# CS1020 Sit-in Lab 03 A - Messaging Application

Semester 2 AY2013/2014

## Requirement

Use Java LinkedList to help you in solving this sit-in lab. **Not using LinkedList will result in correctness marks being halved !**

## Problem Description - Part A

In this problem you are asked to write a program to simulate a messaging application. In this application, a user can send messages to any other user in the system. This is the **message** operation which is as follows:

1. message <user1> <user2> <message>

   (a) *user1* and *user2* are both single word names and are guaranteed to be unique and different from each other.

   (b) *user1* is the user sending the *message* represented in this simulation by an integer. *message* is unique for each operation and has value $\geq 0$.

   (c) *user2* is the user receiving the *message*.

Each user will keep a list of messages he has **sent and received**. However he can keep at most **10 messages** and if there are $>$ 10 messages, the **least recent** message in his list will have to be deleted to allow subsequent messages to be stored.

### Input

The 1st line in the input is an integer **N** (**N** $\geq$ 2), giving the number of users of the application. The following N lines are the names of the N users. These are single word names and are unique. After that is a line containing the integer **M** (**M** $\geq$ 1) indicating the number of operations. The next M lines are the M operations given in **chronological order**. An example input is given below:

```
4     <-- 4 users
John  <-- 1st user
Mark  <-- 2nd user
Betty <-- 3rd user
Tom   <-- 4th user
3     <-- 3 operations
message John Mark 1   <-- John messages Mark with message 1
message John Betty 10 <-- John messages Betty with message 10
message Mark Betty 21 <-- Mark messages Betty with message 21
```

**Output**

For each operation, print out the operation followed by the list of messages for each user. The order of users must be the same order as given in the input. For each user, first print out the user name followed by a SPACE, then print the value of each message in his message list from earliest message to latest message. The messages are to be separated by a COMMA, WITHOUT any space. If a user has no message print an @. Output for each operation should be followed by an empty line. Example output for the given input above is shown below:

```
message John Mark 1
John 1
Mark 1
Betty @
Tom @

message John Betty 10
John 1,10
Mark 1
Betty 10
Tom @

message Mark Betty 21
John 1,10
Mark 1,21
Betty 10,21
Tom @

```

## Skeleton Program

Your program is to be named **MessageApp.java** (do not change this name). A skeleton program is provided, but you can change it any way or add new class(es) and method(s) as you deem fit. **Write all your code in MessageApp.java, do not create new files!** The skeleton program is as follows:

```
import java.util.*;

/* Service class representing a user */
class User {

}

/* Client class to simulate the messaging application */
public class MessageApp {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

    }
}
```

## Testing your program

The following **sample** input and output files are in your plab account:

```
MessageAppTC1.in, MessageAppTC2.in, MessageAppTC3.in, MessageAppTC4.in
MessageAppTC1.out, MessageAppTC2.out, MessageAppTC3.out, MessageAppTC4.out
```

MessageAppTC1.in and MessageAppTC2.in are input test cases for **part A**, while MessageAppTC3.in and MessageAppTC4.in are input test cases for **part B**.
MessageAppTC1.out, MessageAppTC2.out, MessageAppTC3.out and MessageAppTC4.out are the expected output for the respective test cases.
After you have compiled your program, to test it with say MessageAppTC1.in, you type:

```
java MessageApp < MessageAppTC1.in
```

Please complete part A before starting on part B !

## Problem Description - Part B

In addition to the message operation, user1 can also **reply** to a message sent to him by user2. The reply will be stored in both user1 and user2's message list. In addition, the message being replied to will now become the **2nd most recent** message for both user1 and user2, while the reply becomes the **most recent**. The format of the reply operation is as follows:

1. reply <user1> <user2> <message1> <message2>
   (a) *user1* and *user2* are both single word names and are guaranteed to be unique and different from each other.
   (b) *user1* is the user replying with *message1*. *message1* is a unique integer $\geq 0$.
   (c) *user2* is the user receiving *message1* in reply to his message *message2*. *message2* is guaranteed to already exist.

For example

```
reply Mark John 31 1
```

means John has previously sent message 1 to Mark and Mark now replies with message 31. Message 1 will now become the 2nd most recent message in both John and Mark's message list, while message 31 will become the most recent. **Note that the message being replied to should have its order changed before any required deletion is performed.**

### Input

Input is the same as before except now there is an additional reply operation. An example input with the reply operation is given below:

```
3        <-- 3 users
John     <-- user 1
Mark     <-- user 2
Betty    <-- user 3
3        <-- 3 operations
message John Mark 1   <-- John messages Mark with message 1
message John Betty 10 <-- John messages Betty with message 10
reply Mark John 31 1  <-- Mark reply John message 1 with message 31
```

### Output

Same as before. Example output for the given input above is shown below:

```
message John Mark 1
John 1
Mark 1
Betty @

message John Betty 10
John 1,10
Mark 1
Betty 10

reply Mark John 31 1
John 10,1,31
Mark 1,31
Betty 10
```

## Grading Scheme

1. Program correctness = 70 marks, Design = 20 marks, Programming = 10 marks
2. No marks awarded if the program does not compile.
3. Marks will be deducted if **student particulars and program description** are not filled up in the top portion of the source code.
4. There are 10 test cases. Each test case is worth 7 marks. There will be 7 test cases for part A and 3 for part B.
5. Correctness marks will be halved if LinkedList is not used, or if any other ADT is used.
6. Things to look out for under design
   (a) correct usage of programming constructs (eg use correct type for variables)
   (b) Not overly complicated logic
   (c) No redundant logic
7. Things to look out for under programming style
   (a) Meaningful comments (including pre- and post-condition description if necessary)
   (b) Proper indentation
   (c) Meaningful identifiers