The Agenda for Today

- Assignment Example
- Alert, alert! Clean Code Please
- Review Object-Oriented Concepts
- JAVA Cup 3
What's the big deal about OOPs

- Encapsulation
- Polymorphism
- Inheritance

JAVA Cup 3

- Inheritance: classes
- Privatization: private and protected
- Constructors calling constructors
- Methods calling methods
- Designing classes and class hierarchies
- Abstract classes and methods
- Interfaces: requirements, documentation

Inheritance: Classes

- OOP languages have classes, instances and inheritance
- Keyword to specify superclass: extends
Example

- Define a superclass PERSON
- Define GRINGOTTS a subclass
- Define MUGGLEBANK another subclass
- Draw a class-hierarchy diagram

Inheritance: Instances

- Instances exhibit
  - Instance variables of their class
  - Public instance variables in superclasses
- Instances serve as targets for
  - Instance methods in their class
  - Public instance methods in superclasses

Design criteria

- Locate instance variables and public instance methods so that
  - There is no needless duplication of a public instance variable or method
  - Each public instance variable and method is useful to all subclasses of the class
Sequence

• Constructors: first calls zero-parameter constructor in direct superclass
• Methods: searches through subclass-superclass chain for first method of that name
• Subclass / superclass: must define superclass before subclass
• Only single inheritance allowed.

Overloading vs. Overriding

• Examples of polymorphism
• Overloading: two procedures with same name but different method signatures
• Shadowing or overriding: two procedures with same name and signatures but in different class hierarchies

Privatization

• Prevents direct instance-variable access
• Keyword private (instance variables)
  – allows access to instance-variable values from instance methods in the same class
  – prevents access to variable values outside of the class
• How to access private instance variables
• Idea extends to private methods
Protected

- Keyword protected
  - allows access to instance variables and methods from same class or from any subclass
  - Prevents access from …
- Order of declaration
  - public, protected, private

Constructors calling constructors

- Principle: avoid duplication
- Default: first calls zero-parameter constructor in direct superclass
- Within this class: this(<parameters>)
- From superclass: super(<parameters>)
- Must be first call in constructor
- Cannot explicitly call more than one constructor

Methods calling methods

- Using implicit targets
  - Drop target and field-selection operator
  - E.g. getName() instead of hp.getName()
- Using explicit targets
  - this.<method name>()
  - super.<method name>()
  - Calls can appear anywhere
Design Principles

- Explicit Representation
- No Duplication
- Look It Up
- Need To Know
- “Is-a” Versus “Has-a”

Abstract Classes and Methods

- Purpose:
  - To define inheritable, shared variables and methods
  - To require a method in a subclass
  - Shifts requirement-managing responsibility to Java compiler
- Keyword: abstract
  - In class definition: public abstract class * {...}
  - In method defn: public abstract int rating ();

Things to Note

- Abstract methods can only be defined in abstract classes
- You cannot create an instance of any abstract class
- You can declare a variable typed by an abstract class that is an instance of one of its subclasses. You cannot call a subclass method with a superclass variable as target.
Tree Relationship

- All classes form a tree
- The Object class is the root
- Classes marked final cannot be extended
- Final classes form the leaves
- Abstract classes lie high in the tree
- Can a class be both abstract and final?

Interfaces

- Keywords: interface, abstract, implements
  - `public interface InterfaceName {...}
  - `public abstract int methodName ();
  - `public ClassName implements InterfaceName {...}

- Purpose:
  - Impose requirements via abstract methods
  - Memory of design decisions

Interfaces: implementation

- You can implement multiple interfaces
  - Like multiple inheritance
- A variable typed by an interface
  - Can only call methods specified by interface
  - Can call subclass methods only if variable is cast as a subclass instance
- What is the difference between Java's `interface` and its `public interface`?
Interfaces: Advantages

- Excellent for documentation
  - Interfaces not cluttered by code
  - Convention to place descriptions here
- Allows good programming practice
  - Shifting requirement-managing responsibility to Java compiler
  - Encourage documentation of classes

JAVA Cup 3 Summary

- Inheritance
- Privatization
- Constructors Calling Constructors
- Methods Calling Methods
- Design Criteria
- Abstract Classes and Methods
- Interfaces