Recursive Drawings

A recursive method is one that calls itself, usually with an increasing or decreasing argument. In this activity, you will trace the execution of recursive methods that draw 2D graphics.

Manager:		Recorder:	
Presenter:		Reflector:	

Content Learning Objectives

After completing this activity, students should be able to:

- Use the Java 2D API to draw a repeating pattern on shapes.
- Explain how base cases and recursive calls affect execution.
- Identify similarities and differences in recursive methods.

Process Skill Goals

During the activity, students should make progress toward:

- Tracing the flow of execution using print statements. (Information Processing)

Facilitation Notes

Make sure that teams spend no more than five minutes on the first question of each model. They will need time to set up and run the program, but they should use the questions to guide their discussion about the code.

Point out that Model 1 is a warm-up to become familiar with the provided code. There is no recursion yet, just standard Java 2D graphics. Remind students that $(0, 0)$ is at the upper-left corner of the `Drawing`. Show students how to terminate the program and avoid leaving old `Drawing` windows open in the background.

At the beginning on Model 2, briefly explain how `VeeTree` inherits from `Drawing` and overrides the `draw()` method. Because there is no constructor defined in *VeeTree.java*, the the compiler automatically generates one that invokes `super()`. Recommend that students change the `DELAY` constant to 250 ms for this drawing.

After Model 3, point out that it's very common for classes to have a public non-recursive method that calls a private recursive method.

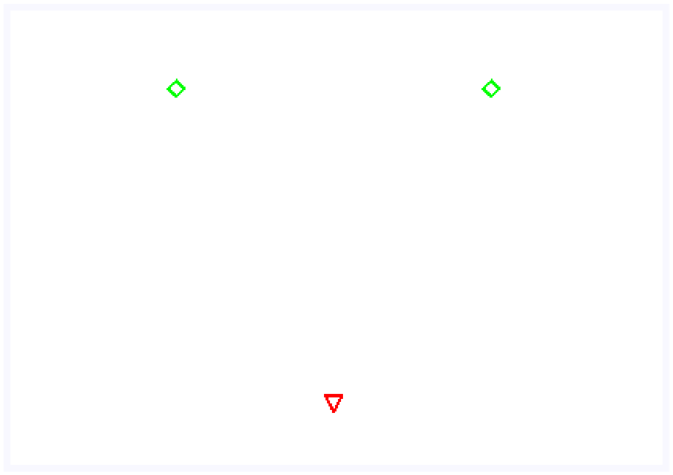
Key questions: #6, #12, #18

Source files: *Drawing.java*, *Triangles.java*, *VeeTree.java*

Model 1 Drawing and Tracing

Open *Drawing.java* and run the program. Keep an eye on both the Drawing window and the Console output. Notice the order in which the shapes are drawn. Run the program again, as needed, so that all team members can see its behavior. Then answer the questions below to explore and discuss the source code as a team.

Drawing (cropped)



Console output

```
diamond(300, 200)
    triangle(400, 400)
diamond(500, 200)
```

Questions (15 min)

Start time:

1. Fill in each blank with IS-A, HAS-A, or USES-A:

a) Drawing IS-A Canvas

b) Drawing HAS-A Graphics2D

c) Drawing USES-A Color

d) Drawing USES-A JFrame

2. Based on the `Drawing()` constructor:

a) What is the Canvas width? 800

b) What is the Canvas height? 600

c) What is the JFrame title? `"Drawing"`

d) What is "in" the JFrame? `this`
Hint: see Line 33.

3. Summarize in your own words what each method does:

a) `paint(Graphics g)`

Called by the window toolkit to paint this Canvas. Initializes `this.g2` and calls the `draw()` method. Prevents the `draw()` method from being called multiple times.

b) `draw()`

Calls the `diamond`, `triangle`, and `trace` methods to draw the shapes and debugging output. Uses specific coordinates for each of the shapes.

4. What is the purpose of the `g2` attribute? (i.e., How is it used in the program?)

The `diamond` and `triangle` methods use `g2` to do the actual drawing. It provides methods like `setColor` and `drawLine`.

5. Consider the "Console output" (from Model 1) and the `trace()` method:

a) Why is the "triangle" line indented?

The `trace` method was called with `level` set to 1.

b) Why are the "diamond" lines ***not*** indented?

The `level` was 0, and the loop (in `trace`) repeats `level` times.

c) How long is the delay after drawing each shape?

500 milliseconds (i.e., 1/2 second).

6. Modify the `draw()` method to draw and trace many diamonds and triangles. Use `for` loops to put each shape at a different (x, y) location. Reduce the `DELAY` so you can see the final result. Paste your source code below:

Answers will vary; this one fills up most of the canvas:

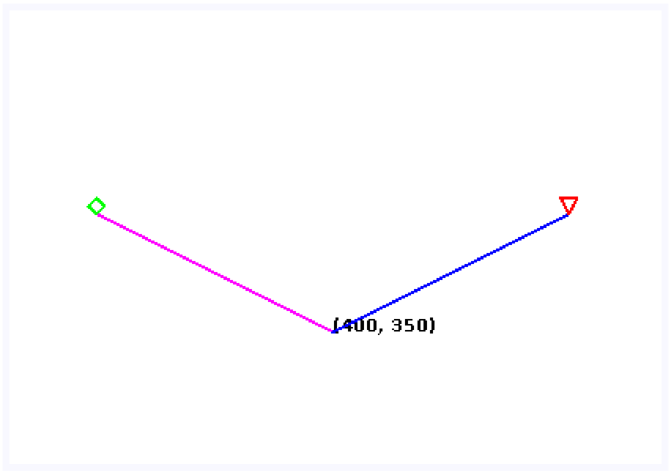
```
for (int x = 20; x < 780; x += 20) {
    for (int y = 20; y < 580; y += 20) {
        diamond(x, y);
        trace(0, "diamond(%d, %d)", x, y);

        triangle(x + 10, y + 10);
        trace(1, "triangle(%d, %d)", x + 10, y + 10);
    }
}
```

Model 2 The “Vee” Tree

Open *VeeTree.java* and run the program. Adjust the `DELAY` in *Drawing.java*, as needed, to notice the order. Then answer the questions below to explore and discuss the source code as a team.

Drawing (cropped)



Console output

```
Starting vee(400, 350)
diamond(250, 275)
triangle(550, 275)
Finished vee(400, 350)
```

Questions (15 min)

Start time:

7. The `vee()` method uses `g2` to draw strings and lines. Where is this variable declared?

In *Drawing.java*; it’s an inherited attribute.

8. Review the Javadoc comment for the `vee()` method. In terms of its variables, what are the following coordinates?

a) The bottom of the “Vee”: (x , y)

b) The diamond on the left: (left , top)

c) The triangle on the right: (right , top)

9. Add the following line at the end of the left branch, after the `trace` for diamond. What happens when you run the program now? Explain both the drawing and the Console output.

```
vee(level + 1, left, top);
```

The left branch continues drawing “forever” (until a `StackOverflowError`). Based on the output, the `vee` method calls itself repeatedly with an increasing `level` each time.

10. Add the following code at the beginning of the method, right after the first `trace`. What happens when you run the program now? Explain both the drawing and the Console output.

```
if (level == 3) {
    trace(level, "Aborting vee(%d, %d)", x, y);
    return;
}
```

The program draws a "Vee" shape at the top of each left branch, but it stops after 3 levels to avoid going on forever. The output increases to level 3, and then it decreases back to level 0.

11. Based on the most recent Console output:

a) How many times was `vee()` called? 4

b) How many times did `vee()` return? 4

c) How many diamonds were drawn? 3

d) How many triangles were drawn? 3

12. Add the following line at the end of the right branch, after the `trace` for triangle. What happens when you run the program now? Explain both the drawing and the Console output.

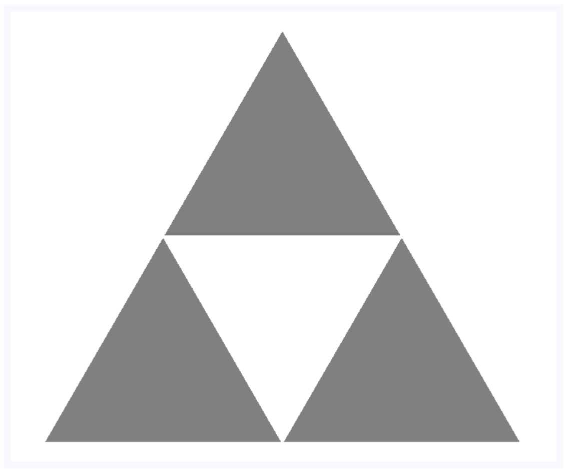
```
vee(level + 1, right, top);
```

The drawing is now symmetrical—there is a "Vee" shape at the top of each branch. Along the top of the drawing, there are eight branches that abort the recursion. The output both increases and decreases in level as the drawing moves up and down.

Model 3 Sierpiński Triangle

Open *Triangles.java* and run the program. Then change `LEVELS` to 1 and run the program again. Observe other `LEVELS` from 2 to 5. Adjust the `DELAY` in *Drawing.java*, as needed, to see the full drawing. Then answer the questions below to explore and discuss the source code as a team.

Drawing (cropped)



Console output

```
Starting tri (88, 570), (400, 30), (712, 570)
    Starting tri (88, 570), (244, 300), (400, 570)
    Finished tri (88, 570), (244, 300), (400, 570)
    Starting tri (244, 300), (400, 30), (556, 300)
    Finished tri (244, 300), (400, 30), (556, 300)
    Starting tri (400, 570), (556, 300), (712, 570)
    Finished tri (400, 570), (556, 300), (712, 570)
Finished tri (88, 570), (400, 30), (712, 570)
```

Questions (15 min)

Start time:

13. How many times is the `tri()` method called...

a) in the source code? 4

b) when `LEVELS = 0`? 1

c) when `LEVELS = 1`? 4

d) when `LEVELS = 2`? 13

14. Consider the vertices in the drawing above (when `LEVELS == 1`). Using the boxes below, indicate the location of `p1`, `p2`, `p3`, `p4`, `p5`, and `p6`. *Hint:* see Lines 49–51 and 71–73 of the code.

p2

p4 p5

p1 p6 p3

15. When the `tri()` method calls itself, what value does it pass for `level`?

It increases the current level by one.

16. What prevents the recursive process from continuing forever?

The "base case" when `level == LEVELS`. The `tri()` method starts at `0` and counts up.

17. When starting the drawing with higher `LEVELS`, what do the blue outlines represent?

Each blue outline represents a currently active method (i.e., stack frame). At the end of the method, the blue outline is overwritten with a white outline.

18. Compare the `VeeTree` program from Model 2 with `Triangles`. In terms of recursion, what do they do in common? How are they different?

Both programs have a drawing method that calls itself. They both use `level` to know when to stop repeating. In the case of VeeTree, `level` increases to 3. In the case of Triangles, `level` increases to `LEVELS`. Either way, `level` makes progress toward its goal.