Object-Oriented Programming
(in C++)

Inheritance

Professor Yi-Ping You (游逸平)
Department of Computer Science
http://www.cs.nctu.edu.tw/~ypyou/

Important OO Concepts: P.I.E.

- Encapsulation (abstract data type)
  - "Black box" – information hiding

- Inheritance
  - Related classes share implementation and/or interface, allowing reuse of codes
- Polymorphism
  - Ability to use a class without knowing its type

Outline

- Introduction to Inheritance
- Types of Inheritance
- Constructors and Destructors in Derived Classes
- A Case Study: Employees
- Multiple Inheritance

Recall: Real World vs Programming World

- Real world
  - Objects that share the same attributes form a class
- Programming world (programmers are God)
  - Programmers define classes that they need and create objects from the classes
  - Programmers design computer programs to *describe/represent* real-world objects or virtual objects

Software Reusability: Composition

- Composition is a form of software reuse
  - Objects as data members
  - An X object **has a** Y object
  - For example
    - A *computer* has a *CPU*
    - A *human being* has two *feet*
    - A *car* has a *steering wheel*, a *brake pedal*, and many other components
- "has-a" relationship between two classes

Software Reusability: Inheritance

- Inheritance is another form of software reuse
  - You can create a class that absorbs an existing class's members and enhances them with new capabilities
  - An X object **is a** Y class object
  - For example
    - A *car* is a *vehicle* (but not vice versa)
      - Both have wheels, and can run, stop, etc.
    - A *student* is a *person* (but not vice versa)
      - Both have ID, name, and birthday, and can eat, sit, sleep, etc.
- "is-a" relationship between two classes

Inheritance

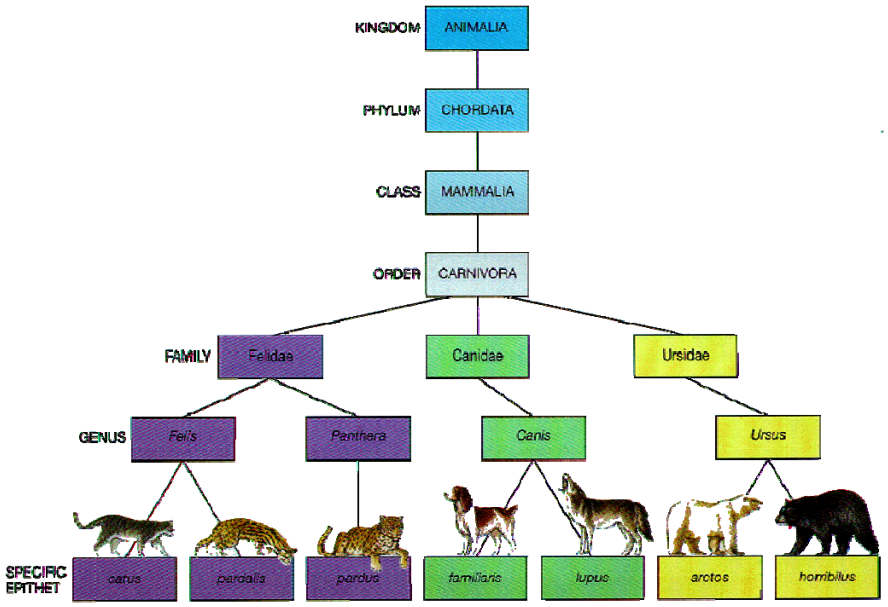
- The definition of a class inherits from the definition of another class
- A derived class inherits from a base class
  - The base class also called superclass
  - The derived class also called subclass
- Derived class
  - A more specialized group of objects
  - Members are inherited from its base class
  - Can have additional members
  - Can customize its members

Base Classes and Derived Classes

- Base class typically represents larger set of objects than derived classes
  - Example:
    - Base class: `Vehicle`
      - Includes cars, trucks, boats, airplanes, bicycles, etc.
    - Derived class: `Car`
      - A smaller, more specific subset of vehicles

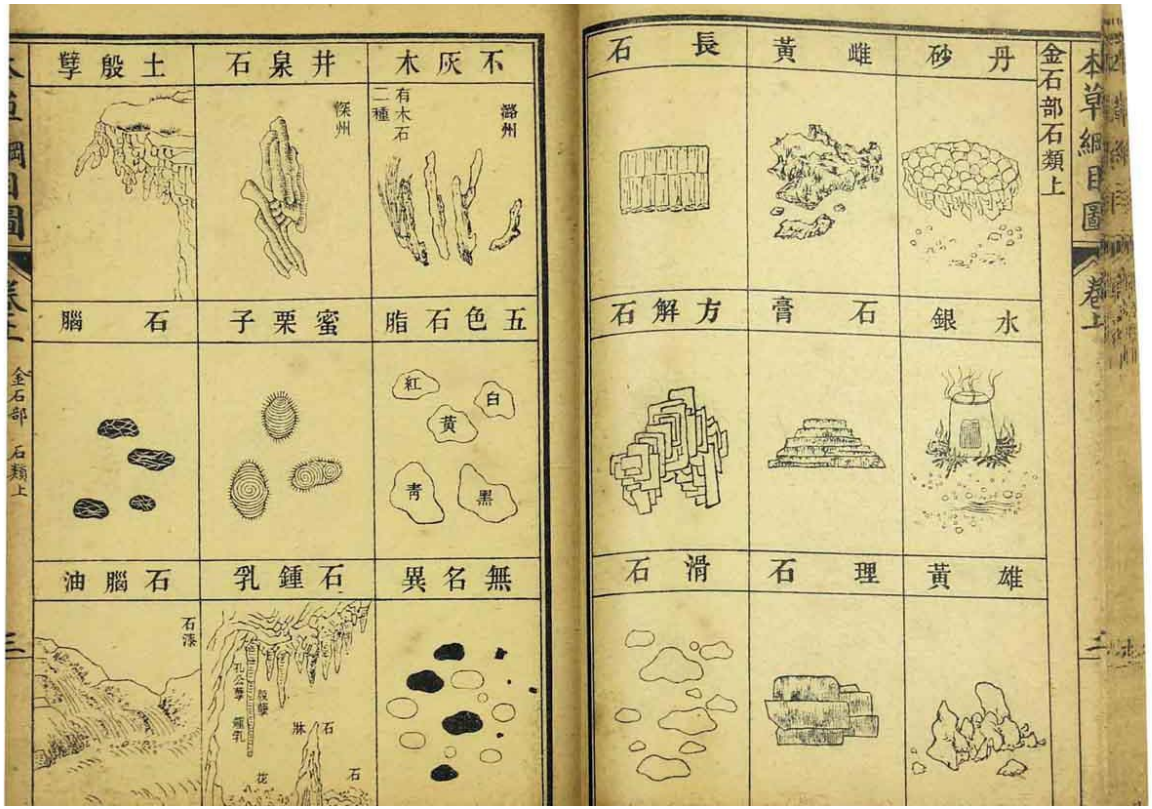
Inheritance Relationships

- Programmers write computer programs to simulate the real world
  - Thus, programmers have to write many classes and define inheritance relationships among classes if necessary
  - Just like we did for biological classification



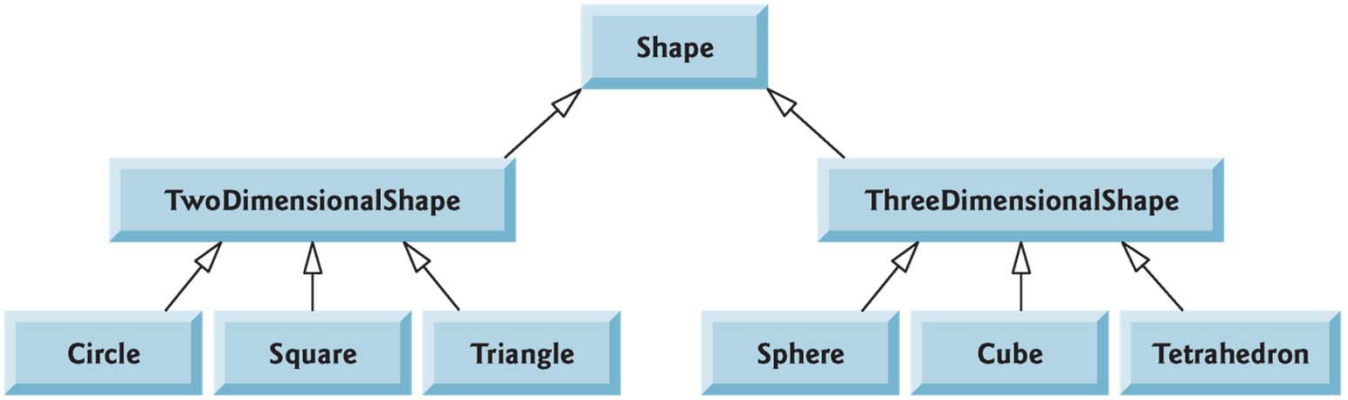
Chinese Ancient Taxonomy

李時珍《本草綱目》



Inheritance Hierarchy

- Inheritance relationships: tree-like hierarchy structure



- Each class becomes either
  - A base class
    - Supplies data/behaviors to other classes

  OR

  - A derived class
    - Inherits data/behaviors from other classes

Inheriting All Members of Base Class

- In principle, a derived class inherits all members (data members and member functions) of its base class, except constructors, destructor, and overloaded assignment operators
  - Just with different access permissions
    - Depending on which type of inheritance is used
  - `friend` declarations are not members, and thus not inherited
- The size of a derived-class object depends on the data members declared in its class definition *and* the data members inherited its direct and indirect base classes

Syntax of Inheritance

```
class Child : access-specifier Parent {
  // customized members or/and
  // additional members
};
```

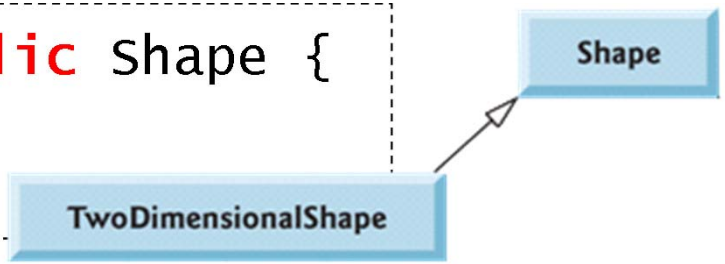
- `access-specifier` determines the type of inheritance
  - `public`
    - `public` inheritance
  - `protected`
    - `protected` inheritance
  - `private` (by default if not specified)
    - `private` inheritance

Outline

- Introduction to Inheritance
- Types of Inheritance
- Constructors and Destructors in Derived Classes
- A Case Study: Employees
- Multiple Inheritance

public Inheritance (1/2)

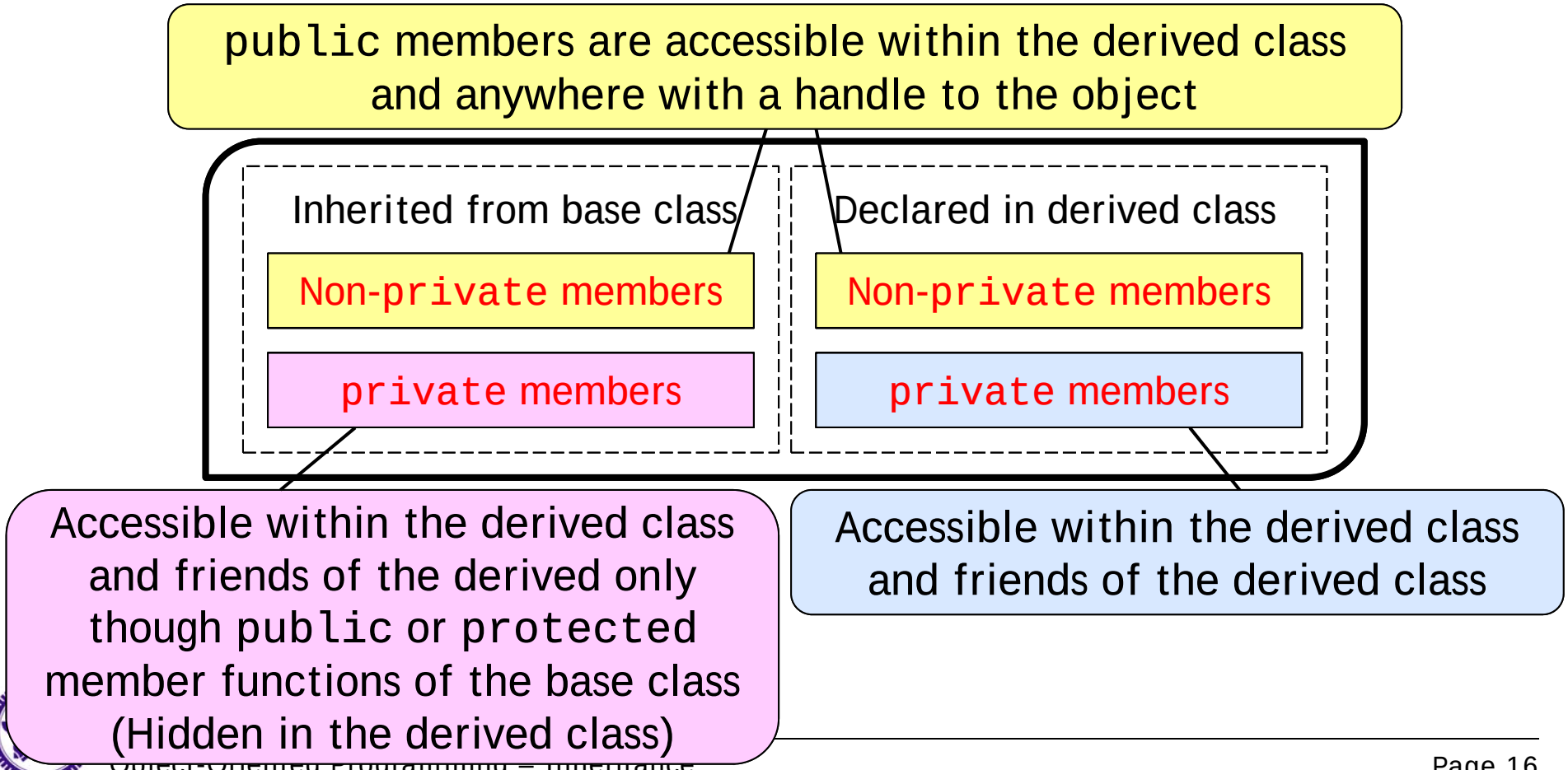
```
class TwoDimensionalShape : public Shape {
  ...
};
```



- private base-class members are not accessible directly from that the derived class when they become members of the derived class
- Non-private base-class members retain their original member access when they become members of the derived class

public Inheritance (2/2)

- The object of a derived class behaves as if it were of the type of its base class with additional attributes or/and behaviors



An Example: Classes without Inheritance

```
class Polygon {
private:
  int width, height;
public:
  void set_values(int a, int b)
    { width=a; height=b; }
};
```

Polygon

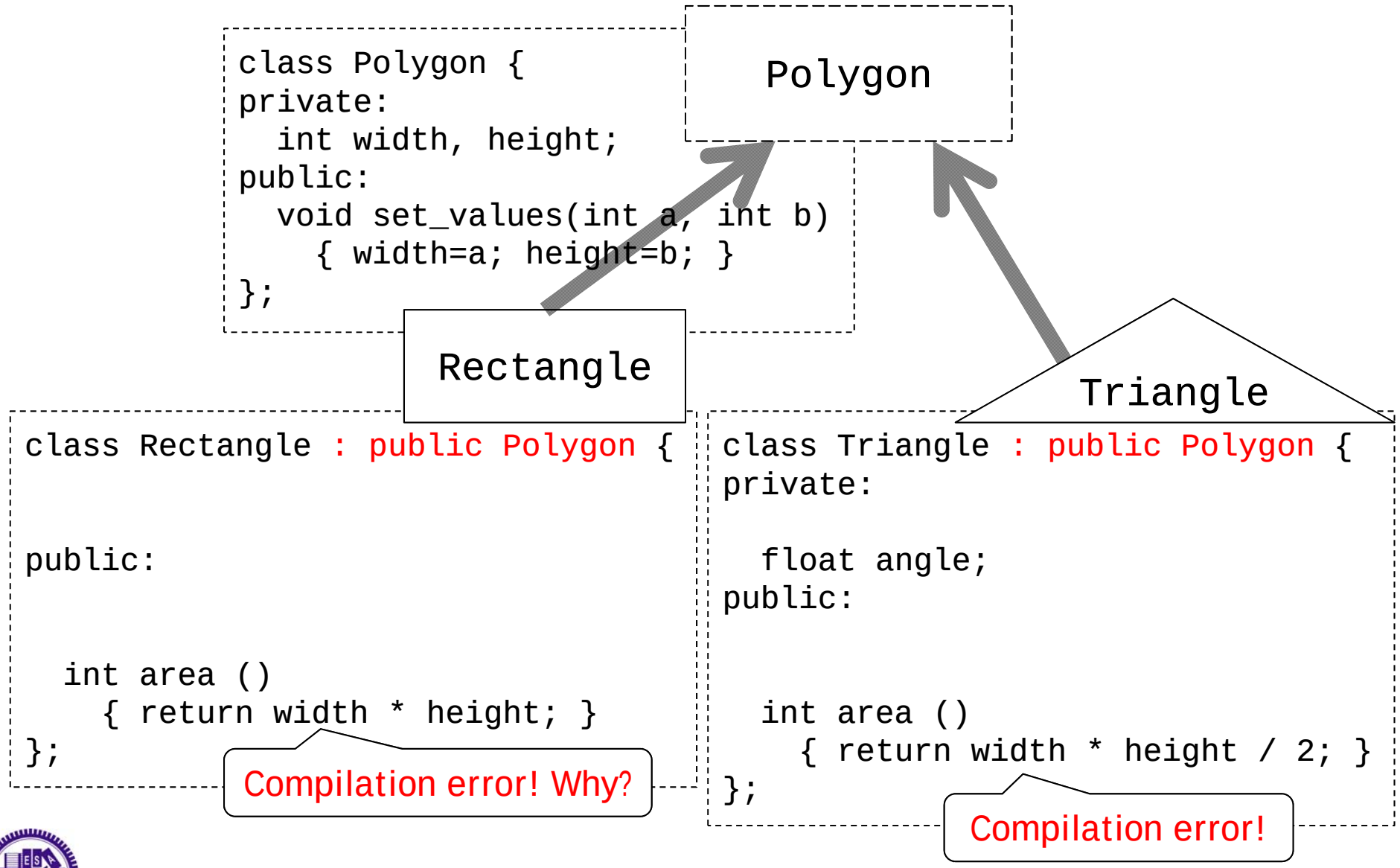
Rectangle

Triangle

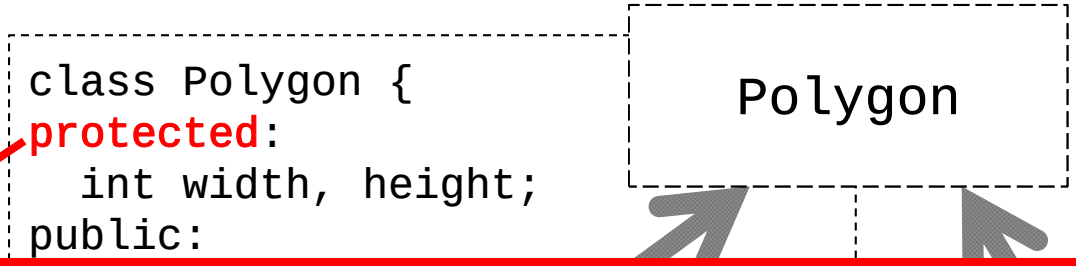
```
class Rectangle {
private:
  int width, height;
public:
  void set_values(int a, int b)
    { width=a; height=b; }
  int area ()
    { return width * height; }
};
```

```
class Triangle {
private:
  int width, height;
  float angle;
public:
  void set_values(int a, int b)
    { width=a; height=b; }
  int area ()
    { return width * height / 2; }
};
```

An Example: Classes with Inheritance (1/2)



An Example: Classes with Inheritance (2/2)



Allows derived classes (and of course `Polygon` class and the `friends` of `Polygon` class) to access the protected members, but disallows others to access them

```
class Rectangle : public Polygon {


public:


  int area ()
    { return width * height; }
};
```

```
class Triangle : public Polygon {
private:

  float angle;
public:


  int area ()
    { return width * height / 2; }
};
```

`protected` Members

- `protected` access
  - Intermediate level of protection between `public` and `private`
  - `protected` members are accessible within
    - Base class
    - Derived class
    - `friend`s of base class
    - `friend`s of derived class

Access Control and Inheritance

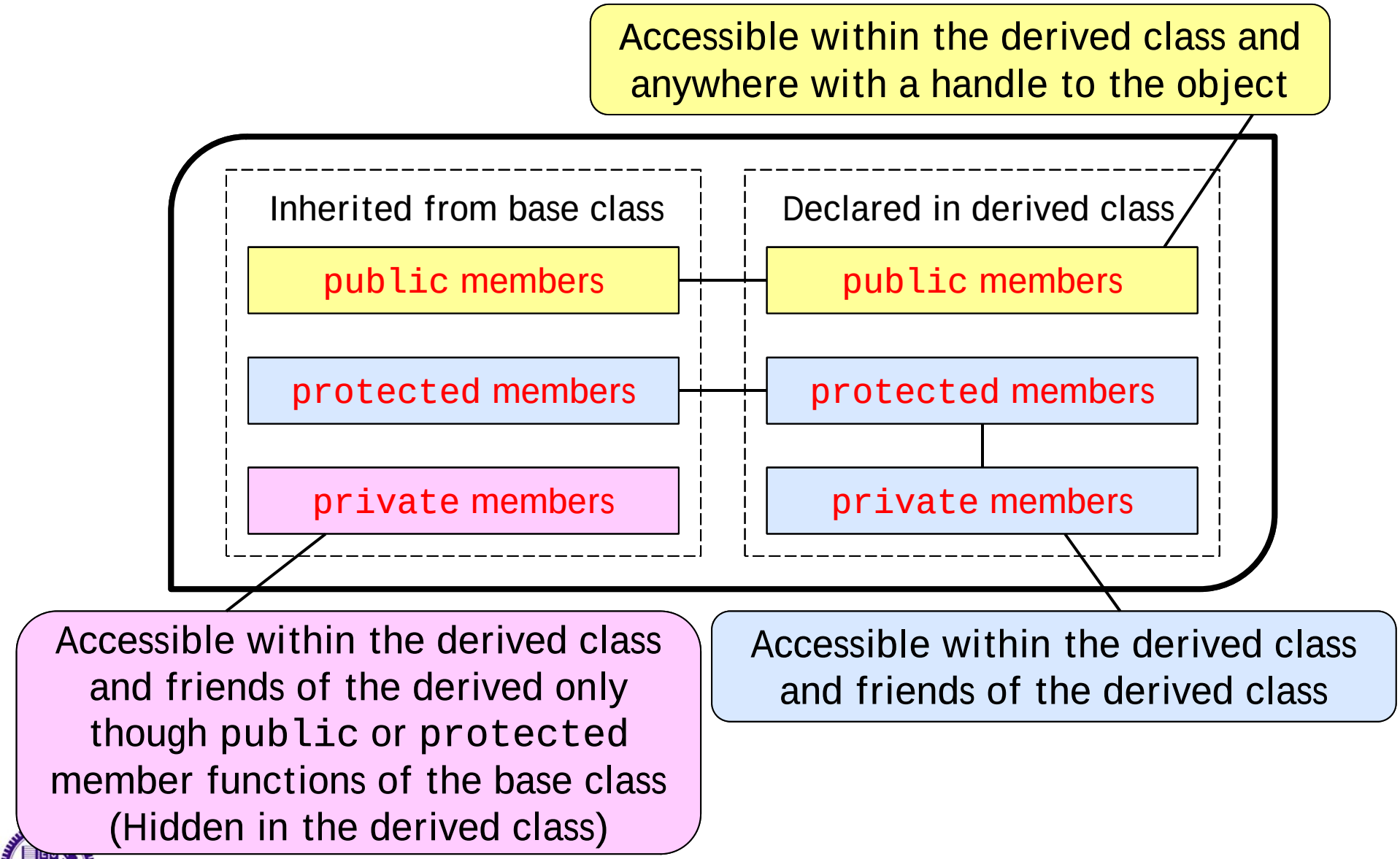
- Can it access a member of a class?

```
class theClass {
  // a member
};
```

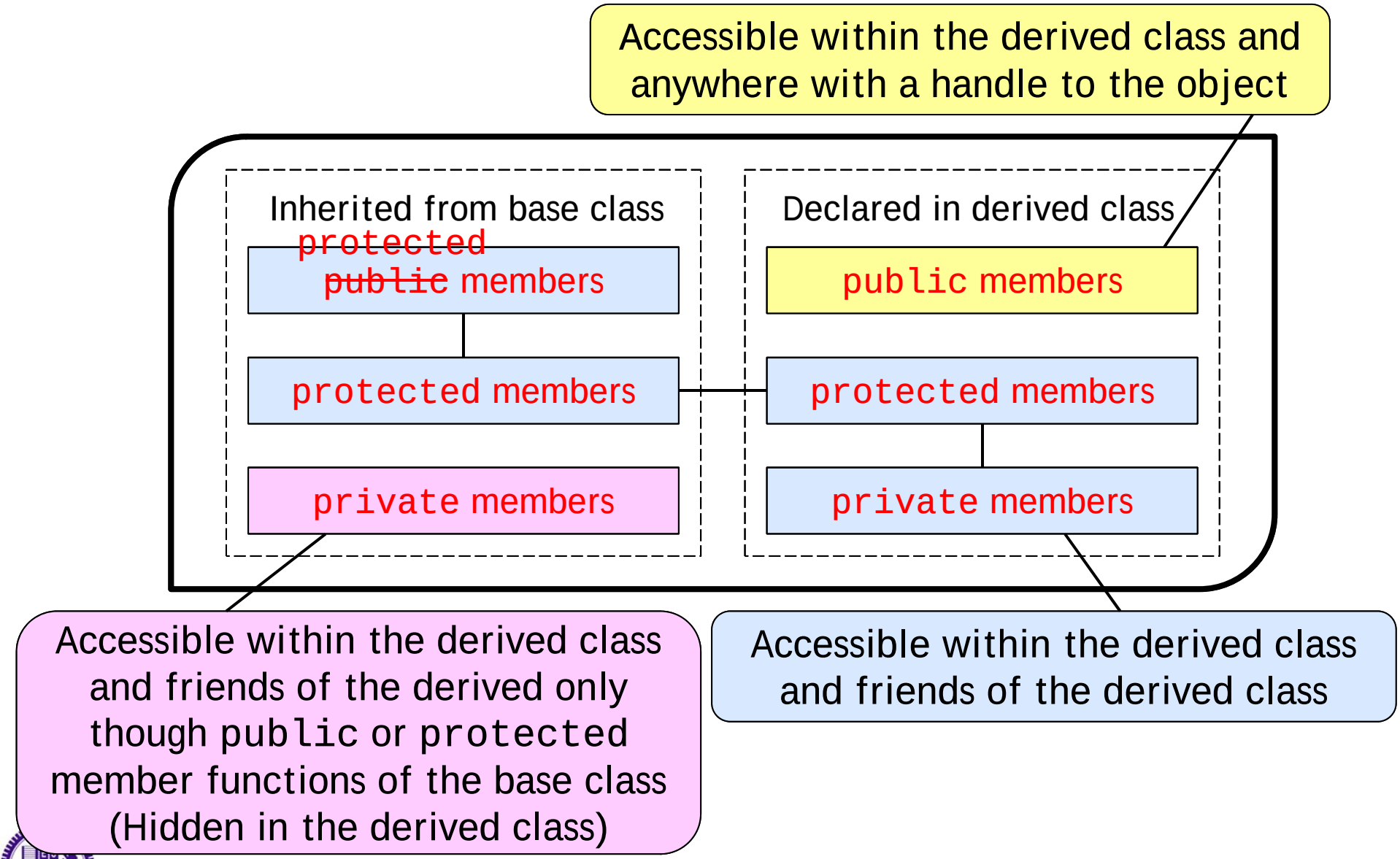
```
class derivedClass : public theClass {
  ...
};
```

Access	Members of the class		
	`public`	`protected`	`private`
Inside the class	yes	yes	yes
`friend` function or `friend` class	yes	yes	yes
Derived class (outside the class)	yes	yes	no
Outside the class	yes	no	no

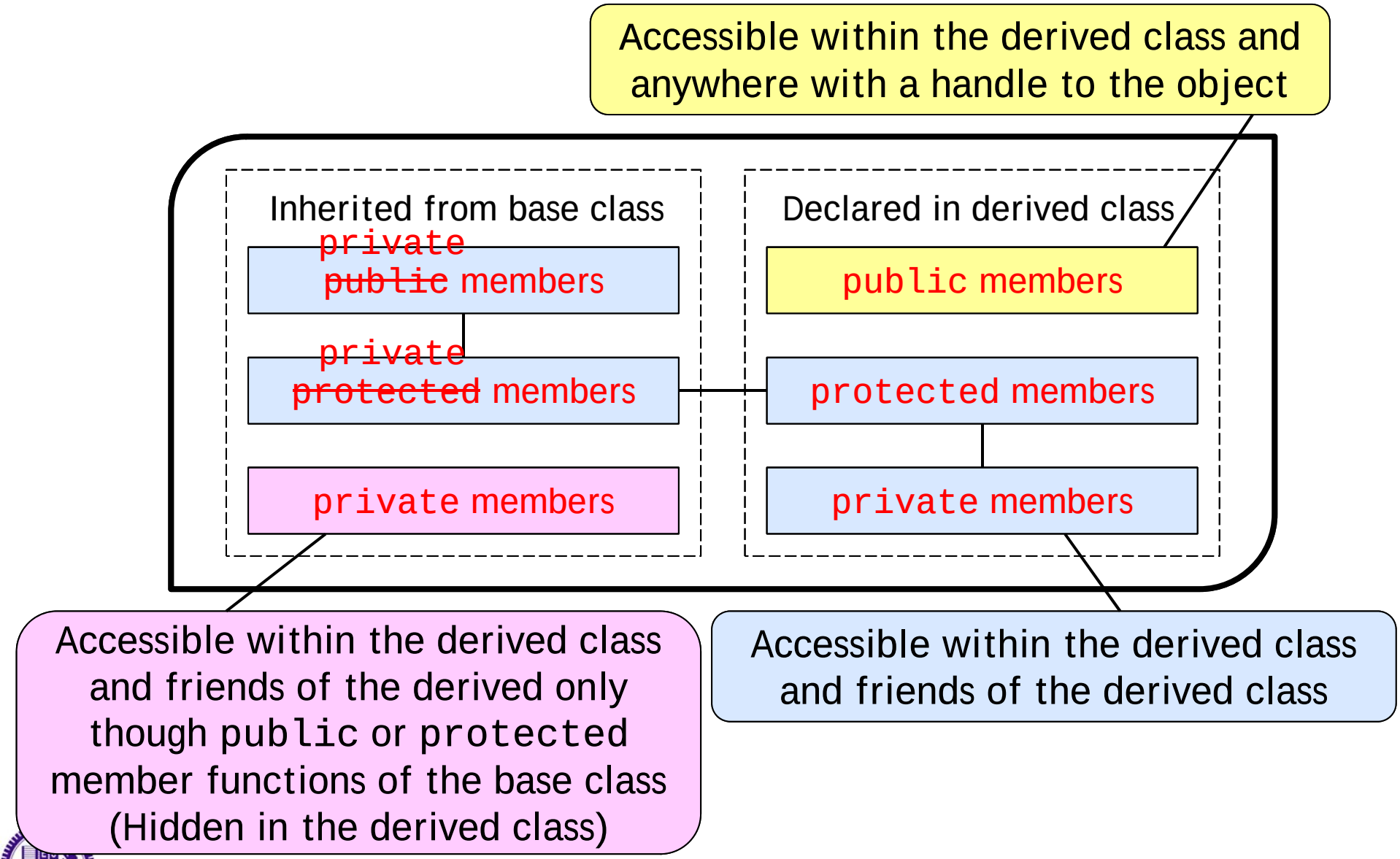
public Inheritance



protected Inheritance



`private` Inheritance



Types of Inheritance

- Member accessibility of a base class changes in a derived class

Inheritance	Members of base class	Members of derived class
public inheritance	public	public
	protected	protected
	private	private (hidden)
protected inheritance	public	protected
	protected	protected
	private	private (hidden)
private inheritance	public	private
	protected	private
	private	private (hidden)

“is-a” Relationship

- A derived-class object is a its base-class object only if the derived `public` inherits its base class
- “is-a” relationships between a derived-class object and its base-class object no longer last if the derived class is derived using `protected` or `private` inheritance

An Example of `private` Inheritance (1/2)

```
class single_linked_list {
  private:
    class node {
      public:
        node *link;
        int contents;
    };
    node *head;
  public:
    single_linked_list() {head = 0;}
    void insert_at_head(int) {...}
    void insert_at_tail(int) {...}
    int remove_at_head() {...}
    int empty() {...}
};
```

An Example of `private` Inheritance (2/2)

```
class stack : public single_linked_list {
  public:     private
    stack() {}
    void push(int value) {
      single_linked_list :: insert_at_head(value);
    }
    int pop() {
      return single_linked_list :: remove_at_head();
    }
};
class queue : public single_linked_list {
  public:     private
    queue() {}
    void enqueue(int value) {
      single_linked_list :: insert_at_tail(value);
    }
    int dequeue() {
      single_linked_list :: remove_at_head();
    }
};
```

- Clients of both `stack` and `queue` classes can access all of the public members of `single_linked_list`

Reading

- How are "private inheritance" and "composition" similar?
  - http://www.parashift.com/c++-faq/priv-inherit-like-compos.html

Outline

- Introduction to Inheritance
- Types of Inheritance
- Constructors and Destructors in Derived Classes
- A Case Study: Employees
- Multiple Inheritance

Constructors and Destructor

- A derived-class constructor must call it's direct base class's constructor before performing its own tasks
  - Explicitly calling via a member initializer

  or
  - Implicitly calling the base-class default constructor
- The derived-class destructor always implicitly calls it's direct base class's destructor after performing its own tasks

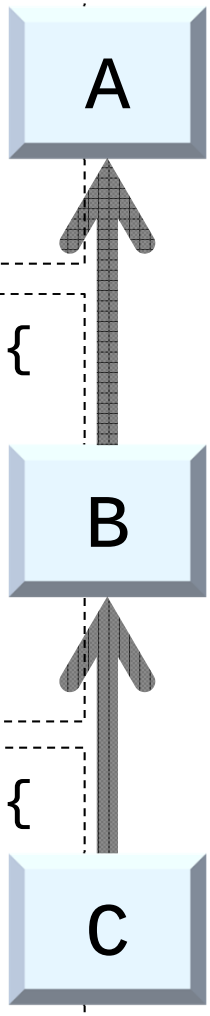
- A class has at least one constructor, but it has exactly only one destructor

Constructing and Destructing An Object

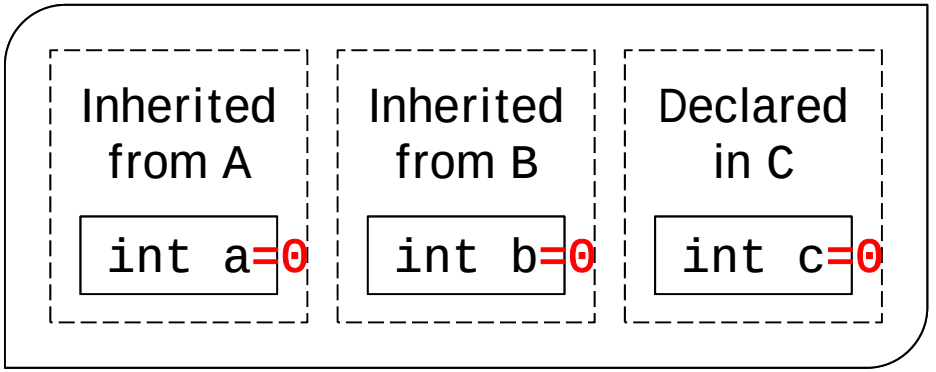
```
class A {
public:
  A() { a = 0; }
  int a;
};
```

```
class B : public A {
public:
  B() : A()
    { b = 0; }
  int b;
};
```

```
class C : public B {
public:
  C() { c = 0; }
  int c;
};
```



- When creating a C object
  1. Explicitly calling A()
  2. Implicitly calling B()
  3. Implicitly calling C()



- When destroying a C object
  1. Implicitly calling ~C()
  2. Implicitly calling ~B()
  3. Implicitly calling ~A()

Outline

- Introduction to Inheritance
- Types of Inheritance
- Constructors and Destructors in Derived Classes
- A Case Study: Employees
- Multiple Inheritance

Case Study: Employees

- We'd like to design two classes for two types of employees
  - `CommissionEmployee`
    - Commission employees are paid a percentage of their sales
  - `BasePlusCommissionEmployee`
    - Base-salaried commission employees receive a base salary plus a percentage of their sales

A Naïve Approach:

Designing `CommissionEmployee` and `BasePlusCommissionEmployee` without Using Inheritance

CommissionEmployee.h

```
 9  class CommissionEmployee
10  {
11  public:
12     CommissionEmployee( const string &, const string &, const string &,
13        double = 0.0, double = 0.0 );
14
15     void setFirstName( const string & ); // set first name
16     string getFirstName() const; // return first name
17
18     void setLastName( const string & ); // set last name
19     string getLastName() const; // return last name
20
21     void setSocialSecurityNumber( const string & ); // set SSN
22     string getSocialSecurityNumber() const; // return SSN
23
24     void setGrossSales( double ); // set gross sales amount
25     double getGrossSales() const; // return gross sales amount
26
27     void setCommissionRate( double ); // set commission rate (percentage)
28     double getCommissionRate() const; // return commission rate
29
30     double earnings() const; // calculate earnings
31     void print() const; // print CommissionEmployee object
32  private:
33     string firstName;
34     string lastName;
35     string socialSecurityNumber;
36     double grossSales; // gross weekly sales
37     double commissionRate; // commission percentage
38  }; // end class CommissionEmployee
```

CommissionEmployee.cpp (Incomplete)

```
3  #include <iostream>
4  #include "CommissionEmployee.h" // CommissionEmployee class definition
5  using namespace std;
6
7  // constructor
8  CommissionEmployee::CommissionEmployee(
9     const string &first, const string &last, const string &ssn,
10    double sales, double rate )
11 {
12    firstName = first; // should validate
13    lastName = last; // should validate
14    socialSecurityNumber = ssn; // should validate
15    setGrossSales( sales ); // validate and store gross sales
16    setCommissionRate( rate ); // validate and store commission rate
17 } // end CommissionEmployee constructor
```

```
67 // set commission rate
68 void CommissionEmployee::setCommissionRate( double rate )
69 {
70    commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
71 } // end function setCommissionRate
```

```
79 // calculate earnings
80 double CommissionEmployee::earnings() const
81 {
82    return commissionRate * grossSales;
83 } // end function earnings
```

BasePlusCommissionEmployee.h

```
10  class BasePlusCommissionEmployee
11  {
12  public:
13     BasePlusCommissionEmployee( const string &, const string &,
14        const string &, double = 0.0, double = 0.0, double = 0.0 );
15
16     void setFirstName( const string & ); // set first name
17     string getFirstName() const; // return first name
18
19     void setLastName( const string & ); // set last name
20     string getLastName() const; // return last name
21
22     void setSocialSecurityNumber( const string & ); // set SSN
23     string getSocialSecurityNumber() const; // return SSN
24
25     void setGrossSales( double ); // set gross sales amount
26     double getGrossSales() const; // return gross sales amount
27
28     void setCommissionRate( double ); // set commission rate
29     double getCommissionRate() const; // return commission rate
30
31     void setBaseSalary( double ); // set base salary
32     double getBaseSalary() const; // return base salary
33
34     double earnings() const; // calculate earnings
35     void print() const; // print BasePlusCommissionEmployee object
36  private:
37     string firstName;
38     string lastName;
39     string socialSecurityNumber;
40     double grossSales; // gross weekly sales
41     double commissionRate; // commission percentage
42     double baseSalary; // base salary
43  }; // end class BasePlusCommissionEmployee
```

BasePlusCommissionEmployee.cpp (Incomplete)

```
 3  #include <iostream>
 4  #include "BasePlusCommissionEmployee.h"
 5  using namespace std;
 6
 7  // constructor
 8  BasePlusCommissionEmployee::BasePlusCommissionEmployee(
 9     const string &first, const string &last, const string &ssn,
10     double sales, double rate, double salary )
11  {
12     firstName = first; // should validate
13     lastName = last; // should validate
14     socialSecurityNumber = ssn; // should validate
15     setGrossSales( sales ); // validate and store gross sales
16     setCommissionRate( rate ); // validate and store commission rate
17     setBaseSalary( salary ); // validate and store base salary
18  } // end BasePlusCommissionEmployee constructor
```

```
81  // set base salary
82  void BasePlusCommissionEmployee::setBaseSalary( double salary )
83  {
84     baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
85  } // end function setBaseSalary
```

BasePlusCommissionEmployee.cpp (Incomplete)

```
87  // return base salary
88  double BasePlusCommissionEmployee::getBaseSalary() const
89  {
90     return baseSalary;
91  } // end function getBaseSalary
92
93  // calculate earnings
94  double BasePlusCommissionEmployee::earnings() const
95  {
96     return baseSalary + ( commissionRate * grossSales );
97  } // end function earnings
98
99  // print BasePlusCommissionEmployee object
100 void BasePlusCommissionEmployee::print() const
101 {
102    cout << "base-salaried commission employee: " << firstName << ' '
103       << lastName << "\nsocial security number: " << socialSecurityNumber
104       << "\ngross sales: " << grossSales
105       << "\ncommission rate: " << commissionRate
106       << "\nbase salary: " << baseSalary;
107 } // end function print
```

A Better Approach:

Designing `BasePlusCommissionEmployee` by Inheriting `CommissionEmployee`

BasePlusCommissionEmployee.h

```
 7  #include <string> // C++ standard string class
 8  #include "CommissionEmployee.h" // CommissionEmployee class declaration
 9  using namespace std;
10
11  class BasePlusCommissionEmployee : public CommissionEmployee
12  {
13  public:
14     BasePlusCommissionEmployee( const string &, const string &,
15        const string &, double = 0.0, double = 0.0, double = 0.0 );
16
17     void setBaseSalary( double ); // set base salary
18     double getBaseSalary() const; // return base salary
19
20     double earnings() const; // calculate earnings
21     void print() const; // print BasePlusCommissionEmployee object
22  private:
23     double baseSalary; // base salary
24  }; // end class BasePlusCommissionEmployee
25
26  #endif
```

BasePlusCommissionEmployee.cpp (1/2)

If we do not explicitly make this call, the default constructor of the base class (i.e., CommossionEmployee( )) will be implicitly called, causing a compilation error (since such a constructor does not exist)

CommissionEmployee.h

```
 9  class CommissionEmployee
10  {
11  public:
12     CommissionEmployee( const string &, const string &, const string &,
13        double = 0.0, double = 0.0 );
```

Compiler provides a default constructor (with no parameters) if none included

```
 3  #include <iostream>
 4  #include "BasePlusCommission
 5  using namespace std;
 6
 7  // constructor
 8  BasePlusCommissionEmployee::BasePlusCommissionEmployee(
 9     const string &first, const string &last, const string &ssn,
10     double sales, double rate, double salary )
11     // explicitly call base-class constructor
12     : CommissionEmployee( first, last, ssn, sales, rate )
13  {
14     setBaseSalary( salary ); // validate and store base salary
15  } // end BasePlusCommissionEmployee constructor
16
17  // set base salary
18  void BasePlusCommissionEmployee::setBaseSalary( double salary )
19  {
20     baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
21  } // end function setBaseSalary
22
```

BasePlusCommissionEmployee.cpp (2/2)

```
23  // return base salary
24  double BasePlusCommissionEmployee::getBaseSalary() const
25  {
26     return baseSalary;
27  } // end function getBaseSalary
28
29  // calculate earnings
30  double BasePlusCommissionEmployee::earnings() const
31  {
32     // derived class cannot access the base class's private data
33     return baseSalary + ( commissionRate * grossSales );
34  } // end function earnings
35
36  // print BasePlusCommissionEmployee object
37  void BasePlusCommissionEmployee::print() const
38  {
39     // derived class cannot access the base class's private data
40     cout << "base-salaried commission employee: " << firstName << ' '
41        << lastName << "\nsocial security number: " << socialSecurityNumber
42        << "\ngross sales: " << grossSales
43        << "\ncommission rate: " << commissionRate
44        << "\nbase salary: " << baseSalary;
45  } // end function print
```

Fix the Problem in the Better Approach:

Designing `BasePlusCommissionEmployee` by Inheriting `CommissionEmployee` with `protected` Data

CommissionEmployee.h

```
 9  class CommissionEmployee
10  {
11  public:
12     CommissionEmployee( const string &, const string &, const string &,
13        double = 0.0, double = 0.0 );
14
15     void setFirstName( const string & ); // set first name
16     string getFirstName() const; // return first name
17
18     void setLastName( const string & ); // set last name
19     string getLastName() const; // return last name
20
21     void setSocialSecurityNumber( const string & ); // set SSN
22     string getSocialSecurityNumber() const; // return SSN
23
24     void setGrossSales( double ); // set gross sales amount
25     double getGrossSales() const; // return gross sales amount
26
27     void setCommissionRate( double ); // set commission rate
28     double getCommissionRate() const; // return commission rate
29
30     double earnings() const; // calculate earnings
31     void print() const; // print CommissionEmployee object
32  protected:
33     string firstName;
34     string lastName;
35     string socialSecurityNumber;
36     double grossSales; // gross weekly sales
37     double commissionRate; // commission percentage
38  }; // end class CommissionEmployee
```

Problems When Using `protected` Members

- The derived-class object does not have to use a `set` function to set the value of the base class's `protected` data member
  - An invalid value can easily be assign to it
- The member function derived class are likely to be written depending on the base-class implementation
  - If the base-class implementation changes, all derived classes need to be modified
  - E.g., if the names of data members `firstName` and `lastName` were changed to `first` and `last` for some reason, all occurrences of these data members in derived classes must be changed as well

A Good Software-Engineering Approach:

Designing `BasePlusCommissionEmployee` by Inheriting `CommissionEmployee` with `private` Data

CommissionEmployee.h

```
 9  class CommissionEmployee
10  {
11  public:
12     CommissionEmployee( const string &, const string &, const string &,
13        double = 0.0, double = 0.0 );
14
15     void setFirstName( const string & ); // set first name
16     string getFirstName() const; // return first name
17
18     void setLastName( const string & ); // set last name
19     string getLastName() const; // return last name
20
21     void setSocialSecurityNumber( const string & ); // set SSN
22     string getSocialSecurityNumber() const; // return SSN
23
24     void setGrossSales( double ); // set gross sales amount
25     double getGrossSales() const; // return gross sales amount
26
27     void setCommissionRate( double ); // set commission rate
28     double getCommissionRate() const; // return commission rate
29
30     double earnings() const; // calculate earnings
31     void print() const; // print CommissionEmployee object
32  private:
33     string firstName;
34     string lastName;
35     string socialSecurityNumber;
36     double grossSales; // gross weekly sales
37     double commissionRate; // commission percentage
38  }; // end class CommissionEmployee
```

CommissionEmployee.cpp (Incomplete)

```
77  // calculate earnings
78  double CommissionEmployee::earnings() const
79  {
80     return getCommissionRate() * getGrossSales();
81  } // end function earnings
82
83  // print CommissionEmployee object
84  void CommissionEmployee::print() const
85  {
86     cout << "commission employee: "
87        << getFirstName() << ' ' << getLastName()
88        << "\nsocial security number: " << getSocialSecurityNumber()
89        << "\ngross sales: " << getGrossSales()
90        << "\ncommission rate: " << getCommissionRate();
91  } // end function print
```

BasePlusCommissionEmployee.cpp (1/2)

```
 3  #include <iostream>
 4  #include "BasePlusCommissionEmployee.h"
 5  using namespace std;
 6
 7  // constructor
 8  BasePlusCommissionEmployee::BasePlusCommissionEmployee(
 9     const string &first, const string &last, const string &ssn,
10     double sales, double rate, double salary )
11     // explicitly call base-class constructor
12     : CommissionEmployee( first, last, ssn, sales, rate )
13  {
14     setBaseSalary( salary ); // validate and store base salary
15  } // end BasePlusCommissionEmployee constructor
16
17  // set base salary
18  void BasePlusCommissionEmployee::setBaseSalary( double salary )
19  {
20     baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
21  } // end function setBaseSalary
22
```

BasePlusCommissionEmployee.cpp (2/2)

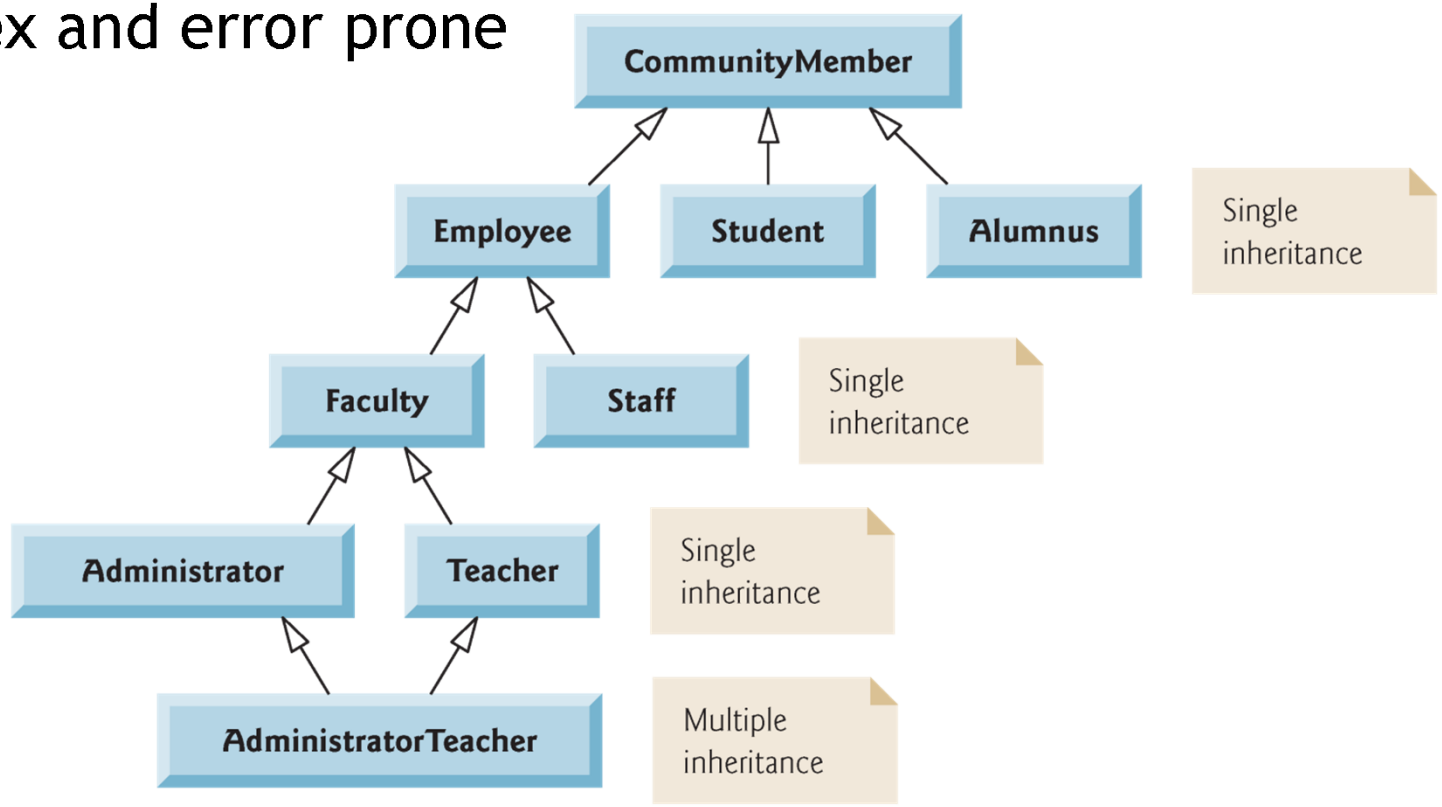
```
23  // return base salary
24  double BasePlusCommissionEmployee::getBaseSalary() const
25  {
26     return baseSalary;
27  } // end function getBaseSalary
28
29  // calculate earnings
30  double BasePlusCommissionEmployee::earnings() const
31  {
32     return getBaseSalary() + CommissionEmployee::earnings();
33  } // end function earnings
34
35  // print BasePlusCommissionEmployee object
36  void BasePlusCommissionEmployee::print() const
37  {
38     cout << "base-salaried ";
39
40     // invoke CommissionEmployee's print function
41     CommissionEmployee::print();
42
43     cout << "\nbase salary: " << getBaseSalary();
44  } // end function print
```

Outline

- Introduction to Inheritance
- Types of Inheritance
- Constructors and Destructors in Derived Classes
- A Case Study: Employees
- Multiple Inheritance

Multiple Inheritance

- C++ allows multiple Inheritance (Chapter 24)
  - Complex and error prone




```
class AdministratorTeacher : public Administrator, Teacher {
  ...
};
```