**Important:**You are not allowed to use the classes/interfaces defined in the Java Collection Framework. You can use/extend the examples/solutions I shared on our course's Microsoft Teams page. You are not allowed to use code copied from the Internet.

**Background:**You find yourself in a predicament with a critically important file encrypted with a 12-character secret key. This file holds invaluable notes and insights from your Algorithms course. Regrettably, the secret key has slipped your memory, but you recall it comprises 5 characters inspired by your favorite coffee: **'M'**, **'O'**, **'C'**, **'H'**, **'A'**. Using brute force to guess the key would take an overwhelming 512 rounds. However, you possess a clever hack: a function that, given a guessed secret key, returns the count of matched positions between the guessed key and the correct key.

**Technical Description:**The correct secret key contains exactly 12 letters, each of which must be either **'M'**, **'O'**, **'C'**, **'H'**, or **'A'**. It is managed by a class **SecretKey.**You need to modify the class **SecretKeyGuesser (refer to source code section below).** The class contains one required public method (you can add more private methods/attributes as needed):

**void start()**

In this **start()** method, your code must:

1. First, create a new instance of the **SecretKey** class (I will provide a sample **SecretKey** implementation - you don't need to change anything)

2. Then, repeatedly call the **guess()** method of the **SecretKey** instance.

* The argument for each call must be a String containing exactly 12 letters, with each letter being **'M'**, **'O'**, **'C'**, **'H'**, or **'A'**. An invalid argument results in -1 as the return value.
* The**guess()** method will return a value from 0 to 12, indicating how many positions in your guessed value match the correct secret key.

3. The process must terminate when the **guess()** method returns 12. At this point, display the correct secret key and the number of guesses that has been achieved.

Ensure that your code adheres to the specified requirements and maintains the integrity of the guessing process.

**More information about testing your program:**

The following steps will be used to test your program.  
1. A secret key is generated (your program will NOT know anything about this key)  
2. An instance of your **SecretKeyGuesser** class is created  
3. The **start()** method of the **SecretKeyGuesser** instance will be called  
4. Anytime your program calls the **guess()** method, a counter value is increased (i.e. the number of guesses counter)  
5. Your program must try to make this counter as small as possible when it finds out the correct secret key  
6. I will run your program with three test cases. Then, the three counter values are added together. This is the performance of your program (the smaller, the better)

**Source code:** Study the source code provided here[**<click here>**](https://rmiteduau.sharepoint.com/:f:/r/sites/COSC2658_COSC2469_COSC2203_DSA_AA_Sem3_2023/Shared%20Documents/Lecture_sessions/Assessment/Group%20Project?csf=1&web=1&e=qSFXeY)**.**If you compile and execute the code, the number of guesses is high since it is using brute force.

Deliverables

You need to provide me with the following outputs regarding the system you develop

* A README.txt file that describes the contribution score of all members (explained in the Contribution Score section below) and contains the link to the project video (explained in the Video Demonstration section below)
* Source code: Java source code (no library included)
* Technical report: a PDF document that contains the following sections
  + An overview and high-level design of your system (Java classes, methods, their relationships, software design patterns, etc.)
  + The data structures and algorithms you apply or develop (note: you are not restricted to the data structures and algorithms covered in this course; you can create your own data structures and algorithms). You need to describe in detail the working of your data structures and algorithms.
  + Complexity analysis: you need to provide the complexity analysis of the algorithms you proposed.
  + Evaluation: you need to describe how you evaluate the correctness and efficiency of your system experimentally.
* Video demonstration: create a short video (around 5 minutes and less than 8 minutes) to show a demo of your system. You have to upload your video to YouTube and present its URL in the REAME.txt file.

Contribution Score

The starting score for each student is 3 points. Each group can decide to add/remove points to/from each member, but the total point of the whole group is kept unchanged (it is = (the number of members) \* 3). Additional rules:

* The maximum point for a member is 5.
* If a member gets zero points => that member gets zero for the whole group project assignment. In this case, the total point of the whole group = (the number of members whose scores > zero) \* 3.
* The contribution score must be agreed upon by all members. If there are disagreements, you must inform the lecturer/coordinator before the due time.
* The maximum score for the whole project is 30. If you get more than 30 (due to a high contribution score), the final score is 30.

Support Resources

This assessment requires that you meet RMIT's expectations for academic integrity. More information and advice on how to avoid plagiarism are available in the Getting Started module.