

Object Oriented Programming (CS1143)

Week 7

Department of Computer Science

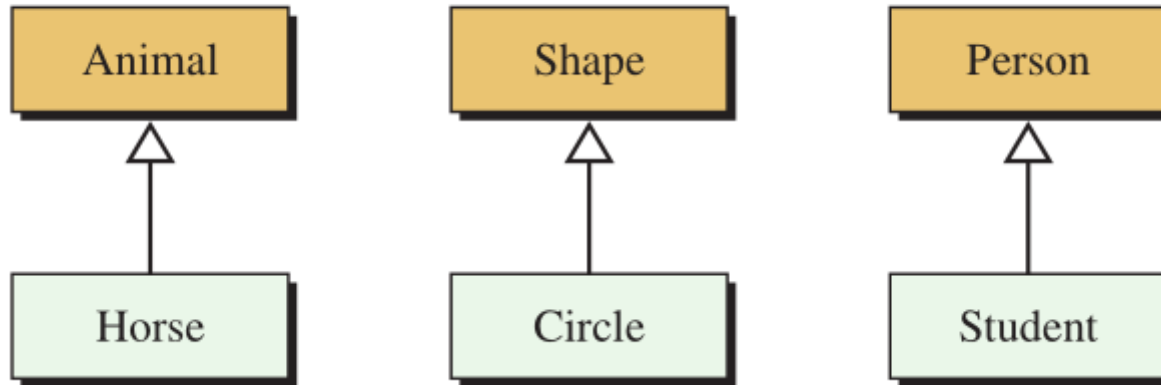
Capital University of Science and Technology (CUST)

Outline

- Single Public, Private, Protected Inheritance
- Constructor and Destructor Chaining

Inheritance

- Inheritance enables you to define a general class (i.e., a base class) and later extend it to more specialized classes (i.e., derived classes).
- For example we can give the definition of an animal; we can then add to the definition to create the definition of a horse.
- There is an is-a relation from the more specific to the more general.
 - All horses are animals.



Base Class and Derived Class

- In C++, the most general class is called the base class (or super class) and a more specific class is called the derived class (or subclass).
- The derived class inherits all of the data members and member functions of the base class (with the exception of constructors and destructors), and it can create new data members and member functions.

Private, Public and Protected Inheritance

- To create a derived class from a base class, we have three choices in C++: private inheritance, protected inheritance, and public inheritance.
- To show the type of the inheritance we want to use, we insert a colon after the class, followed by one of the keywords (private protected, or public)
- The most common is the public inheritance.
- In the figure below, B is the base class and D is the derived class.

```
class D : public B
{
    ...
};
```

Public inheritance

```
class D : protected B
{
    ...
};
```

Protected inheritance

```
class D : private B
{
    ...
};
```

Private inheritance

Public Inheritance

- The most common type of inheritance is public inheritance. The other two types of inheritance are rarely used.
- Some other object-oriented languages, like Java, have only public inheritance
- We will discuss it using an example

Example

Example

- We design two classes, Person and Student
- The class Student inherits from the class Person.
- Person class uses only one data member: identity.
- Student class needs two data members: identity and gpa.
- However, since the identity data member is already defined in the class Person, it does not need to be defined in the class Student because of inheritance.
- Similarly the Student class does not need to define its own set and get functions for identity.

Person Class

```
5  class Person
6  {
7      private:
8          int identity;
9      public:
10         void setId (int identity);
11         int getId( ) const;
12     };
13
14 void Person :: setId (int id)
15 {
16     identity = id;
17 }
18
19 int Person :: getId ( ) const
20 {
21     return identity;
22 }
```

Student Class

```
25 class Student : public Person
26 {
27     private:
28         double gpa;
29     public:
30         void setGPA (double gpa);
31         double getGPA () const;
32 };
33
34 void Student :: setGPA (double gp)
35 {
36     gpa = gp;
37 }
38
39 double Student :: getGPA() const
40 {
41     return gpa;
42 }
```

main

```
45 int main ( )
46 {
47     Person person;
48     person.setId (1111);
49     cout << "Person Information: " << endl;
50     cout << "Person's identity: " << person.getId ( );
51     cout << endl << endl;
52
53     Student student;
54     student.setId (2222);
55     student.setGPA (3.9);
56     cout << "Student Information: " << endl;
57     cout << "Student's identity: " << student.getId() << endl;
58     cout << "Student's gpa: " << student.getGPA();
59
60     return 0;
61 }
```

Output

D:\Work Umair\4_CUST (1-9-22)\1_Teaching\2_ACS1143-OOP\Practice Programs\W7-P1.exe

Person Information:

Person's identity: 1111

Student Information:

Student's identity: 2222

Student's gpa: 3.9

Process exited after 0.02672 seconds with return value 0

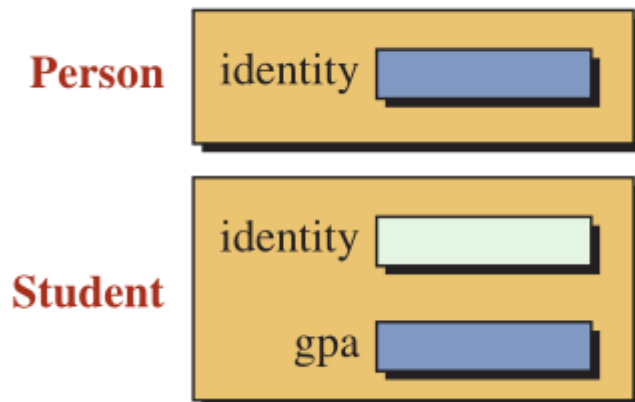
Press any key to continue . . . ■

Description

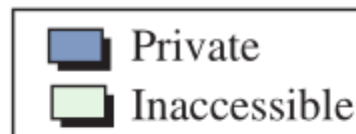
- On line 54, an object of the student class calls the member function of its base class (Person) as if it were its own function

Private Members in Public Inheritance

- A private data member in the base class is inherited in the derived class, but it becomes inaccessible in the derived class; it must be accessed only through its own class member functions.



Legend:



Note:

The inherited private data members become inaccessible in the derived class. They need to be accessed through the base class.

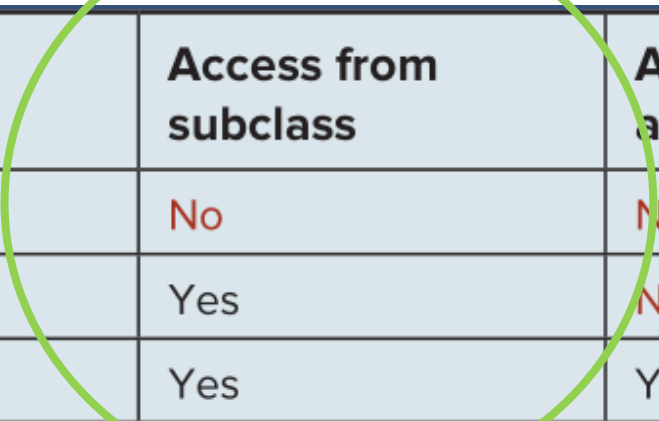
Public and Protected Members in Public Inheritance

- A public member in the base class becomes a public member in the derived class.
- A protected member in the base class becomes a protected member in the derived class.
- The functions defined in the derived class can easily access a protected member without having to call a function inherited from the base class

Public
inheritance



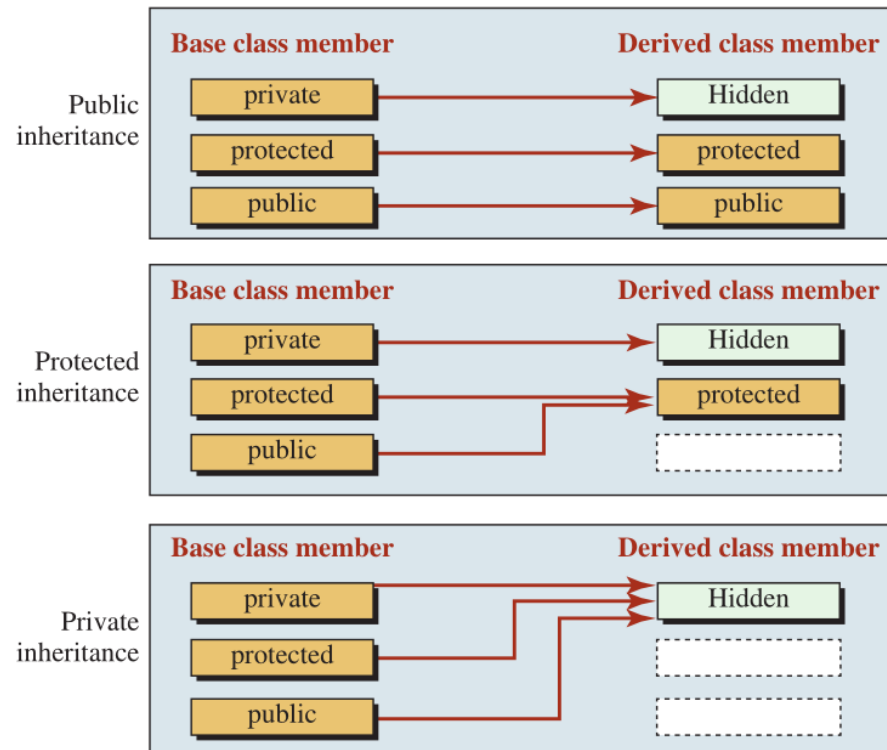
Week3. Slide 6.



Modifier	Access from same class	Access from subclass	Access from anywhere
private	Yes	No	No
protected	Yes	Yes	No
public	Yes	Yes	Yes

Three Types of Inheritance

- Although public inheritance is by far the most common type of derivation, C++ allows us to use two other types of derivation: private and protected.



Outline

- Single Public, Private, Protected Inheritance
- Constructor and Destructor Chaining

Constructors and Destructors

- Unlike data fields and functions, the constructors and the destructor of a base class are not inherited in the derived class.
- The constructors and the destructor are not inherited because an object of a derived class naturally has more data members than a corresponding base class.
 - The constructor of a derived class must construct more; the destructor of a derived class must destruct more.

Calling Base Class's Constructors

- You can invoke the base class's constructor from the constructor initializer list of the derived class.

```
DerivedClass(parameterList): BaseClass(argumentList)
{
    // Perform initialization
}
```

- If a base constructor is not invoked explicitly, the base class's no-argument constructor (default constructor) is invoked by default.

Constructor Chaining

- When constructing an object of a derived class, the derived class constructor first invokes its base class constructor before performing its own tasks.
- This is called constructor chaining.

Destructor Chaining

- The destructors are automatically invoked in reverse order.
- When an object of a derived class is destroyed, the derived class destructor is called. After it finishes its tasks, it invokes its base class destructor.
- This is called destructor chaining.

Example

Calling base class constructor implicitly

```

3  class Person
4  {
5      public:
6          Person();
7          ~Person();
8  };
9  Person::Person()
10 {
11     cout << "Performs tasks for Person's constructor" << endl;
12 }
13
14 Person::~~Person()
15 {
16     cout << "Performs tasks for Person's destructor" << endl;
17 }

```

```


20 class Employee: public Person
21 {
22     public:
23         Employee();
24         ~Employee();
25 };
26
27 Employee::Employee()
28 {
29     cout << "Performs tasks for Employee's constructor" << endl;
30 }
31 Employee::~~Employee()
32 {
33     cout << "Performs tasks for Employee's destructor" << endl;
34 }

```

```

37 int main()
38 {
39     Employee emp;
40     return 0;
41 }

```

 D:\Work Umair\4_CUST (1-9-22)\1_Teaching\2_ACS1143-OOP\Practice Programs\W7-P2.exe

Performs tasks for Person's constructor
 Performs tasks for Employee's constructor
 Performs tasks for Employee's destructor
 Performs tasks for Person's destructor

 Process exited after 0.05352 seconds with return value 0
 Press any key to continue . . . █

Example

Calling base class constructor explicitly

```

1  #include <iostream>
2  using namespace std;
3  class Person
4  {
5      private:
6          string name;
7      public:
8          Person();
9          Person(string n);
10         ~Person();
11 };
12 Person::Person(string n)
13 :name(n)//initializing name with n
14 {
15     cout << "Person's parameter constructor" << endl;
16 }
17 Person::Person()
18 {
19     cout << "Person's default constructor" << endl;
20 }
21
22 Person::~~Person()
23 {
24     cout << "Person's destructor" << endl;
25 }

```

```

28 class Employee: public Person
29 {
30     private:
31         int id;
32     public:
33         Employee(int i, string name);
34         Employee();
35         ~Employee();
36 };
37
38 Employee::Employee(int i, string name)
39 :Person(name),id(i)//calling Person's constructor and initializing id with i
40 {
41     cout << "Employee's param constructor" << endl;
42 }
43
44 Employee::Employee()
45 {
46     cout << "Employee's default constructor" << endl;
47 }
48
49 Employee::~~Employee()
50 {
51     cout << "Employee's destructor" << endl;
52 }

```

```
55 int main()  
56 {  
57     Employee emp2(555, "Ahmad");  
58     return 0;  
59 }
```

D:\Work Umair\4_CUST (1-9-22)\1_Teaching\2_ACS1143-OOP\Midterm Exam\W7-P3.exe

Person's parameter constructor
Employee's param constructor
Employee's destructor
Person's destructor

Process exited after 0.02455 seconds with return value 0
Press any key to continue . . .