Introduction to Computers and Programming

**Lecture 9: More Loops**

Chris Conway
New York University
Road map

• Sentinel loop case study
• do-while loops
• Nested control structures
• for loops
• Good practice and style using for loops

• Reading:
  – Liang: Chapter 4 : 4.4, 4.5
review

• What are the three elements of a while loop?
• Are you guaranteed to execute the body of a while loop at all?
• What is an infinite loop?
• True or False: You never want to use an infinite loop?
review

• What is a sentinel value?
• How do you pick a good sentinel value?
• Why shouldn’t you generally use floats or doubles as control variables in loops?
• What are the two possible values of a Boolean variable?
What is the output of this switch statement?

```java
int a = 90;
switch (a) {
  case 80:
    System.out.println (80);
  case 90:
    System.out.println (90);
  case 100:
    System.out.println (100);
}
```
Case Study: Using Sentinel Controlled Loops
Problem becomes:

*Develop a class-averaging program that will process an arbitrary number of grades each time the program is run.*

- Unknown number of students
- How will the program know to end?

Use sentinel value

- Loop ends when user inputs the sentinel value
- Sentinel value chosen so it cannot be confused with a regular input (such as \(-1\) in this case)
Formulating Algorithms with Top-Down, Stepwise Refinement

• Top-down, stepwise refinement
  – Begin with a pseudocode representation of the top:
    *Determine the class average for the quiz*
  – Divide top into smaller tasks and list them in order:
    *Initialize variables*
    *Input, sum and count the quiz grades*
    *Calculate and print the class average*

• Many programs have three phases:
  – Initialization: initializes the program variables
  – Processing: inputs data values and adjusts program variables accordingly
  – Termination: calculates and prints the final results
Formulating Algorithms with Top-Down, Stepwise Refinement

• Refine the initialization phase from *Initialize variables* to:
  
  *Initialize total to zero*
  
  *Initialize counter to zero*

• Refine *Input, sum and count the quiz grades* to
  
  *Input the first grade (possibly the sentinel)*
  
  *While the user has not as yet entered the sentinel*
  
  *Add this grade into the running total*
  
  *Add one to the grade counter*
  
  *Input the next grade (possibly the sentinel)*
Refine *Calculate and print the class average* to

*If the counter is not equal to zero*

*Set the average to the total divided by the counter*

*Print the average*

*else*

*Print “No grades were entered”*
Initialize total to zero
Initialize counter to zero

Input the first grade (possibly the sentinel)

While the user has not as yet entered the sentinel
 Add this grade into the running total
 Add one to the grade counter
 Input the next grade (possibly the sentinel)

If the counter is not equal to zero
 Set the average to the total divided by the counter
 Print the average
else
 Print “No grades were entered”

Fig. 4.8 Class-average problem pseudocode algorithm with sentinel-controlled repetition.
// Fig. 4.9: Average2.java
// Class-average program with sentinel-controlled repetition.
import java.text.DecimalFormat; // class to format numbers
import javax.swing.JOptionPane;

public class Average2 {

    public static void main( String args[] ) {
        int total; // sum of grades
        int gradeCounter; // number of grades entered
        int grade; // grade value

        double average; // number with decimal point for average
        String gradeString; // grade typed by user

        // initialization phase
        total = 0; // initialize total
        gradeCounter = 0; // initialize loop counter

        // processing phase
        // get first grade from user
        gradeString = JOptionPane.showInputDialog("Enter Integer Grade or -1 to Quit:");

        // convert gradeString to int
        grade = Integer.parseInt( gradeString );
    }
}
// loop until sentinel value read from user
while ( grade != -1 ) {
    total = total + grade;
    gradeCounter = gradeCounter + 1;
}

// get next grade from user
gradeString = JOptionPane.showInputDialog("Enter Integer Grade or -1 to Quit:");

// convert gradeString to int
grade = Integer.parseInt( gradeString );

} // end while

// termination phase

// if user entered at least one grade...
if ( gradeCounter != 0 ) {

    // calculate average of all grades entered
    average = (double) total / gradeCounter;

    // display average with two digits of precision
    JOptionPane.showMessageDialog( null,
        "Class average is " + average ,
        "Class Average", JOptionPane.INFORMATION_MESSAGE );

} // end if part of if...else
else // if no grades entered, output appropriate message
    JOptionPane.showMessageDialog(null, "No grades were entered", "Class Average", JOptionPane.INFORMATION_MESSAGE);

    System.exit( 0 );  // terminate application

} // end main

} // end class Average2
Case Study: Nested Control Structures
Nested control structures

• Problem
  – A college has a list of test results (1 = pass, 2 = fail) for 10 students
  – Write a program that analyzes the results
    • If more than 8 students pass, print "Raise Tuition"

• Notice that
  – The program must process 10 test results
    • Counter-controlled loop will be used
  – Two counters can be used
    • One for number of passes, one for number of fails
  – Each test result is a number—either a 1 or a 2
    • If the number is not a 1, we assume that it is a 2
Nested control structures

• Top level outline
  
  *Analyze exam results and decide if tuition should be raised*

• First Refinement

  *Initialize variables*

  *Input the ten quiz grades and count passes and failures*

  *Print a summary of the exam results and decide if tuition should be raised*

• Refine *Initialize variables* to

  *Initialize passes to zero*

  *Initialize failures to zero*

  *Initialize student counter to one*
Nested control structures

• Refine *Input the ten quiz grades and count passes and failures* to

  *While student counter is less than or equal to ten*
  *Input the next exam result*
  *If the student passed*
    *Add one to passes*
  *else*
    *Add one to failures*
  *Add one to student counter*

• Refine *Print a summary of the exam results and decide if tuition should be raised* to

  *Print the number of passes*
  *Print the number of failures*
  *If more than eight students passed*
    *Print “Raise tuition”*
Initialize passes to zero
Initialize failures to zero
Initialize student to one

While student counter is less than or equal to ten
    Input the next exam result
    If the student passed
        Add one to passes
    else
        Add one to failures
    Add one to student counter

Print the number of passes
Print the number of failures

If more than eight students passed
    Print “Raise tuition”

Fig 4.10 Pseudocode for examination-results problem.
// Fig. 4.11: Analysis.java
// Analysis of examination results.
import javax.swing.JOptionPane;

public class Analysis {

    public static void main( String args[] )
    {
        // initializing variables in declarations
        int passes = 0;       // number of passes
        int failures = 0;     // number of failures
        int studentCounter = 1;  // student counter
        int result;           // one exam result

        String input;         // user-entered value
        String output;        // output string

        // process 10 students using counter-controlled loop
        while ( studentCounter <= 10 ) {

            // prompt user for input and obtain value from user
            input = JOptionPane.showInputDialog(
                "Enter result (1 = pass, 2 = fail)" );

            // convert result to int
            result = Integer.parseInt( input );

            // if result 1, increment passes; if...else nested in while
            if ( result == 1 )
                passes = passes + 1;
        }
    }
else // if result not 1, increment failures
    failures = failures + 1;

    // increment studentCounter so loop eventually terminates
    studentCounter = studentCounter + 1;

} // end while

// termination phase; prepare and display results
output = "Passed: " + passes + "\nFailed: " + failures;

// determine whether more than 8 students passed
if ( passes > 8 )
    output = output + "\nRaise Tuition";

JOptionPane.showMessageDialog( null, output,
                       "Analysis of Examination Results",
                       JOptionPane.INFORMATION_MESSAGE );

System.exit( 0 ); // terminate application

} // end main

} // end class Analysis
do/while Loop
The **do/while** Repetition Structure

• The **do/while** repetition structure
  – Similar to the **while** structure
  – Condition for repetition tested after the body of the loop is performed
    • All actions are performed at least once
  – Format:

```markdown
  do {
    statement(s);
  } while ( condition );
```
4.8 The do/while Repetition Structure

- Flowchart of the do/while repetition structure
Summary of Looping so far
Summary of Looping

• Two broad types of loops:
  – Counter-controlled repetition
    • A counter controls the number of repetitions.
    • Also known as a *definite repetition*, because we know in advance how many times the loop will be executed.
  – Sentinel-controlled repetition
    • A sentinel controls the number of repetitions
    • Also known as *indefinite repetition*, because we do not know in advance how many times the loop will be executed.

• In either case, watch out for infinite loops!

• If your program requires some kind of loop, first determine which *kind* of loop you want.
Summary of Looping

• Once you know which kind of loop you want, determine which while loop you want:
  – While loops
    • condition is tested first; then action occurs.
    • While loops are much more common than do/while loops.
  – Do/while loops
    • action is run first; then, condition is tested.
    • Use this if you want to make sure that your loop is guaranteed to be run at least once.
For Loops
Parts of a Loop (reminder)

• Every loop will always contain three main elements:
  – Priming: initialize your variables.
  – Testing: test against some known condition.
  – Updating: update the variable that is tested.
Loop types (reminder)

• Indefinite Loop:
  – You do not know ahead of time how many times your loop will execute.
  – For example, you do not know how many books a person might order.

• Definite Loop:
  – You know exactly how many times you want the loop to execute.
  – not at compile time necessarily
For loops

• Another type of loop in Java is the `for` loop.
• It is very good for definite loops
• All the parts (priming, testing and updating) are in one place.
• format:
  ```java
  for (prime expression; test expression; update expression)
  ```
• Since the expressions are all in one place, many people prefer `for` to `while` when the number of iterations is known.
Basic For Loop Syntax

• **for** loops are good for creating definite loops.

```java
int counter;

1. Priming: Set the start value.
2. Test Condition: Set the stop value.
3. Update: Update the value.

for (counter=1; counter <= 10; counter++)
    System.out.println (counter);
```

Note that each section is separated by a semicolon.
for Loop Flowchart

1: Priming
Set counter = 1

2: Test
counter <= 10

TRUE
Body: print counter

FALSE
3: Update counter++;
**Infinite Loop**

- You can still end up with an infinite loop when using for loops

```java
for (counter = 0; counter <= 10; counter--)
```
For Loop Variations

• The limit can be a variable:
  
  ```c
  for ( i = 1; i <= limit; i++)
  ```
  
  This counts from 1 to limit

• Update may be negative:
  
  ```c
  for (i = 100; i >= 1; i--)
  ```
  
  This counts from 100 to 1.

• Update may be greater than 1:
  
  ```c
  for (i = 100; i >= 5; i -= 5)
  ```
  
  This counts from 100 to 5 in steps of 5
The for Structure: Notes and Observations

• Arithmetic expressions
  – Initialization, loop-continuation, and increment can contain arithmetic expressions. If \( x \) equals 2 and \( y \) equals 10

\[
\text{for ( } j = x; j <= 4 * x * y; j += y / x \) }
\]

is equivalent to

\[
\text{for ( } j = 2; j <= 80; j += 5 \) }
\]

• Notes about the for structure:
  – If the loop continuation condition is initially \text{false}
    • The body of the for structure is not performed
    • Control proceeds with the next statement after the for structure
  – Control variable
    • Often printed or used inside for body, but not necessary
The Comma Operator

• By using commas, you can put more than one statement in priming or updating expression

```
for (i = 100, y = 0; i >= 1; i--)
or
for (i = 1; j + i <= 10; i++, j++)
{
    code;
}
```

• Commas known here as *comma operators*
Warnings

• Do not use a float or double for the counter
  – May result in imprecise counter values and faulty evaluation for loop termination purposes

• Don’t use commas instead of semicolons to separate the components of the for loop
  – (very common error)

• As in the if and while, do not put a semicolon ; right after the parentheses – will be an empty loop!
Off-by-one error

• In the first example, shown here, the following
  
counter < 10
  
would execute 9 times, not the desired 10 times

for (counter = 1; counter <= 10; counter++)
{
    System.out.println (counter);
} /* end for counter */
Help avoid off-by-one errors

• Try to make your conditions in the form \( \leq \) not \(<\)
  – Avoid code like counter \(< 11\) or counter \(< 10\)

• There are times when we will break this rule
  – arrays
  – Strings
Good Programming Practices

• Do not change the value of the counter variable in your loop.
  – Do not attempt to manually adjust it to force an exit
  – Can cause subtle errors that are difficult to catch
  – Instead change your logic

• Do not put other expressions in the for control structure
  – Manipulations of other variables should appear before or within the body of the loop depending on whether or not you want to repeat them

• Put a blank line before and after each major control structure

• Try to limit nesting to three levels, if possible
  – More than three levels of indentation can get confusing

• Limit the for control header to one line if possible