OOPs Components

-by Praveen Choudhary

components of oops

- ***Class.***
- ***Objects.***
- ***Abstraction.***
- ***Encapsulation.***
- ***Inheritance.***
- ***Data Hiding.***
- ***Polymorphism***.

class

- Classes has the data and its associated function wrapped in it. Classes are also known as a collection of similar objects or objects of same type. In the OOPs concept the variables declared inside a class are known as "Data Members" and the functions are known as "Member Functions".

- Syntax:

```
class class-name
{
private:
variable declaration;
function declaration;
public:
variable declaration;
function declaration;
};
```

COMPONENTS OF CLASS

- **MEMBER DATA : Member data is the attribute of class.It may private , public or protected.We can use these access specifier.**

- **MEMBER FUNCTION : Member function is the operation performed by the class on the member data.Member function can be protected by using three keywords private , public or protected.**

object

- Object is a real world entity.
- Objects sometimes correspond to things found in the real world. For example, a graphics program may have objects such as "circle," "square," "menu." An online shopping system will have objects such as "shopping cart," "customer," and "product." The shopping system will support behaviors such as "place order," "make payment," and "offer discount.“
- Syntax :

```
void main()
{
   class_name obj_name;
 }
```

a program to illustrate oops concept

```
#include<iostram.h>
#include<conio.h>
class sum
{
private :
int a , b , c;
public :
  void input()
{
cout<<"enter two numbers";
cin>>a>>b;
}
void display()
{
    c = a + b;
    cout<<"sum of two numbers is : "<<c;
}
};
void main()
{
    sum s;
    s.input();
    s.display();
    getch();
}
```

abstraction

- **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.**

- **Data abstraction is a programming (and design) technique that relies on the separation of interface and implementation.**

- 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.

encapsulation

- Encapsulation is the packing of data and functions into a single component. The features of encapsulation are supported using classes in most object-oriented programming languages, although other alternatives also exist. It allows selective hiding of properties and methods in an object by building an impenetrable wall to protect the code from accidental corruption.

- In programming languages, encapsulation is used to refer to one of two related but distinct notions, and sometimes to the combination thereof:

* A language mechanism for restricting access to some of the objects components.

* A language construct that facilitates the bundling of data with the methods (or other functions) operating on that data.

- Some programming language researchers and academics use the first meaning alone or in combination with the second as a distinguishing feature of object-oriented programming, while other programming languages which provide lexical closures view encapsulation as a feature of the language orthogonal to object orientation.

inheritance

- 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 base class, and the new class is referred to as the derived class.

Base & 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.

- **Consider a base class Shape and its derived class Rectangle as follows:**

```
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);

   // Print the area of the object.
   cout << "Total area: " << Rect.getArea() << endl;

   return 0;
}
```

data hiding

- Data hiding is a software development technique specifically used in object-oriented programming (OOP) to hide internal object details (data members). Data hiding ensures exclusive data access to class members and protects object integrity by preventing unintended or intended changes.
- Data hiding also reduces system complexity for increased robustness by limiting interdependencies between software components.
  Data hiding is also known as data encapsulation or information hiding.

polymorphism

- **Polymorphism is a greek word in which ‘poly’ means ‘many’ and ‘morph’ means ‘forms’.**
- **In object oriented programming, polymorphism (from the Greek meaning "having multiple forms") is the characteristic of being able to assign a different meaning or usage to something in different contexts - specifically, to allow an entity such as a variable, a function, or an object to have more than one form. There are several different kinds of polymorphism.**

a) **Run time polymorphism** : The appropriate member function could be selected while the programming is running. This is known as run-time polymorphism. The *run-time* polymorphism is implemented with inheritance and virtual functions.

Virtual function : A function qualified by the virtual keyword. When a virtual function is called via a pointer, the class of the object pointed to determines which function definition will be used. Virtual functions implement polymorphism, whereby objects belonging to different classes can respond to the same message in different ways.

b) <u>Compile time polymorphism :</u> The compiler is able to select the appropriate function for a particular call at compile-time itself. This is known as compile-time polymorphism. The *compile-time* polymorphism is implemented with templates.

- Function name overloading : Using a single function name to perform different types of tasks is known as function overloading.

  Using the concept of function overloading, design a family of functions with one function name but with different argument lists. The function would perform different operations depending on the argument list in the function call. The correct function to be invoked is determined by checking the number and type of the arguments but not on the function type.

- Operator overloading : The process of making an operator to exhibit different behaviours in different instances is known as operator overloading.

The general form of an operator function is:

```
return type classname: : operator(op-arglist)
          {
              Functionbody;
            }
```