**INTRODUCTION**

An algorithm is a set of steps of operations to solve a problem performing calculation, data processing, and automated reasoning tasks. An algorithm is an efficient method that can be expressed within finite amount of time and space.

An algorithm is the best way to represent the solution of a particular problem in a very simple and efficient way. If we have an algorithm for a specific problem, then we can implement it in any programming language, meaning that the algorithm is independent from any programming languages.

A permutation, also called an “arrangement number” or “order,” is a rearrangement of the elements of an ordered list S into a one-to-one correspondence with S itself. A string of length n has n! permutation. Printing all permutations of a given string is an example of backtracking problem. Backtracking is a general algorithm for finding all (or some) solutions to some computational problems, that incrementally builds candidates to the solutions, and abandons a candidate (backtracks) as soon as it determines that the candidate cannot possibly be completed to a valid solution. The complexity of this algorithm is O(n!). It is a huge complexity. When the string size increases, it takes a longer time to finish the task.

String Matching is the classical and existing problem, despite the fact that the real world aspects belonging to the research field of computer science. In this domain one or several strings called " Pattern " is to be searched within a well-built string or " Text ". String matching strategies or algorithms provide key role in various real world problems or applications. A few of its imperative applications are Spell Checkers, Spam Filters, Intrusion Detection System, Search Engines, Plagiarism Detection, Bioinformatics, Digital Forensics and Information Retrieval Systems etc.

**DESIGN**

ALGORITHM:

TO FIND GIVEN PERMUTATIONS:

1) Read a string

2) Create an empty array list to store permutations

3) Initialize the list with first character of string

4) Do for every character using for loop

5) Remove current partial from list

6) Backtrack and insert next character in all possible appositive of current partial permutations

7) Insert all the strings in the list

STRING MATCHING ALGORITHM:

1) Array list is converted to array arr[i]

2) Read the string with which you want to match the permutations with

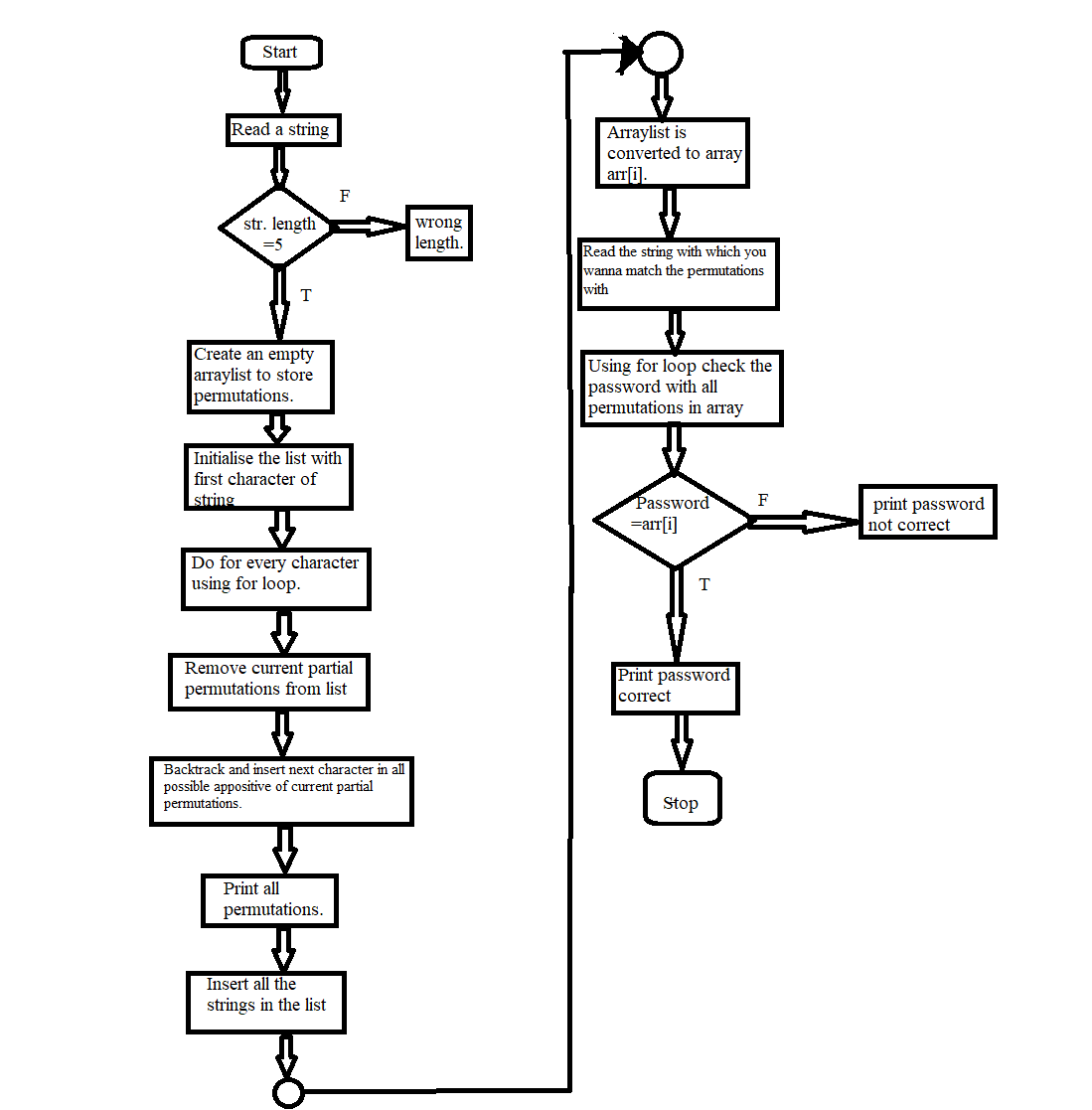
3) Using for loop check the password with all permutations in array

4) If password == arr[i]

5) Print password correct

6) Else print password not correct

FLOWCHART:



**IMPLEMENTATION**

package ada;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.Scanner;

class print {

// Iterative function to generate all permutations of a String in Java

// using Collections

public static void permutations(String s)

{

// create an empty ArrayList to store (partial) permutations

List<String> partial = new ArrayList<>();

// initialize the list with the first character of the string

partial.add(String.valueOf(s.charAt(0)));

// do for every character of the specified string

for (int i = 1; i < s.length(); i++)

{

// consider previously constructed partial permutation one by one

// (iterate backwards to avoid ConcurrentModificationException)

for (int j = partial.size() - 1; j >= 0 ; j--)

{

// remove current partial permutation from the ArrayList

String str = partial.remove(j);

// Insert next character of the specified string in all

// possible positions of current partial permutation. Then

// insert each of these newly constructed string in the list

for (int k = 0; k <= str.length(); k++)

{

// Advice: use StringBuilder for concatenation

partial.add(str.substring(0, k) + s.charAt(i) +

str.substring(k));

}

}

}

System.out.println(partial);

String[] arr=new String[partial.size()];

 arr=partial.toArray(arr);

Scanner S=new Scanner(System.in);

System.out.println("Enter the password:");

    String str1=S.nextLine();

int len=arr.length;

   for( int i=0;i<len;i++)

   {

        if  (str1.equals(arr[i])) {

         System.out.println("password correct");

         System.exit(0);

        }

}

         System.out.println("password incorrect");

    }

// Iterative program to generate all permutations of a String in Java

public static void main(String[] args)

{

Scanner S=new Scanner(System.in);

        System.out.println("Enter the string with one special character and atleast 1 number:");

        String str=S.nextLine();

          int n=str.length();

        if(n==5)

        {

 permutations(str);

        }

        else

        {

          System.out.println("wrong length");

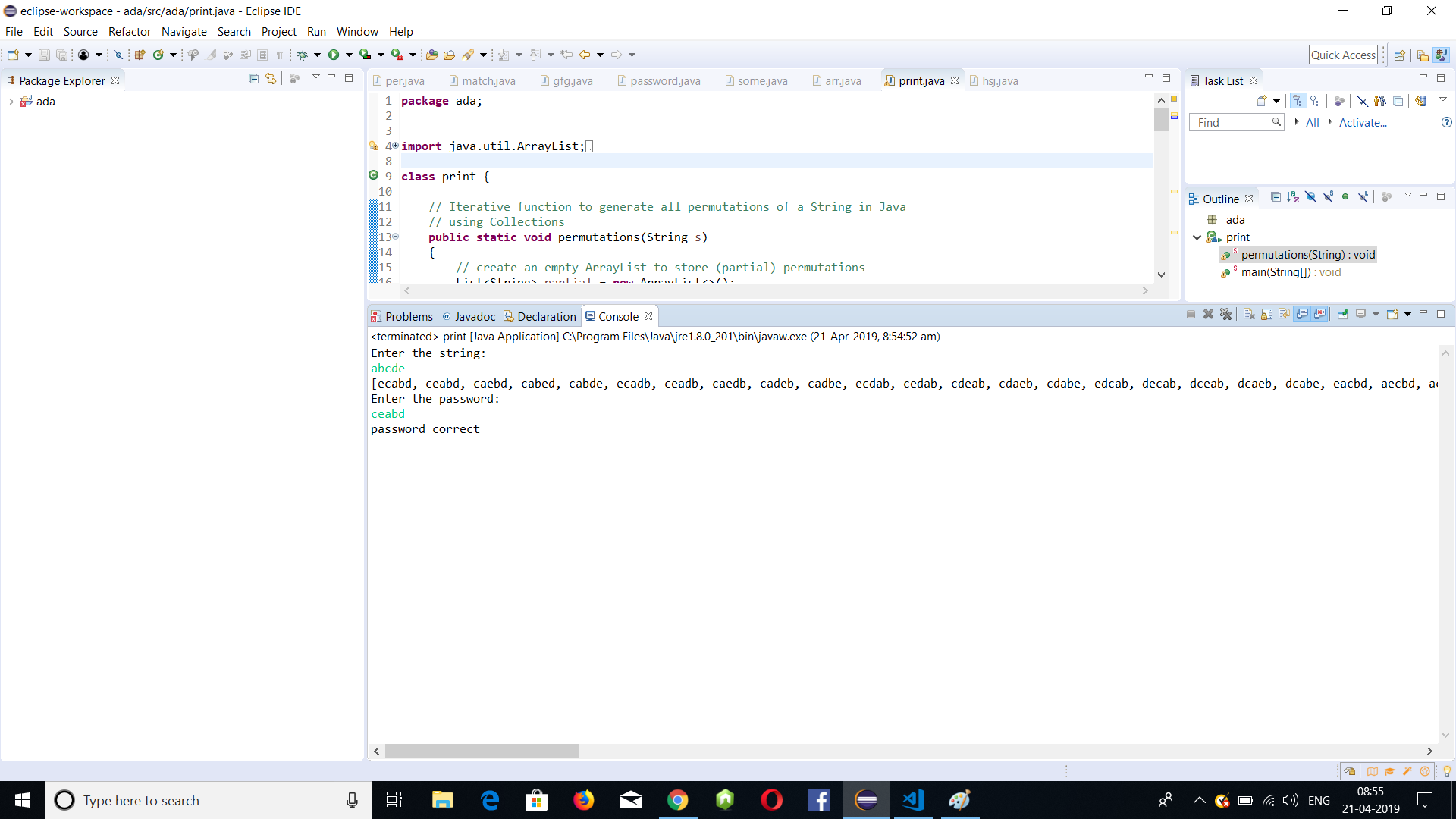
        }

 }

}

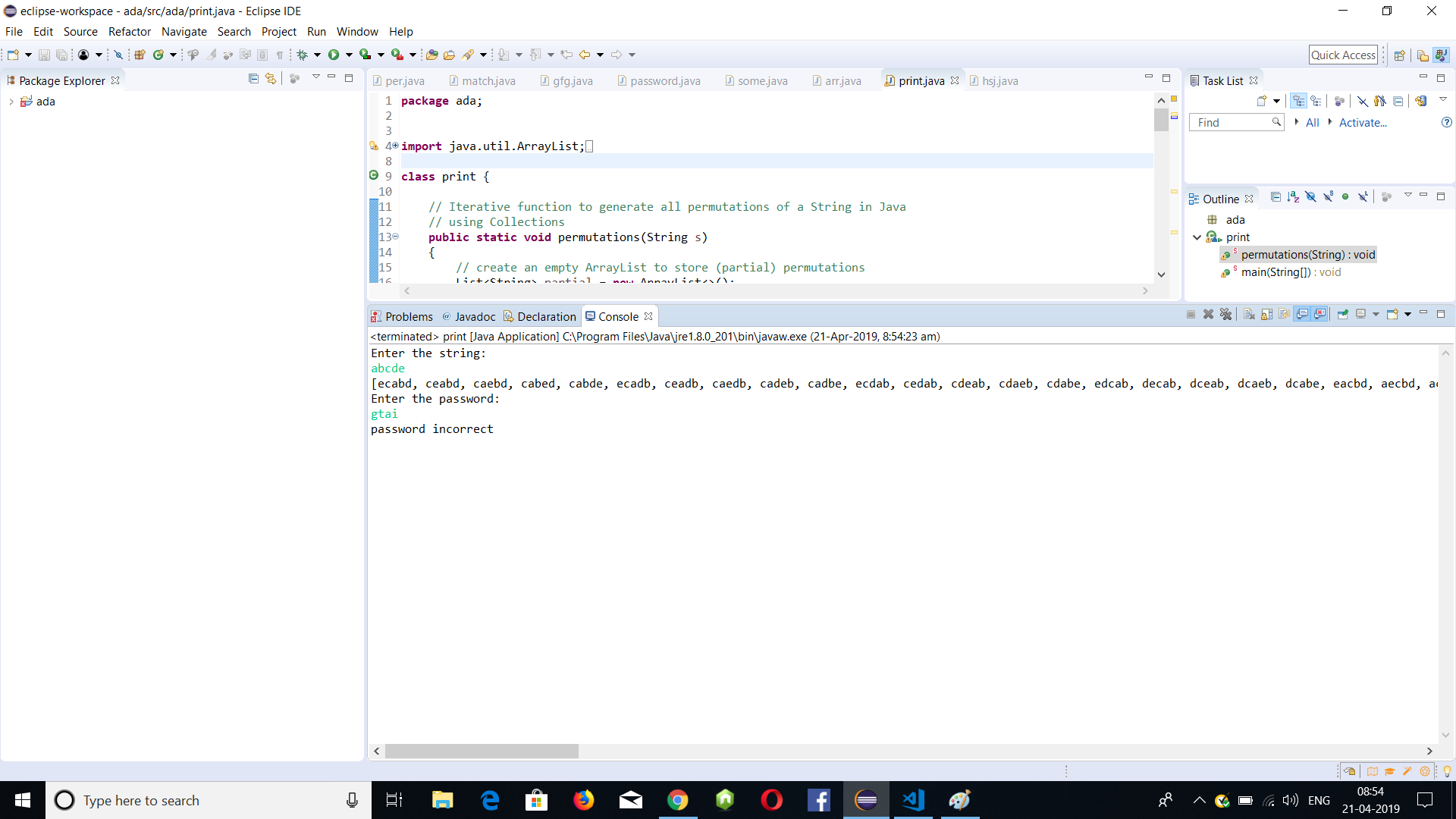
**OUTPUT**

Run 1:



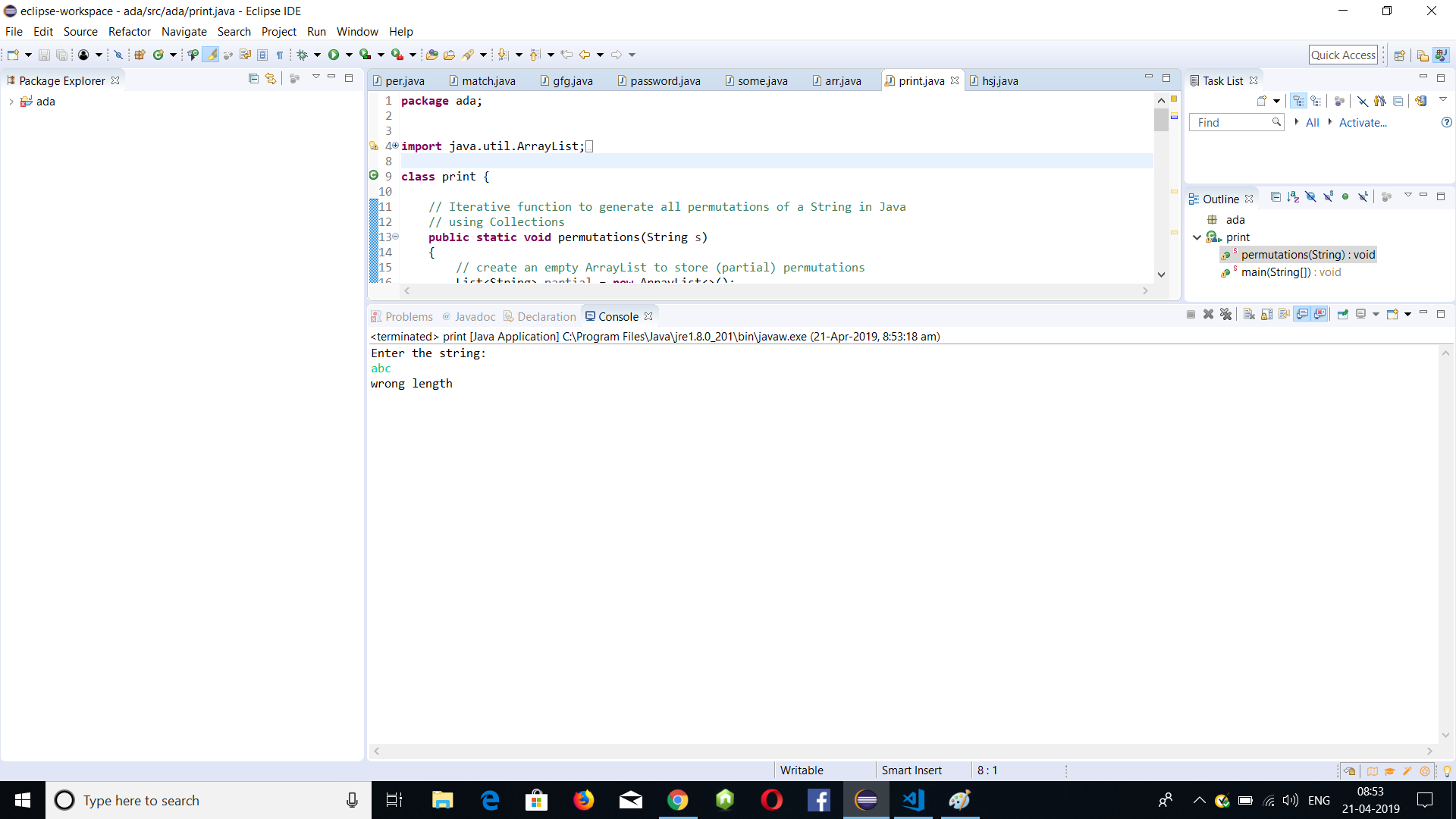
The password is correct.

Run 2:



The password is incorrect.

Run 3:



Wrong length.

**CONCLUSION**

In the project, we find all the permutations of the user input string. The permutations are then stored in an array list. The password entered by the user is then compared with all the permutations in the array using string matching algorithm. If the entered password matches with any of the permutation stored in the array, then the “password correct” is the output else “password incorrect” Is the output.

# FUTURE ENHANCEMENTS

We can create a system which notifies the user when the password is placed incorrectly multiple times.

**REFERENCES**

[1]*.* Christian Charras and Thierry Lecroq,*” Handbook of Exact String\_Matching Algorithms”,* Published in King’s college publication, Feb 2004.

[2]. Alberto Apostolico and ZviGalil*,” Pattern Matching Algorithms”* Published in Oxford University Press, USA, 1st edition, May 29, 1997.

[3*]* Ching-Tung Wu, Kwang-Ting Cheng, Qiang Zhu and Yi-Leh Wu, *“Using Visual Feature For Anti-Spam Filtering”*, In the proc. of IEEE International Conference on Image Processing (ICIP2005), pp. 509–512, 2005.

[4]. Hyunjin Kim, Hong-Sik Kim and Sungho Kang,"*A Memory-Efficient Bit-Split Parallel String Matching Using Pattern Dividing for Intrusion Detection Systems*” IEEE Transactions on Parallel and Distributed Systems, Volume:22 , Issue: 11, pp. 1904-1911, Nov 2011