EECS402 Lecture 22

Andrew M. Morgan

Savitch Ch. 14
Inheritance

Inheritance

- Inheritance is one of the basic properties of an object-oriented language
- C++ allows inheritance (since it is considered OO)
- Inheritance allows one or more classes to obtain properties and functionality that is defined in another class
- New properties and/or functionality can be added to this inherited functionality
- A "base class" is a high-level class that contains attributes that all inherited classes have.
- A "derived class" is a lower-level class which gets some of its attributes from the base class, and adds others

Inheritance, Motivation


```
class RotatePartClass
{
  public:
    int      material;
    char     *name;
    axisType axis;

    void setName(char *name);
    void rotate(int amt);
};
```

```
class RemovablePartClass
{
  public:
    int    material;
    char   *name;
    bool   isRemoved;

    void setName(char *name);
    void togglePart();
};
```

- All part types share some common information that could be grouped together
- Create an inheritance tree, with base class containing this common information

Inheritance, Design

The power of inheritance, so we don't have to duplicate codes

```
class PartClass
{
  public:
    int   material;
    char  *name;

    void setName(char *name);
};
```

```
class RotatePartClass:public PartClass
{
  public:
    axisType axis;

    void rotate(int amt);
};
```

```
class RemovablePartClass:public PartClass
{
  public:
    bool   isRemoved;

    void togglePart();
};
```

Inheritance, Access To Members

- There is one final access class, in addition to private and public
- Public data and methods can be accessed from any place in the code where you have an object
- Private data and methods can be accessed only in the member functions of that class and *only* that class
- Protected data and methods can be accessed from within the member functions of that class *and* any class that is derived from it

Inheritance, Access Examples


```
class BaseClass
{
  public:
    int i;
  protected:
    int j;
  private:
    int k;
};
class DerivedClass:public BaseClass
{
  public:
    void print();
    int l;
  private:
    int m;
};
```

```
void DerivedClass::print()
{
  cout << i; //legal
  cout << j; //legal
  cout << k; //illegal!
  cout << l; //legal
  cout << m; //legal
}

void globalPrint(DerivedClass *obj)
{
  cout << obj->i; //legal
  cout << obj->j; //illegal
  cout << obj->k; //illegal
  cout << obj->l; //legal
  cout << obj->m; //illegal
}
```

- Methods are accessed the same as variables

Inheritance, Ctors and Dtors

- Constructors are called in the following order:
  - Base class ctor called first, followed by derived class ctor
  - If no explicit ctor is called for the base class, the default ctor is called (as always)
- Destructors are called in the following order:
  - Derived class dtor called first, followed by base class dtor
  - Note: this is backwards from the order ctors are called
- While you can not call a base class ctor from a derived class ctor, you CAN "call" one from the ctor initializers

Inheritance, Bad Constructor Initializers


```
class Point2Class
{
  public:
    Point2Class(int inX, int inY):x(inX),y(inY)
    {}
  protected:
    int x;
    int y;
  private:
    Point2Class() //def. ctor is private!
    {
      x = y = 0;
    }
};
class Point3Class:public Point2Class
{
  public:
    Point3Class(int inX, int inY, int inZ):x(inX),y(inY),z(inZ)
    {}
  private:
    int z;
    Point3Class()
    {
      x = y = z = 0;
    }
};
```

All illegal! Calls the default ctor of Point2Class, which is private! See how to fix it, next slide.

Inheritance, Good Constructor Initializers


```
class Point2Class
{
  public:
    Point2Class(int inX, int inY):x(inX),y(inY)
    {}
  protected:
    int x;
    int y;
  private:
    point2() //def. ctor is private!
    {
      x = y = 0;
    }
};
class Point3Class:public Point2Class
{
  public:
    Point3Class(int inX, int inY, int inZ):Point2Class(inX,inY),z(inZ)
    {}
  private:
    int z;
    Point3Class():Point2Class(0,0)
    {
      z = 0;
    }
};
```

Legal! Calls the public ctor in the point2 class

Inheritance, Shadowing Members


```
class Point2Class
{
  public:
    Point2Class(int inX, int inY):x(inX),y(inY)
    { }
    int x;
    int y;
    void print()
    {
      cout << "X: " << x << endl;
      cout << "Y: " << y << endl;
    }
};
class Point3Class:public Point2Class
{
  public:
    Point3Class(int inX, int inY, int inZ):Point2Class(inX,inY),z(inZ)
    { }
    void print()
    {
      cout << "X: " << x << endl;
      cout << "Y: " << y << endl;
      cout << "Z: " << z << endl;
    }
};
```

In this case, the function point3::print() shadows the function point2::print(). When a point3 object calls print(), point2::print() is never called.

Inheritance, Overcoming Shadowing


```
class Point2Class
{
  public:
    Point2Class(int inX, int inY):x(inX),y(inY)
    { }
    int x;
    int y;
    void print()
    {
      cout << "X: " << x << endl;
      cout << "Y: " << y << endl;
    }
};
class Point3Class:public Point2Class
{
  public:
    Point3Class(int inX, int inY, int inZ):Point2Class(inX,inY),z(inZ)
    { }
    void print()
    {
      Point2Class::print();
      cout << "Z: " << z << endl;
    }
};
```

This allows us to reuse the code written for Point2Class::print() and still be able to add new functionality as well.

Inheritance, Keyword Public

- Recall the use of the keyword public when inheriting:

```
class Point3Class:public Point2Class
{
  ...
};
```

- You can also inherit as protected or private
  - This is hardly ever done - very rare
- Private inheritance means that all public members of the base class have access type private in the derives class
- Protected means public members of the base class have protected access type in the derived class
- You **never** get weaker access due to different types of inheritance!

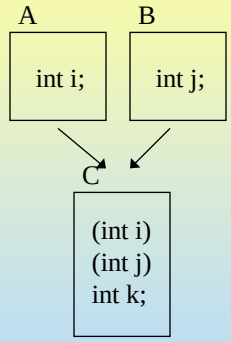
Jrava does not offer multiple inheritance

Multiple Inheritance

- A class can derive attributes and functionality from more than one base class

```
class A
{
  public:
    int i;
};

class B
{
  public:
    int j;
};
class C:public A, public B
{
  public:
    int k;
};
```

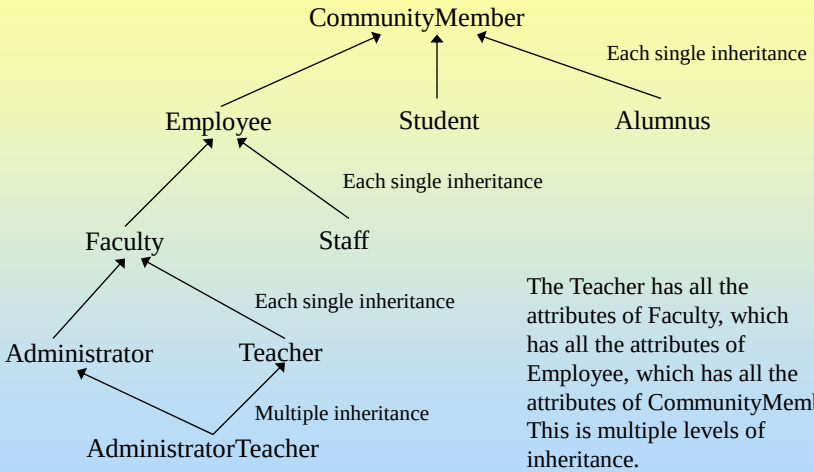


Multiple Levels Of Inheritance

- A multi-level hierarchy can be developed using inheritance.
- Multiple levels of inheritance is different from multiple inheritance.
  - Multiple inheritance: A class derives attributes from multiple other classes
  - Multiple levels of inheritance: One class derives attributes from another. A third class then derives all attributes from the derived class.
- Multiple levels of inheritance can be used to develop an "inheritance tree"
- Note: This is different from multiple inheritance!

An Inheritance Tree

- This diagram was taken from Deitel & Deitel, 2nd Edition



The Teacher has all the attributes of Faculty, which has all the attributes of Employee, which has all the attributes of CommunityMember. This is multiple levels of inheritance.