Planning and developing the validator project

Group members:

Shiyuan

Pejman

Contents

[Overview of the project: 2](#_Toc129158974)

[Describing the methods of the project: 3](#_Toc129158975)

[isAlphaNum(): 3](#_Toc129158976)

[isSpecialChar(): 3](#_Toc129158977)

[isPrefixChar(): 4](#_Toc129158978)

[isDomainChar(): 5](#_Toc129158979)

[singleAtSign(): 5](#_Toc129158980)

[fetchBeforeAt(): 6](#_Toc129158981)

[fetchAfterAt(): 8](#_Toc129158982)

[isPrefix(): 8](#_Toc129158983)

[isDomain(): 10](#_Toc129158984)

[isEmail(): 12](#_Toc129158985)

[isUsername(): 13](#_Toc129158986)

[safePassword(): 15](#_Toc129158987)

[Individual team assignments 17](#_Toc129158988)

[Problems that arose during the project 18](#_Toc129158989)

[Scrum meetings 18](#_Toc129158990)

[1 - The call graph of the method isAlphaNum 3](#_Toc129114607)

[2 - The call graph of the method isSpecialChar 4](#_Toc129114608)

[3 - The call graph of the method isPrefixChar 4](#_Toc129114609)

[4 - The call graph of the method isDomainChar 5](#_Toc129114610)

[5 - The call graph of the method singleAtSign 6](#_Toc129114611)

[6 - The call graph of the method fetchBeforeAt 7](#_Toc129114612)

[7 - The call graph of the method fetchAfterAt 8](#_Toc129114613)

[8 - The call graph of the method isPrefix 9](#_Toc129114614)

[9 - The flowchart of the method isPrefix 10](#_Toc129114615)

[10 - The call graph of the method isDomain 11](#_Toc129114616)

[11 - The flowchart of the method isDomain 12](#_Toc129114617)

[12 - The call graph of the method isEmail 13](#_Toc129114618)

[13 - The call graph of the method isUsername 14](#_Toc129114619)

[14 - The flowchart of the method isUsername 15](#_Toc129114620)

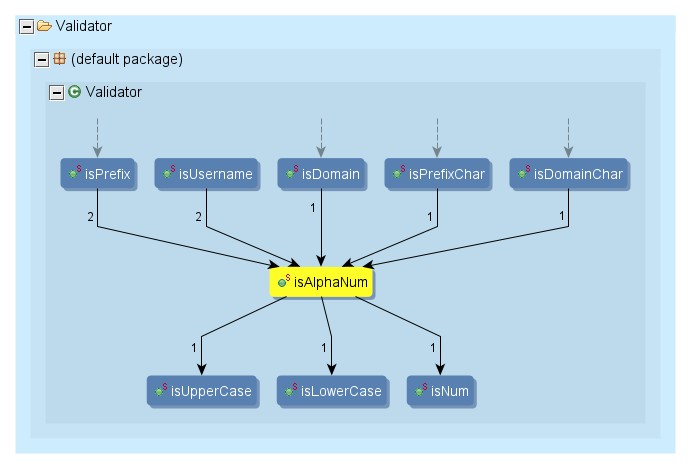
[15 - The call graph of the method safePassword 16](#_Toc129114621)

[16 - The flowchart of the method safePassword 17](#_Toc129114622)

# Overview of the project:

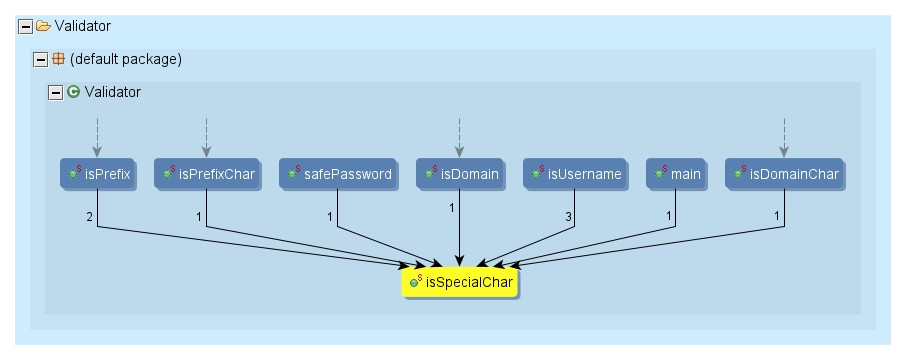
Based on the provided instructions, the final project should be done by groups of two members. By following specific rules, a Java program as a validator evaluated some conditions. For all the evaluations, the program contained methods with different inputs and results. Among the methods, there were both complex and simple codes.  
Obviously the complex methods were more challenging and contained the simple ones; therefore in the complex methods the simple ones were called.  
Since the most efficient evaluation for methods in Java is using specific inputs, the “main” method was used for examining the project methods; by calling them.  
All the methods were public and since creating objects was not necessary, they were all static. The differences in the introductions were the data type each method accepted and their outputs; according to their goals, the inputs and outputs could be “integer”, “boolean”, “String” or “char”.  
The team was asked to handle the project professionally; the members were expected to type helpful comments, in order to understand completely the process of the project at the time of reviewing. There are brief explanations of the goals of the methods in the explanations too.  
Also the names they chose for the variables were in camel case letters and represented their essences.  
Instead of repeating some limited structures, the team tried to use the different concepts of the course as much as possible, especially if these concepts would summarize the codes.  
The members of the group shared their works in “Github”; a source where they could save every step of their progress. For every aspect of the project, a separate folder was created in the Github repository.  
The members of the group had scrum meetings every day, where they shared their information and evaluated each other works. Each meeting had a scrum master who noted what the team did during the meeting.

# Describing the methods of the project:

isAlphaNum():  
To check if a character is alphanumeric, it is checked if it’s either a letter of the English alphabet or a number between 0 and 9. It is evaluated by its ASCII code and since the program is case sensitive, the evaluations of capital and small English alphabet letters are separated. If only one of the three conditions is true, the method returns “true”, otherwise it returns “false”.  


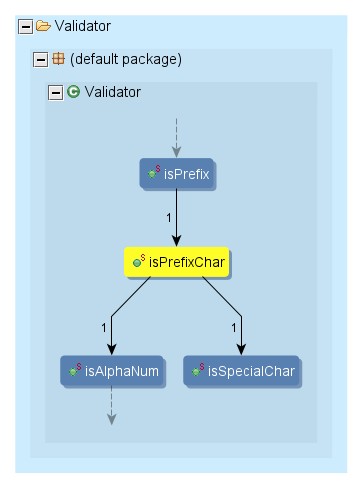
1 - The call graph of the method isAlphaNum

## isSpecialChar():

In this method, there is a specific explanation for a special character; if the character is either a dash or a period, it’s definitely a special character and if the user input “true” as the second argument of the method, the underscore will also be a special character, therefore there’re two conditions in the codes of the method. If the character is a special character, the method returns “true” and otherwise, it returns “false”.  


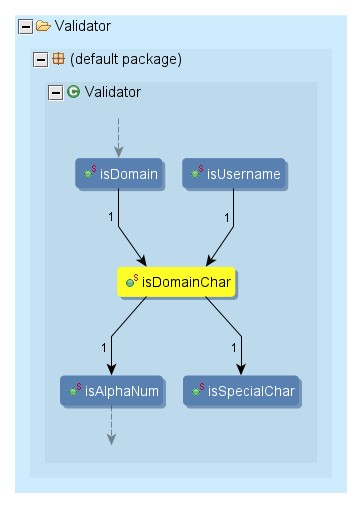
2 - The call graph of the method isSpecialChar

## isPrefixChar():

This method checks if a character is allowed to be a prefix or not. The conditions of being qualified for a prefix are in the instructions. All has to be done is to use the previous two methods and receive true from both of them.  


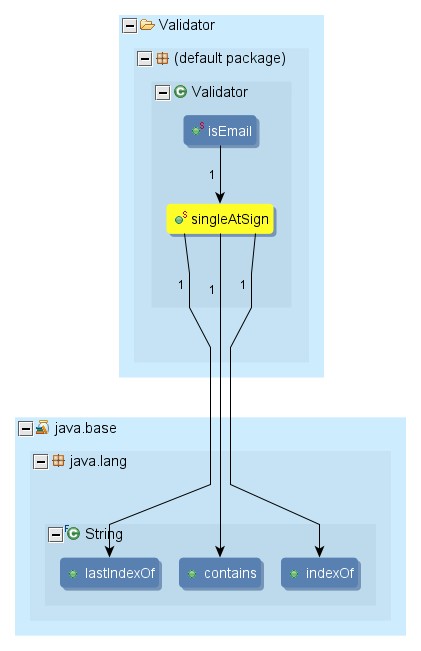
3 - The call graph of the method isPrefixChar

## isDomainChar():

The only difference between this method and the previous method is that in this method underscore is not allowed in the prefix, therefore the boolean input is false.  


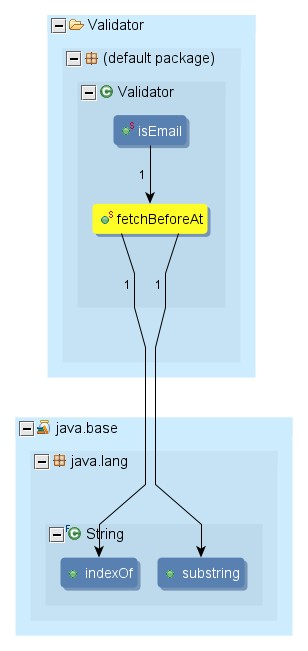
4 - The call graph of the method isDomainChar

## singleAtSign():

It’s obvious that a valid Email address has only one @; not more than one and not less than one, so it’s essential to check if an Email address has only and only one @. Among the methods of the project, this was the first one that received a String as its input, and in these methods, usually it’s important to be sure that the input is not “null” before assigning the other conditions.  


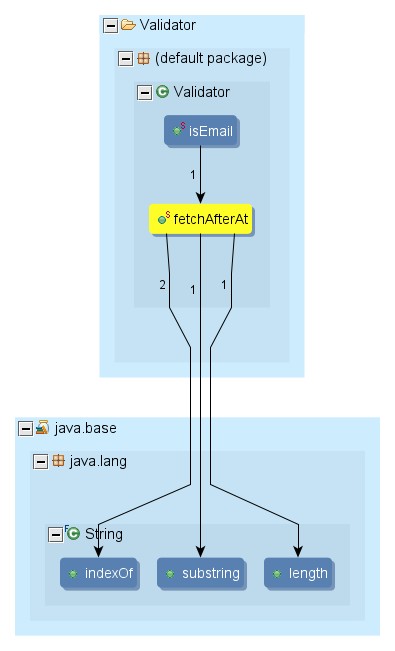
5 - The call graph of the method singleAtSign

## fetchBeforeAt():

After becoming sure that the input has only one @, it’s time to separate what is before @ and what is after it. This method extracts what is before @. As was explained in the explanations of the previous method first it’s essential to be sure that the input is not “null”, then the method separates every letter before @ by using the “substring” method.  


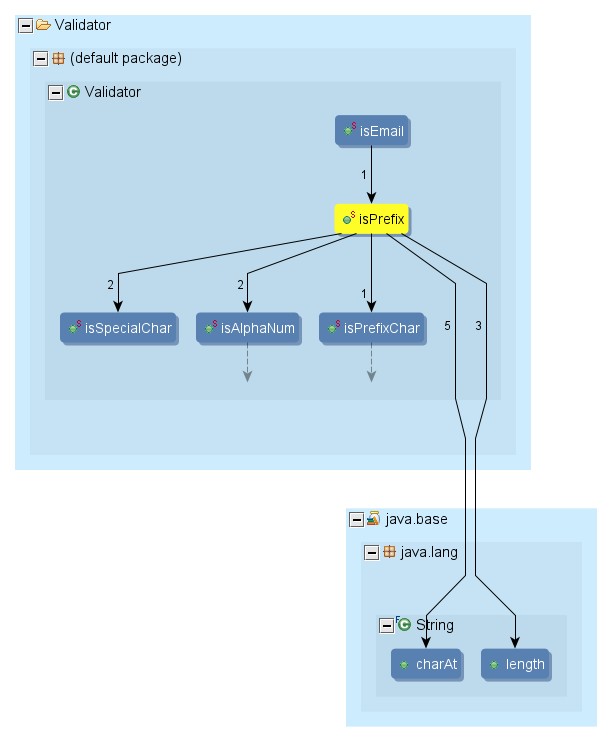
6 - The call graph of the method fetchBeforeAt

## fetchAfterAt():

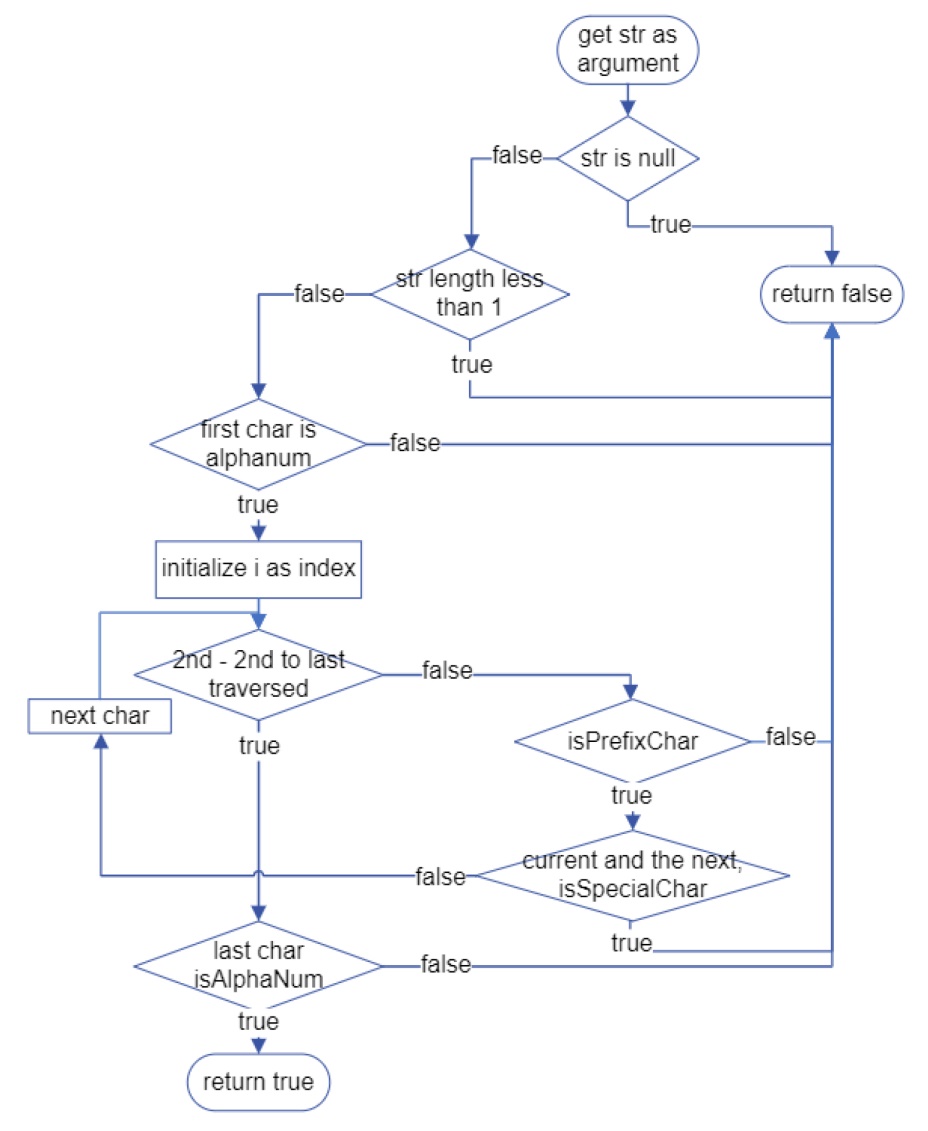
By following the same logic as the previous method, this method separates what is after @.  


7 - The call graph of the method fetchAfterAt

## isPrefix():

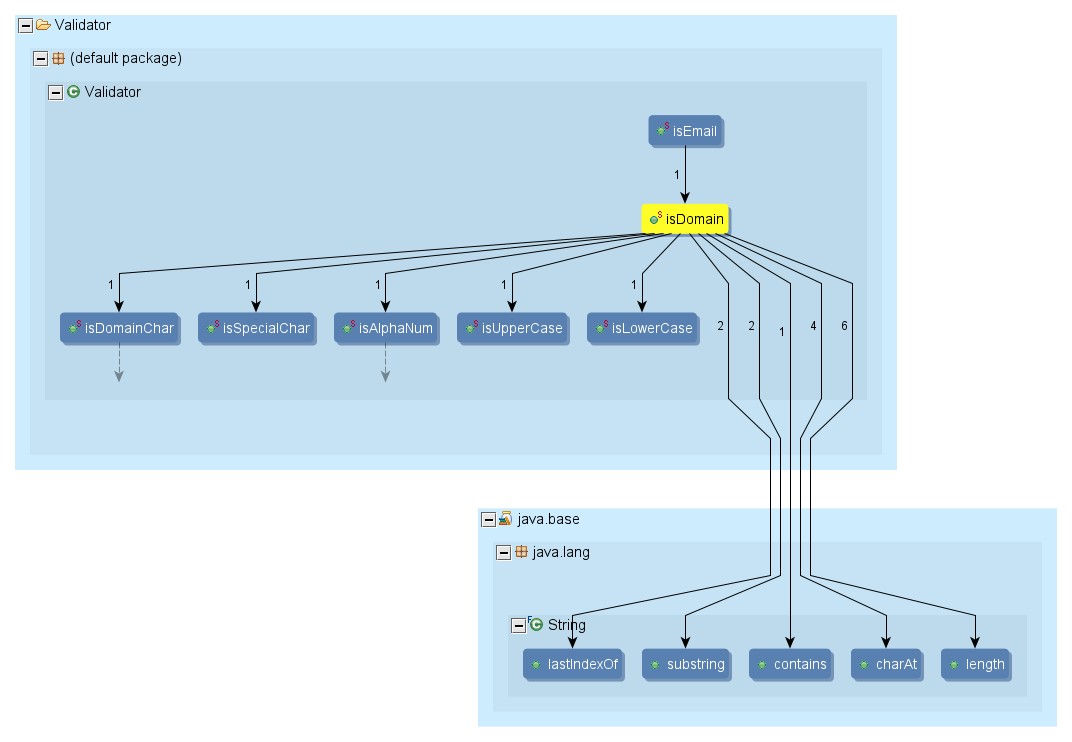
This method evaluates if a String can be a valid Email prefix. The conditions of a valid Email prefix are mentioned in the instructions:   
- The String should contain at least one character, by using the “length” key word the number of the characters are limited to be greater than zero.  
- The first character should be alphanumeric, since the method for evaluating chars to be alphanumeric had already been created; it was used in this method for evaluating the first character.  
- All the characters should be either alphanumeric or underscore or period or dash, by using the isPrefixChar method and the isSpecialChar method which had already been created, all the characters of the String are evaluated to have this condition.  
- An underscore, period, or dash must always be followed by at least one alphanumeric character, in a loop each index is evaluated to have the condition, if there are two characters in a row that are either dash or period or underscore, this condition is not approved.  


8 - The call graph of the method isPrefix

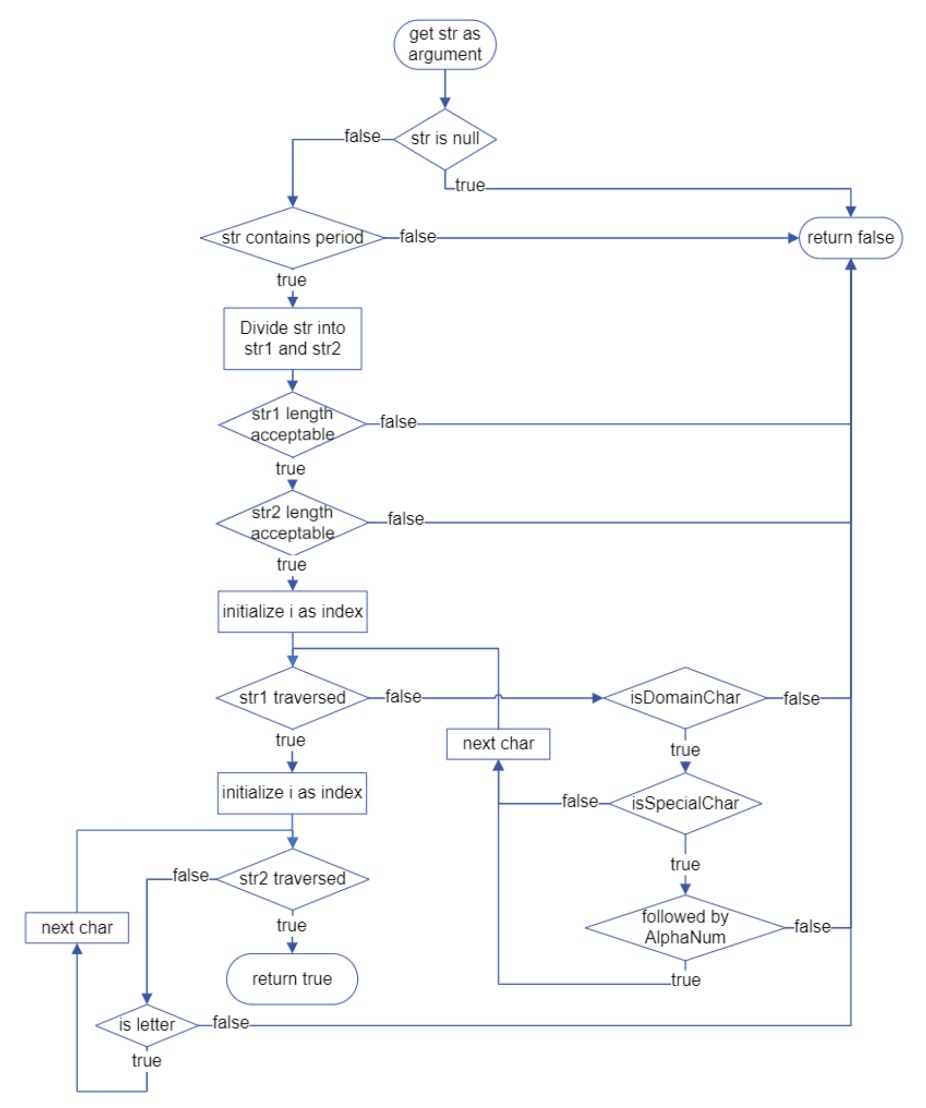


9 - The flowchart of the method isPrefix

## isDomain():

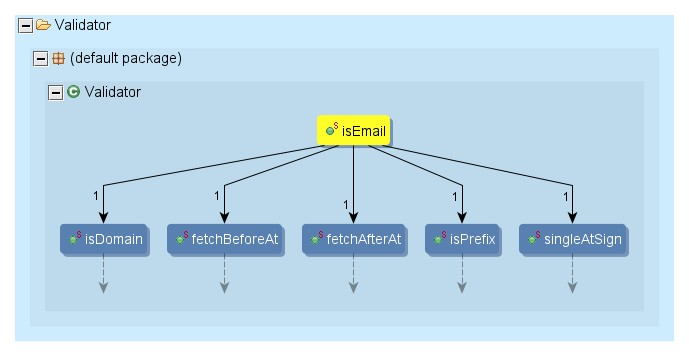
Unlike the previous method, this method evaluates if a String can be a valid Email domain. The conditions of a valid Email domain are mentioned in the instructions:   
- Made up of two portions separated by a period, which means there should be a period in the String.  
For the second and third condition the String is separated into two portions and each has a different limitation.  
- The first portion contains at least one character and the second portion at least two; these limitations are applied by the “length” key word.  
- First portion contains only alphanumeric characters, periods and dashes; by using the isDomainChar method all the letters of the first portion are evaluated.  
- A period or dash, in the first portion must be followed by one or more alphanumeric characters; each character of the first portion is evaluated by its following character, in case if it’s a period or a dash.  
- The second portion contains only letters of the alphabet; since each letter of the alphabet can be either a capital letter or a small letter, both are evaluated.  


10 - The call graph of the method isDomain



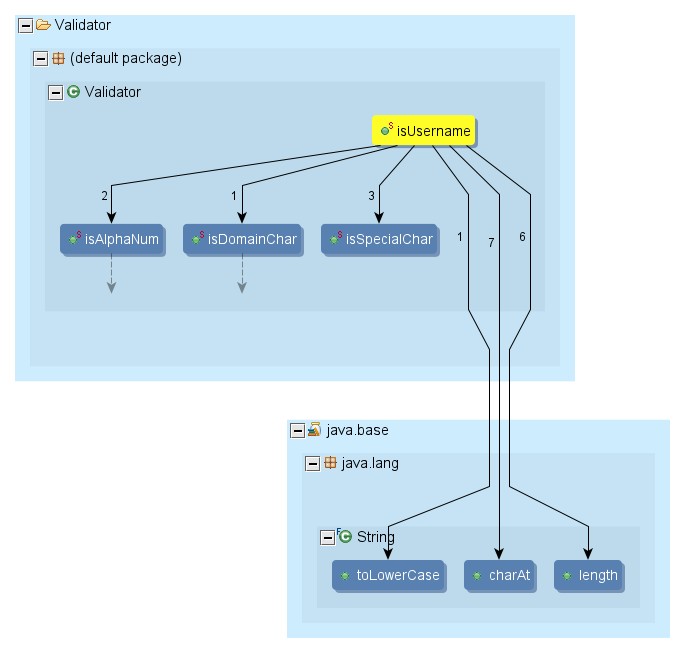
11 - The flowchart of the method isDomain

## isEmail():

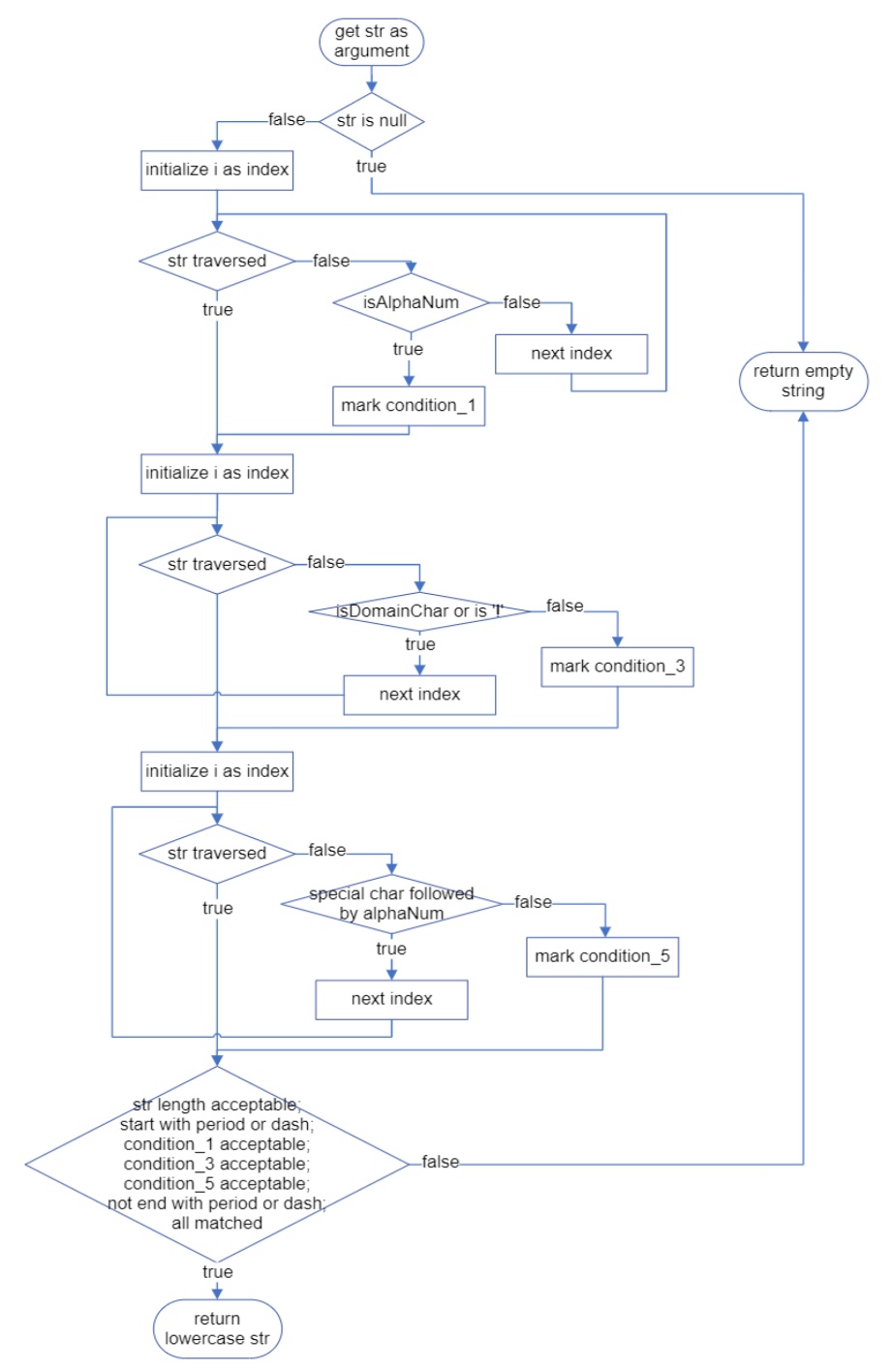
This method evaluates if a String can be a valid Email address. The conditions of a valid Email address are mentioned in the instructions.   
First it’s essential to have only one @ in the String; therefore the singleAtSign method is used.  
What is before @ is called prefix, which is extracted by the fetchBeforeAt method, and it’s evaluated by the isPrefix method.  
What is after @ is called domain, which is extracted by the fetchAfterAt method, and it’s evaluated by the isDomain method.  


12 - The call graph of the method isEmail

## isUsername():

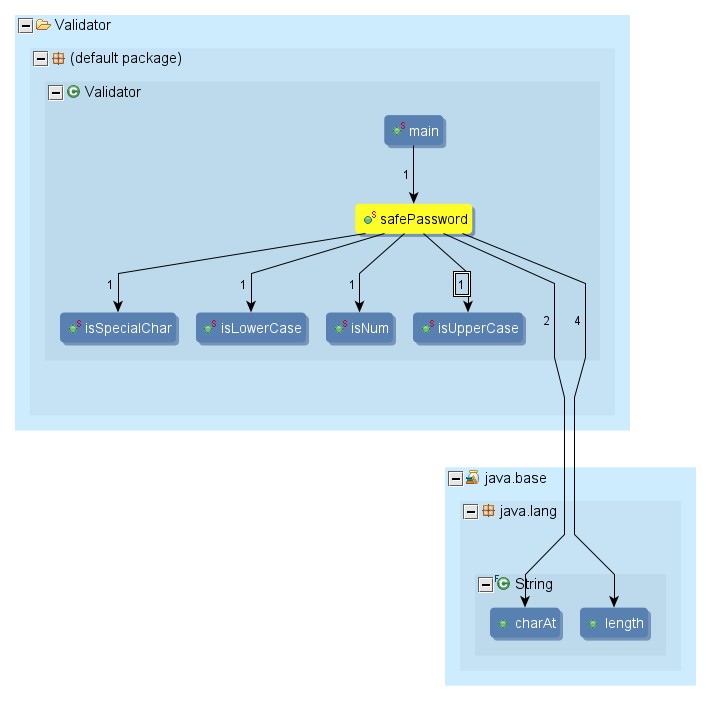
This method evaluates if a String can be a valid username. The conditions of a valid username are mentioned in the instructions.   
- A valid username contains at least one alphanumeric character; each character of the String is evaluated by the isAlphaNum method.  
- Contains only alphanumeric characters, periods, dashes, or an exclamation point; except for evaluating the exclamation point, the other conditions are evaluated by the isDomainChar method.  
- A period, or dash must always be followed by at least one alphanumeric character; each character is evaluated by the isSpecialChar method with the evaluation of the next character by the isAlphaNum method.  
- A valid username contains seven or less characters; by using the “length” key word the numbers of the characters are limited.  
- A valid username must start with a period, or dash; this condition is approved by evaluating the first character of the String with the isSpecialChar method.  


13 - The call graph of the method isUsername

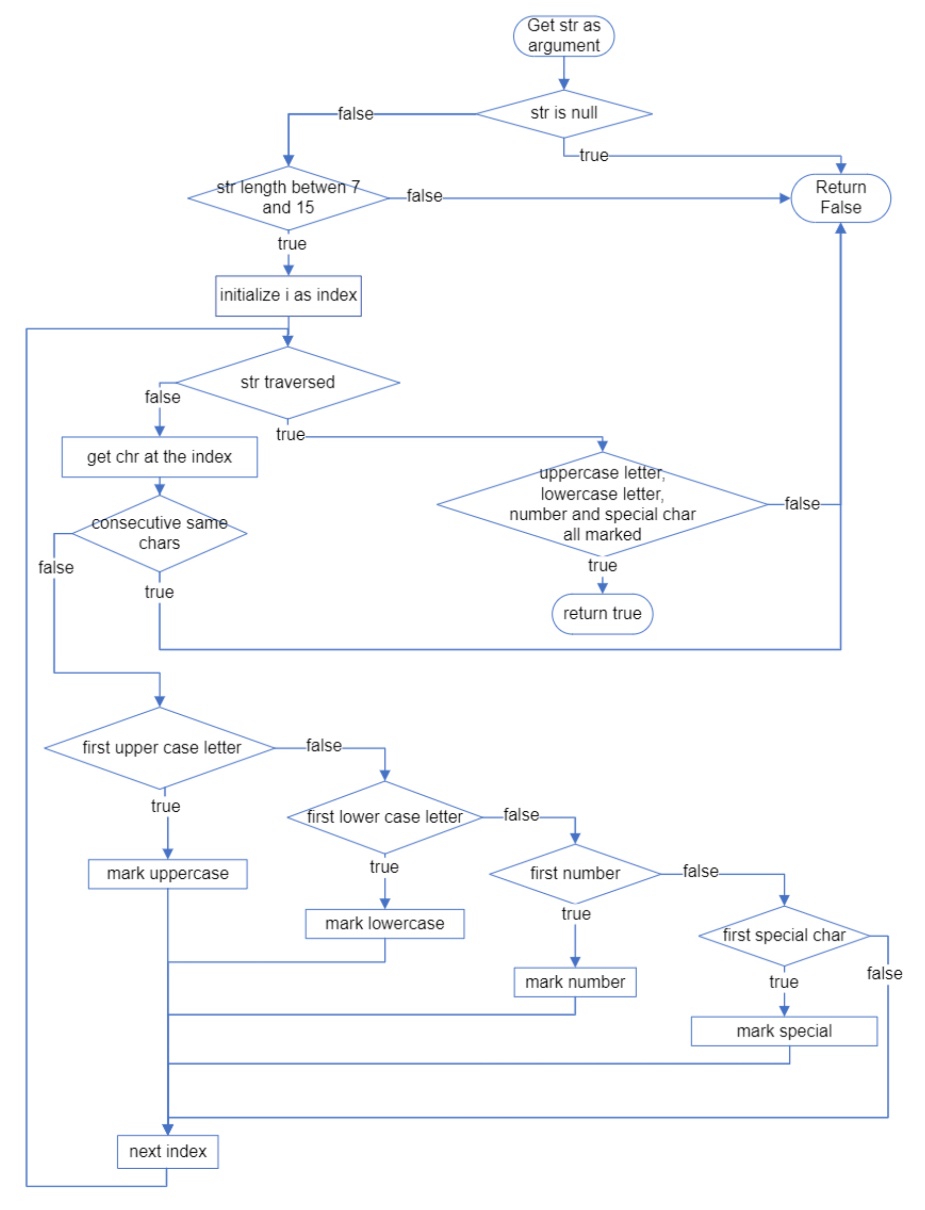


14 - The flowchart of the method isUsername

## safePassword():

This method evaluates if a String can be a safe password. The conditions of a safe password are mentioned in the instructions.  
- A safe password contains a minimum 7 characters and maximum 15 characters; by using the “length” key word this limitation is assigned.  
- A safe password contains at least one uppercase letter, one lowercase letter, one number, and one period, dash or underscore; for each of these conditions a boolean variable is introduced and if each character meet a specific condition, the boolean variable may change.  
- The same character must never be repeated more than twice; each character is evaluated with its following character.  
- A safe password contains at least one alphanumeric character; the limitation of being either an English alphabet letter or a number between 0 and 9 are evaluated by the boolean variables in the second condition.  


15 - The call graph of the method safePassword



16 - The flowchart of the method safePassword

# Individual team assignments

The team tried to share the assignments equally; it was especially essential for programming the methods. The last three methods were much more challenging than the others, so the member who did one of those three methods had a bigger share of the others.  
One member of the group did the debugging process by sharing the screen, with preparing the flowcharts and the call graphs; the other prepared the Word document.  
As was mentioned in the part of “Overview of the project”, the whole group managed the scrum meetings.  
Obviously the team shared every step of the project on “Github” and each member evaluated and supervised the other member’s work, so each member of the group dominates the whole project.

# Problems that arose during the project

* The team didn’t have their scrum meetings in person; the meetings were online, therefore it was difficult to communicate and sometimes the team had to wait for the next meeting.
* Each member had a different style of codding, so the team had to change some of the codes to have one unique style for the project.
* Sometimes understanding the codding logic of the other member wasn’t easy and it was essential to spend some time to figure it out.
* The debugging process had to be done in a separate meeting, one member shared the screen and for the other member it wasn’t easy to follow the process.

# Scrum meetings

First Meeting Summary:

Date: March 3, 12:00 PM to 12:30 PM  
Attendees: Entire group  
Meeting Minutes Taker: Pejman  
  
Agenda:   
1. The group reviewed the project together.  
2. The group discussed their opinions and questions about the project.  
3. A GitHub repository was created for uploading the files.  
4. The group decided to spend the first three days on the Source code and the last two days for the Word documentation.  
  
Tasks for the next meeting:   
1. Dividing the GitHub repository into four folders:  
 - Scrum meetings  
 - Word documentation  
 - Source code  
 - Flowchart and call graphs  
2. Working on the Java project individually.

Second Meeting Summary:

Date: March 4, 11:00 AM to 11:15 AM  
Attendees: Entire group  
Meeting Minutes Taker: Shiyuan  
  
Agenda:   
1. Confirmed the steps of the project.  
2. Discussed the code so far.  
  
Tasks for the next meeting:   
1. To complete coding and upload to Github repository.  
2. To start checking and Debugging each other's code.  
3. To solve the issues that has been raised.

Third Meeting Summary:

Date: March 5, 11:00 AM to 11:30 AM  
Attendees: Entire group  
Meeting Minutes Taker: Pejman  
  
Agenda:   
1. The group reviewed some methods of the source code.  
2. The group provided some questions that should be answered.  
3. The group discussed to finish the project in the following day.  
  
Tasks for the next meeting:   
1. Debugging the source code as a team work.  
2. Working on the Word document.

Fourth Meeting Summary:

Date: March 6, 11:30 AM to 11:45 AM  
Attendees: Entire group  
Meeting Minutes Taker: Shiyuan  
  
Agenda:   
1. Discussed the Division in writing document.  
2. Solved the questions so far.  
  
Tasks for the next meeting:  
1. Finishing debugging.  
2. Preparing the first version of the document.

Fifth Meeting Summary:

Date: March 7, 10:30 AM to 11:00 AM  
Attendees: Entire group  
Meeting Minutes Taker: Pejman  
  
Agenda:   
1. Preparing the flowcharts and call graphs  
2. Apply the new changes of the source code project  
3. Reviewing the Word document