Designing New Classes (cont.)

Lecture 12

Borrowing from slides by Alan Hu, Kurt Eiselt, Paul Carter, and Tamara Munzner

Announcements

- 1st Midterm Exam: Wed, Oct 7, 6:30 pm
- Locations:
  - Henning 200, 201, 202
  - Woodward 4, 5
- Room allocation by last name
  - List of names vs rooms is online (WebCT and section web page)
- Material: Chapters 1-4

Reading Assignments

- Reading for this week:
  - Edition 2: Ch. 6.1-6.4
  - Edition 3: Ch. 5.1-5.4

Objectives for Today

- Design, implement, and try out another class
- Gain a lower-level, operational view of how method calls work.
- Understand class and local scope of declarations.

Recap: Control Flow Between Modules

Client Code

double mpg;
MileageComputer foo =
   new MileageComputer();
foo.addData(1.0,0.1);
foo.addData(1.0,0.1);
mpg = foo.avgMileage();
foo.addData(1.0,0.1);
foo.reset();

Class Methods

public MileageComputer() {
   ...
}
public void addData(...) {
   ...
}
public double avgMileage(...) {
   ...
}
public void reset() {
   ...
}

Recap: Formal vs. Actual Parameters

- **formal** parameter: in declaration of class
- **actual** parameter: passed in when method is called
- variable names may or may not match
- Java uses call by value:
  - Value of actual parameter copied into formal parameter when method is called
- For primitive types, changing formal parameter inside method body does not change actual parameter value outside
- What if parameter is an object?
Recap: Data Flow Between Modules

Client Code

```java
foo.addData(1.0, 0.1);
```

Class Methods

```java
public void addData(double miles, double gas)
{
    totalMiles = totalMiles + miles;
    totalGas = totalGas + gas;
}
```

Introduction to Scope

- Private fields and methods of class have **class scope**: accessible anywhere in class
- Parameters of method and any variables declared within body of method have **local scope**: accessible only to that method
  - not to any other part of your code
- In general, scope of a variable is block of code within which it is declared
  - **block** of code is defined by braces `{ }`

Scope – Variable Types

- Variable types (cont):
  - Fields – variables belonging to an object
    - Exist as long as the object they belong to
    - Can be accessed from any method in object’s class
      - (or by everybody if declared public)
    - Values assigned by constructors and mutator methods
      - (or by everybody if declared public)

Scope – Variable Types

- Scope gives rise to different variable types
  - Local variables – variables defined within the body of a method
    - Exist while method is being executed
    - Can be accessed from within method body
    - Values are assigned inside method body
  - Formal parameters of method
    - Exist while method is being executed
    - Can be accessed from within method body
    - Values assigned externally by calling method

Another Example Class

- BankAccount
  - Want to be able to
    - Create account with name, initial balance
    - Query the name of the account holder
    - Query the balance
    - Deposit a certain amount
    - Withdraw a certain amount
    - Transfer a certain amount from one account to another
### Cleanup Pass
- Would we hand in our code as it stands?
  - good use of whitespace?
  - well commented?
    - every class, method, parameter, return value
  - clear, descriptive variable naming conventions?
  - constants vs. variables or magic numbers?
  - fields initialized?
  - good structure?
  - follows specification?
- Ideal: do as you go
  - commenting first is a great idea!
- Acceptable: clean up before declaring victory

### Commenting Code
- Conventions
  - explain what classes and methods do
  - plus anywhere that you've done something non-obvious
    - usually better to say why than what
  - not useful
    - int wishes = 3; // set wishes to 3
  - useful
    - int wishes = 3; // follow fairy tale convention

### javadoc Comments
- Specific format for method and class comments
  - running javadoc program will automatically generate HTML documentation
- Rules
  - /** to start, first sentence used for method summary
  - @param tag for parameter name and explanation
  - @return tag for return value explanation
  - other tags: @author, @version, etc.
  - */ to end
- Running
  - % javadoc Die.java
  - % javadoc *.java

### javadoc Method Comment Example
```java
/**
 * Sets the die shape, thus the range of values it can roll.
 * @param numSides the number of sides of the die
 */
public void setSides(int numSides) {
    sides = numSides;
}
```

### javadoc Class Comment Example
```java
/**
 * Die: simulate rolling a die
 * @author: CPSC 111, Section 206, Spring 05-06
 * @version: Jan 31, 2006
 * This is the final Die code. We started on Jan 24,
 * tested and improved in on Jan 26, and did a final
 * cleanup pass on Jan 31.
 */
```

### Change of Gears: Conditional Statements
- So far, execution goes line-by-line in order:
  - int x = 3;
  - int y = 5;
  - int z = x + y;
  - System.out.println("The answer is " + z);
- One of the powerful ideas that distinguishes a computer is the ability to make a decision:
  - Decide to do something or not, depending on what’s been computed.
Java's if statement

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

- Examples:
  ```java
  if (tax < 0) tax = 0;
  ```
  ```java
  if (age < 19)
    System.out.println("Sorry, you may not buy alcohol.");
  ```

Kurt's Example

```java
import java.util.Scanner;
public class Feelgood {
  public static void main(String[] args)
  {
    int age;
    Scanner scan = new Scanner(System.in);
    System.out.println("Enter your age: ");
    age = scan.nextInt();
    if (age < 20) System.out.println("Really, you look like you are "+ (age + 5) + ".");
    System.out.println("You don't look a day over "+ (age - 10) + "!");
  }
}
```