虚拟继承和虚基类

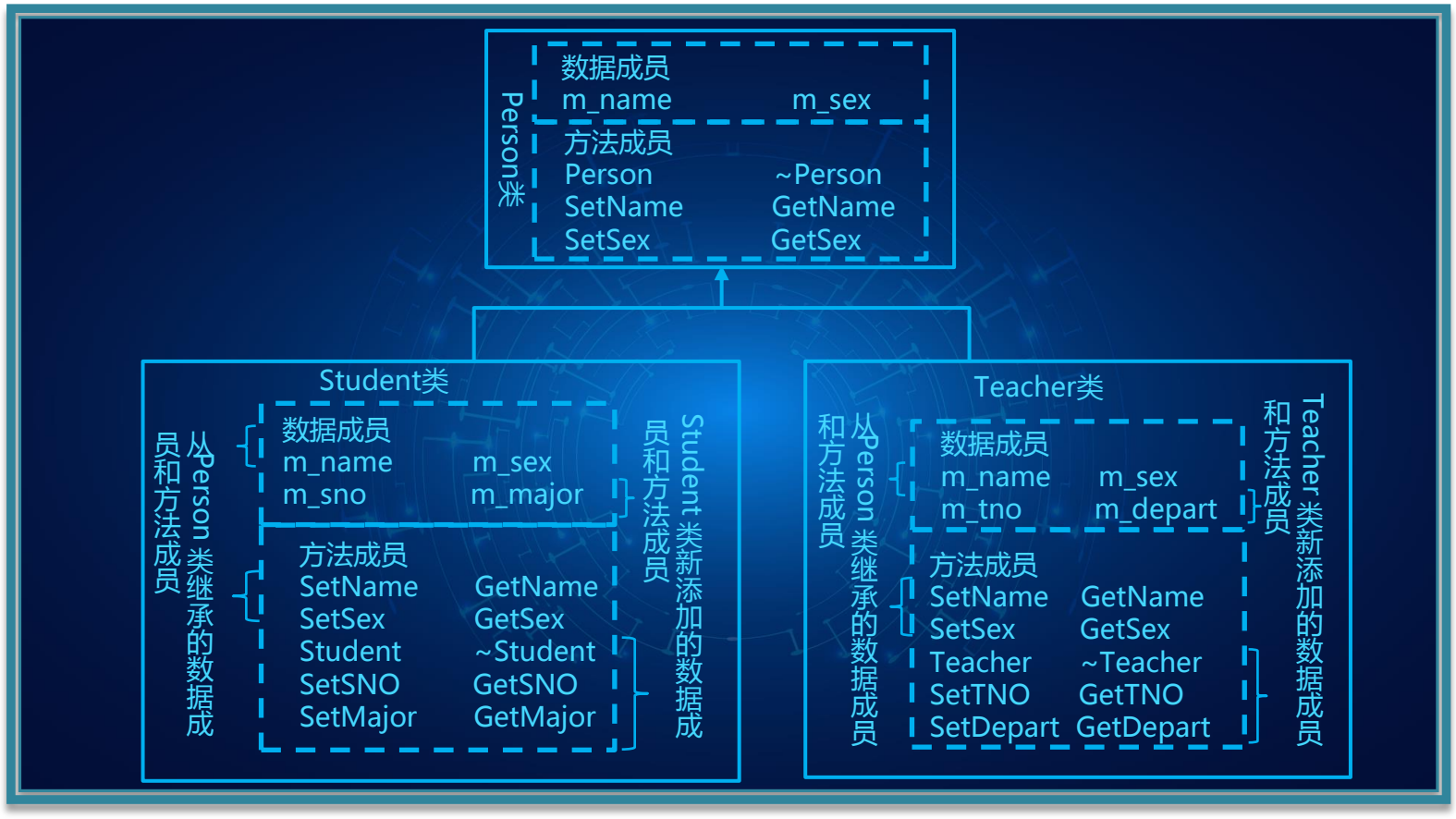
- 在定义派生类时，可以通过虚拟继承方式将基类声明为虚基类。虚基类中的成员在类的继承关系中只会被继承一次，从而解决上述二义性问题。虚拟继承的语法为：

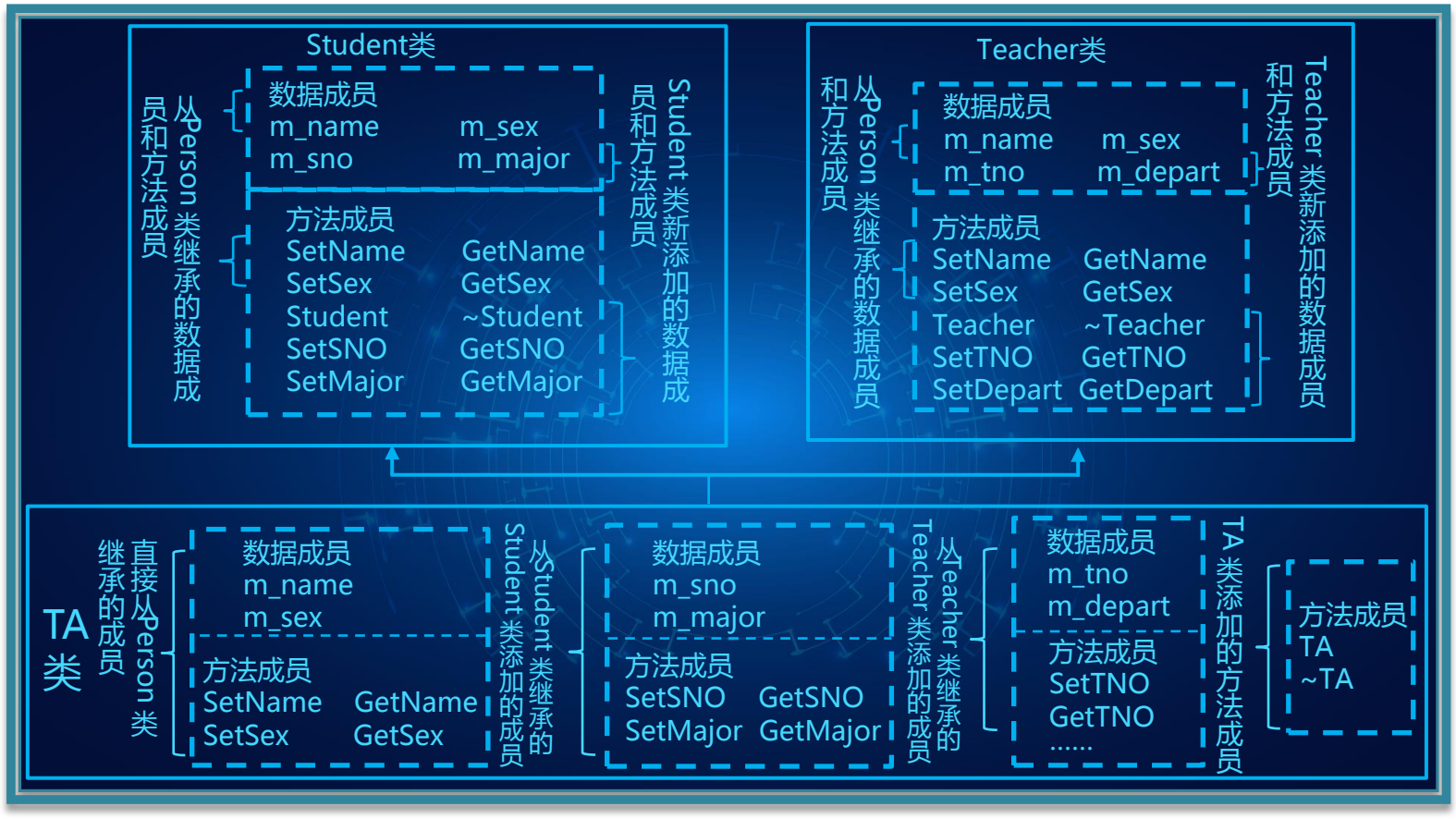
```
class 派生类名 : virtual 继承方式 虚基类名
{
    ……
}
```

其中，关键字virtual和继承方式的顺序可以调换。

例如，在例3-3中定义Student类和Teacher类时，如果采用虚拟继承方式：

```
class Student : virtual public Person
{    ……    }
class Teacher : virtual public Person
{    ……    }
```


Person类
数据成员
m_name
m_sex
方法成员
Person
~Person
SetName
GetName
SetSex
GetSex
Student类
从Person类继承的数据成员和方法成员
数据成员
m_name
m_sex
m_sno
m_major
Student类新添加的数据成员和方法成员
方法成员
SetName
GetName
SetSex
GetSex
Student
~Student
SetSNO
GetSNO
SetMajor
GetMajor
Teacher类
从Person类继承的数据成员和方法成员
数据成员
m_name
m_sex
m_tno
m_depart
Teacher类新添加的数据成员和方法成员
方法成员
SetName
GetName
SetSex
GetSex
Teacher
~Teacher
SetTNO
GetTNO
SetDepart
GetDepart


Student类
从Person类继承的数据成员和方法成员
数据成员
m_name
m_sex
m_sno
m_major
方法成员
SetName
GetName
SetSex
GetSex
Student
~Student
SetSNO
GetSNO
SetMajor
GetMajor
Student类新添加的数据成员和方法成员
Teacher类
从Person类继承的数据成员和方法成员
数据成员
m_name
m_sex
m_tno
m_depart
方法成员
SetName
GetName
SetSex
GetSex
Teacher
~Teacher
SetTNO
GetTNO
SetDepart
GetDepart
Teacher类新添加的数据成员和方法成员
TA类
直接从Person类继承的成员
数据成员
m_name
m_sex
方法成员
SetName
GetName
SetSex
GetSex
从Student类继承的Student类添加的成员
数据成员
m_sno
m_major
方法成员
SetSNO
GetSNO
SetMajor
GetMajor
从Teacher类继承的Teacher类添加的成员
数据成员
m_tno
m_depart
方法成员
SetTNO
GetTNO
.....
TA类添加的方法成员
方法成员
TA
~TA

- 则Person类成为虚基类。此时，TA类从虚基类Person类中继承Person类的成员，从Student类和Teacher类中则只继承了Student类和Teacher类新定义的成员，从而保证TA类中只包含一份Person类的成员、解决了二义性问题。
- 由于虚基类后继类层次中的类都是直接从虚基类继承其成员，而对这部分成员的初始化需要调用虚基类的构造函数来完成。因此，如果一个派生类同时从基类和虚基类继承了成员，那么在定义该类的构造函数时，除了要调用基类的构造函数、还要调用虚基类的构造函数；析构函数也是如此，当销毁该派生类的对象时，除了会直接调用基类的析构函数、还会直接调用虚基类的析构函数。

【例3-4】采用虚拟继承方式改写例3-3，解决二义性问题。

```
// Person.h
#include <iostream>
using namespace std;
#ifndef _PERSON_H
#define _PERSON_H
class Person
{
public:
    Person(char *name, bool sex)
    {
        strcpy(m_name, name);
        m_sex = sex;
        cout<<"Person类构造函数被调用！"<<endl;
    }
    ~Person() { cout<<"Person类析构函数被调用！"<<endl; }
    void SetName(char *name) { strcpy(m_name, name); }
    char* GetName() { return m_name; }
    void SetSex(bool sex) { m_sex = sex; }
    bool GetSex() { return m_sex; }

private:
    char m_name[20];
        // 姓名
    bool m_sex;
        // 性别（true：男，false：女）
};
#endif
```

```
// Student.h
#include "Person.h"
class Student : virtual public Person
{
public:
    Student(char *sno, char *name, bool sex, char *major) : Person(name, sex)
    {
        strcpy(m_sno, sno);
        strcpy(m_major, major);
        cout<<"Student类构造函数被调用！"<<endl;
    }
    ~Student() { cout<<"Student类析构函数被调用！"<<endl; }
    void SetSNO(char *sno) { strcpy(m_sno, sno); }
    char* GetSNO() { return m_sno; }
    void SetMajor(char *major) { strcpy(m_major, major); }
    char* GetMajor() { return m_major; }
private:
    char m_sno[8];              // 学号
    char m_major[20];           // 专业
};
```

```
// Teacher.h
#include "Person.h"
class Teacher : virtual public Person
{
public:
    Teacher(char *tno, char *name, bool sex, char *depart) : Person(name, sex)
    {
        strcpy(m_tno, tno);
        strcpy(m_depart, depart);
        cout<<"Teacher类构造函数被调用！"<<endl;
    }
    ~Teacher() { cout<<"Teacher类析构函数被调用！"<<endl; }
    void SetTNO(char *tno) { strcpy(m_tno, tno); }
    char* GetTNO() { return m_tno; }
    void SetDepart(char *depart) { strcpy(m_depart, depart); }
    char* GetDepart() { return m_depart; }
private:
    char m_tno[6];              // 教师号
    char m_depart[20];          // 系
};
```

```
// TA.h
#include "Student.h"
#include "Teacher.h"
class TA : public Student, public Teacher
{
public:
    TA(char *sno, char *name, bool sex, char *major, char *tno, char *depart)
        : Teacher(tno, name, sex, depart), Student(sno, name, sex, major),
        Person(name, sex)    // 需要调用虚基类的构造函数
    {       cout<<"TA类构造函数被调用！"<<endl;    }
    ~TA() { cout<<"TA类析构函数被调用！"<<endl; }
};
```

```
// testTA.cpp
#include "TA.h"
int main()
{
    TA ta("1210102", "王五", true, "计算机应用", "09110", "计算机科学与技术系");
    cout<<ta.GetName()<<endl;          // 输出姓名
    return 0;
}
```

提示：

当创建TA类对象ta时，会先调用虚基类的构造函数，再按继承顺序调用基类的构造函数，最后调用TA类的构造函数，因此，程序会在屏幕上输出：

Person类构造函数被调用！

Student类构造函数被调用！

Teacher类构造函数被调用！

TA类构造函数被调用！

析构函数的调用顺序与构造函数相反，因此，在程序结束、销毁ta对象前会在屏幕上输出：

TA类析构函数被调用！

Teacher类析构函数被调用！

Student类析构函数被调用！

Person类析构函数被调用！