Encapsulation, cont’d
Polymorphism, pt 1

COMP 401, Spring 2016
Lecture 07
2/2/2016
Encapsulation In Practice
Part 2: Separate Exposed Behavior

• Define an “interface” for all exposed behavior
  – In Java, an interface is like a contract.
    • Indicates that a certain set of public methods are available.
    • One or more classes can indicate that they implement the interface.
  – Name of interface can be used as a type name
    • Just like class names are used as type names.
    • Value of an interface type variable can be set to any object that is an instance of a class that implements the interface.

• Mark constructors as public
Interfaces in Java

• Like classes, should go in their own .java file
  – Should have same name as file
  – Only one public interface per file.
  – Body of interface is a just list of method signatures.
    • Implementing classes MUST declare these methods as public

• Form:
  public interface InterfaceName {
    type method1(parameters);
    type method2(parameters);
    // etc...
  }

• Classes specify which interfaces they implement with “implements” modifier as in:
  public class ClassName implements InterfaceA, InterfaceB {

Interface Naming Conventions

• Interface name must be different from class names that implement the interface.

• Convention A
  – Start all interface names with “I” for interface.
    • For example: ITriangle, IPoint
  – Class names can be anything that is not in this form.

• Convention B
  – Use generic abstraction name for interface.
  – Make class names descriptive of implementation
    • If no natural way to do this, simply append “Impl” to generic abstraction name to differentiate.

• Personally, I generally go with convention B.
Programming To An Interface

• lec6.v6
• Separates Point into an interface and an implementing class.
  – Notice that distanceTo() and equals() are part of behavior I want the abstraction to expose.
    • Must be marked public
• Notice that main method uses variables with type Point (the interface name), but that the actual object is created as an instance of a specific class that implements the interface Point.
• Notice that Triangle only interacts with the methods specified in the Point interface.
Advantage of Encapsulation

• Can provide different implementations of the same behavior
  – lec6.v7
    • Create a new implementation of Point based on polar coordinates.
Exposed vs Internal Behavior

• Exposed behavior should be reflected in the interface(s) that a class implements
  – Recall that any method declared in an interface must be defined by an implementing class as a \textit{public} method.

• Internal behavior should be hidden
  – Use \textit{private} modifier on these methods to ensure that access only occurs within the class
• Continued application of encapsulation principle to Triangle by...
  – ... defining Triangle as an interface
  – ... rewriting what used to be the class Triangle as the class PointTriangle that implements the interface
  – ... hiding internal behaviors as private methods
Summing Up

• A Java file defines one public class or public interface.

• To support encapsulation:
  – Define exposed behavior as one or more interfaces
    • JavaBeans getters and setters for direct or derived properties.
    • Other methods that are part of the abstraction.
  – A class provides the implementation of one or more interfaces.
    • All fields within a class are marked as private.
    • Public constructor
    • Methods that implement any interface(s) must be public.
    • Internal methods marked as private.
Do you always need an interface?

• Best practice is to separate an abstraction into an interface and a class that implements it.
  – Allows you to have multiple classes that implement the interface in different ways.

• For simple classes for which you know that there will only be one implementation, you can get away without defining the interface separately.
  – Should still mark fields as private, constructor as public, and make a distinction between public methods for external behavior and private methods for internal behavior.
Polymorphism

• Poly = many, morph = forms
• General principle of providing access to an abstraction or method in many forms
  – Idea is that different forms “fit” different contexts
  – Goal of the underlying functionality is the same.
• In OO programming, principle is evident in a number of different places.
  – Today: Constructor overloading
Constructors

• What happens when you don’t define a constructor.
  – Default constructor with no arguments.
    • Creates new object with all fields set to default value
      – Numeric fields set to 0
      – Boolean fields set to false
      – String, Array, and any other sort of reference value field set to null.

• lec7.ex1
An aside on putting more than one class in a file.

• A Java file can only have one “public” class.
  – A class marked public can be imported into other code.
  – Must match file name

• One or more “non-public” classes in a Java file can also be defined
  – Only available to code in the same package.
  – Does not have to match file name
  – Often used to define classes that are related to the public class in that file.
Constructor Overloading

• Can define multiple versions of the constructor.
  – Distinguished from each other by type and number of parameters
    • Must be some difference otherwise the compiler won’t be able to tell them apart.
  – When you use the constructor, the right one will be chosen based on the parameters provided.
  – Note that once you define at least one constructor, the implicit, default no-argument constructor is not automatically available.

• lec7.ex2
Constructor Chaining

• Common pattern is to “chain” one constructor off of another.
  – To do this, the first line of code in the constructor must be the *this* keyword used as if a function with parameters.
  – The matching constructor is called first and allowed to execute.
  – Then any subsequent code is executed.
  – Can chain multiple constructors one on to another

• lec7.ex3