420-WA5-AB PROGRAMMING I

Final Project: Validator

Mankai, Mohamed Amine | Setrakova, Iana

March 26, 2024

**Table** **of** **contents**

[Project overview 2](#_Toc162296432)

[Planning and development of the code 2](#_Toc162296433)

[Basic descriptive process of each method 2](#_Toc162296434)

[Complications that arose during the coding process 6](#_Toc162296435)

[Individual team assignments 6](#_Toc162296436)

[Perspective about project comprehension, workload, and difficulty 7](#_Toc162296437)

# Project overview

The Programming 1 final project includes the creation of a validation program by developing Java methods. The goal of the project is to create a code that verifies various types of data, including email addresses, usernames, and passwords, while strictly adhering to predetermined criteria for naming conventions and code structure.

All methods must be in a class named Validator.

# Planning and development of the code

We initiated our project by dividing the first methods equally, with each team member tasked to code four methods. As we progressed, we followed this workflow:

- each one of us worked on all the remaining methods.

- after finishing each method, we shared our code, troubleshooted errors together, and optimized the code as much as possible.

- we also set up a GitHub repository to share code and to be able to work efficiently.

# Basic descriptive process of each method

* **isAlphaNum()**

This method checks if a character is alphanumeric. We used the method isLetterOrDigit which determines if the input character is a letter of the English alphabet (case insensitive) or a number between 0 and 9.

* **isSpecialChar()**

This method checks if a character is an acceptable special character. The method takes a char, and boolean as its arguments.

The method always returns true if the char provided is a dash (-), or a period (.). If the boolean is true, the method also returns true if the char provided is an underscore(\_).

* **isPrefixChar()**

This method checks if a character is a character allowed in the prefix. A valid prefix can contain only alphanumeric characters, dashes, periods, or underscores.

We used the methods isAlphaNum(), and isSpecialChar() with underscore(\_) allowed.

* **isDomainChar()**

This method checks if a character is a character allowed in the prefix. A valid domain can contain only alphanumeric characters, dashes, or periods.

We used the methods isAlphaNum(), and isSpecialChar() with underscore(\_) not allowed.

* **singleAtSign()**

This method checks if a String contains a single at sign (@).

The String characters are looped through and if there is ‘@’ we count its appearance. If the number of appearances equals 1, the method returns true. Otherwise, it returns false.

A diagram of a computer program

Description automatically generated

* **fetchBeforeAt()**

This method gets the first part of an email address. We assume that the String argument contains only one @ symbol.

The email String characters are looped through and as soon as the character ‘@’ appears, we use break to escape the loop. The method returns a String containing the portion before @.

* **fetchAfterAt()**

This method gets the second part of an email address. We assume that the String argument contains only one @ symbol.

The email String characters are looped through and as soon as the character ‘@’ appears, we use break to escape the loop. The method returns a String containing the portion after @.

* **isPrefix()**

The method checks if the start of a String is a valid email prefix.

Since: the String must contain only alphanumeric characters, underscores, periods, and dashes **and** an underscore, period, or dash must always be followed by at least one alphanumeric character; we can conclude that the last character must be alphanumeric.

So, at first we verify the following conditions :

* The String contains at least one character
* The first and last characters are alphanumeric

If the first conditions are met, the method checks all the characters one by one, checking if:

* They are allowed characters, using isPrefixChar()
* The special characters are always followed by at least one alphanumeric character

* **isDomain()**

The method checks if the end of a String is a valid email domain.

First, we determine the index of the last dot.

Second, if such an index exists, we check if the first letter of String is alphanumeric, and the last character is a letter (since the second portion contains only letters). Then we split the string into two parts separated by the last dot.

Then we do a quick check for two conditions before we move to verifications using loops.

First condition: The first portion must end with an alphanumeric character\*.

(\*Since the first portion contains only alphanumeric characters, periods, and dashes and a period or dash must also be followed by at least one alphanumeric character; we can conclude that the first portion must end with an alphanumeric character. This will also help us to avoid dealing with the edge case later when we loop through the first portion.)

Second condition: The second portion must contain at least two characters.

Once these conditions are met, we move to the other verifications using loops.

The first for loop allows us to check if the second portion contains only letters.

The second loop allows us to check that all the characters are valid domain chars, and that a period or dash is always followed by at least one alphanumeric character.

* **isEmail()**

The method checks if a String is a valid email address.

We verify if String contains a single at sign(@).

Then we use the methods fetchBeforeAt() and fetchAfterAt() to get the beginning (prefix) and the ending (domain) of String.

Then we use isPrefix() and isDomain() to check if the prefix and the domain are valid.

If everything is correct, the method returns true, otherwise it returns false.

* isUsername()

The method checks if a String is a valid username.

First, the method verifies if String contains seven or less characters, starts with a period, or dash, and its last character is alphanumeric\*.

(\* Since String contains only alphanumeric characters, periods, or dashes and a period, or dash must be followed by at least one alphanumeric character; we can conclude that the last character is alphanumeric. This also covers the condition “Contains at least one alphanumeric character”.)

If these conditions are met, we check if all the characters are valid domain chars, and that a period or dash must always be followed by at least one alphanumeric character.

If everything is correct, the method returns lowercase String, otherwise it returns an empty String.

* **safePassword()**

The method checks if a string is considered a safe password.

We verify that String contains minimum 7 characters and maximum 15 characters.

Then we loop through String. We check that the same character is never repeated consecutively, and calculate the number of alphanumeric characters, lower case letters, upper case letters, numbers, and special characters.

Finally, we check that String contains at least one from each category.

If everything is correct, the method returns true, otherwise it returns false.

* **validEmails()**

The method checks which Strings contained in an array are valid emails, and return them in a new array.

At first, we loop through the array, count the valid emails, and replace each invalid one by an empty String.

We create a new array which length is equal to the number of valid emails.

Finally, we loop again through the array provided and assign the valid emails to the new array, while ignoring the empty Strings.

* **validUsernames()**

The method checks which Strings contained in an array are valid usernames, and return them in a new array.

At first, we loop through the array, and count the valid usernames.

We create a new array which length is equal to the number of valid passwords.

Finally, we loop again through the array provided, and assign the valid usernames to the new array, while ignoring the empty Strings.

* **validPasswords()**

The method checks which Strings contained in an array are valid passwords, and return them in a new array.

At first, we loop through the array, count the valid passwords, and replace each invalid one by an empty String.

We create a new array which length is equal to the number of valid passwords.

Finally, we loop again through the array provided, and assign the valid passwords to the new array, while ignoring the empty Strings.

# Complications that arose during the coding process

We faced some challenges making sure our methods followed the rules. Maybe the most difficult one was isDomain(), and especially the case where the String includes more than a dot. It took us some time to figure it out, because we tried many options before finding the right one.

Another difficulty was to deal with edge cases, and we had to pay a particular attention to avoid errors like "IndexOutOfBoundsException”.

We also encountered some minor issues using GitHub, but we could resolve them quickly.

# Individual team assignments

r: review

|  |  |  |
| --- | --- | --- |
| **Methods** | **Amine** | **Iana** |
| isAlphaNum | **\*** | r |
| isSpecialChar | **\*** | r |
| isPrefixChar | **\*** | r |
| isDomainChar | **\*** | r |
| singleAtSign | r | **\*** |
| fetchBeforeAt | r | **\*** |
| fetchAfterAt | r | **\*** |
| isPrefix | r | **\*** |
| isDomain | **\*** | **\*** |
| isEmail | **\*** | **\*** |
| isUsername | **\*** | **\*** |
| safePassword | **\*** | **\*** |
| validEmails | **\*** | **\*** |
| validUsernames | **\*** | **\*** |
| validPasswords | **\*** | **\*** |

# Perspective about project comprehension, workload, and difficulty

We understood the project well because the guidelines were clear and easy to follow. We didn’t just divide the work in two, so the workload was demanding, but it was a great way to practice our coding skills.