ITSC 1212 Module 3 Lab

In this lab, we'll be introducing and practicing the creation and use of objects, as well as showing you how new behaviors can be added to an existing class.

Concepts covered in this lab:

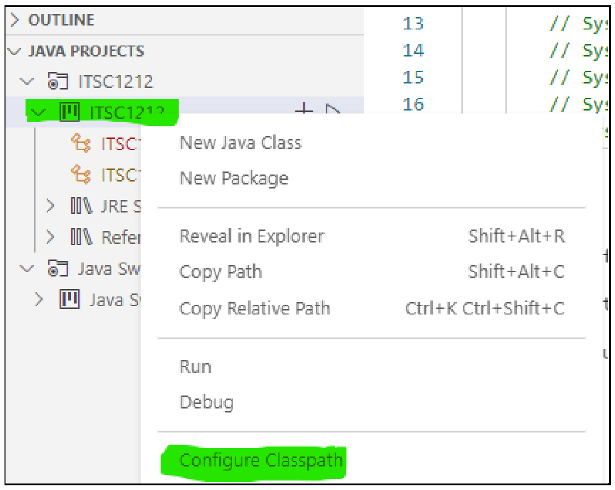
- Using Java libraries(packages)
- Creating objects
- Defining methods
- Changing object state using methods
- Using method parameters

Required files or links

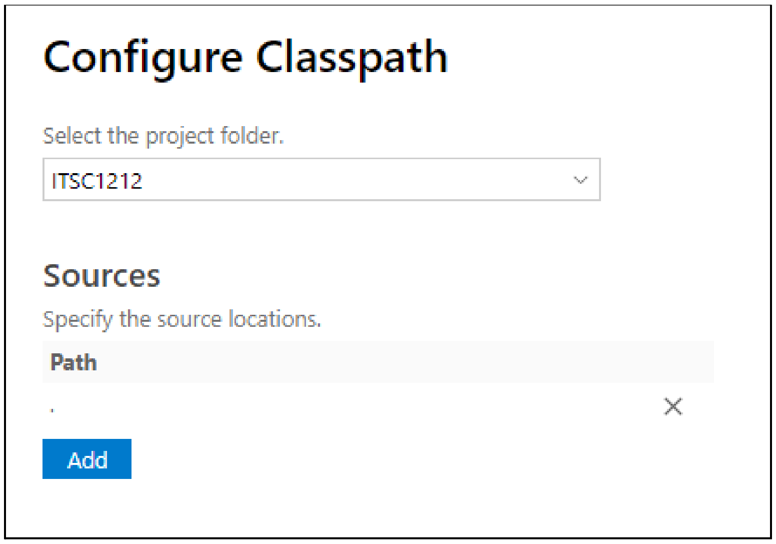
- TurtleJavaSwingCode.zip
- ITSC1212Lab3Hexagon.txt

Part A: Adding a source path

1. For this lab we'll be using the same classes and objects that were introduced to you during the prepwork. If you haven't completed that yet, it might be helpful to quickly review some of those topics. I will take this as an opportunity to remind you of the importance of completing the prepwork activities in the CS Awesome textbook. The work we do in the labs will make a lot more sense. Once you're feeling confident, download the zip file listed under Required Files, available on Canvas. Unzip this folder, and place it within your ITSC1212 folder.
2. Before we can start using those classes, we need to add them to our **classpath**. Essentially, a classpath is what allows the Java class we're currently writing in to see and use class files in other places on our systems. To start, navigate to the Java Projects sidebar of VSCode, and right-click on the ITSC1212 label highlighted in the screenshot below and select "Configure Classpath":



3. In the window that opens, click the add button.



Configure Classpath

Select the project folder.

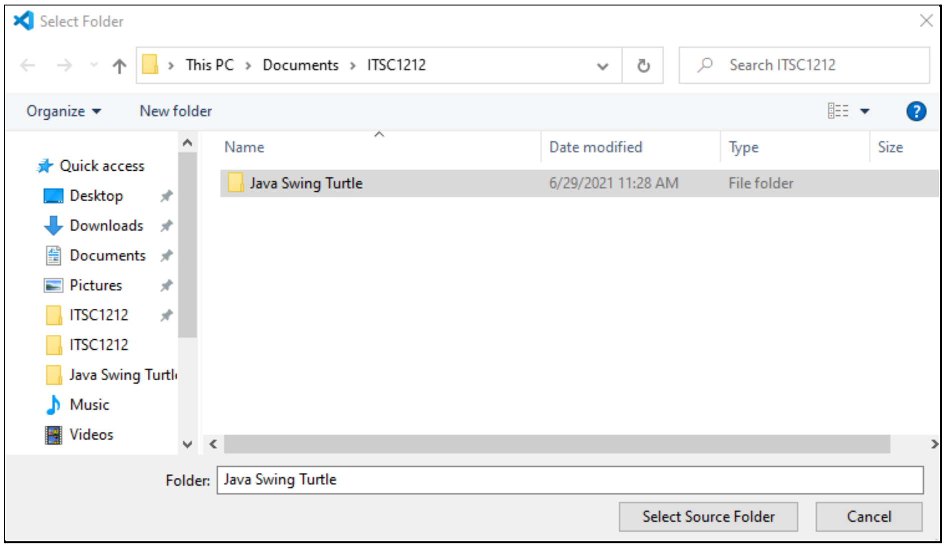
ITSC1212

Sources

Specify the source locations.

Path
.

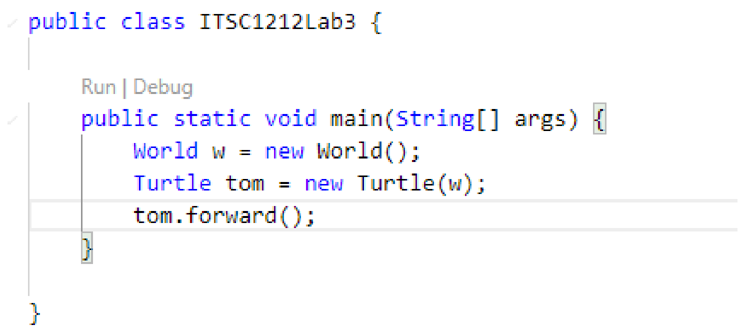
4. When you click "Add", navigate to and select the JavaSwingTurtle folder you downloaded and unzipped.



Name	Date modified	Type	Size
Java Swing Turtle	6/29/2021 11:28 AM	File folder	

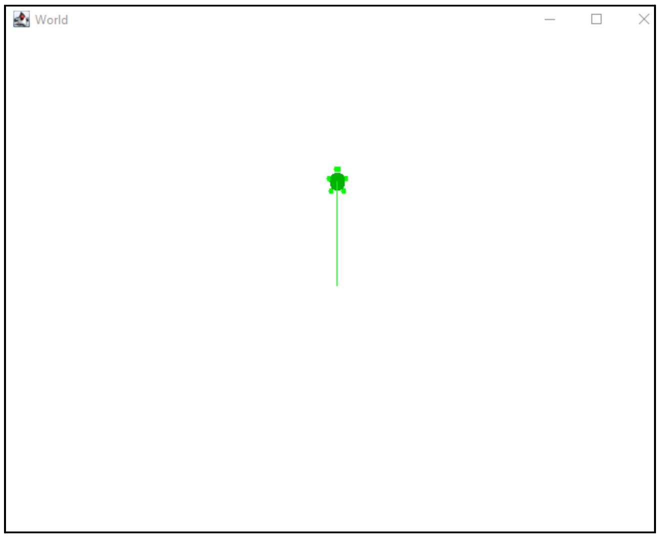
Folder: Java Swing Turtle

5. Now it's time to test if this has worked! Create a new Java file called ITSC1212Lab3.java. Give this class a main method. Inside your main method, you type the code below exactly.



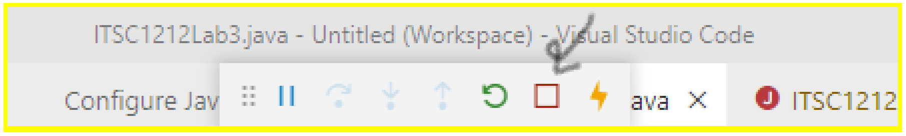
```
public class ITSC1212Lab3 {

    Run | Debug
    public static void main(String[] args) {
        World w = new World();
        Turtle tom = new Turtle(w);
        tom.forward();
    }

}
```

When you run this code, a new window should appear that looks like this:



If you don't see this window, ask your instructor or a TA for help before moving on.

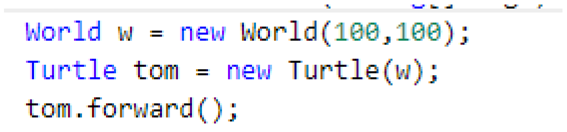
NOTE: When you run this code, or any code in the future that involves the creation of Worlds, you'll need to use this stop button to end the window before you can run your code again. You can also cancel a program in progress by clicking into the terminal tab at the bottom and pressing Control+C.


ITSC1212Lab3.java - Untitled (Workspace) - Visual Studio Code
Configure Jav
ava ×
ITSC1212

Part B: Objects and Constructors

1. As you learned in the prepwork, objects are instances of classes. By this, we mean that the attributes and behaviors coded into a class are able to be referenced and used through objects of that class. We'll expand on this in Part C
2. Look back at the code you added to your Java class in Part A, specifically the following lines:
```
World w = new World();
Turtle tom = new Turtle(w);
```
3. You should recognize these lines from your prepwork as lines where objects are being created and constructors are being called. The role of a constructor is to initialize the attributes of an object that is being created.
4. Notice that for the World object, we didn't need to provide any values. In contrast, the constructor for our Turtle tom did require a value. Delete the "w" from between the parentheses of the Turtle constructor and see describe what happened. Why do you think the "w" is required there?

5. By altering the way we use constructors, we can create objects that look or act differently than if we had left them alone. For instance, change your code so that your constructors look like this:



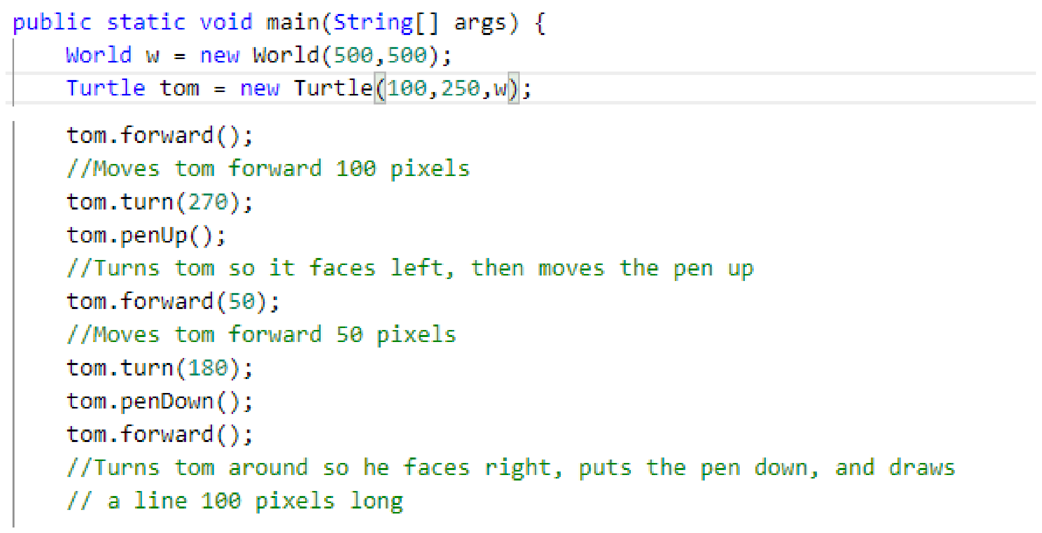
```
World w = new World(100,100);
Turtle tom = new Turtle(w);
tom.forward();
```

6. When you run the code, what has changed, and why?

7. Now let's practice using our objects. Worlds don't really do much besides act as a container for a Turtle, so we'll use our Turtle for this practice. One of the hallmarks of objects is that they can do things. Typically, we refer to these as behaviors or by the more technically correct name, methods. You'll recognize the word method from the "main method" we create in our class files. We'll talk about methods a lot in future units, but in a nutshell a method is a chunk of code we can call and execute on demand.
8. Look back at the code we have written. The line *tom.forward()* is us telling our Turtle object to execute its forward method, which moves the turtle forward. This is an example of good naming in programming: variable and method names should describe their purpose or function.
9. Now that you understand what the purpose of methods and how to call them, let's introduce a few more behaviors of Turtles:

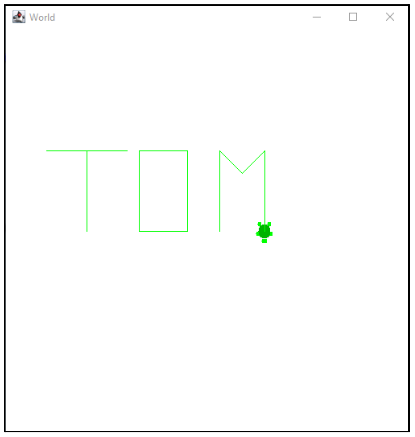
forward()	Moves the Turtle forward 100 pixels
forward(x)	Moves the Turtle forward x pixels
turn(x)	Rotates the Turtle in place x degrees clockwise (negative numbers turn counter-clockwise)
penUp()	Moves the Turtle's "pen" to the up position, so lines will not be written as the Turtle moves
penDown()	Moves the Turtles "pen" to the down position, so lines will be written as the Turtle moves
setHeading(x)	Rotates the Turtle to face a heading x. Ex. 0 is facing up, 180 is facing down

10. Alter the code in your Lab file so it looks like this. Notice the constructors for the World and Turtle have changed. Include the comments.



```
public static void main(String[] args) {
    World w = new World(500,500);
    Turtle tom = new Turtle(100,250,w);

    tom.forward();
    //Moves tom forward 100 pixels
    tom.turn(270);
    tom.penUp();
    //Turns tom so it faces left, then moves the pen up
    tom.forward(50);
    //Moves tom forward 50 pixels
    tom.turn(180);
    tom.penDown();
    tom.forward();
    //Turns tom around so he faces right, puts the pen down, and draws
    // a line 100 pixels long
```

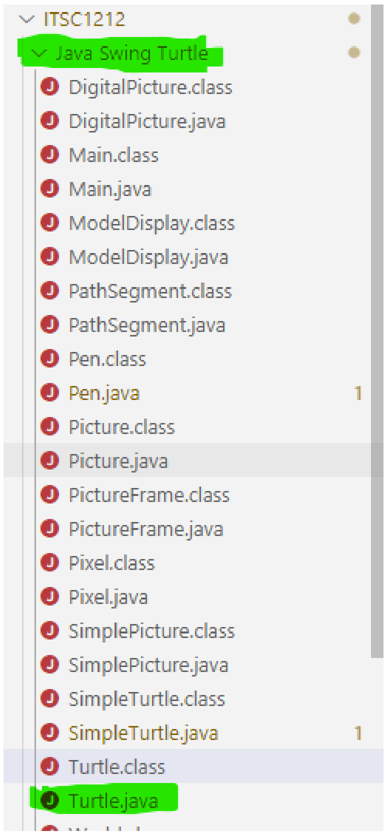
11. When you run this code tom has drawn us a nice looking letter “T”!! Look over each of the steps in the code until you understand how each of these steps adds together to create this letter.
12. Your task now is to add in the rest of the code necessary for Tom to write the rest of their name. Keep in mind that the “T” you’ve drawn occupies a 100x100 space, and the rest of the letters should be approximately the same size. Letters should also be spaced apart, which you can accomplish by moving the pen up, moving Tom to the right approximately 15 pixels, and then putting the pen back down to start the next letter. <u>We recommend writing out some basic steps using comments before you start writing code.</u>



13. Once you’re satisfied with your drawing, add comments that explain your steps then show your work to an instructor or a TA to receive credit for this section.

Part C: Creating and calling a new method

1. Now that you're comfortable with calling behaviors of objects, let's explore adding new behaviors. Let's start by opening the Turtle.java file that contains the code that makes a Turtle do Turtle things. You can open the file by finding it in the JavaSwingTurtle folder using File -> Open, or you can open it from your workspace like in the screenshot below.

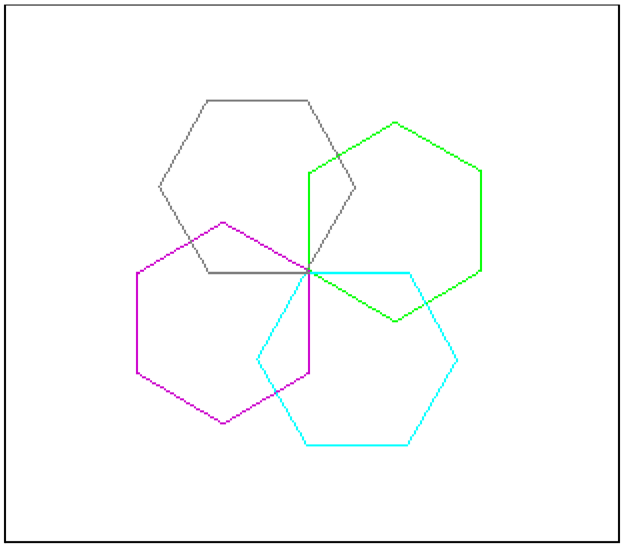


2. There's going to be a lot of code inside this file that will look very foreign to you at this point, but trust us that it won't for much longer. Be sure not to alter any of the code besides what you're told to, or your Turtles might stop working correctly.
3. Scroll down to the bottom of this file until you see a section marked "methods". We're going to add a new behavior for our Turtles below this comment header. After this dividing header, but before the curly brace that ends the Turtle class, copy and paste the code from the ITSC1212Lab3Hexagon.txt file available on the Canvas page for this lab into this space.
4. Once you've placed this code into your Turtle class, take a moment to look it over. Although there are a few things that look unfamiliar, most of these lines should look similar to the lines you used to draw Tom's name from Part B, only now we're using the word *this* instead of *tom.* The *this* keyword is essentially a shorthand for saying "the turtle that calls this method should do behavior X".
5. Switch back over to your Lab 3 Java file, and comment out the lines from Part B to draw Tom's name. Above those comments, tell tom to draw a hexagon using the new method/behavior we just added like so

```
World w = new World(500,500);
Turtle tom = new Turtle(w);
//Tom starts at the center of the world, facing up and pen down

tom.hexagon();
```

Notice that we've changed the Turtle constructor again, so that Tom starts in the center of the world window again.

6. Examine the drawing that appears when you run this code. The result isn't exactly surprising since the name was *hexagon()*, which is another example of good naming. What isn't a good example of good coding practice, however, is that the hexagon method we added is missing comments explaining its function. Head back over to the hexagon method you added, and write a few comments that explain what is happening.
7. Examine the picture below. It will be your job to recreate it



8. A few notes to help you:
    a. Each new Turtle object created (up to 4) will have a different line and body color. These colors will cycle from green to cyan to magenta to black then back to green with each new Turtle object that is created. Therefore, one way to accomplish this is to create 4 Turtles, each with a different color. There is also a way to manually change the pen color, using the setPenColor() method.

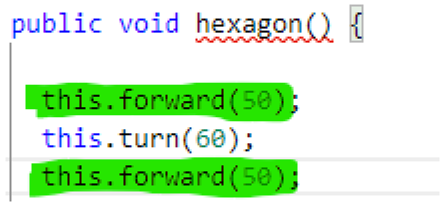
    ```
    tom.setPenColor(Color.CYAN);
    ```

    b. The first line of the hexagon is drawn in the direction the Turtle is facing
    c. Unless an alternate point is given in the Turtle constructor, each Turtle starts at a central point.
    d. B & C means that each Turtle should be created at the central point, then rotated differing amounts before being given the command to draw a hexagon.
    e. You can make the Turtles invisible by using the *.setVisible(false)* method.
    f. You can find the documentation for Turtles in section 2.2.5 of your text.

9. Once you are satisfied with your drawing, show it to the instructor or a TA for credit for this section.

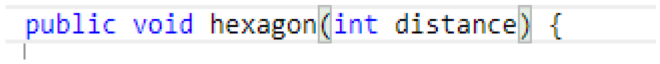
Part D: Adding and using parameters

1. One of the things you might have noticed about some of the Turtle behaviors we used earlier is that some had more than one version of the same behavior which could accept a number to customize the behavior. These supplied values are called ***arguments***, and methods that accept arguments do so through ***parameters***. In keeping with this theme, let's create a second version of our hexagon method that uses a parameter to accept arguments.
2. Let's start by copying and pasting our hexagon method in its entirety. Do that now, but notice that red error squigglies appear under the name of our copy.
3. If you mouse over the error notification you will see the message "Duplicate method hexagon() in type Turtle". In Java, methods need to have unique names (otherwise, how would the system know which one to run) EXCEPT when those methods have a different number or types of parameters. Since we intend this new copy to have a parameter, this error isn't something we need to deal with right now.
4. The parameter we want to add to this new method is one that can control the size of the hexagon being drawn. Currently, the size of the hexagon is being controlled by the lines where the Turtle is being moved forward



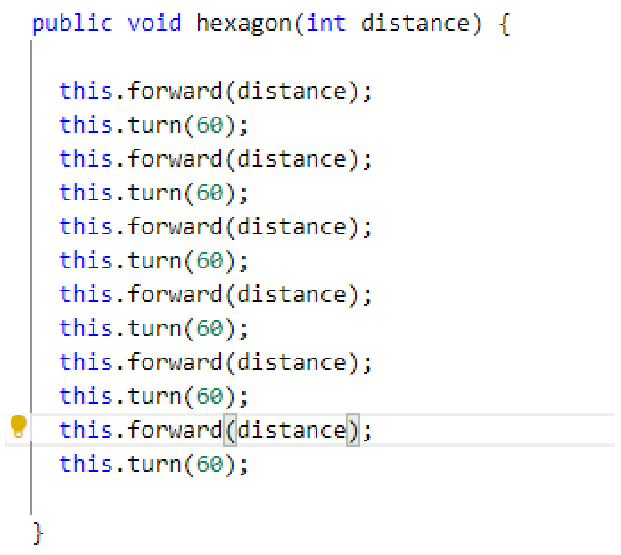
```
public void hexagon() {

    this.forward(50);
    this.turn(60);
    this.forward(50);
```

5. This method, as it's currently written, can only ever make hexagons with sides of length 50. However, we can replace that "50" with our new parameter to create a customizable hexagon size. First, we need to declare our parameter like any other variable in Java. For parameters, we do that inside the parentheses at the top of the method like this



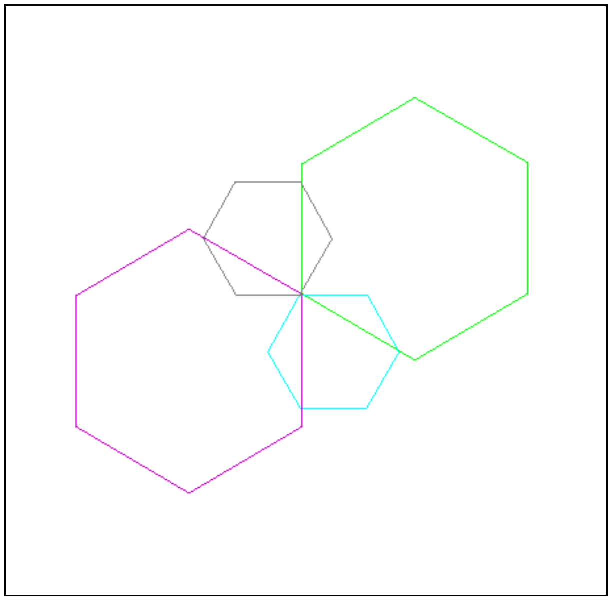
```
public void hexagon(int distance) {
```

   Notice that we have called our parameter "distance" since this will be how far the Turtle moves for each side of the hexagon, and variables should have names that explain their function or meaning.
6. Now that we have our parameter variable, we can replace each instance of "50" in lines that move the Turtle forward with it. Alter the code inside this method so it looks like this:

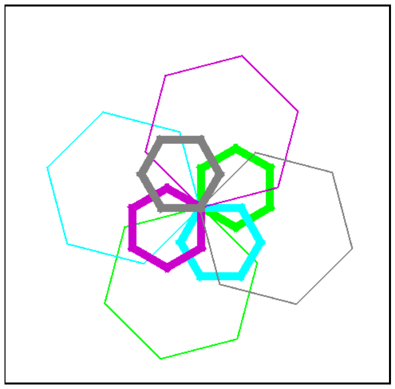


```java
public void hexagon(int distance) {

  this.forward(distance);
  this.turn(60);
  this.forward(distance);
  this.turn(60);
  this.forward(distance);
  this.turn(60);
  this.forward(distance);
  this.turn(60);
  this.forward(distance);
  this.turn(60);
  this.forward(distance);
  this.turn(60);

}
```

7. Take a moment and appreciate what is happening here. This new method will now accept a number which will determine how far the Turtle moves each time it draws one of the sides of the hexagon.
8. Switch back to your Lab 3 java file, and try to create a hexagon using different sizes. For instance, if you had two of your Turtles create hexagons of size 100 and the other two using hexagons of size 50, your image might look like this



9. Use this new ability to create hexagons of different sizes to create one last image collage. What you do is entirely up to you.



For this image, I also made use of the *.setPenWidth(x)* behavior to create lines of different thickness. Experiment with these different options and have fun! When you're done, show the instructor or a TA your image to receive credit for this section.