Inheritance
- Public
- Protected
- Private

Constructors under Inheritance

Destructors under Inheritance

Q&A
Inheritance

- Software reusability
- Create new class from existing class instead of building it entirely from the scratch
  - Existing class’s data and behaviors
  - Adding new capabilities
  - Derived class inherits from base class
Class hierarchy

- Direct base class
  - Inherited explicitly (one level up hierarchy)

- Indirect base class
  - Inherited two or more levels up hierarchy

- Single inheritance
  - Inherits from one base class

- Multiple inheritance
  - Inherits from multiple base classes
    - Base classes possibly unrelated
Three Types of Inheritance

**public**

- Every object of derived class is also an object of base class
  - Base-class objects are not objects of derived classes
    - All cars are vehicles, but not all vehicles are cars
- Can access non-private members of base class
  - To access private base-class members
    - Derived class must use inherited non-private member functions

**private (later)**

- Alternative to composition

**protected (later)**

- Rarely used
“is-a” vs “has-a”

“is-a”
- Inheritance
- Derived class object can be treated as base class object
  - Car is a vehicle
    - Vehicle properties/behaviors also apply to a car

“has-a”
- Composition
- Object contains one or more objects of other classes as members
  - Car has a steering wheel
Base Classes and Derived Classes

Object of one class “is an” object of another class

- Rectangle is quadrilateral
  - Class Rectangle inherits from class Quadrilateral
    - base class: quadrilateral
    - derived class: rectangle

Base class typically represents larger set of objects than derived classes

- Base class: Vehicle
- Derived class: Car
Base Classes and Derived Classes Cont’d

- GraduatedStudent class is derived from Student class
- GraduatedStudent class is inherited from Student class
- Student class is super class of GraduatedStudent
- Graduated class is child class or subclass of Student class
Public Inheritance

class TwoDimensionalShape : public Shape

- Class TwoDimensionalShape inherits from class Shape

Base class private members
- Not accessible directly, but still inherited
- Accessed through inherited public member functions

Base class public and protected members
- Inherited with original member access

friend functions
- Not inherited

```cpp
class BaseClass {
    // ...
};

class DerivedClass : public BaseClass {
    // ...
};
```

class BaseClass {
    public:
        void public_method();
    protected:
        void protected_method();
    private:
        void private_method();
};

class DerivedClass : public BaseClass {
    public:
        void public_method();
    protected:
        void protected_method();
};
Protected Access Specifier

- Intermediate level of protection between public and private
- Protected methods/data cannot be accessible by other classes except for subclasses
- Other classes consider protected members as normal “private” members
- Subclasses consider protected members as normal “public” members

Protected Access Specifier Cont’d

- Protected members are accessible to:
  - Base class members
  - Base class friends
  - Derived class members
  - Derived class friends
Protected Access Specifier Cont’d

- Public members in base class is public in derived class
- Protected members in base class are protected in derived class
- Derived-class members
  - Refer to public and protected members of base class
    - Simply use member names
  - Redefined base class members can be accessed by using base-class name and binary scope resolution operator (::)
Protected Inheritance

public and protected members in base class become protected in derived class

class BaseClass {
    public:
        void public_method();
    protected:
        void protected_method();
    private:
        void private_method();
};
class DerivedClass : protected BaseClass {
    protected:
        void public_method();
    protected:
        void protected_method();
};

Private Inheritance

class BaseClass {
    public:
        void public_method();
    protected:
        void protected_method();
    private:
        void private_method();
};

class DerivedClass : private BaseClass {
    private:
        void public_method();
    private:
        void protected_method();
};

Protected Data Members

- **Advantages**
  - Derived class can modify values directly
    - No set/get method call overhead

- **Disadvantages**
  - No validity checking
    - Derived class can assign invalid value
  - Implementation dependent
    - Derived class more likely dependent on base class implementation
    - Base class implementation may result in derived class’s modification
    - Fragile software
Class CommissionEmployee

```c++
#ifndef COMMISSION_H
#define COMMISSION_H

#include <string>
using std::string;

class CommissionEmployee
{
public:
    CommissionEmployee( const string &, const string &, const string &, double = 0.0, double = 0.0 );

    void setFirstName( const string & );
    string getFirstName() const;

    void setLastName( const string & );
    string getLastName() const;

    void setSocialSecurityNumber( const string & );
    string getSocialSecurityNumber() const;

    void setGrossSales( double );
    double getGrossSales() const;
    void setCommissionRate( double );
    double getCommissionRate() const;
    double earnings() const;
    void print() const;

protected:
    string firstName;
    string lastName;
    string socialSecurityNumber;
    double grossSales;
    double commissionRate;

};

#endif
```

```cpp
#ifndef BASEPLUS_H
#define BASEPLUS_H

#include <string> // C++ standard string class
using std::string;

#include "CommissionEmployee.h"

class BasePlusCommissionEmployee : public CommissionEmployee
{
public:
    BasePlusCommissionEmployee( const string &, const string &, const string &, double = 0.0, double = 0.0, double = 0.0 );
    void setBaseSalary( double );
    double getBaseSalary() const;
    double earnings() const;
    void print() const;

private:
    double baseSalary;

};

double BasePlusCommissionEmployee::earnings() const
{
    return baseSalary +
        ( commissionRate * grossSales );
}
#endif
```

The Best Software Engineering Practice

- Declare data members as private
  - Enables programmers to change the base-class implementation without having to change derived-class implementations
  - Use the protected access specifier when a base class should provide a service (i.e., a member function) only to its derived classes (and friends), not to other clients

- Provide public get and set functions
- Use get method to obtain values of data members

Set/get method slightly slower than direct access

- But today’s optimizing compiler inlines set/get methods
- Or you can explicitly specify “inline” keyword

```cpp
class BaseClass
{
public:
  inline int getx() const { return x; }
  inline void setx(int v)
  {
    if (v > 100) error();
    else x = v;
  }
private:
  int x;
}
```

Selection of public/protected/private Methods

According to the service range

- for all other classes: public
- for itself and subclasses: protected
- only for itself: private
Constructors under Inheritance

- Constructor in base class
  - Does not construct derived class specific parts

- Constructor in derived class
  - Initialize its own data members
  - Invokes the constructor of the base class
    - Implicitly or explicitly
Constructors under Inheritance Cont’d

- Base of inheritance hierarchy
  - Last constructor called in chain
  - First constructor body to finish executing
  - `CommissionEmployee/BasePlusCommissionEmployee` hierarchy
    - `CommissionEmployee` constructor called last
    - `CommissionEmployee` constructor body finishes execution first

- Initializing data members
  - Each base-class constructor initializes its data members that are inherited by derived class
Constructors under Inheritance
Cont’d

class BaseClass
{
    public:
        BaseClass() { x = 1; }  
        BaseClass( int a ) { x = a; }

    private:
        int x;
};

class DerivedClass : public BaseClass
{
    public:
        DerivedClass() { y = 2; }
        DerivedClass( int x ) : BaseClass( x ) { y = 2; }

    private:
        int y;
};

Constructors under Inheritance

Cont’d

class BaseClass
{
    public:
        // BaseClass() { x = 1; }
        BaseClass( int a ) { x = a; }

    private:
        int x;
};
class DerivedClass : public BaseClass
{
    public:
        DerivedClass() { y = 2; }
        // error

    private:
        int y;
};

class BaseClass
{
    public:
        BaseClass() {
            cout << "base";
            cout << endl;
        }
};
class DerivedClass : public BaseClass
{
    public:
        DerivedClass() {
            cout << "derived";
            cout << endl;
        }
};

Destructors under Inheritance

- **Destroying derived-class object**
  - Chain of destructor calls
    - Reverse order of constructor chain
    - Destructor of derived-class called first
    - Destructor of next base class up hierarchy next
      - Continue up hierarchy until final base reached
      - After final base-class destructor, object removed from memory

Destructors under Inheritance Cont’d

class BaseClass
{
    public:
        BaseClass() { x = new int[100]; }
        ~BaseClass() { delete[] x; }
    private:
        int* x;
};

class DerivedClass : public BaseClass
{
    public:
        DerivedClass() { y = new int[10]; }
        ~DerivedClass() { delete[] y; }
    private:
        int* y;
};

# Base-class Member Accessibility in a Derived Class

<table>
<thead>
<tr>
<th>Base-class member-access specifier</th>
<th>Type of inheritance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>public inheritance</td>
</tr>
<tr>
<td><strong>public</strong></td>
<td>public in derived class. Can be accessed directly by member functions, friend functions and nonmember functions.</td>
</tr>
<tr>
<td><strong>protected</strong></td>
<td>protected in derived class. Can be accessed directly by member functions and friend functions.</td>
</tr>
<tr>
<td><strong>private</strong></td>
<td>Hidden in derived class. Can be accessed by member functions and friend functions through public or protected member functions of the base class.</td>
</tr>
</tbody>
</table>
Customizing and Reusing Existing Software

- Derived class can
  - re-implement the behaviors of the base class
    - Known as “method overriding”
  - add new behaviors or data
    - We can use not only new behaviors but also the existing behaviors

- Factor out common attributes and behaviors and place these in a base class

- Use inheritance to form derived classes, endowing them with capabilities beyond those inherited from the base class

- The creation of a derived class does not affect its base class’s source code