# *Programming I (420-B10-HR)*

# *Lab 11 – The while Statement; Constructors*

Date assigned: Tuesday, November 10, 2015

Date due: **Tuesday, November 10, 2015**

**Learning Objectives**

Upon successful completion of this lab exercise, the student will be able to:

1. use a **while** loop to create a table;
2. use a **while** loop to read until the end of the input data;
3. use a **while** loop to do error checking on input.
4. create a nested loop;
5. define and use a constructor method;
6. overload a constructor;

**Commands Used:**

**while** **(***condition***)**

*statement***;**

**while** **(***condition***)**

**{**

*block of statements to be executed while the condition is true***;**

**}**

**To Be Handed In:**

1. The files ***username*\_B10\_L11\_while** folder should be zipped and uploaded to **Moodle**. Make sure that you have reformatted all your Java files to make them properly indented.
2. The **Lab 11 Review Quiz** should be completed on **Moodle**.
3. The **Lab 11 Review Questions** should be handed in.

**To Start:**

1. Download and unzip the **B10\_L11\_while** folder from **Moodle** to your **H:\420-B10\Labs** folder. Rename it to ***username*\_B10\_ L11\_while**.
2. Start **Eclipse**. Use your **H:\420-B10\Labs** folder as the workspace.
3. Create a **New Java Project** called ***username*\_B10\_ L11\_while**.

# A while loop using a sentinel value

***Purpose:*** Use a loop to input a number of values until a sentinel value is encountered.

***To Do:***

## Open **TestSentinel** in the **whileLoop** package. It contains a loop to read a number from the user and uses the Accumulator class to accumulate the number of numbers, the sum and the product. Run **TestSentinel**. Use any number of values as input. When you wish to stop inputting, enter -999.

***Questions:***

What is the last number read? \_\_-999\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Is the last number read included in the sum? \_\_no\_\_\_\_\_\_\_\_

Which line is the initializer for the loop?

\_\_\_\_line 26\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

What is the boundary condition for the loop?

\_\_\_!-999\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Sentinel*

A sentinel value is an input value that is used to determine when to stop looping. It should be a value that could not possibly be valid input (e.g. -999 for a mark.) A sentinel is normally used when the number of input values is not known.

What is the sentinel value for this loop? \_-999\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Which line is the updater for the loop?

\_\_\_\_\_\_line 31\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Comment the updater statement.

***Questions:***

What happens?

\_\_It creates an infinite loop\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Why?

\_\_Because you can’t update the number and you won’t be able to enter the sentinel value.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Uncomment the updater.

## Comment the loop initializer.

***Questions:***

What happens?

\_\_The first number inputted is outputted as the sum. You’re only able to enter two values.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Why?

\_\_\_Because it’s no longer a loop, it’ll only happen once\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Priming read*

In order for the tested value to have a value the first time the loop is entered, the first value must be read before the loop begins.

## Uncomment the statement that reads in the first number.

## Change the sentinel to -1 and rerun the program.

## Add a method called **calculateAverage()** to the **Accumulator** class. It should calculate the average using the sum and count instance variables. The average should be a double precision number.

## In **TestSentinel**:

### if the count is greater than 0, use the **printf()** method to print the value returned from the **calculateAverage()** method with two decimal positions

### otherwise, display a message stating that the average is undefined.

## Test your change on the following test cases.

| **Scenario** | **Input Data** | **Expected Output** |
| --- | --- | --- |
|  | **number** |  |
| 1. One valid number is input | 178  -1 | Count is 0 Number is 178  The value of count after the loop is 1  The sum is 178  The product is 178  The average is 178.00 |
| 2. More than 1 number is input | 5  13  7  29  -1 | Count is 0 Number is 5  Count is 1 Number is 13  Count is 2 Number is 7  Count is 3 Number is 29  The value of count after the loop is 4  The sum is 54  The product is 13195  The average is 13.50 |
| 3. -1 is the first number input | -1 | The value of count after the loop is 0  The sum is 0  The product is 1  The average is undefined |

# A while loop to do error checking

***Purpose:*** Use a loop to repeatedly input a value until it is valid.

***To Do:***

## Open **Validator** and run it for 700, 101, 100, -1 and 6.

***Questions:***

What happens when 700 is entered?

700 is not less than 100. Please Re-enter: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***To Do:***

## Modify the program to accept only numbers greater than or equal 0 and less than or equal 100. Execute the modified program and the numbers above.

# While loop Review

***Purpose***: Review how to use a while loop to repeat instructions and read until a sentinel value.

To Do:

## Create a class called **BottlesOnTheWall** in the **miscellaneous** package. Your class will contain a main() method that will use a while loop to print out all the lyrics to the song “99 bottles of beer on the wall”. The song finishes when there are no bottles left. (You may substitute “milk” or “coke”.) The first few lines of output using milk would be:

99 bottles of milk on the wall. 99 bottles of milk.

Take one down and pass it around. 98 bottles of milk on the wall.

98 bottles of milk on the wall. 98 bottles of milk.

Take one down and pass it around. 97 bottles of milk on the wall.

97 bottles of milk on the wall. 97 bottles of milk.

Take one down and pass it around. 96 bottles of milk on the wall.

The last few lines of output using milk would be:

2 bottles of milk on the wall. 2 bottles of milk.

Take one down and pass it around. 1 bottles of milk on the wall.

1 bottles of milk on the wall. 1 bottles of milk.

Take one down and pass it around. 0 bottles of milk on the wall.

## Open the **GuessingGame** class. It uses **Math.random()** to generate a random letter between ‘a’ and ‘z’ and asks the user to guess the letter. Run the program to see how it works.

## Modify **GuessingGame** so that it tells the user whether her guess is before or after the letter in the alphabet and lets them guess until they guess correctly. Count the number of guesses. Here is a sample run:

I have picked a letter between a and z. What do you think it is?

c

Sorry - you are wrong. The letter is after c in the alphabet. Guess again.

k

Sorry - you are wrong. The letter is after k in the alphabet. Guess again.

m

Sorry - you are wrong. The letter is after m in the alphabet. Guess again.

s

Congratulations! You guessed it with 4 guesses.

# Nested Loops

***Purpose***: Learn how to use a nested while loop.

To Do:

## The user wants to be able to replay the **GuessingGame** as many times as she wants without having to restart the program. To allow this, we are going to put another loop around the guessing loop. The outer loop will use a sentinel value to determine whether or not to keep looping. To do this:

### Open **GuessingGame** in the **Miscellaneous** package.

### Add a char variable called **again**.

### Initialize **again** to ‘Y’.

### Add a while statement to loop as long as **again** is ‘Y’.

### Before exiting the loop, ask the user if she wants to play again. Read the answer into **again**. Convert it to uppercase as follows:

**again = Character.toUpperCase(again);**

## Test your changes.

# While Review Questions

***Purpose:*** Review while loops.

***To Do:***

## Complete the **Lab 11 Review Questions**.

# Default Constructors

***Purpose***: Learn how to create a default constructor.

Definitions:

A constructor method is used to create an object for a class

A default constructor is a constructor with no parameters.

To Do:

## Open **College.java** and **Course.java**.

## Run **College.java**. What is the output?

The new course number is null\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The course name is null\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ At the moment 0 students have completed the course.\_\_

## Add the following default constructor to the **Course** class after the instance variables are declared:

public Course()

{

System.out.println("In the Course() constructor");

courseNumber = "Unassigned";

courseName = "Unknown";

numStudents = 0;

totalMarks = 0.0;

} // Course()

## Run the program again. What is the output?

In the Course() constructor\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The new course number is Unassigned\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The course name is Unknown \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ At the moment 0 students have completed the course\_\_\_\_\_\_\_

What has changed and why?

An extra println statement has been added and the courseNumber and\_\_\_\_\_ courseName variables have been given a default of ‘unassigned’ and ‘unkown’

## 

# Overloaded Constructors

***Purpose***: Learn how to overload constructors.

Definitions:

The ***signature***of a method consists of its name and the number and type of its formal parameters. A class may not contain more than one method with the same signature.

An ***overloaded method*** is a method with the same name, but a different signature as an existing method in a class.

To Do:

## Add the following method to the **Course** class:

public Course(String number, String name)

{

System.out.println("In the Course(String, String) constructor");

courseNumber = number;

courseName = name;

numStudents = 0;

totalMarks = 0.0;

} // Course(String, String)

This method overloads the Course() constructor method created in the previous part of this lab. It has the same name (Course), but a different signature.

The signature of the first constructor is:

Course()

The signature of this constructor is:

Course(String, String)

## Add a second **Course** object instantiation for **oldCourse** in **main** to:

Course oldCourse = new Course("420-G99","Advanced Gobble Degook");

## Add statements to display the course number, name and number of students in **oldCourse**.

## Run the program again. What is the course number for **newCourse**? \_\_unassigned\_\_\_\_\_\_\_ **oldCourse**? \_\_420-G99\_\_\_\_\_\_\_\_

Which constructor is used to create **newCourse**?

\_Course newCourse = new Course();\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Which constructor is used to create **oldCourse**?

\_Course oldCourse = new Course(“420-G99” , “Advanced Gobble Degook”);\_\_\_

## Add a third constructor to **Course**. It should have three parameters: one for course number, one for course name and one for number of students.

## Instantiate another **Course** object in the **College** class. It should use the constructor you just created. Use any course number, name and number of students. Display the course number, name and number of students for your new course. Test your changes.

# Review

## Complete the **Lab 11 Review Quiz** on the **Programming I** course page in **Moodle**.