CS202: Programming Systems

Week 4 – Inheritance

10/2016

CS202 – What will be discussed?

- Introduction to inheritance
- Types of inheritance
- Derived class
  - Constructor
  - Destructor
  - Copy constructor & assignment operator
- Virtual functions and dynamic binding

Introduction

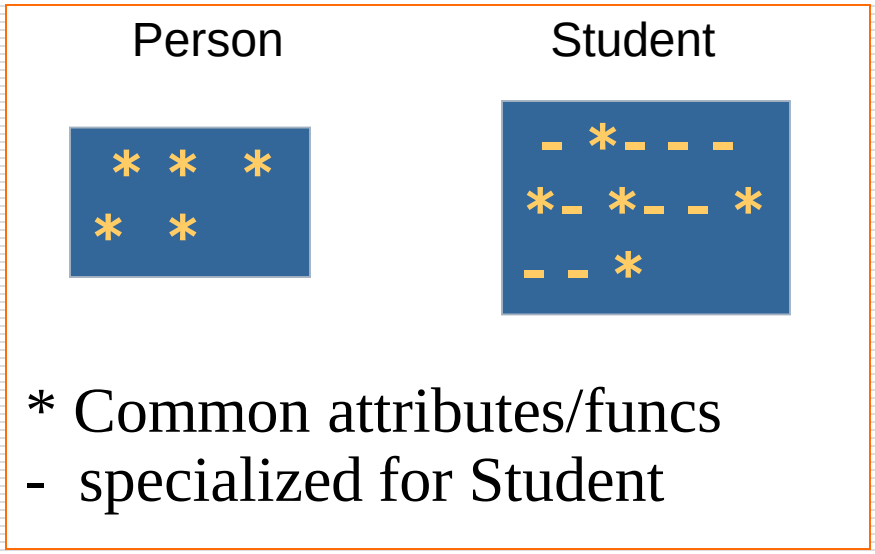
- A new concept does not come alone. When we introduce a class of **`car`** or **`employee`** for example. It may lead us to describe:
  - wheel, engine, driver etc.
  - Or: manager, director…
- To model them, we can use **`class`**. However, how can we model the relationship between them?

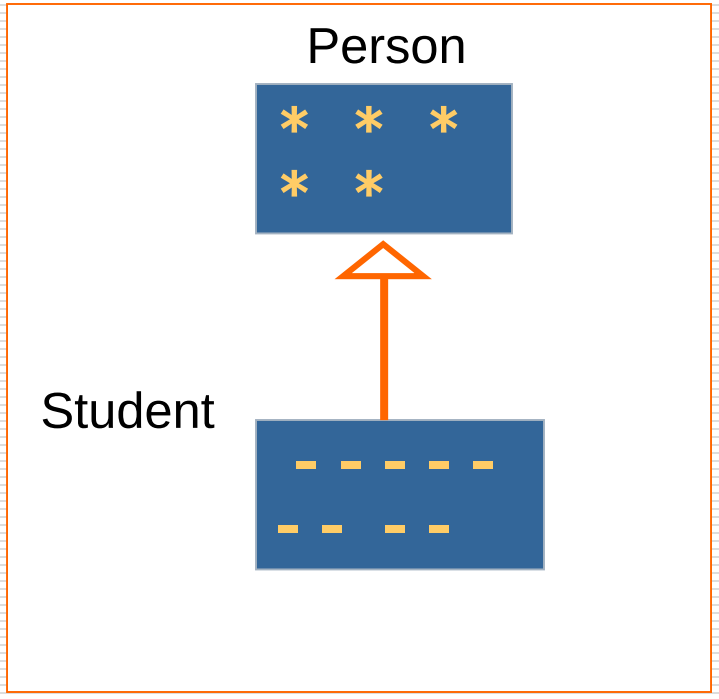
Introduction (cont.)

- In addition, re-usability of existing classes is one of the features of OOP.
- All of the features including attributes and behaviors of a class are also the features of another class.

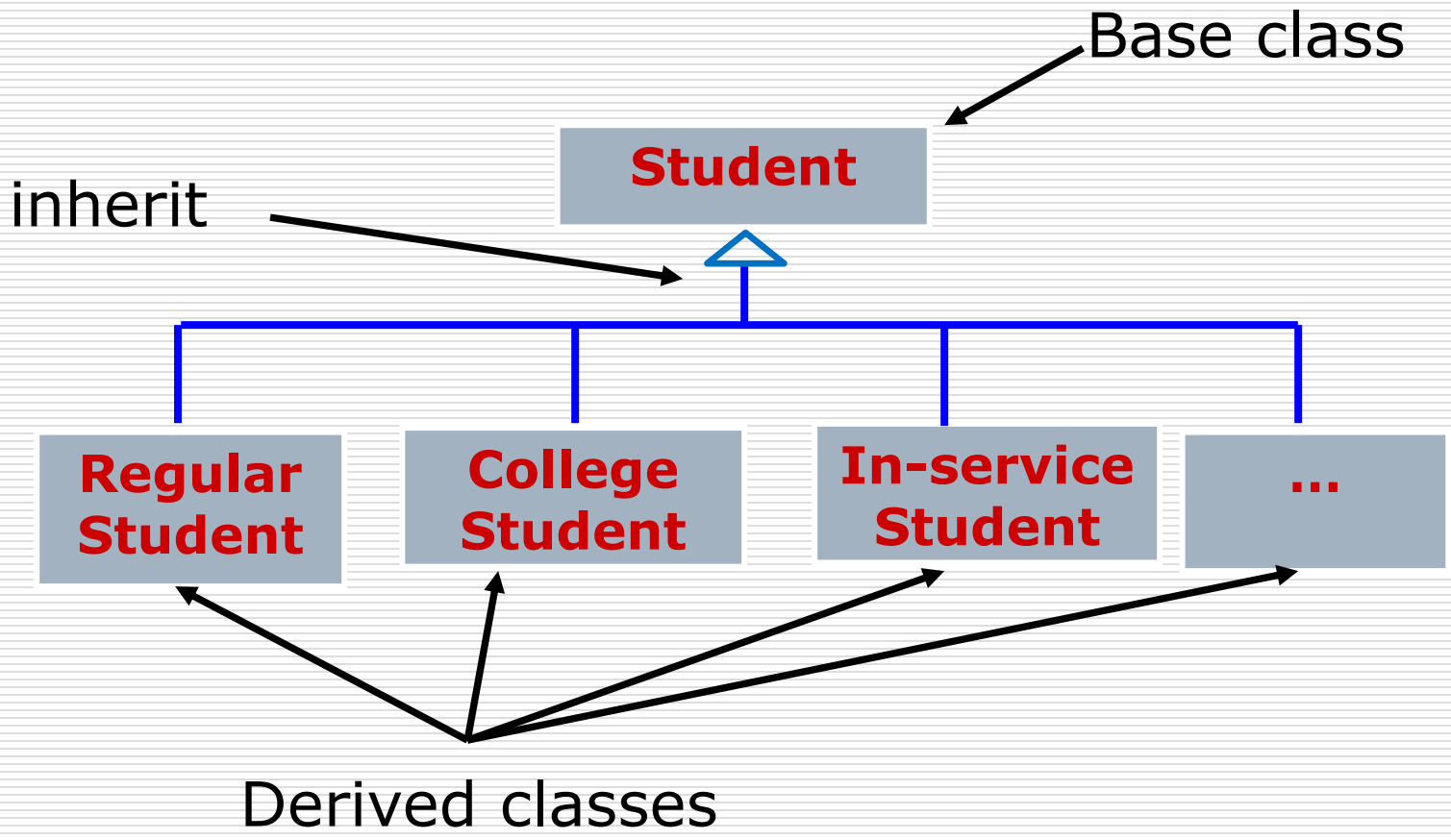
Inheritance

- Student “is a” person





An example


Base class
Student
inherit
Regular Student
College Student
In-service Student
...
Derived classes

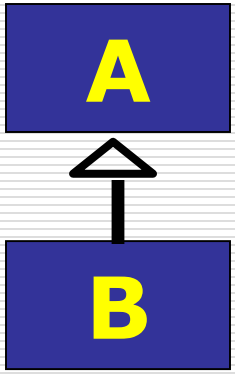
Syntax

- In C++, the inheritance is described as:

```
class <Derived Class> : <TypeOfInheritance> <Base class>
{
      public:
            <public attributes/functions>

      private:
            <private attribute/functions>
};
```

Inheritance: notes



- Attributes/functions in **`public`** of A will become attributes/functions in B
- **`Private`** of A will be part of B but it is only accessible via **`public`** or **`protected`** of A

protected keyword

- **protected** of A is accessible from the derived class B but not from outside

```
class A
{
   public:
      ...
   protected:
      int t;
};
```

```
class B: public A
{
   public:
      void test()
      {
         cout << t;
      }
      ...
};
```

Types of inheritance in C++

There are 3 types:

- **`public`** inheritance
- **`protected`**
- **`private`**

Notes: from now on, if there is no mention of what type of inheritance, it means public inheritance

Types of inheritance

- **`public`**: public and protected of the base class become public and protected of the derived class.
- **`protected`**: public and protected of the base class become protected of the derived class.
- **`private`**: public and protected of the base class become private of the derived class.

Inheritance: member functions

- Member functions of the base class are inherited in the derived class.
- What happens to:
  - Constructors
  - Destructors
  - Assignment operators
- Notes: private members or functions of the base class are inherited but ***only accessible via other public/protected functions*** of A

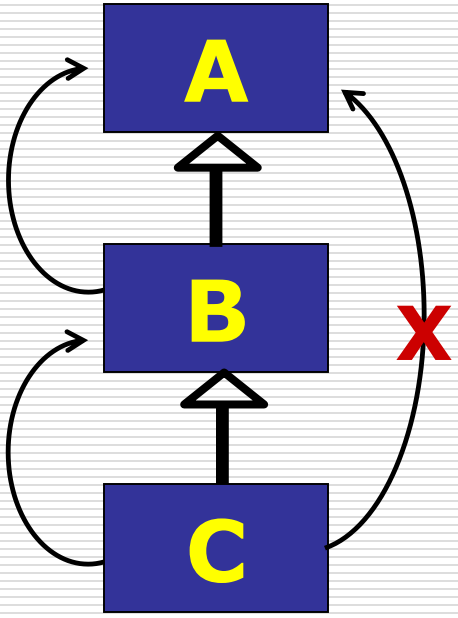
Constructors in inheritance

- When a new object of a derived class is created
  - The constructor of the base class is invoked first.
  - Then, the constructor of the derived class is invoked.
  - In the constructor of the derived class, we can specify which constructor of the base class is called. Otherwise, the default constructor of the base class will be invoked.

Constructors in inheritance

Notes:

- The constructor of the derived class is able to specify the constructor of the immediate base class to be called.



An example

```
class A
{
public:
      A();
      A(int);
};

class B : public A
{
public:
      B(int);
};
```

```
class A {
public:
      A();
      A(int);
};

class B : public A
{
public:
      B(int t) : A(t) {
            ...
      }
};
```

Destructor in inheritance

- When an object of the derived class finishes its lifespan:
  - The destructor of the derived class is invoked first.
  - Then, the destructor of the base class is called later.

“Re-define” member functions

- Sometimes, we need to “re-define” the member functions of the base class in the derived class.
  - It can be done by re-defining the functions inside the derived class

Notes: this re-definition will hide other overloading member functions of this function from the base class.

An example

```
class A {
public:
    void test();
    void test(int);
    void test(int, int);
    ...
};

class B : public A
{
public:
    void test(int);
};
```

```
int main()
{
    B b;
    int x, y;
    ...
    b.test(x); // OK
    b.test(x, y); //error
    ...
}
```

using keyword

```
class A {
public:
    void test();
    void test(int);
    void test(int, int);
    ...
};

class B : public A
{
public:
    using A::test;
    void test(int);
};
```

```
int main()
{
    B b;
    int x, y;
    ...
    b.test(x); // OK
    b.test(x, y); //OK
    ...
}
```

Assignment operator

- It is not inherited from the base class
- To implement the assignment operator for the derived class:
  - Calling the assignment operator of the base class to assign data members of the base class part in the two objects first.
  - Then, implement the assignment for data member of the derived class part.

An example

```
B& B::operator=(const B& src)
{
   if (this == &src)
      return *this;
   A::operator=(src); //call the BASE
   delete [] ptr;
   iSize = src.iSize;
   ptr = new int [iSize];
   for (int i=0; i<iSize; ++i)
      ptr[i] = src.ptr[i];
   return *this;
}
```