Project #2 – 160 points

Due Date – Monday, October 14, by 11:59 pm.

Submission

(1) Zip the project folder and submit the zipped file to Canvas.
(2) The zip file must include the following grading items.
- **The source folder "src"** containing the Java files (*.java) [**110 points**]
  (a) Include at least 2 Java packages, details in the Project Requirement section, or **-2 points**
  (b) Use all lowercase letters for the package names or **-2 points**
  (c) Add the **@author** tag in the comment block on top of EVERY Java class and put the name(s) who implemented the Java class or **-2 points**
- **JUnit test classes**, including the test cases implemented to test the methods below.
  (a) The `isValid()` method of the `Date` class. [**15 points**]
  (b) The `compareTo()` method of the `Profile` class. [**10 points**]
  (c) The `add()` and `remove()` methods of the generic `List<E>` class. [**10 points**]
- **Class Diagram**. [**10 points**]
- The **Javadoc folder** "**doc"** that includes all the .html files generated by Javadoc. [**5 points**]

(3) The submission button on Canvas will disappear after **October 14, 11:59 pm**. Do not wait until the last minute to submit the project. You are responsible for ensuring the project is well-received in Canvas by the due date and time. **You get 0 points** if you do not have a submission on Canvas. **Projects sent through emails or other methods will not be accepted.**

Project Description

RU clinic is adding imaging services to the Piscataway and Bridgewater locations and adding more appointment slots to accommodate the patients' needs. The imaging services include X-rays, ultrasounds, and CAT scans. Each location providing the imaging services has three imaging rooms: one for X-rays, one for ultrasounds, and one for CAT scans. The clinic has contracted with more service providers to support the new services. You are tasked to extend the functionality of the scheduler software developed in Project 1 to support the new services, including the office visits and imaging services. The clinic will provide the list of providers in a text file from a data source. In addition to the requirements listed in Project 1, the clinic has added the list of new requirements below.

1. The system shall be able to import the list of providers from a text file provided by the clinic.
2. The system shall provide 6 slots in the morning and afternoon, respectively, a total of 12 slots per weekday. Each slot is 30 minutes, the first in the morning is 9:00 AM, and the first in the afternoon is 2:00 PM. The last appointment is at 4:30 PM.
3. The system shall be able to schedule a new office appointment with a doctor for a patient with the requested date, time, and the doctor's national provider identification number (NPI.)
4. The system shall be able to schedule a new imaging appointment for a patient with the requested date, time, and the indicated imaging service, X-ray, ultrasound, or CAT scan. The system shall automatically assign the next available technician to the appointment.
5. The system shall be able to display the list of providers ordered by provider profile.
6. The system shall be able to display the list of office or imaging appointments ordered by location.
7. The system shall be able to display the expected credit amounts for each service provider for seeing patients.

The S command in Project 1 will be deprecated and replaced by the D command and T command for new appointments. New commands shall also be added to support the new requirements.

- **D** command to schedule a new office appointment with a doctor. The user shall enter the appointment information in the following sequence: the date, timeslot, patient's first name, last name and date of birth, and the doctor's NPI. Below is an example of a command line for adding a new office appointment. NPI is unique to each doctor.

  ```
  D,9/30/2024,1,John,Doe,12/13/1989,120
  ```

  Users make mistakes, including invalid data tokens or missing data tokens. Your software must check if the data tokens are invalid in the below order or catch the exceptions if there are missing data tokens. All data tokens after the D command are not case-sensitive.

  1. The date of an appointment is
     - an invalid calendar date,
     - today,
     - a date before today
     - a date on Saturday or Sunday
     - a date not within six months from today.
  2. The timeslot does not exist.
  3. The date of birth of a patient is not a valid calendar date, is today, or is a future date.
  4. Missing data tokens.
  5. An appointment with the same patient profile, date, and timeslot already exists.
  6. The NPI is not numeric or does not exist.
  7. The provider associated with the NPI is not available at the specified timeslot.
- **T** command to schedule a new imaging appointment. The user shall enter the appointment information in the following sequence: the date, timeslot, patient's first name, last name, date of birth, and the room type of imaging service. Below is an example of a command line for adding a new imaging appointment.

  ```
  T,9/30/2024,1,John,Doe,12/13/1989,xray
  ```

  In addition to checking #1 ~ #5 in the D command above, the system shall check the conditions below for an imaging appointment.

  6. The requested imaging service is not provided.
  7. Cannot find a technician with the timeslot and imaging service available.

  For #7, the system shall maintain a circular list of technicians who are associated with their locations. The system shall use the list as the rotation to assign the technician to an appointment. It shall first check if the next technician on the list is available at the timeslot. If unavailable, it moves forward to the next technician on the circular list. If available, then it checks if the indicated imaging room is available at the technician's associated location. Note that each location only has one room for each type of imaging service.
- **C command** to cancel an existing office or imaging appointment, given the appointment date, timeslot, and patient profile. This command works the same as Project 1.
- **R command** to reschedule an office appointment with the date, timeslot, patient profile, and a new timeslot. For simplicity, imaging appointments cannot be rescheduled. This command works the same as Project 1.
- **PA, PP, PL, PS, and Q** commands work the same as Project 1.
- **PO** command to display the list of office appointments, sorted by the county name, then date and time.
- **PI** command to display the list of imaging appointments, sorted by the county name, then date and time.
- **PC** command to display the expected credit amounts for the providers for seeing patients, sorted by provider profile.

Project Requirement

1. You MUST follow the Coding Standard posted on Canvas under Modules/Week #1. **You will lose points** if you violate the rules listed in the coding standard.
2. You are required to follow the Academic Integrity Policy. See the **Additional Note #13** in the syllabus posted on Canvas. If your team uses a repository hosted on a public website, you MUST set the repository to private. Setting it to public is considered as violating the academic integrity policy. The consequences of violation of the Academic Integrity Policy are: **(i) all parties involved receive 0 (zero) on the project, (ii) the violation is reported, and (iii) a record on your file of this violation.**
3. Test cases for grading are included in **Project2TestCases.txt.** The associated output generated from the test cases is in **Project2Output.txt**. The data for the test cases are fictional for testing purposes. Your project should be able to ignore the empty lines between the test cases in **Project2TestCases.txt**. You MUST use the **Scanner class** to read the command lines from the standard input (**System.in**), DO NOT read from an external file, EXCEPT the "**providers.txt**" for loading the providers, or you will **lose 5 points**.
4. The graders will run your project by copying the test cases in **Project2TestCases.txt** and pasting them to the terminal. They will compare your output with the expected output in **Project2Output.txt**. You will **lose 2 points** for each output not matching the expected output OR for each exception not caught, causing your project to terminate abnormally.
5. Each source file (.java file) can only include one public Java class; the file name is the same as the Java class name or **-2 points**.
6. Your program MUST handle bad commands; **-2 points** for each bad command not handled.
7. You CANNOT import any Java library classes, EXCEPT the **Scanner**, **StringTokenizer, Calendar,** and **DecimalForamt** class, or the **Java Exception** classes and the packages you developed. **You will lose 5 points** for EACH additional Java library class imported, with a **maximum of -10 points**.
8. You CANNOT use the Java library class **ArrayList** anywhere in the project OR use any Java library classes from the Java Collections, or **you will get 0 points for this project**!
9. Be specific when importing Java library classes and DO NOT import unnecessary classes or the whole package. For example, `import java.util.*;` this will import all classes in the `java.util` package. You will **lose 2 points** for using the asterisk "*" to include all the Java classes in the `java.util` package, or other java packages, with a **maximum of -4 points.**
10. In this project, you will be refactoring your code and adding new functionality. You must define the inheritance hierarchy below and reuse your code as much as possible. **-5 points** for each class missing or incorrect inheritance relationship. **-5 points** for repeating the same block of code in multiple places.
    - **Person** class defines the profile of each instance in the software system. See the details of this class in the requirement #12 below.
    - **Patient** class extends the Person class and includes the instance variable `private Visit visit;` to keep track of a linked list of visits.
    - **Provider** class extends the Person class and includes the instance variable `private Location location;` to keep track of the practice location. This class replaces the Enum Provider class in Project 1. This is an *abstract* class with an abstract method, `public abstract int rate();` that returns the provider's charging rate per visit for seeing patients.
    - **Doctor** class extends the Provider class and includes two instance variables below.

```
private Specialty specialty;//encapsulate the rate per visit based on specialty
private String    npi; //National Provider Identification unique to the doctor
```

- **Technician** class extends the Provider class and includes the instance variable `private int ratePerVisit;` to keep track of the technician's charging rate per visit.
- **Imaging** class extends the Appointment class from Project 1 to include the instance variable `private Radiology room;` to keep track of the imaging room, X-ray, ultrasound, or CAT scan for the imaging appointment.

11. **Polymorphism** is required, i.e., dynamic binding for all the `equals(), compareTo(), toString(),` and the `rate()` method, and the methods in the generic `List<E>` class, or you will **lose 10 points.**
12. You MUST implement or refactor the Java classes below. **-5 points** for each class missing or NOT used. You should define necessary constant names in UPPERCASE letters and **DO NOT** use MAGIC NUMBERs, or **you will lose points** listed in the Coding Standard.

    You CANNOT use System.in or System.out statements in ALL classes, EXCEPT the user interface class `ClinicManger.java` or the testbed `main()` methods. **-2 points** for each violation, with a **maximum of -10 points.** You must always add the `@Override` tag for overriding methods or **-2 points** for each violation.

    (a) **Appointment class –** change the data type for patient and provider to Person. You can change the modifiers to "protected" or keep them as "private". You CANNOT change or add instance variables. **-2 points** for each violation.

```
public class Appointment implements Comparable <Appointment> {
    protected Date     date;
    protected Timeslot timeslot;
    protected Person   patient;
    protected Person   provider;
}
```

    (b) **Person class –** this is the superclass of the Patient and Provider class. You CANNOT change or add instance variables. **-2 points** for each violation. You can define the modifier as "protected" or "private".

```
public class Person implements Comparable<Person>{
    protected Profile profile;
}
```

    (c) **Timeslot class –** this is the Enum class in Project 1; you must change it to a regular class since we add more slots. **-5 points** if you define this class as an Enum class.

```
public class Timeslot implements Comparable<Timeslot> {
    private int hour;
    private int minute;
}
```

    - You can add additional constants, constructors, and methods. Instance variables must be private; you CANNOT change or add instance variables. **-2 points** for each violation.
    - You must implement `equals(), toString() and compareTo()`, or **-2 points** for each violation.

    (d) **Imaging class –** this class extends the Appointment class to hold additional data for imaging appointments. You cannot add or change the instance variable or **-2 points**.

```
public class Imaging extends Appointment {
    private Radiology room;
}
```

    (e) **Radiology class.** Define a simple Enum type to include three types of imaging services – `CATSCAN, ULTRASOUND,` and `XRAY;`

(f) **List<E> class –** this is a generic class for code reuse. Refactor the List class in Project 1 by changing the methods to take a generic type. You must remove ALL print methods **or -5 points**.

```
public class List<E> implements Iterable<E> {
    private E[] objects;
    private int size;
    public List() { } //default constructor with an initial capacity of 4.
    private int find(E e) {}
    private void grow() {}
    public boolean contains(E e) {}
    public void add(E e) {}
    public void remove(E e) {}
    public boolean isEmpty() {}
    public int size() {}
    public Iterator<E> iterator() {}
    public E get(int index) {} //return the object at the index
    public void set(int index, E e) {} //put object e at the index
    public int indexOf(E e) {} //return the index of the object or return -1
    private class ListIterator<E> implements Iterator<E> {
        public boolean hasNext(){}//return false if it's empty or end of list
        public E next(){} //return the next object in the list
}
```

- `E` is the generic type where you can replace it with any reference type. Always use this class whenever you want to hold a collection of objects or **-5 points**. You should eliminate the MedicalRecord class from Project 1.
- You cannot add or change the instance variables and must implement the API listed, or **-2 points** for each violation.
- You cannot add other constructors or additional methods, or **-2 points** for each violation. Extend this class if you need to add anything.
- You must use a single instance of List<Appointment> to hold a list of office and imaging appointments and use a single instance of List<Provider> to hold a list of all providers; **-5 points** for each violation.

(g) **Sort class –** this utility class handles the sorting. You should define only static methods for sorting. For example, sort the appointments ordered by different keys. Your team should make design decisions so you can reuse the code as much as possible. The following method signatures are examples for your reference.

```
public static void appointment(List<Appointment> list, char key) {}
public static void provider(List<Provider> list) {}
```

(h) **ClinicManager class**

- This is the user interface class to process the command lines entered on the terminal. This class replaces the `Scheduler.java` in Project 1. An instance of this class can process a single command line or multiple command lines at a time, including the empty lines. **You will lose 10 points** if it cannot process multiple command lines.
- When your software starts running, it shall display `"Clinic Manager is running."`. It shall automatically load the list of providers from the text file "**providers.txt**" in the project folder and display the list after it is loaded, sorted by provider profile. It also creates a rotation list of technicians for scheduling imaging appointments. See the Project2Output.txt for the order of the rotation list. It will continuously read and process the command lines until the "Q" command is entered. If the Q command is entered, display `"Clinic Manager terminated"`, then the software stops normally. **-2 points** for each violation.

- You must define a `run()` method that includes a while loop to continuously read the command lines until a "Q" command is entered. You will **lose 5 points** if the `run()` method is missing. You MUST keep this method **under 40 lines** for readability, or you will **lose 3 points**. You can define necessary instance variables and private helper methods to handle each command.

(i) **RunProject2 class.** This is a driver class that runs your software. The graders will run this class to grade your project.

```
public class RunProject1 {
    public static void main(String [] args) {
        new ClinicManager().run();
    }
}
```

13. Create two Java packages to organize the source files – a Java package "util" under the "src" folder to include the utility classes you developed for this project. For example, List<E>, Sort, Date, or circular linked list classes; another Java package to include the other classes. **-3 points** if this is not done properly.
14. **JUnit test classes.** Create 3 JUnit test classes and write code to test the following methods.
    (a) `isValid()` in the `Date` class; **four** invalid and **two** valid test cases; reuse the code from Project 1.
    (b) `compareTo()` method in the `Profile` class; **three** test cases return -1, **three** test cases return 1, and **one** test case returns 0; reuse the code from Project 1.
    (c) `add()` and `remove()` methods in the `List<E>` class;
    - `add()` – **one** test case for adding a Doctor object and **one** for adding a Technician object to a `List<Provider>` object.
    - `remove()` – **one** test case to remove a Doctor object and **one** to remove a Technician object from a `List<Provider>` object.
15. **Class Diagram.** Create a class diagram to document your software structure for Project 2. Hand-drawing the diagram is not acceptable, and you'll **lose 10 points**. You can follow the links below to create a class diagram online and save it to your device as a picture, which can be included in your submission. You can also use other UML tools if you like. For each rectangle (class), include ONLY the instance variables and public methods.

    DO NOT include the constants, private methods, the JUnit test classes, Java library classes, and RunProject2.

    - http://draw.io/ → Create New Diagram → UML → Class Diagram
    - https://online.visual-paradigm.com/app/diagrams/ → Create New → Class Diagram
16. You must **generate the Javadoc** after you properly comment your code. Set the scope to 'private" so your Javadoc will include the documentation for the private data members, constructors, and private and public methods of all Java classes.

    Generate the Javadoc in a single folder and include it in the zip file to be submitted to Canvas. **You are responsible for double-checking your Javadoc folder after you generate it. Submitting an empty folder will result in 0 points for this part.** Open the Javadoc folder and look for the **index.html** file. Double-click the file and check every link to ensure all comments are NOT EMPTY. You will lose points if any description in the Javadoc is empty. You will **lose 5 points** for not including the Javadoc.