8/3/2023

Eric Sadowski, XIAOXING PAN

Validation Project

Table of Contents

[Goal, Planning and Development 2](#_Toc129166573)

[Team Assignments & Workflow 2](#_Toc129166574)

[Algorithm Layout of the Master Methods 2](#_Toc129166575)

[*isEmail()* 2](#_Toc129166576)

[*isUsername()* 3](#_Toc129166577)

[*safePassWord()* 4](#_Toc129166578)

[Basic and Intermediate Methods 5](#_Toc129166579)

[Basic Methods 5](#_Toc129166580)

[isAlphaNum() 5](#_Toc129166581)

[isSpecialChar() 5](#_Toc129166582)

[singleAtSign() 6](#_Toc129166583)

[fetchBeforeAt() and fetchAfterAt() 6](#_Toc129166584)

[reSpecial() 6](#_Toc129166585)

[hasCaps() 7](#_Toc129166586)

[hasLow() 7](#_Toc129166587)

[hasNumber() 7](#_Toc129166588)

[reChar() 7](#_Toc129166589)

[Intermediate Methods 8](#_Toc129166590)

[isPrefixChar() 8](#_Toc129166591)

[isDomainChar() 9](#_Toc129166592)

[isPrefix() 10](#_Toc129166593)

[isDomain() 10](#_Toc129166594)

[Problems in Development 12](#_Toc129166595)

[Meeting Minutes 13](#_Toc129166596)

[Figure 1. Flowchart of isEmail() 3](#_Toc129167335)

[Figure 2. Flowchart of isUsername() 4](#_Toc129167336)

[Figure 3. Flowchart of isPrefixChar() 8](#_Toc129167337)

[Figure 4. Flowchart of isDomainChar() 9](#_Toc129167338)

[Figure 5: Flowchart of isDomain() 11](https://johnabbott-my.sharepoint.com/personal/2339295_johnabbottcollege_net/Documents/project%20for%20programming%20I/project%20programming%20i.docx#_Toc129167339)

# Goal, Planning and Development

The goal of the project is to create a series of methods in Java and then compound these basic methods to validate an email address, username and password. Making sure they meet specific requirements.

The planning stage of the project consisted of figuring out where to start, how to have a steady workflow, and at each step connecting methods to other methods. To find a starting point all we had to do was figure out which methods were independent and dependent. As well as determining which methods would be required to execute the “master methods” *isEmail()*, *isUsername()* and *isPassword()*.   
By separating the tasks, we were better able to see the ladder of development and the blocks of code required to build the towering master methods. This separation of tasks and “box in a box” mentality allowed us to establish a workflow where we could each build certain blocks and report back each day to build atop them. Thus, the task was completed.

# Team Assignments & Workflow

At the beginning, each of us was assigned a few basic methods to get the project started. Eric was responsible for *isAlphaNum()* and *fetchBefore()/fetchAfter().* And Xiaoxing took *isSpecialChar()* and *singleAtSign().*

And as the project progressed as such, we decided to split the workload in half at each rung on the development ladder. As the project continued, both of us utilized active daily communication and decided on which methods to take as we progressed. This created effective methods that used each person’s source code and allowed us to check each other's work as we progressed.

# Algorithm Layout of the Master Methods

The Algorithm is built of a series of conditional Booleans and 2 String fetch tasks. It can be viewed as a Pyramid, Tower, or a set of Russian Dolls. All the Intermediate Methods are dependent on the Basic Methods and all the Master Methods are dependent on both the Intermediate and Basic Methods. They all work together to achieve the goal of validating the input for each Master Method Algorithm. Certain steps in the algorithm are composed of several other smaller tasks taking place within a larger task.

## *isEmail()*

The largest Master Method *isEmail()* utilizes nearly every other method directly or indirectly in our source code. With the goal of validating an Email address. First, an email is inputted to the method. It then checks that it has a single “@” sign and there are no repeated special characters. Then, it splits the String in half from the “@” sign and writes each section to a new string (domain and prefix). Finally, the prefix and domain Strings are checked with the Intermediate Methods *isDomain()* and *isPrefix()* to determine if both ends are valid. If both are valid the return is true, otherwise false.

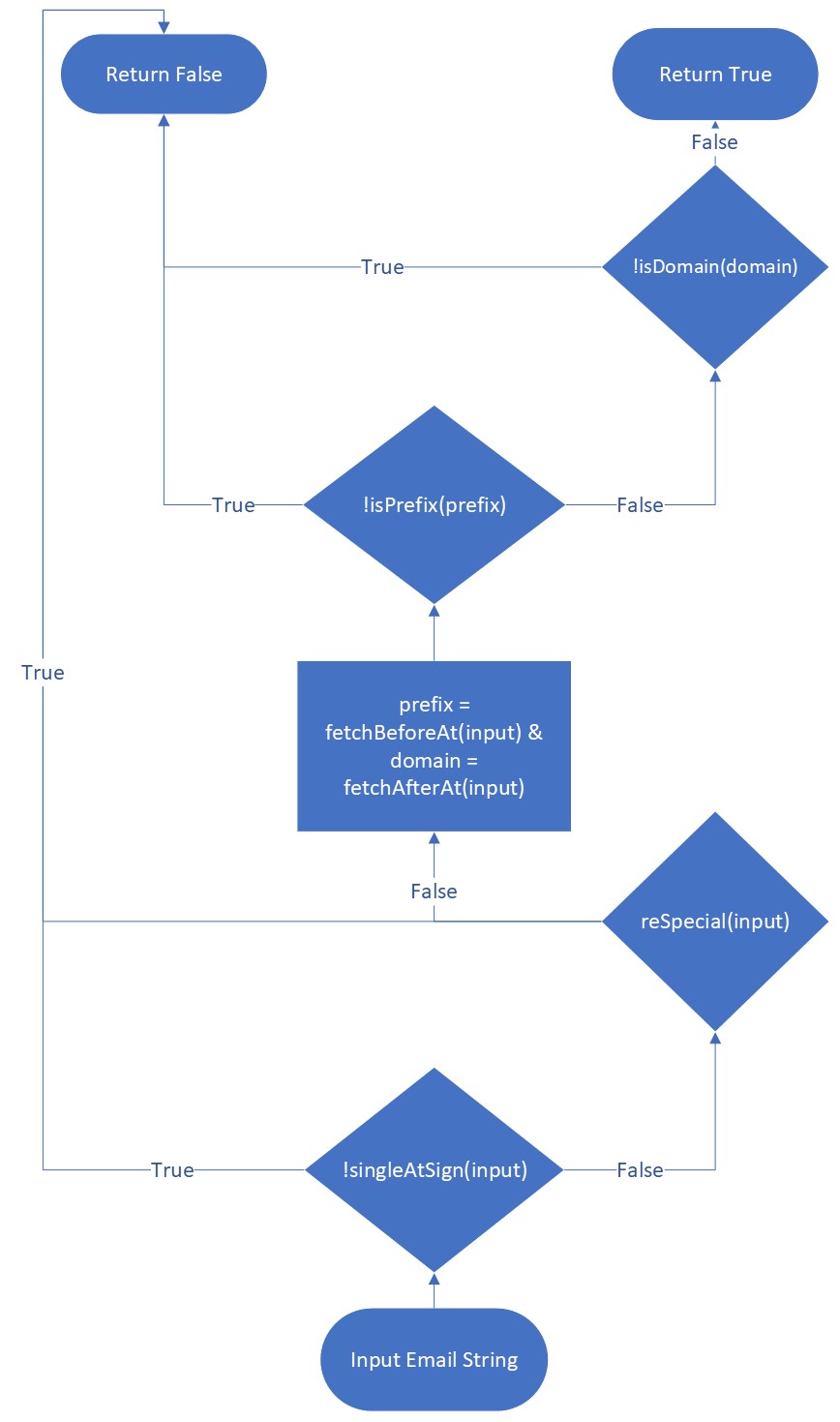


Figure . Flowchart of isEmail()

## *isUsername()*

The username validator *isUsername()* has a 2-part algorithm flow. It is a method that takes a String and returns a String that is either a Username or a blank. It is a unique master method in this way; in that it is not a Boolean. First, a string is put into the method then it is checked by a variety of conditions. These are: Is there anything in the input, is the input less than 1 character, is the input greater than 7 characters, are there any repeating special characters, and does it start with a “.” or “-”. The last 2 of these conditionals are both Basic Methods. If any of these are met or not met (case depending) the method will return a blank string (“”). After the first section of conditionals the string is put into a for loop that iterates through each character checking if it contains at least 1 alphanumeric character and if all the characters meet the conditions of *isDomainChar()* as well as if it is an exclamation mark (!). If any character is found that is not either one of these things the string will return blank “”. If all the above conditions are met the string will return the username input in lower-case.

Diagram

Description automatically generated

Figure . Flowchart of isUsername()

## *safePassWord()*

The password validator *safePassWord()* is the most dissimilar from the other Master Methods in the way that it uses several methods not used in any of the others. First, a password is inputted to the method as a string. Then, it is checked by a multitude of statements in an if conditional. These include: is there an input, does it have more than 7 characters, does it have more than 15 characters, does it have repeating characters, does it have any capital letters, does it have any lower-case letters and does it have a number. The last 4 of these methods are all basic conditions that have been designed specifically for this Master Method (*reChar()*, *hasCaps()*, *hasLow()*, *hasNumber(*). After going through a gauntlet of conditionals the input is then scanned through character by character looking for a special character. The Basic Method *isSpecialChar()* is used here in a for loop to iterate through the input String. If a special character is found in any one of these iterations the method returns true, and it is deemed a sufficient password. If it finds none it returns false, condemning it as an unsafe password.

# Basic and Intermediate Methods

In total, there are 17 Methods used in our project. They can be split into 3 Categories. These are: Basic Methods, Intermediate Methods, and the Master Methods. Above we covered the algorithmic structure of the 3 Master algorithms. In this section we will cover the 14 Basic and Intermediate Methods that make up the larger 3.

## Basic Methods

The basic building blocks of this project are these methods. Most are Booleans that return a true or false value. The inputs of these methods are both char and string depending on the task asked of it. These methods allow every other method to work in our code.

### *isAlphaNum()*

This method checks if a character is alphanumeric. There are at least two ways to achieve this. One way is to compare the input with the characters in Unicode, the other is to call another 2 of java’s built-in methods, *Character.isLetter()* and *Character.isDigit()*.

Table .The logic of isAlphaNum()

|  |  |  |
| --- | --- | --- |
| Input Argument | Conditions that Return True | Condition Type  (|| or &&) |
| * char | is an uppercase letter | || |
| is a lowercase letter | || |
| is a number | || |

### *isSpecialChar()*

This method checks for an acceptable special character. Only the dash and period are accepted, and the underscore is conditionally accepted.

Table . The logic of isSpecialChar()

|  |  |  |
| --- | --- | --- |
| Input Argument | Conditions that Return True | Condition Type  (|| or &&) |
| * char * Boolean – if underscore is allowed | is a dash | || |
| is a period | || |
| is an underscore && it’s allowed | || |

### *singleAtSign()*

This method checks if a String contains a single at sign (@). Our strategy is to check each character of the input string, count the numbers of @ and return true value only if the count is 1.

Table . The logic of singleAtSign()

|  |  |  |
| --- | --- | --- |
| Input Argument | Conditions that Return True | Condition Type  (|| or &&) |
| * string | has a ‘@’ | && |
| ‘@’ shows up only 1 time | || |

### *fetchBeforeAt()* and *fetchAfterAt()*

These 2 methods are aimed at getting the beginning/ending of an email address. Our strategy is to use the ‘@’ as a delimiter to split the input string into 2 parts, get an array of a string with these 2 pieces and then return the first element as the beginning and the second element as the ending.

### *reSpecial()*

This method checks for repeat dashes, underscores and periods that are allowed in other statements. We use a comparison operator (||) for the 9 different possible cases and return true if those cases are detected. It’s created for ease of reading and utilization of D.R.Y. coding.

Table . The logic of reSpecial()

|  |  |  |
| --- | --- | --- |
| Input Argument | Conditions that Return True | Condition Type  (|| or &&) |
| * string | has ‘\_-’ or has ‘\_\_’ or has ‘\_.’ | || |
| has ‘.-’ or has ‘.\_’ or has ‘..’ | || |
| has ‘--’ or has ‘-\_’ or has ‘-.’ | || |

### *hasCaps()*

This method (used only in *safePassword()*) checks if a string has any capital letters. It does this by changing the string to lower case then saving it to a variable and comparing it to the input string. If they are both the same, it means there are no capital letters, and it returns false. Otherwise, it returns true.

### *hasLow()*

This method (used only in *safePassword()*) checks if a string has any lower-case letters. It does this by changing the string to upper case then saving it to a variable and comparing it to the input string. If they are both the same there are no lower-case letters it returns false. Otherwise, it returns true.

### *hasNumber()*

This method (used only in *safePassword()*) checks if a string has any numbers. It does this by changing the string to a char array and iterating through each character, checking with *isDigit().* If one of the array items is a number, it returns true. Otherwise, it returns false.

### *reChar()*

This method (used only in *safePassword()*) checks if a string has any sequentially repeated characters. It does this by iterating through each character and comparing it to the last character. The for loop skips the first iteration with the continue keyword and then starts the main inner loop. This loop assigns a parameter to the previous character and checks it to the current character in the loop. If they match the method returns true, otherwise false.

## Intermediate Methods

Intermediate methods are like the bridge between the basic methods and the master methods. They are implemented by calling the basic methods. They are built to check a certain requirement of the master method and they make the master method more readable and easier to manage.

### *isPrefixChar()*

This method checks if a character is allowed in the prefix. We compare the input character with alphanumeric characters and accepted special characters (underscore is allowed here) by calling the *isAlphaNum()* and *isSpecialChar()* method, if it matches one of them, return true, false otherwise.

Diagram

Description automatically generated

Figure . Flowchart of isPrefixChar()

### *isDomainChar()*

This method checks if a character is allowed in the domain. We compare the character with alphanumeric characters and accepted special characters (underscore is not allowed here). If it matches one of them return true, false otherwise.

Diagram

Description automatically generated

Figure . Flowchart of isDomainChar()

### *isPrefix()*

This is a method that checks if the part before the @ of an input string is a valid email prefix. First the input is checked for a first and last alphanumeric number. After, the conditions that return a true value are quite complex and difficult to list. So, our strategy is to call the i*sAlphaNum()*, *reSpecial()* and *isPrefixChar()* methods, searching for the opposite condition. In these cases, if the opposite returns true, the return is false.

Table . The Logic of isPrefix()

|  |  |  |
| --- | --- | --- |
| Input Argument | Conditions that Return False | Condition Type  (|| or &&) |
| * string | first or last character is not alphanumeric | || |
| is null or length < 1 | || |
| has repeat special characters | || |
| has unallowed prefix characters | || |

### *isDomain()*

This is a method that checks if the part after the @ of an input string is a valid email domain. We use the same strategy as the *isPrefix()*. The steps are more complicated since we must check 2 parts of the input string, both the one before and after the last period. So, we start with finding the index of the last period in the input string and then use it as a delimiter to split the input string into 2 parts, firstPortion and secondPortion. Then we call the i*sAlphaNum()*, *reSpecial()* and *isDomainChar()* and *Character.isLetter()* methods, searching for the opposite. List all the conditions that give a false value, and then all the rest situations return true.

Table 6. The Logic of isDomain()

|  |  |  |  |
| --- | --- | --- | --- |
| Input Argument | Conditions that Return False | | Condition Type  (|| or &&) |
| * string | whole | no period | || |
| first portion | is null or length < 1 | || |
| has unallowed domain characters | || |
| has repeat special characters | || |
| last character is not alphanumeric | || |
| second portion | is null or length < 2 | || |
| has other characters except alphabet | || |

Diagram

Description automatically generated

Figure : Flowchart of isDomain()

# Problems in Development

* Forgot to check the last character in certain methods.
* Forgetting to use Intermediate methods to make other Intermediate Methods, going back to update.
* Sometimes there are lots of situations that return true, it’s hard to list them all completely, in this case, we can check for the opposite.
* Found necessary methods described in project insufficient to check all the conditions found therein the project. Decided to make custom methods.
* PDF clarified. Realized a method that checks for repeat characters was necessary. Created it and updated documentation to include it.

# Meeting Minutes

03/03

1. Self-introduction.
2. Go through the project and find out what should we do – 12 methods, documentation, etc;
3. Decide the basic strategy to fulfill the project – step by step up the stairs, start from the non-relative methods.
4. Create the git-hub repository and cooperate it.
5. Assign the tasks and get started – Eric gets *isAlphaNum()* and *fetchBeforeAt()* , xiaoxing gets *singleAtSign()* and *isSpecialChar().*
6. Fix the meeting time for tomorrow.

03/04

1. Reviewed methods finished
2. Looked at larger methods
3. Decided to finish tomorrow

03/05

1. Communicated through chat
2. Updated *isPrefix()* and *isDomain()* to contain Intermediate Methods
3. Finished *isEmail()* & *isUsername()*
4. Shared documentation folder

03/06

1. Accessed what was remaining to do in project.
2. Talked about test.
3. Finished *isPassword()*.

03/07

1. Worked on documentation together on OneDrive via Teams chat.
2. PDF clairified. Created a new method reChar and updated to account for that.