07.05 Iterative Structures II Assignment Instructions

Purpose: Write an Object-Oriented Program to animate an object using turtle graphics methods.

Materials: index cards, paper, pencils, colored highlighters, algorithmic thinking cap

Exercise: Trace the flow of control throughout this program. Notice how the **do-while** loop performs, as well as the overall OOP design of the program.

Proposed Output:
It is good programming practice to begin with the end in mind. In other words, what should this program do when it is finished? When the program runs, a graphic object should appear to move across the screen. If the object moved in slow motion you would actually be able to see that it is not one, but two slightly different objects. To give the appearance of motion, after the first object is displayed it is wiped out by a solid rectangle the same color as the background. Then the second object is displayed and also quickly wiped away. Repeating this sequence gives the impression of motion.

Stage 1: Pre-Designing Objects that Animate

1. Use graph paper to draw a simple geometric shape such as a triangle or a rectangle. Label the positions of important coordinates. Make choices about such things as pen width and color at this time.
2. Draw a second geometric shape that is slightly different in size or shape from the original object. Your goal should be to eliminate as much guess work as possible so that when you start coding you will use your time efficiently.
3. Draw a third object big enough to cover up the first two objects. This can usually be something as simple as a solid white box that makes the other two objects seem to disappear.

Stage 2: Designing the Program

1. List the attributes of your objects (e.g., x- and y-coordinate positions, colors, dimensions of shapes). These nouns will be your variables, so decide on their types ahead of time.
2. List the behaviors your object will exhibit (e.g., move, expand, change color, etc.).
3. Write a preliminary outline describing how you think the program should be organized based on OOP design principles. All of the methods will be placed in a class called **Animation**. All of the statements that invoke the methods will be placed in a class called **AnimationTester**.
4. There is no need to be overly concerned about details at this stage; the outline is only a first stab at the design. Think about what needs to be in each class, the order variables need to be declared and assigned, and the names of objects that will be constructed.
5. Your objects will be moving horizontally across the screen, so go ahead and decide about the starting position, ending position, and the movement increment. Also decide on which kind of loop(s) you want to use to move the objects.
6. Before writing any code, you may want to put your plan aside for a little while and do something else. Taking a break will probably allow you to return to the project with fresh ideas that will save you time and effort. Make any revisions to your outline as needed.

Stage 3: Planning the Program Details

1. Think about the details of what will be in each method. The sub-headings in your outline can probably be turned into method headers.
2. Start by writing just the method headers of the **Animation** class at the top of an index card. Each time you write a header, write the corresponding invoking statement for the **AnimationTester** class on a sheet of paper. List any variables or values that will be passed as parameters.
3. Arrange the index cards in the order they will appear in the **Animation** class. Beneath each header, jot down the code for each method in as much detail as you can.
4. At this stage of the planning you may want to color code the variables on the index cards with the corresponding variables or values on the sheet of paper where making notes about the **AnimationTester** class.

Stage 4: Writing the Program

1. If you followed the algorithm presented for this assignment, most of the hard work is behind you, so let the fun begin.
2. Do not be in a rush as you write the program. Write an invoking statement in the tester class and the corresponding header in the **Animation** class. Write small portion of the method then compile and run the program to catch errors early.
3. As you alternate between writing and testing code in the two classes the program will slowly grow until the final product is successfully completed.
4. Use the Java style conventions to make sure your program is well organized and easy to read.
5. Include comments in the program to document the purpose of each method and each variable.
6. Write a thoughtful PMR based on the required and optional questions listed in the rubric.
7. Answer the Analysis Questions.
8. Submit the program, the PMR, and the Analysis Questions for a grade.

Note: You may have noticed when you run a **Turtle** graphics program that all you see is the final product. The compiler runs so fast that it is impossible to see motion. We can solve this problem by pressing a key (e.g., the c key to continue) and stepping through the animation one scene at a time. Feel free to adapt the demo program presented earlier in this lesson for this purpose. Using this technique will simultaneously display the terminal window and the graphics window causing the computer to pause and wait for your keyboard input. If you line both windows up side-by-side, you can watch your objects move across the screen by repeatedly pressing the “c” key. It will probably be easiest to place the demo code in the **main()** method and the invoking statements inside the body of the loop.

Analysis Questions: Submit your answer to the following along with your program and the PMR.

1. How could the animation be made more realistic?
2. What question(s) of your own did you answer while writing this program?
3. What unanswered question(s) do you have after writing this program?

Print