Lab Test [25 marks]

General Instructions:

1. You will perform the lab test on your personal laptop.

2. You are not allowed to communicate in any way during the test.

3. You can refer to any file on your laptop.
   Any violation of the instructions above will result in a zero score for your lab test.

4. Make sure your code can generate exactly the same output as we show in the sample runs. You may get some marks deducted for missing spaces, missing punctuation marks, misspelling, etc. in the output.

5. Do not hardcode. We will use different test cases to test and grade your solutions.

   Failure to do the following will attract a penalty up to 20% of your score for that question.

6. Include your name as author in the comments of all your submitted source files. For example, if your registered name is "Sotong TAN" and email ID is sotong.tan.2020, include the following block of comments at the beginning of each source file (.java) you write.

```
/*
 * Name: Sotong Tan
 * Email ID: sotong.tan.2021
 */
```

7. Follow standard Java coding conventions (e.g. naming of getter and setter methods, choice of identifier names for classes, methods and variables) as well as indent your code correctly.

8. Ensure that all your Java code can compile without compilation errors. They must compile with any test class(es) that are provided. You may wish to comment out the parts in your code, which cause compilation errors. But remember that commented code will NOT be graded. **Only code without compilation errors SHALL be graded.** You may need to comment out parts of the Test class' code that you have not implemented in order to test your implemented solutions.

9. Download the source code and rename the root folder in the zip file to your **Email ID (e.g. `sotong.tan.2021`)**

Question 1 [10 marks]

You are given the class diagram in the appendix, and the API documentation in the api folder.

1. [3 marks] Implement `InvalidTestException`. It is an unchecked exception.

2. [5 marks] Implement `ArtTest`. This is an abstraction of the Antigen Rapid Test (ART) used to screen for COVID-19.
   a. The attribute, `result` stores the raw result of the test. If the value is
      i. "CT", the test is positive.
      ii. "C", the test is negative.
      iii. "T" or "", then the test is invalid.

   b. The constructor throws an InvalidTestException if the `result` is not a valid string ("CT", "C", or "T").

   c. `toString()` returns a String in the following format
   "<employeeID> <Invalid|Positive|Negative>(<date>)".
   If employeeId is "Beng", result is "CT", date is "04/11/2021 13:50", this method returns the following string "Beng(Positive, 04/11/2021 13:50)".

   d. The `isNegative` method throws an `InvalidTestException` if the test is invalid.

   e. Two ArtTest are equal if and only if they
      i. have the same result and
      ii. taken at the same date & time and
      iii. taken by the same employee.

3. [2 marks, ***] Implement `ArtComparator`. This comparator allow `ArtTest` objects to be sorted in the following order:
   a. `employeeId` in **ascending** order (e.g. 'a' before 'b', 'ab' before 'ac') .
   b. `date` in **ascending** order (e.g. 2020 before 2021, 1 (Jan) before 12 (Dec), 1 (day of month) before 31) .
   c. the test's validity (negative, positive, then invalid).

Question 2 [15 marks]

You are given the class diagram in the appendix, and the API documentation in the api folder.

Implement the following static methods.

1. **[3 marks]** Implement `getInOfficeDates` (`CovidUtility1.java`). It takes in two parameters:
    a. `target` (type: `String`): the employee's ID.
    b. `entries` (type: `List<Entry>`): Each `Entry` states the enter and exit time of an employee. An employee could enter and exit the building multiple times in a day.
    Note:
        1. For the same employee, you can assume that the entries (enter and exit timings) are non-overlapping. You will not have entries pairs like (10:00 - 11:00) and (10:30 - 11:00) with an overlapping time period from 10:30 - 11:00.
        2. `entries` are `NOT` sorted in any particular order.
        3. Each `Entry` object represents the enter and exit timings of an employee. The enter and exit timings will happen on the same day.

    It returns the dates where the target is in office, sorted in ascending order.

2. **[3 marks]** Implement `getTotalTimeInOffice` (`CovidUtility2.java`). It takes in two parameters:
    a. `entries` (type: `List<Entry>`). Use the same assumptions as Q2 Part 1.

    b. `target` (type: `SimpleDate`): The date of interest.

    This method returns the employees who came to the office on the `target` date and their time (in minutes) spent (`employeeId` as the key, the total duration as the value). The output should be sorted by `employeeId` in ascending order.

For Q3 - Q5, you can assume that the parameters are non-`null`.

3. **[3 marks]** Implement `getViolators` (`CovidUtility3.java`). It takes in two parameters:
    a. `employees` (type: `List<Employee>`). Employees are either essential or non-essential.
        i. Essential employees can come into the office every day.
        ii. Non-essential employees can come into office on certain days of the week (1 - Monday, 2 - Tuesday, …, 7 - Sunday).

        The `Employee` class represents the essential workers.
        The `NonEssentialEmployee` class represents the non-essential employees.

    b. `entries` (type: `List<Entry>`): Use the same assumptions as Q2 Part 1.

    This method returns the unique list of employees who are in the office on a day of week that he is not supposed to. The result should be sorted in ascending alphabetical order.

4. **[3 marks]** Implement a method called `getCloseContacts` (`CovidUtility4.java`). This method takes in two parameters:
    a. `target` (type: `String`): The employee of interest
    b. `entries` (type: `List<Entry>`): Use the same assumptions as Q2 Part 1.

    This method returns all the employees who spent a cumulative of 15 minutes and more in the same office with the `target` on the same day based on the `entries`. The time spent together need not be consecutive. For example, they can spend 7 minutes together in the morning, and 8 minutes in the afternoon.

5. **[3 marks]** Implement `getPeak` (`CovidUtility5.java`). It takes in one parameter:
    a. `entries` (type: `List<Entry>`): Each entry states an entry and exit time of an employee. An employee could enter and exit the building multiple times in a day.
    Note: For the same employee, you can assume that the entries (enter and exit timings) are not overlapping.

    This method returns the highest number of employees in the office at a particular date and time.