Inheritance

Definition

- Inheritance is one of four pillars of Object-Oriented Programming (OOPs).
- It is a feature that enables a class to acquire properties and characteristics of another class.
- Inheritance allows you to reuse your code since the derived class or the child class can reuse the members of the base class by inheriting them.
- Consider a real-life example to clearly understand the concept of inheritance.

  A child inherits some properties from his/her parents, such as the ability to speak, walk, eat, and so on. But these properties are not especially inherited in his parents only. His parents inherit these properties from another class called mammals. This mammal class again derives these characteristics from the animal class. Inheritance works in the same manner.
- During inheritance, the data members of the base class get copied in the derived class and can be accessed depending upon the visibility mode used. The order of the accessibility is always in a decreasing order i.e., from public to protected.

Why and When to Use Inheritance?

Inheritance makes the programming more efficient and is used because of the benefits it provides -

- **Code reusability**: One of the main reasons to use inheritance is that you can reuse the code. For example, consider a group of animals as separate classes - Tiger, Lion, and Panther. For these classes, you can create member functions like the predator() as they all are predators, canine() as they all have canine teeth to hunt, and claws() as all the three animals have big and sharp claws. Now, since all the three functions are the same for these classes, making separate functions for all of them will cause data redundancy and can increase the chances of error. So instead of this, you can use inheritance here. You can create a base class named carnivores and add these functions to it and inherit these functions to the tiger, lion, and panther classes.
- **Transitive nature**: Inheritance is also used because of its transitive nature. For example, you have a derived class mammal that inherits its properties from the base class animal. Now, because of the transitive nature of the inheritance, all the child classes of ‘mammal’ will inherit the properties of the class ‘animal’ as well. This helps in debugging to a great extent. You can remove the bugs from your base class and all the inherited classes will automatically get debugged.

What Are Child and Parent classes?

To clearly understand the concept of Inheritance, you must learn about two terms on which the whole concept of inheritance is based - Child class and Parent class.

- Child class: The class that inherits the characteristics of another class is known as the child class or derived class. The number of child classes that can be inherited from a single parent class is based upon the type of inheritance. A child class will access the data members of the parent class according to the visibility mode specified during the declaration of the child class.
- Parent class: The class from which the child class inherits its properties is called the parent class or base class. A single parent class can derive multiple child classes (Hierarchical Inheritance) or multiple parent classes can inherit a single base class (Multiple Inheritance). This depends on the different types of inheritance in C++.

Syntax

The syntax for defining the child class and parent class in all types of Inheritance in C++ is given below:

```
class parent_class
{
   //class definition of the parent class
};
class child_class : visibility_mode parent_class
{
  //class definition of the child class
};
```

Visibility Mode

The visibility mode specifies how the features of the base class will be inherited by the derived class. There are three types of visibility modes for all types of Inheritance in C++:

Public Visibility Mode:

In the public visibility mode, it retains the accessibility of all the members of the base class. The members specified as public, protected, and private in the base class remain public, protected, and private respectively in the derived class as well. So, the public members are accessible by the derived class and all other classes. The protected members are accessible only inside the derived class and its members. However, the private members are not accessible to the derived class.

The following code snippet illustrates how to apply the public visibility mode in a derived class:

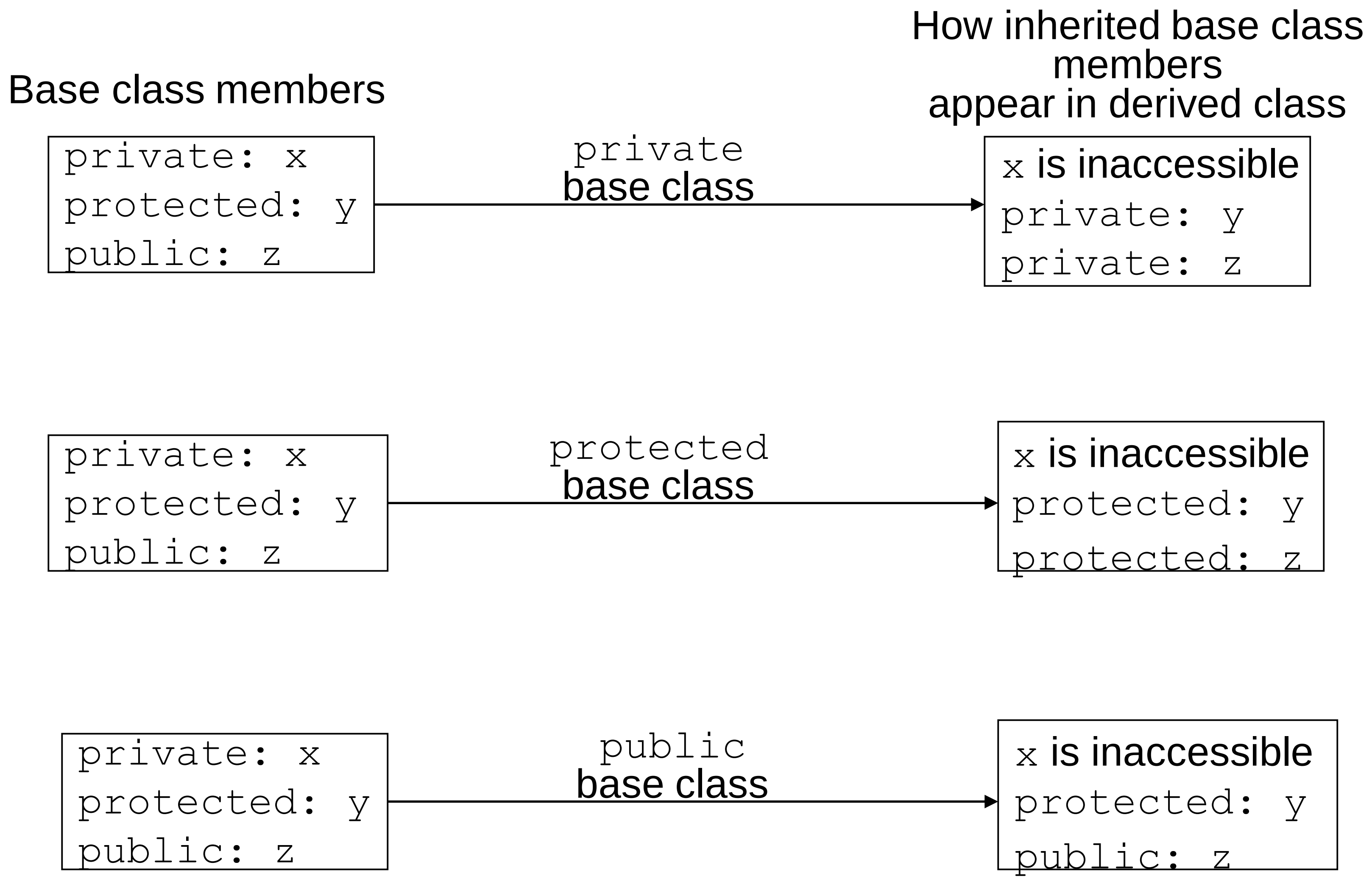
```
class base_class_1
{
   // class definition
};
class derived_class: public base_class_1
{
   // class definition
};
```

Example

The following code displays the working of public visibility mode with all three access specifiers of the base class:

```
class base_class
{
private:
    //class member
    int base_private;
protected:
    //class member
    int base_protected;
public:
    //class member
    int base_public;
};
class derived_class : public base_class
{
private:
    int derived_private;
    // int base_private;
protected:
    int derived_protected;
    // int base_protected;
public:
    int derived_public;
    // int base_public;
};
int main()
{
    // Accessing members of base_class using object of the //derived_class:
    derived_class obj;
    obj.base_private;   // Not accessible
    obj.base_protected; // Not accessible
    obj.base_public;    // Accessible}
```

Inheritance vs. Access



Inheritance vs. Access

class Grade

private members:
```
  char letter;
  float score;
  void calcGrade();
```
public members:
```
  void setScore(float);
  float getScore();
  char getLetter();
```

class Test : public Grade

private members:
```
  int numQuestions;
  float pointsEach;
  int numMissed;
```
public members:
```
  Test(int, int);
```

When `Test` class inherits from `Grade` class using `public` class access, it looks like this:

private members:
```
  int numQuestions:
  float pointsEach;
  int numMissed;
```
public members:
```
  Test(int, int);
  void setScore(float);
  float getScore();
  char getLetter();
```

Inheritance vs. Access

class Grade

private members:
```
  char letter;
  float score;
  void calcGrade();
```
public members:
```
  void setScore(float);
  float getScore();
  char getLetter();
```

class Test : protected Grade

private members:
```
  int numQuestions;
  float pointsEach;
  int numMissed;
```
public members:
```
  Test(int, int);
```

When `Test` class inherits from `Grade` class using `protected` class access, it looks like this:

private members:
```
  int numQuestions:
  float pointsEach;
  int numMissed;
```
public members:
```
  Test(int, int);
```
protected members:
```
  void setScore(float);
  float getScore();
  float getLetter();
```

Inheritance vs. Access

class Grade

private members:
```
  char letter;
  float score;
  void calcGrade();
```
public members:
```
  void setScore(float);
  float getScore();
  char getLetter();
```

class Test : private Grade

private members:
```
  int numQuestions;
  float pointsEach;
  int numMissed;
```
public members:
```
  Test(int, int);
```

When `Test` class inherits from `Grade` class using `private` class access, it looks like this:

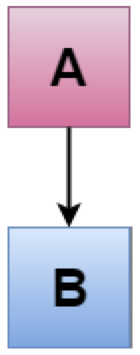
private members:
```
  int numQuestions:
  float pointsEach;
  int numMissed;
  void setScore(float);
  float getScore();
  float getLetter();
```
public members:
```
  Test(int, int);
```

Types of Inheritance

- There are mainly five types of Inheritance in C++
- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

Single Level Inheritance

- **Single inheritance** is defined as the inheritance in which a derived class inherits only one base class.



Where 'A' is the base class, and 'B' is the derived class.

Example 1

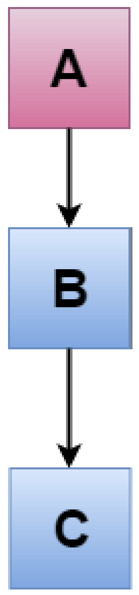
```
#include <iostream>
using namespace std;
 class Account {
   public:
   float salary = 60000;
 };
   class Programmer: public Account {
   public:
   float bonus = 5000;
   };
int main(void) {
     Programmer p1;
     cout<<"Salary: "<<p1.salary<<endl;
     cout<<"Bonus: "<<p1.bonus<<endl;
    return 0;
}
```

Example2

```
class A  {
    int a = 4;
    int b = 5;
    public:
    int mul() {
        int c = a*b;
        return c;  }
};
  class B : private A  {
    public:
    void display()  {
        int result = mul();
        std::cout <<"Multiplication of a and b is : "<<result<< std::endl;  }
};
int main()  {
   B b;
   b.display();
    return 0;
}
```

Multilevel Inheritance

When one class inherits another class which is further inherited by another class, it is known as multi level inheritance in C++. Multi level Inheritance is transitive so the last derived class acquires all the members of all its base classes.

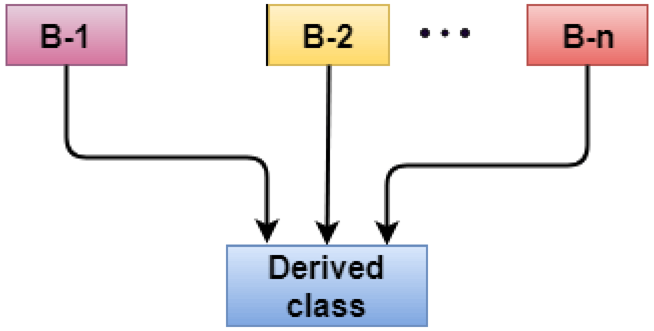


Example

```
 class Animal {
  public:
 void eat() {
   cout<<"Eating..."<<endl;   }
  };
  class Dog: public Animal
  {
     public:
    void bark(){
   cout<<"Barking..."<<endl;    }
  };
  class BabyDog: public Dog
  {
     public:
    void weep() {
   cout<<"Weeping...";    }
  };
int main(void) {
   BabyDog d1;
   d1.eat();
   d1.bark();
    d1.weep();   return 0;  }
```

Multiple Inheritance

Multiple inheritance is the process of deriving a new class that inherits the attributes from two or more base classes.

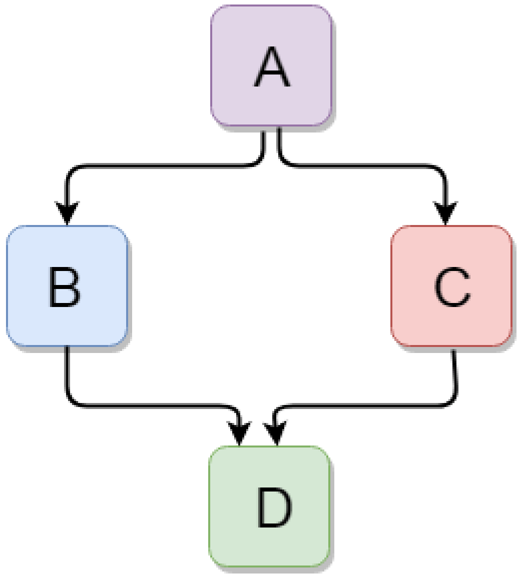


Example

```
#include <iostream>
 class A  {
    protected:
     int a;
    public:
    void get_a(int n)  {
        a = n;  }
};
  class B  {
    protected:
    int b;
    public:
    void get_b(int n)  {
        b = n;  }
};
class C : public A, public B  {
   public:
    void display()  {
          std::cout << "The value of a is : " <<a<< std::endl;
          std::cout << "The value of b is : " <<b<< std::endl;
          cout<<"Addition of a and b is : "<<a+b;  }
};
int main()  {
   C c;
   c.get_a(10);
   c.get_b(20);
   c.display();
 return 0;
}
```

Hybrid Inheritance

Hybrid inheritance is a combination of more than one type of inheritance.



Example

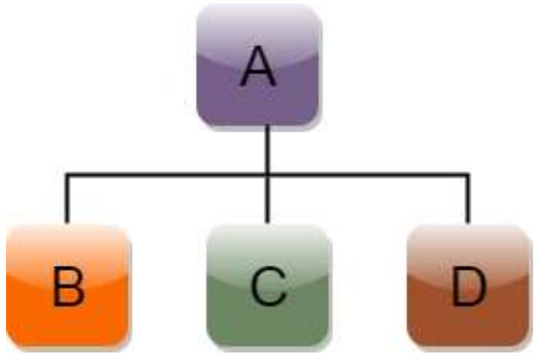
```
class A  {
   protected:
   int a;
   public:
   void get_a()  {
      std::cout << "Enter the value of 'a' : " << std::endl;
      cin>>a;  }
};
  class B : public A   {
   protected:
   int b;
   public:
   void get_b()  {
       std::cout << "Enter the value of 'b' : " << std::endl;
      cin>>b;   }
};
class C
{
   protected:
   int c;
   public:
void get_c()  {
       std::cout << "Enter the value of c is : " << std::endl;
       cin>>c;   }
};
```

Continue..

```
class D : public B, public C
{
   protected:
   int d;
   public:
   void mul()
   {
       get_a();
       get_b();
       get_c();
       std::cout << "Multiplication of a,b,c is : " <<a*b*c<< std::endl;
   }
};
int main()
{
   D d;
   d.mul();
   return 0;
}
```

Hierarchical Inheritance

Hierarchical inheritance is defined as the process of deriving more than one class from a base class.



Example

```
class Shape  {
   public:
   int a;
   int b;
   void get_data(int  n, int m)  {
      a= n;
      b = m;
   }
};
class Rectangle : public Shape    {
   public:
   int rect_area()  {
      int result = a*b;
      return result;
   }
};
class Triangle  : public Shape
{
   public:
   int triangle_area()
   {
      float result = 0.5*a*b;
      return result;   }
};
```

Continue

```
int main()
{
   Rectangle r;
   Triangle t;
   int length, breadth, base, height;
   cout << "Enter the length and breadth of a rectangle: " << endl;
   cin>>length>>breadth;
   r.get_data(length, breadth);
   int m = r.rect_area();
   cout << "Area of the rectangle is : " <<m<< endl;
   cout << "Enter the base and height of the triangle: " << endl;
   cin>>base>>height;
   t.get_data(base, height);
   float n = t.triangle_area();
   cout <<"Area of the triangle is : "  << n<<endl;
   return 0;
}
```