Multiple Inheritance and Diamond Problem

Multiple Inheritance

is a feature of C++ where a class can inherit from more than one class.

The constructors of inherited classes are called in the same order in which they are inherited. For example, in the following program, B's constructor is called before A's constructor.

```
#include <bits/stdc++.h>
using namespace std;

class A
{
    public:
      A() { cout << "A's constructor called" << endl; }
};

class B
{
    public:
      B() { cout << "B's constructor called" << endl; }
};

class C: public B, public A  // Note the order
{
    public:
      C() { cout << "C's constructor called" << endl; }
};

int main()
{
    C c;
    return 0;
}
```

Output:

```
B's constructor called
A's constructor called
C's constructor called
```

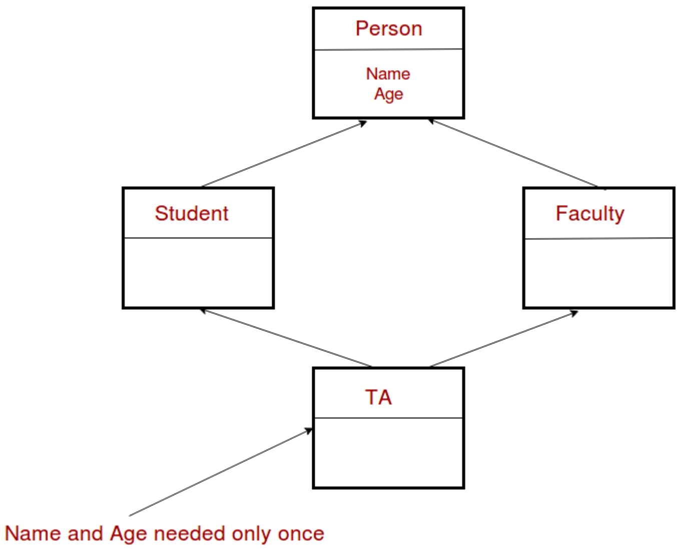
The Diamond Problem

The diamond problem occurs when two superclasses of a class have a common base class. As an example, in the following diagram, the TA class gets two copies of all attributes of the Person class that leads to an ambiguity.

```
#include <bits/stdc++.h>
using namespace std;

class Person
{
    // Data members of person
    public:
        Person(int x) {
            cout << "Person::Person(int ) called" << endl;
        }
};

class Faculty : public Person
{
    // data members of Faculty
    public:
        Faculty(int x) : Person(x) {
            cout << "Faculty::Faculty(int) called" << endl;
        }
};

class Student : public Person
{
    // data members of Student
    public:
        Student(int x) : Person(x) {
            cout << "Student::Student(int) called" << endl;
        }
};

class TA : public Faculty, public Student
{
    public:
        TA(int x): Student(x), Faculty(x) {
            cout << "TA::TA(int) called" << endl;
        }
};

int main()  {
    TA ta(30);
    return 0;
}
```

Output:

```
Person::Person(int ) called
Faculty::Faculty(int) called
Person::Person(int ) called
Student::Student(int) called
TA::TA(int) called
```



In the above program, the constructor of the *Person* is called two times. The destructor of *Person* will also be called two times when the object *ta* is destructed. So object *ta* has two copies of all members of *Person* , this causes ambiguity. The solution to this problem is using **virtual classes**. We make the classes *Faculty* and *Student* as virtual base classes to avoid two copies of *Person* in the *TA* class. As an example, consider the following program:

```
#include <bits/stdc++.h>
using namespace std;

class Person {
    public:
        Person(int x) { cout << "Person::Person(int) called" 
        Person() { cout << "Person::Person() called" << endl;
};

class Faculty : virtual public Person {
    public:
        Faculty(int x) : Person(x)   {
            cout << "Faculty::Faculty(int) called" << endl;
        }
};

class Student : virtual public Person {
    public:
        Student(int x) : Person(x) {
            cout << "Student::Student(int) called" << endl;
        }
};

class TA : public Faculty, public Student  {
    public:
        TA(int x) : Student(x), Faculty(x) {
            cout << "TA::TA(int) called" << endl;
        }
};

int main()  {
    TA ta(30);
}
```

Output:

```
Person::Person() called
Faculty::Faculty(int) called
Student::Student(int) called
TA::TA(int) called
```

In the above program, the constructor of the *Person* is called once. One important thing to note in the above output is the default constructor of

Person is called. When we use the virtual keyword, the default constructor of grandparent class is called by default even if the parent classes explicitly call the parameterized constructor

How to call the parameterized constructor of the *Person* class? The constructor has to be called in *TA* class:

```
#include <bits/stdc++.h>
using namespace std;

class Person
{
    public:
        Person(int x) { cout << "Person::Person(int) called"
        Person() { cout << "Person::Person() called" << endl;
};

class Faculty : virtual public Person
{
    public:
        Faculty(int x) : Person(x) {
            cout << "Faculty::Faculty(int) called" << endl;
        }
};

class Student : virtual public Person
{
    public:
        Student(int x) : Person(x) {
            cout << "Student::Student(int) called" << endl;
        }
};

class TA : public Faculty, public Student  {
    public:
        TA(int x) : Student(x), Faculty(x), Person(x) {
            cout << "TA::TA(int) called" << endl;
        }
};

int main()  {
    TA ta(30);
    return 0;
}
```

Output:

```
 Person::Person(int) called
Faculty::Faculty(int) called
Student::Student(int) called
TA::TA(int) called
```

In general, it is not allowed to call the grandparent's constructor directly, it has to be called through a parent class. It is allowed only when the virtual keyword is used.