

CS1322 Homework 5

Due: Thursday, February 17 @ 6:00 pm

Introduction

In this homework, you will continue to develop programs with multiple classes. You should be thinking about how to design classes well using modular decomposition, encapsulation, and other object-oriented principles. You also will begin to use conditional and iteration statements.

General

On the last couple HW assignments, you began to learn about javadoc. More specifically, in the last homework you learned about how to do javadoc for methods. Now you will learn a couple of additional tags for class headers. In the class header you must add two tags after you write the description of the class.

The first tag is `@author MyName gtnum`. This identifies the person who wrote the class. It may seem unimportant to you now as a comment, but in a real software project, it is important to know who wrote and maintains a specific class in the application. (We already touched on this one in HW3, but it's good to review.)

The second tag is `@version versionnum`. This identifies which version of the class is present. Again, it may seem unimportant, but in the real world, with many people potentially changing the code, it can become confusing which version of a class someone is using. Each project will have its own numbering systems for versions. In this class, you can just make your initial version of a class 1.0. As you make subsequent changes to the class, the version number may change. Smaller changes modify the decimal place (version 1.1, 1.3) and larger changes modify the integer part (version 2.0, 5.0). If you have multiple versions of a class for 1322, use whatever numbering system seems useful to you. If you have only one version, then just put 1.0 as your version.

For example:

```
/**
 * This is a description of the entire class that may be some length.
 *
 * @author Mia Hamm gtg987s
 * @version 1.0
 */
```

By convention the first line of a javadoc comment is always a summary sentence. Other sentences can then be included to detail the desired description.

Important javadoc note: From now on, we will relax our requirement that you write javadoc comments for all your methods. While this may be important when producing large class libraries for external use, adding javadoc comments to a one-line accessor or modifier method

does seem like overkill on a class assignment. So for future assignments, you only need to add javadoc comments to “important non-trivial” methods.

Assignment 5.1: Family Planning

How many children must a couple have in order to make sure they have at least one boy and one girl? In this program, you will develop a simulation that will help you estimate the answer to that question. Your objective is to model a couple that continues to have children and only stops once they have at least one boy and one girl. Assume that for any particular childbirth, the chances of having a boy or a girl are even. Model this with a simple coin toss (i.e., use `Random()`).

More specifically, write a program that prompts the user for the number of couples in the simulation. The program then should simulate each couple’s child birth pattern, printing out the order of boys and girls born. Once the requisite number of families has been simulated, print out both the average and maximum number of children necessary to achieve the problem constraints.

After that, prompt the user whether they want to run another simulation and continue in the same fashion until the user decides to quit.

A sample run might look like the following:

```
c:\java\hw5\> java FamilySimulator
Ready to run a family simulation. Enter the number of families:
4
Simulating Families
1 - BBG
2 - GB
3 - GGGB
4 - BG
The average number of children was 2.75 and maximum was 4.

Would you like to run another simulation? (y/n)
Y

Ready to run a family simulation. Enter the number of families:
25
1 - GGB
2 - BG
...
```

Notes & Hints:

- You can assume that the user will only enter an integer for the number of families.
- That said, the user may enter an unallowable integer, and may respond with an inappropriate character to the continue question. Your program should handle these situations well.
- Do NOT code this entire program in `main()`. Think about a good program design and the set of classes you might create.
- Try a big number like 100 and see what your answers are.

To turn in:

FamilySimulator.java

All the other classes you create for the assignment.

Assignment 5.2: Space – The Final Frontier

In this part of the homework you'll be making an application that displays a space scene within a window. A space scene minimally consists of the following:

- Some GUI components (described below) at the top of your main window.
- A main graphics area (size 600 x 600) with a black background.
- A random number (between 20 and 50) of stars. Each star should be a white circle with a diameter of 8. A text string in the bottom left corner of the space scene should tell the number of stars currently being shown.
- A planet represented by a circle of diameter 40. The position and color of the planet is user controlled.

The program should have 5 text fields at the top. Two of the text fields will determine the x,y coordinates of the center of the planet. The other three text fields will determine the color of the planet, with components for red, green and blue that, when valid, are integers between 0 and 255. Your program should also have a push button. The user can enter and change numeric values as often as s/he wants, but the scene should only change (and reflect the current text values) when the user clicks the push button. When the button is pushed, a new number of stars with new positions should be generated as well. This means that you do not need to save the individual star's positions anywhere. Now that you know conditional statements, think about what happens (and what you should do) if the user enters invalid numeric values in the text fields. Handle this intelligently, but you don't need to worry about deeper errors such as letters/text strings being entered there.

You'll need at least three classes to accomplish these goals:

`Space.java` - the main class, that holds the JFrame that is the actual window of the application. The design of the Space class is fairly straightforward. Follow the examples at the end of chapter 4. This should be a minimal class with the little bit of work being done in the main method. Instantiate the proper GUI components and set up the program.

`SpaceCanvas.java` - extends JPanel. Clearly, it will need to have a constructor and an important `paintComponent` method that coordinates much of drawing and graphics.

`Planet.java` - a class that keeps track of all the information for a planet in the space scene. The Planet class should have instance data for all the important attributes of the planet: position, size, color, etc. It will need a constructor, accessor and modifier methods, a drawing method, and any other methods that are useful. You'll also need a class to handle the event processing. Hint: Look up the constructor for the Color class that takes three integers (R,G,B).

For this assignment, you may need to use the JPanel method `repaint()` that will force a redraw of the panel and hopefully its contents. Think carefully about the two different kinds of redraw

occurrences that may occur: one driven by the user clicking the push button and one driven by simply needing to refresh the application (e.g., minimizing and maximizing the window). The number of stars should only be changed to a new random value when the push button is hit. When the window is refreshed without the button pressed, the number of stars should stay the same but it's OK if their positions change.

Note that the performance of this program will be a little “weird” or may not behave correctly in the sense that anything causing the window to refresh, such as moving another window across part of the front of it, will cause `paintComponent()` to be called and the scene to be redrawn. You may even see strange update patterns. Think about why that is occurring and if there is some simple way to remedy it. Remember though that it's OK for the stars to move when the window is refreshed in this assignment (i.e., you should not store the stars in instance variables) but the number of stars should only change when the button is pressed. In the future we'll learn about other java constructs that will help us to store more data about the stars and refresh the display without any changes.

The above description describes the minimal program design that will achieve full credit for the problem. For those of you who are a little more ambitious, feel free to add other space objects, fancy graphics, more controls, etc., but do make sure that the planet and stars function as in the description above. Designs that are particularly creative and illustrate the many concepts that we've learned in class recently may potentially earn extra credit points.

To turn in:

`Space.java`
`SpacePanel.java`
`Planet.java`

All the other classes you create for the assignment.