Lecture 10a
Pure Virtual Methods & Slicing
March 30th, 2016

Announcements
• No reading due this week
• Recitations:
  – Valgrind as a profiler
  – Mandatory
• PA6 is due today
  – Any questions?
• PA7 is due one week from today
• Midterm 2 weeks from today
• ACM tonight
  – Pizza & Bounce Technologies

Object Oriented Programming (Review)
1. Encapsulation
   – Collect data & code that operates on that data in one object
   – Provide a single, public interface
   – Changes to the implementation are local
2. Polymorphism (inheritance)
   – Abstraction via “is-a” relation
   – Write code at difference levels of abstract
   – Avoid redundant code
3. Inheritance as Union
   – A child class is the union of parent and new field/methods

What happened to interfaces?
• Java has classes and interfaces
• Interfaces provide a ‘public face’
  – Shared among multiple classes
  – But no implementation:
    • No constructors
    • No data fields
    • No implemented methods
      – Well, static methods are allowed in interfaces in Java 8
      – And constant values
• Nonetheless, interfaces are needed

Interfaces in C++
• In C++, interfaces are just classes
• Example: ZooProperty might be an interface
  – Animals might be zoo property
  – Toys in the gift shop might be zoo property
  – Garbage cans might be zoo property
• These are totally different concepts
  – But they share properties
    • Monetary value
    • Location in zoo
Java Interfaces

• In Java, interfaces are limited
  – They can define methods like
    • Value()
    • Location()
  – But they can only implement static methods
  – And they cannot store data like
    • Double value
    • (x,y) location
• Why does Java have these limitations?

C++ ZooProperty Interface

// A C++ interface as a class
class ZooProperty {
public:
  ZooProperty(double value, int x, int y)
  : loc_x(x), loc_y(y) {}  
  inline double Value() const {return value;}
  inline double X() const {return loc_x;}
  inline double Y() const {return loc_y;}
protected:
  double value;
  double loc_x;
  double loc_y;
};

C++ Interfaces

• In C++, interfaces are just classes
  – Often with no parent
  – At the least, orthogonal to the main is-a hierarchy
• Multiple inheritance allows them to be added to other classes
• C++ interface classes support
  – Data fields
  – Non-static (& static) methods
  – Constructors & Destructors

Pure Virtual Methods

• Imagine a ZooProperty method called InCaseOfEmergency()
  – For animals, rush them to shelter
  – For toys, close shop
  – For garbage cans, do nothing
• How do you implement this method at the level of ZooProperty?
  – Note: this is a lot like Animal::WarmUp()
• Answer: you don’t
  – You define it, but don’t implement it

Pure Virtual Example

// A C++ interface as a class
class ZooProperty {
public:
  ZooProperty(double value, int x, int y)
  : loc_x(x), loc_y(y) {}  
  virtual bool InCaseOfEmergency() = 0;
  inline double Value() const {return value;}
  inline double X() const {return loc_x;}
  inline double Y() const {return loc_y;}
protected:
  double value;
  double loc_x;
  double loc_y;
};

Pure Virtual Methods (II)

• Syntax: virtual signature = 0;
  – Note that only virtual methods can do this
• Semantics:
  – The method is defined but not implemented
  – Think of it like
    • Allocating an entry in the Virtual Function Pointer Table
    • And then filling that entry with NULL
  – It is now an error to create an instance of the class
    • In our example, ZooProperty zp; is now an error.
  – It is an error to create an instance of any class the inherits this one, unless that class overrides the method
Pure Virtual Methods (III)

- Pure virtual methods enforce interfaces
- Any class that inherits a class with pure virtual methods must override those methods
- Otherwise, it cannot be instantiated
- Classes with pure virtual methods are therefore interfaces
  - But not all interfaces have pure virtual methods
  - Referred to as Pure Virtual Classes
  - Even though they may have other, implemented methods and data fields