CS1020 Sit-in Lab 03 B – Email Service
Semester 2 AY2014/2015

IMPORTANT NOTE: You are to use the appropriate Java API (LinkedList/Queue/Stack) for this problem. You do not need to implement LinkedList/Queue/Stack yourself.

Problem Description

Part 1

We want to simulate the operations of an email service. We want to maintain a person's inbox. At any point of time, a person may receive a new email or may want to read a few emails. Always the most recent email is read first. Consider a threaded view of emails, where each thread is defined by the sender and the subject of an email. When we receive a new email, if there is already an email from the same sender with the same subject in the inbox, then all such mails belonging to the same thread also become recent. The relative order of emails inside a thread need to be maintained

We consider that in our scheduling system we have the following 2 operations:

1. **receive <*sender*> <*subject*> <content>**
   a. *sender* is a single word and is the name of the sender
   b. *subject* is a single word and is the subject of the email
   c. *content* is a single word and is the content of the email
2. **read <numberOfMailsToRead>**

   The user will specify the number of emails he/she wants to read. The most recent emails from the inbox should be deleted after this operation. If the number of mails a user wants to read exceeds the number of emails in the inbox, the program should delete all the emails.

Input

The first line of input is a positive integer *N* indicating the number of operations that will follow. The next *N* lines are the operations to be performed. Consider the following example. (Note that comments on the right are for explanatory purpose and they do not appear in the input.)

`7`	There are 7 operations
`receive C Programming C++`	Receive email from *C* with subject *Programming* and content *C++*
`receive K CS Old`	Receive email from *K* with subject *CS* and content *Old*
`receive K CS New`	Receive email from *K* with subject *CS* and content *New*
`receive B AI Prolog`	Receive email from *B* with subject *AI* and content *Prolog*
`receive A DBMS SQL`	Receive email from *A* with subject *DBMS* and content *SQL*
`receive K CS Now`	Receive email from *K* with subject *CS* and content *Now*
`read 2`	Read *2* emails

Output

After each operation the program should display the current state of inbox from most recent to least recent.

If there are no emails in the inbox, print out *Inbox Empty!*

Output for each operation should be followed by a blank line. For the above input the output is shown on the right:

```
C,Programming,C++

K,CS,Old
C,Programming,C++

K,CS,New
K,CS,Old
C,Programming,C++

B,AI,Prolog
K,CS,New
K,CS,Old
C,Programming,C++

A,DBMS,SQL
B,AI,Prolog
K,CS,New
K,CS,Old
C,Programming,C++

K,CS,Now
K,CS,New
K,CS,Old
A,DBMS,SQL
B,AI,Prolog
C,Programming,C++

K,CS,Old
A,DBMS,SQL
B,AI,Prolog
C,Programming,C++
```

Important notes:

- Please check the input and output files in your account.
- 7 test cases will be used for Part 1, and 3 test cases for Part 2 (see below).

Part 2

Now consider another functionality that is present in your inbox. A user at any point of time may mark one conversation thread as important. If a thread is marked as important, then emails of that thread are made recent. Note that if a thread contains multiple emails then the relative ordering of the emails need to be maintained.

The following operation needs to be supported by the email service:

3. **mark <*sender*> <*subject*>**
    a. *sender* is a single word and is the name of the sender
    b. *subject* is a single word and is the subject of the email

Input: Consider the following input:

```
7
receive C Programming C++
receive K CS Old
receive K CS New
receive B AI Prolog
mark B AI
mark K CS
read 2
```

There are 7 operations
Receive email from *C* with subject *Programming* and content *C++*
Receive email from *K* with subject *CS* and content *Old*
Receive email from *K* with subject *CS* and content *New*
Receive email from *B* with subject *AI* and content *Prolog*
Mark thread with sender *B* and subject *AI* as important
Mark thread with sender *K* and subject *CS* as important
Read 2 emails

Output:

The output for the above input is shown below

```
C,Programming,C++

K,CS,Old
C,Programming,C++

K,CS,New
K,CS,Old
C,Programming,C++

B,AI,Prolog
K,CS,New
K,CS,Old
C,Programming,C++

B,AI,Prolog
K,CS,New
K,CS,Old
C,Programming,C++

K,CS,New
K,CS,Old
B,AI,Prolog
C,Programming,C++

B,AI,Prolog
C,Programming,C++
```

Skeleton Program

You are provided with a skeleton program **EmailService.java** which is shown on the next page. Do not create any other file. Do not change the name of the **EmailService** class.

The class **Email** has been done for you. There is no need to change this class. Apart from this, you may change the skeleton code in the **EmailService** class, add methods and classes any way you deem fit.

```
import java.util.*;

/* Class representing an email */
 class Email {
    String sender;
    String subject;
    String content;

    public Email(String sender, String subject, String content) {
        this.sender = sender;
        this.subject = subject;
        this.content = content;
    }

    public String getSender() { return sender; }

    public void setSender(String sender) { this.sender = sender; }

    public String getSubject() { return subject; }

    public void setSubject(String subject) {
        this.subject = subject;
    }

    public String getContent() { return content; }

    public void setContent(String content) {
        this.content = content;
    }

    public boolean isSameThread(Email mail1) {
        return (this.sender.equals(mail1.sender)
                && this.subject.equals(mail1.subject));
    }

    public String toString() {
        return new String(sender + "," + subject + "," + content);
    }
}

/* Class representing EMail service */
public class EmailService {
   private Stack<Email> inbox;


}
```