Project 02 (due 9/21) builds on Lab 01. First, we show you how extend Lab01, to accomplish 2 goals:

1. Modify the body of `hook( int numberParam )` coded in `ArtisticTurtle.java` so the parameter value actually **DOES** control the size of the hook whenever the `hook( )` method is called.

2. Modify the body of `DrawWithMethodsApp`'s `main( )` method so that the application draws two different size hooks in two different corners of the `World`. (It draws those instead of Lab01’s stairway to Java glory.) Of course, `main( )` does not command the `Turtle` to draw anything directly. Instead, `main( )` does two method calls, calling the `hook( )` method twice, once to draw each hook. Since the parameter value given to `hook( )` now actually does control the size, the two hooks have different sizes because the **two calls** to the same **one method** `hook( )` call `hook( )` with two different parameter values.

```java
public class DrawWithMethodsApp {
    public static void main(String[] a) {
        System.out.println("Hello from main!");
        World wref = new World();
        ArtisticTurtle tref = new ArtisticTurtle( wref );
        tref.setPenWidth( 4 ); //Remove this if you like.
        tref.setPenColor( java.awt.Color.RED );
        tref.penUp();
        tref.moveTo(50,463);
        tref.penDown();
        tref.hook( 3 );
        tref.penUp();
        tref.moveTo(550,75);
        tref.penDown();
        tref.hook( 1 );
        tref.penUp();
    }
}
```

```java
public class ArtisticTurtle extends Turtle {
    public void hook( int numberParam ) {
        System.out.println( "The hook method has been called on ArtisticTurtle " +
        this +
        "with parameter value equal to " +
        numberParam );
        this.forward( 50*numberParam ); //Use the param. times 50 for how far to go.
        this.turn( 90 );
        //IN LAB01, YOU DID finish the programming job by typing 2 more lines below:
        this.forward( 35*numberParam ); //Use the param. times 35 for how far to go.
        this.turn( -90 );
    }
    public ArtisticTurtle( World wrefParam ) {
        super( wrefParam );
    }
    public static void main(String[] a) {
        System.out.println("DONT RUN ArtisticTurtle!!");
    }
}
```
YOUR PROJECT 02 ASSIGNMENT--on-time due, Sept 21, 11:59PM. Turn it in like Lab01 to Proj02:

1. Code into ArtisticTurtle one additional method that, like hook( ), contains instructions to instruct this Turtle to draw some original logo composed of straight lines. Your logo's size is controlled by the parameter value. The parameter value, a number, will be given in the calls you will code in main( ); you must NEVER write that number inside the logo method's code! When main( ) calls your method with a big parameter value, your method must make this Turtle draw a big logo. When main( ) calls your method with a small parameter value, your method must make this Turtle draw the same logo in a small size.

2. Code into the main( ) method of DrawWithMethodsApp THREE different calls to that logo method you created in step 1. Those three calls must pass three different numeric parameter values (one in each call, of course) to your one logo method to make it draw the logo in three different sizes.

You must also code tref.penUp( ), tref.penDown( ), and tref.moveTo( ) method calls before, after and between your three logo method calls to get the Turtle to draw the three logos in three different places near the top-left to bottom-right diagonal of the World. There must be no extra lines connecting or between the hooks and the logos. (These are rubric points 1, 2, and 3.)

For illustration's sake, here is result computed by the sample solution created by Prof. S.Chaiken--his logo is SC. Your logo (for full credit) must be complex enough so programming it gives you real practice (without being too tedious). Here's the rest of the rubric: (4) It MUST have two separate pieces that are separated by clearly visible space; they must not touch. (You'll have to use this.penUp( ) and this.penDown( ) to make that happen.) (5) Each of the two separate pieces must appear to have at least three lines, not all in the same direction. (6) The whole logo must be drawn with minimum of 7 lines (SC's has 8). (7) The 3 logos and the 2 hooks must never touch or cross each other, and none of them must ever touch the border of the World. (8) Don't go overboard before you learn much better ways to program really complex figures: Don't bother changing the hook and limit your original logo to 13 lines or less.

During your project work, you will almost certainly see that your logo drawings are not what you want, don't meet all the rubric's requirements, are not located where you want them to be, will touch or overlap each other or the hooks, or touch the border. Almost certainly there will be extra lines or missing lines because of missing penUp( ) and penDown( ) operations. It is normal and expected that you will go back and forth between ArtisticTurtle.java and DrawWithMethodsApp.java making, compiling and testing changes in your code to gradually make the results better, often making them worse for a while. Soon fixing syntax errors becomes easy! Welcome to the activity that programmers spend enormous amounts of time on: Debugging.

If you want to, you may read, BUT NOT COPY VERBATIM, the complete code for SC's sample solution, in a separate document. It also has exhaustive comments that express the author's thoughts about the purpose for almost every line. Write any comments like that if you feel they are helpful to you. BUT: A wrong comment is worse than none, and prepare for quizzes and exams that ask you to EXPLAIN THE PURPOSE of individual lines of code here and there in programs we ask you to study.