HOMEWORK - SUMMER 2016

HOMEWORK 5 – due Monday, August 15th, before 9:00pm.

REMINDERS:

- **Be sure your code follows the coding style for CSE214.**
- **Make sure you read the warnings about academic dishonesty. *Remember, all work you submit for homework or exams MUST be your own work.***
- **Click here to read through the steps to make sure you hand in your homework successfully. Also, login to your grading account and click "Submit Assignment" to upload and submit your assignment. Make sure that you submit the same files on both systems.**
- **You may use any Java API class.**
- **You may use Scanner, InputStreamReader, or any other class that you wish for keyboard input.**

The goal of this assignment is to write a program that manages an email system (like a stripped down version of Outlook) except that you can't send and receive emails (naturally). However, this program will have all of the usual email features like deleting emails, creating new emails, moving emails to different folders, adding and removing emails, and the like.

---

1. Write a fully documented class named `Email`. This class will contain the following information about each email.

- `private String to`
  The String literal which stores the "to" field.
- `private String cc`
  The String literal which stores the "cc" field.
- `private String bcc`
  The String literal which stores the "bcc" field.
- `private String subject`
  The String literal which stores the "subject" field.
- `private String body`
  The String literal which stores all of the text in the email's body.

- **`private GregorianCalendar timestamp`**
  Represents the time that this email was created.

The Email class should have a Constructor, mutator and accessor methods for each instance variable. In order for your program to be able to save Email objects to a file, all objects in it must be serializeable, so *this class must implement Serializable*.

2. Write a fully documented class named `Folder`. This class represents an email folder and will handle all of the logic for adding and removing emails. It contains the following data values:

- **`private ArrayList<Email> emails`**
  Stores all of the emails contained in this folder. (It can be any data collection, ArrayList is only an example)
- **`private String name`**
  Stores the name of the folder.
- **`private String currentSortingMethod`**
  Stores the current sorting method (however that you see fit) so that emails added can be properly sorted without having to first resort the folder. Notes: Default is date descending.

It also has the following methods (including the usual constructor and accessor/mutators methods):

- **`public void addEmail(Email email)`**
  Adds an email to the folder according to the current sorting method.
- **`public Email removeEmail(int index)`**
  Removes an email from the folder by index.
- **`public void sortBySubjectAscending()`**
  - **Sorts the emails alphabetically by subject in ascending order.**
- **`public void sortBySubjectDescending()`**
  - **Sorts the emails alphabetically by subject in descending order.**
- **`public void sortByDateAscending()`**
  - **Sorts the emails by date in ascending order.**
- **`public void sortByDateDescending()`**
  - **Sorts the emails by date in descending order.**

3. Write a fully documented class named `Mailbox`. This class represents an email box and it will contain all of the folders and the remaining logic. It contains the following data values:

- **`private Folder inbox`**
  Stores the inbox, which is a special folder which should never be allowed to be deleted or renamed. All new emails go here.
- **`private Folder trash`**
  - **Stores the trash, which is a special folder which should never be allowed to be deleted or renamed. When an email is deleted, it is moved here.**
- **`private ArrayList<Folder> folders`**

Stores all of the custom folders contained in the mailbox. (It can be any data collection, ArrayList is only an example)

- **`public static Mailbox mailbox`**
  Stores the main mailbox that will used by the application. This mailbox should be instantiated in the main method.

It also contains the following methods (and a main method):

- **`public void addFolder(Folder folder)`**
  Adds the given folder to the list of custom folders. Note: check to make sure a folder with that given name does not already exist. If it does, return an error in some manner.
- **`public void deleteFolder(String name)`**
  Removes the given folder from the list of custom folders
- **`public void composeEmail()`**
  Gets user input on the contents of the email and adds it to the inbox.
- **`public void deleteEmail(Email email)`**
  Moves the email to the trash. (This method shouldn't remove any emails from folders, the method removeEmail should be called and then deleteEmail is called on the result)
- **`public void clearTrash()`**
  Removes all emails from the trash folder.
- **`public void moveEmail(Email email, Folder target)`**
  Takes the given email and puts in in the given folder. If the folder cannot be found, instead move it to the Inbox. (Again, this method shouldn't remove any emails from folders, the method removeEmail should be called and then moveEmail is called on the result)
- **`public Folder getFolder(String name)`**
  Returns a folder by folder name.

4. When the program begins, it should see if the file "mailbox.obj" exists in the current directory. If so, it should initialize the mailbox with the data in this file using serialization. Otherwise, your program will start with an empty mailbox. Then, the program should provide the user with a menu with the following options.

A – Add folder
Prompts the user for folder name, and creates and adds a new folder with the specified name to the list of folders.

R – Remove folder
Prompts the user for folder name and removes the folder if a folder with the given name exists.

C – Compose email
Prompts the user for TO, CC, BCC, Subject, and then Body and creates a new email and adds it to the inbox.

F – View Folder
Prompts the user for folder name, if the folder is found, the folder submenu is opened for that specific folder which displays all of the emails in the folder.
I – View Inbox
Opens the folder submenu for the inbox.
T – View Trash
Opens the folder submenu for the trash.
E – Empty Trash
Empties the trash folder of all emails.
Q - Quit the program
Remember to save the contents of the mailbox to the file, "mailbox.obj".
There is no option to load or save a file in the menu. This is because you should automatically search for a file named "mailbox.obj" upon startup, and save to the file when the user wishes to quit the program.

Folder submenu:
M – Move email
Prompts the user for email index and then displays a list of folders and asks the user to input the name of the folder to move to. If the specified folder was not found, cancel the entire operation. User should be able to move to Inbox, Trash, and the current folder.
D – Delete email
Prompts the user for email index and moves the email to the trash folder.
V – View email contents
Prompts the user for email index and prints the information about the corresponding email.
SA – Sort by subject ascending
Sorts the emails by subject in ascending order.
SD – Sort by subject descending
Sorts the emails by subject in descending order.
DA – Sort by date ascending
Sorts the emails by date in ascending order.
DD – Sort by date descending
Sorts the emails by date in descending order.
R – Return to main menu

4. Include exception classes or additional classes as needed.

OUTPUT FORMAT

- **Be sure to have a menu after each operation.**
- **Display to the user any errors that seem appropriate at any point during execution of your program.**

EXTRA CREDIT (12 points)

- **Use Graphical User Interface (GUI) for program input and display the results. (12 points)**

  NOTE: The GUI must include headers (like the "Subject", "Date" headers in your average mailbox system) and these headers must be clickable to sort. Clicking a header switches from ascending to descending (and vice versa) and changes the sort parameter to whatever the header specifies. And rather than accessing emails by index, all emails should have two buttons (Delete and Move) next to it that the user uses to delete and move emails from folders.

  i.e. If the current sorting method is Date Ascending and the user clicks on the Subject header, the new sorting method is Subject Descending.

SAMPLE COMPARABLE CODE

```
import java.util.*;

/*
 * An example of type abstraction that implements Comparable
 * and Comparator interfaces.
 *
 * The Comparable/Comparator interfaces provide a standard means
 * for communication with yet unknown types of objects.
*/

public class CollectionsTester {
         public static void main(String[] args) {
           ArrayList<Employee> staff = new ArrayList<Employee>();

           staff.add(new Employee("Joe",100000, 177700010));
           staff.add(new Employee("Jane",200000, 111100010));
           staff.add(new Employee("Bob",66666, 1999000010));
           staff.add(new Employee("Andy",77777, 188800010));

           Collections.sort(staff);                                    // Sort by salary
           System.out.println("Lowest paid employee: "+staff.get(0));  // Prints Bob

           Collections.sort(staff, new NameComparator());              // Sort by aplahabetical order
           System.out.println("First employee in list: "+staff.get(0)); // Prints Andy

           Collections.sort(staff, new IdComparator());                // Sort by ID number
           System.out.println("Employee with lowest ID: "+staff.get(0)); // Prints Jane
         }
}

public class Employee implements Comparable {
        private String name;
        private int salary;
        private int id;
        public Employee(String initName, int initSal, int initId) {
                id     = initId;
                name = initName;
                salary = initSal;
        }
        public String getName(){ return name; }
        public int getSalary() { return salary; }
        public int getId(){ return id; }
        public void setSalary(int newSalary) {
                salary = newSalary;
        }
```

```
        public int compareTo(Object o) {
                Employee otherEmp = (Employee)o;
                if (this.salary == otherEmp.salary)
                        return 0;
                else if (this.salary > otherEmp.salary)
                        return 1;
                else
                        return -1;
        }
        public String toString() {
                return name + ", $" + salary + ", "+ id;
        }
}

public class NameComparator implements Comparator {
        public int compare(Object o1, Object o2) {
                Employee e1 = (Employee) o1;
                Employee e2 = (Employee) o2;
                return (e1.getName().compareTo(e2.getName()));
        }
}

public class IdComparator implements Comparator {
       public int compare(Object o1, Object o2) {
           Employee e1 = (Employee) o1;
           Employee e2 = (Employee) o2;
           if (e1.getId() == e2.getId())
               return 0;
           else if (e1.getId() > e2.getId())
               return 1;
           else
               return -1;
       }
}
```

EXAMPLES

- **ObjectInputStream.readObject():**

```
MyClass myObject; //MyClass implements Serializable
try {
  //If file is found, open it
  FileInputStream   file = new FileInputStream("mySaveFile.obj");
  ObjectInputStream fin  = new ObjectInputStream(file);
  myObject = (MyClass) fin.readObject();
  file.close();
} catch(IOException a){}
  catch(ClassNotFoundException c){} //Bottoms up!
//Note that myObject may still be null
```

- **ObjectOutpurStream.writeObject():**

```
try {
  FileOutputStream   file = new FileOutputStream("mySaveFile.obj");
  ObjectOutputStream fout = new ObjectOutputStream(file);
  fout.writeObject(myObject);
  fout.close();
} catch(IOException a) { /*Error*/ }
```

- **System.out.printf():**

```
System.out.printf("%4d %4d %4d %-40s", 1, 2, 3, "A string");
System.out.printf("%4d %4d %4d %-40s", 10, 20, 30, "A longer
string");
System.out.printf("%4d %4d %4d %-40s", 10, 100, 1000, "Very longer
than previous strings");
//output:
//   1    2    3 A string
//  10   20   30 A longer string
//  10  100 1000 Very longer than previous strings
```

SAMEPL INPUT/OUTPUT

```
Comments in green. User input in black. Program output in blue.

Previous save not found, starting with an empty mailbox.

Mailbox:
--------
Inbox
Trash

A - Add folder
R - Remove folder
C - Compose email
F - Open folder
I - Open Inbox
T - Open Trash
Q - Quit

Enter a user option: A

Enter folder name: Other Folder

Mailbox:
--------
Inbox
Trash
Other Folder

//Menu not included to save space.

Enter a user option: C

Enter recipient (To): phillip.smith@stonybrook.edu, psmith@ic.sunysb.edu

Enter carbon copy recipients (CC): psmith@cs.sunysb.edu

Enter blind carbon copy recipients (BCC): phroph@yahoo.com

Enter subject line: Testing

Enter body: This is a test of the mailbox program!

Email successfully added to Inbox.

Mailbox
--------
Inbox
Trash
Other Folder
```

```
//Menu not included to save space.

Enter a user option: I

Inbox

Index |        Time       | Subject
-----------------------------------
  1   |  12:38PM 4/8/2012 | Testing

M - Move email
D - Delete email
V - View email contents
SA - Sort by subject line in ascending order
SD - Sort by subject line in descending order
DA - Sort by date in ascending order
DD - Sort by date in descending order
R - Return to mailbox

Enter a user option: M

Enter the index of the email to move: 1

Folders:
Inbox
Trash
Other Folder

Select a folder to move "Testing" to: Other Folder

"Testing" successfully moved to Other Folder.

Inbox

Inbox is empty.

//Menu not included to save space.

Enter a user option: R

Mailbox:
--------
Inbox
Trash
Other Folder

//Menu not included to save space.

Enter a user option: F

Enter folder name: Other Folder

Other Folder

Index |        Time       | Subject
-----------------------------------
  1   |  12:38PM 4/8/2012 | Testing

//Menu not included to save space.

Enter a user option: V

Enter email index: 1
```

```
To: phillip.smith@stonybrook.edu, psmith@ic.sunysb.edu
CC: psmith@cs.sunysb.edu
BCC: phroph@yahoo.com
Subject: Testing
This is a test of the mailbox program!

Other Folder

Index |        Time       | Subject
-----------------------------------
  1   |  12:38PM 4/8/2012 | Testing

//Menu not included to save space.

Enter a user option: D

Enter email index: 1

"Testing" has successfully been moved to the trash.

Other Folder

Other Folder is empty.

//Menu not included to save space.

Enter a user option: R

Mailbox
--------
Inbox
Trash
Other Folder

//Menu not included to save space.

Enter a user option: E

1 item(s) successfully deleted.

Mailbox
--------
Inbox
Trash
Other Folder

//Menu not included to save space.

Enter a user option: Q

Program successfully exited and mailbox saved.
```