Medium Scale Structure – Inheritance

- Assume familiarity with subclassing, i.e.,
  - `class NewClass extends OldClass`

- `interfaces`: like a class, but has only declarations of methods; useful when the interface will be implemented by multiple classes

- `abstract` classes are incomplete; they cannot be instantiated, and must be subclassed

- `final` classes cannot be subclassed
Interfaces

• **class** `NewClass extends OldClass implements I1, I2, I3`
• **the supertypes of** `NewClass` **are** `OldClass and I1, I2, I3`
• `NewClass` **is a subtype of the supertypes**
Use of Subtypes

• A reference to NewClass can be used ‘polymorphically’ anywhere a reference to one of its supertypes can be used

• E.g., Comparable interface

  ```java
  public interface Comparable {
    int compareTo(Object o);
  }
  ```

• Generic sort

  ```java
  class Sorter {
    public static Comparable[] sort(Comparable[] list) {
      // implementation details ...
      return list;
    }
  }
  ```
Interface Declarations Contain

• Constants (implicitly `public static final`)
• Methods (implicitly `abstract public`)
• Nested classes and interfaces
When to Use Interfaces or Abstract Classes?

• Interfaces
  – for multiple inheritance, any class that will be extended should implement an interface because this supports the choice of
    • 1) extending the implementation of the interface, or
    • 2) use composition and forwarding to implement the interface

• abstract classes
  – some, or much, of the implementation will not change
Access Control

• On class, interface and member declarations
• Member access modifiers
  – *Hidden*
    • `private`: accessible only in the class
    • `package`: (the default) accessible in the package (including the class)
  – *Contractual*
    • `protected`: accessible in subclasses, the package, and the class
    • `public`: accessible anywhere
Access Control

– In general, use least accessibility possible
– i.e., use `get_X()` and `set_X()` methods on inaccessible instance variables
– Package and public can be applied to class and interface declarations.
Large Scale Structure

• Large bodies of reusable code require organization
• Java solution
  – *Package*: contains a set of types, resources and sub-packages
Package benefits

- Group related code
- Create a namespace
- Provide a protection domain
  - Use `public` access to make types visible outside your package
  - Default to less access
- Code within a package is considered ‘trusted’
- But sub-packages are not
Package Naming

• Hierarchical names
  – java.util.ArrayList

• Reference
  – Fully qualified name, or
  – Import (all or part of) a package
    • (java.lang is implicitly imported)

• Standard name conflict resolution
  – Reversed internet domain name
  – E.g. edu.nyu.cs.PQC.

• Package specified at top of source
  – Missing specification indicates the unnamed package
Archive files and Package Specification

• **java.util.jar**
  – Manipulate Java ARchive (JAR) files
  – Based on ZIP format
  – JAR files used to distribute classes and resources of a Package as a single unit

• Package specification
  – Name, version, vendor, etc.
  – *In the* Package *object*
  – Obtained with getPackage()
References