Lesson 3.12 — Cumulative Algorithms

Overview

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

- **Find and correct** syntax errors in conditional cumulative algorithms.

Assessments — *Students will...*

- **Write, modify, and correct** programs written by others.

Homework — *Students will...*

- **Read** BJP 5.1 (skip "do/while Loops")
- **Complete** Chapter 4 Programming Project #2

Materials & Prep

- **Projector and computer** (optional)
- **Whiteboard and markers**
- **Classroom copies** of WS 3.12
- **Plastic binder sleeves** (1 per student)

Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Introduction to cumulative algorithms and min/max loops	15min
Programming Activity – Programmer 1 input	10min
Programming Activity – Programmer 2 input	15min
Programming Activity – Edits and Reflection	10min

Procedure

Bell-work and Attendance [5 minutes]

Introduction to Cumulative Algorithms and Min/Max Loops [15 minutes]

1. As your hook, conduct an experiment with your students:
   - Announce that you're going to read a list of six numbers between 1 and 100, and the first person who is done calculating the sum correctly wins [classroom reward of your choice].
   - Students should indicate they are done calculating by raising their hand. Explain that you will keep track of raised hands, and that you'll call on each person until you encounter the correct answer.
   - Read off the numbers, and keep track of who finishes early, and who finishes late. Ask students how they computed their answers.

     You're looking for an answer from the "quick" students that they kept a running tally, and that the "late" students that they added all the numbers at the end.
2. Now propose that you want to tally sixteen numbers. Ask students how students what method you should use, and ask them to describe the algorithm for solving the problem using pseudocode.
   - Translate the pseudocode into a loop using the following example: use a for loop with a code for adding using keyword sum:

     ```
     int sum = 0;
     ```

- We always initialize at 0, because sum needs an initial value to start with.

  for (all numbers to sum) { obtain "next" with Scanner sum += next

- This is pseudocode! In real life, we'd begin with building a scanner in the main method. Let's look at some real code here. Since we know we want user input, what is the first thing we do before creating our class?

  ```
  import java.util.*;
  ```

3. Now we create our class and main method (have students tell you what to do):

```
public class ExamineNumbers1 {

    public static void main (String[] args) {
        System.out.println("This program adds your numbers.");
        System.out.println();
```

4. Add our Scanner (have students give you the code):

```
        Scanner scanner = new Scanner(System.in);
        System.out.print ("How many numbers do you want to add?");
        int totalNumber = scanner.nextInt();
```

5. And finally our loop with the sum keyword:

```
        double sum = 0.0;                              // By using totalNumber instead
        for (int j = 1;  j <= totalNumber;  j++) {     // of an actual number, we give
            System.out.print("#" + j + "? ");          // our program flexibility.
            double next = scanner.nextDouble();
            sum += next;
        }
```

   Be sure to briefly discuss the use of `double` *vs.* `int` (where would you have to change code if you wanted `int`?), the variable `i` *vs.* `j` (you can make it anything as long as you're consistent), and variables `next` and `sum`.

6. Finally we have to add the code that returns the sum so that the user can see the result:

```
        System.out.println ();
        System.out.println ("Your numbers add to " + sum);
    }
}
```

7. Invite 3 students up to the board (now that the complete program is written)

   - One student acts as Java to trace flow of control (this student narrates what is happening on each line)
   - Another student acts as console output, writing output as Java creates it

     Try to have console output written on the same area of the board each time you do this type of exercise.

   - The other student acts as the values being stored in sum and next
     - This student should be writing their values in a location on the board that you never use for writing console output. Reserving a physical space for "internal workings we don't see" will help students keep execution vs. output separate.
     - Seated students should help the student figure out what the values in sum and next are being updated to with each execution of the loop.
   - Get sample user input from the class.

     If that example went well, move on to min/max loops; otherwise, work through the example with new user input. To scaffold this activity, use whole numbers and keep the loop short.

As students continue to work on their sample game, they may want to track minimum or maximum scores. Ask for student examples of when this might be the case (e.g. success in the fewest number of steps, maximum score)

8. In this case we stick with the for loop control structure, but modify it to keep track of whether the newest value is larger than (or smaller than) the values the user has input to Java so far:

   // Initialize max to lowest possible value or to first value. *for (all numbers to compare) { obtain "next" with scanner if (next > max) // or: if (next < min) { max = next // min = next*

   - Let's build a real program with this:

```
int min = value;
int max = value;

for (int i = 1; i < length; i++) {
    Scanner gets next number
    if (value > max) {
        max = value;
    } else if (value < min) {
        min = value;
    }
}
```

9. Ask students to discuss what this program does. If a more concrete example is needed, work through the first programming question together as a whole class. Encourage students to tell you what code to write down for each step, as much as possible.

Programming Activity – Programmer 1 Input [10 minutes]

Emphasize with students...

Content - Collaboration tools for programming - Standardized source code documentation As you complete this collaborative programming exercise, think carefully about the tools and skills that you used. Very few programs are written by one programmer. There is often a team of programmers all working on the same program, and collaboration is therefore very important.

Also, be sure to leave standardized source code documentation that your partner can understand. Your comments should clearly and effectively explain the code without being too lengthy. Writing clear and concise comments is a skill you will develop as you continue on in your programming journey.

---

1. Hand out WS 3.12
2. Read the instructions on the sheet aloud, then ask students to explain them back to you.
   - Ask students why pseudocode is so important to include in this exercise.
   - Explicitly point out the space for a structure diagram or program outline. Ask students why this is such an important structure to include.
   - Make sure that students are programming in pen, not in pencil, so that all editing steps are visible.
3. Give students 10 minutes to build the first part of the program.

   If students are struggling, you may extend the time, or offer universal helpful tips.

Programming Activity – Programmer 2 Input [15 minutes]

Have students trade papers with their assigned programming partner. 1. Remind students to add their name to the paper as "programmer 2." 2. Give students another 15 minutes to build on the program and make edits.

Programming Activity – Edits and Reflection [10 minutes]

Have students return each others' papers, and have the first student make edits and fill in the comment section.

Accommodation and Differentiation

As a "running activity," have your students create a new blank sheet titled "Tricky Code Cheat Sheet," and give them a plastic sleeve to protect this reference sheet in their binder. Encourage students to start writing down hints, tips, and useful snippets of code that they can refer to during homework, classwork, programming projects, and even tests.

Every time you encounter a useful snippet of code, recommend that students include it on their "Cheat Sheet," so students can drill the useful code throughout the year. Some tricks that show up on the AP include:

- Using `% 2 == 0` to identify even numbers, use `% 2 == 1` to identify odd numbers.
- To find the last digit of a large number, use `% 10`.

Tips from today's class would include:

- To keep a running total, use sum in a for loop (initialize to 0, put the range of numbers in the loop header).
- To access the min or max value, use an `if/else` statement (initialize to first or highest/lowest number, put data range in for loop header).

Ask students what a tip for Scanner would look like, or a tip for when to `import java.util.*`

Video

- BJP 4-4, *Programming with if/else and Cumulative Sums* http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoPlayer.php?id=c4-4
- CSE 142, *Cumulative Sum* (23:07-38:40) https://www.youtube.com/watch?v=fo9_kOSs1Y8&start=1387

Forum discussion

Lesson 3.12 Cumulative Algorithms (TEALS Discourse account required)