **(c)** Finally access the value at the address available in the pointer variable. This is done by using unary operator * that returns the value of the variable located at the address specified by its operand. Following example makes use of these operations −

```
#include <iostream.h>
using namespace std;
int main ()
 {
int  var = 20;   // actual variable declaration.
int  *ip;        // pointer variable
ip = &var;       // store address of var in pointer variable
cout << "Value of var variable: ";
cout << var << endl;
// print the address stored in ip pointer variable
cout << "Address stored in ip variable: ";
cout << ip << endl;
// access the value at the address available in pointer
cout << "Value of *ip variable: ";
cout << *ip << endl;
return 0;
}
```

VIII. CONCLUSION

We understood concepts of oop.