UCLA

CS 31:
Introduction To Computer Science I

Howard A. Stahl

UCLA

Agenda

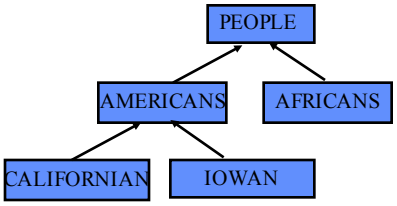
- Inheritance
- `protected` Qualifier
- Virtual Functions

Inheritance

- Often, Classes Are Made From Existing Classes
- Base Class
  - starting point for defining a set of classes
  - most general attributes and methods defined here
- Derived Class
  - extends the definition of a base class in some way

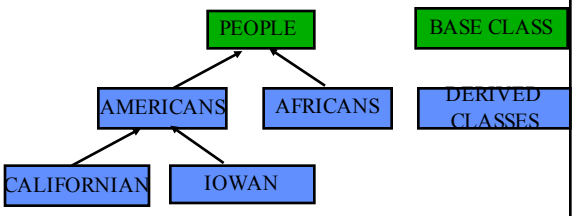
Hierarchy

- A Very Natural Notion



Hierarchy

- A Very Natural Notion



Introduction to Inheritance

- Object-Oriented Programming
  - Powerful Programming Technique
  - Provides Abstraction Dimension Called *Inheritance*
- General Form Of Class Is Defined First
  - Specialized Versions Then Inherit Properties Of The General Class
  - And Add/Modify Functionality As Needed

"Is-A" vs. "Has-A" Relationships

- Inheritance
  - Considered An "Is-A" Class Relationship
    - An HourlyEmployee "Is A Kind Of" Employee
    - A Convertible "Is A Kind Of" Automobile
- Aggregation
  - Considered A "Has-A" Class Relationship
  - One Class Has Data Of Another Class Type
    - Automobile "Has A" Steering Wheel, Engine, Tailpipe

Inheritance Terminology

- Simulates Family Relationships
- Parent Class
  - Refers To Base Class
- Child Class
  - Refers To Derived Class
- Ancestor Class
  - Parent And Their Parents…
- Descendant Class
  - Children And Their Children…

Inheritance Basics

- New Class Inherited From Another Class
- Base Class
  - "General" Class From Which Others Derive
- Derived Class
  - New Class
  - Automatically Has Its Base Class's:
    - Member variables
    - Methods
  - Additional Methods And Members Can Be Added

Derived Classes

- Consider The Concept Of: "Employees"
  - All Have Names And Social Security Numbers
- Composed Of:
  - Salaried Employees
  - Hourly Employees
- Each Is A "Subset" Of Employees
  - Another Might Be Those Paid A Fixed Wage Per Week Or Month…

Introducing The Employee Class

- Many Members Of The "Employee" Class Apply To All Types Of Employees
  - Data Elements
    - SSN
    - Name
    - Pay
  - And Their Associated Accessors And Mutators

Textbook Example

Display 14.3 **Interface for the Derived Class** `HourlyEmployee`

```
1
2  //This is the header file hourlyemployee.h.
3  //This is the interface for the class HourlyEmployee.
4  #ifndef HOURLYEMPLOYEE_H
5  #define HOURLYEMPLOYEE_H

6  #include <string>
7  #include "employee.h"

8  using std::string;

9  namespace SavitchEmployees
10 {
```

Textbook Example

```
11      class HourlyEmployee : public Employee
12      {
13      public:
14          HourlyEmployee( );
15          HourlyEmployee(string theName, string theSsn,
16                                  double theWageRate, double theHours);
17          void setRate(double newWageRate);
18          double getRate( ) const;
19          void setHours(double hoursWorked);
20          double getHours( ) const;
21          void printCheck( ) ;
22      private:
23          double wageRate;
24          double hours;
25      };

26  }//SavitchEmployees

27  #endif //HOURLYEMPLOYEE_H
```

You only list the declaration of an inherited member function if you want to change the definition of the function.

HourlyEmployee Class Additions

- An "Additive Model" – Don't Repeat What's Already There!
- HourlyEmployee adds:
  - Constructors
  - wageRate, hours member variables
  - setRate(), getRate(), setHours(), getHours() Methods

Derived Class Constructor Example

- HourlyEmployee Constructor:

```
HourlyEmployee::HourlyEmployee(string  theName,
                        string theNumber, double theWageRate,
                        double theHours)
                : Employee(theName, theNumber),
                wageRate(theWageRate), hours(theHours)
{
     //Deliberately empty
}
```

- The Portion After : Is An "initialization section"
  - Invokes The Parent Class Construtor Which In This Case Is Employee

Another HourlyEmployee Constructor

- Another HourlyEmployee Constructor:

```
HourlyEmployee::HourlyEmployee()
        : Employee(), wageRate(0), hours(0)
{
    //Deliberately empty
}
```

- Default, No Argument Parent Class Constructor Is Called
- You Should ALWAYS Invoke A Base Constructor
  - Lacking Any Call, C++ Calls The Base Default Constructor So It Better Have One Available!

Pitfall: Private Access Modifier

- Derived Class "Inherits" Private Member Variables And Private Methods
  - But Cannot Directly Access Them
  - `private` Really Is Private!

`protected`: Access Modifier

- It Is `private` To Driver Code And Other Clases
- It Is `public` To You And All Your Derived Classes
- Allows You To Plan Ahead For Inheritance Purposes…

Doors Example

- Let's Design A Set Of Doors Classes For An Adventure Game
- What Are The Common Characteristics Of All Doors?

Door Object

- Knows:
  - Its Status (Open or Shut)
- Can Do:
  - Initialize Itself As Shut
  - Open Itself, If Possible
  - Close Itself
  - Tell Whether Or Not It Is Open

Class Door

- A Generic Base Class

```
class Door {
public:
  Door();
  bool isOpen()  const;
  void open();
  void close();
protected:
  bool isShut;
}
```

`protected` Qualifier

- `protected` Is A Compromise Between Private And Public
  - `protected` Is Public To Base Classes
  - `protected` Is Public To Friends Of The Base Classes
  - `protected` Is Public To Derived Classes
  - `protected` Is Public To Friends Of Derived Classes
  - `protected` Is Private To Other Classes

Time For Our First Demo!

- Door.cpp

(See Handout For Example 1)

Summarizing Our First Demo!

- Inheritance Is An Important Part Of Good OO Design And Implementation

Derived Lockable Door From Door

- A Derived Class

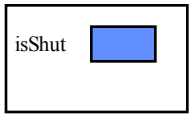
```
class LockableDoor  : public Door {
public:
  LockableDoor();
  bool isLocked()  const;
  void open();
  void lock();   void unlock();
protected:
  bool thelock;
}
```

Lockable Door Object

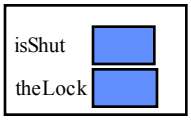
- Member Data
  - `isShut`
  - `theLock`
- Member Functions
  - `isLocked()`
  - `open()`
  - `lock()`
  - `unlock()`
  - `isOpen()`
  - `close()`

Comparing Door And LockableDoor

- Door



- LockableDoor



Inheritance Behavior

- By Default, All Member Functions And Member Attributes Are Inherited Down To Derived Classes
  - happens without mentioning these functions and attributes in the derived class definition
- Any Member Function Or Member Attribute Can Be Redefined In The Derived Class
  - hides access to the base class versions

Example Redefinition

- LockableDoor's open( ) Function

```
void LockableDoor::open(  )
{
  if (!isLocked())  {
    Door::open();
  }
}
```

Using The Doors

```
Door hallDoor;
LockableDoor  frontDoor;

hallDoor.open();
frontDoor.lock();
frontDoor.open();
if (!frontDoor.isOpen())
   frontDoor.unlock();
```

Time For Our Next Demo!

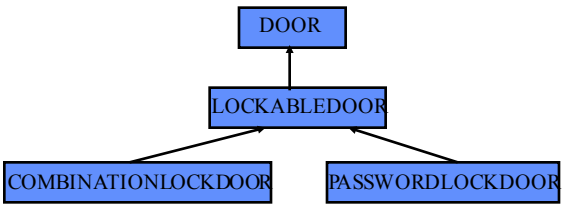
- LockableDoor.cpp

(See Handout For Example 2)

Summarizing Our Next Demo!

- `protected` Members Are Accessible To Derived Classes
- Using The Scope Operator ::, You Can Specify Just Which Version Of A Function To Call

Derived Classes Can Be A Base Class



CombinationLockDoor

- Combinations Are Single Integers

```
class CombinationLockDoor  : public
LockableDoor  {
public:
  CombinationLockDoor(  int combo  = 0);
  void unlock(  int combo  );
protected:
  int thecombination;
}
```

CombinationLockDoor Object

- Member Data
  - `isShut`
  - `thelock`
  - `thecombination`
- Member Functions
  - `isLocked()`
  - `open()`
  - `lock()`
  - `unlock( int )`
  - `isOpen()`
  - `close()`

Time For Our Next Demo!

- CombinationLockDoor.cpp

(See Handout For Example 3)

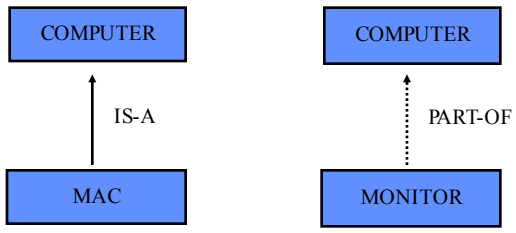
Summarizing Our Next Demo!

- `protected` Members Are Accessible To Derived Classes
- Redefined Members Hide Access To The Parent Class Versions

Relationships Between Object

- IS-A
  - one class "is a kind of" another class
  - base class is a general class
  - derived class is a specialization of the general concept
- PART-OF
  - one class "is a part of" another class
  - often used to represent compound objects

Relationships Between Objects



Relationships Between Objects

```
class Computer {
}

class Mac:public
          Computer {
}
```

IS-A

```
class Monitor {
}

class Computer {
private:
  Monitor theMonitor;
}
```

PART-OF

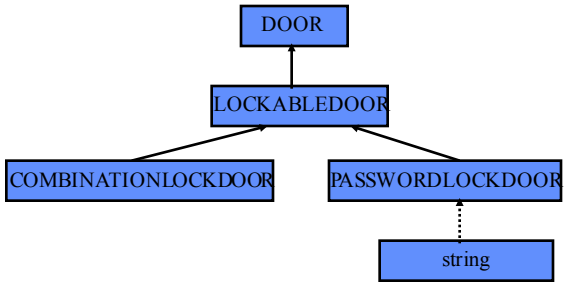
PasswordLockDoor

- Doors That Require A Password To Open Or Close
  - "open sesame" "sounds like fun"
- Let's Represent The Password As `string`
- The Password Is "Part-Of" A PasswordLockDoor

PasswordLockDoor Object

```
class PasswordLockDoor : public
LockableDoor {
public:
  PasswordLockDoor(const char c[]="");
  void unlock( const char c[]="");
protected:
  string thepassword;
}
```

Object Diagram For Doors



PasswordLockDoor Object

- Member Data
  - `isShut`
  - `thelock`
  - `thepassword`
- Member Functions
  - `isLocked()`
  - `open()`
  - `lock()`
  - `unlock(char[])`
  - `isOpen()`
  - `close()`

Pointers To Base Classes

- Pointers Can Be Made To Point To Derived Classes

```
typedef  Door*  DoorPtr;
DoorPtr  p = new  Door();
p->open();   // calls  Door::open
...
p = new  LockableDoor();
p->open();   // which  open??
```

`virtual` Functions

- Late Binding Allows The Selection Of Which Implementation Of A Member Function To Execute To Be Determined At Run-Time
- C++ Performs Late Binding Via `virtual` Functions

`virtual` Functions

- A Generic Base Class

```
class Door {
public:
  Door();
  bool isOpen() const;
  virtual void open();
  void close();
protected:
  bool isShut;
}
```

Time For Our Next Demo!

- VirtualFunctions.cpp

(See Handout For Example 4)

Summarizing Our Next Demo!

- `virtual` Functions Allow For Late Binding To Runtime Objects To Determine Which Version Of A Function Actually Gets Called

Summary

- Inheritance
- `protected` Qualifier
- Virtual Functions