Instance Variables
Constructors
If Statements

Lecture 8

Some slides borrowed from Kurt Eiselt, Tamara Munzner, and Steve Wolfman. Some learning goals from Beth Simon.

Readings
This Week’s Reading: Review Ch 1-4
(that were previously assigned)

(Reminder: Readings are absolutely vital for learning this stuff!)

Labs and Tutorials
This week is Lab #4.

Midterms – Save the Dates!
- Midterm #1 is 5:30-6:30pm on February 10 (Tuesday) in Woodward IRC 2
- Midterm #2 is 6-7pm on March 11 (Wednesday) in Woodward IRC 2

Midterm Study Tips:
- Old midterms on-line.
- Try programming without notes.
- Try programming without computer!

Programming Assignment 1
- Assignment 1 is up on WebCT!
  - Click on the “Assignments” icon.
- Due at NOON, February 17 (Tuesday), via electronic hand in.
  - It may take me a couple days to setup the electronic hand in, so if you’re really fast, please wait a day or two.
- Start early!

Learning Goals
By the end of the next several lectures you will be able to...
- Create your own classes, with:
  - Public and private fields and methods
  - Helpful documentation that works with javadoc
  - Basic principles of abstraction and encapsulation (information hiding)
- Explain why abstraction and information hiding are important.
Learning Goals
By the end of class today you will be able to...
- Create classes with instance fields.
- Access public instance fields directly.
- Protect your instance fields by making them private.
- Provide accessor and mutator methods for private instance fields.
- Create constructor methods to initialize the instance fields.
- Have your program make decisions using if statements.

Review: How to Create a Class
- Client Programmer:
  MyClass a;
- File MyClass.java
  public class MyClass {
  }

Review: How to Create a Class
- Client Programmer:
  MyClass a;
- File MyClass.java
  public class MyClass {
    public void sayHello() {
      System.out.println("Hello");
    }
  }

Review: Passing Parameters
- In the object user (the caller):
  a.flatter("fabulous");
- In the class definition (the callee):
  public void flatter(String adjective) {
    System.out.println("Wow, you look " + adjective + "!");
  }

Review: Passing Parameters
- In the object user (the caller):
  a.flatter("fabulous");
- In the class definition (the callee):
  public void flatter(String adjective) {
    System.out.println("Wow, you look " + adjective + "!");
  }

Review: return Statement
- How do we get information back from the method?
- We use a return statement. ("return" as in "election returns" – results that come back to you)
Review: return Statement
- The return statement in a method tells the method to “return” to the caller.
- Syntax:
  ```java
  return;
  return expression;
  ```
- With an expression, it gets evaluated and that value is what the method call returns.

Questions?

Objects Can Contain Variables
- It’s convenient for objects to remember things about themselves, e.g.:
  - java.awt.Rectangle class in the book
  - if we created a UBCStudent class...
- To allow this, objects can have their own variables.
- Declare the variables in the class.
- But each (instance of an) object gets its own copy:
  - Called “instance fields” or “instance variables” or “attributes”

Example Class: Celebrity
```java
public class Celebrity {
  String name;
  String description;

  // ...declare methods, too...
}
```

Each Object Has Its Own Instance Fields
- You declare the instance fields once, but each object gets its own copy???

Each Object Has Its Own Instance Fields
- You declare the instance fields once, but each object gets its own copies???
- Remember: the class is like a blueprint or pattern. It says how to make objects:
  - A car blueprint shows a steering wheel, but each car gets its own steering wheel.
  - A dress pattern shows one pocket, but each dress has its own pocket.
  - Celebrity class shows String variables for name and description, but each object gets its own.
Accessing Instance Fields
- If the instance fields are public, users of the class can treat them like normal variables:
  ```java
  prof.name = "Alan";
  prof.description = "dignified";
  ```
  (If you say neither “public” nor “private”, the default is called “package access”, which is basically public for our purposes.)

Keep Instance Fields Private!
- If the instance fields are public, users of the class can treat them like normal variables:
  ```java
  prof.name = "Alan";
  prof.description = "dignified";
  ```
  - If the instance fields are public, users of the class can misuse or mess up your instance fields!
  - Good style is to keep them private.

Public vs. Private
- public means accessible outside of this class
- private means accessible only within this class.
```
public class Celebrity {
    private String name;
    private String description;
}
```

Keep Instance Fields Private!
- Good style is to keep them private.
- If the instance fields are private, users of the class can’t access them. How do they use your class?

Accessor and Mutator Methods
- Good style is to keep them private.
- If the instance fields are private, users of the class can’t access them. How do they use your class?
- The standard Java convention is to provide public methods to access (read) or mutate (modify) the instance fields.
  - E.g., getName() or setName(String name)
  - You can control access to fields this way.

Questions?
Constructors

- What happens if the user forgets to set the instance fields?

It's best to supply a default for instance fields at the moment the object is created. This way, all objects are always properly created. We do this by writing our own code for the constructor methods.

Constructors

To create a constructor method, declare a method with:
- the same name as the class
- no return type (at all, not even void)
- parameters, if you want
- Put code in the method to initialize the instance fields.

Constructors

public Celebrity() {
    this.name = "Joe Schmoe";
    this.description = "forgettable";
    this.salary = new BigInteger("0");
}

Using vs. Designing Classes

<table>
<thead>
<tr>
<th>Using a Class</th>
<th>Designing a Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read API.</td>
<td>Write API.</td>
</tr>
<tr>
<td>Get and use objects.</td>
<td>Provide blueprint/pattern for objects.</td>
</tr>
<tr>
<td>Instantiate objects with</td>
<td>Define constructor</td>
</tr>
<tr>
<td><strong>new ClassName(...)</strong></td>
<td><strong>ClassName(...) {}</strong></td>
</tr>
<tr>
<td>Call methods to get things</td>
<td>Implement methods. Say how to actually do things.</td>
</tr>
<tr>
<td>done.</td>
<td>Access public things only.</td>
</tr>
<tr>
<td>Decide what is public and</td>
<td>Decide what is public and what is private.</td>
</tr>
</tbody>
</table>
Questions?

Jumping ahead a bit...
- We’re going to jump ahead a little bit.
  - This is useful for Assignment 1.
  - Also makes it easier to write more interesting example classes...
  - You can skim Ch 5.1 and 5.2 for more details.

Making Decisions
- Recall that a sliderule can compute multiplication, but can’t make decisions.
- But even a transistor can make a decision!
- How do we get our Java programs to make decisions?

Java’s if statement
- Syntax:
  ```java
  if (condition) statement;
  ```
- Examples:
  ```java
  if (tax < 0) tax = 0;
  if (age < 19) {
    System.out.println("Sorry, you may not buy alcohol.");
  }
  ```

Java’s if statement
- Intuitively, a lot like English:
  - If you’re hungry, have a sandwich.
  - `if (tax < 0) tax = 0;`

Java’s if statement
- You can control a group of statements with curly braces.
- Syntax:
  ```java
  if (condition) {
    statement;
    ...
  }
  ```
- Example:
  ```java
  if (age < 19) {
    System.out.print("Sorry, ");
    System.out.print("you may not ");
    System.out.print("buy alcohol.");
  }
  ```
Java’s if statement with else

- Syntax:
  
  ```java
  if ( condition ) statement;
  else statement;
  ```

- Example:
  
  ```java
  if (age < 19)
      System.out.println("You may not buy alcohol.");
  else
      System.out.println("Buy beer!");
  System.out.println("This lecture sponsored by Molson");
  ```

Questions?

Implementing Die

```java
/**
 * Provides a simple model of a die (as in pair of dice).
 */
public class Die {
    
    
}
```
Random Numbers

- **Random class in java.util package**
  - public Random()
  - Constructor
  - public float nextFloat()
    - Returns random number between 0.0 (inclusive) and 1.0 (exclusive)
  - public int nextInt()
    - Returns random integer ranging over all possible int values
  - public int nextInt( int num )
    - Returns random integer in range 0 to (num-1)

Trying It Out

- **Die class has no main method.**
- Best is to write another class that instantiates some objects of your new class and tries them out.
- Sometimes called a “tester” or “testbench”

Implementing RollDice

```java
class RollDice
{
    public static void main ( String [] args)
    {
    }
}
```

Questions?