Object Oriented Design

Java Review – OOP

George Blankenship

CSCI 253

Object Oriented Design:

Java Review – OOP

George Blankenship

Overview

- Objects & class
- References & alias
- “this” & “super” reference
- Constructor & initialization block
- Garbage collection & destructor
- Modifiers
  – Public, Private, Protected
  – Static
  – Final

Object Oriented Programming (OOP)

- OO Principles
  – Abstraction
  – Encapsulation
- Abstract Data Type (ADT)
  – Implementation independent interfaces
  – Data and operations on data
- Java
Object & Class

- **Object**
  - Abstracts away (data, algorithm) details
  - Encapsulates data
  - Instances exist at run time
- **Class**
  - Blueprint for objects (of same type)
  - Exists at compile time

References & Aliases

- **Reference**
  - A way to get to an object, not the object itself
  - All variables in Java are references to objects
- **Alias**
  - Multiple references to same object
  - “X == Y” operator tests for alias
  - “X.equals(Y)” tests contents of object (potentially)

References & Aliases – Issues

- **Copying**
  - References
    - X = new Object();
    - Y = X; // Y refers to same object as X
  - Objects
    - X = new Object();
    - Y = X.clone(); // Y refers to different object
- **Modifying objects**
  - X = new Object();
  - Y = X;
  - X.change(); // modifies object for Y
“this” Reference
• Reserved keyword
• Refers to object through which method was invoked
• Allows object to refer to itself
• Use to refer to instance variables of object

Inheritance
• Definition
  – Relationship between classes when state and behavior of one class is a subset of another class
• Terminology
  – Superclass / parent ⇒ More general class
  – Subclass ⇒ More specialized class
• Forms a class hierarchy
• Helps promote code reuse

“super” Reference
• Reserved keyword
• Refers to superclass
• Allows object to refer to methods and encapsulated data in superclass
• Examples
  – super.x // accesses x in superclass
  – super() // invokes constructor in superclass
  – super.foo() // invokes method foo in superclass
Constructor

• Method invoked when object is instantiated
• Helps initialize object
• Method with same name as class w/o return type
• Implicitly invokes constructor for superclass
  – If not explicitly included

Constructor – Example

class foo {
  foo() { … } // constructor for foo
}
class bar extends foo {
  bar() { // constructor for bar
    // implicitly invokes foo() here
    …
  }
}
class bar2 extends foo {
  bar2() { // constructor for bar
    super(); // explicitly invokes foo() here
  }
}

Initialization Block

• Definition
  – Block of code used to initialize static & instance variables for class
• Motivation
  – Enable complex initializations for static variables
  • Control flow
  • Exceptions
  – Share code between multiple constructors for same class
Variable Initialization

- Variables may be initialized
  - At time of declaration
  - In initialization block
  - In constructor
- Order of initialization
  - Declaration, initialization block
    - (in the same order as in the class definition)
  - Constructor

Garbage Collection

- Concepts
  - All interactions with objects occur through reference variables
  - If no reference to object exists, object becomes garbage (useless, no longer affects program)
- Garbage collection
  - Reclaiming memory used by unreferenced objects
  - Periodically performed by Java
  - Not guaranteed to occur
  - Only needed if running low on memory

Destructor

- Method with name finalize()
- Returns void
- Contains action performed when object is freed
- Invoked automatically by garbage collector
  - Not invoked if garbage collection does not occur
- Usually needed only for non-Java methods
Method Overloading

• Description
  – Same name refers to multiple methods

• Sources of overloading
  – Multiple methods with different parameters
    • Constructors frequently overloaded
  – Redefine method in subclass

Package

• Definition
  – Group related classes under one name

• Helps manage software complexity
  – Separate namespace for each package
    • Package name added in front of actual name
  – Put generic / utility classes in packages
    • Avoid code duplication

Package – Import

• Import
  – Make classes from package available for use
  – Java API
    • java.* (core)
    • javax.* (optional)

• Example
  import java.util.Random;  // import single class
  import java.util.*;      // all classes in package

  // class definitions
Scope

- Part of program where a variable may be referenced
- Determined by location of variable declaration
  - Boundary usually demarcated by `{ }`

Modifier

- Java keyword (added to definition)
- Specifies characteristics of a language construct
- (Partial) list of modifiers
  - Public / private / protected
  - Static
  - Final
  - Abstract

Visibility Modifier

- Properties
  - Controls access to class members
  - Applied to instance variables & methods
- Four types of access in Java
  - Public
    - Most visible
  - Protected
  - Package
    - Default if no modifier specified
  - Private
    - Least visible
Visibility Modifier – Scope

- **“public”**
  - Referenced anywhere (i.e., outside package)
- **“protected”**
  - Referenced within package, or by subclasses outside package
- None specified (package)
  - Referenced only within package
- **“private”**
  - Referenced only within class definition
  - Applicable to class fields & methods

Visibility Modifier - Use

- For instance variables
  - Should usually be private to enforce encapsulation
  - Sometimes may be protected for subclass access
- For methods
  - Public methods – provide services to clients
  - Private methods – provide support other methods
  - Protected methods – provide support for subclass

Modifier – Static

- Static variable
  - Single copy for class
  - Shared among all objects of class
- Static method
  - Can be invoked through class name
  - Does not need to be invoked through object
  - Can be used even if no objects of class exist
  - Can not reference instance variables
Modifier – Final

- **Final variable**
  - Value can not be changed
  - Must be initialized in every constructor
  - Attempts to modify final are caught at compile time
- **Final static variable**
  - Used for constants
- **Final method**
  - Method can not be overridden by subclass
  - Private methods are implicitly final
- **Final class**
  - Class can not be a superclass (extended)
  - Methods in final class are implicitly final
- **Using final classes**
  - Prevents inheritance / polymorphism
  - May be useful for
  - Security
  - Object oriented design

Modifier – Abstract

- **Description**
  - Represents generic concept
  - Can not be instantiated
- **Abstract class**
  - Placeholder in class hierarchy
  - Can be partial description of class
  - Can contain non-abstract methods
  - Required if any method in class is abstract