**Summary**

You will write a complete application ("app") program that (1) constructs one **World** (2) constructs one **ArtisticTurtle** in that **World** and (3) makes that **ArtisticTurtle** draw several regular polygons of various sizes and numbers of sides, and maybe an non-regular one too. You will accomplish that by programming your application's main method to alternately move the **ArtisticTurtle** and call one of the methods that you add today into the **ArtisticTurtle** class. You will extend the talents of **ArtisticTurtles** beyond G&E Turtles, to draw polygons with various numbers of sides and sizes. Please name your "app" **PolygonArtMaker**

We also want you to start making and using new, separate, fresh directories/folders, under (that is, inside) your CSI201 dir/folder, for each separate Lab and Project, beginning now.

Exactly what to draw is up to you: Be creative! The main objective is to solidly demonstrate two ideas:

1. The code that might (or might not) eventually, in the future, instruct this **Turtle** to alternately go **forward** and **turn** (which is how a **Turtle** can draw one polygon) **is written only once** into the **ArtisticTurtle.java** file. To write it, we edit a **method definition** into the body of the **ArtisticTurtle** class definition.

2. The method you have defined by writing its code into **ArtisticTurtle.java** can actually be run as many times as you like. (Even none at all!) You can write into the body of your application's **main(...)** just one instruction for each time you want the computer to repeat the long alternation of **forwards** and **turns**. We say the **one-line instruction that commands drawing one polygon is a call instruction**. The code to command the call is a one-liner: It specifies (A) on which **ArtisticTurtle** the method should be called (using **tRef** below); (B) it specifies the name of the method (say **pentagon**); and (C) perhaps parameter values (to vary features of the polygon, such as its size).

Your TAs will ask you to show your lab classmates exactly where certain things are: Within the **PolygonArtMaker.java** and **ArtisticTurtle.java** files, (1) the Java code that **defines** the pentagon method, (2) declares and uses the **parameter variable(s)**, (3) instructs **this Turtle**, to actually go forward and turn, (4,5) specifies which **ArtisticTurtle on which the pentagon method is called**, (4:in the call. 5:in the method body) (6) specifies the **name** of the method (**pentagon**) and (7) specifies the **parameter value(s)**.

Similar to Lab01 and Proj02, the application is to be a .**java** file named **PolygonArtMaker.java** and it must be accompanied by the file defining Lab03's **ArtisticTurtle** class named **ArtisticTurtle.java**

You must also upload the files **PolygonArtMaker.class** and **ArtisticTurtle.class** to prove that you successfully compiled everything you wrote. To learn to do with no bother the Information Technology for submitting project work (that counts more) you must retrieve and upload your work to Blackboard to get your grade for this lab. It's due the Monday eve. after this week.
**Concepts described by name and definition.**

**complete program**: A program, application, "app" that somebody can copy onto their computer and, at some future time, when he or she runs the program, makes that computer do some particular things. You can explain what a complete program will make a computer do and can usually tell whether or not it does the task you want.

**straight line program or code**: In computer science, this is a program or piece of a program whose structure is a simple sequence of commands or calculations that the computer will perform in exactly the order they are written in the program. (A straight line program has none of the loops and conditionals and methods that we have not taught yet, but will be taught very soon.) Most instructions or directions that people tell each other are straight line programs. **WARNING**: It does **NOT** mean a program that draws straight lines!

**polygon** and **regular polygon**: In plane geometry, a polygon is a "closed broken line lying in the plane" (Coxeter and Greitzer 1967, p. 51; quote quoted from [http://mathworld.wolfram.com/Polygon.html](http://mathworld.wolfram.com/Polygon.html)) The resulting line segments are joined at their endpoints, forming angles. The polygon is regular means all the line segments are equally long and all the angles have equal angle measure (expressed in degrees or radians). See the five examples above.

**The Math (people actually call this "Turtle geometry")**

To draw a regular polygon, a turtle could repeatedly do two things in alternation: (1) go forward a constant, particular amount of distance and then, (2) to prepare for the next line, turn a constant particular amount of angle. When the distances or angles vary, it's still a polygon but not a regular one.

The amount of angle must be properly related to the number of sides. For example, to get a square (which you know has 4 sides), it must draw 4 lines and turn 90 degrees between each. To get an equilateral (i.e., regular) triangle, it draws 3 lines and turns 120 degrees between each.

**Before starting:**

If you don't have the **CSI201** directory/folder, or the Book Classes set up from Lab01, get help right away so you don't fall behind. Get detailed instructions from the paper copies of the Lab01 (Not-prepared version) handout in the front of the lab room, from the TA.

Log in (as in Lab01). Your shell will be in your home directory (something like `/home/z/y/YourNetId`) Use the **cd CSI201** shell command to go into your **CSI201** directory. **THERE IS A PROBLEM** if there is any message! Get help from a friend or a TA if you don't fix it right away.

On the next page, we'll tell you how to do this: (1) MAKE a NEW DIRECTORY/folder named **Lab03** under (that is, inside) your **CSI201** dir/folder; (2) Copy your **Proj02** work (or **Lab01** work if you don't have **Proj02**) into that **Lab03** dir., and (3) MAKE SURE your Proj02 app. (or Lab01) compiles and runs from inside today's **Lab03** directory.

Every effective technologist WILL actually test, verify, and often necessarily fix, things that he or she thinks "ought to work". **DON'T SKIP THIS STEP**: It is fast and you will WASTE A LOT OF TIME if you cannot build right away on a functional copy of your previous work.
Making a new folder/directory for each separate Lab or Project, starting NOW!

The goal is to make the copies of some of your previous work and put them in a new dir/folder named Lab03.

A simple and effective way is to open your Proj02 or Lab01 DrawWithMethodsApp.java, click Edit/Save as..., MAKE THE NEW Lab03 dir/folder from within DrJava, and finish saving the opened file in that new dir/folder. Then repeat for ArtisticTurtle.java, except skip making the new Lab03 dir/folder.

The more professional and flexible way (on the lab Unix system) is to learn and use the Unix keyboard console shell commands cd (change directory), mkdir (make directory) and cp (copy). But there will be time for that throughout the semester.

If you didn't do either Lab01 or Proj02, there is now barely enough time to catch up. Follow the handouts for Lab01 which instruct you to set things up and get copies of the two files ArtisticTurtle.java and DrawWithMethodsApp.java. With these 2 files, you can proceed with today's Lab03. But...new, tricky things (like variables, methods, syntax, assignment operations, computations, etc.) have already built up.

Instructions continue on the next pages.
Details to get started

Here is an actual image of a G&E World overlaid by a plan I drew. The image shows the first line of a pentagon and the Turtle that (who?) drew it, overlaid with the plan for the remaining 4 lines.

Mark below and discuss in the lab:

After the Turtle draws one side of say the pentagon, how much should it turn (clockwise) to prepare drawing the next side? The diagram shows the exterior and interior angles.

(A) The amount of the exterior angle, 72 degrees which is 360/5, in the case of the pentagon.

(B) The amount of the interior angle, which for the pentagon, is 180-360/5 which equals 108 degrees.

1. Start DrJava with `/usr/local/depts/cs/geintro/drjava` in the black command window where you used your NetId and password (same you use for MyUAlbany) to log in. (When you type the password, what you type is not visible.)

2. Type in the big, upper right edit box:

   ```java
   public class PolygonArtMaker
   {
       public static void main(String[] a)
       {
       }
   }
   ```

3. Save this file. Make sure it is saved in your new Lab03 dir/folder, alongside `ArtisticTurtle.java` and `DrawWithMethodsApp.java`

4. Click Compile and then Run. Yes, a program that makes the computer do nothing is OK! It is useful to check that the compiling and running setup on your computer is OK. It also teaches the lesson that a program that does nothing is a valid program!

5. Type the code below inside the body of the main method. Then compile and test again.

   ```java
   World wRef = new World( );
   ArtisticTurtle tRef = new Artistic( wRef );
   tRef.forward( 50 );
   ```

6. Verify that the computer draws a vertical line with the Turtle figure at the top.. If it isn't, get help from the TA or a neighbor right away.
Details to carry on and finish

1. Based the answer to the above (A)-(B) choice question and discussion of it during the lab, define the method named **pentagon** by *adding code into ArtisticTurtle.java*. ALSO code in *PolygonArtMaker.java*’s main code to call your pentagon method on the *ArtisticTurtle* referred to by *tRef*, to make your application draw a pentagon when it runs.

   You will find that the code to alternately make this turtle go forward and turn follows a regular pattern.

   We developed this code live during a lecture last week, and the result is posted in the public web site.

   The important idea is that the steps express a regular pattern. The more you can relate the repeated steps to the results, the easier it is to figure out how to create better and better results.

   A fine way to actually type several repetitions of the same two lines of code is to highlight them both, *Copy* from the *Edit* menu or the Control-c shortcut and repeatedly *Paste* or the Control-v shortcut.

   The next course topic, to come soon, but after variables, is how to code loops and understand exactly how they work to make the computer repeat something. Compared to loops, coding the same things over and over (even with copy-paste) is almost always a terrible way to program repetition.

2. Add code **in main only** to make a few more regular pentagons in various locations in the *World* and with various different sizes. Try to plan where and roughly how big the next polygon will be.

   Then: choose a side length and move the *ArtisticTurtle* to the starting point to try to realize your plan, roughly. Just do as many different examples as you like in the lab to become good with the idea!

   You might use a bigger world constructed with the command new *World(800, 800)* instead of new *World( )*.

3. Now create and add to your *ArtisticTurtle* at least one but preferably two more methods capable of drawing polygonal figures with different numbers of sides from 5, and parametrized so the parameter value in the call (from within main, of course) controls the length of each side.

   If you feel like it, try making "star-shaped regular polygons" like the pentagram below by thinking about how to get the *Turtle* to move around in order to trace out such a shape with its sharp angles. Ask yourself: Should the *Turtle* turn more or turn less to make star-shaped compared to non-self-intersecting shapes?

---

**Homework: Upload check**

Like you did for Lab01, AFTER LAB (or in Lab if you brought a laptop) access your *PolygonArtMaker.java, ArtisticTurtle.java, PolygonArtMaker.class* and *ArtisticTurtle.class* files and upload it (as 4 attachments) to the Lab03 item under the Homework, Projects, Lab, etc tab on Blackboard. Make sure you pick the file ending in .java and AVOID the .java~ (with the tilde) [We don't care if you upload ~ files but you get a 0 if the .java and .class files are missing!]

---