



CSE 143

Programming as Modeling
Programming Practice

Reading: Ch. 1-3 (review)





6/23/2004
(c) University of Washington
01-1

Building Virtual Worlds

- Much of programming can be viewed as building a *model* of a real or imaginary world in the computer
 - a banking program models real banks
 - a checkers program models a real game
 - a fantasy game program models an imaginary world
 - a word processor models an intelligent typewriter
- **Running the program (the model)** simulates what would happen in the modeled world
- Often it's a lot easier or safer to build models than the real thing
 - Example: a tornado simulator

6/23/2004
(c) University of Washington
01-2

Java Tools for Modeling

- **Objects** in Java can model *things* in the (real or imaginary) world
 - The bank: Customers, employees, accounts, transactions...
 - Checkers: The Checkerboard, pieces, players, game history
 - Video game: Characters, landscapes, obstacles, weapons, treasure, scores
 - Documents: paragraphs, words, symbols, spelling dictionaries, fonts, smart paper-clip
- **Objects have**
 - Responsibilities – what you can ask them to do
 - Properties – what they know

6/23/2004
(c) University of Washington
01-3

Basic Java Mechanisms for Modeling

- A class describes a *template* or *pattern* for things; an object or instance of a class is a *particular* thing
- Constructors model ways to create new instances
- Methods model *actions* that these things can perform (i.e., to carry out their responsibilities)
- Messages (method calls) model requests from one thing to another
- Instance variables model the state or properties of things
- **public vs. private** models how much an objects wants to reveal about itself
 - Private: "Please put me on your do-not-call list"

6/23/2004
(c) University of Washington
01-4

What Makes a Good Model?

- Often, the closer the model matches the (real or imaginary) world, the better
 - More likely it's an accurate model
 - Easier for human readers of the program to understand what's going on in the program
- Sometimes, a too detailed model of reality is not a good thing. Why?

6/23/2004

(c) University of Washington

01-5

What Else Makes a Good Model?

- The easier the model is to extend & evolve, the better
 - May want to extend the model...
 - May need to change the model...
- Sad law of life: "A Program is Never Finished"
- Why??

6/23/2004

(c) University of Washington

01-6

Coupling and Cohesion

- A qualitative way to evaluate the organization of classes or modules
- **Coupling** – the degree to which a class interacts with or depends on another class
- **Cohesion** – how well a class encapsulates a single notion
- A system is more robust and easier to maintain if
 - **Coupling** between classes/modules is **minimized**
 - **Cohesion** within classes/modules is **maximized**

6/23/2004

(c) University of Washington

01-7

A Review Example

```
/* Representation of an employee in a personnel system
 * @author Hal Perkins
 * @version CSE143 Sp03 lecture example */
public abstract class Employee {
    // instance variables
    private String name;    // employee name
    private int id;         // employee id number
    private double pay;     // employee weekly pay

    /** Construct a new employee with the give name, id number, and weekly pay
     * @param name Employee's name
     * @param id Employee's id number
     */
    public Employee(String name, int id, double pay) {
        this.name = name;
        this.id = id;
        this.pay = pay;
    }
    ...
}
```

6/23/2004

(c) University of Washington

01-8

Employee Example (2)

```
/**
 * Return the name of this employee
 * @return Employee name
 */
public String getName() {
    return name;
}

/**
 * Return the id number of this employee
 * @return Employee id number
 */
public int getId() {
    return id;
}

...
```

6/23/2004

(c) University of Washington

01-9

Employee Example (3)

```
...

/**
 * Return the pay earned by this employee
 * @return Employee's pay for the current pay period
 */
public double getPay() {
    return pay;
}

/** Set this employee's pay
 * @param newPayRate new pay rate for this employee
 */
public void setPay(double newPayRate) {
    pay = newPayRate;
}
}
```

6/23/2004

(c) University of Washington

01-10

Programming Practice

- Programmers do certain things beyond just getting the program to run
- These are often matters of “practice” rather than functional requirements
- Some examples:
 - toString methods
 - commenting conventions
 - formatting
 - style
 - main methods

6/23/2004

(c) University of Washington

01-11

toString: Recommended for All Classes

- A method with this exact signature:
`public String toString();`

```
/** Return a string representation of this employee */
public String toString() {
    return "Employee(name = " + name + ", id = " + id +
        ", pay = " + pay + ")";
}
```

- Java treats toString in a special way
 - In many cases, will automatically call toString when a String value is needed:
System.out.println("My bank account: " + account);

6/23/2004

(c) University of Washington

01-12

toString

- Good while debugging
`System.out.println(anObject);` // calls `anObject.toString()`
- Secret Java lore:
 - All Objects in Java have a built-in, default `toString` method
 - So why define your own??

6/23/2004

(c) University of Washington

01-13

Java Documentation Comments

- Java provides a clean way of including documentation as part of the source code – **JavaDoc** comments
 - Begin with `/**` and end with `*/`
- Special tags to control formatting
 - `@author` – specify author
 - `@version` – version number, date, etc.
 - `@param` – description of a method parameter
 - `@return` – description of a non-void method result
 - Others (links, see also, ...), plus can use arbitrary html
- Used to produce all online Java API documentation

6/23/2004

(c) University of Washington

01-14

Javadoc Tool

- Properly formed comments can be extracted automatically
- Produce web documentation with convenient hyperlinks, etc.
- All the API documentation we use was generated this way: directly from comments in the code!
- Built-in support in current DrJava, Eclipse
- Command-line tool available: `javadoc`
- Learn to use it!

6/23/2004

(c) University of Washington

01-15

main: Another Common Practice

- Place a static `main` method in a class
- The Java VM will look for `main` and start execution there automatically

```
/** Create and test some of the Employee operations */
public static void main (String[ ] args) {
    Employee bob = new Employee("Joe Bob", 314, 1000.00);
    bob.setPay(1200);
    System.out.println(bob.getName());
    System.out.println(bob); // automatically calls bob.toString()
}
```

```
} // end of Employee
```

6/23/2004

(c) University of Washington

01-16

Required vs. Recommended

- Writing toString is "recommended"
- Creating main methods is "recommended"
- You've probably been given other recommendations:
 - comments, variable naming, indentation, etc.
 - Use this library, don't use that library
- Why bother, when the only thing that matters is whether your program runs or not?
 - Answer: Whether your program runs or not is *not* the only thing that matters!
Yes, it needs to work, but people need to be able to read and understand it

6/23/2004

(c) University of Washington

01-17

Software Engineering and Practice

- Building good software is not just about getting it to produce the right output
- Many other goals may exist
- "Software engineering" refers to practices which promote the creation of good software, in all its aspects
 - Some of this is directly code-related: class and method design
 - Some of it is more external: documentation, style
 - Some of it is higher-level: system architecture
- Attention to software quality is important in CSE143
 - as it is in the profession

6/23/2004

(c) University of Washington

01-18