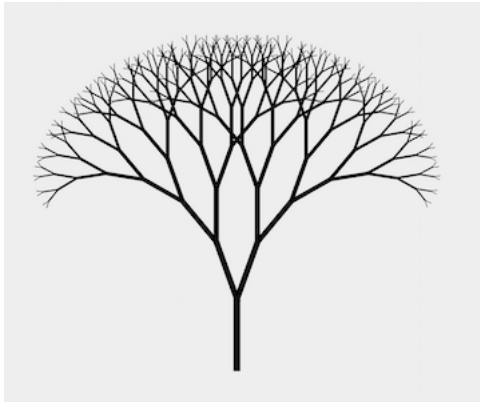


For me, trees have always been the most penetrating preachers. I revere them when they live in tribes and families, in forests and groves. And even more I revere them when they stand alone.
– Herman Hesse

The clearest way into the Universe is through a forest wilderness. – John Muir



9 segs, 2-way branching,
20° split angle, no noise, no leaves



9 segs, 2-way branching,
20° split angle with random noise,
branch length with noise
leaves when less than 4 segs remaining

In the lab, we review a few fundamental concepts about the Java ...

1. In your cs120 folder on your student network drive, create a new folder “workspace”.
2. In Eclipse, create a project folder named *01userid* where *userid* is your Austin College Network ID. For example, Rusty Buckle’s userid is rbuckle15.
3. Create a new class inside your project named **Tree**. An instance of this class knows how to render (draw) a tree given a java Graphics helper object. [See discussion below this list of steps.](#)
4. Create a new class (**TreeFrame**) that extends JFrame. Provide a suitable constructor (to create and hold onto a single tree) and override the paint method. Your paint method should paint the single tree (ie, tell the tree to paint itself).
5. Create a new class (**MainDriver**) with a static main algorithm. The main algorithm should create an instance of the MainFrame and then tell it to become visible.
6. **Adding Leaves...** Add/define another method to the Tree class used to render a leaf pattern at a given location x,y and direction (angle). Add a


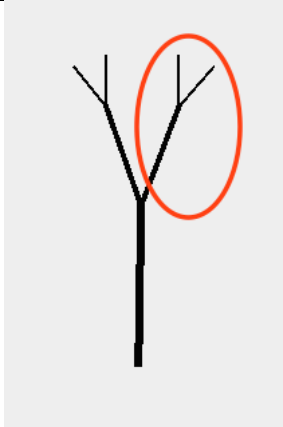
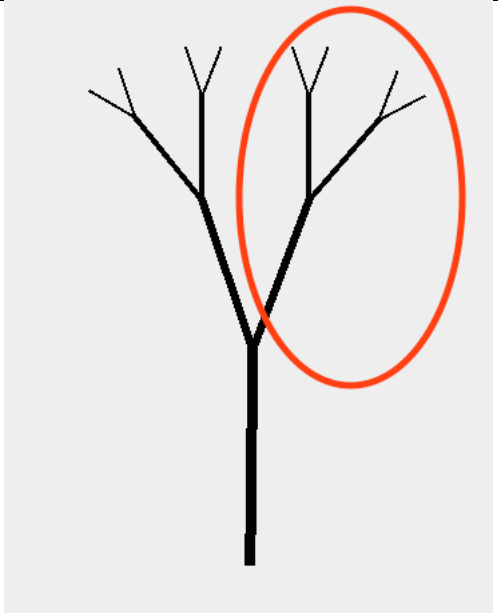
- constant to control when (segment number) leafs start to appear.
7. **Add random noise** to the calculation of *length of current branch* segment. Use a constant to control the max amount of noise for branch length.
 8. **Add random noise** to the *angle* at which two subtrees grow. Use a constant to control the max amount of noise.
 9. **Add random noise** to the shade of green for your leaf structures. Use a constant to control the max amount of noise.
 10. Play around with your constants, split angle, max number of segments, amount of noise. Tune them to your taste.
 11. Create an alternative leaf structure. Make it your own. Have fun with it.
 12. Make sure you comment your lab according to our cs120 class guidelines (see Moodle).
 13. Export your project within Eclipse to a zip file.
 14. Submit your zip file to moodle.

Class Name: Tree**Superclass: Object***Instance Variables (Data Members)::*

Variable	Purpose
splitAngle : double	angle (in degrees) between the next two subtrees after the current branch. 0.0 means EAST/RIGHT and -90.0 or 270.0 means NORTH/UP. Defaults to 20 degrees.
maxSegments: int	maximum number of branch segments we use to draw the tree.

Instance Methods:

Method Category	Method	Purpose
constructor	Tree(splitAngle: double, maxSegments:int)	initializes the instance variables
getters/setters	setSplitAngle (aVal:double)	sets the tree's base trunk angle
	setSegments(aVal: int)	sets the tree's max number of segments
	getSegments(): int	
	getSplitAngle(): double	
drawing	+drawOn(g: Graphics, x: int, y: int, angle: double)	public algorithm to render the tree with the help of the graphics object provided (g). kicks off the recursive algorithm by calling drawTree using the current parameter values.
	-drawTree(g:Graphics, sx:int ,sy:int, double sAngle, segsRemaining: int)	private, recursive algorithm. if segRemaining is less than 4 then draws leaves at the current sx,sy. if segsRemaining is greater than zero, then draws the next branch segment from sx,sy in the direction of sAngle and then draws two smaller trees sprouting at +/- split angle..
	-drawLeaf(g:Graphics, x:int, y:int, double: theta)	private helper algorithm called by drawTree algorithm in order to draw one or more leaves at a specified location (x,y) oriented by the specified theta.
	-branchLength(segsRemaining: int): int	computes and returns the length of the current branch to be rendered given the number of segments remaining. See discussion below.

		
2 seg tree	A 3 seg tree is a trunk branch with two 2 seg trees growing at left and right angles (ie, +/- splitAngle).	A 4 seg tree is just a trunk branch with two 3 seg trees growing at left and right angles (ie, +/- splitAngle).

To **draw a tree** rooted at a certain point (sx, sy) and growing in a certain direction/angle (θ)....

drawTree (g:Graphics, sx:int, sy:int, theta: double, remainingSegments:int)

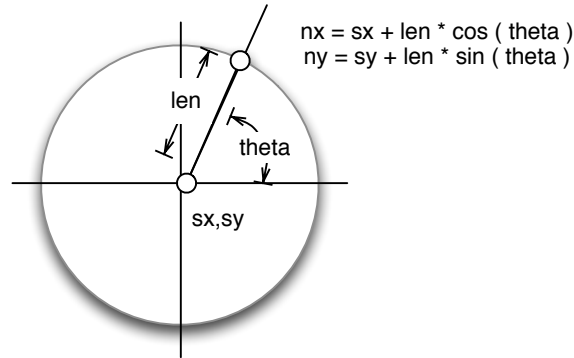
1. draw leaves at (sx, sy) if appropriate
2. if more segments to draw...
 - 2a. draw the next segment/branch starting at (sx, sy) to (nx, ny)
 - 2b. draw a tree at the other end of the branch (nx, ny) growing at a new angle to the left with one fewer segments
 - 2c. draw a tree at the other end of the branch (nx, ny) growing at a new angle to the right with one fewer segments.

For each branch segment starting at the base trunk, we calculate the location of the other end of the branch (nx, ny) using the current location (sx, sy) and a computed branch length (len) and angle of growth (θ). Minus 90 degrees is growing straight up. We can let the *branch_length* be a function of the segments remaining to render. This ensures the trunk is the longest segment and the very last segments at the top of the tree are quite short. For example, a simple linear function like this...

$$len = 10 * segsRemaining$$

But you may want to play around with this function. You should create a branch length function/method to compute this value given the number segments remaining to render.

Once we know how long the branch will be, and the angle/direction of growth, we can use our knowledge of the unit circle to compute the precise next point on the other end of the branch/trunk (nx, ny).



Once we locate the other end of the branch, we can draw a line from (sx, sy) to (nx, ny) . Now we can pretend to start drawing two other trees rooted at (nx, ny) growing off at two different angles ($a1$, $a2$ below). We simply have to calculate the new angles and make the recursive call.

To calculate the angle of growth for the next tree branch...

$$a1 = \text{theta} + \text{splitAngle} + \text{angleNoise};$$

$$a2 = \text{theta} - \text{splitAngle} + \text{angleNoise}$$

Now we can simply call the current algorithm `drawTree` rooted at the new point (nx, ny) at the new angle ($a1$ or $a2$ above) with one less segments remaining.

When we say “noise” we really mean a random number between $-x$ and $+x$. For example, I might decide to define angle noise to be random value between -6.0 and $+6.0$. If our base split angle is 20.0° then

Please read about random numbers using `java.util.Random`.

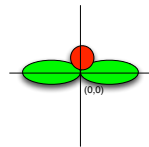
drawLeaves (g: Graphics2D, x:int, y:int, theta: double)

```
AffineTransform xf = g.getTransform(); // safe affine transform state
```

```
g.translate(x, y); // move the coordinate origin to x,y
```

```
g.rotate(angle); // and rotate to align with the current growth orientation angle
```

```
// now draw the leaf pattern as if you were drawing around an origin.
```



```
g.setTransform(xf); // reset affine transform back to original state
```

Changing the line width (also known as stroke), we need to create a Stroke object.

Changing the Stroke Width / Line Width

```
Graphics2D g2 = (Graphics2D) g;
```

```
BasicStroke stroke = new BasicStroke( 15 );
```

```
g2.setStroke(stroke);
```

```
// get a better handle on the graphics object
```

```
// line will be 15 pixels wide!
```

```
// now the graphics system will use this stroke
```