Inheritance

CS 310
Inheritance

• In object-oriented languages, classes can be organized into a hierarchical structure based on the concept of *inheritance*

• Inheritance: property that instances of a child class (*subclass*) can access both the data and behavior (methods) associated with the parent class (*superclass*)
Examples

- Car is a subclass of Vehicle
- Florist is a subclass of Shopkeeper
- EditorWindow is a subclass of Window
- Window is a subclass of GraphicalObject

Inheritance should be used when two classes exhibit an “is-a” relationship
Advantages of Inheritance

• Software Reusability
• Code Sharing
• Rapid Prototyping
• Software Components
• Polymorphism & Frameworks
Forms of Inheritance

• Specialization
  – A car is a vehicle

• Extension
  – An Extended Queue is a Queue with extra features

• Construction (Implementation Inheritance)
  – A Polynomial is implemented in terms of an Extended Queue
Forms of Inheritance cont’d

• Specification
  – inheritance used for *abstract classes*
  – Abstract class Shape has subclasses Rectangle and Circle

• Other forms: Limitation, Generalization, Variation do not meet “is a” relationship

• Multiple Inheritance
Inheritance in C++: Terminology

- A **client** is a program or module that uses a class.
- In addition to **public** and **private** members, a class can have **protected** members.
- Protected members are hidden from clients of a class but are available to:
  - its own member functions (and friends)
  - member functions (and friends) of a derived class
Inheritance in C++: Terminology

• Membership categories
  – public members can be used by anyone
  – private members can be used only by member functions and friends of the class
  – protected members can be used only by member functions and friends of both the class and any derived class
Kinds of Inheritance in C++

- **Public** inheritance: public and protected members of base class remain public and protected members of derived class
- **Protected** inheritance: public and protected members of base class are protected members of the derived class
- **Private** inheritance: public and protected members of base class are private members of derived class

- Remember: *Private members of base class cannot be accessed by derived classes*
Inheritance in C++

class derived_class: kind base_class {
}

where kind is either public, protected, or private
Inheritance in C++

• Derived classes

```cpp
class derived_class_name: base_class_name {
};
```

```cpp
class derived_class_name: public base_class_name {
};
```

keyword public makes methods of base class available to clients of new class
default: if keyword public is left out, private inheritance
When to use a specific kind of inheritance

- Public: extension, specialization, specification
- Private: construction (implementation inheritance)
class sphereClass
{
public:
    // constructors
    sphereClass();
sphereClass(double Initial Radius);

    // sphere operations
    void SetRadius (double NewRadius);
    double Radius () const;
    double Diameter () const;
    double Circumference () const;
    double Area () const;
    double Volume () const;
    double DisplayStatistics () const;
private:
    double TheRadius;  // the sphere's radius
};
• We can define a new class `ballClass` which inherits all the members of `sphereClass` except for the constructors and destructors.

  • `sphereClass` is called the *base* class and `ballClass` is the *derived* class.

We can also

• add a new data member (name for the ball)
• add new member functions to manipulate the name and radius
• revise the `DisplayStatistics` routine to show the ball's name in addition to the sphere's statistics
const int MAX_STRING = 15;
class ballClass: public sphereClass
{
public:
  // constructors
  ballClass();
  ballClass(double Initial Radius, const char InitialName[]);
  
  // additional operations
  void GetName (char CurrentName[]) const;
    // get name of ball
  void SetName (char NewName[]) const;
    // alter name of existing ball
  void ResetBall (double NewRadius, const char NewName[]);
    // alters radius and name of existing ball
  double DisplayStatistics () const;
    // displays statistics of a ball

private:
  char TheName[MAX_STRING+1];  // the ball's name
};
• Can add as many new members to a derived class as you like

• Cannot revise an ancestor's private data members and should not reuse their names

• But you can redefine other ancestor members.
• ballClass has two data members:
  • TheRadius (inherited) and
  • TheName
• Since TheRadius of sphereClass is private, it can only be referenced within ballClass by using sphereClass's public member functions: SetRadius and Radius
• What does the implementation for the new members look like?
ballClass::ballClass () : sphereClass()
{ SetName(""); }                 // default constructor

ballClass::ballClass(double Initial Radius,
    const char InitialName[])
    : sphereClass(InitialRadius)
{ SetName(InitialName); }

void ballClass::GetName (char CurrentName[]) const
{ strcpy(CurrentName, TheName);}  // get name of ball

void ballClass::SetName (char NewName[]) const
{ strcpy(NewName, TheName);}   // alter name of existing
// ball

void ballClass::ResetBall (double NewRadius,
    const char NewName[])
{ SetRadius(NewRadius);       
    SetName(NewName); } // alters radius and name of
// existing ball
double ballClass::DisplayStatistics () const
{
    cout << "Statistics for a " << TheName << ":";
    sphereClass::DisplayStatistics();
}                        // displays statistics of a ball
• The constructors (destructor) for `ballClass` invoke the corresponding constructors (destructor) of `sphereClass`

  • Constructor initializer list used to call the base class constructor

    ```cpp
    derived_class_name::derived_class_name(arglist)
      : base_class_name(arglist2) { }
    ```

• Can use the member functions that `BallClass` inherits from `sphereClass`; e.g. see `ResetBall`

• Objects of a derived class can invoke the public members of the base class:

  • Example: `ballClass Ball(5.0, "Volleyball")`;

  • This means `Ball.Diameter()` returns Ball's diameter (10.0) using the member function `Diameter` that is inherited from `sphereClass`
• If **Sphere** is an instance of **sphereClass** and **Ball** is an instance of **ballClass**, then
  
  • **Sphere.DisplayStatistics** will invoke **Displaystatistics** from **sphereClass**
  
  • **Ball.DisplayStatistics** will invoke **Displaystatistics** from **ballClass**

The compiler will do *static binding* of these functions, i.e. determine which is which at compilation time.
Implementation Inheritance

• Used when one class can be implemented in terms of an existing class
• Example: polynomial class can be implemented in terms of an extended queue
• However, a polynomial is not a queue!
class Polynomial: private Extended_queue {
   // Use private inheritance.

public:
   void read();
   void print() const;
   void equals_sum(Polynomial p, Polynomial q);
   void equals_product(Polynomial p, Polynomial q);
   double evaluate(int value) const;
   int degree() const;

private:
   void mult_term(Polynomial p, Term t);
};
Abstract classes

class Shape {
public:
    virtual void rotate(int) = 0;
    virtual void draw() = 0;
    virtual double Area() = 0;
};

Shape s; //error
Abstract classes

• Can only be used as an interface and base for other classes

```cpp
class Circle: public Shape {
public:
    void rotate(int) { };
    void draw() ;
    double Area() { return (PI*radius*radius); };
private radius;
}
```
Abstract classes

• Important use is to provide an interface without exposing any implementation details

• Used in implementing frameworks for specific application classes