Assignment A5 CS100J Fall 2007 See CMS for deadline

This assignment introduces you to graphics. You will write procedures that draw interesting designs, bouncing balls, and Koch snowflakes in a JFrame. 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 should work together and not independently.

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

To save time and effort, we give you complete specifications of all methods you write. Please study them carefully. Note how precise and thorough they are. You should aim for this quality of specification when you write your own specifications. At the end of this document, we tell you what to submit.

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

1. A file `A5.java`, which extends class `Turtle`.

2. Several .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.

3. A directory `doc`, which contains specifications of class `Turtle`. You will use these specs to learn how to write method calls to make turtles do what you want.

A `Turtle` is 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 `moveForward` or `moveBackward`, 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`.

Open `doc/index.html` in your favorite browser and study the specifications of methods in class `Turtle` (the javadoc files). Here are some important points:

- Class `Turtle` uses the `Graphics` object that is attached to a `JPanel`. It builds on class `Graphics` by maintaining the "turtle", which has a position and an angle and holds a pen of a color; the pen is up or down. You can use many turtles at the same time; they all use the same `JPanel`.
- The coordinates and angle of the turtle are maintained using type **double.** This is needed for maximum accuracy. If we used **int**, errors might crop up after many calculations. But whenever a point is to be placed in the window, its x- and y-coordinates are rounded to the nearest integer because that is what the graphics space needs.
- Function `intColor` allows you to use an integer in the interactions pane to obtain an object of class Color, for various oft-used colors.
- Manually resizing the `JFrame` window also changes the size of the panel on which the drawing occurs. But you have to move the mouse inside the panel to see this.
- Procedure `jumpTo(ang, x, y)` can be used to move the turtle, without drawing, to `(x, y)` and face it at angle `ang`.
- Call `pause(p)` to pause execution for `p` milliseconds. Judicious use of this method will allow you to watch something being drawn in slow motion.

Class A5.java contains procedure `drawTwo` to show you how graphics works. After compiling class `A5`, in DrJava's interaction pane, create an instance of class `A5` and execute a call on `drawTwo`. A `JFrame` should be created and two lines should be drawn on it.

In the interactions pane (or in a method in class `A5`), draw some lines, rectangles, circles, etc, to familiarize yourself with class `Turtle`. After that, perform the tasks given below.

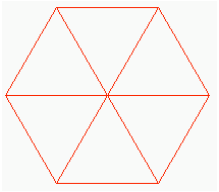
Task 1. Complete function `A5.toString`. Follow the instructions given in the function itself. You will see why in Task 2. Here is an example of what our `toString` function produces, and yours should be similar:

pos. (250, 250), facing 0.0 degrees, java.awt.Color[r=0,g=0,b=0], pen is down

Your output does not have to be *precisely* like this, but it should contain the same information. Note that class `Color` has its own function `toString`.

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.

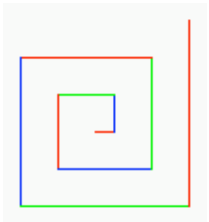
Task 3. Complete `drawHexagon` to draw a red hexagon, as shown to the right of this paragraph. It should call procedure `drawTriangle` 6 times. Some lines will be drawn twice, but that is OK. Of course, when the procedure call terminates, the turtle's properties (position, angle, pen color, and whether the pen is up or down) should be the same as when the procedure call began.

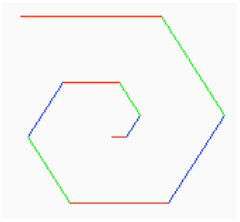


The procedure should draw the shape at the turtle's current position and at the turtle's current angle. Don't move the turtle before beginning.

Task 4. Do either 4A or 4B. After you have completed the assignment, if you are interested, do the other two! It is fun, seing how easy it is to do these neat things.

Task 4A: Draw a spiral. The first picture to the right is done by drawing 10 lines. The first line has length 5; the second, 10; the third, 15, etc. After each line, 90 degrees is added to the angle. The second diagram to the right shows a similar spiral but with 60 degrees added to the angle after each line.



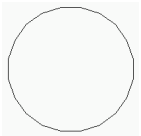


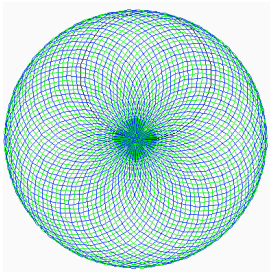
Complete procedure `spiral`. When you first test your method, use 5 for `d` and 0 for `sec`. Try different angles, like 90 degrees, 92 degrees, 88 degrees, etc. You can also use `msec` = 500 or `msec` =1000 (which is 1 second) in order to see the lines drawn one at a time.

You will be amazed at what method `spiral` does. Find out by trying these calls, assuming that `x` is an instance of `A5`. Use procedure `clear()` to erase the panel before each one:

```
x.spiral(500, 90, 1, 0);    x.spiral(500, 135, 1, 0);    x.spiral(500, 60, 1, 0);
x.spiral(500, 121, 1, 0);   x.spiral(500, 89, 1, 0);     x.spiral(500, 150, 1, 0);
x.spiral(500, 120, 1, 0);   x.spiral(500, 119, 1, 0);
```

Task 4B: Draw many polygons. The first image to the right is a 20-sided polygon, drawn using procedure drawPolygon, which we give you. The second image to the right is a series of 90 such polygons, the first started at angle 0, the second started at angle 4, the third at angle 8, and so on. The polygon colors alternate between blue and green.





Complete procedure `multiPolygonVariation` so that your program can draw such designs. When finished, experiment, to see what neat designs come out. Try, for example, `multiPolygonVariation(4, 200, 0)` and `multiPolygonVariation(7, 50, 0)`.

Task 5: Bouncing balls. Procedure `Turtle.fillCircle` draws a disk —a filled-in circle. You can use this procedure to draw a bouncing ball. Suppose the ball is at some position `(x, y)`. To make it look like the ball is moving, repeat the following process over and over again:

1. Move the ball: (1) Draw the ball at its current position using color white, thus erasing it, (2) change the position of the turtle, and (3) draw the ball in its own color.
2. Pause for 100 milliseconds.

(a) Add to class `A5` three private **double** fields: `radius` gives the radius of the ball and fields `vx` and `vy` describe the speed of the ball; when the ball is moved, it moves `vx` pixels in the horizontal direction and `vy` pixels in the vertical direction. Write getter methods for these fields.

(b) Complete constructor `A5(r, x, y, c, vx, vy)`, which initializes a new `A5` object. You know the constructor works properly when you see the ball in the panel.

To make things easier, also complete constructor `A5(r, vx, vy)`, which creates a black ball at the midpoint of the panel with speed `(vx, vy)` and radius `r`. Then, create several balls, with different starting points, radii, and colors, before going on to the next step.

(c) Complete procedure `moveOnce()`, *following these directions*:

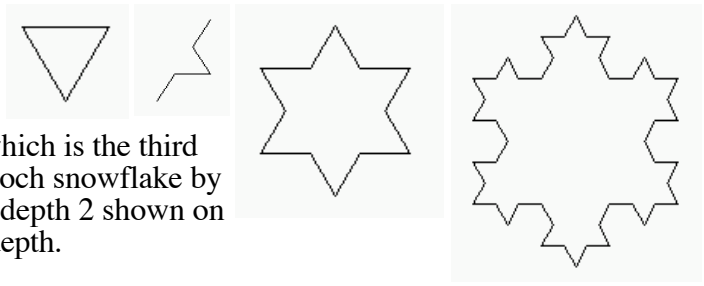
1. To move the ball: (1) draw it with a white pen; (2) move the turtle; and (3) draw the ball in its original color. So, the program must remember the original color. Have a local variable `originalColor` to contain the original color and, after drawing the ball white, set the turtle color back to the value of variable `originalColor`.
2. *After* moving the ball, if the vertical motion is negative and the ball just touches or goes partially over the top of the panel, then negate the `y`-coordinate speed (using `vy= – vy;`) so that the next move will be in the opposite direction. You have to do the same kind of thing for the other three walls, one wall at a time.

Test method `moveOnce()` carefully, as follows. In the interactions pane, create a ball of radius 45 that starts in the middle of the panel and has velocity `(0, –35)`. Then, repeatedly execute `d.moveBallOnce()` and watch what happens. Makes sure that it bounces properly off the top and bottom walls. Then do the same kind of test for the left and right walls.

(d) Complete procedure `perpetualMotion()`. Its body should be a loop that does not terminate and that has a repetend that (1) moves the ball once and then (2) pauses for 100 milliseconds. In the interactions pane, create a new `Ball d`, call `d.perpetualMotion();`, and watch the ball move. Stop this execution by hitting the DrJava reset button.

(e) Complete method `perpetual2`, which puts two balls in motion forever. In the interactions pane, call this procedure and make sure you see both balls bouncing.

Task 6. Koch snowflakes. Directly to the right is a triangle. It is called a "Koch snowflake " of level 0. Replacing each line of this Koch snowflake by four lines as drawn in the second diagram gives the Koch snowflake of depth 1, which is the third figure. And, replacing each of the lines of this Koch snowflake by the same four linesgives the Koch snowflake of depth 2 shown on the extreme right. This can be continued to any depth.



Doing this is extremely easy using recursion! Complete procedures Koch and KochL in class A5:

/** Make sure that the graphics panel is as large as possible in the window. Clear the

window, place
the turtle facing east at position (width/5, 2*height/3). Then draw a Koch snowflake of depth d with
line segment length F. Pause s milliseconds after drawing each line segment.
Precondition: d ≥ 0*/
```
public void Koch(int d, double F, int s) {}
```

/** Draw a KochL snowflake of depth d with the current turtle. Parameters d, F, and s are as in procedure Koch.
Precondition: d ≥ 0 */
```
public void KochL(int d, double F, int s) {}
```

Procedure `Koch` is the easiest. First, call `setPanelSize()` in order make the graphics panel as large as possible within the window. Then, call procedure `moveTo` to move the turtle as in the specification (and make the turtle face east). Finally, draw as shown below.

The commands to be placed in `Koch` and `KochL` are described by the following two *patterns*:

- `Koch`: `L – – L – – L`
- `KochL`: `L + L - - L + L`

Here's how to interpret each of them. Each symbol is a command to do something, as follows.

`L`: If `d` (depth of recursion) is 0, draw a line of length `F`; otherwise, call `KochL(d-1, F, s)`.
`+`: add 60 degrees to the turtle's angle.
`–`: subtract 60 degree from the turtle's angle.

So, the patterns are simply a simple, terse, way of describing what each of the method bodies should do. Such a system of patterns is called a "Lindenmayer System", after Aristid Lindenmayer, who co-authored a book titled *The Algorithmic Beauty of Plants* (Springer Verlag, 1990).

Your task, then, is to complete the bodies of `Koch` and `KochL` according to the patterns given above for them —and test and debug until they are correct. The following hint may help make things easier. In each procedure, first test whether `d` is `0`; if it is, then carry out all the commands under that assumption and return. Then, carry out all the commands under the assumption that `d > 0`.

Task 7! Do something of your choice. Make sure you say what it does in its specification. We don't care what your procedure does, as long as it is non-trivial. You could draw a face whose size depends on a parameter. You could make some interesting design with a few stars. You could write a method with a parameter n that draws an n-pointed star. You could have two bouncing balls change direction when they collide --when they occupy the same space. You could have more bouncing balls; when two collide, the smaller one blows up (goes completely off the screen). You could change the initial color of the panel to something other than white and then put a ball in motion, so that you see the path it takes. How about placing some rectangles at the top of the panel; when a ball hits them, the rectangle disappears and the ball changes direction. You could find some other interesting recursive procedure that draws something. Use your imagination. We will make the most interesting procedures available on the course website.

What to submit. Make sure class `A5` is indented properly. Add a comment at the top of it giving the name of your interesting procedure. Submit file `A5.java` on the CMS.