Java By Abstraction: Chapter 2

Programming by Delegation

Some examples and/or figures were borrowed (with permission)
from slides prepared by Prof. H. Roumani

Object Oriented Programming (OOP)

- Encapsulate real-world entities in a **class**
  - Class usually represents a **noun** (i.e., a thing)
  - One-word class names begin with a capital letter
    - E.g., First, Rectangle3, Check01
  - Multi-word names begin each word with capital
    - E.g., FirstApp, PrintStream
- Instances of a class are called **objects**

Object Oriented Programming (OOP)

- Characteristics are represented as **attributes**
  - Attribute also usually represents a **noun**
  - One-word attribute name all in lowercase
    - E.g., width, height
  - Multi-word names begin second and subsequent words with capital
    - E.g., countPositive, cardNumber
  - Constant attribute name all in UPPER_CASE with words separated with an underscore

Object Oriented Programming (OOP)

- Operations are represented as **methods**
  - Method usually represents a **verb** (i.e., an action)
  - Always followed by **parentheses** (even if empty)
  - Additional data (called **parameters**) included in parentheses if necessary
  - One-word method name all in lowercase
    - E.g., equals(*anotherObject*), round()
  - Multi-word names begin second and subsequent words with capital
    - E.g., scale(*x, y, w, h*), getArea()

Accessing Attributes

- Assume **r** represents a Rectangle3 object
- Attributes of type int: **width, height**
- Attribute access syntax
  - *objectIdentifier.attributeName*
- Examples
  - int currentWidth = **r.width**;
  - int newWidth = 8;
    r.width = newWidth ;

Invoking a Method

- Assume r represents a Rectangle3 **object**
- Method **getArea()** returns area as **int**
- Method invocation syntax
  - *objectIdentifier.methodName(parameters)*
- Examples
  - int area = **r.getArea()**;

Instantiating Objects

- Use the keyword **new** to instantiate (i.e., create) an object
- Invoke the class's **constructor** method to initialize the object's state
- Object declaration and instantiation syntax
  - *ClassName identifier* = **new** *ClassName*();
- Example
  - Rectangle3 r = **new** Rectangle3();

Using Objects (Example)

```
…
int width = 8;
int height = 5;
Rectangle3 r = new Rectangle3();
r.width = width;
r.height = height;
int rArea = r.getArea();
System.out.println(rArea);
…
```

Utility Classes

- Uses Procedural Paradigm
  - Performs computation, not data storage
- Represent computations, not objects
- E.g., Math class
- All methods and attributes are **static**
  - Can be called without first declaring an object
  - E.g., Math.PI, Math.E, Math.round(), Math.log()
- Non-utility classes may also have some static methods and/or attributes

Main Classes

- Can be run from the command-line
- Starting point for a Java application
- Coordinates use of helper classes (i.e., components)

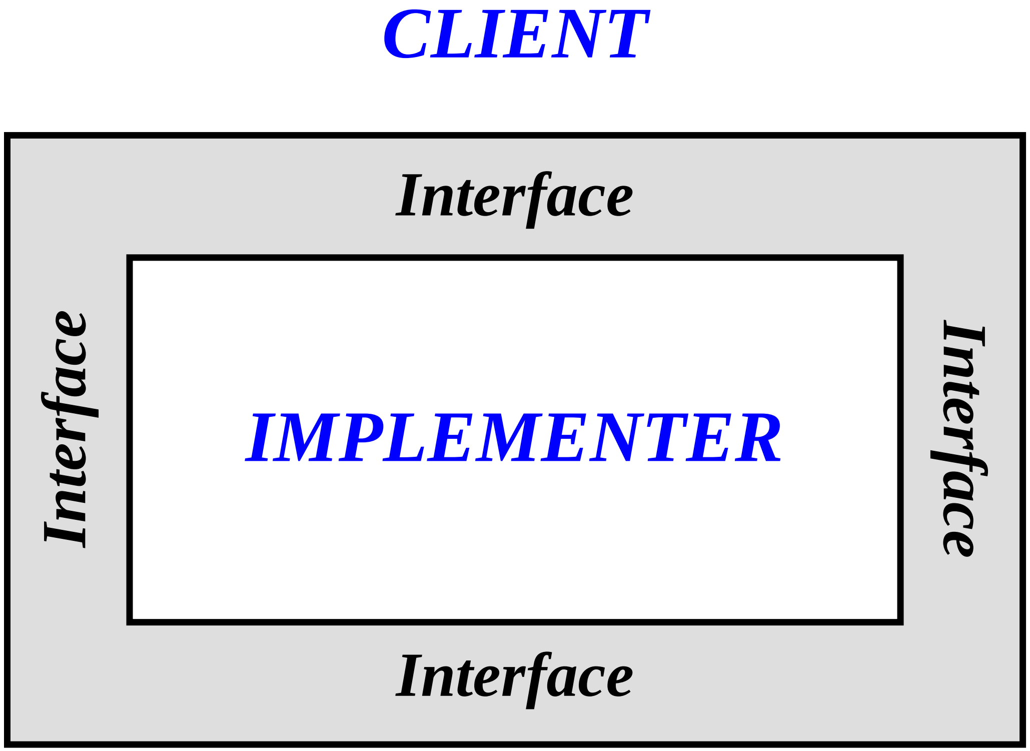
Delegation by Abstraction

- Determine what needs to be done
- Which helper class can accomplish each task
- Abstract the details of how each is accomplished
- Bread analogy in text (p. 56)
  - Difficult to grow, harvest, and mill wheat, to bake into bread
  - Instead, coordinate with a farmer, miller, and baker

The Client View

- The **client** develops the main class
  - Understands the big picture, the purpose of the application
  - Knows what each component does but not how it does it
- The **implementer** develops a component
  - Focuses only on the inner details of one component
- Client and Implementer share info on a need-to-know basis

The Client View



Access Modifiers

- Hide implementation details from clients
- Apply to classes, methods, and/or attributes
  - Features with **public** access appear in the API and are accessible to clients
  - Features with **private** access are not in the API and are **not** accessible to clients
  - Features with protected access are in the API, but are accessible only to other implementers
  - Features with no specified access are not in the API and are available only classes in the same package (i.e., directory)

Contracts

- Guarantee between client and implementer
- Precondition
  - What the client must satisfy
- Postcondition
  - What the implementer must deliver
- Liability
  - Pre. is satisfied and post. is satisfied → Good
  - Pre. is satisfied and post. is not satisfied → Implementer at fault
  - Pre. is not satisfied → Client at fault
  - If no precondition stated, then client need not satisfy anything

Contracts in Java

- Methods in the Java specify contracts as follows:
  - Precondition is always true unless stated otherwise
  - Postcondition is specified under Returns and Throws
- Example:

`double squareRoot(double x)`

Returns the square root of the given argument.

Parameters:

`x` - an argument.

Returns:

the positive square root of `x`.

Throws:

an exception if `x` < `0`.

TYPE and Java Standard Library

- Contains over 3000 components
- Class details contained in TYPE API and Java API
- Organized into packages and subpackages
- Examples
  - type.lib.Rectangle3
  - java.util.Scanner

`java.awt`	Provides support for drawing graphics. AWT = Abstract Windowing Toolkit
`java.beans`	Provide support for Java Beans.
`java.io`	Provides support for file and other I/O operations.
`java.lang`	Provides the fundamental Java classes. This package is auto-imported by the compiler.
`java.math`	Provides support for arbitrary-precision arithmetic
`java.net`	Provides support for network access.
`java.rmi`	Provides support for RMI. RMI = Remote Method Invocation
`java.security`	Provides support for the security framework.

`java.sql`	Provides support for databases access over JDBC
	JDBC = Java Database Connectivity,
	SQL = Structured Query Language

`java.text`	Provides formatting for text, dates, and numbers.
`java.util`	Miscellaneous utility classes including JCF. JCF = Java Collection Framework
`javax.crypto`	Provides support for cryptographic operations.
`javax.servlet`	Provides support for servlet and JSP development. JSP = Java Server Pages
`javax.swing`	Provides support for GUI development. GUI = Graphical User Interface
`javax.xml`	Provides support for XML processing. XML = eXtensible Markup Language

Importing Packages and Classes

- Indicate use of Java Standard Library (other than java.lang.*) or other Java library (e.g., TYPE)
- Import one or all classes in a subpackage (using *)
- Import statement syntax
  - **import** *package.subpackage.class*; // imports a single class
  - **import** *package.subpackage.**; // imports all classes in subpackage
- Example
  - **import** java.util.Scanner; // imports only the Scanner class
  - **import** type.lib.*; // imports all classes in the lib subpackage

Ready-Made Input and Output

- import java.util.Scanner; // place at top of file
  - Captures user input from the terminal
  - Parses lines, words, and primitive data types
- import java.io.PrintStream; // place at top of file
  - Outputs text to the terminal
  - Formats output
    - Field width
    - Specify number of decimal places

Parsing Input

- Scanner input = new Scanner(System.in);
  - Tokenizes input (i.e., separates using whitespace)

- next()
  - Returns the next word
- nextLine()
  - Returns the next line
- nextBoolean()
- nextChar()

- nextInt()
  - Parses next token as int
- nextDouble()
  - Parses next token as double
- nextLong()
- nextFloat()

Formatting Output

- PrintStream output = new PrintStream(System.out);

- print(*variable*) or print("*string literal*")
  - Outputs text to the terminal
- println(*variable*) or println("*string literal*")
  - Outputs text to the terminal and appends a newline character
- printf("*format string*", *variable…*)
  - Outputs formatted text to the terminal

Formatting Output

- Format string syntax (see p. 108)
  - %[flags][width][.precision]conversion
  - flag: , or 0
  - width: field width (text: left aligned; digits: right aligned)
  - precision: number of decimals
  - conversion: d (integer), f (real), s (text), or n (newline)
- Can also include non-format text
- Example
  - double x = 15.753;
    output.printf("Cost: %.2f", x); // outputs Cost: 15.75

Program Template

- See page 70
- Template for all of your 1020 Java programs
- Memorize it