Version Number:
1.0 (Oct. 13, 2019)

Assignment 2

Appointment Scheduler, Part 2

To be submitted online not later than Monday, October 28th, 2019, 11:59 p.m.

Description:

In this lab you'll continue to add new features to the appointment scheduler. Along the way, you'll demonstrate your understanding of the following course learning requirements (CLRs), as stated in the *CST8284—Object Oriented Programming (Java)* course outline:

1. Write java programming code to solve a problem, based on the problem context, including UML diagrams, using object-oriented techniques (CLR II)
2. Use basic data structures (CLR III), including implementing arrays of primitive and reference types
3. Implement inheritance and polymorphism (CLR IV)
4. Use ArrayLists to manage objects (CLR V)
5. Implement program Input/Output operations by storing objects to a file using serialization (CLR VII)
6. Produce code that has been tested and executes reliably (CLR VIII)
7. Debug program problems using manual methods and computerized tools in an appropriate manner. (CLR X)
8. Identify appropriate strategies for solving a problem (CLR XI)

Worth
6%
of your total mark

Assignment 2

Appointment Scheduler, Part 2

Program Description

In this assignment you'll add additional features to the Appointment Scheduler you started in Assignment 1. For those students who failed to submit that assignment, or for those whose submission was defective in whole or part, you may use my copy of Assignment 1, posted on Brightspace, as the starting point for your Assignment 2 submission. (Even if you're satisfied with your Assignment 1 code, you might want to look over the sample code anyway, to see how you might have implemented things differently.) Whatever your reasons, you may use the Assignment 1 code in whole or in part, without citation, in building your Assignment 2.

I. Load the Assignment2 project and copy your existing classes to the new Project

a) Download the CST8284_Assignment2.zip file and unzip it in Eclipse, just as you would one of your labs. To the `CST8284_19F_Assignment2` folder, copy the `cst8284.asgmt1.scheduler` package from Assignment1, including all its classes, to the new project., Refactor the name of the old package to `cst8284.asgmt2.scheduler`. Then, following the UML diagram below, make the modifications indicated in this document.

b) The UML diagram for this assignment has been redacted to reflect the changes in Assignment 2. This may affect some of the declarations for methods you created in Assignment 1, which will need to be refactored for this assignment. So check the UML in this document carefully for changes in the declarations, access modifiers, etc.

c) Before making the changes to your code specified in Section II, note that the same 'rules' apply to this assignment as the last, briefly stated as:

1. Follow the UML diagram *exactly as it is written*, according to the most up-to-date version of this document. You cannot add members that are not written in the UML diagram, nor can you neglect any of them either, *except where indicated in this document.*
2. All new methods indicated in the UML are intended to be used (again, except for some getters and setters). Take the UML as your guide not just of what needs to be coded, but of how your code is to be connected in a well-written, optimized application.
3. Employ best practices at all times, especially code reuse.

II. Add the following new features to your Appointment Scheduler

a) Replace every array with an ArrayList

Remove the `appointments` array and replace it with an ArrayList of type `Appointment` (along with any other arrays you may have used in creating your application in Assignment 1). Then change all the code used by the `appointments` array and replace it with appropriate ArrayList methods instead. Note that `saveAppointmentToArray()` should be renamed to `saveAppointment()`.

Remove `getNextAptIndex()` and `getAptIndex()`, along with the underlying `aptIndex` field from your code; ArrayList automatically adds new `Appointment`s to the end of its list, hence we don't need to keep track of this location anymore. Correct any other methods that were impacted by these deletions (note that ArrayList's `size()` method returns the same information as `aptIndex`).

In `findAppointment()`, replace the existing loop with an *enhanced for* loop, assuming you aren't already using an *enhanced for* in this code.

b) Add new methods to the Scheduler

Start by adding a `deleteAppointment()` method, designed to allow the user to remove an

`Appointment` based on its `Calendar` date and time. (See sample output below for details.) Note that you should use existing methods for this purpose as much as possible, along with any ArrayList methods you find appropriate, rather than rewriting the same code over again. If you're practicing good code reuse, this method can be implemented in less than a dozen lines of code.

This new feature should be added as the second option in the menu. If you implemented the fixed constants correctly in Assignment 1, adding this feature should not result in any major rewrites to your existing menu or code, just a renumbering of the named (fixed) constants, a new line in the menu, and a new `case` in the `switch` statement, one for each new feature added.

Similarly, add a `changeAppointment()` method that allows the user to change the date and time of an existing appointment. Use the current appointment date and time to first locate the Appointment object, and then edit the time and date of that appointment to their new values. Again, this should take about a dozen lines of code to execute using the methods already in existence. This operation should be made accessible as the third item in the menu.

It should be noted that while it is possible to combine the above two methods into one method—they require much of the same information, and only differ in their overall effect—the improvements in code and efficiency are minimal, and this is more than compensated for by the lack of clarity that results. So while code reuse is generally the rule of the day, this should be avoided in situations where the end result produces code that is more, rather than less, confusing. This is one such case.

In additional, you'll need to add two other new methods to your program, `saveAppointmentsToFile()` and `loadAppointmentsFromFile()`, also accessible via the appropriate menu items. See part (c) below for details of their operation.

c) Add File IO to the existing code

Using the hybrid lectures and notes as your guide, add file I/O functionality to your application. You'll need to create two new methods in Scheduler, one to save the appointments ArrayList to a file, the other to retrieve this information. Details are provided below.

saveAppointmentsToFile()

In this method. load each `Appointment` object in the ArrayList into a file called `CurrentAppointments.apts`. DO NOT hardcode the file location to a particular subdirectory, as your code will not work correctly when it is transferred to another platform. Also, do not prompt the user for a filename; the `CurrentAppointments.apts` file is to be used internally by your program. Simply use the default directory associated with your project (which is typically the `src` folder, or one of its subdirectories) to store the file.

loadAppointmentsFromFile()

This method performs the opposite operation, loading the file contents into the `appointments` ArrayList. As indicated in the hybrids video, you should use EOFException to terminate loading the ArrayList.

Students often initially encounter problems loading and unloading the files. The most common cause is that the file you think you are loading from/to does not actually exist. Check carefully (using debug, of course) to ensure that there's actually a file where you think it is; your code needs to check for the existence of the file first. If it doesn't exist, it will need to be created.

Another problem: students frequently assume that because their program works fine with the file loaded on their laptop, it will work fine everywhere. But when your lab instructor runs your program, there's no guarantee that the same `apt` file will be available on their laptop. If this happens, you've lost marks, because a major component of the assignment will not execute at all. So test your code thoroughly. In particular,

stress test your code by deleting any existing `CurrentAppointments.apt` files to ensure that your program still works correctly without this default file.

Another potential complication is *serialization*—to be discussed in class shortly. To ensure this doesn't cause you problems, add the following line to the `Appointment` class:

```
public static final long
            serialVersionUID = 1L;
```

Note that this is the identifier Oracle assigns for this purpose. While the convention is to spell a final identifier in ALL_CAPS, in this case you must use the 'camel case' format indicated above.

Also, don't forget to cast the `Object` returned from the file as an `Appointment`, or you won't be able to store it to the `appointments` ArrayList. (Again, all of this is covered in the hybrid videos on File IO.)

Finally, add these two methods to the Scheduler class so that `CurrentAppointments.apt` is loaded into the `appointments` ArrayList whenever a new `Scheduler` is loaded, and saved before it is exited, even if the user does not specifically make this request.

d) Personize the Scheduler using the abstract Employee class

The `cst8284.asgmt2.employee` package downloaded with the CST8284_Assignment2.zip file contains an abstract class designed to hold `Employee` information. (While we could add additional features to this class, we won't bother; the existing class will suffice for the purposes of this assignment.) One of things about each employee's job is that it consists of certain responsibilities and activities that are unique to that position. The abstract method `getActivityType()` is intended to provide this information for any subclass of `Employee`. For our purposes, we'll assume the activities a dentist performs will always be one of the following:

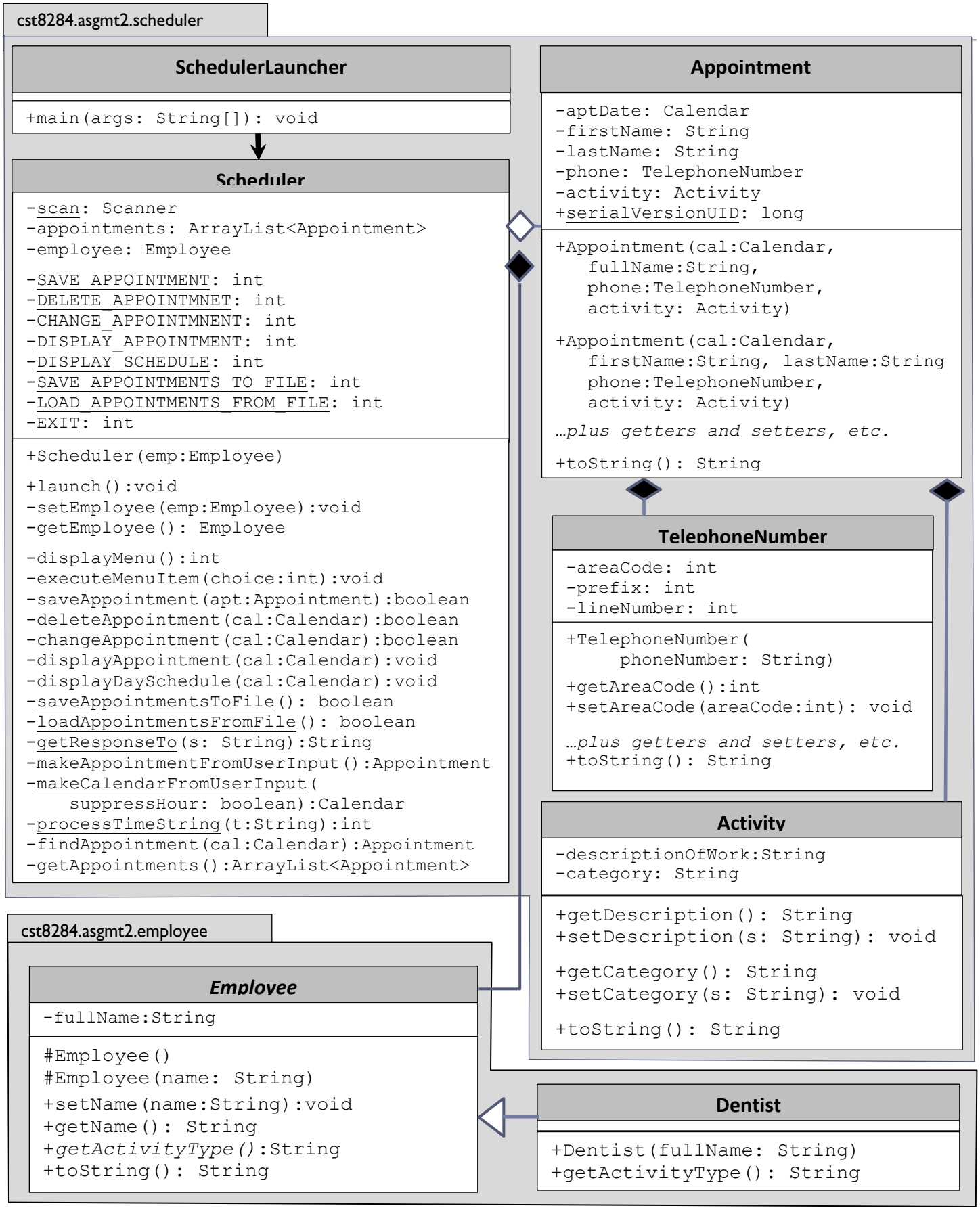
1. `Assessment`
2. `Filling`
3. `Crown`
4. `Cosmetic Repairs`

To the `cst8284.asgmt2.employee` package, add a `Dentist` class that extends `Employee` and overrides `getActivityType()` with a method that (a) prompts the user with the options shown above, and (b) returns the String "Assessment", "Filling", etc. depending on which integer is selected.

In this particular case, you are free to implement the code in any fashion you wish: you are not confined to using the fields and methods indicated on the UML diagram. The one exception to this is that any newly instantiated `Dentist` class must have their full name loaded by the `Employee` superclass constructor. Aside from that, your code to prompt the user and return a String can be implemented any way you like, with any members you require, whether already written in the UML or not.

Once you have written the `Dentist` class, parameterize your `Scheduler()` constructor to take an `Employee` as its sole parameter. Modify the `Scheduler`'s fields to store an `Employee` property and store the `Employee` subtype passed into the `Scheduler` constructor, using a `setEmployee()` method you'll need to build. All of this is indicated in the UML.

Finally, when you instantiate a new `Appointment` in `Scheduler`, use the `Employee`'s `getActivityType()` method to initialize the `Activity`'s category String—the one we left unused in Assignment 1. Thus the user should be prompted whenever a new `Activity` is created to enter the type/category of the appointment using the `Emplolyee`'s `getActivityType()` method. So, while the *description* of the `Activity` may vary from one appointment to the next; the `category` of the appointment will always be restricted to one of the four strings that reflects our `Dentist`'s presumed activity types.


cst8284.asgmt2.scheduler
SchedulerLauncher
+main(args: String[]): void
Scheduler
-scan: Scanner
-appointments: ArrayList<Appointment>
-employee: Employee
-SAVE_APPOINTMENT: int
-DELETE_APPOINTMNET: int
-CHANGE_APPOINTMNENT: int
-DISPLAY_APPOINTMENT: int
-DISPLAY_SCHEDULE: int
-SAVE_APPOINTMENTS_TO_FILE: int
-LOAD_APPOINTMENTS_FROM_FILE: int
-EXIT: int
+Scheduler(emp:Employee)
+launch():void
-setEmployee(emp:Employee):void
-getEmployee(): Employee
-displayMenu():int
-executeMenuItem(choice:int):void
-saveAppointment(apt:Appointment):boolean
-deleteAppointment(cal:Calendar):boolean
-changeAppointment(cal:Calendar):boolean
-displayAppointment(cal:Calendar):void
-displayDaySchedule(cal:Calendar):void
-saveAppointmentsToFile(): boolean
-loadAppointmentsFromFile(): boolean
-getResponseTo(s: String):String
-makeAppointmentFromUserInput():Appointment
-makeCalendarFromUserInput(
suppressHour: boolean):Calendar
-processTimeString(t:String):int
-findAppointment(cal:Calendar):Appointment
-getAppointments():ArrayList<Appointment>
Appointment
-aptDate: Calendar
-firstName: String
-lastName: String
-phone: TelephoneNumber
-activity: Activity
+serialVersionUID: long
+Appointment(cal:Calendar,
fullName:String,
phone:TelephoneNumber,
activity: Activity)
+Appointment(cal:Calendar,
firstName:String, lastName:String
phone:TelephoneNumber,
activity: Activity)
…plus getters and setters, etc.
+toString(): String
TelephoneNumber
-areaCode: int
-prefix: int
-lineNumber: int
+TelephoneNumber(
phoneNumber: String)
+getAreaCode():int
+setAreaCode(areaCode:int): void
…plus getters and setters, etc.
+toString(): String
Activity
-descriptionOfWork:String
-category: String
+getDescription(): String
+setDescription(s: String): void
+getCategory(): String
+setCategory(s: String): void
+toString(): String
cst8284.asgmt2.employee
Employee
-fullName:String
#Employee()
#Employee(name: String)
+setName(name:String):void
+getName(): String
+getActivityType():String
+toString(): String
Dentist
+Dentist(fullName: String)
+getActivityType(): String

III. Notes, Suggestions, Tips, and Warnings

a. As with Assignment 1, before requesting assistance, you should set breakpoints in your code at the 'last known good' location and step forward from there in debug until the error is encountered. Reset the breakpoint to the next 'good' location. Repeat as required. And then, if you're truly stuck, contact the instructor.

b. As before, each class must include, at the top, the following information:

```
/*  Course Name:
    Student Name:
    Class name:
    Date:
*/
```

c. Students are reminded that:
- You should not need to use code/concepts that lie outside of the ideas presented in the course notes.
- You *must* cite all sources used in the production of you code according to the information provided in Module00. Failure to do so *will* result in a charge of plagiarism. The one exception is the information in the course notes themselves
- Students must be able to explain the execution of their code. If you can't explain its execution, then it is reasonable to question whether you actually wrote the code or not. Partial marks, including a mark of zero and a charge of plagiarism, may be awarded if a student is unable to explain the execution of code that he/she presumably authored.

d. The instructor's version of the code will be released at midnight, Oct. 30th, for those students who did not complete this lab on time, or who wish to build their Assignment 3 code on top of the instructor's version (if their own effort was unstable or incomplete.) Note however, that once the 'official' version is released, it essentially nullifies any late submissions.

e. Sample data is shown at the end of this document. *Your code must be able to run using this data as written*; marks will be removed if it does not.

IV. Submission Guidelines

Your code should be uploaded to Brightspace (via the link posted) in a single zip file obtained by:

1) In Eclipse, selecting the ***project*** name (`CST8284_19F_Assignment2`)

2) right clicking on 'Export' and selecting General/Archive File; click Next;

3) in the Archive File menu make sure *all* of the project subfolders are selected (src, bin, .settings) and the 'Save in zip format' and 'Create in Directory Structure' radio buttons are selected

4) In the 'To Archive File' window, save your zip file to a location you'll remember. But make certain the name of your zip file corresponds to the following format, as outlined in Module 00:

Assignment2_SecXXX_Yourlastname_Yourfirstname.zip

including the underscores and capitals, but with *your* last and first name inserted as indicated Note that XXX represents your lab section number, which will be one of 301, 302, 303, 304, 311, 312, 313, 314.. Failure to label your zip file correctly will result in lost marks.

Note:
- A good safety precaution is to always take the .zip file you've just uploaded to Brightspace and open it on your laptop. Better still, in Eclipse, import the zipped file and check to make sure everything is there;
- You can upload as many attempts at Assignment 2 as you'd like, but only the final attempt is marked

Addenda:
(As needed)

Sample Output

(Forthcoming: Check for Version 1.10, out shortly)