

Software Design
CSE 335, Spring 2006

Object-oriented programming

Stephen Wagner

Michigan State University

C++ Classes

- A class consists of
 - data (members)
 - methods to manipulate the data
- Constructors
- Destructors

```
class A {  
    private:  
    string name;  
    public:  
    A(const string & aName) { name=Aname; }  
};
```

```
class B {  
    private:  
    string name;  
    public:  
    B(const string & aName) : name(aName) { }  
};
```

Invoking Constructors

- `A::A(const string & aName) {name=Aname; }`
 - `string::string()` constructor is called
 - `string::operator=(const string &)` is then called
- `B::B(const string & aName) : name(aName) {}`
 - `string::string(const string &)` constructor is called
 - `string::operator=(const string &)` is never called

Some more C++ stuff

- What happens here?
`string a=b;`

Some more C++ stuff

- What happens here?

```
string a=b;
```

- Static:

```
class A
{
    protected:
    string name;
    static int count;
};
```

Some more C++ stuff

- What happens here?

```
string a=b;
```

- Static:

```
class A
{
    protected:
    string name;
    static int count;
};
```

```
int A::count;
```

Example

Suppose we have Employees and Managers. Both Employees and Managers have first and last names, a hiring date, and a department. A Manager has a level, and is also responsible for a group of employees.

How to implement this in C++?

Example

```
class Employee {
    public:
        string first_name;
        string last_name;
        Date   hiring_date;
        short  department;
};

class Manager {
    public:
        Employee   emp;
        list<Employee *> group;
        short      level;
};
```

- Isn't a Manager also an Employee?
- Can a Manager have another Manager in her group?

Derived Classes

Define new class by *extending* existing class

- *Base class*
 - Class that is extended
 - Defines function and data members that are *inherited* by the derived class.
- *Derived class*
 - *Inherits* function and data members from base class
 - May add additional function/data members
 - May also *override* inherited function members with new methods

How to declare a derived class

```
class DerivedClassName : public BaseClassName
{
    public:
    // New function/data members.
    protected:
    //
};
```

Example

```
class Manager : public Employee
{
    protected:
        list<Employee*> group;
        short          level;
};
```

Terminology:

- Manager called the *derived class* (also *subclass*)
- Employee called the *base class* (also *superclass*)
- Manager *inherits* from Employee

Pointers/references & inheritance

Pointer (or reference) to an object of a derived class can be used as pointer (or reference) to an object of its base.

```
void foo()
{
    Manager    m, *mPtr;
    Employee   e, *ePtr;

    ePtr = &m;    // OK
    mPtr = &e;    // Error
    e = m;       // Error
}
```

Advantages of inheritance

- Factor out code that is common in multiple classes
 - Less code to maintain
 - Fix errors once
- Reuse functions that operate on base-class objects
- Represent domain relationships explicitly in code

Constructors

```
class Employee {  
    protected:  
        string first_name;  
        string last_name;  
        ...  
    public:  
        Employee(const string &f, const string &l)  
            : first_name(f), last_name(l) {};  
};
```

Constructors

```
class Manager : public Employee {  
    protected:  
        list<Employee*> group;  
        short           level;  
    public:  
        Manager( ... )  
};
```

How do you write a constructor for Manager?

Example: Class Figure

```
class Figure {
    public:
        Figure (unsigned x, unsigned y)
            : xLocation(x), yLocation(y) {}

        unsigned getXLocation() const { return xLocation; }
        unsigned getYLocation() const { return yLocation; }

    protected:
        unsigned xLocation;
        unsigned yLocation;
};
```

```
class Rectangle : public Figure {
public:
    Rectangle (unsigned x, unsigned y,
              unsigned length, unsigned height)
        : Figure(x,y), lengthDimension(length),
          heightDimension(height) {}

    unsigned getLength() const { return lengthDimension;}
    unsigned getHeight() const { return heightDimension;}
protected:
    unsigned lengthDimension;
    unsigned heightDimension;
};
```

Example: Reusing Functions

```
bool nearerOrigin ( const Figure & f1,
                    const Figure & f2 )
{
    unsigned  f1X = f1.getXLocation();
    unsigned  f1Y = f1.getYLocation();
    unsigned  f2X = f2.getXLocation();
    unsigned  f2Y = f2.getYLocation();

    return (f1X*f1X + f1Y*f1Y) <
           (f2X*f2X + f2Y*f2Y);
}
```

```
int main(void)
{
    Figure    fig1(20,30), fig2(30,50);
    Rectangle rect1(10,40,13,28);
    Rectangle rect2(5,48,101,50);

    if (nearerOrigin(fig1, fig2)) { ... }

    if (nearerOrigin(rect1, rect2)) { ... }

    if (nearerOrigin(fig2, rect1)) { ... }
}
```

Question

How is it possible to pass a reference to a Rectangle object to a function that expects a reference to a Figure object?

Public, Protected, Public

- *public* members can be accessed by anyone
- *protected* members can be accessed by derived classes
- *private* members can only be accessed by the class itself (ignoring friends).

```

class A
{ private:
  int x;
  protected:
  int y;
  public:
  int z;
};

class B : public A
{
  B() {
    z=5; // legal
    y=4; // legal
    x=2; // not legal
  };
};

int main()
{
  A a;
  a.z=4; // legal
  a.y=5; // not legal
}

```

Access control

- Base class A with (data or function) member m
- Derived class B, which inherits from A
- Function fa that uses an object of class A
- Function fb that uses an object of class B

| | A | B | fa | fb |
|-----------|---|---|----|----|
| public | Y | Y | Y | Y |
| protected | Y | Y | | |
| private | Y | | | |

Member-function over-riding

```
class Employee {  
    protected:  
        string first_name;  
        string last_name;  
        Date   hiring_date;  
        short  department;  
    public:  
        void print (ostream & ) const;  
};
```

Member-function over-riding

```
void Employee::print(ostream & s) const
{
    s << "Name: " << first_name << " " << lastname;
}

main()
{
    Manager bob("Bob", "Jones");

    bob.print(cout);          // Manager inherits print
                             // from Employee
}
```

```
class Employee {
    protected:
        string first_name;
        string last_name;
        Date    hiring_date;
        short   department;
    public:
        void print (ostream & ) const;
};

class Manager : public Employee {
    protected:
        list<Employee *> group;
        short level;
    public:
        void print (ostream & ) const; }
```

Member-function over-riding

```
void Manager::print(ostream & s) const
{
    s << "Name: " << first_name << " " << lastname;
    s << "Level = " << level << endl;
}
```

Member-function over-riding

```
void Manager::print(ostream & s) const
{
    s << "Name: " << first_name << " " << lastname;
    s << "Level = " << level << endl;
}
```

```
void Manager::print(ostream & s) const
{
    Employee::print(s);
    s << "Level = " << level << endl;
}
```

Inheritance in other languages

- Java

```
public class Manager extends Employee
```

- C#

```
public class Manager : Employee
```

- Objective-C

```
@interface Manager : Employee
```

Terminology

- Distinction between *operation* and *method*
 - Operations are over-ridden with new methods
 - *print* is an operation; `Employee::print` and `Manager::print` are methods.
- We would like to be able to invoke an operation, as opposed to invoking a particular method.