Lesson 3.01 — Parameters

Overview

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

- **Correctly construct** formal and actual parameters (arguments).
- **Predict** output of programs that use parameters

Assessments — *Students will...*

- **Teach** a mini-lesson explaining the relationship between parameters and values
- **Submit** Practice questions

Homework — *Students will...*

- **Read** BJP 3.1 "Limitations of Parameters" and "Multiple Parameters"
- **Complete** self-check questions 4-7

Materials & Prep

- **Projector and computer**
- **Whiteboard and markers**
- **Assignments** for 5 student groups
- **GIF** of software development gone awry (https://g.redditmedia.com/tPgaGgDXL0yZyLe_4pjr1ZO2_qpIOk8t5eopSErOwVk.gif?w=320&s=c5345cd1b395f874f92f51e2509e97ae)

Pacing Guide

Section	Total Time
Bell-work and attendance	5min
Instruction	15min
Student mini-lesson preparation	10min
Student mini-lesson delivery	10min
Practice Activity	10min

Procedure

Your hook for today is an example of what happens when we don't stay flexible with our programming. Have the gif up and looping, and explain that this image is an illustration of what can happen to our programs if we don't make them adaptable.

Bell-work and Attendance [5 minutes]

Instruction [15 minutes]

1. Begin class with a lecture/discussion that introduces parameters
    - Part of the reasons that humans are so good at solving new problems is that they generalize the solution to whole categories of problems.
    - The book gives an example of walking as a generalizable task—walking 20 steps and walking 10 steps can be described as the task of "walking forward," and the part that varies (we call this a parameter) is the number of steps.
        - In programming, when we make code flexible by "**parameterizing**" parts of the task that are likely to change, we end up with programs that are shorter, easier to understand for other coders, better organized, and reusable.

    * Imagine how difficult life would be if you had to separately learn all the movements required to walk 10 steps down that runway in addition to the procedure for walking 20 steps it takes to get to the stage? Instead, your brain computes a general rule something like "walk only the number of steps to the next obstacle," then (ideally) your eyes and ears input how many steps that should be.

    * Java is the same way—it takes less memory and computing power to execute a program if you write code that is flexible/reusable with different parameters. (In fact, in this chapter we're going to learn how to use user input—like the information we get from our eyes and ears—to influence the behavior of the programs we write!)

  – What are some other behaviors that we "parameterize" every day? (If students need help getting started, suggest:

    * Braiding hair: different heads require a different number of braids, but the braiding is always the same procedure.

    * Turning on the stereo or TV: the methods to turn on the TV, radio, or stereo are always the same, but you adjust the volume to different levels depending on the time of day. What would the parameters be in this example? (Time, volume)

2. If your students had a problem from the last couple of weeks that drove them crazy, use that example instead, since they'll already have a clear memory of it. Otherwise, review this programming example from the book:

   - In the last chapter, we inserted spaces into a drawing by calling a method:

     ```
     writeSpaces();
     ```

   - If we wanted to tell method writeSpaces to output 10 spaces, we might decide to declare a variable:

     ```
     int number = 10;  // Can anyone explain why this won't work?
     writeSpaces();
     ```

     → This won't work because `number`'s scope is outside the `writeSpaces` method.

   - Instead we parameterize the number of spaces by changing the method header as highlighted below (this used to be empty parentheses, remember?)

     ```
     public static void writeSpaces (int number) {
         for (int i = 1; i <= number; i++) {
             System.out.print(" ");
         }
     }
     ```

   - Now when we call the method, we can (and must!) include the parameterized value:

     ```
     writeSpaces(10);
     writeSpaces();          // ERROR: No parameter specified
     writeSpaces(24901);
     writeSpaces(9/3-2);
     ```

   - On the board or in Eclipse, point out the difference between an actual parameter (argument) and a formal parameter (or just "parameter"). Point to a few different examples of each on sample code, asking students which one is which, and how they know.

Student Mini-Lesson Preparation [10 minutes]

1. Students are going to get a chance to teach the second half of class—the mechanics of parameters. Assign each group one section of the sample code in section 3.1 "Mechanics of Parameters."

   Student groups should take 10 minutes to review the program, read the example on the pages following the example, then figure out how they want to explain to class what values are being stored in the computer's memory.

2. On the board or overhead, give students a few things they should consider in planning their mini lesson:

a. Who is going to speak when?

b. How are you going to illustrate the flow of control?

c. What do you need to have up on the board to illustrate your mini-lesson, and who is in charge of writing it out?

d. Where and how will you feature the output produced by your code segment?

Student Mini-Lesson Delivery [10 minutes]

Have student groups sequentially teach through the example in the book, demonstrating the changes to the stored value, predicting output, and tracing the flow of control.

Practice Activity (if needed) [10 minutes]

If the majority of students are understanding the content, allow students to work individually on practice self-check problems:

a. Self-Check 3.1: methodHeaderSyntax
b. Self-Check 3.2: MysteryNums
c. Self-Check 3.3: Oops3-errors

Accommodation and Differentiation

If students are struggling with content, you might opt to work on the practice probme as a whole group setting. In some classes, the teaching exercise might take an entire class period.

- To keep from losing too much instructional time, remind students of their presentation time limit while they are planning, and during their presentation.
- Check students as they are coordinating their mini-lesson; offer feedback on the timing of their lesson.
- Use a timer during student lessons, and hold up a "1 minute" warning sign to keep students on pace. This can become a fun/silly challenge if that suits your teaching style. Use a buzzer or gong to keep the lessons on schedule in a non stressful way.

If students are speeding through the content, encourage them to complete one of the following projects (depending on how much time is available):

- Create a poster for the classroom that diagrams/illustrates how to correctly parameterize a method.
- Tackle Ch.3 Programming Project #1 for extra credit. (This might be completed over the course of several days.)

Misconceptions

- When passing parameters to methods, students may include the type when the parameter is a variable: `whiteSpace (int x);`. The student may be pattern matching method declaration with method invocation.

Video

- BJP 3–1, *Mechanics of Parameters* http://media.pearsoncmg.com/aw/aw_reges_bjp_2/videoPlayer.php?id=c3-1
- CSE 142, *Single Parameters* (21:05-33:05) https://www.youtube.com/watch?v=otR7yEbRZAs&start=1265
- CSE 142, *Multiple Parameters* (33:06-50:0) https://www.youtube.com/watch?v=otR7yEbRZAs&start=1985
- CS Homework Bytes, *Functions and Parameters, with Komal* https://www.youtube.com/watch?v=UuZCQWErV-A

Forum discussion

Lesson 3.01 Parameters (TEALS Discourse account required)