CSE 143 Java

Programming as Modeling

Reading: Ch. 1-6

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

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

Basic Java Mechanisms for Modeling

- A ***class*** describes a *template* or *pattern* for things
- An ***object*** or ***instance*** is a *particular* thing
- ***Constructors*** model ways to create new instances
- ***Methods*** model *behaviors* or *actions* that these things can perform
- ***Method calls*** model ***messages***: requests from one object to another
- ***Instance variables*** model the state or properties of things

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?

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??

More Techniques for Good Modeling

- Separating STATE from BEHAVIOR is a useful design strategy
- One way to capture this is to define good ***interfaces*** separate from the implementation (code)
- An interface specifies to clients (users of the class) what are the operations (methods) that can be invoked; state is not part of the interface

State vs Behavior

- State
  - has blue hair
  - wearing glasses
  - wearing blue shoes
  - is hopping mad
- Behavior
  - clenches fist
  - raises arm
  - hops up and down
  - screams

Which is More Fundamental?

- Behavior or State?
- What do you think, and why?

7/1/2003 (c) University of Washington 01-9

Behavior vs. State in Java

- Example: Bank accounts have balances
  - When you're at the ATM you can check your balance
- Does this mean BankAccount class should have a “balance” instance variable?
- Does this mean BankAccount class should have a "getBalance" method?
- These two questions look superficially similar

7/1/2003 (c) University of Washington 01-10

"Balance" Question Rephrased

- "Who cares if a class has an instance variable named *balance*?
  - Answer, Nobody cares, except the poor guy implementing the class
- "Who cares if a class has a method named *getBalance*?"
  - Answer, potentially a lot of people!

Boss: "You're hired. Now create a BankAccount class, and it had better have a *getBalance* method -- or you're fired."

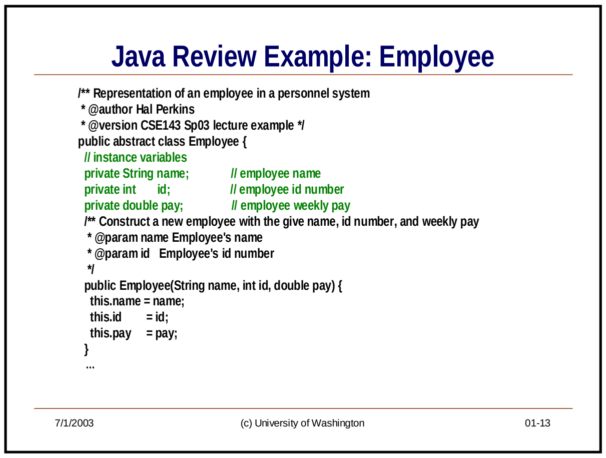
Programmer: "What instance variable should I use?"

Boss: "Don't bother me with that trivia – or you're fired."

7/1/2003 (c) University of Washington 01-11

Appendix
Some Java Review Examples

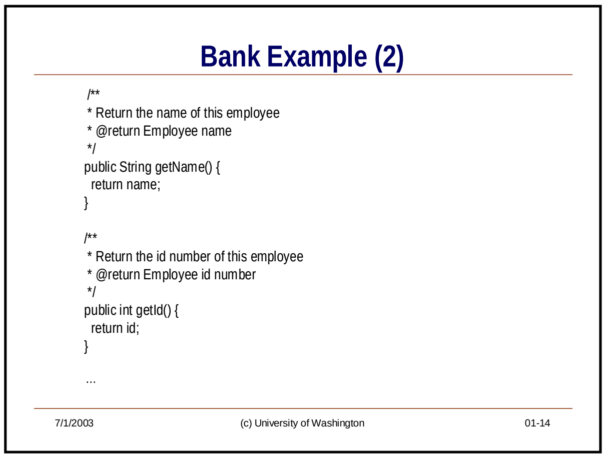
7/1/2003 (c) University of Washington 01-12



Java Review Example: Employee

```
/** 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;
  }
  ...
```

7/1/2003 (c) University of Washington 01-13



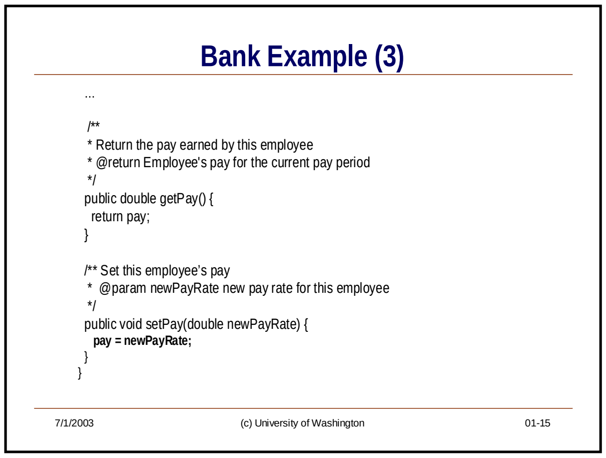
Bank Example (2)

```
  /**
   * 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;
  }

  ...
```

7/1/2003 (c) University of Washington 01-14



Bank 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;
  }
}
```

7/1/2003 (c) University of Washington 01-15