RGPVNOTES.IN

Program : **B.Tech**

Subject Name: **Object Oriented Programming & Methodology**

Subject Code: **CS-305**

Semester: **3rd**

UNIT-II

Encapsulation and Data Abstraction:

Definition: It is the method of combining the data and functions inside a class. Thus the data gets hidden from being accessed directly from outside the class. This is similar to a capsule where several medicines are kept inside the capsule thus hiding them from being directly consumed from outside. All the members of a class are private by default, thus preventing them from being accessed from outside the class.

Why Encapsulation

Encapsulation is necessary to keep the details about an object hidden from the users of that object. Details of an object are stored in its data members. This is the reason we make all the member variables of a class private and most of the member functions public. Member variables are made private so that these cannot be directly accessed from outside the class and so most member functions are made public to allow the users to access the data members through those functions.

For example, we operate a washing machine through its power button. We switch on the power button, the machine starts and when we switch it off, the machine stops. We don't know what mechanism is going on inside it. That is encapsulation.

Benefits of Encapsulation

There are various benefits of **encapsulated classes**.

- Encapsulated classes reduce complexity.
- **Help protect our data.** A client cannot change an Account's balance if we encapsulate it.
- **Encapsulated classes are easier to change.** We can change the privacy of the data according to the requirement without changing the whole program by using access modifiers (public, private, protected). For example, if a data member is declared private and we wish to make it directly accessible from anywhere outside the class, we just need to replace the specifier private by public.

Let's see an example of Encapsulation.

```
#include<iostream>

usingnamespacestd;

classRectangle
{
        intlength;
        intbreadth;
        public:
                voidsetDimension(intl,intb)
                {
                        length=l;
                        breadth=b;
                }
```

```
                intgetArea()
                {
                        returnlength*breadth;
                }
};

intmain()
{
        Rectanglert;
        rt.setDimension(7,4);
        cout<<rt.getArea()<<endl;
        return0;
}
```

Data Abstraction

Data abstraction is one of the most essential and important feature of object oriented programming in C++. Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.
Consider a real life example of a man driving a car.The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. .

Advantages of Data Abstraction:

- Helps the user to avoid writing the low level code
- Avoids code duplication and increases reusability.
- Can change internal implementation of class independently without affecting the user.
- Helps to increase security of an application or program as only important details are provided to the user.

Concept of Objects: State, Behavior &Identity of an object

Object: Object is a real world entity, for example, chair, car, pen, mobile, laptop etc.In other words, object is an entity that has state and behavior. Here, state means data and behavior means functionality. Object is a runtime entity, it is created at runtime. Object is an instance of a class. All the members of the class can be accessed through object.

Bundling code into individual software objects provides a number of benefits, including:

1. **Modularity:** The source code for an object can be written and maintained independently of the source code for other objects. Once created, an object can be easily passed around inside the system.
2. **Information-hiding:** By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.
3. **Code re-use:** If an object already exists (perhaps written by another software developer), you can use that object in your program. This allows specialists to implement/test/debug complex, task-specific objects, which you can then trust to run in your own code.
4. **Pluggability and debugging ease:** If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its

replacement. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, you replace *it*, not the entire machine.

State: Represents data (value) of an object. Dogs have state (name, color, and breed, hungry)

Behavior: Represents the behavior (functionality) of an object such as deposit, withdraw etc.

Identity: Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But it is used internally by the JVM to identify each object uniquely.

Classes: Identifying classes and candidate for classes Attribute and services

Class: The building block of C++ that leads to Object Oriented programming is a **Class**.

- A class is an abstract data type similar to '**C structure**'.
- The Class representation of objects and the sets of operations that can be applied to such objects.
- The class consists of Data members and methods.

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**. There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members 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 apply brakes, increase speed etc.

The primary purpose of a class is to hold data/information. This is achieved with attributes which are also known as data members. The member functions determine the behavior of the class i.e. provide a definition for supporting various operations on data held in form of an object.

Class Advantages

- A class is a user-defined and reusable entity.
- A class reduces complicity of holding data.
- Classes' just holds data and do operations with help of member variables and member functions.
- A class maintains software quality and is well structured.
- An Object is a realization of the particular class.

Syntax

classclassname {


```
   Access - Specifier :
   Member Varibale Declaration;
   Member Function Declaration;
}
```

Example

```
#include<iostream>
usingnamespacestd;

// Class Declaration

class Person {
//Access - Specifier
public:
//Member Variable Declaration
string name;
int number;

//Member Functions read() and print() Declaration

voidread(){
//Get Input Values For Object Variables
cout<<"Enter the Name :";
cin>> name;

cout<<"Enter the Number :";
cin>> number;
  }

voidprint(){
//Show the Output
cout<<"Name : "<< name <<",Number : "<< number <<endl;
  }
};

intmain(){
// Object Creation For Class
  Person obj;

cout<<"Simple Class and Object Example Program In C++\n";

obj.read();
obj.print();

return0;
}
```


Access Modifiers

Access modifiers are used to implement important feature of Object Oriented Programming known as **Data Hiding**. Consider a real life example: What happens when a driver applies

brakes? The car stops. The driver only knows that to stop the car, he needs to apply the brakes. He is unaware of how actually the car stops. That is how the engine stops working or the internal implementation on the engine side. This is what data hiding is.
Access modifiers or Access Specifiers in a class are used to set the accessibility of the class members. That is, it sets some restrictions on the class members not to get directly accessed by the outside functions.
There are 3 types of access modifiers available in C++:

1. **Public**
2. **Private**
3. **Protected**

Note: If we do not specify any access modifiers for the members inside the class then by default the access modifier for the members will be **Private**.

Public: All the class members declared under public will be available to everyone. The data members and member functions declared public can be accessed by other classes too. The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.
Private: The class members declared as **private** can be accessed only by the functions inside the class. They are not allowed to be accessed directly by any object or function outside the class. Only the member functions or the friend functions are allowed to access the private data members of a class.
Protected: Protected access modifier is similar to that of private access modifiers, the difference is that the class member declared as Protected are inaccessible outside the class but they can be accessed by any subclass (derived class) of that class.

Demonstrating use of private and public Data members and Member functions

```
#include <iostream>
usingnamespace std;

class Example
{
        private:
                intval;
        public:
                //function declarations
                voidinit_val(int v);
                voidprint_val();
};
//function definitions
void Example::init_val(int v)
{
        val=v;
}

void Example::print_val()
{
        cout<<"val: "<<val<<endl;
}
int main()
{
        //create object
```

```
        Example Ex;
        Ex.init_val(100);
        Ex.print_val();

        return 0;
}
```

Static Members of Class: Data member and Member function

A data member of a class can be qualified as static. The properties of a static member variable are similar to that of a C static variable. A static member variable has certain special characteristics. These are:

- It is initialized to zero when the first object of its class is created. No other initialization is permitted.
- Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.
- It is visible only within the class, but its lifetime is the entire program.

Static variables are normally used to maintain values common to the entire class. For example, a static data member can be used as a counter that records the occurrences of all the objects. Program illustrates the use of a static data member.

```
#include
using namespace std;

class item
{
        staticint count;
        int number;
        public:
                voidgetdata(int a)
                {
                        number = a;
                        count ++;
                }
                voidgetcount(void)
                {
                        cout<< "Count: ";
                        cout<< count <<"\n";
                }
};
int item :: count;

int main()
{
        itema,b,c;
        a.getcount();
        b.getcount();
        c.getcount();

        a.getdata(100);
        b.getdata(200);
```

```
        c.getdata(300);

        cout<< "After reading data"<<"\n";
        a.getcount();
        b.getcount();
        c.getcount();
        return 0;
}
```

The output of the program would be:
Count: 0
Count: 0
Count: 0
After reading data
Count: 3
Count: 3
Count: 3

Static Member Function

Like static member variable, we can also have static member function. A member function that is declared static has the following properties:

- A static function can have access to only static members declared in the class.
- A static member function can be called the class name as follows:

Class-name:: function-name

Static member functions are used to maintain a single copy of a class member function across various objects of the class. Static member functions can be called either by itself, independent of any object, by using class name and :: (scope resolution operator) or in connection with an object.

Characteristic static member functions are:

- A static member function can only have access to other static data members and functions declared in the same class.
- A static member function can be called using the class name with a scope resolution operator instead of object name.
- Global functions and data may be accessed by static member function.
- A static member function does not have this Pointer.
- There can not be a static and a non-static version of the same function.
- They cannot be declared as const or volatile.
- A static member function may not be virtual.

Instance

An instance is an object of a class type created via the new operator.
For example:

String *str = new String();
str is a pointer to an instance of class String.
if you won`t create memory for an object so we call that **object** as **instance**.
example: Student std;
here**Student** is a class and **std** is a instance(means a just a copy that class),with this we won`t do anything until unless we create a memory for that.
s

Message Passing

Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

Message Passing is nothing but sending and receiving of information by the objects same as people exchange information. So this helps in building systems that simulate real life. Following are the basic steps in message passing.

- Creating classes that define objects and its behavior.
- Creating objects from class definitions
- Establishing communication among objects

In OOPs, Message Passing involves specifying the name of objects, the name of the function, and the information to be sent.

Construction and Destruction of Object

The process of initializing and starting up a class is called construction. The opposite of this is the process of tearing down a class and cleaning it up, which is called destruction or finalization. Class construction occurs either when an object is being initialized or upon first reference to a type.

In typical case, the process is as follows:

- Calculate the size of an object - the size is mostly the same as that of the class but can vary. When the object in question is not derived from a class, but from a prototype instead, the size of an object is usually that of the internal data structure (a hash for instance) that holds its slots.
- allocation - allocating memory space with the size of an object plus the growth later, if possible to know in advance
- binding methods - this is usually either left to the class of the object, or is resolved at dispatch time, but nevertheless it is possible that some object models bind methods at creation time.
- calling an initializing code of super class
- calling an initializing code of class being created

Construction Rules

The rule for constructing an object of a simple class is:
1. Call the constructor/initializer for each data member, in sequence.

2. Call the constructor for the class.

The rule for constructing an object of a derived class is:
1. Call the constructor for the base class (which recursively calls the constructors needed to Completely initialize the base class object.)
2. Call the constructor/initializer for each data member of the derived class, in sequence.
3. Call the constructor for the derived class

Object Destruction

It is generally the case that after an object is not used, it is removed from memory to make room for other programs or objects to take that object's place. However, if there is sufficient memory or a program has a short run time, object destruction may not occur, memory simply being deallocated at process termination. In some cases object destruction simply consists of deallocating the memory, particularly in garbage-collected languages, or if the "object" is actually a plain old data structure. In other cases some work is performed prior to deallocation, particularly destroying member objects (in manual memory management), or deleting references from the object to other objects to decrement reference counts (in reference counting). This may be automatic, or a special destruction method may be called on the object.

Destruction Rules

When an object is deleted, the destructors are called in the opposite order.
The rule for an object of a derived class is:
1. Call the destructor for the derived class.
2. Call the destructor for each data member object of the derived class in reverse sequence.
3. Call the destructor for the base class.

Constructor

A constructor is a special member function of the class which has the same name as that of the class. It is automatically invoked when we declare/create new objects of the class. It is called constructor, because it constructs the value of data members of the class.

The constructor functions have some special characteristics as follows –

1. They should be declared in the public section.
2. They are called automatically when the objects are created.
3. They do not have return types, not even void and they cannot return values.
4. They cannot be inherited, though a derived class can call the base class constructor.
5. Like other C++ functions, they can have default arguments.
6. Constructors cannot be virtual.
7. They cannot be referred by their addresses.
8. They make ‘implicit calls’ to the operators new and delete when memory allocation is required.

Syntax

classclass_name {


```
public:
class_name() {
      // Constructor code
   }

   //... other Variables & Functions
}
```

Types Of Constructor

- Default Constructor
- Parameterized Constructor
- Copy Constructor

Default Constructor

A default constructor does not have any parameters or if it has parameters, all the parameters have default values.

Syntax

```
classclass_name {
public:
   // Default constructor with no arguments
class_name();
   // Other Members Declaration
}
```

Parameterized Constructor

If a Constructor has parameters, it is called a Parameterized Constructor. Parameterized Constructors assist in initializing values when an object is created.

Example Program for Parameterized Constructor In C++

```
#include<iostream>
#include <iostream>
using namespace std;
class Employee {
public:
int id;//data member (also instance variable)
string name;//data member(also instance variable)
float salary;
Employee(int i, string n, float s)
      {
id = i;
name = n;
salary = s;
      }
void display()
      {
```

```
cout<<id<<"  "<<name<<"  "<<salary<<endl;
      }
};
int main(void) {
   Employee e1 =Employee(101, "Sonoo", 890000); //creating an object of Employee
   Employee e2=Employee(102, "Nakul", 59000);
e1.display();
e2.display();
return 0;
}
```

Copy Constructor

A copy constructor is a like a normal parameterized Constructor, but which parameter is the same class object. Copy constructor uses to initialize an object using another object of the same class.

The copy constructor is used to −

- Initialize one object from another of the same type.
- Copy an object to pass it as an argument to a function.
- Copy an object to return it from a function.

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 (constClassName&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; }

   intgetX()          {  return x; }
   intgetY()          {  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();

    return 0;
}
```

Destructor

Destructor: it is a member function which destructs or deletes an object. The destructor function has the same as the constructor, but it is preceded by a (tilde sign)

1. Destructors are special member functions of the class required to free the memory of the object whenever it goes out of scope.
2. Destructors are parameter less functions.
3. Name of the Destructor should be exactly same as that of name of the class. But preceded by '~' (tilde).
4. Destructor does not have any return type. Not even void.
5. The Destructor of class is automatically called when object goes out of scope.

Example

```
#include<iostream>
usingnamespace std;

class Marks
{
        public:
                intmaths;
                int science;

        //constructor
        Marks()
        {

                cout<< "Inside Constructor"<<endl;
                cout<< "C++ Object created"<<endl;
        }

  //Destructor
  ~Marks()
{
        cout<< "Inside Destructor"<<endl;
        cout<< "C++ Object destructed"<<endl;
  }
};

int main( )
{
  Marks m1;
  Marks m2;
return 0;
```

```
}
```

Output

```
Inside Constructor
C++ Object created
Inside Constructor
C++ Object created

Inside Destructor
C++ Object destructed
Inside Destructor
C++ Object destructed
```

RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at https://qp.rgpvnotes.in .

If you have any queries or you want to submit your study notes please write us at rgpvnotes.in@gmail.com

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in