CSE 303
Concepts and Tools for Software Development

Magdalena Balazinska

Winter 2007

Lecture 17 – Manipulating objects and inheritance in C++

Plan for Today

- Finish studying our first C++ class from last lecture
- Discuss when objects are created and destroyed
  - Creating objects on the stack
  - Creating objects on the heap
  - Copy constructors
  - Passing objects to functions
    - call-by-value vs call-by-reference
- Inheritance in C++
- Casting in C++ (we will do this next time)
- Virtual functions (we will do this next time)

Our Simple C++ Class

Examine the `Property` class (continued)

- Class definition is in `.h` file
  - Includes member function declarations
  - Can include function definitions but not recommended
    - Instead separate interface from implementation
- Member function definitions are in `.cc` file
- Pay close attention to the constructors & destructor
- Note the access specifiers: `public, private`
- Note that we can use pointer `this` (in `toString`)
- How the `static` attribute is declared and initialized
- The use of namespaces

Memory Management with Objects

- Examine the function `main`
  - See how we can declare an object
    - On the stack: `p1` and `p3`
    - On the heap: `p2`
  - See how we can pass an object by value
    - Function: `by_value`
    - Note that **we are making a copy**!
  - See how we can pass an object by reference
    - Function: `by_reference` **(no copy)**
- Examine the output that the program produces
  - Observe calls to constructors and destructors

Dynamic Memory Allocation

- In C++, dynamic memory allocation is done with `new` and `delete`
- `new`
  - Does not require any size specification
  - Invokes the constructor of the object
  - Returns a pointer of the right type
- `delete` invokes the destructor of the object
- Example:

```
Property *p2 = new Property(price,size);
delete p2;
```

New and Delete Examples

```
// Simple example
int *p_int = new int;
delete p_int;

// With initialization
int *p_int2 = new int(3);
delete p_int2;

// Allocating an array
int *p_array = new int[10];
delete [] p_array;
```

New and Delete Examples

```
// Allocating an object on the heap
Property *p2 = new Property(price,size);
delete p2;

// Allocating an array of objects
Property *p2_array =
             new Property[10](price,size);
delete [] p2_array;
```

Copy Constructor

- A copy constructor is invoked every time you create a new object from an existing object
- Example:

```
Property p1(price,size);
Property p3 = p1;
```

  Invokes: `Property(Property& p);`

- Other examples: passing an object by value or returning an object by value from a function
- If you do not provide a copy constructor, the default behavior is a memberwise copy
  - Not always the right thing: shallow copy vs deep copy

Where We Are in Our Plan

- Finish studying our first C++ class from last lecture
- Discuss when objects are created and destroyed
  - Creating objects on the stack
  - Creating objects on the heap
  - Copy constructors
  - Passing objects to functions
    - call-by-value vs call-by-reference
- Inheritance in C++
- Casting in C++ (we will cover this next lecture)
- Virtual functions (we will cover this next lecture)

Inheritance in C++

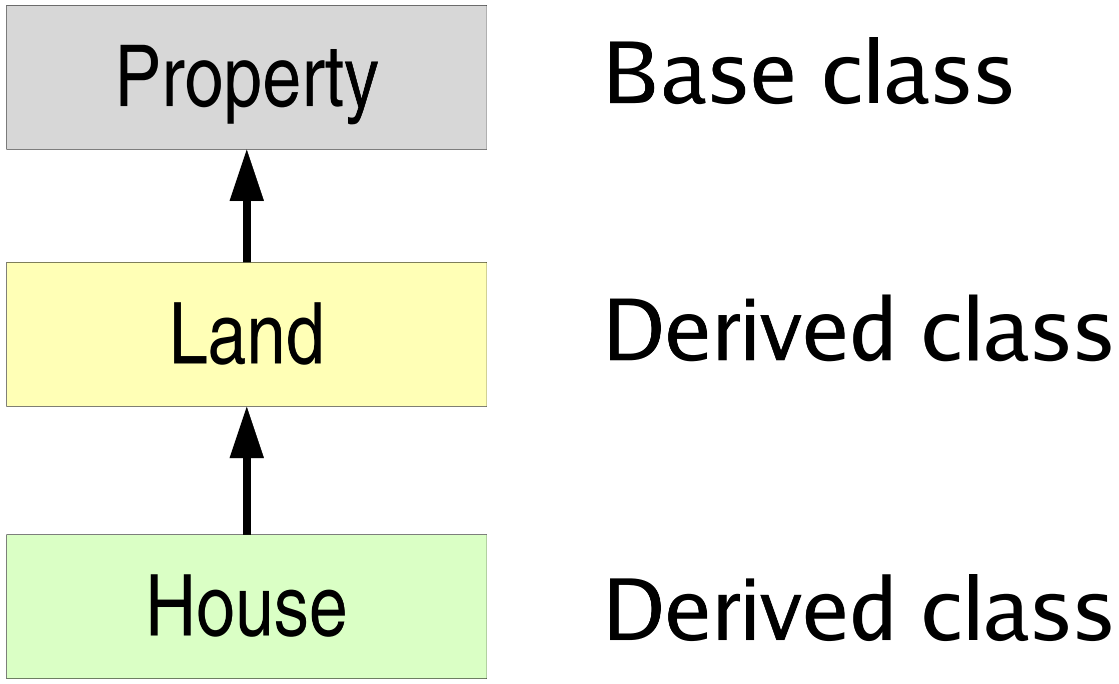
- Three types: public, protected, and private
- Public inheritance is used most frequently
  - public in base class -> public in derived class
  - protected -> protected
  - private -> not accessible in derived class
    - Facilitates encapsulation (information hiding)
- Protected data members are accessible from
  - Member functions
  - Member functions of derived classes

Base Class and Derived Class

```
class Land : public Property {
    ...
};
```

- Class `Land` inherits from class `Property`
- `Land` is called the derived class
- `Property` is called the base class

Inheritance Example



Constructors and Destructors

- Examine the output of program `estate`
  - Notice that the `Property` constructor is also called when a `Land` object is constructed
  - Notice that the `Property` destructor is also called when a `Land` object is destructed
- Invoked implicitly by default or
- Specific constructor can be invoked explicitly
  - Example: examine constructor of class `Land`
  - It invokes one of the constructors of `Property`

Function Overriding

- Derived class can override parent member function
- It simply declares a member function with
  - Same name as function in parent class
  - Same parameters
  - Example: `toString`
- To access parent member function from derived class, use the scope resolution operator
  - `Property::toString()`
- What is the difference between overloading and overriding?

Readings

- Carefully study the code that accompanies today's lecture