Object Oriented Programming CS250

Dr Ahmed Rafat Abas

Computer Science Dept, Faculty of Computers & Informatics, Zagazig University

arabas@zu.edu.eg

http://www.arsaliem.faculty.zu.edu.eg

Example: Show the effect of the use of virtual functions in a base class when other classes are derived from it.

(Q1 Chap. 6)
Consider the class definitions:

```
class BaseClass
{
protected:
        int Num1;
public:
        BaseClass () : Num1(0) {}
        void TestFunc() { Num1 += 10;}
};
```

```
class DerClass : public BaseClass
{
public:
    void TestFunc() { Num1 += 20; }
};
```

After the code:

```
BaseClass Object1;
Object1.TestFunc ();
```

What is the value of Object1.Num1?

10 function call answered by parent class.

After the code:

```
DerClass Object2;
Object2.TestFunc ();
```

What is the value of Object2.Num1?

20 function call answered by child class.

After the code:

```
BaseClass *Object3 = new BaseClass;
Object3->TestFunc ();
```

What is the value of Object3.Num1?

10 function call answered by parent class.

After the code:

```
BaseClass *Object4 = new DerClass;
Object4->TestFunc ();
```

What is the value of Object4.Num1?

10 function call answered by parent class.

The declaration of TestFunc () in BaseClass is now replaced by the line:

```
virtual void TestFunc () { Num1 += 10; }
```

After the code:

```
BaseClass *Object5 = new DerClass;
Object5->TestFunc ();
```

What is the value of Object5.Num1?

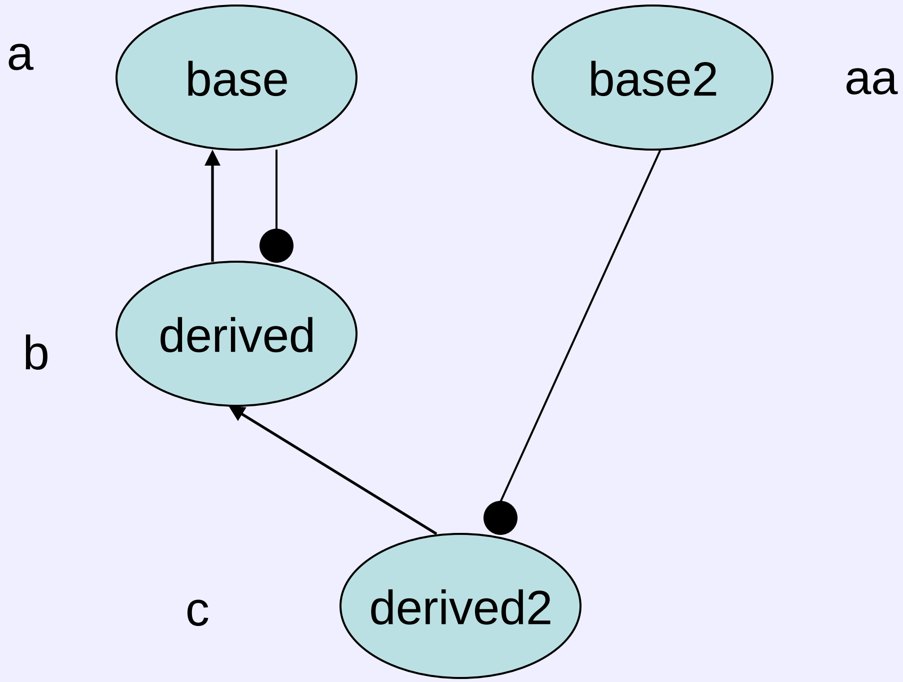
20   function call answered by child class.

Note:

- defining an object on a derived class makes no problem when function overloading occurs.

- However, defining a pointer to the base class and assigning to it an address to the derived class requires the overloaded functions in the base class to be virtual in order to run the overloading functions in the derived class.

Example: show the order of execution of constructor and destructor functions of classes that contain member objects and are related by inheritance.



```
// base.h
class base
{
  protected:
    float a;
  public:
    base(float s=0);
    ~base();
};
```

```
// base.h
class base2 {
   protected:
        float aa;
   public:
        base2(float s=0);
        ~base2();
};

class derived:public base {
  private:
      float b;
      base obj1;
   public:
      derived(float f=0,float g=0);
      ~derived();
};
```

```
// base.h
class derived2:public derived
{
  private:
    float c;
    base2 obj2;
  public:
    derived2(float f=0,float g=0, float h=0);
    ~derived2();
};
```

```
// base.cpp
#include "base.h"
#include <iostream.h>

base::base(float d):a(d) {
    cout<<"base class constructor"<<'\n'<<"a= "<<a<<endl;
    return; }

base::~base() {
    cout<<"base class destructor"<<'\n'<<"a= "<<a<<endl;
    return; }

base2::base2(float d):aa(d) {
    cout<<"base2 class constructor"<<'\n'<<"aa=
    "<<aa<<endl;
    return; }
```

```
// base.cpp
base2::~base2() {
    cout<<"base2 class destructor"<<'\n'<<"aa= "<<aa<<endl;
    return; }

derived::derived(float d,float e):base(d),b(e) {
    cout<<"derived class constructor"<<'\n'<<"b= "<<b<<endl;
    return; }

derived::~derived() {
    cout<<"derived class destructor"<<'\n'<<"b= "<<b<<endl;
    return; }
```

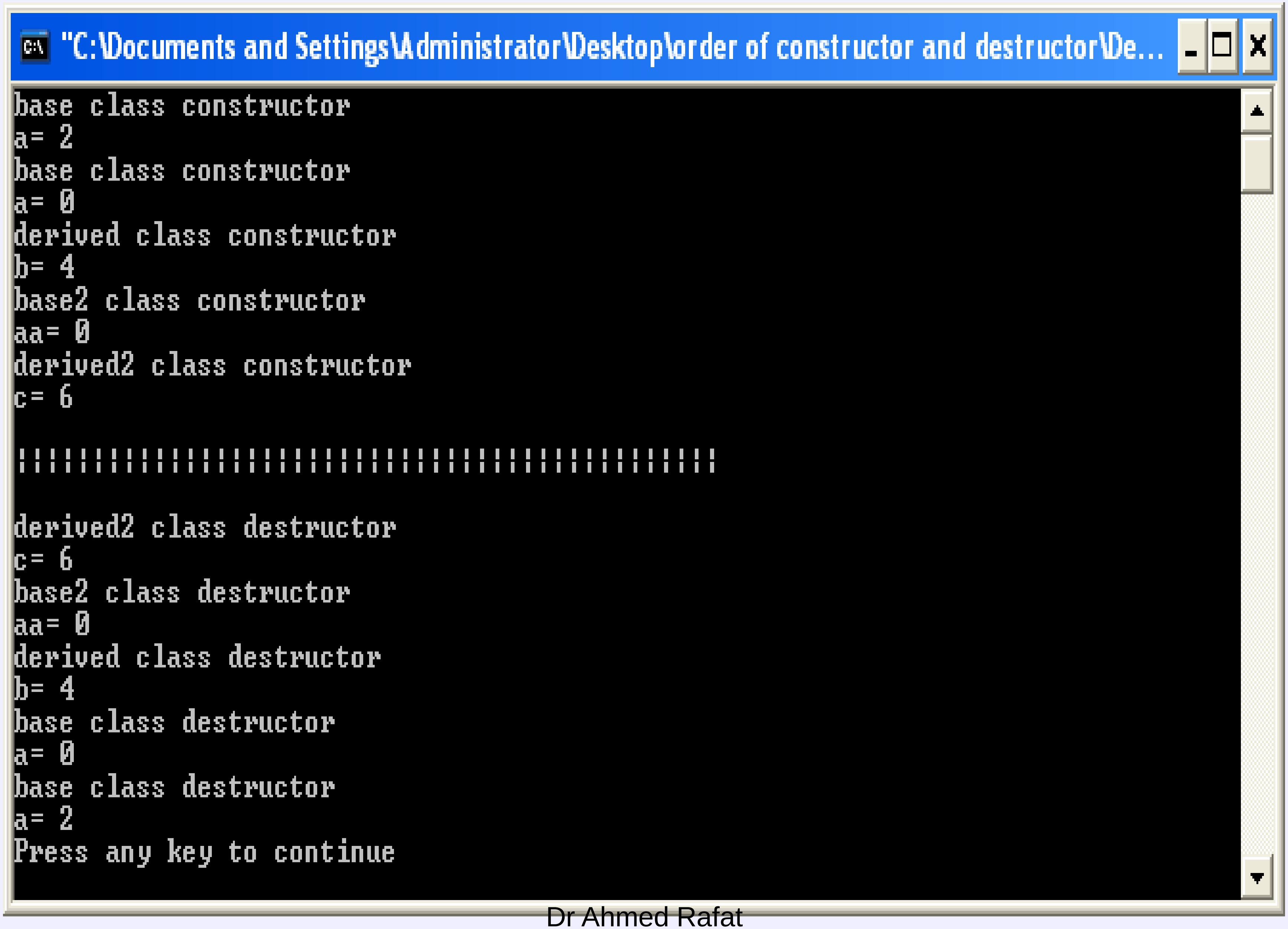
```
// base.cpp
derived2::derived2(float d,float e,float k):derived(d,e),c(k) {
    cout<<"derived2 class constructor"<<'\n'<<"c= "<<c<<endl;
    return; }

derived2::~derived2() {
    cout<<"derived2 class destructor"<<'\n'<<"c= "<<c<<endl;
    return; }
```

```
// main.cpp
#include "base.h"
#include<iostream.h>
showorder();

main() {
    showorder();
    return 0; }

showorder() {
    derived2 dcc(2,4,6);
    // base, base, derived, base2, derived2
    // a = 2, a = 0, b = 4, aa = 0, c = 6
    cout<<endl;
    cout<<"||||||||||||||||||||||||||||||||||||||||||||||"<<endl<<endl;
    return 0; }
```



"C:\Documents and Settings\Administrator\Desktop\order of constructor and destructor\De...

```
base class constructor
a= 2
base class constructor
a= 0
derived class constructor
b= 4
base2 class constructor
aa= 0
derived2 class constructor
c= 6

¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦

derived2 class destructor
c= 6
base2 class destructor
aa= 0
derived class destructor
b= 4
base class destructor
a= 0
base class destructor
a= 2
Press any key to continue
```

Note:

The order of execution of constructor functions of classes with member objects and related by inheritance is:

- Initialization list of the derived class
- Initialization list of the base class
- Member objects of the base class
- Constructor of the base class
- Member objects of the derived class
- Constructor of the derived class

Derived classes

Chapter 6

Dr Ahmed Rafat

5.1 Single inheritance

- ***Single inheritance*** **is the relationship between classes that results when a new class is created using the properties of an existing class.**

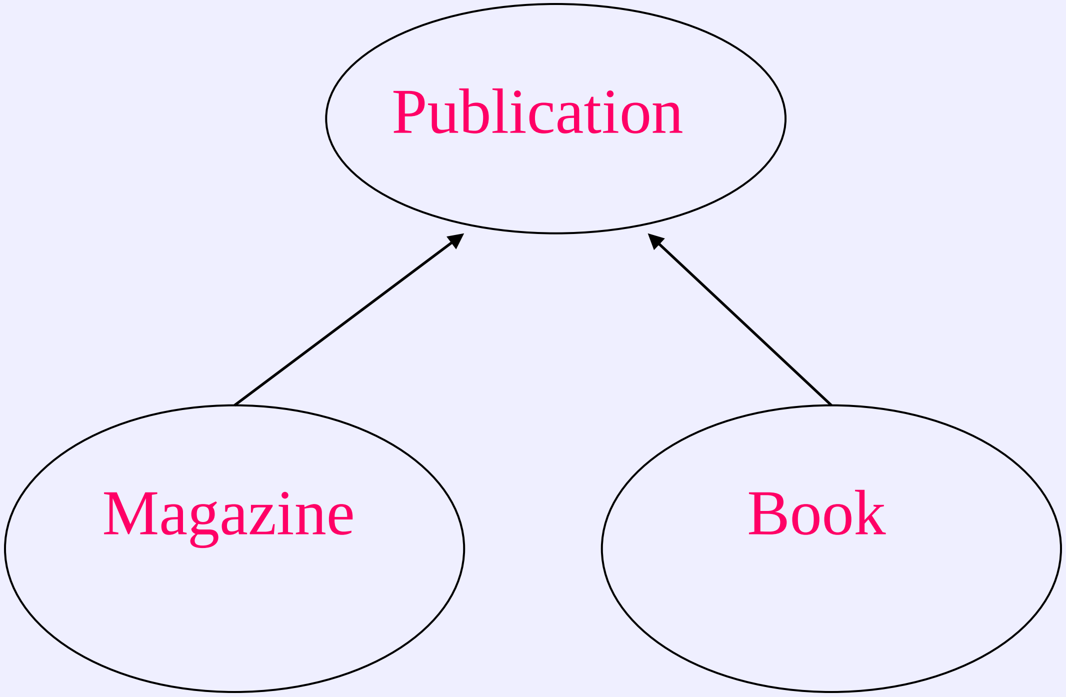
- **The new class, called the *derived* class, shares the structure and behavior of the original class, called the *base* class.**

- Inheritance relationships enable derived classes to borrow attributes and operations from existing classes, thus reducing the amount of redundant code in programs.

- similarities can be expressed through a connected, directed graph, called inheritance tree.

- An *inheritance tree* is a directed graph in which derived classes point toward their base classes.

Example: The following graph shows the inheritance tree of a base class called Publication and two derived classes from it called Magazine and Book.



Using C++. the classes are expressed as follows:

```
#include "fstring.h"
class Publication
{
   public:
      void Set Publisher ( const char * s );
      void SetDate( unsigned long dt );
   private:
      FString publisher;
      unsigned long date;
} ;
```

```
class Magazine :public Publication
{
public:
    void SetIssuesPerYear( unsigned n );
    void SetCirculation( unsigned long n );
private:
    unsigned issuesPerYear;
    unsigned long circulation;
} ;

class Book :public publication {
public:
    void SetISBN( const char' s );
    void SetAuthor( const char' s );
private:
    FString ISBN;
    FString author;
};
```

5.2 Friends in Derived Classes

- A friend function is granted access to all public, private, and protected members of a class.
- But a friend of a base class is not automatically a friend of classes derived from that base.

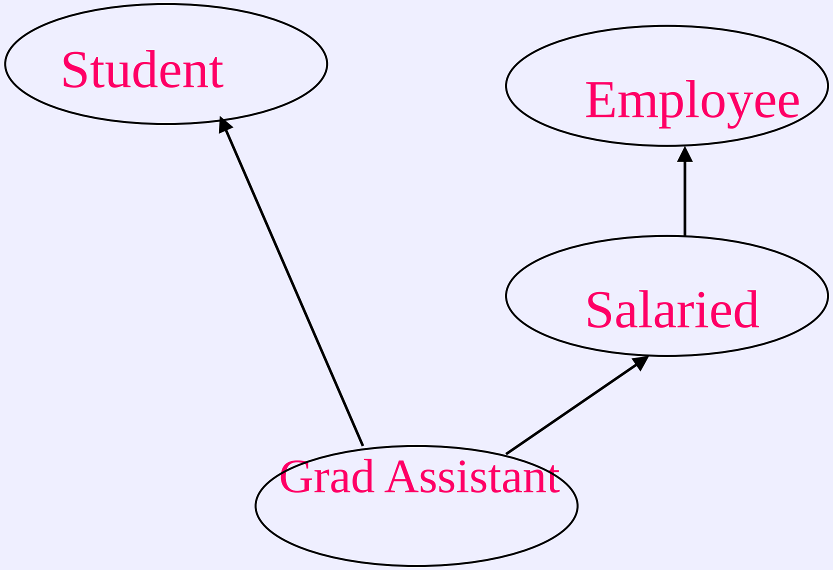
Example:

```
class Employee : public Person
{
public:
   friend float CalcPay( Employee & E };
};
```

5.3 Multiple inheritance

- Sometimes, a single class contains attributes and properties inherited from two or more base classes.
- This creates a *multiple inheritance* relationship.

Example:



class GradAssistant :public Student, public Salaried

- A *base List* is a list of one or more classes from which the current class is derived.

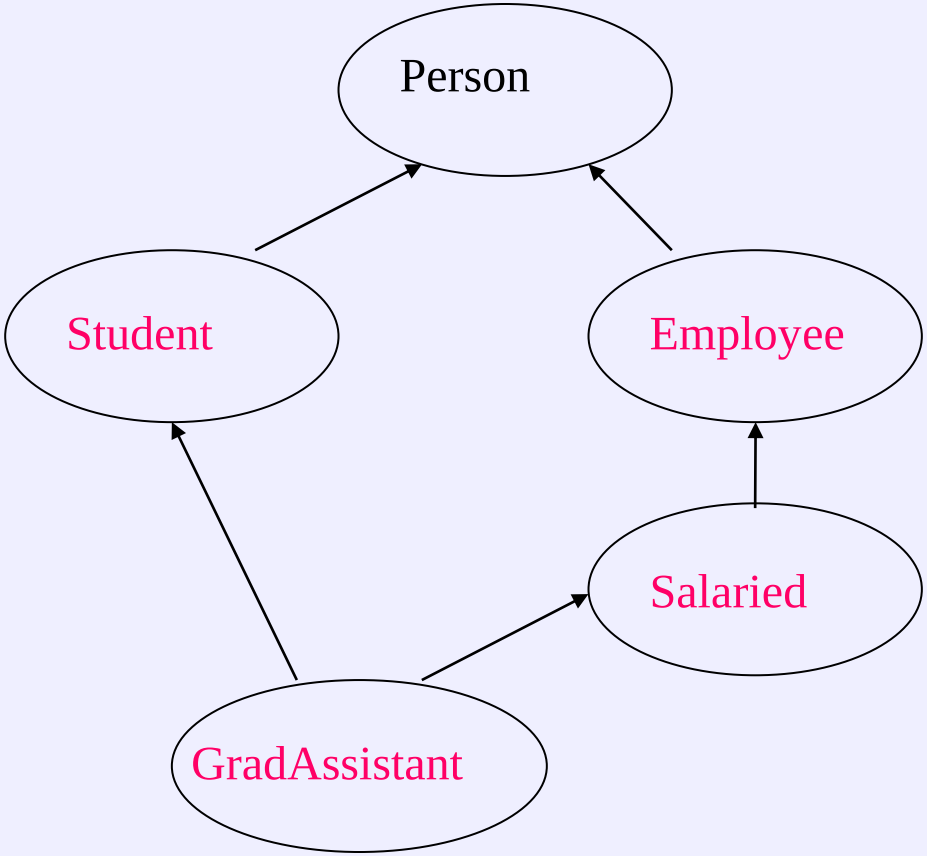
- A *member List* is a list of data members and function members that make up a class definition.

5.4 Virtual base classes

- Multiple inheritance can create naming problems when identical names appear in more than one base class.

- This ambiguity can be resolved by using a *virtual base class* that contains all identical names in the class hierarchy.

Example:



```
class Student :public virtual person
{
   // …
};

class Employee :public virtual Person
{
   // …
};
```

- With these changes, it is now possible to call any function in the Person class without causing an ambiguous member reference.