Project #1 – 100 points

Due Date and Time

Monday, October 2, by 11:59pm.

Submission

(1) Zip your project folder and submit the zipped file to Canvas. Include the following grading items.
- **Source folder src,** containing the source files (*.java) [60 points]
  (a) MUST create a Java package to hold the source files; MUST use all lowercase letters for the package name; you will **lose 2 points** if this is not done properly.
  (b) MUST include the **@author** tag in the comment block on top of EVERY Java class and put the name(s) who implemented the Java class, or you will **lose 2 points**.
- **Test specification**. Use a document editor and design the test cases to test the **isValid()** method of the **Date class**, and the **equals()** method of the **Event class**. [15 points]
- **Testbed main()** in the following Java classes implementing the test cases listed in your test specification.
  (a) Date class [15 points]
  (b) Event class [5 points]
- **Javadoc** folder named **doc** to include all the files generated by the Javadoc. [5 points]

(2) The submission button on Canvas will disappear after **October 2, 11:59pm**. It is your responsibility to ensure your Internet connection/speed is good for submitting the project on time. **You get 0 points** if you do not have a submission on Canvas. **Projects sent through the emails will not be accepted.**

Project Description

Your team will develop a simple software to manage an event calendar, which is used to schedule the events held at different campus locations listed below.

HLL114, Hill Center, Busch
ARC103, Allison Road Classroom, Busch
BE_AUD, Beck Hall, Livingston
TIL232, Tillett Hall, Livingston
AB2225, Academic Building, College Avenue
MU302, Murray Hall, College Avenue

The software shall allow the users to book a room for an event or cancel an event on the calendar. The software shall be able to display the event calendar sorted by event date/time, by campus/building/room, or by the hosting department. Let's assume the event calendar only opens to five departments: CS (computer science), EE (electrical engineering), ITI (information technology and informatics), MATH (mathematics) and BAIT (business analytics and information technology).

The software is an interactive system where the users enter command lines on the console/terminal, and the system immediately generates responses and output the results on the console/terminal. That is, when the user hits the enter key, the system reads the data entered, process the data, and immediately output the results.

A command line always begins with a command and followed by additional data tokens delimited by one or more spaces. Commands are in uppercase letter(s) and **case-sensitive**, which means the commands in lowercase letters are invalid. Below is a list of commands you must implement.

- **A** command; to **add an event** to the event calendar. To book a timeslot on the calendar, an event shall include the event date, timeslot, location, contact and the duration of the event. Below is an example of a command line for

adding an event to the calendar. You can assume that **the user will always enter enough data tokens** in the order shown below.

```
A  2/29/2024  afternoon  HLL114  CS  cs@rutgers.edu  60
```

The above command line starts with the **A command**, followed by a future event date, timeslot, location, contact and duration in minutes. The date shall be given in **mm/dd/yyyy** format, and the date should be within 6 months from today's date. All the campus locations provide three timeslots daily: 10:30am, 2:00pm and 6:30pm. A timeslot is entered as either "morning", "afternoon", or "evening". A location is entered in acronym listed in page #1, for example, HIL114. A contact of an event includes the hosting department and an email address. A department name is entered in acronym listed in page #1, for example, CS. The email address must have the domain name @rutgers.edu. The event duration is a positive integer representing the number of minutes. The duration is at least 30 minutes and at most 120 minutes.

The command is case-sensitive as mentioned above, however, all data tokens in the same command line are NOT case-sensitive. Your software shall check the following conditions before adding an event to the calendar. Refer to the sample output for the error messages to display.

1. An event is not a valid calendar date.
2. An event date is not a future date.
3. An event date is more than 6 months away from today's date.
4. An invalid timeslot.
5. A location that is not one of the six locations listed.
6. A department name that is not one of the five departments listed.
7. An invalid email address containing the wrong format or wrong domain name.
8. Conflict of schedule - an event with the same date/timeslot/location is already on the calendar.

- **R** command, to **cancel an event** and remove the specified event from the calendar, for example,

```
R 12/22/2023 MORNING HLL114
```

You must check if the date is valid. Refer to the sample output for the messages to display.

- **P** command, to display the event calendar on the console/terminal, with the current order in the array**.**
- **PE** command, to display the event calendar on the console/terminal, sorted by the event date and timeslot of the timeslot. That is, if two events have the same date, display the events in the order of the timeslots.
- **PC** command, to display the event calendar on the console/terminal, sorted by campus and building/room. That is, if two events are held on the same campus, display the events in the order of building/room number.
- **PD** command, to display the event calendar on the console/terminal, sorted by the department in the contact. If two events have the same hosting department, the order doesn't matter.
- **Q** command, to stop the execution of the software and display `" Event Organizer terminated."`

Project Requirement

1. You MUST follow the Coding Standard posted on Canvas under Week #1 in the "Modules". **You will lose points** if you are not following the rules.
2. You are responsible for following the Academic Integrity Policy. See the **Additional Note #14** in the syllabus.
3. Test cases for grading are included in the file **Project1TestCases.txt** and the associated output is in the file **Project1ExpectedOutput.txt**. Your project should be able to read the test cases from console in the same order with the test cases provided in **Project1TestCases.txt**, line by line without getting any exceptions. Your program should be able to ignore the empty lines. The graders will run your project with the test cases in

Project1TestCases.txt and compare your output with the expected output in **Project1ExpectedOutput.txt**. You will **lose 2 points** for each output not matching the expected output, OR for each exception causing your project to terminate abnormally. You MUST use the **Scanner class** to read the command lines from standard input (**System.in**), DO NOT read from an external file, or you will **lose 5 points**.

4. Each source file (.java file) can only include one public Java class, and the file name is the same with the Java class name, or you will lose **-2 points**.
5. Your program MUST handle bad commands; **-2 points** for each bad command not handled, with a **maximum of losing 6 points**.
6. You are not allowed to use any Java library classes, EXCEPT the **Scanner**, **StringTokenizer, Calendar** and **DecimalForamt** class. **You will lose 5 points** FOR EACH additional Java library class imported, with a **maximum of losing 10 points**.
7. You are not allowed to 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**!
8. When you import Java library classes, be specific and DO NOT import unnecessary classes or import 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 losing 4 points.**
9. You MUST include 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 2 points**. A good approach is to use Java Enum classes or a public class to define all the constants.

   You **CANNOT use System.in** or **System.out** statements in ALL classes, EXCEPT the interface class EventOrganizer.java, **-2 points** for each violation, with a **maximum of 10 points off.**

   You must **always add the @Override** tag for overriding methods, or **-2 points** for each violation.

   (a) **Event class**

```
public class Event implements Comparable<Event> {
    private Date     date;      //the event date
    private Timeslot startTime; //the starting time
    private Location location;
    private Contact  contact;  //include the department name and email
    private int      duration; //in minutes
}
```

- Define necessary constants, constructors and methods. However, you CANNOT change or add instance variables. **-2 points** for each violation.
- Must override **equals()** and **toString()** methods, and implement the **compareTo()** methods to compare the event date and timeslot. You must add the **@Override** tags. **-2 points** for each violation.
- The **toString()** method returns a textual representation of an event in the following format.

[Event Date: 10/21/2023] [Start: 2:00pm] [End: 3:00pm] @HLL114 (Hill Center, Busch) [Contact: Computer Science, cs@rutgers.edu]

- The **equals()** method returns true if two dates, timeslots and locations are equal.
- The **compareTo()** method compares the dates first, then the timeslots if the dates are the same.

(b) **EventCalendar class**

This is an array-based implementation of a linear data structure to hold the list of events. A new event is always added to the end of the array. An instance of this class is a growable list with an initial array capacity of 4, and it automatically increases the capacity by 4 whenever it is full. The list does not decrease in capacity.

```
public class EventCalendar {
    private Event [] events;    //the array holding the list of events
    private int      numEvents; //current number of events in the array
     ...
    private int find(Event event) { } //search an event in the list
    private void grow() { } //increase the capacity by 4
    public boolean add(Event event) { }
    public boolean remove(Event event) { }
    public boolean contains(Event event) { }
    public void print() { } //print the array as is
    public void printByDate() { } //ordered by date and timeslot
    public void printByCampus() { } //ordered by campus and building/room
    public void printByDepartment(){ } //ordered by department
}
```

- Define necessary constants, constructors, and methods. However, you CANNOT change or add instance variables. **-2 points** for each violation.
- Must implement and use the methods listed above; you CANNOT change the signatures of the methods. **-2 points** for each violation.
- You CAN use **System.out** ONLY in the four **print()** methods listed above.
- The **find()** method searches an event in the list and returns the index if it is found, it **returns -1** if it is not in the list. You must define a constant identifier "NOT_FOUND" for the value -1.
- The **remove()** method delete an event from the calendar. This method maintains the relative order of the events in the array after the deletion, i.e., shift every element in the array up one position. **-3 points** if this is not done correctly.
- You must use an "in-place" sorting algorithm to implement the sorting, i.e., the order of the objects in the array will be rearranged after the sorting. You CANNOT use **Arrays.sort()** or **System.arraycopy()** or any other Java library classes or utilities for sorting. You must write the code yourself for sorting. You will **lose 10 points** for the violation.

(c) **EventOrganizer class**

- This class is the user interface class to process the command lines. An instance of this class can process a single command line or multiple command lines at a time. If it cannot process multiple command lines at a time, **you will lose 10 points**.
- Your software starts running by instantiating an object of this class. It shall display "`Event Organizer running...`". Next, it will read and process the command lines continuously until the "Q" command is entered. Before the software stops running, display "`Event Organizer terminated`".
- You must define a `public void run()` method that includes a while loop to continuously read the command lines from the console 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 for handling the commands.

(d) **RunProject1 class** is a driver class to run your software.

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

(e) **Date class**

```
public class Date implements Comparable<Date> {
    private int year;
    private int month;
    private int day;
    public boolean isValid() //check if the date is a valid calendar date
}
```

- Define necessary constants, constructors, and methods. However, you CANNOT change or add instance variables, **-2 points** for each violation.
- Must implement the Comparable Interface and implement the **compareTo()** method, or **lose 2 points.**
- Must include a testbed main(), or you **lose 15 points**. You CAN use **System.out** in the testbed main.
- Must implement the **isValid()** method to check if a date is a valid calendar date.
  - For the month, January, March, May, July, August, October and December, each has 31 days; April, June, September and November, each has 30 days; February has 28 days in a non-leap year, and 29 days in a leap year. DO NOT user **magic numbers** for the months, days and years. Below are some examples for defining the constant names.

    ```
    public static final int QUADRENNIAL = 4;
    public static final int CENTENNIAL = 100;
    public static final int QUATERCENTENNIAL = 400;
    ```

    To determine whether a year is a leap year, follow these steps:

    Step 1. If the year is evenly divisible by 4, go to step 2. Otherwise, go to step 5.
    Step 2. If the year is evenly divisible by 100, go to step 3. Otherwise, go to step 4.
    Step 3. If the year is evenly divisible by 400, go to step 4. Otherwise, go to step 5.
    Step 4. The year is a leap year.
    Step 5. The year is not a leap year.

  - You MUST design the test cases to thoroughly test the **isValid()** method. Follow the instructions under the "Test Design" section in the **Coding Standard** posted on Canvas. You must implement the test cases in the testbed main or **lose 15 points**. Refer to the demo posted on Canvas.

(f) **Contact class**

```
public class Contact {
    private Department department;
    private String     email;
    public boolean isValid()
}
```

- Define necessary constants, constructors and methods. However, you CANNOT change or add instance variables, **-2 points** for each violation.
- The **isValid()** method checks if the department name and email are valid, or **lose 2 points**.

(g) **Enum classes**

- Department, use the department acronyms for the constant names, or **lose 3 points**. You can include a property to represent the full name or override the toString() method to produce the full name.
- Location, use the room number as the constant name, and include the building name and campus as the properties, or **lose 3 points.**
- Timeslot, use MORNING, AFTERNOON and EVENING for the constant names, and include the hour and minute as the properties, or **lose 3 points**.

10. You are required to **generate the Javadoc** after you properly comment your code. Set the scope to 'private" so your Javadoc will include the documentations for the private data members, constructors, 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 to double check your Javadoc folder after you generated it. Submitting an empty folder will result in 0 points.** Open the folder and look for the **index.html** file. Double click the file and check every links 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.