Seneca College

Oct 29, 2016

Applied Arts & Technology Fall/16

SCHOOL OF COMPUTER STUDIES

JAC444 **Due date: Nov 25**

Assignments 2 & 3

1. Goals

The goals of these assignments are: (1) develop, and implement Java classes for a business application using specific data structures and algorithms; (2) write code for testing the soundness and completeness of your solution.

2. Tasks

Your project is to design and develop an inventory system for keeping track of mobile devices rented from labs in a college.

We start with the assumptions: (1) labs have unique names and, (2) a lab can stockpile a variable number of mobile devices. Furthermore, a unique name and a value tag (an integer with values between -100 and 100) define a mobile device type. The same device type can be stored in more than one lab.

1. A lab receives a rent request for a device and a period of time
2. The period is defined by two dates: the request date and the due date.
3. If the device is part of lab's inventory and the device is available, then the device is rented.
4. If the device is not part of lab's inventory or the device is already rented, the request is sent to another lab. If the request cannot be satisfied, then it is dropped.
5. There are no data structures such as queues, for keeping track of the requests. However, the inventory system works based on the first-come, first-served principle.

A template of the required classes is given to you. Some methods are complete, some are partially implemented and some have only the signature defined. You can find the project template on your online course web page. It is the last course item called *Assessments*. On the assessment section scroll down for *Assignment 2* link.

For example, the class `MobileDevice` has been defined as:

```
class MobileDevice {

String       deviceName;  // the device name
int          valueTag;    // an integer between -100 and 100
Lab          lab;         // the lab having this device it its inventory
RentSettings rs;          // rent settings
 ...

  // inner class
  private class RentSettings {
       private String  rentDate;          // date when the item is requested
       private String  dueDate;           // date when the item must be returned
       private boolean borrowed = false;  // true if the item is rented
    ...
  }
}
```

The most important method of `MobileDevice` class is

```
public boolean rentDevice(String rentDate, String dueDate, Lab lab) { ... }
```

The method takes three arguments: two strings dates for the renting period and a lab from where the device is rented. If dates are not valid it creates `DateFormatException` and returns false. If `rentDate` > `dueDate` throws `RentPeriodException` and returns false.
If no exceptions occur a `RentSettings` object must be created for the mobile device that is rented.

The `DateFormatException` and `RentPeriodException` are user-defined exceptions you have to create and they are part of project template.

The class `Lab` has two fields: a lab name and a collection of devices stored in the lab (data structure `Vector`).

```
public class Lab implements MaxTagValue {

    String               labName;  // lab name
    Vector<MobileDevice> devices;  // data structure that keeps all devices

       ...
}

public interface MaxTagValue {

    /**
     * The method returns an integer.
     * The integer is the greatest value of all tagValues of the lab devices
     */
    int findMaximumValueTag();
}
```

The most important method of `Lab` class is
`public boolean rentRequest(MobileDevice wanted, String requestDate, String dueDate)`

The method takes as arguments:
-The mobile device that needs to be rented
-The date when the device can be rented
-The date when the device must be returned to the lab
The method returns true, if the device can be rented from this lab, and false otherwise

The class `Labs` creates all the `Lab` objects and stores the devices in each lab.

```
public class Labs {

    public Lab[] labs;        // an array that stores all labs
    public int numberOfLabs;  // number of labs in collection

    public Labs(int numberOfLabs) {
        this.numberOfLabs = numberOfLabs;
        labs = new Lab[numberOfLabs];
    }
       ...
}
```

The devices definitions are read from a file with the following structure: *device name, tag value*.
For example a file named `Newnham.txt` contains lines that define four devices

```
Android,53
Android,25
Blackberry2,-95
iPhone,45
```

A lab object can be built from a file with the method.
`public Lab buildLabFromFile(String labName, String fileName)`

The most important method of `Labs` class is
`public Lab rentDeviceAvailable(MobileDevice md, String requestDate, String dueDate) {`

The method takes as arguments the mobile device that needs to be rented with the dates. It searches all labs for a device that can be rented. It returns the `Lab` object where the device is available, if any. If all devices are rented returns `null.`

There are other auxiliary classes that will help you to build the required functionality. For example the class `Helper` contains static methods for date validation.

```
public class Helper {

    public static boolean isValidDate(String date) {
        boolean valid = true;

        DateFormat formatter = new SimpleDateFormat("MM/dd/yyyy");
        formatter.setLenient(false);
        try {
            formatter.parse(date);
        } catch (ParseException e) {
            valid = false;
        }
        return valid;
    }
```

The important classes of your project must have method such as: `toString(), equals(), hashCode().` All classes and methods must have *javadoc documentation*. The important classes are defined on codeboard.io template.

In order to build and run a tailored personal project you solution must have a number of labs equal to your last digit of your student ID. If this number is less than 2, than you have to have at least 2 labs. For instance, if your student ID is *354874345* than your project must create 5 labs (two labs are given to you `Seneca@York` and `Newnham,` so you have to create additional 3 labs).
Each lab you create must contain a number of devices equal to the second last digit of your student ID. If this number is less than 4, you have to have at least 4 devices in each lab. You can work with as many devices as you like.

Several implementation clarifications:

1. You can assume that files are properly formatted and any line contains *device name, tag value*. You do not have to validate, but you can skip the lines that are no correctly structured.
2. The device is defined by a name and a tag value. The tag that is unique for a device with random values between -100 and 100. You can assume that a device can appear more than once in a lab.
3. When you develop your solution be sure you DO NOT CHANGE the signatures of the given methods. However, you can add new methods to enforce the behaviour defined by the project requirements.

To demonstrate your solution you have to write code to resolve the following tasks:

TASK 1
- build labs from files - at least two labs
- print lab and the devices from it

The device must be printed with the following format:
(device name, value tag) - if the device is not assigned to a lab, and
(device name, value tag => lab name) RentSettings (rent date, due date, lab name, borrowed) - if the device belongs to a lab.

TASK 2
- ask for a device that is not in any lab inventory. For example, ask about a device that has the name “Unknown”

TASK 3 - ask for a device that is in a lab inventory
- issue a rent request and print the device object

- issue the same rent request and print the device object
- return the device
- issue the rent request with new dates and print the device object

TASK 4 - ask for the same device in all labs
- look for the same device in all labs and return all the labs where the device is in the lab inventory
- look for the same device in all labs and return all the labs where the device is available to be rented.

TASK 5
- calculate the greatest value tag of all the devices from a lab

TASK 6
- inquire about a device: is it rented?, is it overdue?, could it be found in more than one lab?, when can it be rented?

3. Submission Requirements – two mandatory procedures

Procedure 1.

Zip only the Java files. The zip file should be named after your last name followed by the first 3 digits of your student ID. For example, if your last name is *Savage* and your ID is *354874345* then the file should be called *Savage354.zip* Upload the zip file to Blackboard

Procedure 2.
Cut and paste your code with documentation on your course page on codeboard.io project called *Assignment 2*.
Remove the package name for each Java file (the first statement).
The code must completely compile and run on *codeboard.io*

4. Marking Scheme

Every class must be properly documented using *javadoc* style created in a package having your Last Name followed by the first 3 digits of your student ID.
(remove the package statement from the code uploaded to *codeboard.io*)
The core classes must implement `toString, equals,` and `hashCode`
Follow strictly Java code conventions.

Note: tasks from 1 to 3 will be marked as assignment 2, and tasks from 4 to 6 will be marked as assignment 3.
However, since the submission date and content for both assignments are the same you will receive a single mark out of 20 marks, instead of one out of 10 from assignment 2 and one out of 10 from assignment 3.

5. Bonus

Each one of these tasks will be worth extra marks:

TASK 7:
- If a device is rented in all labs, find the lab that will have the closest device available to the requested date.

TASK 8:
- Instead of using `Vector<MobileDevice>` in class `Lab` use `LinkedList` from your first assignment. The new `LinkedList` must be transformed into generic class `LinkedList<T>` and instantiated as `LinkedList<MobileDevice>`

Here is an example of a potential output (your output could look different, but must solved the given tasks):
** TASK 1 **
Lab = Seneca@York
[
(Android, 53 => Seneca@York)
(Android, 25 => Seneca@York)
(Blackberry2, -95 => Seneca@York)
(iPhone, 45 => Seneca@York)
]
Lab = Newnham
[
(Android, 53 => Newnham)
(Blackberry, -55 => Newnham)
(iPhone, 45 => Newnham)
(Tablet, -20 => Newnham)
(Android, 25 => Newnham)
(iPhone2, 65 => Newnham)
(Blackberry10, 20 => Newnham)
(Android4, 65 => Newnham)
]

** TASK 2 **
Device (Android20, 20) does not exist!

** TASK 3 **
Device (Android, 25) is availble at 10/30/2016 from lab: Seneca@York
wanted = (Android, 25 => Seneca@York) RentSettings{rentDate='10/30/2016', dueDate='11/10/2016'Seneca@York, borrowed=true}
Device (Android, 25 => Seneca@York) RentSettings{rentDate='10/30/2016', dueDate='11/10/2016'Seneca@York, borrowed=true} is not availble for 10/30/2016
Device (Android, 25) is availble at 11/30/2016 from lab: Seneca@York
wanted = (Android, 25 => Seneca@York) RentSettings{rentDate='11/30/2016', dueDate='12/10/2016'Seneca@York, borrowed=true}

** TASK 4 **
Device (Android, 25) is availble at 10/30/2016 from lab: Newnham
wanted = (Android, 25 => Newnham) RentSettings{rentDate='10/30/2016', dueDate='12/10/2016'Newnham, borrowed=true}
Device (Android, 25 => Newnham) RentSettings{rentDate='10/30/2016', dueDate='12/10/2016'Newnham, borrowed=true} is not availble for 10/30/2016
Device (Android, 25) is availble at 10/30/2016 from lab: Newnham
wanted = (Android, 25 => Newnham) RentSettings{rentDate='10/30/2016', dueDate='12/10/2016'Newnham, borrowed=true}

** TASK 5 * ...*