Mail System Assignment – Make Up Assignment for Programming Exam 2 – Spring 2016

For this assignment, you are to implement a prototype electronic mail system. I'm calling the users of the system Agents. Agents send messages to each other through a shared MailBox. In order to send a message, Agents need to know who to send it to. As each Agent thread starts, it registers its name with a (shared) Directory object. The Directory keeps track of all Agents available. Messages can be randomly generated by the Message object, or the Agent can create its own message string. NOTE: This is not a client-server / networking application. It is a multi-threaded application.

The grade on this assignment will replace the grade you received on the programming part of Exam 2 – unless it is lower, in which case I will use the higher grade. If you choose to do this assignment, I will need to get your graded exam back from you so that I can change your grade.

You can implement the classes in any order you would like, but I strongly recommend writing a main class in each to test functionality as you go. That way you know whether parts are working correctly or not before you move on, and you can isolate bugs easier.

The MailSystem class is the one which starts all the action. Given a command line argument holding the number of agents to start, it starts each of those agents as a thread, waits for them to complete (see Agent below, for completion times), and then asks each to print out all the messages it has received. Finally, it queries the Mailbox to print out how many messages were undelivered. The API for MailSystem is shown below.

```
public class MailSystem
-----------------------------------------------------------------------
void   main(String[] args)    // Starts n agents, waits for them to
                              //   finish, and then asks each to print
                              //   out the messages it has received.
                              //   Finally, print out undelivered
                              //   messages.
```

The Directory class is responsible for maintaining a list of agents that are registered with the system (think of signing up for a gmail account). As each agent starts up, it registers its name with the Directory. Think about this class carefully. There is only one Directory in the project – do you need to instantiate it, or can you use it as a static class? When an agent is going to send a message, it needs to know the name of another agent, so it may use the getRandomAgent method to find a valid agent name. (Unlike the real world, these guys don't communicate with each other directly outside the mail system, so they don't know who's there.)

```
public class Directory
-----------------------------------------------------------------------
void   addAgent(String name) // Adds an agent to the shared directory.
String getRandomAgent()      // Returns the name of a random agent in
                             //    in the shared directory.
```

The Mailbox class represents a shared mailbox where agents can add messages or retrieve those messages that are addressed to them. When messages are retrieved by an agent, the Mailbox deletes these from the shared mailbox. Finally, the printUndelivered method allows the Mailbox to report any messages that have not been retrieved. Since some agents may stop running before other agents, they may still be receiving messages, but these are never retrieved. As with the Directory class, there is only one Mailbox that is shared by all Agents, so you should consider carefully how to implement this.

```
public class Mailbox
-----------------------------------------------------------------------
void   addMessage(Message m) // Adds a message to the shared mailbox.
ArrayList<Message>
       retrieveMessages(String to)
                             // Retrieves all the messages for a
                             //   a particular agent. Once retrieved,
                             //   the messages are deleted.
void   printUndelivered()    //  As agents complete, they no longer
                             //   retrieve messages. This function
                             //   prints out any remaining messages.
```

The Message class defines the format of a message. Each message must contain the agent it is intended for, the agent it was sent from, and the text of the message. In the stub code, you will notice that I've put in an array of MESSAGES which you can randomly choose to send. Or you can construct a Message with your own text. (Many of my random messages will look suspiciously like those in the Magic8Server example from class. In case you were wondering where I got them.) The API for Message is shown below.

```
public class Message
-----------------------------------------------------------------------
       Message(String to, String from)
                             // Constructor for the Message. It
                             //   a message with the sender and
                             //   receiver specified, but with a
                             //   random message string.
       Message(String to, String from, String message)
                             // Constructor for the Message. It
                             //   creates a message as above, but
                             //    with a specified message.
String getTo()               // Returns the name of the agent the
                             //   message is to
String getFrom()             // Returns the name of the agent the
                             //   message is from
String getMessage()          // Returns the message contents
String toString()            // Returns a string of Message variables.
```

The last class to discuss is the Agent class. The Agent class represents the individuals sending messages to each other and retrieving messages. The constructor sets the name of the agent and chooses a random number of iterations for which the agent will run. This number should be

between 10 and 20 iterations. Its run method must first register its name in the Directory, and then iterate and randomly choose between sending a message and retrieving its messages. Once it has completed running, it will be asked to print out the messages it has retrieved, so it must store these in some format.

```
public class Agent implements Runnable
-----------------------------------------------------------------------
       Agent(String name)   // Constructor for the Agent with its name.
                            //   It randomly sets the number of
                            //   send/receive iterations it will go
                            //   through when it runs. This is a number
                            //   between 10 and 20.
void   run()                // Starts the agent. The agent first
                            //   registers with the directory. It then
                            //   makes a random choice on whether to
                            //   send or receive messages.
void   printMessages()      // Prints the list of message the agent
                            //   has retrieved from the Mailbox.
```

The program is started with a single integer command line argument that tells it how many agents to run. Once each Agent has completed its iterations, the main program goes through each Agent and asks it to print out the messages it has retrieved. Then it asks the Mailbox to print out those messages that were undelivered. As you implement this, think carefully about what objects have shared access that you might have to protect so that there are no lost update problems.

When running the program, I came up with a sample run as shown below. Notice that I've cleverly named my agents by number (which was converted to a String). Here is sample output when I ran mine with 10 agents. Note that you will not get the same output because of the randomization, but the format of your output should be the same. Don't be terribly concerned if one or more agents don't receive any messages – if you do several trial runs , they likely all will. Also don't be concerned about the undelivered messages. Once an agent quits checking its messages, it is still listed in the Directory, so other agents may still be sending to it. Finally, the messages don't really make any sense – they are not a conversation, just random messages, so again, don't get too concerned over that.

```
% java MailSystem 10

Agent 0
To: 0 From: 1 Message: Is the semester over yet?
To: 0 From: 3 Message: Yes definitely
To: 0 From: 0 Message: Is that right?
Agent 1
To: 1 From: 4 Message: Goodbye.
To: 1 From: 1 Message: Goodbye.
To: 1 From: 9 Message: Hello.
Agent 2
To: 2 From: 4 Message: Is that right?
To: 2 From: 2 Message: What time is it?
To: 2 From: 9 Message: Goodbye.
To: 2 From: 4 Message: Most likely.
```

To: 2 From: 2 Message: I'm good, how about you?
To: 2 From: 2 Message: What time is it?
Agent 3
To: 3 From: 2 Message: Hello.
To: 3 From: 2 Message: I'm good, how about you?
To: 3 From: 0 Message: How are you?
Agent 4
To: 4 From: 7 Message: Is the semester over yet?
To: 4 From: 0 Message: Most likely.
To: 4 From: 0 Message: Most likely.
To: 4 From: 9 Message: Go away.
To: 4 From: 0 Message: Most likely.
Agent 5
To: 5 From: 3 Message: Yes.
To: 5 From: 5 Message: Yes definitely
Agent 6
To: 6 From: 4 Message: How are you?
To: 6 From: 7 Message: What are you up to?
To: 6 From: 8 Message: Maybe.
To: 6 From: 3 Message: Nice day, isn't it?
To: 6 From: 0 Message: What time is it?
To: 6 From: 7 Message: What are you up to?
To: 6 From: 3 Message: Nice day, isn't it?
To: 6 From: 2 Message: Yes.
Agent 7
To: 7 From: 4 Message: Is that right?
To: 7 From: 1 Message: No.
To: 7 From: 7 Message: Are we having fun yet?
Agent 8
To: 8 From: 7 Message: Yes.
To: 8 From: 3 Message: No.
To: 8 From: 0 Message: Without a doubt
To: 8 From: 3 Message: Maybe.
To: 8 From: 9 Message: How are you?
To: 8 From: 0 Message: Without a doubt
To: 8 From: 8 Message: How are you?
To: 8 From: 4 Message: Go away.
To: 8 From: 8 Message: Yes.
Agent 9
Undelivered:
To: 6 From: 3 Message: Nice day, isn't it?
To: 0 From: 1 Message: Yes definitely
To: 3 From: 3 Message: Is the semester over yet?
To: 3 From: 7 Message: Nice day, isn't it?
To: 0 From: 7 Message: Maybe.
To: 1 From: 9 Message: I'm good, how about you?
To: 0 From: 5 Message: No.
To: 3 From: 4 Message: How are you?
To: 1 From: 8 Message: Is the semester over yet?
To: 4 From: 8 Message: How are you?
To: 0 From: 6 Message: Yes definitely
To: 7 From: 2 Message: Yes definitely
To: 1 From: 6 Message: What is your name?
To: 2 From: 6 Message: Most likely.
To: 9 From: 6 Message: Maybe.

```
To: 5 From: 6 Message: Hello.
To: 1 From: 6 Message: Nice day, isn't it?
To: 3 From: 6 Message: Is the semester over yet?
```

Good luck and have fun!