CSE 303 Concepts and Tools for Software Development

Richard C. Davis
UW CSE – 11/27/2006

Lecture 21 – C++ Inheritance

Administravia

- Having a partner != getting a free ride
  - You'll evaluate your partner at end of HW7
  - Work things out yourselves, if possible
    - Back-stabbing has risks
  - This simulates real life
- HW7 posted tomorrow
  - Discussion of Profilers postponed
  - You'll still have just over a week

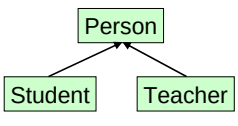
Where We Are

- Up to now
  - Linux/Shell scripting
  - C
  - C++
  - Tools for Developers
- From now on
  - More C++ features
  - Other tool/development topics

Today

- C++ Inheritance
  - Concepts
  - Syntax
  - Semantics

Today's Example



Base Class

Derived Classes

See Example in *Inheritance.cpp*

Inheritance in C++

- Three types
  - public (most common and most Java-like)
  - protected
  - private
- Public inheritance
  - public in base class → public in derived class
  - protected → protected
  - private → not accessible in derived class
    - Facilitates encapsulation (information hiding)
- Protected data members accessible from
  - Member functions
  - Member functions of derived classes

Base Class and Derived Class

```
class Student : public Person {
…
};
```

- Class Student inherits from class Person
  - Student is called the derived class
  - Person is called the base class

Syntax Differences with Java

- Declaring derived classes
  - Use "`public`" instead of Java "`extends`"
- Accessing base class constructor
  - 0-arg constructor → called automatically
  - Other constructors →must put in initializer list
- Accessing base class methods
  `Base::method()`

Dynamic Dispatch

```
class Person {
…
   void print() {
…
class Student : public Person {
…
   void print() {
…
Person *p = new Student();
p->print();
```

- Which method gets called?
  - Regular dispatch: `Person::Print()`
  - Dynamic dispatch: `Student::Print()`

C++ and Java Dispatch

- Default Dispatch
  - Java: Dynamic
  - C++: Regular
    - Must declare methods `virtual` to get dynamic
    - Only declare this way in class definition
- Big Problem with C++ default
  - Destructors must be declared virtual
    - Else deleting base class pointer can cause leak!
  - Automatically created destructor isn't virtual!

Inheritance and the "Big Three"

- Inherited Component is like a data member
- Constructors (Copy and 0-arg)
  - Base class constructor is called first
  - Then constructors called on data members
- Operator=
  - Analagous to constructors
- Destructor
  - Memory for derived class freed
  - Destructor for base class called

Slicing

```
Person p = Student();
```

- Casting stack-based objects causes slicing
  - Data in derived class lost
  - If objects inherit, use pointers or references

Dynamic casts

```
Person *p1 = new Student();
Person *p2 = new Teacher();

Student *s1 = (Student*)p1;                 //Fine
Student *s2 = (Student*)p2;                 //Subtle bugs
Student *s3 = dynamic_cast<Student*>(p1); //Fine
Student *s4 = dynamic_cast<Student*>(p2); //s4 == NULL
```

- Casting up the inheritance tree works
  - But if cast fails, can introduce subtle bugs
  - `dynamic_cast` returns `NULL` if cast fails

Other Differences we won't cover

- Declaring abstract classes
- No "Interfaces" in C++
  - Can us Multiple Inheritance instead

Summary

- C++ Inheritance is very similar to Java
- Big Gotchas
  - Methods are non-virtual by default
  - Declare destructors virtual
  - Slicing
- Want Java semantics for objects?
  - Make every (non-constructor) method virtual
  - Only allocate objects using new
  - Access all objects through pointers

Reading

- C++ for Java Programmers
  - Chapter 6: OO Programming: Inheritance
    - Skip 6.4, 6.8, and 6.9
    - 6.10 Recommended

Next Time

- Societal Implications
  - DRM