Assignment 9
CSc 210 Fall 2017
Programs Due November 2nd, 8:00 pm MST

Introduction

We have been discussing what inheritance is, what its purpose is, and how it is used to better the design of systems. It allows us to abstract concepts within our systems. For this assignment, you will apply your knowledge gained from lecture to actual programs.

You will turn in your code via github classroom. Here is the link to receive your repository: https://classroom.github.com/a/wFtY-ydk

For each Java program assigned, you will turn in a .java file for that program. We should be able to run "javac *file*.java" and then "java *file*" on lectura and your program should compile and run successfully.

Your Java programs should adhere to our style guidelines we will give to you. Here is the link: https://www2.cs.arizona.edu/classes/cs210/fall17/StyleGuidelines.pdf

Inheritance Shapes

For this project, you will create an inheritance hierarchy representing some shapes. Additionally, you will create a simple user interface to interact with your program and print shapes out to a screen.

First, you will create a base shape class. This base class should define common characteristics between shapes. This may include values such as fill (what character is inside of a shape), size, position of shape on screen, or anything shapes might share. You should also define common methods for shapes in your base class. Among the methods you should define in your base class are ones for

1. Drawing a shape.
2. Printing out information about the shape.
3. Getters and setters for shared variables.
4. Growing or shrinking a shape but a factor passed in as a double.

You should define several classes which inherit from the base shape class. You should have a rectangle class, a triangle class, a parallelogram class, and a hourglass class. Details of these classes are defined later.

You should also create a REPL (Read-Eval-Print Loop) to interact with your shape objects. Essentially, you will have a loop that reads input until there is no more left to be read. You will read commands in and respond to them. This REPL will read input in from standard in. You will read in commands in the following format:

- A define command will create a shape and assign it to the name given. You will need a way to store and recall the shapes defined. Each shape will have a unique "initializing function" defined later. You will find the DEFINE keyword, the name of the shape, the shape type in all caps, then the parameters for creating that shape which are given later. Note you can have redefinitions of shape names. The command will look like the following
    - *DEFINE shape_name SHAPE_TYPE ...parameters...*

- A draw command to print the shape to the screen. It will print out an ascii version of the shape inside of a "screen". This screen is a window that is 32 characters tall and 96 characters wide. This screen has borders that are the asterisk (*) character. For all x,y positions on the screen, y goes down and x goes to the right. The "pixels" in the screen will be 0 indexed. The screen size 32x96 includes the border, however you can not overwrite the pixels of the border. The size of a shape will define how you print it out. If a shape goes outside of the screen, print out the message "Shape exceeds bounds of screen" to standard output. You still must print the parts of the shape that are inbounds. The draw command will look like
    - *DRAW shape_name*

- A transform command will change the size of a shape. It will take in the name of a shape to transform and a either DOUBLE to double the shapes size, or HALF to halve it. The specifics of what doubling and halving mean are defined below.
    - *DOUBLE shape_name*
    - *HALF shape_name*

- A dump command will print out information about a shape. It will print out things like area, fill, amount of times it has been printed, and more based on what type of shape it is. The exact information to print out will be given later. The command will look like the following
    - DUMP *shape_name*

- If you read a command that does not start with one of the keywords given above, print out "Invalid command." to standard output and read the next command. Otherwise, if the first keyword matches, assume the command has correct syntax and all inputs are valid.

Here you will find some implementation details of shapes:

- Rectangle
  - You should maintain a width and a height for a rectangle.
  - The format for the initializing function of a rectangle is "RECTANGLE *int_width int_height "char_fill" int_xpos int_ypos*" where each parameter is separated by space. Note the quotes around the fill char. To double a rectangle, multiply the width and height by two. To halve it, divide the width and height by two.
  - The dump message for a rectangle should be in the following format "RECTANGLE (name:*name*) (x:*xpos*) (y:*ypos*) (width:*width*) (height:*height*) (area:*area*) (fill:*fillchar)* (draw_amount:*draw_amount*)" where the italicised values are the actual values associated with the shape. This should all be printed on one line.

- Triangle
  - You will maintain one parameter for height (height is really the length of each side in # of characters).
  - You should maintain a parameter to know if you should print the point facing upwards or downwards. If the parameter is down, you know to print the triangle with a point facing downwards. If it is up, you know to print the triangle with a point facing upward.
  - The format for the initializing function of a triangle is "TRIANGLE *int_height direction "char_fill" int_xpos int_ypos*" where each parameter is separated by space. Note the quotes around the fill char. The direction parameter is either UP or DOWN, dictating the direction for the triangle to point.
  - To double the size of a triangle, just double its height. To halve it, divide its height by two.
  - The dump message for a triangle should be in the following format "TRIANGLE (name:*name*) (x: *xpos*) (y:*ypos*) (height:*height*) (area:*area*) (direction:*dir*) (fill:*fillchar*) (draw_amount:*draw_amount*)" where the italicised values are the actual values associated with the shape. The *dir* value should be UP or DOWN. This should all be printed on one line.

- Parallelogram
  - This is essentially a rectangle, but the left and right edges of it are shifted to lean to the right. Think about how this relates to the rectangle class. What characteristics does it share? What needs to change?
  - Each layer of your parallelogram will be shifted to the right some amount. For instance, a 3x3 parallelogram would have the top layer shifted two characters right, the middle layer would be shifted one character right, and the bottom layer would not be shifted. See the example output for an example. The format for the initializing function of a parallelogram is

“PARALLELOGRAM *int_width int_height “char_fill” int_xpos int_ypos*” where each parameter is separated by space. Note the quotes around the fill char. To double a parallelogram, multiply the width and height by two. To halve it, divide the width and height by two.
    - The dump message for a parallelogram should be in the following format “PARALLELOGRAM (name:*name*) (x:*xpos*) (y:*ypos*) (width:*width*) (height:*height*) (area:*area*) (fill:*fillchar*) (draw_amount:*draw_amount*)” where the italicised values are the actual values associated with the shape. This should all be printed on one line.

- Hourglass
    - This shape is essentially two triangle, one on top of another (How could you use this fact to simplify the hourglass class?).
    - The format for the initializing function of a hourglass is “HOURGLASS *int_height “char_fill” int_xpos int_ypos*” where each parameter is separated by space. Note the quotes around the fill char.
    - To double a hourglass, multiply the height by two. To halve it, divide the height by two.
    - If an hourglass has an odd height, print a triangle on top that has height ceil(height/2) and a triangle on bottom that has floor(height/2).
    - The dump message for a hourglass should be in the following format “HOURGLASS (name:*name*) (x:*xpos*) (y:*ypos*) (height:*height*) (area:*area*) (fill:*fillchar)* (draw_amount:*draw_amount*)” where the italicised values are the actual values associated with the shape. This should all be printed on one line.

A few additional points

- All initializing functions for shapes will take in an x and y position on our screen. This is the top left corner of the bounding box of the shape.

- When halving sizes, always take the floor of the result from the division for the new size value.

Looking at the provided input and output will be a good place to start and understand what this program is doing.

You are required to turn in at least the following files: ShapeREPL.java (contains the code for your REPL), Shape.java, Rectangle.java, Triangle.java, Parallelogram.java, and Hourglass.java.

Miscellaneous

In order to receive a grade, your repo must only contain your source java files in the root directory of your repo.

This assignment will be submitted through github classroom. Make sure all of your code you would like to submit is in your repository when the due date arrives.

Note: Your output must match what is defined here in the spec. We will give you a small selection of test cases so you can make sure you have the right format. Do not print extraneous output, you may lose a lot of points (This includes prompts when reading in input!). **These test cases can be found in your repo.**

Remember, do not cheat! Refer to the syllabus and first lecture for more information.