Activity 3.2 - MiniFig (Part 2)

Part 2 Instructions

This activity continues working with the MiniFig class. In this part of the activity, you will use methods from the Graphics class to make improvements to your existing MiniFig avatar.

Here's an example to give you some ideas.

NOTE: This is to show you what is possible, you may not have time during class to add as many details.

Instructions

Getting Started

1. Open your existing "`Activity3.1`" project in VScode.
2. Add the following features to your existing "`MyAvatar.java`" class.

Specifications

Draw lines, ovals, rectangles, and arcs using the methods from the Graphics class to do the following. Don't forget to look at the MiniFig API documentation. Sizes and positions should adjust when the window is resized.

1. Draw some background scenery.
2. Customize your MiniFig (see Bob's hat Tutorial)

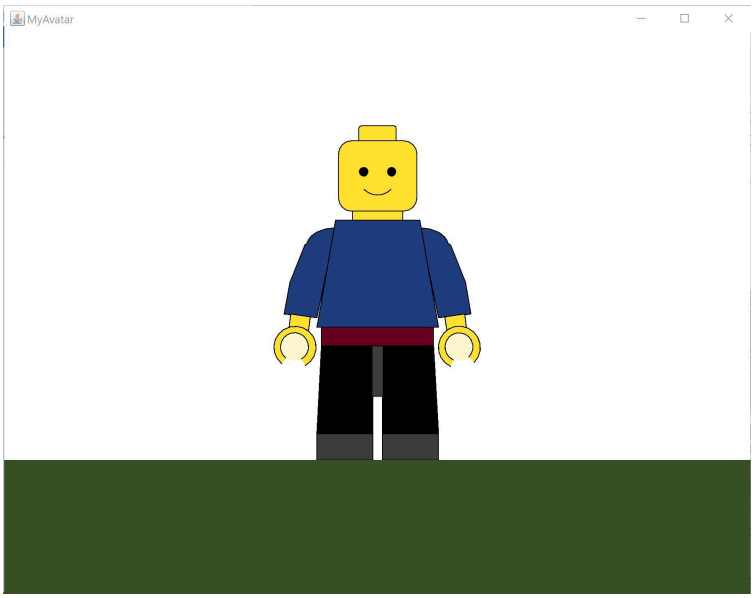
a. Give your MiniFig an accessory to hold in its hand.
    i. Use the `getLeftHandCenterPoint()` or `getRightHandCenterPoint()` methods to place your accessory.
b. Give your MiniFig a hat or some hair.
    i. Use the `getCapPoint()`, `getFaceWidth()`, and `getFaceHeight()` methods to help place your headpiece.

Terminology Identification

In your code add comments identifying examples of the following: coordinates, anchor point, RGB. These should be identified with an @keyterm tag within the comment.

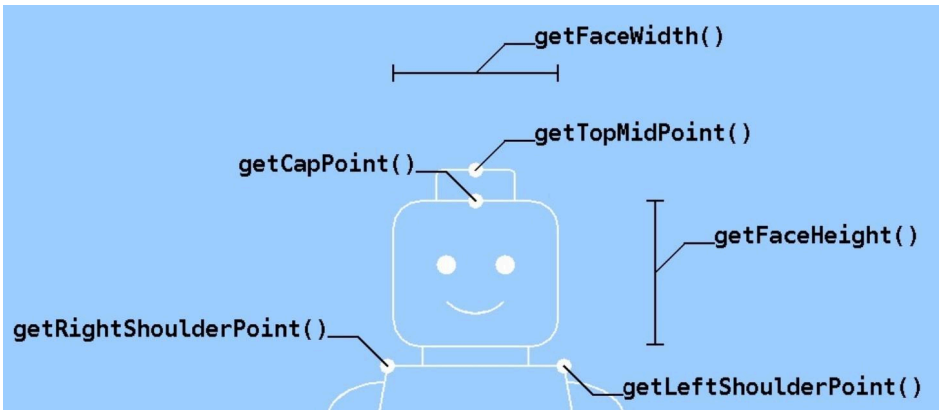
Bob's Hat, a Tutorial

This is MiniFig Bob. Bob is currently sporting a nice warm blue sweater, but Bob is in need of a hat. How can we give him a simple half-oval stocking cap that matches his sweater? Also importantly - how can we give him a hat that is always the right size and stays on his head no matter how we resize Bob and his window?



To get a solid-colored half-oval, we'll need the Graphics fillArc() method. We know that to get the top half of an oval we'll need to start at 0 degrees and sweep through 180 degrees.

(Or, if you're that kind of person, start at 180 degrees and sweep through -180 degrees.) To be scaled and positioned correctly, though, none of the other parameters of that arc can be hard-coded literal values; they need to be calculated relative to current Point locations and dimensions we can get from Bob. From the MiniFig API documentation, we know we have methods that can give us the top center point of Bob's face and the current width and height of his face. These will form the basis for our arc.

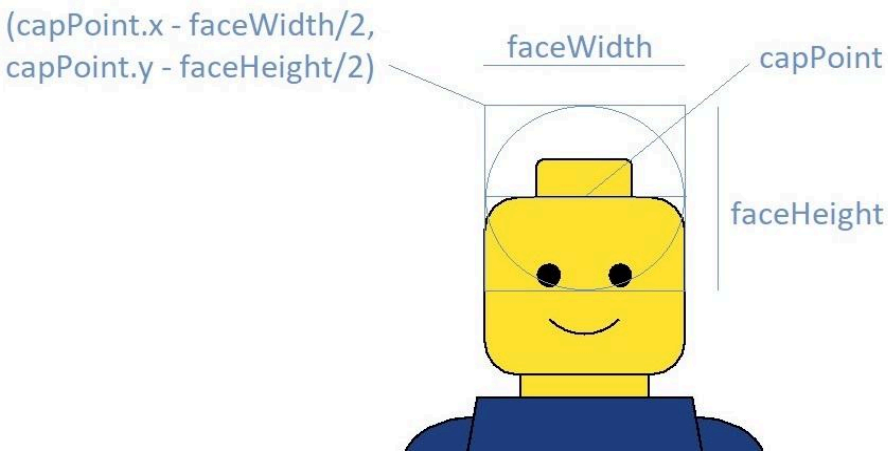


For clarity, let's create variables for each of these three values.

```
Point capPoint = bob.getCapPoint():
int faceHeight = bob.getFaceHeight();
int faceWidth = bob.getFaceWidth();
```

An arc requires us to define the whole oval that it is a part of, and for an oval we need to define its bounding rectangle. We'll want the oval to be as wide as Bob's face, of course, but how tall should it be? Let's guess that the hat should be half as tall as Bob's face, so the whole oval would be the full height of Bob's face. That's convenient - we already have those variables.

We have a width and a height, now, but how do we position the oval? The oval should be centered at the capPoint so the half-oval will sit right above Bob's face. Calculating the upper-left corner of the bounding rectangle, then, requires figuring out how much offset to add or subtract from the capPoint.x and capPoint.y values. Since we're centering on capPoint, we need to subtract half of the oval width from capPoint.x and half the oval height from capPoint.y.

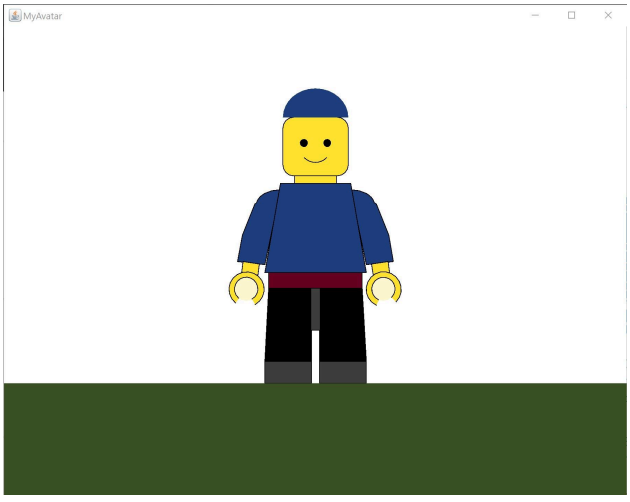


Again, for clarity, we'll define variables for the hat's x and y anchor point values.

```
int hatX = capPoint.x - faceWidth / 2;
int hatY = capPoint.y - faceHeight / 2;
```

Assuming the Graphics Color is already set to the same Color as Bob's shirt, all that is left to do is call the fillArc() method with our values:

```
g.fillArc( hatX, hatY, faceWidth, faceHeight, 0, 180 );
```



Bob is better prepared for winter with his hat. Now all he needs is a snow shovel, a warm house, the sun, a snowman friend...

Rules of thumb for painting with Graphics

- Start by calculating dimensions and *then* figure out positions. Positions often use dimensions in their calculations, as we saw with Bob's hat.
- When you don't have pre-scaled values to work from (like Bob's Points and dimensions), try to base your drawings on only one of the window's current dimensions. For example, calculate a unit dimension as the current window height / 10 and make all other dimensions a multiple of your unit dimension. The size of your divisor in the unit calculation correlates to the level of detail you'll be working with. Smaller divisors will result in coarser, blockier images while larger divisors allow finer, more detailed images. I recommend trying divisor values of 10 or 20 to start.

Code Review

When you are finished with this activity, pair up with a classmate and show off your MiniFig (again)!

Submission

After completing the assignment, use the assignment link in Canvas and follow the submission instructions there. You will upload your `MyAvatar.java` file and submit your reflection in the "Comments" box.

Reflection Requirements

Write a one paragraph reflection describing your experience with this activity. The reflection should also include the name of your code review partner AND something interesting you found in their code. Please review the activity rubric for more details.