CSE214

HOMEWORK 6- Spring 2023

HOMEWORK 6 - due Thursday, April 18th no later than 7:00PM

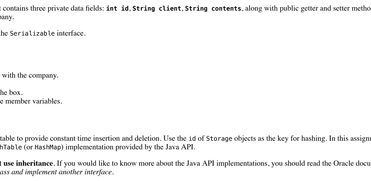
REMINDERS:

- **Submit your files only via CodeGrade in the Content page on Brightspace. Access CodeGrade by clicking "HW6 - CODEGRADE SUBMISSION LINK" for your submission.**
- **Use of a package is optional. If you wish to use it, make sure to name it "hw6" (all in lower case). Otherwise, you will lose points.**
- **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.***
- **Login to your grading account and click "Submit Assignment" to upload and submit your assignment.**
- **You may use (and are encouraged to use) any Java API Data Structures you like to implement this assignment.**
- **You may use Scanner, InputStreamReader, or any other class that you wish for keyboard input.**

When working with data, we must choose the best data structure to manipulate the data set in order to achieve efficient results. A very basic array can store information without a problem, but is inefficient when we are constantly storing and searching for data. The idea of a table is created to increase the efficiency and avoid such problems. Tables store data using a tuple &ltKey, Value&gt. We choose a special field of an object, and use it to uniquely identify the entire object. Therefore, when we store and search for an object, we will use the key to save or search for the object. Ideally, this will reduce the algorithm to O(1). For this assignment, you are allowed to use a HashTable, or a HashMap.

For this assignment, we will be modeling self storage boxes! In the Rocky Stream self storage, there are a lot of boxes, many of which aren't always full (*this isn't a very successful business*). Therefore, it is more efficient to store information only about the occupied boxes. Each box has an ID, and there may be an owner and a description of the contents.

Approximate UML Diagram



Required Classes

The following sections describe classes which are required for this assignment. Each section provides a description and the specifications necessary to complete each class. If you feel that additional methods would be useful, feel free to add them during your implementation as you see fit. However, all the variables and methods in the following specifications must be included in your project.

NOTE: All classes listed should implement the Serializable interface, except for StorageManager.

1. Storage

Write a fully documented class named Storage that contains three private data fields: **`int id`**,**`String client`**,**`String contents`**, along with public getter and setter methods for each of these fields. This class will be used to represent a storage box registered with the company.

As mentioned above, this class should implement the `Serializable` interface.

- **`private int id`**
  - The unique ID of the storage box.
- **`private String client`**
  - The name of the client storing the box with the company.
- **`private String contents`**
  - A brief description of the contents of the box.
- Getter and Setter methods for the above three member variables.

2. StorageTable

The database of `Storages` will be stored in a hash table to provide constant time insertion and deletion. Use the `id` of `Storage` objects as the key for hashing. In this assignment, you may provide your own implementation for the StorageTable class, or you may use the `HashTable` (or `HashMap`) implementation provided by the Java API.

Note: If you utilize the Java API classes, **you must use inheritance**. If you would like to know more about the Java API implementations, you should read the Oracle documentation for java.util.Hashtable and java.util.HashMap. *Hint: a class can extend one class and implement another interface*.

Just as the above classes, this class should implement the `Serializable` interface.

- **`public void putStorage(int storageId, Storage storage) throws IllegalArgumentException`**
  - *Brief:*
    - Manually inserts a `Storage` object into the table using the specified key.
  - *Parameters:*
    - storageId
      - The unique key for the `Storage` object.
    - Storage
      - The `Storage` object to insert into the table.
  - *Preconditions:*
    - `storageId` ≥ `0` and does not already exist in the table.
    - `Storage` ≠ `null`
  - *Postconditions:*
    - The `Storage` has been inserted into the table with the indicated key.
  - *Throws:*
    - IllegalArgumentException: If any of the preconditions is not met.
- **`public Storage getStorage(int storageID)`**
  - *Brief:*
    - Retrieve the `Storage` from the table having the indicated `storageID`. If the requested storageID does not exist in the StorageTable, return null.
  - *Parameters:*
    - storageID
      - Key of the `Storage` to retrieve from the table.
  - *Returns:*
    - A `Storage` object with the given key, **null otherwise.**

3. StorageManager

Write a fully-documented class named StorageManager. This class will allow the user to interact with the storage database by listing the storage boxes occupied, allowing the user to add or remove storage boxes, searching for a box by id, and listing all the boxes for a user. In addition, the class should provide the functionality to load a saved (serialized) StorageTable or create a new one if a saved table does not exist.

On startup, the StorageManager should check to see if the file storage.obj exists in the current directory. If it does, then the file should be loaded and deserialized into a StorageTable. If the file does not exist, an empty StorageTable object should be created and used instead. In either case, the user should be allowed to fully interact with the storage table, inserting, removing, selecting, and reading entries.

When the user enters 'Q' to quit the program, the storage table should be serialized to the file storage.obj. That way, the next time the program is run, the storages will remain in the database and allow different users to manipulate the storage records. If you would like to 'reset' the storage table, use the "X" command to delete the file, if it exists, when the program quits (you must first check if the file exists, delete it if it does; and do not serialize the current `StorageTable` upon exit).

This class contains:

- **private static StorageTable**:
  - This is the hash table of for storing the storage objects.
- **public static void main(String[] args)**
  - *Brief:*
    - Implement the following menu options:
      - **P - Print all storage boxes**
      - **A - Insert into storage box**
      - **R - Remove contents from a storage box**
      - **C - Select all boxes owned by a particular client**
      - **F - Find a box by ID and display its owner and contents**
      - **Q - Quit and save workspace**
      - **X - Quit and delete workspace**

Serializable Interface

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.Serializartion interface.

Example: Your `StorageTable` class contains information for all `Storage` objects saved in the electronic database. You would want to preserve this data, so you can load this data the next time you run your program. You would do the following:

1. Modify the `StorageTable` so that it implements the `Serializable` interface. Also, the `Storage` class should also make this implementation. No other changes are necessary.

   ```
   public class StorageTable implements Serializable
   {
       // Member methods as is
   }
   ```

2. In your application that contains the `StorageTabe`, you can include code that will save that class's data into a file so it can be read in again later. To do this, you need to create an `ObjectOutputStream` to send the data to, and then use the writeObject method to send the hash to the stream, which is stored in the specified file.

   ```
   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();
   ```

3. When the same application (or another application) runs again, you can initialize the member using the serialized data saved from step 2 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
   ```

Sample Input/Output:

```
Hello, and welcome to Rocky Stream Storage Manager

P - Print all storage boxes
A - Insert into storage box
R - Remove contents from a storage box
C - Select all boxes owned by a particular client
F - Find a box by ID and display its owner and contents
Q - Quit and save workspace
X - Quit and delete workspace

Please select an option: A
Please enter id: 1
Please enter client: SBU CS Department
Please Enter Contents: 2006 Dell Workstations

Storage 1 set!

//Options not shown in Sample IO
Please select an option: A
Please enter id: 4
Please enter client: 214 TAs
Please Enter Contents: Unreturned Exams

Storage 4 set!

//Options not shown in Sample IO
Please select an option: A
Please enter id: 12
Please enter client: 214 TAs
Please Enter Contents: Spare Null Pointer Exceptions

Storage 12 set!

//Options not shown in Sample IO
Please select an option: A
Please enter id: 11
Please enter client: SBU CS Department
Please Enter Contents: Dashed Hopes and Dreams

Storage 11 set!

//Options not shown in Sample IO
Please select an option: P

Box#         Contents                          Owner
-------------------------------------------------------------
1            2006 Dell Workstations            SBU CS Department
4            Unreturned Exams                  214 TAs
11           Dashed Hopes and Dreams           SBU CS Department
12           Spare Null Pointer Exceptions     214 TAs

//Options not shown in Sample IO
Please select an option: C
Please enter the name of the client: SBU CS Department

Box#         Contents                          Owner
-------------------------------------------------------------
1            2006 Dell Workstations            SBU CS Department
11           Dashed Hopes and Dreams           SBU CS Department

//Options not shown in Sample IO
Please select an option: F
Please enter ID: 12
Box 12
Contents: Spare Null Pointer Exceptions
Owner: 214 TAs

//Options not shown in Sample IO
Please select an option: R
Please enter ID: 4
Box 4 is now removed.

//Options not shown in Sample IO
Please select an option: P

Box#         Contents                          Owner
-------------------------------------------------------------
1            2006 Dell Workstations            SBU CS Department
11           Dashed Hopes and Dreams           SBU CS Department
12           Spare Null Pointer Exceptions     214 TAs

//Options not shown in Sample IO
Please select an option: Q
Storage Manager is quitting, current storage is saved for next session.

//The Grading TA Restarts the program, rubbing his/her eyes and wondering how soon he/she can get to sleep

Hello, and welcome to Rocky Stream Storage Manager

P - Print all storage boxes
A - Insert into storage box
R - Remove contents from a storage box
C - Select all boxes owned by a particular client
F - Find a box by ID and display its owner and contents
Q - Quit and save workspace
X - Quit and delete workspace



Please select an option: P

Box#         Contents                          Owner
-------------------------------------------------------------
1            2006 Dell Workstations            SBU CS Department
11           Dashed Hopes and Dreams           SBU CS Department
12           Spare Null Pointer Exceptions     214 TAs

//Options not shown in Sample IO
Please select an option: A
Please enter id: 7
Please enter client: Shady Guy in South P
Please Enter Contents: Lots of Hash (tables)

Storage 7 set!

//Options not shown in Sample IO
Please select an option: P

Box#         Contents                          Owner
-------------------------------------------------------------
1            2006 Dell Workstations            SBU CS Department
7            Lots of Hash (tables)             Shady Guy in South P
11           Dashed Hopes and Dreams           SBU CS Department
12           Spare Null Pointer Exceptions     214 TAs

Please select an option: X
Storage Manager is quitting, all data is being erased. //#kbai
```

// Comment in green, input in red, output in black