Thinking in Objects

Reading

- Section 5.9
- Sections 8.7-8.11
- Section 9.4
- Chapter 10
Motivations

- Textbook uses a programming first, OO second approach
- This session emphasizes OO principles
- Session also covers material not emphasized earlier in the course

Objectives

- To create immutable objects from immutable classes to protect the contents of objects (§10.2).
- Determine the scope of variables in the context of a class (§10.3).
- Use the keyword this to refer to the calling object (§10.4).
- Apply class abstraction (§10.5).
- Develop classes for modeling composition relationships (§10.7).
- Design programs using the OO paradigm (§§10.8-10.10).
- Follow the class-design guidelines (§10.11).
StringBuffer Class

- Alternative to String class
- StringBuilder is almost the same as StringBuffer
- You can add, insert, or append new contents into a StringBuffer object
- Sample methods
  - append
  - deleteCharAt
  - insert
  - replace
  - reverse
- Use String if no need to change the string

Immutable Objects and Classes

- If the contents of an object cannot be changed once the object is created, the object is called an immutable object and its class is called an immutable class
- For a class to be immutable, it must
  - mark all data fields private
  - provide no mutator (i.e., setter) methods
  - Provide no accessor (i.e., getter) methods that returns a reference to a mutable data field object.

A String is represented as an array of char

Is the String class immutable?
Scope of Variables (Review)

- The **scope of a variable** is the part of a program where the variable can be referenced.
- The scope of instance and static variables is the entire class. They can be declared anywhere inside a class.
- The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable.
- A local variable must be initialized explicitly before it can be used.
- A parameter is a local variable whose scope is the entire method.
- A variable in a for loop header has scope in the entire loop.

The this Keyword

- The this keyword is the name of a reference that refers to an object itself.
- Common uses:
  - Reference a class’s hidden data fields.
  - Enable a constructor to invoke another constructor of the same class.

```java
private String name;
...
public void setName(String name) {
    this.name = name;
}
```
Calling Overloaded Constructors

```java
public class Circle {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    public Circle() {
        this(1.0);
    }

    public double getArea() {
        return this.radius * this.radius * Math.PI;
    }
}
```

Class Abstraction and Encapsulation

- Class abstraction - separate class implementation from the use of the class
- The creator of the class provides a description of the class and let the user know how the class can be used
- User of the class does not need to know implementation
- Detail of implementation is encapsulated and hidden from the user

Class Contract (Signatures of public methods and public constants)

Clients use the class through the contract of the class

Class implementation is like a black box hidden from the clients
Java Virtual Machine

- Referred to as the JVM
- Basis for execution of Java programs
- Accepts a form of intermediate code, called Java bytecode
- Includes an area of memory for dynamic memory allocation (referred to as the heap)
- Execution of blocks of code (e.g., methods) uses a stack for memory allocation

Static Methods

- Remember the main method header?
  
  ```java
  public static void main(String[] args)
  ```
- What does static mean?
  - associates a method with a particular class name
  - any method can call a static method either:
    - directly from within same class
    - or
    - using class name from outside class
  
  ```java
  Math.sqrt(x);
  ```
Calling Static Methods Directly

public class StaticCallerWithin
{
    public static void main(String[] args)
    {
        String song = getSongName();
        System.out.println(song);
    }
    public static String getSongName()
    {
        return "Piano Man";
    }
}

Output?
Piano Man

(c) 2011 Pearson Education, Inc. & Robert F. Kelly

Calling External Static Methods

public class StaticCallerFromOutside
{
    public static void main(String[] args)
    {
        System.out.print("Random Number from 1-100: ");
        double randomNum = Math.random();
        System.out.print(randomNum*100 + 1);
    }
}

• What’s the method header for Math.random?

    public static double random()

(c) 2011 Pearson Education, Inc. & Robert F. Kelly
Uses for Static Methods

- Static methods are commonly used to provide libraries of useful and related functions
- Used primarily when
  - All needed parameters are supplied as explicit parameters
  - Method needs to access only static fields of class
- Examples:
  - the Math class
    - has useful one and done methods
    - functions include pow, sqrt, max, min, etc.

The Math Class

- Includes constants
  - Math.PI (approximately 3.14159)
  - Math.E (base of natural logarithms, approximately 2.72)
- Includes static methods:
  - round, floor, and ceil
  - Math.round returns the nearest whole number
    - Math.round(3.3) returns 3
    - Math.round(3.7) returns 4
  - Math.floor returns the nearest whole number (as a double) that is equal to or less than its argument
    - Math.floor(3.3) returns 3.0
    - Math.floor(3.7) returns 3.0
  - Math.ceil (short for ceiling) returns the nearest whole number (as a double) that is equal to or greater than its argument
    - Math.ceil(3.3) returns 4.0
    - Math.ceil(3.7) returns 4.0
Static Variables

- **Not** instance variables
  - For instance variables, each instantiated object has its own instance variables w/ values
  - For a static variable, an entire class shares a variable
- May be public or private but are usually private for the same reasons instance variables are.
- Only one copy of a static variable and it can be accessed by any object of the class.
- Why use static variables?
  - Can be used to let objects of the same class coordinate information, like counting the number of objects

Example

```java
public class NamedPerson
{
    private String name;
    private static int numPeople = 0;

    public NamedPerson(String initName)
    {
        name = initName;
        numPeople++;
    }

    public String getName()
    {
        return name;
    }

    public static int getNumPeople()
    {
        return numPeople;
    }
}
```

numPeople counts the number of NamedPerson objects
The constructor for NamedPerson increments the count.

public class StaticVariablesDriver
{
    public static void main(String[] args)
    {
        System.out.println(peopleToString());
        NamedPerson joe = new NamedPerson("Joe");
        System.out.println(peopleToString());
        NamedPerson bob = new NamedPerson("Bob");
        System.out.println(peopleToString());
    }

    public static String peopleToString()
    { return "So far, " + NamedPerson.getNumPeople()
        + " people have been constructed";
    }
}

OUTPUT: So far, 0 people have been constructed
So far, 1 people have been constructed
So far, 2 people have been constructed

Static/Non-static Rules

- Static methods
  - cannot directly reference instance variables
  - cannot directly call non-static methods
  - can directly call static methods

- Non-Static methods
  - can directly reference instance variables
  - can directly reference static variables
  - can directly call non-static methods
  - can directly call static methods
Visibility Modifiers (review)

- public - Class, data, or method is visible to any class in any package
- private - data or methods can be accessed only by the declaring class

get and set methods are used to read and modify private properties

NOTE

- An object cannot access its private members, (b)
- It is OK if the object is declared in its own class, (a).
Array of Objects

- An array of objects is actually an array of reference variables
- Invoking circleArray[1].getArea() involves two levels of referencing

```java
Circle[] circleArray = new Circle[10];
```

Object-Oriented Thinking

- Textbook (chapters 1-6) introduced fundamental programming techniques
- Classes provide more flexibility and modularity for building reusable software
Coherence in Class Design

- A class should describe a single entity, and all the class operations should logically fit together to support a coherent purpose.
- Example
  - You can use a class for students
  - but you should not combine students and staff in the same class (because students and staff have different entities)

Separation of Responsibilities

- A single entity with too many responsibilities can be broken into several classes to separate responsibilities.
- The classes String, StringBuilder and StringBuffer deal with strings, but have different responsibilities
  - String class deals with immutable strings
  - StringBuilder class is for creating mutable strings
  - StringBuffer class is similar to StringBuilder except that StringBuffer contains synchronized methods for updating strings
Other Class Design Considerations

- Provide a public no-arg constructor
- Override the equals method and the toString method defined in the Object class whenever possible
- Follow standard Java programming style and naming conventions
- Choose informative names for classes, data fields, and methods
- Place the data declaration before the constructor
- Place constructors before methods
- Always provide a constructor and initialize variables

Using Visibility Modifiers

- A class should use the private modifier to hide its data from direct access by clients
- You can use get methods and set methods to provide users with access to the private data, but only to private data you want the user to see or to modify
- A class should also hide methods not intended for client use
- A property that is shared by all the instances of the class should be declared as a static property