The following Java application represents a simple e-mailbox management system.
Note that the parts of the problem are independents.

First, consider the two classes: **UserAccount** and **Message**.

A **Message** has a sender (userAccount), one or more receivers (arraylist of userAccount), and a message body (String).
A **UserAccount** describes the information of a user account: the first name (String), last name (String), the e-mail address (String), and a password (String). It also contains the Mailbox (arraylist of Message(s)).

UserAccount
- FistName : String - LastName : String - Email : String - Password : String - MailBox: ArrayList<Message>
+ UserAccount(String firstname, String lastname, String password) + SendMessage(String text, ArrayList<UserAccount> receivers) : Message + ReadMessage (int i) : String + PrintInfo() : String + getMailBox : ArrayList<Message>

Message
- Sender : UserAccount - Receivers : ArrayList<UserAccount> - Body : String
+ Message (UserAccount sender, ArrayList<UserAccount> receivers, String subject, String body) + getBody() : String + getReceivers() : ArrayList<UserAccount>

For a UserAccount :

- We suppose that the e-mail address is automatically generated in the constructor to be in the format FirstName.LastName@mailbox.com.
- The method SendMessage takes the body of a message (text) and the list of receivers (to whom the message would be sent). This method should add a new Message that has the body (text) to the MailBox of the receivers. The method returns the sent message.

- The method ReadMessage returns the body of the message located in the position i of the MailBox.

- The method PrintInfo returns the first name and the last name of the UserAccount, separated by the space character.

1. Write the two classes Message, and UserAccount.

2. In the driver:

   a. Write the method **UserAccount CreateAccount()** that allows to create a UserAccount. The method should prompt the user to enter the first name, the last name and the password. The password should be composed of minimum 8 characters and contains at least one capital letter. In case of wrong password, it should be reentered.

   b. Write a method **void Friends(UserAccount UA)** that takes the UserAccount UA and print the list of all users (First name, last name) from whom UA has received messages and that without redundancy.

   c. Write a the main method, that allows to creates accounts (using the CreateAccount method), send messages, and print the list of friends (using the method Friends) of a giving account.

3. In this part, we would like to avoid duplicated UserAccount with the same e-mail address.
   Suppose the user (Fist name: "X", last name "Y") has already a UserAccount (his e-mail address is then X.Y@mailbox.com). If a user with the same information (Fist name: "X", last name "Y") would like to create another UserAccount, he will get another e-mail address: X.Y1@mailbox.com. Again, If a user with the same information (Fist name: "X", last name "Y") would like to create another UserAccount, he will get another e-mail address: X.Y2@mailbox.com, and so on.

   Find a solution (in the class UserAccount) for this problem, by adding the necessary codes to the class UserAccount.

4. Now we suppose that there are two types of UserAccount (inheritance): Trial and Premium.
   a. A Trial UserAccount has at most 100 messages that he can send. Write the class TrialUserAccount, while taking into consideration the above assumption.

   b. A Premium UserAccount may send unlimited number of messages, but he has a list of blocked UserAccount to whom he would not be able to send messages (but he can still receive from them). Write the class PremiumUserAccount, which should implement the method **blockUserAccount** that allows to block a UserAccount, and should override the method **sendMessage** of the super class to take into consideration the above assumption.
   c. Re-write the main method of the driver in order to take into consideration the two types of accounts, and test all the possible choices.