Project 1 – 75 points

For this project we will be constructing an appointment book in an incremental fashion. It will be drawing on everything we have learned in the course thus far. You have some flexibility here in how you design this, so if you make any assumptions about the requirements, be sure to state them in your comments. This project is more independent than the Labs/Assignments. So, while the TAs and I are happy to help you with course concepts and requirements, we won't help you design or code for this project.

Part 1 (35 points)

Create an AppointmentBook class that stores a collection of `Appointment`s. Implement a superclass `Appointment` and subclasses `Onetime`, `Daily`, and `Monthly`. An appointment has a textual description (for example, "hit the gym" or "go see Star Wars again") and a date of type GregorianCalendar (see below). A `Onetime` appointment occurs only once, `Daily` occurs every day, and `Monthly` occurs once a month on a provided day. Daily and Monthly appointments continue(recur) indefinitely, that is, there is no end date. **There is no such thing as just an "Appointment", rather each appointment object instance is one of the three (sub)types.** (5 points)

Independent Learning through Application Programming Interface API (Technical Document) (10 points)

The date property of each appointment instance will be of type GregorianCalendar, which is a popular date Class in Java (https://docs.oracle.com/javase/7/docs/api/java/util/GregorianCalendar.html). You will be required to read about and understand how to use that class. The API (website) has detailed descriptions of all the properties and public methods, and example usages that you can learn from. You will be graded out of 10 on how well you include, and utilize this Class in your program. Some methods to consider

- Constructor and/or set(…) method (inherited from the **Calendar** Java Class),
- get(…), which is inherited from the **Calendar** Java Class
- There are some useful **Compar**ison methods that allow you to compare dates.

This must be done independently: The TAs and I will not be assisting you in reading and comprehending the API. Look at the example usage on the website!

occursOn Method (20 points)

Write a method for the Appointment class `occursOn(int year, int month, int day)` that checks (returns) whether the appointment occurs on that specified (parameterized) date or not. There is no behavior for a generic Appointment, but each extension (sub class) must have its own implementation. For example, for a monthly appointment, you must check whether the day of the month of the appointment being queried (parameter) matches with the specified day, that is, checking that monthly appointment, X, occurs on the 15th day of the month and that the specified day occurs on or after the appointment being queried.

Part 2 (15 points: 5 points x 3 types of appointments to add)

Continue developing the AppointmentBook class. Add a **single** method to the AppointmentBook class that allows adding new appointments. The method parameters must include the type of the appointment (you must use either an ENUM (link here) or **constant** to represent different types of appointments), the appointment description, and the date (can be in the form of GregorianCalendar date or a String that you later convert to GregorianCalendar).

Part 3 (15 points: 5 points for Save, 10 for loading)

Improve the appointment book program by having a method in the AppointmentBook class that saves a specific single Appointment's data to a provided (parameterized) file AND a separate method to load an Appointments' data from a provided (parameterized) file and add them to an AppointmentBook instance. These parameters can be Strings that represent a file, or a File object. Just make sure you document which you choose in the comments.

The saving part is straightforward: Make a method `save` that takes in any Appointment and file (or file location). For the specific Appointment, save the type, description, and date to a file. You will need it in a format that you can eventually load. Ask yourself, should this be a static method or instance method?

The loading part is not so easy. Create a `load` method that, given a specific (parameterized) file, will load each Appointment described in that file using the format you created above into the specific instance of an AppointmentBook that the method is called on. So, for each appointment in that file, first determine the type of the appointment to be loaded, create an object of that type, and then call your method from Part 2 to add the appropriate Appointment into that AppointmentBook instance. Ask yourself, should this be a static method or instance method?

Tester (10 points)

In a separate tester class, create an Appointment Book. Fill that AppointmentBook's collection of `Appointment` objects with a mixture of appointments (and types) using your method created earlier. Instead of using the 4 testing steps we taught earlier this semester, have the user (from the console) then enter a date and print out all appointments that occur on that date.

Submission

Turn in a .zip file or all your .java files to Canvas by 11:59pm on Friday March 10th. Grading/deductions will be based on conforming to the standards we reviewed in class as well as following the requirements of this project.

Good luck!