OBJECT ORIENTED PROGRAMMING HANDBOOK FOR CAMPUS PLACEMENT

ABSTRACT

This Handbook covers all the necessary topics from OOPS which are helpful for your campus interview preparation make sure you understand all points clearly and practice them well. No interview is done without asking questions from OOPS Concept so make OOPS as your strong point and become ready to ace your placements.

YASH NEMADE
BTECH-Information Technology
Yashnemade16@gmail.com

Object **Oriented** Programming Notes

Q What are the Pillars of Object-Oriented programming?

- Object oriented programming comprises of 4 main pillars on which the software development process takes place
  - ➔ Encapsulation
  - ➔ Abstraction
  - ➔ Inheritance
  - ➔ Polymorphism

Q Explain Encapsulation?

- ➔ Encapsulation is an important feature to be considered during the development process which comes the umbrella of oops
- ➔ Encapsulation refers to the biding of data that means protecting our data from unauthorized access
- ➔ In programming it becomes necessary to ensure security in order to protect our data and thus every software development uses the principle of Encapsulation
- ➔ Encapsulation is also called as capsulating the data i.e creating a barrier to protect data
- ➔ In the world of object-oriented programming encapsulation principle can be achieved using private access specifier
  1) Once the programmer declares the member of class in private mode then member of class can be accessed inside the class only they directly cannot be accessed from the main function that means here protection is enabled and thus the principle of encapsulation is satisfied
  2) but programmer provides interface (getter setter method) to user to access private members indirectly
- ➔ Through Encapsulation we can assure that our data is protected.
- ➔ In website encapsulation is achieved using login
- ➔ In mobile phone encapsulation is achieved using password (fingerprint)
- ➔ In bank encapsulation is achieved using secret passcode to access money locker.

Code to Demonstrate Encapsulation-

1)

```
// implementation of Encapsulation
#include<iostream>
using namespace std;
class Data{
   private:
   int age=21;
   public:
   void printdata()
   {
      cout<<"Age is 21"<<endl;
   }

};
int main()
{
   Data d;
   d.age=25;
   d.printdata();
}
```

2)

```
// Encapsulation
#include<iostream>
using namespace std;
class data {
private:
        int age;
public:
        void input()
        {
                cout<<"Enter age :- ";
                cin>>age;
        }
        void output()
        {
                cout<<"Age is ->"<<age;
        }
};
int main()
{
        data d;
        d.age=0;
        d.input();
        d.output();
}
```

Q Explain Abstraction?

- ➔ Abstraction is one of the important core concepts of object-oriented programming refers to hiding the implementation details from the user.
- ➔ The user is interested in the functionality and working of software irrespective of what’s happening in the backend and how implementation took place.
- ➔ Abstraction principle says that hide the unwanted details and show the relevant details.
- ➔ In programming when we create class and methods and from main() function we call them but user is not interested where is the implementation of this method present.
- ➔ Abstraction is seen in day-to-day life in ATM machine, online chat
- ➔ As a developer we know that whenever any data is to be transmitted from sender to receiver several protocols are followed but this all is hidden from user and this phenomenon is known as Abstraction
- ➔ In ATM machine user are interested in money and not in how internal structure is present and what is the flow of execution of instructions.
- ➔ Hiding the complexity of code

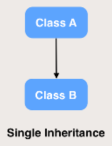
➔ Example –

```
//abstraction
#include<iostream>
using namespace std;
class check
{
    private:
    int dob;
    public:
    void getdata()
    {
        cout<<"entter birth year ";
        cin>>dob;
    }
    string result()
    {
        int temp=2024-dob;
        if(temp>=18)
        return "You are Eligble to vote";
        else
        return "You are not eligible to vote";
    }
};
int main()
{
    check yash;
    yash.getdata();
    cout<<yash.result();
}
```

Q What is Inheritance and types of Inheritance?

- ➔ Inheritance refers to the phenomenon of sharing properties of parent class to child class
- ➔ Inheritance promotes the Code reusability and sharing of methods/data
- ➔ Only public and protected members are inherited in child class
- ➔ Inheritance is of 5 types
    1) Single Inheritance
    2) Multilevel Inheritance
    3) Hybrid inheritance
    4) Hierarchical inheritance
    5) Multiple inheritance

➔ Single Inheritance:

- It is a type of inheritance in which there is single child class which derives the properties from single base class
- Only public and protected properties of child class get inherit
- Parent method can be invoked using child object
- 

- Realtime example:
    1) Bank Account – parent class
       Savings account – child class
    2) Vehicle – parent class
       Car- child class
- Example

```
// Single Inheritance
#include<iostream>
using namespace std;
class data{
    protected:
    int num1,num2;
    public:
    void getdata()
```

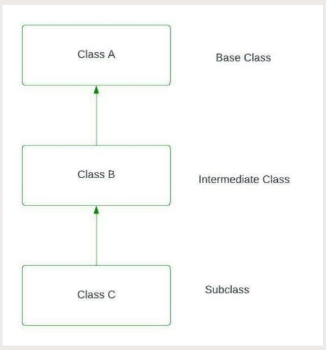
```
        {
            cout<<"Enter 1st number -";
            cin>>num1;
            cout<<"Enter 2nd number -";
            cin>>num2;
        }
    };
    class operations:public data{
        public:
        int add()
        {
            return num2+num1;
        }

    };
    int main()
    {
        operations op;
        op.getdata();
       cout<< op.add();
}
```

➔ Multilevel Inheritance

- It is a type of inheritance in which class B inherits class A and class C inherits class B
- In this inheritance takes place level by level

```
-  class A // base class
-  {
-      ...........
-  };
-  class B : access_specifier A // derived class
-  {
-      ...........
-  } ;
-  class C : access_specifier B // derived from derived class B
-  {
-      ...........
-  } ;
```



- Methods and members of class A can be access by object of class C
- Grandfather -> Father -> Child is the best example of multilevel inheritance
- Example:
    1) Laptop – super class 1
       Tablet – subclass 1
       Mobile phone- Subclass 2
    2)
- Example

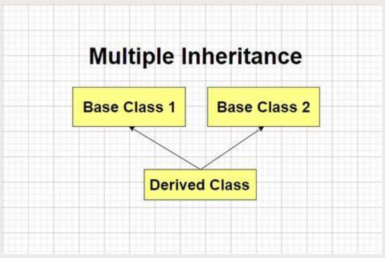
```
//Multi level Inheritance
#include<iostream>
using namespace std;
class data {
protected:
        int num=10;
};
class add:protected data {
protected:
void addvalue()
  {     num=num+10;  }
};
class sub:protected add {
public:
void minus()
{
    addvalue();
        num=num-5;

}
public:
        void show()
        {
                cout<<num;
        }

};
int main()
{
        sub yash;
        yash.minus();
        yash.show();
}
```

➔ **Multiple Inheritance**

- It is a type of inheritance in which we have multiple base classes and single derived class
- The child class inherits the properties of multiple parent class
- Java doesn't support multiple inheritance



- Multiple Inheritance can lead to Ambiquity problem, when the multiple parent class contains the same name functions and child derives it when child object calls the functions the compiler is confused to call function of which parent class??

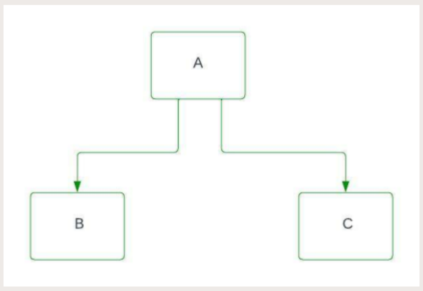
- So to solve this problem of Ambiguity we use Scope resolution Operator(::)
  Derived class object.parent_name_class::function_name
- Example –

```
//multiple Inheritance
#include<iostream>
using namespace std;
class father{
   protected:
   int fd=1000;
};
class mother{
  protected:
  int mf=1000;
};
class son:private father,private mother
{

  public:
  void add()
  {
     cout<<"Son total money are -> "<<fd+mf;
  }
};
int main()
{
   son yash;
   yash.add();
}
```

➔ Hierarchical Inheritance

- It is a type of inheritance in which multiple child class (derived class) derives the properties from single parent class (base class)



- Hierarchical Inheritance forms a tree like structure
- Example:
    1) Car – super class
       Ev car- sub class 1
       Petrol car- sub class 2
       CNG car- sub class
    2) Shape – parent class
       Circle – subclass 1
       Triangle – subclass 2
       Rectangle – subclass 3
- Example –

```
// hierarchicaL Inheritance
#include<iostream>
using namespace std;
class father{
    protected:
    int savings=100000;
};
class son1:protected father{
    public:
    void spend()
    {
        savings=savings-20000;
        cout<<savings<<endl;
    }

};
class son2:protected father{
  public:
  void spend()
  {
      savings=savings-50000;
      cout<<savings<<endl;
  }
};
class son3:protected father
{
    public:
    void save()
    {
        savings=savings+100000;
        cout<<savings<<endl;
    }
};
int main()
{
    son1 s1;
    s1.spend();
    son2 s2;
    s2.spend();
    son3 s3;
    s3.save();
}
```

➔ Hybrid Inheritance

- Hybrid inheritance is a type which is a combination of both multiple and hierarchical inheritance
- Animal, Aquatic -> fish -> shark

Code:

```
//Hybrid Inheritance
#include<iostream>
using namespace std;
class a{
    protected:
    int x=10;
};
class b
{
  protected:
  int y=-10;
};
class c:protected a,protected b
```

```
        {
         protected:
         int z=110;
        };
        class d:protected c
        {
         public:
         void show()
          {
             cout<<x<<y<<endl;
          }
        };
        class e:protected c
        {
         public:
         void display()
          {
             cout<<z<<endl;
          }
        };
        int main()
        {
           d yash;
           e narendra;
           yash.show();
           narendra.display();
}
```

Q what is Class?

- Class is user defined datatype which is the blue print of object
- Class contains several data members and member functions which can be accessed by creating an object of that class
- Size of class= sum of size of all data members
  Size of empty class =1
- Memory is allocated to class only when its Object is created.
- Multiple classes can be created.
- Class provides the benefit of data security and protection
- In class by default all the members of class are private

Q What is Object?

- Object is the realworld instance of class
- Object is the instance of class

Q What is Access specifiers? And how many of them are present?

- In object-oriented programming in c++ programming language it supports three types of access specifiers
    1) Public
    2) Private
    3) Protected
- This access specifiers control the access of members of class outside the region of class

1) Private

- Private access specifiers allow the member functions and data members to be accessed in class only no way it cannot be accessed outside the class
- Private access specifier helps in enabling encapsulation
- Private members don't get inherited

2) Public

- Public access specifier allows the member functions to be accessed in any region of code there is no restrictions in accessing the data members

3) Protected

- Protected access specifier allows accessing the class properties in subclass or superclass (derived class)
- Protected class properties cannot be accessed outside the class region.

Q What is constructor and explain types of constructors?

➔ Constructor is a member function of class which get implicitly called when an object is created.

➔ If programmer don't create/define a constructor explicitly compiler do it internally (Compiler creates default constructor)

➔ Constructor are categorized into 4 types

1) Default Constructor
2) Parametrized constructor
3) Copy constructor

4) Dynamic constructor

- ➔ Constructor can be used to initialize private variables of class
- ➔ Constructor overloading is possible
- ➔ Constructor is defined in public mode
- ➔ Name of constructor is same as name of class
- ➔ There is no return type of constructor
- ➔ In case of Inheritance,
  Parent constructor is called first and then child constructor is called.

■ Default Constructor

1) It is a type of constructor which does not accept any argument (zero argument constructor).
2) Default constructor is automatically created by compiler but if programmer creates it explicitly then there is no error.
3) Default constructor is called during the creation of object.
4) Default constructor overloading is not possible.
5) There is no return type of default constructor
6) Syntax –
```
Class_name()
{

}
```
7) Example-
```
//default constructor
#include<iostream>
using namespace std;
class Data{
    private:
    int age;
    public:
    Data()
    {
        age=21;
    }
    void showdata()
    {
        cout<<age;
    }
};
int main()
```

```
{
   Data d;
   d.showdata();
}
```

8) When creating any object if we don't pass any argument then default constructor is called.

Parametrized Constructor

1) Parametrized constructor is type of constructor which gets invoked when arguments are passed during the creation of objects
2) Overloading of Parametrized constructor is possible
3) Parametrized constructor is not created on its own we the programmer need to define them explicitly
4) Parametrized constructor can be used to initialize private data members
5) Syntax-

```
Class_name(arguments)
{

}
```

6) Example-

```
//parametrized constructor
#include<iostream>
using namespace std;
class Data{
  private:
  int age;
  string name;
  public:
  Data(int a,string b)
  {
     age=a;
     name=b;
  }
  void show()
  {
     cout<<age<<name;
  }
};
int main()
```

```
{
   Data d(21,"Yash");
   d.show();
}
```

■ Copy Constructor

1) Copy constructor is a type which is used to create object from the reference of previous object
2) It is used to create a duplicate copy
3) Example –

```
//copy constructor
#include<iostream>
using namespace std;
class Data{
  private:
  int age;
  public:
  Data(int t)
  {
     age=t;
  }
  Data(const Data&data)
  {
     age=data.age+25;

  }
  void show()
  {
     cout<<age;
  }
};
int main()
{
   Data yash(21);
   Data YASH=yash;
   YASH.show();
}
```

➔ Q What is Destructor?

➔ Destructor is a member function of a class which is opposite to constructor
➔ The way constructor is called during the creation of object similarly destructor is called before the object termination
➔ The name of destructor is same as class_name only extra "~" is added
➔ Destructor overloading is not possible
➔ There are no types of destructors
➔ In case of Inheritance,
Destructor of child class is called first and then destructor of parent class is called.
➔ Example –

```
//destructor
#include<iostream>
using namespace std;
class Data{
  private:
  int age=21;
  public:
  Data()
  {
      cout<<"Constructor is called"<<endl;
  }
  ~Data()
  {
      cout<<"Destructor is called"<<endl;
  }
};
int main()
{
    Data d2;
    {
        Data d1,d2;
    }

}
```

Q What is Polymorphism and does it have any types?

- ➔ The word polymorphism comprises of 2 meaning poly means multiple morphisms refers to existence that means polymorphism refers to existence of any particular entity in more than one form
- ➔ In object-oriented programming they may be functions/methods which may be single entity but exist in more than one form and that's the principle of OOPS
- ➔ In program their may be some module which may perform similar actions only but at different location they have different existance
- ➔ Polymorphism are broadly classified into 2 types
  1) Compile time polymorphism
  2) Run time polymorphism

■ Compile time polymorphism

1) It is also known as Static binding
2) In Compile time polymorphism the type of method to be executed is decided at compile time.
3) Compile time polymorphism is categorized into 2 types
   1) Function overloading
   2) Operator overloading

■ Function overloading

- ➔ The method of creating multiple functions with same name but differs in number of arguments is called as function overloading
- ➔ That means in function overloading we will have multiple functions with same name their return type may or may not be same but they will differ in no of passed arguments
- ➔ main() function cannot be overloaded
- ➔ Example –

```
//function overloading
#include<iostream>
using namespace std;
int add(int a,int b)
{
   return a+b;
}
double add(double a,double b)
{
   return a+b;
}
int main()
```

```
{
   cout<<add(3,4)<<" "<<add(3.00980,4.0);
```

Q How Object Oriented programming is related to real world?

➔ Various principles of object-oriented programming are observed in our day to day life

1) We the human entity are
   Student -> school
   Son -> home
   Player -> cricket match
   Guest -> friend house
   Though we are single unit but our instances chances according to location this principle works similar to "polymorphism".

2) Whenever we are going on a trip we protect our bag by locking it with passcode so that no random person can access your items
   That means you are protecting your data from unauthorized access and that is known as "Encapsulation".

3) When we are using any AI tools like (ChatGpt, Video.ai) we focus on functionality and result of the process the tools don't let know the user the internal working structure of the software that means underlying details are hidden from user thus following the principle of "Abstraction".
4) Laptop and mobile phone contain almost same technologies features that means laptop functionality are shared with the mobile phone and this is the application of "Inheritance".

Q Why to study OOPS?

➔ OOPS principles help in software development process making the development process easier and convenient
➔ Oops supports the code reusability through Inheritance.
➔ Code redabillty, usability increases.
➔ Oops concepts help in developing real time complex projects.

Q When we say that language 'X' is object-oriented programming language what does it mean?

- ➔ It implies that oops concept can be implemented using x language and x language supports oops concepts

Q can you explain the Real-world analogy of class and object?

- ➔ India is class which contain several attributes like citizenship(), right_to_vote(), festivals()
- ➔ We the citizens of India(class) are the object of India where we use the services of country
- ➔ That means any country is class and citizens of that country is class of that country

Q What is this keyword?

- ➔ this keyword is a pointer in OOP which refers to the current object
- ➔ it is used to resolve name conflict among function parameter or local variable.
- ➔ Example

```
//this pointer
#include<iostream>
using namespace std;
class Data{
    int age;
    public:
    void getdata(int age)
    {
        this->age=age;
    }
    void showdata()
    {
        cout<<age;
    }
};
int main()
{
    Data d;
    d.getdata(12);
    d.showdata();
}
```

Q Does overloading works with inheritance?

- ➔ In c++ function overloading in case of inheritance works
- ➔ Example

```
//function overloading in inheritance
#include<iostream>
using namespace std;
class one{
  public:
  int print(int k)
   {
      return k;
   }
};
class two:public one{
    public:
    double print(double p)
    {
       return (double(p));
    }

};
int main()
{
    two obj;
    cout<<obj.print(2)<<endl;
    cout<<obj.print(23.45556);
}
```

Q How can we call the base methods without creating the instance?

- ➔ In order to call the methods from the base class without creating its instance we can use the OOP pillar known as Inheritance simply create the instance of derived class and call the base methods using derived class object.

Q Is it possible for a class to inherit the constructor of base class?

- ➔ Directly inheriting the constructor of base class is not possible but it can be done explicitly using the collan(:) operator in derived class
- ➔ Example

```
//constructor inheritance
#include<iostream>
using namespace std;
class A{
    public:
    A()
    {
        cout<<"A is called\n";
    }
};
class B:public A{
  public:
  B():A()
  {
      cout<<"B is called";
  }
};
int main()
{
    B obj;
}
```

Q What is friend function?

- ➔ Friend function is a non-member function of class which is used to access private members of class
- ➔ Friend function is declared inside the class and defined outside the class
    - ➔ Friend function is declared using the keyword" friend".
- ➔ Example –

```
//use of friend function
#include<iostream>
using namespace std;
class Data{
  private:
  int age=21;
  friend void showdata(Data d);
};
void showdata(Data d)
{
   cout<<"Age is -> "<<d.age;
}
int main()
{
   Data d;
   showdata(d);
}
```

Q What is exception handling?

- ➔ During the execution of programs, they may be some error which could arise due to wrong input of user so in this case it is necessary to manage the exception. C++ language provides the feature to support exception handling by using try, throw, catch block
- ➔ Syntax –

```
try{
condition
throw
}
catch()
{

}
```

- ➔ Catch(…) is used to capture all types of error
- ➔ Example –

```
//exception handling
#include<iostream>
using namespace std;
int main()
{
   int a=12,b=0;
   try{
      if(b==0)
      throw "cannot divide by zero";
      else
      cout<<a/b;
```

```
            }
            catch(const char *e)
            {
                cout<<e;
            }
        }
```

Q Write a C++ program to swap private data member of two different classes using friend function

➔

```
//swapping of two number
#include<iostream>
using namespace std;
class two;
class one{
    int x=10;
    public:
    friend void before(one &a,two &b);
    friend void after(one &a,two &b);
};
class two{
    private:
    int y=20;
    public:
     friend void before(one &a,two &b);
    friend void after(one &a,two &b);
    };
    void before(one &a,two &b)
    {
        cout<<"value before swapping "<<a.x<<" "<<b.y;
    }
    void after(one &a,two &b)
    {
        int temp=a.x;
        a.x=b.y;
        b.y=temp;
        cout<<"\nValue after swapping "<<a.x<<" "<<b.y;
    }
```

```
int main()
{
    one a;
    two b;
    before(a,b);
    after(a,b);

}
```

Further programs are covered for self-practice session link will be provided go through them once and then you are done with oops
Remember to revise the subject thoroughly

https://github.com/YashNemade-svg/OOPS-CODES-PRACTICE

THANK YOU FOR READING TILL LAST

Contact details of Author:

Name: Er Yash Satish Nemade

BTECH-(Information Technology)

Ph no: +91 7249728564

Email: yashnemade16@gmail.com