Inheritance

- The advantages of inheritance approach are characterized by the phrase "code reuse", which means, simply, that the ***functions*** that manipulate base-class object are reused in all derived-class objects.

Relationships between Classes

- Friendship.

- Composition (has-a):

  an object of one class is a member of another class.

Relationships between Classes (cont)

- Inheritance (is-a):
  1. one class is formed by **extending** the definition of another class to include additional fields or member functions.
  2. the original class is the base class, and the extended definition forms a derived class.
  3. if a derived class inherits fields from more than one class, this process is usually called multiple inheritance.

Public Inheritance - Access Control

```
class Base {
    int x;

public:
    int get_x() { return x; }
}

class Derived : public Base {
    int y;

public:
    void show_sum()      { cout << x + y << endl; }        // ERROR!
    void show_sum1()     { cout << get_x() + y << endl; }
}
```

Public Inheritance - Access Control (2)

- Derived class inherits all members of base class.
- All public members of the base class become public members of the derived class. Private members of the base remain private to it and are inaccessible by the derived class.
- Permissions: anything one can do to / with a base object, can be done with a derived object.

Public Inheritance - Access Control (3)

- The protected access specifier is equivalent to the private specifier with the sole exception that protected members of a base class are accessible to members of any class ***derived from that base***.

Public Inheritance - Access Control (4)

- Static members (data and functions) are inherited.
- "friend" is not inherited.

- You cannot derive a class from the same base class twice.

Type Conversion (Casting) Pointers

- Derived class object is a base class object.
- It is always safe to convert a derived-class object, reference, or pointer to a base-class object, reference, or pointer.
- However, cannot use base-class pointer to traverse derived-class array.

Type Conversion (Casting) Pointers (cont)

```
Base *p;
   // assume a member function print()
Derived array[10];

p = array;                    // works fine, no errors
for (i = 10; --i >= 0; )      // traverse array
   (p++)->print();
        // no error, but does not work
```

Initialization

- Base-class constructors are called first, because the members of the base-class are available to the derived class constructor.
- If there are multiple base classes, they are called in the order that base classes are listed in the class declaration.
- The destructor functions are executed in reverse order.

Basic Inheritance Example

- Illustrating the order in which constructors and destructors are called.

```
#include <iostream>

using namespace std;
```

```
class BaseClass {
    int y;

    public:

    BaseClass(int y) {
        this->y = y;
        cout << "BaseClass object created with y = " << this->y << "\n";
    }

    ~BaseClass() {
        cout << "BaseClass destructor with y = " << y << "\n";
    }
};
```

```
class DerivedClass : public BaseClass {
    int x;

    public:

    DerivedClass(int x, int y) : BaseClass(y) {
            this->x = x;
        cout << "DerivedClass object created with x = " << this->x << "\n";
    }

    ~DerivedClass() {
            cout << "DerivedClass destructor\n";
    }
};
```

```
int main() {
    cout << "Creating BaseClass object\n\n";
    BaseClass b1(10);

    cout << "\nCreating DerivedClass object\n\n";
    DerivedClass d1(5, 7);

    cout << "\nProgram ending\n";
    return 0;
}
```

Multiple Inheritance

```
class B1 { int a; }
class B2 { int b; }

class D : public B1, public B2 {
    int c;

public:
    D(int x, int y, int z) : B2(x), B1(y) { c = z; }
}
```

Virtual Base Classes

- A derived class indirectly inherits the same base class more than once, it is possible to prevent two copies of the base from being present in the derived object by having that base class inherited as "virtual" by any derived classes.

Virtual Base Classes (cont)

```
class Base { public int i; }
class Derived1 : public Base {}
class Derived2 : public Base {}
class Derived3 :
   public Derived1, Derived2 {}

class Derived1 : virtual public Base {}
```

Inheritance in Java

- Java is limited by the constraint that each class can only inherit (be derived from) one parent class - single inheritance.

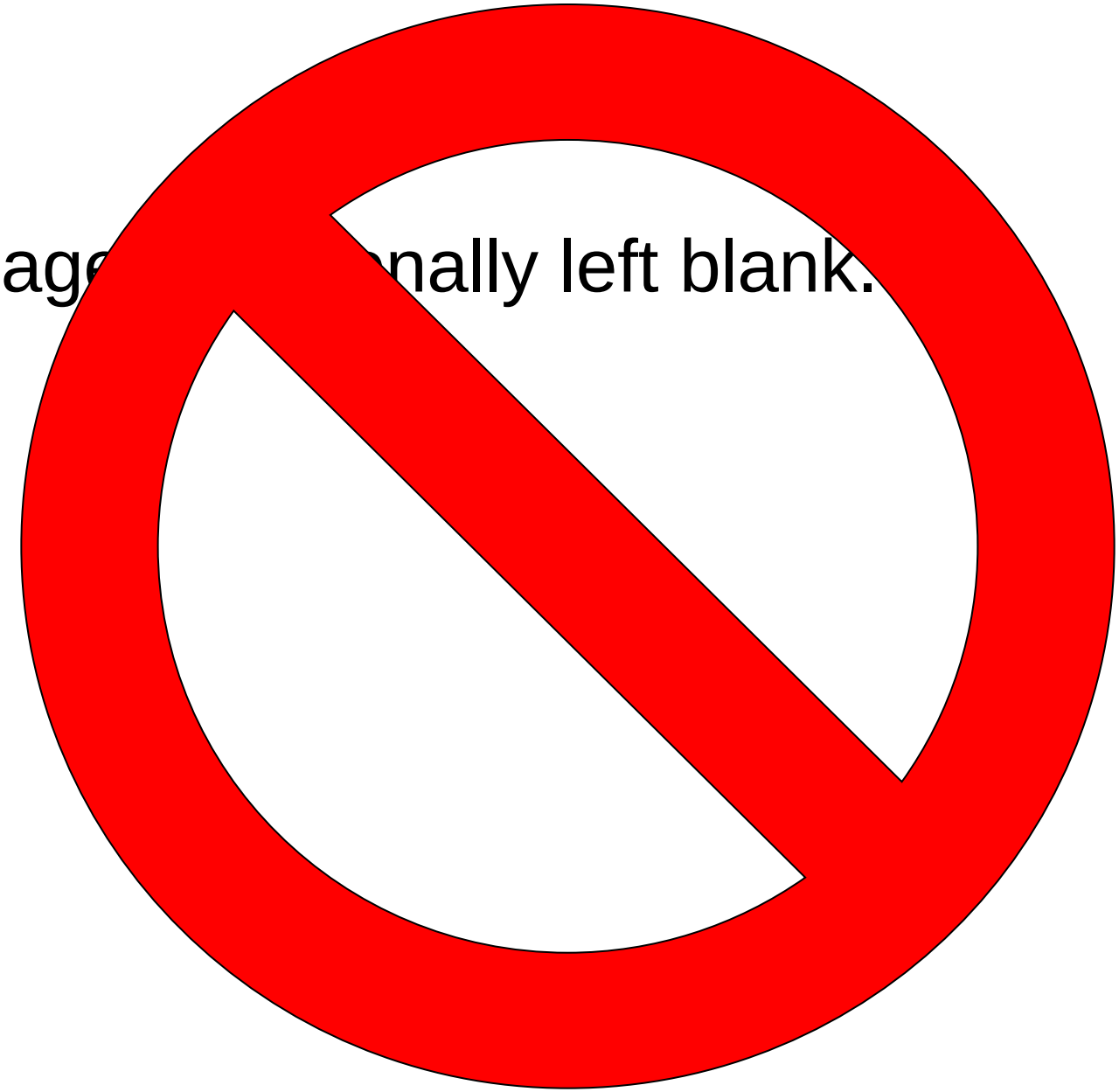
- The parent class is called the super class and represents a generalization of the derived class (or sub class).

Inheritance in Java (cont)

- The sub class is a specialization of the super class and may have extra attributes and / or methods.

- Inheritance is incremental, attributes may be added but not taken away. Methods may be added or modified. This later is called overriding.

This page intentionally left blank.



Chapter 9 - Inheritance

Outline

9.2 Inheritance: Base and Derived Classes

- Base and derived classes
  - Often an object from a derived class (subclass) is also an object of a base class (superclass)
    - A rectangle is a derived class in reference to a square
- Inheritance examples

Base class	Derived classes
`Student`	`GraduateStudent` `UndergraduateStudent`

`Shape`	`Circle`
	`Triangle`
	`Rectangle`

`Loan`	`CarLoan`
	`HomeImprovementLoan`
	`MortgageLoan`

`Employee`	`FacultyMember` `StaffMember`
`Account`	`CheckingAccount` `SavingsAccount`

9.2 Inheritance: Base and Derived Classes

- Implementation of **`public`** inheritance

```
class CommissionWorker : public
Employee {
    ...
};
```

  - Class **`CommissionWorker`** inherits from class **`Employee`**
  - **`friend`** functions not inherited
  - **`private`** members of base class not accessible from derived class

9.3 `protected` Members

- **`protected`** access
  - Intermediate level of protection between **`public`** and **`private`** inheritance
  - Derived-class members can refer to **`public`** and **`protected`** members of the base class simply by using the member names
  - Note that **`protected`** data “breaks” encapsulation

9.4 Casting Base-Class Pointers to Derived Class Pointers

- Derived classes, relationships to base classes
  - Objects of a derived class can be treated as objects of the base class
    - Reverse not true — base class objects cannot be derived-class objects
- Downcasting a pointer
  - Use an explicit cast to convert a base-class pointer to a derived-class pointer
  - Format:

```
derivedPtr = static_cast< DerivedClass * > basePtr;
```

9.5 Using Member Functions

- Derived class member functions
  - Cannot directly access **`private`** members of their base class
    - Maintains encapsulation

9.6 Overriding Base-Class Members in a Derived Class

- To override a base-class member function
  - In the derived class, supply a new version of that function with the same signature
    - same function name, different definition
  - When the function is then mentioned by name in the derived class, the derived version is automatically called
  - The scope-resolution operator may be used to access the base class version from the derived class

9.7 public, private, and protected Inheritance

Base class member access specifier	Type of inheritance		
	`public` inheritance	**`protected`** inheritance	**`private`** inheritance
`Public`	**`public`** in derived class. Can be accessed directly by any non-**`static`** member functions, **`friend`** functions and non-member functions.	**`protected`** in derived class. Can be accessed directly by all non-**`static`** member functions and **`friend`** functions.	**`private`** in derived class. Can be accessed directly by all non-**`static`** member functions and **`friend`** functions.
`Protected`	**`protected`** in derived class. Can be accessed directly by all non-**`static`** member functions and **`friend`** functions.	**`protected`** in derived class. Can be accessed directly by all non-**`static`** member functions and **`friend`** functions.	**`private`** in derived class. Can be accessed directly by all non-**`static`** member functions and **`friend`** functions.
`Private`	Hidden in derived class. Can be accessed by non-**`static`** member functions and **`friend`** functions through **`public`** or **`protected`** member functions of the base class.	Hidden in derived class. Can be accessed by non-**`static`** member functions and **`friend`** functions through **`public`** or **`protected`** member functions of the base class.	Hidden in derived class. Can be accessed by non-**`static`** member functions and **`friend`** functions through **`public`** or **`protected`** member functions of the base class.

9.9 Using Constructors and Destructors in Derived Classes

- Base class initializer
  - Uses member-initializer syntax
  - Can be provided in the derived class constructor to call the base-class constructor explicitly
    - Otherwise base class's default constructor called implicitly
  - Base-class constructors and base-class assignment operators are not inherited by derived classes
    - Derived-class constructors and assignment operators, however, can call base-class

9.9 Using Constructors and Destructors in Derived Classes

- A derived-class constructor
  - Calls the constructor for its base class first to initialize its base-class members
  - If the derived-class constructor is omitted, its default constructor calls the base-class' default constructor
- Destructors are called in the reverse order of constructor calls
  - So a derived-class destructor is called before its base-class destructor

9.10 Implicit Derived-Class Object to Base-Class Object Conversion

- Assignment of derived and base classes
  - Derived-class type and base-class type are different
  - Derived-class object can be treated as a base-class object
    - Derived class has members corresponding to all of the base class's members
    - Derived-class has more members than the base-class object
    - Base-class can be assigned a derived-class
  - Base-class object cannot be treated as a derived-class object

9.10 Implicit Derived-Class Object to Base-Class Object Conversion

- Mixing base and derived class pointers and objects
  - Referring to a base-class object with a base-class pointer
    - Allowed
  - Referring to a derived-class object with a derived-class pointer
    - Allowed
  - Referring to a derived-class object with a base-class pointer
    - Possible syntax error
    - Code can only refer to base-class members, or syntax error

9.12 Composition vs. Inheritance

- "Is a" relationships
  - Inheritance
    - Relationship in which a class is derived from another class
- "Has a" relationships
  - Composition
    - Relationship in which a class contains other classes as members

9.15 Multiple Inheritance

- Multiple Inheritance
  - Derived-class inherits from multiple base-classes
  - Encourages software reuse, but can create ambiguities