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

# *Lab 9 – Boolean operators and Introduction to the Eclipse Debugger*

Date assigned: Tuesday, October 20, 2015

Date due: **Tuesday, October 20, 2015**

**Learning Objectives**

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

1. use Boolean operators in an **if** statement.
2. create and use an enumeration.
3. use de Morgan's Theorem to negate a Boolean expression.
4. use the **Math.random()** method to generate random numbers.
5. use the **Eclipse** debugger to debug a program.

**To Be Handed In:**

1. Your ***username*\_B10\_L09\_Boolean\_Debugger** folder should be zipped and uploaded to **Moodle**.
2. The Moodle review exercise should be completed.

**To Start:**

1. Do the **Lab 9 Terminology Review Quiz** on **Moodle**
2. Download and unzip the **B10\_L09\_Boolean\_Debugger** folder from **Moodle** to your **H:\420-B10\Labs** folder. Rename it to ***username*\_B10\_L09\_Boolean\_Debugger**.
3. Start **Eclipse**. Use your **H:\420-B10\Labs** folder as the workspace.
4. Create a **New Java Project** called ***username*\_B10\_L09\_Boolean\_Debugger**.

# Using the Boolean operators

***Purpose*:** Use the Boolean operators of **and** **(&&)**, **or** **(||)** and **xor (^)** to test compound conditions.

*To Do:*

## Open the **BooleanOperators** class. It contains two Boolean variables A and B and contains four if-else statements to show the values of all the combinations of true and false using the && operator. Run the program and complete the table below:

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A && B** |
| True | True | True |
| True | False | False |
| False | True | False |
| False | False | False |

## Copy the statements to give the values of A && B for all combinations of A and B and change them to use || instead of &&. Run the program and complete the following table:

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A || B** |
| True | True | True |
| True | False | True |
| False | True | True |
| False | False | False |

## Copy the statements to give the values of A && B for all combinations of A and B and change them to use ^ instead of &&. Run the program and complete the following table:

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A ^ B** |
| True | True | False |
| True | False | True |
| False | True | True |
| False | False | False |

## Open and run **MarkChecker** in the **booleanOperators** package. Test it on marks of –16, 0, 56, 100 and 235.

***Question:***

Is the **else** statement of the following statement in the **main()** method ever executed? No\_\_\_

if (me.setFinalMark(mark))

System.out.println("Congratulations you entered a valid mark");

else

System.out.println("Oops, " + mark + " is not a valid mark");

## We want the method **setFinalMark()** in the **Student** class to:

### set **finalMark** to **m** and return a value of **true** if ***0<=m<=100***

### return a value of **false** otherwise.

### Open **Student.java** and add the following if statement at the beginning of the **setFinalMark()** method:

if (m >= 0 && m <= 100)

## Complete the **if** statement in the **setFinalMark()** method so that it works as described in the previous step. Rerun the program and test it on the same marks as before. Is it working correctly now? \_\_Yes\_\_\_\_\_\_

## Change the **&&** in the **setFinalMark()** method to **||**. Run the program again using the same marks. What happens? It stops working.\_\_

Why? \_because all numbers are either above 0 or below 100 \_\_\_\_\_\_\_

## Change the **||** in the **setFinalMark()** method to **^**. Run the program again using the same marks. What happens? Numbers between 0 and 100 are returned false\_\_

Why? \_Because XOR cannot have both statements being true. \_\_\_\_\_

## Change the ^ back to &&.

## Open **DayPlanner**. At the moment it just reads in the day and time. We want to modify it so that it determines what you should be doing at that time. There are four possibilities:

* If the time is 10 pm (22:00) or after and before 6 am (06:00), you should be asleep.
* If it is any other time and it is Saturday or Sunday, it's the weekend.
* If it is not the weekend, and it is 8 am (08:00) or after and before 6 pm (18:00) you should be at school.
* If it is 19:00 or after and before 22:00 on a weekday you should be doing homework.
* If it is any other time you are between activities.

Follow the next two steps to modify the program to do this.

## After reading in the time, determine the hour:

### If the length of the String, **time**, is 2 or less, use **Integer.parseInt()** to convert **time** from a **String** to an **int** and store the result in **hour**:

**hour = Integer.parseInt(time);**

### If the length of the String, **time**, is greater than 2, extract the digits before the colon(:) and convert them to integer using **Integer.parseInt()**. To extract the digits before the colon, use the **substring()** and **indexOf()** methods.

## Write a nested if statement to determine which of the above cases is true. Print the appropriate activity (see table below for sample outputs.) For example, if it is 23:30 on Monday, the output would be:

Enter day abbreviation:

Mon

Enter time:

23:30

Sleep

## Test your program on the following test cases:

| **Scenario** | **Input Data** | | **Expected output** |
| --- | --- | --- | --- |
|  | **Day** | **Time** |  |
| Weekday after 10:00 pm before midnight | Mon | 23:30 | Sleep |
| Weekday after midnight before 6 am | Tues | 3:15 | Sleep |
| Weekend after 10 pm before midnight | Sat | 22:15 | Sleep |
| Weekend after midnight before 6 am | Sun | 5:59 | Sleep |
| Saturday after 6 am and before 10 pm | Sat | 6:13 | Weekend |
| Sunday after 6 am and before 10 pm | Sun | 17:45 | Weekend |
| Weekday at 8 am | Wed | 8 | School |
| Weekday between 8 am and 6 pm | Thurs | 11:50 | School |
| Weekday at 6 pm | Fri | 18:00 | Other |
| Weekday after 6 pm before 10 pm | Mon | 19:00 | Homework. |
| Weekday at 10 pm | Tue | 22:00 | Sleep. |
| Weekday between 6 am and 8 am | Wed | 7:15 | Other |

# Using Enumerations

**Purpose:** To create and use enumerations for a set of options.

To Do:

## We are going to use an enumeration to define the five cases for the **DayPlanner**. Add the following enumeration before the **main()** method header in **DayPlanner.java**:

**enum** Activity { *Weekend*, *School*, *Sleep, Homework, Other*};

This creates a new "data type" called **Activity**. We can now declare variables to be **Activity** and use the different values in tests and output.

## Add a new variable called **currentStatus** after declaring **hour** in the **main()** method. The data type should be an **Activity**.

## Replace the **System.out.println()** statements in your nested **if** statement with an assignment to the appropriate value of **currentStatus**. For example, instead of printing "Sleep", you should have:

currentStatus = Activity.S*leep*;

## Add a println statement to print "It's + **currentStatus +** " time."

## Add a switch statement to test **currentStatus**:

### If it is **School** or **Homework**, print "Work hard!"

### In all other cases, print "Relax!"

# de Morgan's Theorem

**Purpose:** To learn how to properly negate Boolean expressions.

To Do:

## Open the **deMorgansExample** class. It reads in a character. If it is not 'M' or 'F', it should print an error message and exit. Test it on the following test cases:

|  |  |  |
| --- | --- | --- |
| **Test Case Scenario** | **Gender** | **Expected Output** |
| 1. Valid male | m | You entered a valid gender |
| 1. Valid female | F | You entered a valid gender |
| 1. Invalid character | x | X is an invalid gender. The program will end now. |

Do all the test cases work? \_\_\_\_\_ What happens? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

This is a case where de Morgan's theorem wasn't applied.

**de Morgan's Theorem** states that :

**!(condition1 && condition2)** *is equivalent to* **!condition1 || !condition2**

and

**!(condition1 || condition2)** *is equivalent to* **!condition1 && !condition2**

Since the condition for a valid **gender** would be:

**(gender == 'F' || gender == 'M')**,

the condition for an invalid gender is

**!(gender == 'F' || gender == 'M')**

which is equivalent to

**(gender != 'F' && gender != 'M')**

## Modify the **if** statement to work properly and retest the program.

## Add a statement to read an integer **age**. If the age is not between 18 and 65, print a message stating that the person gets a discount. Do not use an **!** in your condition. Test your program on the following test cases:

|  |  |  |
| --- | --- | --- |
| **Test Case Scenario** | **Age** | **Expected Output** |
| 1. Lower boundary - 18 years old | 18 | nothing |
| 1. Upper boundary -65 years old | 65 | nothing |
| 1. Between 18 and 65 | 22 | nothing |
| 1. Younger than 18 | 17 | You get a discount. |
| 1. Older than 65 | 72 | You get a discount. |

# Using the Eclipse Debugger

**Purpose:** To use the Eclipse debugger to set a breakpoint and step through a program.

To Do:

## Open the **Calculator** class in the **usingDebugger** package.

## Open **CalculatorSwitch**. Double-click in the margin beside the line:

**System.out.println("\nThe result is "**

**+ calc.doArithmetic(fnum, snum, opselect));**

A small blue button should appear in the margin. This is called a ***breakpoint***.

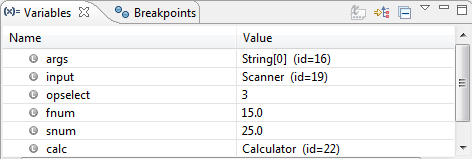
## Right-click on **CalculatorSwitch.java** and select “**Debug as**” and select **Java Application**.

## Enter any two numbersfor the two numbers and choose to do division. A pop-up window will appear with the following:

## Click **Yes**.

## The **Debug** perspective will open showing the program stopped at the line you highlighted earlier. The **Debug** perspective shows 4 panes – the workspace and outline panes as in the Java perspective and on the top, 2 new panes:

**The Variables Pane**

Select the **Variables** tab in the upper right hand pane. You should see the local variables for the **main()** method and their current values:

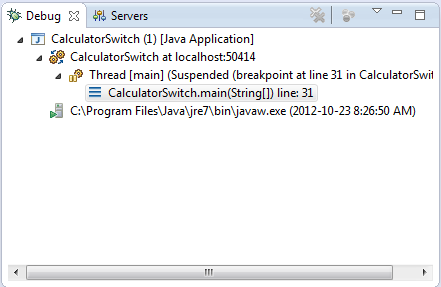
***Question:***

What are the current values of the following?

**fnum** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **snum**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

and **opselect** \_\_\_\_\_\_\_\_\_\_\_\_\_

**The DebugPane**

The **Debug** pane in the upper left-most window shows the currently active methods with the current method highlighted.

## The debug toolbar is along the top of the **Debug** window:



*terminate*

*step into*

*resume*

*step over*

*step return of*

## Use the **Step Into** icon to step into the **doArithmetic()** method of the **Calculator** class. You will have to click it twice.

Notice that the cursor moves to the first executable statement in the **doArithmetic()** method (**switch (code)** ).

Look in the **Variables** window. What are the current values of the following?

**num1** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **num2**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**code** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Where did these values come from? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Step Into** again. What line is the next line to be executed?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Why? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Step Into** again. It has now executed the line that was previously highlighted.

Which variable has changed now? \_\_\_\_\_\_\_\_\_\_\_\_\_

What is the value? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Where did this value come from? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Run the program several times using the Debugger. Experiment with using the different step tools (**Step Into**, **Step Over** and **Step Return**) to see how they work. Try setting other breakpoints and use the **Resume** button.

## To return to the normal view, click **Java** in the upper left-hand corner of the background tab.

# Unit Testing

**Purpose:** Use the debugger to detect and fix run-time errors in a class method.

**To Do:**

## Open **GradeCalculator.java**. This class is used to convert a grade to a percentage and to a letter grade. The class diagram is shown here:



## The **TestGradeCalculator** class has been written to test each of the **GradeCalculator** methods involving calculation: the **isValid()**, **convertMark()** and **convertGradeToMark()** methods. We will test and then debug them one at a time starting with the **isValid()** method.

The following test cases were designed to test the **isValid()** method:

isValid() Test Plan

| **Test Case** | **Before Execution** | **After Execution** | |
| --- | --- | --- | --- |
| **Initial Object State** | **Final**  **Object**  **State** | **Returned**  **Value** |
| 1. Valid score between 0 and perfect | perfect = 20  score = 15 | perfect = 20  score = 15 | true |
| 2. Valid score = 0 | perfect = 20  score = 0 | perfect = 20  score = 0 | true |
| 3. Valid score = perfect | perfect = 20  score = 20 | perfect = 20  score = 20 | true |
| 4. Invalid score < 0 | perfect = 20  score = -10 | perfect = 20  score = -10 | false |
| 5. Invalid score > perfect | perfect = 20  score = 25 | perfect = 20  score = 25 | false |

Run **TestGradeCalculator** with option **1** torun the above tests.

Which test cases failed? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Set a breakpoint on the **if** statement for the first failed test case in the **testisValid()** method in the **TestGradeCalculator** class.

## Debug **TestGradeCalculator.java**. When the debugger stops, step into the **GradeCalculator** class. Expand this in the Variables pane to see the values of **perfect** and **score**. Make sure that they were set correctly. Step through the method until you encounter the error.

## When you find an error, stop the debugger, correct the mistake and try running the test cases again.

## To clear a breakpoint you don't need any more, double-click the breakpoint circle in the margin.

## When all 5 test cases are working, test and correct **convertMark()** and **convertToGrade()**. The test cases used for these methods are shown below. Do **not** change the data types of any of the variables.

convertMark() Test Plan

| **Test Case** | **Before Execution** | **After Execution** | |
| --- | --- | --- | --- |
| **Initial Object State** | **Final**  **Object**  **State** | **Returned**  **Value** |
| 1. score == perfect | perfect = 20  score = 20 | perfect = 20  score = 20 | 100 |
| 2. score == 0 | perfect = 20  score = 0 | perfect = 20  score = 0 | 0 |
| 3. 0 < score < perfect | perfect = 20  score = 15 | perfect = 20  score = 15 | 75 |

convertToGrade() Test Plan

| **Test Case** | **Before Execution** | **After Execution** | |
| --- | --- | --- | --- |
| **Initial Object State** | **Final Object**  **State** | **Returned**  **Value** |
| 1. Grade == A | perfect = 100  score = 100 | perfect = 100  score = 100 | 'A' |
| 2. Grade == B | perfect = 100  score = 85 | perfect = 100  score = 85 | 'B' |
| 3. Grade == C | perfect = 100  score = 75 | perfect = 100  score = 75 | 'C' |
| 4. Grade == D | perfect = 100  score = 65 | perfect = 100  score = 65 | 'D' |
| 5. Grade == F | perfect = 100  score = 55 | perfect = 100  score = 55 | 'F' |

## Modify the **main()** method of **TestGradeCalculator.java** to use a **switch** statement instead of a nested **if** statement to determine which methods to call. Test your change.

## Run **TestGradeCalculator** with option 4 to rerun all the tests. All the test cases should now pass.

## Open **MarkCalculator.java**. This program uses **GradeCalculator** to convert a grade to a percentage and optionally to a letter grade. Run it using the test cases below. Put an ✓ in the column of all the test cases that pass. If any test cases fail, use the debugger to correct the error.

**Test Plan for MarkCalculator.java**

| **Test Case** | **Input Data** | | | **Expected Output** | | **Checked** |
| --- | --- | --- | --- | --- | --- | --- |
|  | **score** | **perfect** | **Letter**  **grade** | **percent** | **grade** |  |
| 1. perfect score, no letter grade | 15 | 15 | n | 100 | - |  |
| 1. perfect score with letter grade (max score) | 15 | 15 | y | 100 | A |  |
| 1. zero score, no letter grade | 0 | 15 | n | 0 | - |  |
| 1. zero score with letter grade (min score) | 0 | 15 | y | 0 | F |  |
| 1. min score with letter grade of A | 18 | 20 | y | 90 | A |  |
| 1. min score with letter grade of B | 16 | 20 | y | 80 | B |  |
| 1. max score with letter grade of B | 178 | 200 | y | 89 | B |  |
| 1. min score with letter grade of C | 21 | 30 | y | 70 | C |  |
| 1. max score with letter grade of C | 158 | 200 | y | 79 | C |  |
| 1. min score with letter grade of D | 24 | 40 | y | 60 | D |  |
| 1. max score with letter grade of D | 138 | 200 | y | 69 | D |  |
| 1. max score with letter grade of F | 15 | 30 | y | 50 | F |  |
| 1. score less than 0 | -5 | 15 | n | Error message | |  |
| 1. score greater than perfect | 16 | 15 | n | Error message | |  |

# To Finish

## Do the **Moodle Lab 9 Review Exercise**.