Week 1, Lab B – Input Validation

# Lab Intro & Prep

In this lab, we will implement various input validation mechanisms to create more robust programs by validating the input received from a user. You should already have Eclipse open from the first part of the lab session.

## Learning Objectives

* Develop robust applications featuring input validation
* Implement static (class) methods

# Exercise 1 – User Input (with Sentinel Value to terminate entry)

In Exercise 3 of Lab A, we allowed the user to enter 5 integer numbers and calculated and displayed the average and total of those numbers. We will now amend this code to allow *any* number of inputs. The user can enter a sentinel value (e.g. -99) to signal they have finished inputting values.

You may choose to create a new class or modify the existing class you created earlier.

The output (Fig 1) should display **how many numbers were entered by the user**, along with the **total** and **average** of those numbers. **Remember**: the sentinel value should not be considered part of the input (i.e. -99 should not be included in any of the calculations).

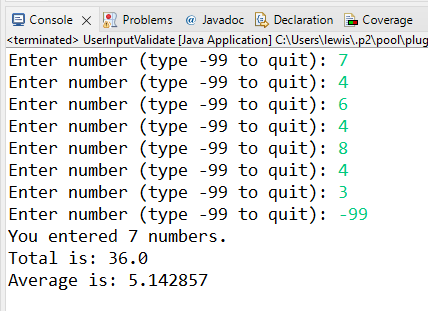


Fig - Exercise 1 Example Input

# Exercise 2 – Validation via hasNext Methods

Create a class called **TaxCalculator** with a main method.

Develop an application that reads (from the console) a person's salary (as in integer) and calculates their earnings after tax (Fig 2). Assume that everyone is taxed 25% of their income. You should make use of the Scanner class' **hasNextInt** method to make sure that the input being supplied is an integer. Print out this salary to the console, in addition to calculating and printing:

* The amount which has been deducted (how much they have been taxed) as a floating-point value
* The salary after deductions have been made (post-tax earnings) as a floating-point value

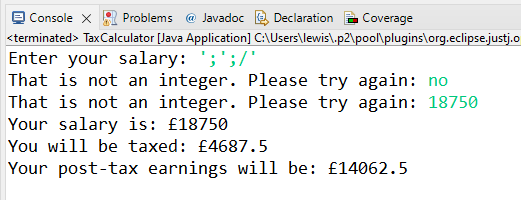


Fig - TaxCalculator Example Output

# Exercise 3 – ValidateInt Function (while loop & hasNext)

We are now going to combine the use of a while loop with the hasNext methods of the Scanner class to create a **static** function that will check to see if the input received is a valid **positive** number.

Recall that a static method is a method that exists in the class and does not require an object of the class in order to call it. For example, when we first saw classes and objects in Programming 1 (Motorbike exercises), we created a Motorbike class that contained (non-static) methods e.g. **render** and **move**. We then created Motorbike objects and called the move and render methods **on** those objects. Static methods, on the other hand, do not operate on objects.

Create a class named **ValidateIntDemo** that will allow the user to enter **two validated positive integers** and output the sum, the product and the difference (example output can be seen in Fig 3). Starter code is provided in Listing 1.

You should use functions or procedures with parameters to :

* Validate the input (1 parameter – scanner object, return the result)
* Calculate the sum (2 parameters, return the result)
* Calculate the product (2 parameters, return the result)
* Calculate the quotient (2 parameters, return the result)
* Calculate the difference (2 parameters, return the result)

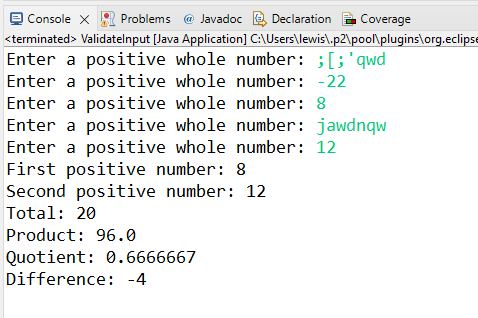


Fig – Exercise 3 Example Output

Listing 1 - ValidateIntDemo Starter Code

|  |
| --- |
| **import** java.util.Scanner;  **public class** ValidateIntDemo {  **public static void** main(String[] args) {  Scanner input = **new** Scanner(System.*in*);    **int** num1 = *validateInt*(input);  **int** num2 = *validateInt*(input);    input.close();  }  **static** **int** validateInt(Scanner input) {    // set to zero or negative to ensure loop runs (while num not positive)  // (local variables not initialised by default)  **int** num = 0;    // TODO: input validation with while loop and hasNextInt  **return** num;  }  } |