Objectives Students will be able to…

* **Use** procedural decomposition to plan complex programs using structure diagrams.
* **Manage** complexity by using method calls

Assessments Students will...

* **Complete** Practice-It problems

Homework Students will...

* **Read** HW 1.5
* **Complete** Ch.1 Exercises 11, 12, 14, 16

# Materials & Prep

* **Projector and computer** (if you are able to/opt to use Eclipse with your students)
* **Whiteboard and marker**
* **Overly complicated diagram** (<http://tinyurl.com/y8a7cry>)

# Pacing Guide

|  |  |
| --- | --- |
| Section | Total Time |
| Bell-work and attendance | 5min |
| Introduction and note-taking | 15min |
| Practice-It questions | 25min |
| Students trade work, check, and turn in | 5min |

# Procedure

*This class introduces many new, intertwined concepts in one class period. These concepts will be re-taught in the next class, but you should be aware that your students have a lot of information to absorb in a short amount of time. This lesson will be a good litmus test—if students have been doing their reading and homework, the class should move along smoothly. If students are not completing the readings, you will probably only get through ~50% of the material. If needed, use this opportunity to convince students of the pace and commitment level required for the class.*

## Bell-work and Attendance [5 minutes]

## Introduction and Note-Taking [15 minutes]

1. Have the **complicated algorithm** up on the board, or printed out for students to pass around. If you have any confusing furniture assembly manuals, or overly complicated directions, bring those to pass around too.

Start class off with a whole group discussion about why the instructions or diagrams are confusing, and ask students what strategies could be used to make them easier to understand. The diagram listed in materials is an actual slide from the Pentagon, and illustrates how too much complexity can cause all meaning to be lost. Guide the conversation towards decomposition to begin your lecture:

* **Decomposition**: dividing a problem into smaller, more manageable pieces.

**V**

* **Procedural decomposition**: dividing a whole program into a series of individual steps or actions to program 1 at a time.
* **Structure diagram**: a way of organizing your approach to building a larger program. Ask students to help you draw a structure diagram for a program with the output shown below:

OUTPUT: STRUCTURE DIAGRAM:

+-------+

Stacked blocks

draw a box insert a blank line

draw a horizontal line draw 2 vertical lines

| |

| |

+-------+

+-------+

| |

| |

+-------+

+-------+

| |

| |

+-------+

2. Have students take 3 minutes to write the DrawBoxes program the long way:

public class DrawBoxes {

public static void main (String[] args){

System.out.println(“+-------+”);

System.out.println(“| |”);

...

3. Point out to students that anything they would cut and paste to save time on creating would make a good unit to turn into a “static method”

* **Static method**: a block of Java statements that is given its own name (ask students to point to a Java statement)
  + Has the same structure we’re familiar with from the main method we already wrote, but we give it a different name than “main”

4. Ask students what simple unit we should build into a static method (have them refer to the structure diagram), and have them suggest a name for the method.

5. Rewrite the method as a class, then show students how to write methods in main that call the new static method. Make sure that students insert the println statements between each method call. It should look something like this:

public class DrawBoxes3 {

public static void drawbox(){

System.out.println(“+-------+”);

System.out.println(“| |”);

System.out.println(“| |”);

System.out.println(“+-------+”);

}

public static void main(String [] args){

drawBox();

System.out.println();

drawBox();

System.out.println();

drawBox();

}

}

6. Call on a student to come to the board and physically trace the flow of control with the marker. Start them off by pointing out that Java always starts with the main method.

* If the student seems nervous, encourage the rest of the class to call out directions to the student. Make sure students are drawing the flow of control on their own notes as well.

## Practice-It questions [25 minutes]

1. Have students complete the following Practice-It questions:

a. Tricky

b. Strange

c. Confusing

d. Lots-of-errors

If you do not have access to computers in your classroom, copy the Practice-it questions to a worksheet and have students complete the practice problems by writing out the answers and using their error-checking algorithm sheets.

Some students will jump right into this activity, but others will need additional assistance from you.

At this point in the school year, we suggest that you insist on structure diagrams with each program. Structure diagrams encourage algorithmic thinking and the creation of efficient solutions; both of which are vital computational thinking skills.

If need be, work on “Tricky” as a whole group, so you can model the correct steps to approaching a problem. If your class decides on an algorithm for “predict the output” type questions, have a student make that algorithm into a poster for the whole class to refer to.

## Students trade work, check, and hand in [5 minutes]

1. Have students trade work and check each other’s responses on Practice-It before submitting.

# Accommodation and Differentiation

In ELL classrooms, this lesson should be delivered over the course of 2 days. Extra time should be spent drilling static methods, methods that call other methods, and flow of control. Try adapting some of the examples from the book to include students’ native language so they can focus on structuring code instead of translating language. One easy way to introduce familiar, repetitive content would be to have students output the lyrics to a song with a refrain. For a physical activity to demonstrate flow-of-control, check out lesson plan 1.6.

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

* Complete the remaining Strange 2, Strange 3, Complicated 2 and Complicated 3 Practice-It questions
* Have the student write a sample test question with output that can be written using method calls. Be sure they include the answer key with the sample question!