Assignment A5 CS1110 Spring 2011 Turtles!

Due on the CMS by 11:59pm Thurs, 31 March.

Purpose of assignment and basic ground rules

This purpose of this assignment is to

- Introduce you to graphics
- Give you practice with simple loops
- Give you practice with recursion
- Give you practice with helper methods and using previously written methods

Read this entire document carefully. At the end of this document, we tell you what to submit on the CMS. Budget your time wisely. Don't wait until the deadline to start this assignment. We advise starting now and working on one method a day. You may also want to experiment, drawing your own graphics designs.

You may work with one other person. If you do so, FORM YOUR GROUP ON THE CMS WELL BEFORE YOU SUBMIT YOUR FILES. Remember, partners must work together and not independently.

Keep track of the time you spend on this assignment. You will be asked to put it in a comment at the top of file `A5.java`.

You need not use a JUnit testing class. You will (mostly) be looking at visual output (graphics) to determine correctnesss.

To save you time, we give you complete specifications of most of the methods you write. Study them carefully. Note how precise and thorough they are. You should aim for this quality of specs when you write your own specs.

Note on DrJava. Please do this immediately: In DrJava, use menu item `Edit->Preferences`; click `Interactions Pane`; and make sure the `Require variable type` box is unchecked. This will allow you to use variables in the Interactions Pane without declaring them. Also, change the indent level to 4 (Preferences category `Miscellaneous`).

Academic integrity

You may, of course, talk in general terms about problems and issues. But to look at or be in possession of someone else's code for this assignment, from a previous semester or this semester, on paper or in electronic form, or to give someone else in the class your code is a violation of the code. It is stupid to copy because you don't learn anything. Moreover, forcing us to grade something that is not yours wastes our time.

This assignment is similar to one given in the past several semesters. Anyone caught using files that are obviously from a previous semester will be prosecuted, with the end result perhaps being to fail the course.

Directions

Download file A5.zip, unzip it, and *put everything in it into a new directory*. It contains:

1. File `A5.java`

2. File `HSV.class`, a simple implementation of the HSV color model (see A4).

3. Package `acm`. It contains a directory of other packages, each of which contains a directory of .class files. These are machine-language versions of .java files. Do NOT load them into DrJava. The only thing you should load into DrJava is file `A5.java`. It will automatically use the .class files.

4. A directory `doc`, which contains specs of all the classes in package `acm`. Double-click `doc/index.html` (or open it in a browser) to see the specs, and refer to them when writing method calls to draw things on the "graphics canvas". Spend some time looking at the methods that are available in classes `GraphicsProgram`, `GTurtle`, and `GPen`. This version of the specs omits some information. If you are curious, you can view the complete documentation here: jtf.acm.org/javadoc/complete/.

Package `acm` was developed under the auspices of the ACM (Association for Computing Machinery), the major CS professional society. Class A5 is a subclass of abstract class `GraphicsProgram`, which is part of package `acm`. A `GraphicsProgram` is associated with a window on your monitor (much like a `JFrame`) that contains a "canvas" on which one can draw. When an instance of `A5` is created, the associated window appears on your monitor. You can then create "turtles" and "pens" to draw (on the canvas) in that window.

An instance of class `acm.graphics.GTurtle` maintains a pen of a certain color at a pixel `(x, y)` that is pointing in some direction given by an angle (0 degrees is to the right, or east; 90 degrees, north; 180 degrees, west; and 270 degrees, south). When the turtle is moved to another spot using procedure `forward`, a line is drawn if its pen is currently "down" and nothing is drawn if its pen is "up". The pen is initially black, but its color, of class `java.awt.Color`, can be changed. A footnote on page 1.5 of the ProgramLive CD contains information about class `Color`.

By following the directions above, you have already looked at `doc/index.html` and studied the specifications of methods in class `GTurtle` as given in the javadoc files. Here are some important points:

- Before an instance of `GTurtle` can be used to draw something, it has to be added to an `A5` object. You can add many turtles to the same `A5` object and draw different things with each. Suppose `x` contains the name of an `A5` object. Function `x.getTurtle()` --already written-- creates a new turtle and adds it to object `x`. Study the function body; make sure you understand what it does and how.
- The coordinates and angle of the turtle are maintained using type **double.** This is needed for maximum accuracy. If **int**s were used, errors might crop up after many calculations. However, whenever a point is to be placed in the window, its x- and y-coordinates are *rounded* to the nearest integer because the graphics space works with **int**s.
- Suppose `t` contains a `GTurtle. Then t.forward(d)` moves the turtle `d` pixels in its current direction, `t.setLocation(x,y)` moves `t` to pixel `(x,y)` without drawing anything, `t.left(a)` turns the turtle `a` degrees counterclockwise, and `t.setDirection(a)` sets its direction to angle `a`.
- The direction may be negative or greater than 360. Adding a multiple of 360 doesn't change the direction the turtle faces, e.g. for angles -360, 0, 360, and 720 the turtle is facing east.
- `t.setDirection(angle)` does not redraw the turtle. To see this, in the Interactions pane, type `a= new A5(); t= a.getTurtle(); t.setDirection(90)`. To repaint the screen so that the turtle faces the right direction, execute `a.repaint()`.
- A turtle `t` moves with a speed, which must be in the range 0 <= speed <= 1 ---0 is the slowest and 1 the fastest. Not much detail on the speed is given, so we can't tell you more. Set `t`'s speed to `sp` using `t.setSpeed(sp)`.

Getting started and completing Tasks 1..5

Class `A5` contains procedure `drawTwoLines` to show you how graphics work. Compile class `A5`. Then, in the Interactions pane, create an instance of class `A5` using `a= new A5();` . The window will appear. Execute the call `a.drawTwoLines(0);`. It causes two lines to be drawn, waiting a few seconds before drawing each one. Study the body of `drawTwoLines.`

In the Interactions pane (or in a method in class `A5`), draw some lines and rectangles to familiarize yourself with class `Turtle`. After that, perform the tasks given below. As you write a method body, refer constantly to the method specification and follow it carefully and rigorously. If you have to call another method, look at its specification and make sure you follow it. A huge number of programming errors arise from not following specifications carefully.

Task 1. Complete function `toString(GTurtle)` in `A5`. Follow the instructions given in the function itself. Below are two examples of what our `toString` function produces, and yours must be the same. The second one has RGB color (6, 7, 8).

> "GTurtle[location=(250.0, 250.0), color=GREEN, direction=270.0]"
> "GTurtle[location=(250.0, 250.0), color=0x60708, direction=180.0]"

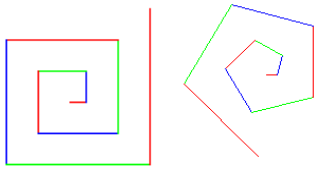
Task 2. Complete procedure `drawTriangle`. Then follow the instructions in the comment in the body to learn about rounding errors when using type **`double`** and why they don't really matter here. The comment asks you to put some information at the top of file A5.java (as a comment).

Task 3. Complete procedure `drawHex` to draw a hexagon, as shown to the right of this paragraph. Follow the specification and hints carefully.

Task 4. Do TWO (repeat, 2) of 4A, 4B, or 4C. After you have completed the assignment, if you are interested, do the remaining one! It is fun to see how easy it is to do these neat things.

Each of these tasks involves creating a helper function with the same name. The first procedure does not have a turtle as parameter; it clears the canvas, creates a turtle, calls the second procedure to do the work, and hides the turtle. This allows one to draw a spiral, for example, without having to create a turtle first. When writing these procedures, write the first one, write the second one, and then test both by using calls of the first one in the interactions pane.

Task 4A: Draw a spiral. The first picture to the right is done by drawing 10 lines. The first line has length 10; the second, 20; the third, 30; etc. After each line, the turtle turns left 90 degrees. The second diagram to the right shows a similar spiral but with the turtle turning left 75 degrees after each line.

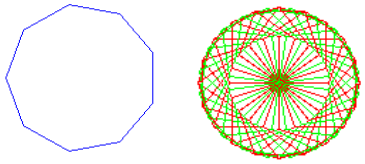


Complete the TWO procedures named `drawSpiral`. When you first test them, use 10 for `d` and 0 for `sp`. Try different angles, like 90 degrees, 92 degrees, 88 degrees, etc. Use sp = 0, or sp = .5 to see the lines drawn one at a time.

You will be amazed at what these methods do. Find out by trying these calls, assuming that `a` is an instance of `A5`:

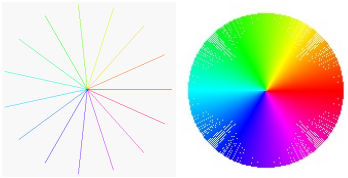
```
a.drawSpiral(.8,1,90,300);   a.drawSpiral(.85,1,135,400);  a.drawSpiral(.95,1,60,100);
a.drawSpiral(.9,1,121,300);   a.drawSpiral(1,1,89,400);     a.drawSpiral(1,1,150,300);
a.drawSpiral(.99,1,121,500); a.drawSpiral(1,1,119,500);
```

Task 4B: Draw many polygons. The first image to the right is a 9-sided polygon. The second image to the right is a series of 40 5-sided polygons of side length 35, the first started at angle 90, the second at an angle of 90 + 360.0/40, the third at an angle of 90 + 2*360.0/40, and so on.



Complete the TWO `multiPolygons` procedures so that your program can draw such designs. Use procedure `drawPolygon`, which we give you. When finished, experiment to see what neat designs come out. Try, for example, `a.multiPolygons(45,3,100,.8);` and `a.multiPolygons(60,30,20,.88);`.

Task 4C: Draw radiating lines. The leftmost picture to the right is done by drawing 15 lines of the same length, radiating out from the current turtle position. The angle between the lines is the same. The second picture has 720 lines. If n lines are drawn, the angle between them is 360.0/n. The color of each line depends on the angle (i.e. the direction) of each line. A line drawn at angle `ang` is drawn with HSV color `(ang,1,1)`. But the HSV value has to be translated to the RGB system, because RGB is used by the turtle graphics system. So, we give you HSV.class, which allows you to create an object of the class and also contains a function that will translate an HSV value to the RGB system. See the note in procedure `A5.radiate`.



Complete the two `radiate` procedures. When finished, test them with small values of n, like 4 or 8. After the procedures are completely tested, try them with 360 lines of length 200, with turtle speed .85. Isn't that neat? Also, do it with 3,000 lines and turtle speed 1; notice how much more filled in the disk becomes.

Task 5. Recursive graphics. We ask you to develop recursive procedures to draw some graphics, recursively. Do **two** of them: Sierpinski triangle, the Grisly snowflake, and the H-tree (but if you like this stuff, do all three!)

A `GTurtle` maintains a position and a direction, and when you ask it to draw using `forward(d)`, it draws a line of length `d` in that direction. A `GPen` (also in package acm.graphics) maintains a position but no direction. Use procedure `drawLine(dx,dy)` to draw a line of length `(dx,dy)` from the pen's current position, ending up at the end of the drawn line, `move(dx,dy)` to do the same thing but without drawing, and `setLocation(x,y)` to move to position (x, y) without drawing. A `GPen` has other useful methods. Spend 5 minutes looking through their specifications.

We use a `GPen` instead of a `GTurtle` in task 5 because (1) there is no need to maintain the direction and (2) `GPen` methods exist to draw regions that are filled in with a color.

Again, for each of these recursive tasks, write *two* procedures: the first does not have a pen as a parameter and the second does.

Task 5A. Sierpinski triangles. Directly to the right is a filled-in equilateral triangle. We call it a Sierpinski triangle of size `s` (the length of a side) and depth 0.

Next to it is a Sierpinski triangle of size `s` and depth 1. It is created by drawing 3 Sierpinski triangles of size `s`/2.0 and depth 0, in each of the corners of what would be a Sierpinski triangle of size `s` and depth 0. All the way on the right is a Sierpinski triangle of size `s` and depth 2; it is created by drawing 3 Sierpinski triangles of size `s`/2.0 and depth 1.

In the same way, draw a Sierpinski triangle of size `s` and depth `d` (> 0) by drawing three Sierpinski triangles of size `s`/2.0 and depth `d-1` in appropriate places.

We have stubbed in two `sierpinski` procedures for you to complete. You can use procedure `fillTriangle` to draw a triangle —it is needed only at depth 0.

The most difficult part may be finding the height of the triangle with side length `s`. Knowing that it is an equilateral triangle, use the Pythagorean theorem to figure this out. Using `h` for the height, visualize a triangle that is 1/2 of the equilateral triangle, with side lengths `s`, `s/2.0`, and `h`. Solve the formula `s**2 = (s/2)**2 + h**2` for `h`.

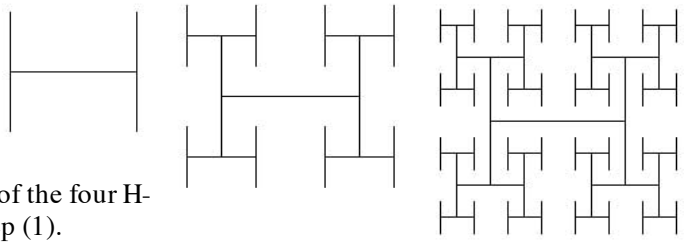
Task 5B. Grisly snowflakes. To the right are three Grisly snowflakes of depth 0, 1, and 2. The depth 0 snowflake is simply a filled-in hexagon. The depth 1 snowflake of side length s is the depth 0 snowflake of side length s replaced by 7 hexagons of side length s/3.

In general, for d > 0, the Grisly snowflake of side length s and depth d consists of 7 Grisly snowflakes of side length s/3 and depth d-1, drawn at the obvious places.

As with Sierpinski triangles, the difficulty will be figuring out where each of the 7 Grisly snowflakes of depth d-1 go. We leave you to try to figure it out but will put some explanation later on on the course website.

We have stubbed in two `grislySnowflake` procedures for you to complete. The first does not have a pen as a parameter; the second one does. We have also written procedure `fillHex` for you.

Task 5C. H-trees. To the right are three H-trees of size `s` and depths 0, 1, and 2. Here's how you draw them --implement this set of instructions as given and you will have no trouble:



(1) Draw an H, with all three lines being of length `s`.

(2) If `d>0`, draw four H-trees of size `s/2` and depth `d-1`. The centers of the four H-trees are at the top and bottom of the two vertical lines drawn in step (1).

H-trees are useful in designing microprocessor chips. The lines are wires that connect circuit components in a tree of interconnections, without wires crossing.

We have stubbed in two procedures `Htree` for you to complete. The first does not have a pen as a parameter, the second one does.

We have also stubbed in procedure `drawH`, which may be useful to you. Complete it if you want to use it. Using pen `p`, draw lines using procedures `p.setLocation` (to move the pen) and `p.drawLine` (to actually draw the line).

What to submit. Put a comment at the top of your `A5.java` that contains the following information.

1. Your name(s) and netid(s);
2. The information requested in Task 2;
3. The names of your two public recursive procedures;
4. What you liked best about this assignment;
5. What you most think could use improvement in this assignment; and
6. The time you spent on this assignment.

Make sure class `A5` is indented properly and that all parts of the program can be seen without horizontal scrolling. Remove any System.out.println commands you may have put in for debugging purposes. Submit file `A5.java` on the CMS.