CSE214

HOMEWORK - SPRING 2018

HOMEWORK 6 - due Tuesday, April 17th no later than 7:00pm

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 assignments MUST be entirely your own work. Also, group efforts are not allowed.***
- **Login to your grading account and click "Submit Assignment" to upload and submit your assignment.**
- **You are allowed to use any built-in Java API Data Structure classes to implement this assignment except where noted.**
- **You may use Scanner, InputStreamReader, or any other class that you wish for keyboard input.**

With the global warming crisis in full swing, it's getting harder and harder to come up with storm names, and even harder to keep track of the details. Because of this, the U.S. Government itself has hired you to make a storm database that can help keep track of all the various storms. The government would like the database users at NOAA to be able to look up any storm by name to view its details, and also to be able to view a list of the storms sorted by windspeed or precipitation amount. Since the U.S. Government has only the best technology, we must plan for the windows XP running computers to be restarted occasionally, which means we must be able to save our database to a file, which we can then use to reload the database on the next run of the program. In this homework, one of the "learning objectives" is to realize how much of the code you *don't* have to write. We will be using the built-in hash map class, serialization for saving, and we will use Collections.sort() to sort the list of hurricanes based on the temperature. **NOTE**: All exceptions explicitly thrown in Required Classes except for IllegalArgumentException are custom exceptions that need to be made by you.

Required Classes

Storm

Write a full-documented class named Storm. As always, each data field must have getters and setters. *Must implement the Serializable interface.*

- private string name
- private double precipitation
- private double windspeed
- String Date - must be formatted Internet-Style YYYY-MM-DD

WindSpeedComparator
Write a fully-documented class named WindspeedComparator that allows us to compare two storms on the basis of the windspeed. The class should implement the Comparator interface and override the compare method.

- int compare (Storm left, Storm right)
  - Should return: -1 if the left windspeed is “less” than the right wind speed, 0 if they are equal, and 1 otherwise.

PrecipitationComparator
Write a fully-documented class named Precipitation Comparator that allows us to compare two storms based on the amount of precipitation. The PrecipitationComparator class should implement the Comparator interface and override the compare method.

- int compare (Storm left, Storm right)
  - Should return: -1 if the left is less than the right, 0 if they are equal, and 1 if the left’s otherwise.

StormStatServer
Write a full-documented class named StormStatServer that will allow the user to interact with a database of Storms. Provide the user with a menu-based interface that allows them to add, remove, and edit storms. You will need to be able to serialize (save) the database at the end of the program and deserialize (load) the database if a file containing the database already exists. The database will have storm name as the key and the associated Storm object as the value. Names should be converted to uppercase.

On startup, the StormStatServer should check to see if the file hurricane.ser exists in the current directory. If it does, then the file should be loaded and deserialized into the database instance. If the file does not exist, a new HashMap should be created and used instead. In either case, the user should be allowed to fully interact with the StormStatServer without issues.

- private HashMap<String, Storm> database
- Public static void main(String[] args)

Serializable Interface

In this homework, you will also work with the idea of persistence. This means that our program should save all data from session to session. When we terminate a program, normally the data will be lost. We will preserve this data by using Serializable Java API and binary object files. All your classes should simply implement the java.io.Serializable interface (with the exception of the LunarSystem class)

Example: (Note - class names here intentionally are different than the homework description above)
To serialize an object, you need to create an ObjectOutputStream to send the data to, and then use the writeObject method to send the data to the stream, which is stored in the specified file. In the following code snippet, we have an instance of the StorageTable class that implements the Serializable Interface.

```
StorageTable storage = new StorageTable(/*Constructor Parameters*/);
// missing code here adds Storage objects to the table.
```

```
FileOutputStream file = new FileOutputStream("storage.obj");
ObjectOutputStream outStream = new ObjectOutputStream(file);
// the following line will save the object in the file
outStream.writeObject(storage);
outStream.close();
```

When the same application (or another application) runs again, you can initialize the member using the serialized data saved from step 1 so you don't have to recreate the object from scratch. To do this, you need to create an ObjectInputStream to read the data from, and then use the readObject method to read the hash from the stream:

```
FileInputStream file = new FileInputStream("storage.obj");
ObjectInputStream inStream = new ObjectInputStream(file);
StorageTable storage;
storage = (StorageTable) inStream.readObject();
inStream.close();
// missing code here can use StorageTable constructed previously
```

Note: If you change any data fields or structure of the serialized class, old saved objects will become incompatible.

Comparable Interface

```
    /*
     * An example of type abstraction that implements the Comparator
interface.
     *
     * The Comparator interfaces allows you to compare two different
objects
     * (as opposed to another object with yourself).
     */
     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));
             System.out.println("Lowest paid employee: "+staff.get(0));
 // Prints Bob
             Collections.sort(staff, new NameComparator());
 // Sort by alphabetical 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 {
         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 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;
        }
    }
```

Note on Exceptions: all exceptions should be handled gracefully - they should be caught in the main, and the user should be notified by a nice printout. Your messages should clearly indicate what the problem is (bad index, full list, negative number, etc.). The program should continue to run normally after an exception is encountered. We will not be checking Input Mismatch cases.

Pretty Printing: Here is a tutorial on how to print tables neatly using printf in java. It is highly encouraged that you print the output neatly, as it makes grading much easier. https://docs.oracle.com/javase/tutorial/java/data/numberformat.html

General Recommendations

You can feel free to add extra methods and variables if you need.

UI Required Functions

- A-Add A Storm
- L-Look Up A Storm
- D-Remove A Storm
- E-Edit A Storm
- R-Print Storms by Rainfall
- W-Print Storms (Sorted by windspeed)
- X-Save and quit
- Q-Quit without saving state (and delete any save-file)

Note: please make sure that the menu is NOT case sensitive (so selecting A should be the same as selecting a).

Program Sample

```
Welcome to the StormStatServer, we may not be able to make it rain, but
we sure can tell you when it happened!

No previous data found.

Menu:
    A) Add A Storm
    L) Look Up A Storm
    D) Delete A Storm
    E) Edit Storm Data
    R) Print Storms Sorted By Rainfall
    W) W-Print Storms by Windspeed
    X) Save and quit
    Q) Quit and delete saved data



Please select an option: A
Please enter name: SANDY
Please enter date: 2012-10-12
Please enter precipitation (cm): 114.3
Please enter windspeed (km/h): 132
SANDY added.

Please select an option: A
Please enter name: DAN
Please enter date: 2018-2-13
Please enter precipitation (cm): 215.0
Please enter windspeed (km/h): 22.0
DAN added.

Please select an option: A
Please enter name: JACK
Please enter date: 2015-4-12
Please enter precipitation (cm): 214.0
Please enter windspeed (km/h): 43
JACK added.

Please select an option: A
Please enter name: JOHNNY
Please enter date: 2017-6-12
Please enter precipitation (cm): 220.0
Please enter windspeed (km/h): 48
JOHNNY added.

Please select an option: A
Please enter name: THOMAS
Please enter date: 2015-6-10
Please enter precipitation (cm): 219.3
Please enter windspeed (km/h): 97
THOMAS added.

Please select an option: A
Please enter name: JUAN
Please enter date: 2017-6-12
Please enter precipitation (cm): 307
Please enter windspeed (km/h): 56.8
JUAN added.

Please select an option: W

Name                    Windspeed      Rainfall
--------------------------------------------
DAN                     22.0           215.0
JACK                    43.0           214.0
```

```
JOHNNY                 48.0           220.0
JUAN                   56.8           307.0
THOMAS                 97.0           219.3
SANDY                  132.0          114.3

Please select an option: X
File saved to hurricane.ser; feel free to use the weather channel in the
meantime.
//A tired, overworked, bored grading TA begrudgingly restarts the
program

Welcome to the StormStatServer, we may not be able to make it rain, but
we sure can tell you when it happened!

hurricane.ser was found and loaded.

Menu:
    A) Add A Storm
    L) Look Up A Storm
    D) Delete A Storm
    E) Edit Storm Data
    R) Print Storms Sorted By Rainfall
    W) W-Print Storms by Windspeed
    X) Save and quit
    Q) Quit and delete saved data

Please select an option: L
Please enter name: JACK

Storm JACK: Date 2015-4-12, 43 km/h winds, 214.0 cm of rain

Please select an option: D
Please enter name: SANDY
Storm SANDY has been deleted.

Please select an option: E
Please enter name: ESMAILI
Key not found.

Please select an option: E
Please enter name: DAN
In Edit Mode, press enter without any input to leave data unchanged.
Please enter date: Feb. 13 1996
Invalid date format.

Please select an option: E
Please enter name: DAN
In Edit Mode, press enter without any input to leave data unchanged.
Please enter date:
Please enter precipitation (cm): 373
Please enter windspeed (km/h):
DAN added.

Please enter name: SANDY
Storm SANDY has been deleted.

Please select an option: W

Name                   Windspeed      Rainfall
-------------------------------------------
JACK                   43.0           214.0
THOMAS                 97.0           219.3
JOHNNY                 48.0           220.0
JUAN                   56.8           307.0
DAN                    22.0           373.0
```

```
Please select an option: Q
Goodbye, it's hard to hold an (electric) candle in the cold November
(and April!) rain!.
```

Extra Credit

You will get up to 8 points extra credit for creating a JavaFX GUI, and up to 12 points for creating an Android App, 15 points for JSP/HTML/CSS. You must make a table with clickable headers for sorting.

Course Info | Schedule | Sections | Announcements | Homework | Exams | Help/FAQ | Grades | HOME