CS 1331 Homework 2

Due Thursday September 6, 2012 8:00PM

Introduction

In this assignment we will begin to require you to comment your code, gradually building up the complexity over the semester. For homework 2, you will start to "javadoc" your code. Javadoc is an application that is part of the JDK that you downloaded. You can run it on the command line by typing "javadoc *.java". This will create documentation for all the classes in your current directory. If you are using an IDE, there is usually a menu option to generate your javadocs. Javadoc creates the nice web-page based documentation that we have been using in class (the API).

This homework covers Scanner and OOP. You will create a program that breaks some dollar amount into monetary increments. Next, you will create your first Object Oriented Program which represents a cube and write a driver class to test it.

Also, remember there is no collaboration. Your work is to be your own.

2.1: Javadoc Commenting

Javadoc comments are a special form of comment. Javadoc recognizes a comment that starts with /** as a javadoc comment. For each new program assignment we will introduce new javadoc features. For homework 2 we will create class header comments.

The class header

Start your class header with the javadoc comment /**. Then write a short description of the class. Note that each line starts with an asterisk (*) and space. End the header comment with */. For example:

```
/**
 * This is HW2, problem 1.
 * This class rids the Earth of bad guys.
 */
```

Javadoc class header comments should be placed below any import statements and right above the class header itself:

```
import javax.swing.JApplet;

/**
 * This is HW2, problem 1.
 * This applet draws some shapes and text.
 */

public class MyApplet extends JApplet {
```

Note that javadoc comments are different from the commenting style given in the book. They *must* begin with /** (not /* or a slash followed by a line of dashes) and should go between the imports and public class line. Little to no credit will be given for comments that imitate the style of the book (since it is not proper javadoc).

Commenting

You should also place good comments into your code when the algorithm you are using is not obvious. For example:

```
int perimeter = a + b + c;
```

A good comment might be:

```
// calculate the perimeter by summing the lengths of the sides
```

A bad comment would be:

```
//add a, b, and c together
```

A frequent syntax problem developers have is mismatched braces. One commenting technique that can really help you is to comment the close brace with a short description of what it is enclosing. For example:

```
    } // end if statement
  } // end main method
} // end class Demo
```

We'll be looking at more javadoc and commenting issues in future homeworks. For now, if you have more questions, you can talk to your TA.

2.2: ChangeCalculator.java

Write an application that prompts for and reads a double value representing a monetary amount. Then determine the fewest number of each bill and coin needed to represent that amount, starting with the highest. Assume that a twenty dollar bill is the maximum size needed. Name your class `ChangeCalculator`. Be sure to javadoc your class with a brief description.

As an example, the user is prompted with “Amount: $”, types 34.43, and hits enter:

```
Amount: $34.43
1 twenty dollar bills
1 ten dollar bills
0 five dollar bills
4 one dollar bills
1 quarters
1 dimes
1 nickels
3 pennies
```

Your program does not need to handle invalid input, that is, input that is negative or exceeds two post decimal digits. All input and output should be done on the command-line.

Since the algorithm you implement will likely involve division and subtraction, it is possible to lose precision from working with doubles. This could result in your program being off by one penny. While we will not deduct points for this, but there is a simpler way to complete this assignment that will eradicate this problem (think modulus).

2.3: Product Advertisement Applet

Create a java applet that draws an advertisement for some real or fictitious product. Name your applet `AdvertisementApplet`. The applet can be as simple or as elaborate as you want, though there are some requirements:

- Use at least three different drawing methods (excluding drawString) in the applet. Filled and unfilled count as two separate methods, so drawRect and fillRect (for example) count as two different methods. ‘drawLine’ also counts as a valid method. There are other methods in your textbook and the API.
- Include some text somewhere in the applet (like the product name or slogan). Remember, drawString does *not* count towards your three drawing method requirement.
- Use at least three different colors in your applet (the default white background doesn't count).
- At least one of the three colors must be a custom color, that is, it should not be one of the colors that the Color class provides.
- Do not use external images--the art in this applet should be all programmed by you
- The size of the applet should be something reasonable, and fit on a typical screen.

- Be sure to javadoc your applet with a brief description.

When you've finished your applet, create the html file to display it. Call the file "AdvertisementApplet.html". The code for creating this page is in your book and was covered in lecture. Make sure you edit the file in plain text (ie, Notepad for Windows).

Do not use a web-site development program to create your html file. You must write the html file manually using a text-editor and not a program like Word where you simply save the file to html format. Use of html generating software will result in point deduction.

Note for Mac users: If you're using TextEdit to create your .html file, be sure you are editing in plain text mode:

1. Open TextEdit
2. Click Format -> Make Plain Text.
3. Type in the html code
4. Click save
   (a) Type in a filename with .html on the end
   (b) Click save
   (c) Click "Use .html"

When you are done writing the java and html file, make sure your applet is displaying properly. We will be grading your applets from the command line, so to test your applet properly, execute:

```
javac AdvertisementApplet.java
appletviewer AdvertisementApplet.html
```

Just because the applet works with your browser, does not mean it works from appletviewer. Most browsers fix some html errors, while appletviewer does not.

For the ambitious: You can modify the size and style of your drawn text with the following command:

```
g.setFont(new Font(“Times New Roman”, Font.BOLD, 24));
```

The first field in the Font constructor is the name of the font, the second is the style (i.e. bold, italic, plain, etc.), and the third is the size. See http://download.oracle.com/javase/6/docs/api/java/awt/Font.html for more details.

Hints:

API:

- Java has a great resource called the Java API. It is a database that contains information for all Java classes, such as their constructors, methods, constants as well as descriptions and uses of each.
- The API can be found here: http://docs.oracle.com/javase/6/docs/api/ or you can just google “Java 6 API” (or Java 7 API depending on which version you are using)
- Sidenote: The API is created from javadocs, and you can use the javadoc program to generate your own API html files from your source code.

Scanner:

- Scanner is an extremely useful program used to take in input. There are several streams that Scanner can read from, but we are only interested in reading data from the command-line at this point.
- To use Scanner, you must **import java.util.Scanner;** Import statements should always be the first thing in a java file, before the actual class declaration itself. See the javadoc example at the beginning of this assignment.
- Scanner has several useful methods to read input. For a more comprehensive list, look up the Scanner class in the API – you can simply google “Java 6 Scanner” and it should be the first result.

Turn-in Procedure

Turn in the following files on T-Square. When you're ready, double-check that you have *submitted* and not just saved as draft.

- AdvertisementApplet.java
- AdvertisementApplet.html
- ChangeCalculator.java

All .java files should have a descriptive javadoc comment.

Verify the Success of Your HW Turn-In

Practice "safe submission"! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.
3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
4. Recompile and test those exact files.
5. This helps guard against a few things.
    a. It helps insure that you turn in the correct files.
    b. It helps you realize if you omit a file or files.** (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)
    c. Helps find last minute causes of files not compiling and/or running.

**Note: Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework (past the grace period of 2 am) will not be accepted regardless of excuse. Treat the due date with respect. The real due date and time is 8 pm Thursday. Do not wait until the last minute!