Lab 8: Get Methodical

IT 1090C Computer Programming I

IT 6090C Java Programming

Prof. Tom Wulf

Fall 2024 v1.8

# Lab 8: Get Methodical

20 points

Lab 8 lab requires tedious details; read carefully. You will create a library of static input methods called **SafeInput**. You will create a "**SafeInput.java**" file to use for the rest of the semester. YOU MUST CREATE THE LIBRARY WITH THE EXACT NAMES AND PARAMETERS AS DIRECTED.

You will include a copy of your library in your IntelliJ project folder in the src folder along with the other files for your programs. 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.

## Example

If I want to get the name of the user. I can only ensure 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 will create my Scanner and name variable and make sure there is a copy of SafeInput in the same folder.

### GetUserName

Here is the **GetUserName,java** 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);

}

}

# Lab

Create an IntelliJ project called: Lab\_08\_GetMethodical

Create a java class file called **SafeInput.java** (not a java main file, it won’t have a main method. just all the static methods)   
Create a single generic main method for scratch testing as you build the library. Call this **DevTest.java**. You will use this as a scratch file to code each static method and test it before adding it to the SafeInput library.

For each of the following method descriptions complete a series of sample test runs that reasonably tests it here in DevTest and then copy it into the **SafeInput** file.   
  
Include copies of each output run to support that you did this.

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. Also, the methods are consistent with one another in terms of the order of the parameters. So, the Scanner pipe is always the first parameter, etc.

Once you have completed and tested all your SafeInput methods, code the rest of the programs using the filenames specified and use the SafeInput.java code for the inputs. All java main classes will be within the Lab\_08\_GetMethodical project.

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

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

Paste your dev code screen shots after each method.

Paste your program run screen shots after each of the programs.

## 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

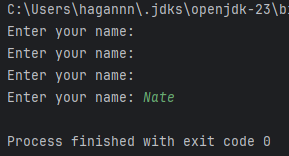
retString = pipe.nextLine();

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

return retString;

}

Paste a screen capture here of this methods test runs in DevTest.



## Part B: getInt

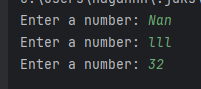
Create a method called getInt that prompts the user to input any integer. (That is that we don't have any pre conception of what the range for the integer might be.) Make sure that you bullet proof the input by using the hasNext methods, reading the trash, and clearing the pipe after reading the value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

Paste a screen capture here of this methods test runs in DevTest.



## Part C: getDouble

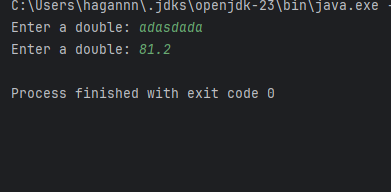
Create a method called getDouble that prompts the user to input any double value. Make sure that you bullet proof the input by using the hasNext methods, reading the trash, and clearing the pipe after reading the value (the newline fix).

**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.

Paste a screen capture here of this methods test runs in DevTest.



## 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

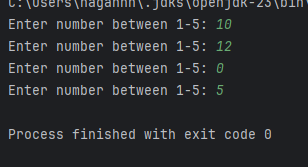
Prompt is the message to display as the prompt for the input.

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

Paste a screen capture here of this methods test runs in DevTest.



## 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)**

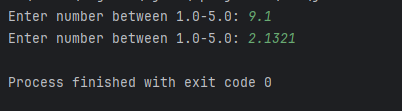
Pipe is a Scanner object that you created in main i.e. in

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

Paste a screen capture here of this methods test runs in DevTest.



## Part F: getYNConfirm

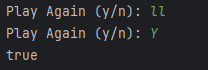
This is an input method that gets a Yes or No [Y/N] returning true for yes and false for no. It should accept yYnN as valid responses and loop until it gets one of them. Read that carefully: it returns true or false not "Y" of "N"!

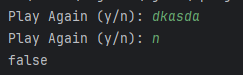
**public static boolean getYNConfirm(Scanner pipe, String prompt)**

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

Paste a screen capture here of this methods test runs in DevTest.





## Part G: getRegExString

Create a method called getRegExString that prompts the user to input a String that matches a RegEx pattern.

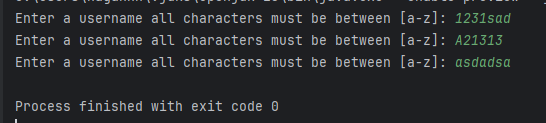
**public static String getRegExString(Scanner pipe, String prompt, String regEx)**

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

Paste a screen capture here of this methods test runs in DevTest.

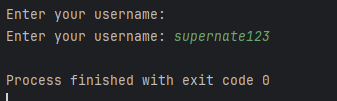


## Program 01 (2 pts):

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 if you didn’t already. Test and make sure that the program runs.

Paste a screenshot or output window copy here.



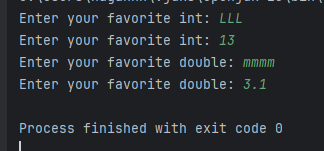
## Program 02 (2 pts):

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 a screenshot or output window copy here.



## Program 03 (2 pts):

Date and Time of Birth (**BirthDateTime.java**)

Use the getRangedInt method to input the year (1950-2015), 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 3 groups here, not 12 different ones!

Paste a screenshot or output window copy here.



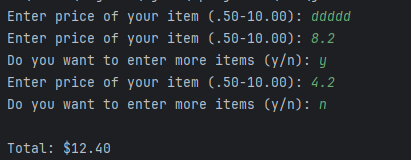
## Program 04 (3 pts):

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 $10.00 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 a screenshot or output window copy here.



## 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}$

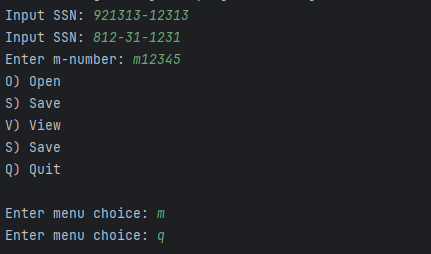
get a UC Student M number using this pattern ^(M|m)\\d{5}$

get a menu choice using this pattern ^[OoSsVvQq]$

these letters are mnemonics for the menu choices Open Save View Quit

Do several test runs FOR EACH PATTERN with strings that match and fail.

Paste a screenshot or output window copy here.



## Part H: Program 06 (3 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 \*\*\*

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

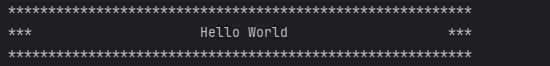
Header description: The top and bottom rows consist of a series of asterisks. A message is displayed on the second of three rows; the text reads, "message centered here." The message is centered withing the second row, but there are three asterisks at the start and end of the row. The three at the end of the row line up with the series of asterisks on the first and third lines.

**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 a screenshot or output window copy here.



## Part H: Program 07 (4pts)

C to F Table Display (CtoFTableDisplay.java):

C to F Data Display: Note this is similar to a program we wrote previously but now you use a method to compute the converted temperature instead of inlining the code in the loop.

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 after main at the bottom of the file within the class. 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. Create the table by using a for loop to drive a printf statement with the two temperatures.

Paste a screenshot or output window copy here.

# Post-Lab: Double-check Your Work

Now that you have completed the lab tasks, build constructive professional work habits by carefully rechecking your work, and abiding by the naming format and submission instructions.

## Rename the file:

Use the **YourLastname\_YourFirstname\_Lab08.docx** format; remember to replace "**YourLastname**" and "**YourFirstname**"withyour actual names.

The naming format will help us efficiently track and organize your files.

## GitHub Link

Paste the link to your GitHub repo here.

## Submit

Submit to Canvas using the assignments mechanism to upload your document. **Don’t submit a link to your document.**