

Inheritance

Learning Objectives

- Inheritance Basics
 - Derived classes, with constructors
 - protected: qualifier
 - Redefining member functions
 - Non-inherited functions
- Programming with Inheritance
 - Destructors in derived classes
 - Multiple inheritance

Introduction to Inheritance

- Object-oriented programming
 - Powerful programming technique
 - Provides abstraction dimension called *inheritance*
- General form of class is defined
 - Specialized versions then inherit properties of general class
 - And add to it/modify it's functionality for it's appropriate use

Inheritance Basics

- New class inherited from another class
- Base class
 - "General" class from which others derive
- Derived class
 - New class
 - Automatically has base class's:
 - Member variables
 - Member functions
 - Can then add additional member functions and variables

Derived Classes

- Consider example:
Class of "Book"
- Composed of:
 - Romance Books
 - Technical Books
 - Paper back Books
- Each is "subset" of Book
 - Another might be those , Ebooks , Hardback

Derived Classes

- Don't "need" type of generic "Book"
 - Since not any item's just a "Book"
- General concept of Book helpful!
 - All have names
 - All have ISBN numbers
 - Associated functions for these "basics" are same among all books
- So "general" class can contain all these "things" about books

Book.cpp

```
#include <iostream>
#include <string>
using namespace std;
class Book
{
public:
    virtual string getDescription() { return "Book"; }
    string issbn;
};

class Paperback : public Book
{
public:
    virtual string getDescription() {
        return "Paperback " + Book::getDescription();
    }
};

class Romance : public Paperback
{
public:
    virtual string getDescription() {
        return "Romance " + Paperback::getDescription();
    }
};

class Technical : public Book
{
public:
    virtual string getDescription() {
        return "Technical " + Book::getDescription();
    }
};
```

```
int main()
{
    Romance novel;
    Book book;

    // outputs "Romance Paperback Book"
    cout << novel.getDescription() << endl;

    // outputs "Book"
    cout << book.getDescription() << endl;}
}
```

Book Class

- Many members of "Book" class apply to all types of books
 - Accessor functions
 - Mutator functions
 - Most data items:
 - ISBN
 - Name
- We won't have "objects" of this class, however

Book Class

- Consider getDescription() function:
 - Will always be "redefined" in derived classes
 - So different book types can have different descriptions
 - Makes no sense really for "undifferentiated" book
 - So function getDescription () in Book class says just "Book"

Deriving from Book Class

- Derived classes from Book class:
 - Automatically have all member variables
 - Automatically have all member functions
- Derived class said to "inherit" members from base class
- Can then redefine existing members and/or add new members

Technical Class Interface

- Note definition begins same as any other
 - the heading:
class Technical : public Book
{ ...
 - Specifies "publicly inherited" from Book
class

Technical Class Additions

- Derived class interface only lists new or "to be redefined" members
 - Since all others inherited are already defined
- Technical adds:
 - Constructors

Technical Class Redefinitions

- Technical redefines:
 - getDescription () member function
 - This "overrides" the getDescription() function implementation from Book class
- It's definition must be in Technical class's implementation
 - As do other member functions declared in Technical's interface
 - New and "to be redefined"

Inheritance Terminology

- Common to simulate family relationships
- Parent class
 - Refers to base class
- Child class
 - Refers to derived class
- Ancestor class
 - Class that's a parent of a parent ...
- Descendant class
 - Opposite of ancestor

Constructors in Derived Classes

- Base class constructors are NOT inherited in derived classes!
 - But they can be invoked within derived class constructor
 - Which is all we need!
- Base class constructor must initialize all base class member variables
 - Those inherited by derived class
 - So derived class constructor simply calls it
 - "First" thing derived class constructor does

The protected: Qualifier

- New classification of class members
- Allows access "by name" in derived class
 - But nowhere else
 - Still no access "by name" in other classes
- In class it's defined → acts like private
- Considered "protected" in derived class
 - To allow future derivations
- Many feel this "violates" information hiding

Redefinition of Member Functions

- Recall interface of derived class:
 - Contains declarations for new member functions
 - Also contains declarations for inherited member functions to be changed
 - Inherited member functions NOT declared:
 - Automatically inherited unchanged
- Implementation of derived class will:
 - Define new member functions
 - Redefine inherited functions as declared

Redefining vs. Overloading

- Very different!
- Redefining in derived class:
 - SAME parameter list
 - Essentially "re-writes" same function
- Overloading:
 - Different parameter list
 - Defined "new" function that takes different parameters
 - Overloaded functions must have different signatures

A Function's Signature

- Recall definition of a "signature":
 - Function's name
 - Sequence of types in parameter list
 - Including order, number, types
- Signature does NOT include:
 - Return type
 - const keyword
 - &

Accessing Redefined Base Function

- When redefined in derived class, base class's definition not "lost"
- Can specify it's use:
Book ordinary;
Technical medical;
ordinary.getdescription(); → calls Book's
getdescription function
medical.getdescriptionk(); → calls Techinal
getdescription function
medial.Book::getdescription(); → Calls Book's
description function!
- Not typical here, but useful sometimes

Functions Not Inherited

- All "normal" functions in base class are inherited in derived class
- Exceptions:
 - Constructors (we've seen)
 - Destructors
 - Copy constructor
 - But if not defined, generates "default" one
 - Recall need to define one for pointers!
 - Assignment operator
 - If not defined → default

Destructors in Derived Classes

- If base class destructor functions correctly
 - Easy to write derived class destructor
- When derived class destructor is invoked:
 - Automatically calls base class destructor!
 - So no need for explicit call
- So derived class destructors need only be concerned with derived class variables
 - And any data they "point" to
 - Base class destructor handles inherited data automatically

Destructor Calling Order

- Consider:
class B derives from class A
class C derives from class B
 $A \leftarrow B \leftarrow C$
- When object of class C goes out of scope:
 - Class C destructor called 1st
 - Then class B destructor called
 - Finally class A destructor is called
- Opposite of how constructors are called

"Is a" vs. "Has a" Relationships

- Inheritance
 - Considered an "Is a" class relationship
 - e.g., A Technical book "is a" Book
 - A Convertible "is a" Automobile
- A class contains objects of another class as it's member data
 - Considered a "Has a" class relationship
 - e.g., One class "has a" object of another class as it's data

Protected and Private Inheritance

- New inheritance "forms"
 - Both are rarely used
- Protected inheritance:
class SalariedEmployee : protected Employee
{...}
 - Public members in base class become protected in derived class
- Private inheritance:
class SalariedEmployee : private Employee
{...}
 - All members in base class become private in derived class

Multiple Inheritance

- Derived class can have more than one base class!
 - Syntax just includes all base classes separated by commas:
class derivedMulti : public base1, base2
{...}
- Possibilities for ambiguity are endless!
- Dangerous undertaking!
 - Some believe should never be used
 - Certainly should only be used by experienced programmers!