Object oriented programming (OOP)

In C++, OOP is a Programming paradigm that uses "objects" to design and build applications. C++ supports all major features of object oriented programming:-

(i) Class and object
(ii) Polymorphism
(iii) Inheritance
(iv) Encapsulation
(v) Abstraction

Class:

Class is a blueprint for creating an object. Class is user-defined data type. Class is a combination of data members (variables) and member functions (methods).

Syntax:

```
class class_Name
{
  Access specifies:
  // Data members (variables)
  // member functions (methods)
};
```

Example:-

```
#include <iostream>
using namespace std;
class student
{
  public:
  string name;
  int age;
  void display()
  {
```

```
    cout << "My name is : " << name;
    cout << "\n My age is : " << age;
  }
};
int main()
{
  Student s1;
  s1.name = "Rajesh";
  s1.age = 20;
  s1.display();
  return 0;
}
```

Object:- An object is an instance of a class. By assigning object with operator, you can access the members of class. An object can be create by the same name of the class. you can create multiple objects from the same class.

Syntax:

```
class_name  obj_name;
```

Example program

```
#include <iostream>
using namespace std;
class person
{
  public:
    string name;
    int age;
    void showdata()
    {
      cout << "Hi, I am " << name << " and I am "
      << age << " years old.";
    }
};
int main()
{
  person p1, p2; // create multiple objects.
```

```
p1.name = "Rajesh";
p2.age = 21;

p2.name = "manoj";
p2.age = 19;

p1.showdata();
p2.showdata();
return 0;
}
```

Output:

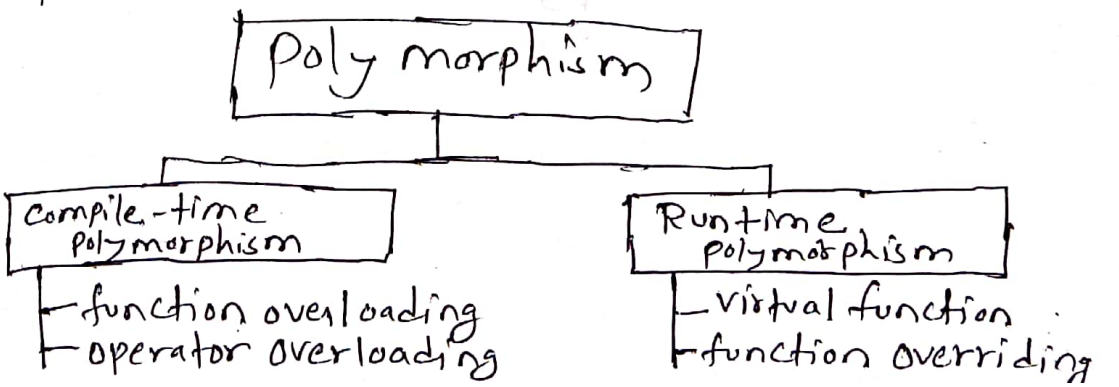
Hi, I am Rajesh and I am 21 years old.
Hi, I am manoj and I am 19 years old.

Polymorphism :-

The word "Polymorphism" is derived from Greek word. poly means many and morphe means form or shape so polymorphism means many forms. polymorphism allows the same function or operator to behave differently in different contexts.

eg one person can act many roles depending situations. like A person can be a father of his son, husband of his wife, an employee of his office.

In C++, there are two types of polymorphism.



Compile-time polymorphism is also known as Early Binding and Runtime polymorphism is known as Late Binding.

Example program of Polymorphism

```
//function overloading program
#include<iostream>
using namespace std;
class CalculateAdd
{
public:
  int add(int a, int b)
  {
    return a+b;
  }
  int add(int a, int b, int c)
  {
    return a+b+c;
  }
};
int main()
{
  CalculateAdd obj;
  cout<< obj.add(5, 10)<<endl;
  cout<< obj.add(5, 10, 20)<<endl;
  return 0;
}
```

output:

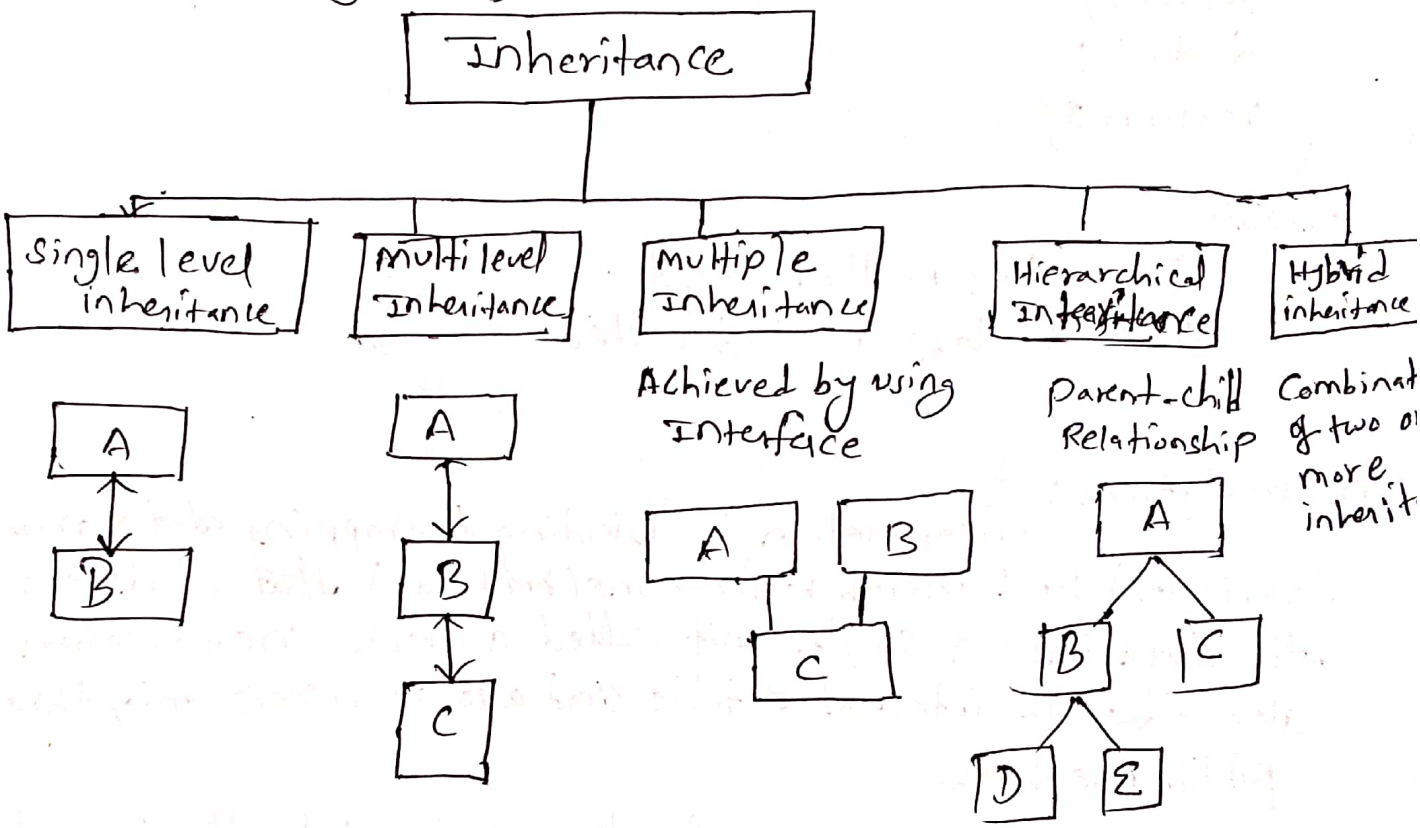
```
15
35
```

Inheritance :-

Inheritance is the process of creating a new class (derived class) by acquiring the properties and behaviors (data members and member functions) of an existing class (base class).

By using inheritace, we can achieve higher level of code reusability.

Different types of Inheritance :-



Program example of Inheritance

```
#include<iostream>
using namespace std;
Class Base
{
  public:
  void b()
  {cout<<"base class method called";
};}
Class derived : public Base
{
```

```
public:
 void d()
 {
  cout<<"\n derived class method called";
};}
 int main()
 {
   derived d;
   d.b();
   d.d();
   return 0;
 }
```

output:

base class method called
derived class method called

Encapsulation :- Encapsulation is binding / wrapping data members (variables) and member functions (methods) that work on the data into a single unit called a class. Encapsulation also hides the internal details and allows access only through public methods.

Encapsulation = Data + functions wrapped together in a class with controlled access.

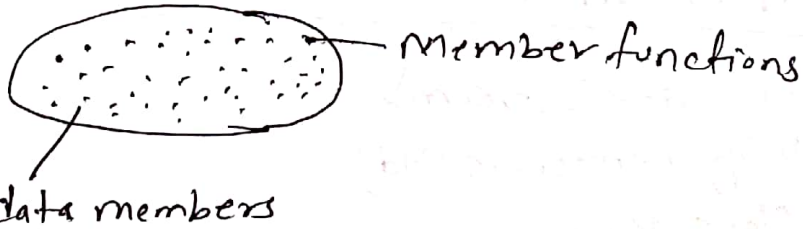


fig: Encapsulation

Class is the best example of encapsulation.

Encapsulation protecting data from unauthorized access.

Example of Encapsulation

```
#include <iostream>
using namespace std;
class student
{
public:
  int getAge()
  {
    return age;
  }
  private:
    int age = 20; // hidden data
};
  int main()
  {
    Student s;
    cout << "Age: " << s.getAge();
    return 0;
  }
```

output:

```
Age : 20
```

Abstractions :- Showing essential features and functionalities but hiding complex details.

eg

```
#include <iostream>
using namespace std;
class calc{
 public:
    void calculator()
    {
      void Add(); // internal working.
      void sub(); // internal working
      cout << "Addition and subtraction";
    }
```

```
private:
    void Add()
    {cout<< "perform Addition "<<endl;
    }
    void sub()
    { cout<<"perform subtraction "<<endl;
    }
};
  int main()
  {
    calc c;
     c. calculator();
    return 0;
}
```

output:

perform Addition
perform subtraction
Addition and subtraction