Java Chapter 4 Part 1

* Loops: Syntax and Usage
* CIS 255 • Shelby-Hoover Campus

Loops

* The syntax from chapter 2 was limited to sequences of statements with no alternate flows or repetition
* The structures from chapter 3 allow a program to evaluate a Boolean expression one time to determine which set of statements to execute
* To repeat a set of statements without manually entering the code multiple times, the program requires a loop (repetition, iteration) structure that evaluates a Boolean expression repeatedly

Increment and Decrement

* Chapter 2 mentions two ways to add 1 to, or subtract 1 from, a variable
  + Regular Assignment with Addition / Subtraction:  
    variable = variable + 1;  
    variable = variable – 1;
  + Combined Assignment:  
    variable += 1;  
    variable -= 1;
* Java includes special unary operators specifically for the operation of adding 1 (**increment**) or subtracting 1 (**decrement**)
  + Increment: variable++;
  + Decrement: variable--;
* Increment and decrement are frequently used with loops

Prefix vs. Postfix

* The increment and decrement operators can be written in either prefix or postfix mode
  + Prefix: ++variable;
  + Postfix: variable++;
* In a standalone statement, there is no difference between prefix and postfix
* The increment and decrement operators can be embedded inside another expression, in which case prefix versus postfix determines when the embedded operation is performed
  + Prefix: increment / decrement the variable first, then use the new value of the variable in the overall expression
  + Postfix: Use the existing value of the variable in the overall expression, then increment / decrement the variable
* Embedding the increment / decrement operation is rarely required

The while Loop

* The simplest loop in Java (the while loop) evaluates the Boolean expression to determine whether or not to execute a set of statements (the **loop body**)
  + If the Boolean expression is true, the loop body is executed, and the Boolean expression will be evaluated again
  + If the Boolean expression is false, the loop body will be skipped
* Each execution of a loop body is known as an **iteration**
* Basic syntax of the while loop:  
  while (Boolean\_expression)  
  {  
   // loop body goes here  
  }
* Note that as with the if statement, there is **no semicolon** at the end of the while loop header

while Loop Examples

* Printing the word “Llama” ten times before printing the word “Duck”:  
  int howMany = 1;  
    
  while (howMany <= 10)  
  {  
   System.out.println("Llama");  
   howMany++;  
  }  
    
  System.out.println("Duck");
* Detecting an excessively-high temperature: CheckTemperature.java (Code Listing 4-4)

Notes about the while Loop

* The variable used in the Boolean expression is known as the **loop control variable**
  + This variable must have a value (assigned or from input) before the program reaches the Boolean expression
  + It should also be updated (arithmetic or input) in the loop body
* Failure to properly use a loop control variable may result in an **infinite loop** that never ends
* Comparison with if:
  + Same: Parentheses around the Boolean expression
  + Same: No semicolon after the Boolean expression
  + Same: Curly brackets are optional for a single-statement body
  + Different: No else after the while loop body
  + Different: Control returns to the Boolean expression after execution of the while loop body

Input Validation

* If a user enters invalid input data, the program should not continue until the user corrects the problem
* This can be accomplished via a while loop:  
    
  import java.util.Scanner;  
    
  public class MonthInputValidator  
  {  
   public static void main(String[] args)  
   {  
   int month;  
   Scanner kb = new Scanner(System.in);  
    
   System.out.print("Enter numeric birth month: ");  
   month = kb.nextInt();  
    
   while ((month < 1) || (month > 12))  
   {  
   System.out.print("Must be between 1 and 12. Try again: ");  
   month = kb.nextInt();  
   }  
    
   System.out.println("Month entry succeeded.");  
   }  
  }
* Another example: SoccerTeams.java (Code Listing 4-5)

The do-while Loop

* Because the while loop is a **pretest** loop, its body may never execute
* Java has a **posttest** loop, the do-while loop, that executes the loop body one time before evaluating the Boolean expression to determine if another iteration is required
* Basic syntax of the do-while loop:  
    
  do  
  {  
   // loop body goes here  
  } while (Boolean\_expression);
* Note the semicolon after the Boolean expression (do-while is the exception, not the rule)

do-while Loop Examples

* Printing the word “Llama” at least once, repeating as indicated by the user:  
    
  String input;  
  char rep;  
    
  do  
  {  
   JOptionPane.showMessageDialog(null, "Llama");  
   input = JOptionPane.showInputDialog("Another llama? (Y or N)");  
   rep = input.charAt(0);  
  } while ((rep == 'Y') || (rep == 'y'));
* Allowing multiple students to average test scores: TestAverage1.java (Code Listing 4-6)

Notes about do-while

* There is no Boolean expression at the top of the do-while loop (only the key word do)
* The loop body will be executed one time even if the Boolean expression at the bottom is false; the expression will not be evaluated until after the first iteration
* Repetition continues until the Boolean expression evaluates to false
* Don’t forget the semicolon after the closing parenthesis (only for do-while)

The for Loop

* A loop is said to be count-controlled when the number of iterations is predetermined (as opposed to a conditional loop that has a number of iterations that cannot be known prior to execution)
* Java’s third loop structure, the for loop, combines the key actions of loop control in the loop header
* Basic syntax of the for loop:  
    
  for (start\_action; test; update)  
  {  
   // loop body goes here  
  }
* The *start action* is performed once at the very beginning
* The *test* section, where the Boolean expression is written, is evaluated before the first iteration (pretest) and each one that follows
* The *update* action is performed after each iteration before the next evaluation of the Boolean expression

for Loop Example

* Counting out ninety-nine llamas:  
    
  int curr;  
  for (curr = 1; curr <= 99; curr++)  
   System.out.println("Llama #" + curr);
* Displaying the squares of the numbers 1 through 10: Squares.java (Code Listing 4-7)
* As with the other loops, a for loop body may not require brackets, as there may be only one statement in the body (the update step is written in the header rather than in the body)

Notes about the for Loop

* If you place an update action in the for loop header, don’t include it in the loop body, or it will be performed twice per iteration
* The update action may be any statement, but it should modify the value of the loop control variable in some manner
* You may declare a variable in the initial assignment section of the loop, but that variable will be local to the loop and will no longer exist after the loop terminates
* To include multiple assignment or update statements, separate them with commas:  
    
  for (a = 1, b = 9; a <= b; a++, b--)  
   System.out.println(a \* b);
* The Boolean expression may be simple or compound
* The initial assignment and update sections may be empty if their actions are performed elsewhere, but the semicolons separating the sections are still required

Loops Applied: Accumulation

* If a program obtains input during each iteration of a loop, the input values may be added to a sum variable known as an **accumulator**
* The accumulator variable should be initialized to 0
* Example: Taking the sum of four grades  
    
  import javax.swing.JOptionPane;  
    
  public class GradeSum  
  {  
   public static void main(String[] args)  
   {  
   int sum = 0, grade, count;  
   String inputHolder;  
    
   for (count = 1; count < 5; count++)  
   {  
   inputHolder = JOptionPane.showInputDialog(  
   "Enter grade #" + count + ":");  
   grade = Integer.parseInt(inputHolder);  
   sum = sum + grade;  
   }  
    
   JOptionPane.showMessageDialog(null, "The sum of the grades is " +   
   sum);  
   }  
  }

Loops Applied: User Control

* A programmer may not know how many values the user wants to input ahead of time, and it may vary from one run to the next
* By using a **sentinel value**, the program allows the user to indicate when there is no more input to process
* Example:  
    
  import java.util.Scanner;  
    
  public class GradeSumSentinel  
  {  
   public static void main(String[] args)  
   {  
   int sum = 0, grade;  
   Scanner kb = new Scanner(System.in);  
    
   System.out.print("Enter a grade (-1 to quit): ");  
   grade = kb.nextInt(); // priming read  
   while (grade != -1)  
   {  
   sum += grade; // add value just read to total  
   System.out.print("Enter a grade (-1 to quit): ");  
   grade = kb.nextInt(); // loop read  
   }  
    
   System.out.println("Grade Sum: " + sum);  
   }  
  }
* Another Example: SoccerPoints.java (Code Listing 4-11)