Lesson 8.03 — Mechanics of Recursion

Overview

Objectives — ***Students will be able to…***

- **Model** how recursive methods execute.

Assessments — ***Students will…***

- **Write** a recursive method
- **Model** the execution of that method for the instructor
- **Model** a method written by their peers

Homework — ***Students will…***

- **Complete** self-check #6, 10 and exercise #3

Materials & Prep

- **Projector and computer** for hook (https://youtu.be/3WBvS_n2oTY)
- **Whiteboard and markers**
- **Classroom copies** of WS 8.3, Teacher Demo 8.3
- **One pair of scissors** for each group
- **Small group assignments** (~4 students per group)
- **Classroom copies** of the textbook (**optional**)
- **Printout** for Teacher Demo 8.3 (**optional**)

Carefully read through the "Mechanics of Recursion" example in section 12.2. You should make a copy of WS 8.3 for yourself and do today's activity before delivering the lesson so you can smoothly demonstrate (and check) how to model a recursive method. If you decide to model an example for the class before the activity, use the pre-printed example so students can't just copy their sample method from the book.

Pacing Guide [Optional, Lessons Vary]

Section	Total Time
Bell-work and attendance	5min
Discussion of hook & introduction	15min
Activity 1: Modeling Recursive Methods	20min
Activity 2: Modeling a Peer's Methods	10min

Procedure

On your classroom projector or screen, have the animation of the Dragon Curve playing on a loop (link in "Materials and Prep"). Adjust the player to play 2x speed to keep student interest; you may want to mute the music. As bell-work, ask students how this animation is an example of recursion. Their explanation should describe the recursive case in pseudocode.

Bell-work and Attendance [5 minutes]

Discussion of Hook & Introduction [15 minutes]

1. Begin by reviewing the hook/Bellwork with students. There may be some debate as to what the recursive and base cases are for this curve; allow time for students to argue and come to consensus. Students should understand that (A) the base case is just a single straight line, and

   (B) the recursive case involves repeated the line with a 90° turn.

2. There's an important distinction to be made here: the animations linked to from this lesson show recursion from the "bottom up." This is not the way that recursion works. The modeling activity is a better representation of the recursive process; the animations are meant only to demonstrate how even the simplest recursive method can quickly yield large, complex results.

   If you feel it might be helpful to show your students a recursive implementation of drawing the dragon curve, you can find one here: https://scratch.mit.edu/projects/65378560/.

   If your class is fairly advanced, you may want to skip Teacher Demo 8.3. Advanced students will benefit from exploring the model on their own as outlined in WS 8.3. However, for many classes additional scaffolding will be needed.

3. To perform the demo, you should:

   - Cut out the cards in the Teacher Demo 8.3
   - If you have a classroom projector, do the demonstration under the capture so your cards are magnified to the point where students in the back of the classroom can see what you are doing. If you do not have a magnifying projector, you may want to do this demo as you walk around the class, showing students what is on each card.
   - Explain to students that you are modeling Java's call stack. A call stack is the way that Java keeps track of the sequence of methods that have been called. The model your using today is another way of tracking flow-of-control.
   - Tell students that in your example you'll be modeling a method that reverses a list of words. When students break into groups for the activity, they will be writing their own recursive methods and modeling Java's call stack for the methods that have been called.
   - Briefly review the reverse method introduced in 12.2.

4. Show students the first piece of paper form Teacher Demo 8.3 with the method definition.

   1. This should look pretty familiar to students, but have them point out to you the recursive case (input.nextLine()) to review.
   2. Point out to students the new feature on your model that holds the local variable line. Ask students what they think this represents? (It is a place in the computer's memory that stores the variable line.)
   3. Ask students to narrate what happens when we call the reverse method.
   4. Write the `this` variable on the slip of paper, and ask students what happens next. The method will get called again—but because recursion works differently than iterative code, we don't just run through the method again like with a loop.
   5. Take out the second copy of the reverse method, and place it over the first paper, as indicated in the textbook. Have students walk you through the method again and tell you what local variable will get written in the storage space at the bottom of the paper ("is").
   6. Repeat these steps until you have reached the base case (the fifth sheet of paper). Ask students what the local variable is, and discuss what happens when there is no word left to reverse. (The input.hasNextLine() returns false, so that method terminates.)
   7. Now remove that fifth card from the pile, revealing the fourth copy of the method again. Let students know that Java returns to where it was before it executed the call that just terminated.

      1. Ask students what happens next. (We print the line "no?" and terminate.) Write this output on the board so students can keep track of the output.
      2. This means that Java now goes to the place where it was before it terminated this call. Ask students what you should do now in your modeling of this recursive method. (Take off this card, and reveal the third method card.)

   8. Continue in this way until you have returned to the first card.

9. If you need to repeat sections of this demo, do so until you feel that 50% of your class understands the process. At this point, students should be able to help each other learn the concepts during their activity.

Activity 1: Modeling Recursive Methods [20 minutes]

Emphasize with students

Content - Advanced programming structures - Managing Complexity Recursion is an interesting but sometimes difficult concept. When dealing with more advanced ideas like this, it's important to manage complexity in order to design and implement complex programs.

This activity helps you organize recursion in your program. It's advised that you use these ideas, and consider these questions, as you create more programs that implement recursion.

---

1. Direct students to read through all the directions first before beginning the activity. Point out to them that they will be completing their own demo (similar to the one you just completed) using their own code.
2. Break students into small groups and distribute worksheets and scissors.
3. Keep students on pace by announcing 10- and 5-minute warnings.
4. If students complain about having to rewrite their recursive code so many times, point out that it's a good thing that we only have to type out the recursive code once! In reality, the method doesn't exist as many copies (that would take up a lot of memory!) We're just using physical examples to keep track of how Java is calling each "round" of the recursive method.
5. As students work, check first for correct recursive methods, then make a second round having students perform their modeling activity for you.
6. Be sure each group explains:
   - Where the base case is, and when the base case occurs.
   - Where the recursive case is.
   - What causes the method to advance to the next round of recursion.
   - What causes the method to terminate.
   - Where Java returns to after a method has terminated.
   - What the output is.
7. If students do not answer all of these questions correctly, give them time to regroup and assess them again after they have corrected their thinking.

Activity 2: Modeling a Peer's Methods [10 minutes]

1. After 20 minutes (or when students finish), have groups trade code sheets, and challenge the groups to work their way through the code again.
2. Keep them on pace by announcing time (they only have 10 minutes for this second activity). Visit each group and ask for groups to model these new methods for you.
3. You can opt to select some student groups that have a smooth presentation to share with the entire class before class dismisses.

Activity 3: Iterative to Recursive and Vice-Versa [Optional]

1. Introduce iterative and recursive methods that produce the same result. For example, here is an iterative and recursive example for solving factorials:

```
// Recursive                             |       // Iteration
int factorial(int n) {                   |       int factorial(int n) {
    if (n == 1) {                        |           int product = 1;
        return 1;                        |           for (int i = 2; i <= n; i++) {
    } else {                             |               product *= i;
        return n*factorial(n-1);         |           }
    }                                    |           return product;
}                                        |       }
```

2. Invite your class to take a recursive function and turn it into an iterative function. You can provide a new recursive function or they can use one's they've previously worked with. To help your class, put the recursive step into a for loop that worked to reach the base case [i (=<>) (#)].
3. Discuss what needed to change between the recursive and iterative calls.
4. Now, ask your class to do the inverse; take an iterative function and turn it into a recursive one. Based off your discussion and the previous change between recursion and iteration, this task should be easier.

Activity 4: Creating Fractals with Recursion [Optional]

1. Explain to your students that the dragon curve we worked with earlier is called a fractal, a fractal being a geometric figure in which each part has the same statistical character as the whole. They follow a pattern in which similar patterns recur at progressively smaller scales.
2. Invite your class to create the dragon curve in pseudocode, they should notice that the figure (they don't know how to draw) rotates 90° around its near point.
3. Now, as an easier activity, we're going to play with the Koch curve, which the code can be found here on stack: http://stackoverflow.com/questions/13000994/koch-snowflake-java-recursion. Explain all parts of this program, specifically the level, turning, the recursion in drawKochCurve, and the gpdraw package being used (myPencil and myPaper).

   The Koch curve follows three rules:

   1. Divide the line segment into three segments of equal length.
   2. Draw an equilateral triangle that has the middle segment from step 1 as its base and points outward.
   3. Remove the line segment that is the base of the triangle from step 2.
4. As a fun lesson, let your class play with the values, altering the Koch curve to create new designs. Award prizes to special and unique fractals students create or require that a fractal (and its code) be turned in at the end of class. If your class is comfortable with the activity, you can introduce color for myPencil which opens a door for interesting designs.

Accommodation and Differentiation

While all students should write their OWN algorithm, you should encourage students to work in pairs or small groups so they can share ideas and help each other organize their thoughts. This is particularly important in ELL classrooms, where emergent English speakers can pair with advanced English learners. If some students want to do this project all on their own, let them.

If you have students who are speeding through this lesson, you should encourage them to:

- Create a mnemonic or acrostic to remember all the steps for checking syntax errors
- Make a poster for the classroom illustrating the mnemonic or acrostic
- Help another student with the worksheet (explain, not solve-for-them)

About Error Checking in Eclipse

If your students are enthusiastic about the Dragon Curve, expand on the discussion by pointing out that there are 2 repetitions in the Dragon curve, one left and one right. This means that there are 2 lists in the recursive code.

Ask students to speculate on what the curve would look like with 4 repetitions. An animation of a 3D dragon curve can be found here: (https://youtu.be/BnUTikyR1CU)

For less-advanced classrooms, you may want to pre-populate the group worksheets with recursive methods. This will allow students to focus on processing the execution of the method rather than struggling with syntax. This is not recommended unless you are very pressed for time. In most cases, students should be practicing properly writing recursive code.

Video

- BJP 12-3, *Implementing a Recursive Method* http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoPlayer.php?id=c12-3

Forum discussion

Lesson 8.03 Mechanics of Recursion (TEALS Discourse account required)