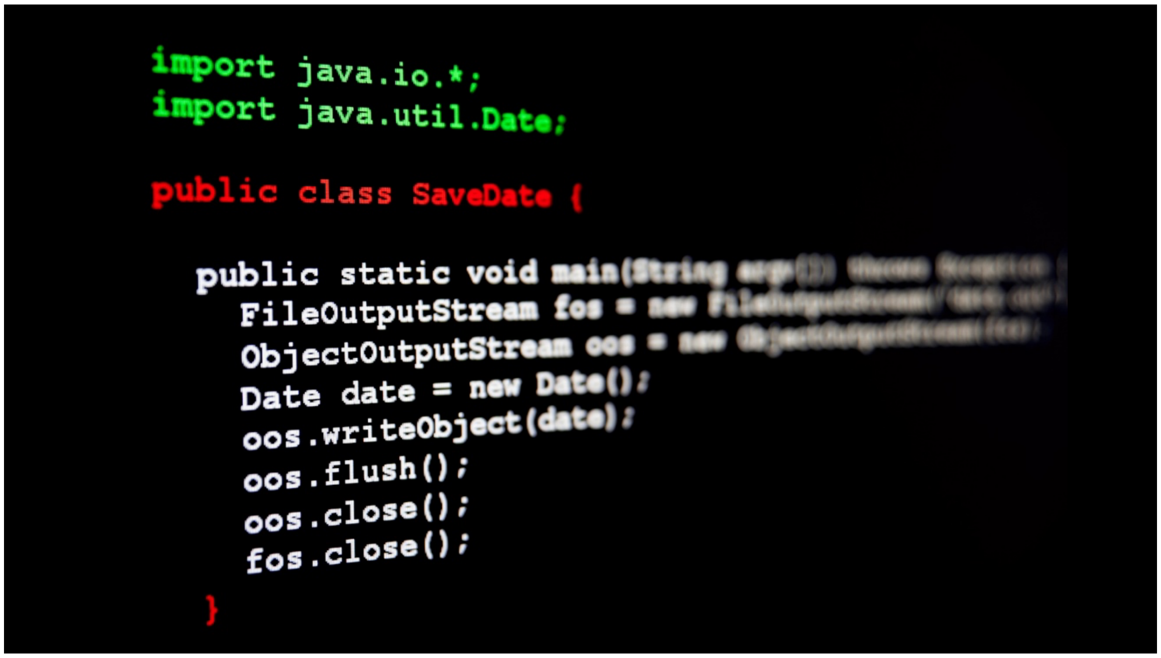
VALIDATOR4

Programming I Final Project



```
import java.io.*;
import java.util.Date;

public class SaveDate {

  public static void main(String [illegible]
    FileOutputStream fos = new [illegible]
    ObjectOutputStream oos = new [illegible]
    Date date = new Date();
    oos.writeObject(date);
    oos.flush();
    oos.close();
    fos.close();
  }
```

FSD-08:
Fawzi Latosh
Dustin Ruck

March 8, 2023

TABLE OF CONTENTS

OVERVIEW

PLANNING & DEVELOPMENT

Our planning process was very simple and straightforward. In our initial meeting, we reviewed all the parameters in the assignment instructions and noted key aspects of our code we would need to be in sync on. This particularly meant focusing a lot on where we can reuse functions when we did code reviews.

In our initial meeting we set out our general idea of a timeline with the understanding it may need to change depending on complexity of the program and any problems that might arise. We decided to break up the workload so that Fawzi tackled methods 1-7 and Dustin took on methods 8-12. Then, in our code review meetings we would look for areas that could be improved, functions that could be reused, and code that could be made more clean and efficient.

MEETINGS

We planned for and expected the project to be completed Tuesday, March 7. This gave us 5 days to work with (Beginning Friday March 3). Scrum meetings were scheduled daily for 6pm. Following our scrum on day 2, 3, 4, and 5, we allocated between 30-60mins for code review.

INDIVIDUAL TEAM ASSIGNMENTS

Fawzi – Methods 1-7, Testing

Dustin – Methods 8-12, Documentation

WORKLOAD, DIFFICULTY & COMPREHENSION

Workload for this project was substantial in comparison the previous course's final project, however it was very manageable. We each spent about 2 hours total on the day 1 & 2, and then 3-5hrs/day on days 3/4/5.

METHODS

METHOD 1
isAlphaNum()
Check if a character is alphanumeric.
"isAlphaNum" takes in a single character as input and returns a Boolean value indicating whether that character is a letter (a-z, A-Z) or a digit(0-9). If the character is a letter or digit, the function returns true; otherwise, it returns false

METHOD 2
isSpecialChar()
Check if a character is an acceptable special character.
This method checks if the character is a period ('.'), a dash ('-'), or an underscore ('_'). If the character is any of these three special characters, the method returns "true". Otherwise, the method returns "false", indicating that the character is not a special character.

METHOD 3
isPrefixChar()

Check there are only alphanumeric characters, dashes, periods, or underscores in prefix.
This method named "isPrefixChar" is a boolean function that takes a single input of type 'char', which represents a character to be checked for validity as a prefix character. The method first checks if the input character is a special character or if it is an alphanumeric character. The 'isSpecialChar' and 'isAlphaNum' are called to verify this. If the input character passes either of these checks, the method returns true. Otherwise, it returns false, indicating that the character is not valid as a prefix.

METHOD 4

isDomainChar()
Domain can contain only alphanumeric characters, dashes, or periods.
The method "isDomainChar" takes a single argument, a character called "domCharac". The method uses the same calling of previous functions to check that a character is considered valid if it is a letter, digit, period, dash, or underscore. If valid, the method returns "true". Otherwise, it returns "false".

METHOD 5

singleAtSign()
Check if an email contains a single at sign (@).
"singleAtSign" takes a string input "str".
The method begins by using a for loop to loop through each character in the input string "str" and checks if each character is equal to "@" using the "charAt()" method. If the character is "@" then the "count" variable is incremented by 1.
After looping through the entire input string, the method checks if the "count" variable equals 1. If it does, the method returns "true" indicating that the input string contains a single "@" character. Otherwise, the method returns "false" indicating that the input string contains either zero or more than one "@" character.

METHOD 6

fetchBeforeAt()

Get the beginning (local part) of an email address.

"fetchBeforeAt" takes a string “str” as input. The input string is expected to be an email address, which is split using the "@" symbol as the delimiter. The method returns the substring before the "@" symbol as a prefix string.

To achieve this, the method uses the split() method of the String class, which splits the input string into an array of substrings based on the @ delimiter. The first element of the resulting array (i.e., parts[0]) is then returned as the prefix string.

METHOD 7

fetchAfterAt()

Get the ending (domain name) of an email address.

This code is identical to method 6 except that it takes the assumed domain, the second part of the array and then returns the “secondPart”.

METHOD 8

isPrefix()

This method first checks if the prefix contains at least two characters and sets word1bt1 to true if it does. Then, it loops over each character in the prefix and checks if each character is a valid character, i.e., a letter, number, underscore, period, or dash. If all the characters in the prefix are valid, then word1bt2 is set to true. If the prefix contains a character that is not valid, then word1bt2 is set to false and the loop is broken.

Next, the method checks if the underscore, period, or dash is always followed by at least one alphanumeric character. To do this, it iterates over each character in the prefix and checks if it is an underscore, period, or dash. If it is, then the method checks if the next character in the prefix is alphanumeric. If it is not, then word1bt4 is set to false and the loop is broken.

Finally, the method checks if the first character in the prefix is alphanumeric. If it is, then word1bt3 is set to true.

After checking all four criteria, the method returns true if word1bt1, word1bt2, word1bt3, and word1bt4 are all true, and false otherwise.

METHOD 9

isDomain()

Check if the end of a String is a valid email domain.

The "isDomain" method checks whether a given string represents a valid domain name or not. It splits the input string into two parts using the period character as the delimiter, and then checks if the first part meets several conditions to be considered valid, including length and character constraints. It also checks the second part of the domain to ensure that it only contains letters and at least 2 of them. If the input string meets all of these conditions, the method returns true, indicating that it is a valid domain. Otherwise, it returns false.

METHOD 10

isEmail

Check if a string is a valid email address.

The "isEmail" method uses the previous methods fetchBeforeAt & fetchAfterAt to split the input string into two parts prefix and domain. It then calls the previous functions singleAtSign, isPrefix, and is domain. If there is a single @ sign and both the prefix and domain meet specific conditions of the called functions, it is considered valid and returns true. Otherwise, it returns false.

METHOD 11

isUsername

Check if a string is a valid username.

The "isUsername" method first checks if the input string is empty or null; if so, it returns an empty string. Then, it checks if the length of the input string is greater than 7; if so, it also returns an empty string. Finally, it checks each character in the input string using the functions isAlphaNum and isSpecialChar and makes sure that it only contains valid characters (i.e., letters, numbers, special characters, and exclamation mark), that input doesn't start with '.' Or '-', and that a period or dash will be followed by an alphanumeric. If the input string meets all of these conditions, the method returns the input converted to lowercase.

METHOD 12

safePassword()

Check if a string is considered a safe password.

“safePassword” takes a String password and verifies it is valid. First it initializes the boolean variables to keep track of whether the password has lowercase, uppercase, digits, special characters, and alphanumeric characters. The **prevChar** variable is set to the null character as a placeholder for the first character of the password, and the **consecutiveCount** variable is set to 1 to keep track of consecutive characters. It uses a for loop to loop through each character. The conditions of having an upper case, lower case, digit, and special character are checked and set to true if so. isSpecialChar method is called to check for the special character. The password is then checked for being alphanumeric and not having two consecutive characters. If password length is correct and all other Booleans are true except for consecutive characters, it is valid and returns true. Otherwise, false.