**IT 1090C Computer Programming I  
IT 6090C Java Programming   
Prof. Tom Wulf**

# Lab 8 – Get Methodical 20 pts (3 pts extra or graduate credit)

# READ VERY CAREFULLY (this lab requires tedious detail!):

**For this lab you will create a library of static input methods called SafeInput.** You will use this library for the rest of the semester. The library will be in a file called **SafeInput.java**.

You include a copy of your library in your IntelliJ project folder in the default package along with the other file(s) in your task. You use the name of the file as the prefix for calling the methods. So SafeInput.METHODNAME is how you call the method from the file. Here is an example. If I want to get the name of the user. I can only insure that it is not a blank String as far as validating it. I’d use my **getNonZeroLenString(Scanner pipe, String prompt)** method for this.

So, I have a java main file called **GetUserName**. In that file I create my Scanner and name variable. I make sure there is a copy of SafeInput in the same folder.

**Here is the GetUserName file (I’ll attach a copy of this with the assignment files):**import java.util.Scanner;

public class GetUserName

{

public static void main(String[] args)

{

Scanner in = new Scanner(System.in);

String firstName = "";

String lastName = "";

firstName = SafeInput.getNonZeroLenString(in, "Enter your first name");

lastName = SafeInput.getNonZeroLenString(in, "Enter your last name");

System.out.println("\nYour full name is: " + firstName + " " + lastName);  
 }

}

Create an IntelliJ project called: **Lab\_08\_GetMethodical**

Create a **java class file** (not a java main file) called **SafeInput.java**

For each of the following method descriptions, develop the indicated method and place it in the SafeInput file. **Note that the methods must be declared exactly as described, don’t change the name, order or type of the operands.**  The methods must be static as shown.  
  
Please note that the methods are very consistent. **In every case, the method should block illegal or incorrect input and continue to loop until the user gets it right.**

Then implement the indicated test programs using the filenames specified. All java main classes will be within the **Lab\_08\_GetMethodical** project. Create a single generic main method for scratch testing as you build the library. Call this **DevTest.java** For each method complete a series of sample test runs that reasonably tests it here in DevTest.. **Include copies of each output run for each one to support that you did this.**

After building the library, complete the additional programs. In this case, leave them all within the single IntelliJ project. Likewise, if you are using GitHub create a single a repo for this entire lab.

(Note details of the programs come after the description of the methods.)

# Part A: getNonZeroLenString

**public static String getNonZeroLenString(Scanner pipe, String prompt)**

* Pipe is a Scanner object that you created in main i.e. *in, console*
* Prompt is the message to display as the prompt for the input

We use this method when we don’t know what form the user’s response will be, but know that it must not be blank. For instance, asking for the user’s name. There is no reasonable way to verify a human name but we do know that it should not be blank.  
  
Here is the code for this first method. Again, place it in the SafeInput.java file.  
  
/\*\*

\*

\* @param pipe a Scanner opened to read from System.in

\* @param prompt prompt for the user

\* @return a String response that is not zero length

\*/

public static String getNonZeroLenString(Scanner pipe, String prompt)

{

String retString = ""; // Set this to zero length. Loop runs until it isn’t

do

{

System.out.print(“\n” +prompt + ": "); // show prompt add space

retString = pipe.nextLine();

}while(retString.length() == 0);

return retString;

}

# Part B: getInt

Create a method called getInt that prompts the user to input any integer. (That is that we don’t have any pre conception of what the range for the integer might be.) Make sure that you bullet proof the input by using the hasNext methods, reading the trash, and clearing the pipe after reading the value (the newline fix).  
  
**public static int getInt(Scanner pipe, String prompt)**

* Pipe is a Scanner object that you created in main in the usual way i.e. *in or console*
* Prompt is the message to display as the prompt for the input.

# Part C: getDouble

Create a method called getDouble that prompts the user to input any double value. Make sure that you bullet proof the input by using the hasNext methods, reading the trash, and clearing the pipe after reading the value (the newline fix).  
  
**public static double getDouble(Scanner pipe, String prompt)**

* Pipe is a Scanner object that you created in main in the usual way i.e. *in or console*
* Prompt is the message to display as the prompt for the input.

# Part D: getRangedInt

Create a method called getRangedInt that prompts the user to input an integer within a specified inclusive range. (inclusive means that low and high are valid inputs) Make sure that you bullet proof the input by using the hasNext methods, reading the trash, and clearing the pipe after reading the value (the newline fix).  
  
**public static int getRangedInt(Scanner pipe, String prompt, int low, int high)**

* Pipe is a Scanner object that you created in main in the usual way i.e. *in or console*
* Prompt is the message to display as the prompt for the input.   
  This should not include the [lo – hi] display. Code your method to use the prompt supplied by the user and append the [lo – hi] to it.
* Low is the low value for the input range
* High is the high value for the input range

# Part E: getRangedDouble

Similarly do one to input double values within a range:

**public static double getRangedDouble(Scanner pipe, String prompt, double low, double high)**

* Pipe is a Scanner object that you created in main i.e. *in*
* Prompt is the message to display as the prompt for the input. Again your method should build and append the range [lo – hi] to the prompt supplied by the user.
* Low is the low value for the input range
* High is the high value for the input range

# Part F: getYNConfirm

This is an input method that gets a Yes or No [Y/N] returning true for yes and false for no. It should accept yYnN as valid responses and loop until it gets one of them. Read that carefully: it returns true or false not “Y” of “N”!

**public static boolean getYNConfirm(Scanner pipe, String prompt)**

* Pipe is a Scanner object that you created in main i.e. *in, console*
* Prompt is the message to display as the prompt for the input

# Part G: getRegExString

Create a method called getRegExString that prompts the user to input a String that matches a RegEx pattern. Make sure that you bullet proof the input by using the hasNext methods, reading the trash, and clearing the pipe after reading the value (the newline fix).  
  
**public static String getRegExString(Scanner pipe, String prompt, String regEx)**

* Pipe is a Scanner object that you created in main in the usual way i.e. *in or console*
* Prompt is the message to display as the prompt for the input.
* RegEx is the regEx pattern in java String format to use for matching

# Program 01 (2 pts):

**Get User Name** (GetUserName.java)  
  
I’ve given you the code for this. It is above and you should have a file asset along with the assignment directions. Place a copy of the file in the default package. Type in the given code for the getZeroLengthString into your SafeInput.java file. Test and make sure that the program runs.  
  
Paste your output results here:

Graphical user interface, text

Description automatically generated

Program 02 (3 pts):

**Favorite Numbers** (FavNumbers.java)  
  
Prompt the user to enter their favorite integer and then their favorite double. Use the unconstrained getInt and getDouble methods for this.  
  
Really you only have two tests here, non-numeric input and valid numeric input for each method.  
  
Paste your output results here:

**Graphical user interface, text

Description automatically generated**

# Program 03 (4 pts):

**Date and Time of Birth** (BirthDateTime.java)  
  
Use the getRangedInt method to input the year (1950-2010), month (1-12), Day\*, hours (1 – 24), Minutes (1-59) of a person’s birth.  
  
Note: use a switch() conditional selector structure to limit the user to the correct number of days for the month they were born in. For instance if they were born in Feb [1-29], Oct [1-31]. HINT: there are only a few groups here not 12 different ones!

Paste your program output results here:

Text

Description automatically generated

# Program 04 (3 pts):

**Check Out at the 10$ Store** (CheckOut.java)

At the 10$ store nothing is more than $10.00. Prompt the user for the price of their item (.50 cents to $9.99 dollars) using the **getRangedDouble** method and continue to input items as long as they indicate that they have more using your **getYNConfirm** method. Display the total cost of the item(s) to 2 decimal places with printf.

Paste your output results here:

Text

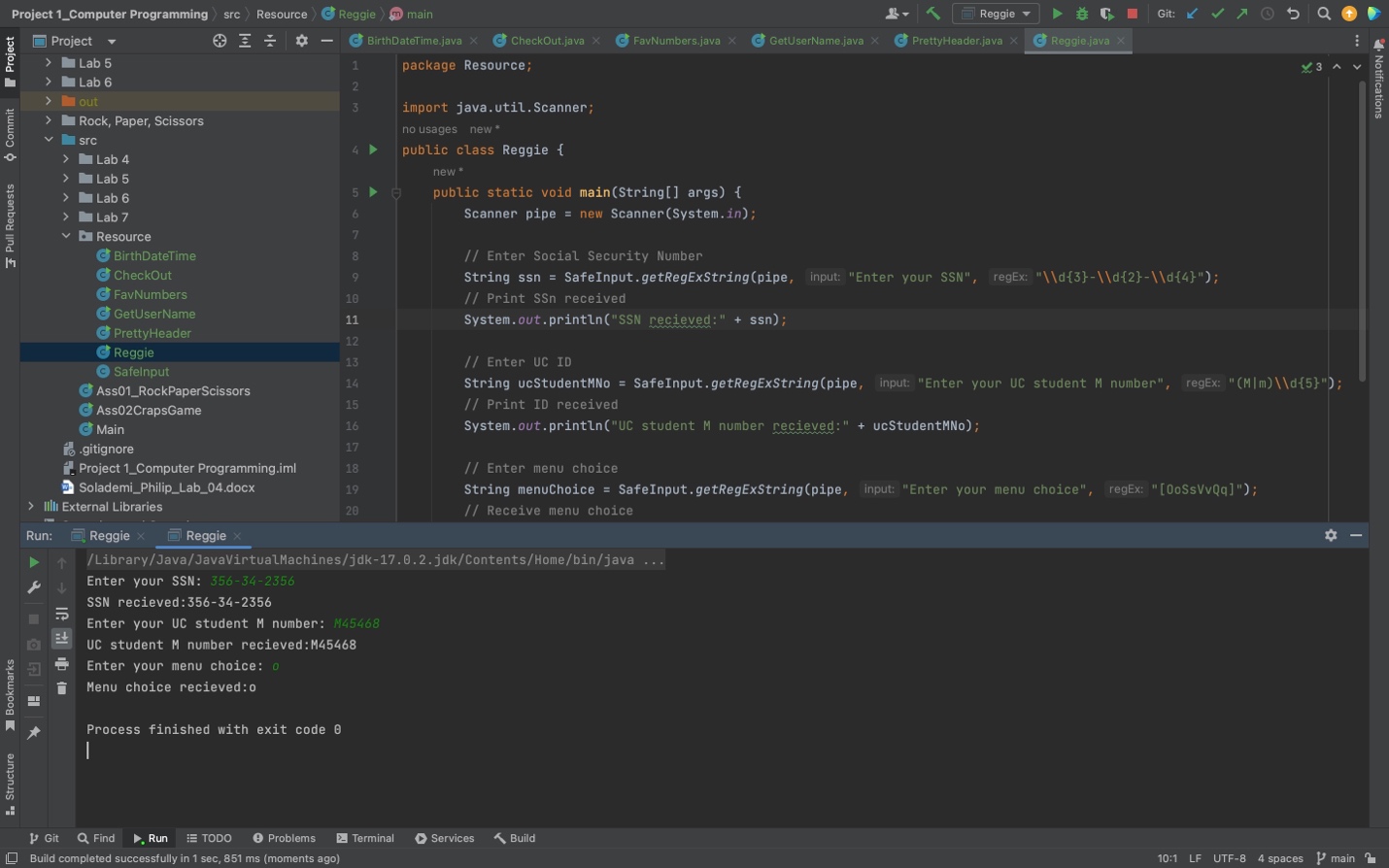
Description automatically generated

# Program 05 (4 pts):

**RegEx is Magic** (Reggie.java)

Use your getRegExString method several times.  
- get a SSN from the user using this pattern: \\d{3}-\\d{2}-\\d{4}  
- get a UC Student M number using this pattern (M|m)\\d{5}

-get a menu choice using this pattern [OoSsVvQq]   
these letters are mnemonics for the menu choices Open Save View Quit   
  
Do several test runs with strings that match and fail each pattern.  
  
  
Paste your output results here:



# Part H and Program 06 (4 pts):

**Pretty Header (**PrettyHeader.java)

This program does not use any of the previous methods but requires you to write an additional one.

Create a method (in SafeInput) that creates a Pretty header like this:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* Message Centered Here \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public static void prettyHeader(String msg)**

The output is always 60 characters wide for each line. Use loops to print out the lines. Long output statements of stars are not allowed! (Use loops instead.)

**Center** the msg on the second line with 3 stars on either end. HINT: use msg.length() to determine the length in characters of the msg and then use this info to calculate how to center it within the 60 character wide header.

Paste your output results here:

Graphical user interface, text

Description automatically generated

# Program 07 (3 pts Extra Credit or Required Graduate Option):

**C to F Table Display** (CtoFTableDisplay.java)**:**

C to F Data Display:

* Create a static method that takes a Celsius value (Java double) and returns the equivalent Fahrenheit value as a double.  
    
  **public static double CtoF(double Celsius)**  
  This is a one off method so place it in the same file as your program code. Note that you place it outside of and after main within the class body.
* Now code a program that creates a tabular display of two columns with the Celsius value on the left and the equivalent Fahrenheit value on the right. The table should show degrees centigrade in 1 degree increments from -100 to 100.

Screen shots here of your program output:  
  
Submitting your work:

Rename this docx file **FirstName\_LastName\_Lab\_08.docx.** Submit it and either your complete zip archive of your IntelliJ project or your GitHub repo link in canvas.