Homework 3: Input Validation

1 Objectives

* control structures
* console-based user input using Scanner class
* writing complete programs using two classes: client and supplier

2 User Interface Specification

This is a console-based I/O program. Display should go to System.out (print or println) and the program will get user input using the Scanner class. The flow of execution should be as follows:

* When the program starts, display a one-line introduction to the user
* Display a menu with 5 options

1. validate zip code

2. validate SSN

3. validate password

4. instructions

0. quit

* Prompt the user to input an option
* Prompt the user for the required input and then display the results
* Repeat the main menu until the user enters the option to quit
* If the user enters an invalid menu option, print a helpful error message and re-prompt. Note, this could be either a wrong value or wrong data type.

3 Code Specifications

For this assignment, you will write *two* different classes. The goal is to separate the user interface code from the code that performs validation. The sample programs [**MyMath.java**](http://webshares.northseattle.edu/cscweb/CSC142/examples/clientSupplier/MyMath.java)and [**TestMyMath.java**](http://webshares.northseattle.edu/cscweb/CSC142/examples/clientSupplier/TestMyMath.java)are an essential reference.

In a supplier class named **Validations**, implement 3 static methods to perform these different validation tasks:

* validate that a String represents a valid zip code. A valid zip code is made up of 5 digits.
* validate that a String represents a 9-digit SSN. A valid SSN follows the pattern XXX-XX-XXXX, including the dashes.
* validate that a String represents 8 symbols, including letters (upper and lower case), numbers, and special symbols like #$&\_.

Though I specify 3 public methods, you may have more if you choose to do some decomposition. Note that this Validations class does not need a 'main' method. However, you may write one for testing purposes if you wish. *You must use parameter lists and return statements to move data in and out of these methods.***No method in this class should use System.out.println or the Scanner class (except an optional test method).** In other words, no method here should interact with the user; that's the job of the other class.

In a class named **ValidateApp**, create a console-based, menu-driven user interface that behaves like the one described above. This class must use a 'main' method to start things off. However, you are encouraged to decompose your solution to organize your code.

No class variables may be used; class *constants*are ok.

Your program should not throw any exceptions to the user. If a user enters an invalid menu option, the program should display a helpful error message and ask for new input.

You must implement your solution using loops, if statements, and switch-case.

4 Testing & Documentation

* As always, I recommend doing this in pieces. For example, you might choose to do the validation code first (testing as you go). Then focus on the user interface.
* Your documentation in both classes is an integral part to your solution since you are the designer/author of these classes.
  + Each class should have an overall description comment at the top
  + Be sure to include a java-doc for each method that describes what it does, describes the parameters and the return.
  + Make sure to include internal comments to guide the reader through your algorithms.

5 File Submission

There are two different .java files for this assignment: **Validations.java** and **ValidateApp.java**. You will need to compress them into one .zip file, or use BlueJ to make a .jar file.

6 Grading

|  |  |
| --- | --- |
| **Area** | **Pct** |
| Design, correctness, implementation | 90% |
| Documentation and style | 10% |
| **Total** | **100%** |

|  |  |
| --- | --- |
| **Achievement** | **Max Points** |
| Compile/runtime errors | 50% |
| Basic objects | 100% |