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

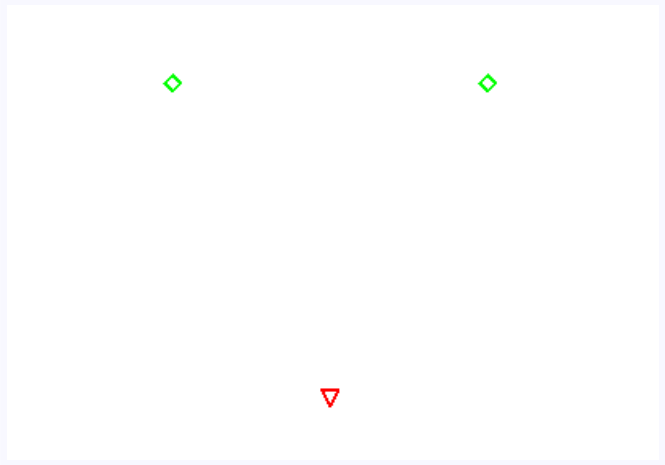
During the activity, students should make progress toward:

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

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 Canvas

b) Drawing Graphics2D

c) Drawing Color

d) Drawing JFrame

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

a) What is the Canvas width?

b) What is the Canvas height?

c) What is the JFrame title?

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

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

a) `paint(Graphics g)`

b) `draw()`

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

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

a) Why is the "triangle" line indented?

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

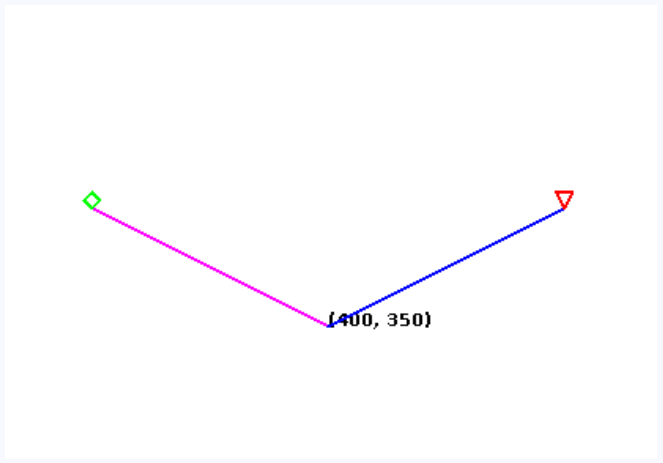
c) How long is the delay after drawing each shape?

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:

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?

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": (,)

b) The diamond on the left: (,)

c) The triangle on the right: (,)

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);
```

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;
}
```

11. Based on the most recent Console output:

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

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

c) How many diamonds were drawn?

d) How many triangles were drawn?

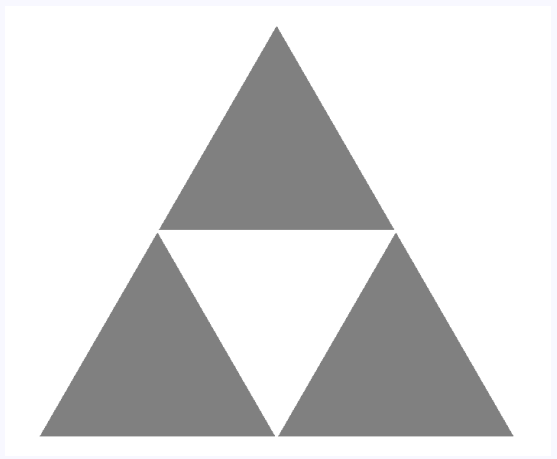
12. Add the following line at the end of the right branch, after the `trace` for triangle. What happens when you run the program now? Explain both the drawing and the Console output.

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

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?

b) when `LEVELS = 0`?

c) when `LEVELS = 1`?

d) when `LEVELS = 2`?

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.

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

16. What prevents the recursive process from continuing forever?

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

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