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.

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: Jia Gu Wen
 * Email ID: jiagu.wen.2022
 */
```

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. `jiagu.wen.2022`)**

Question 1 [5 marks]

A number A contains another number B if all the digits in number B appear in the same order consecutively in number A. For example:

1. The numbers **12**, 3**12**4, **12**4, and 3**12** contain the number 12.
2. The numbers 21, 132 do not contain the number 12.

Implement a static method named `contains` (`Q1.java`). This method takes 2 parameters:

- `nums` (type: `long[]`): This is the list of numbers.
- `target` (type: `long`): This is a positive number.

This method will return all the numbers in `nums` whereby it contains the `target`.

Given:

1. **the class diagram in the appendix,**
2. **the API documentation in the api folder.**

Implement the following static methods.

Question 2 [4 marks]

Implement a static method called `getCommonAttendees(Q2.java)`. It takes in two parameters:

a. `meeting1` (type: `Meeting`): The first scheduled meeting. Assume not `null`.
b. `meeting2` (type: `Meeting`): The second scheduled meeting. Assume not `null`.

It returns the list of employees who will be attending both meetings (`meeting1` and `meeting2`).
The output is sorted by the employee's name in ascending order.

Question 3 [4 marks]

Implement a static method called `getMeetingIdsOfIndividuals (Q3.java)`. It takes in 1 parameter:

a. `meetings` (type: `Meeting[]`): This is an array of all scheduled meetings. Assume not `null`.

This method returns a map where the

a. key is the employee's name, and
b. value is the set of meeting ids that this employee is attending.

The key is sorted by the employee's name in ascending order, and the value is sorted by the meeting id in ascending order.

Question 4 [4 marks]

Implement a static method called `getConflicts` (`Q4.java`). It takes in 1 parameter:

a. `scheduled` (type: `List<Meeting>`): This is the list of meetings planned for an employee. Assume not `null`.

This method returns all the `Conflicts` in `scheduled`. A conflict occurs when two meetings overlap with each other. For example, (22/12/2022 10:00 - 11:00) and (22/12/2022 10:30 - 11:00) have an overlapping time period from 10:30 - 11:00.

Question 5 [4 marks]

Implement a static method called `getTopMeetingCompanions(Q5.java)`. It takes in 1 parameter:

a. `meetings` (type: `List<Meeting>`): This is an array of all the meetings. It contains at least 1 element and is not `null`.

This method returns a map where the

- key is the employee's name
- value is the names of the employees who went into the same meeting **for the highest number of times regardless of meeting duration**.

Note:

1. The key is sorted by the employee's name in ascending order.
2. The value is sorted by the employee's name in ascending order.

Example 1:

The table below shows the meetings attended by each employee:

	Employee A	**Employee B**	**Employee C**	**Employee D**
Meeting 1		X	X	
Meeting 2	X			X
Meeting 3		X	X	
Meeting 4	X	X	X	X

This method returns `{B=[C], C=[B]}`.

- B and C attended the 3 meetings together (the highest). The rest only attended at most 2 meetings together. Thus, B is the top meeting companion of C, and C is the top meeting companion of B.

	Employee A	**Employee B**	**Employee C**	**Employee D**
Employee A		1	1	2
Employee B	1		**3**	1
Employee C	1	**3**		1
Employee D	2	1	1	

Example 2:
The table below shows the meetings attended by each employee:

	Employee A	Employee B	Employee C	Employee D
Meeting 1			X	X
Meeting 2	X	X		
Meeting 3		X		X
Meeting 4	X	X	X	X

This method returns `{A=[B], B=[A, D], C=[D], D=[B, C]}`. The table below shows the number of meetings each pair of employees went in together. The highest number of meetings attended together is 2.

	Employee A	Employee B	Employee C	Employee D
Employee A		**2**	1	1
Employee B	**2**		1	**2**
Employee C	1	1		**2**
Employee D	1	1	**2**	

Question 6 [4 marks]

Implement a static method called `getFreeTimeSlots` (`Q6.java`). It takes in 4 parameter:

b. `busySlots` (type: `List<Timeslot>`): This is the list of busy time slots (ignoring work hours) of an employee over multiple days.
   i. You can assume that the time slots are non-overlapping. You will not have time slots like (12/12/2022 10:00 - 11:00) and (12/12/2022 10:30 - 11:00) with an overlapping time period from 12/12/2022 10:30 - 11:00.
   ii. A time slot will not exceed 24 hours.
   iii. A time slot may start and end on different days. For example, 12/12/2022 23:00 - 13/12/2022 02:00).

c. `targetDate` (type: `ScisTime`): This is the date that we are looking for free `Timeslot`s.

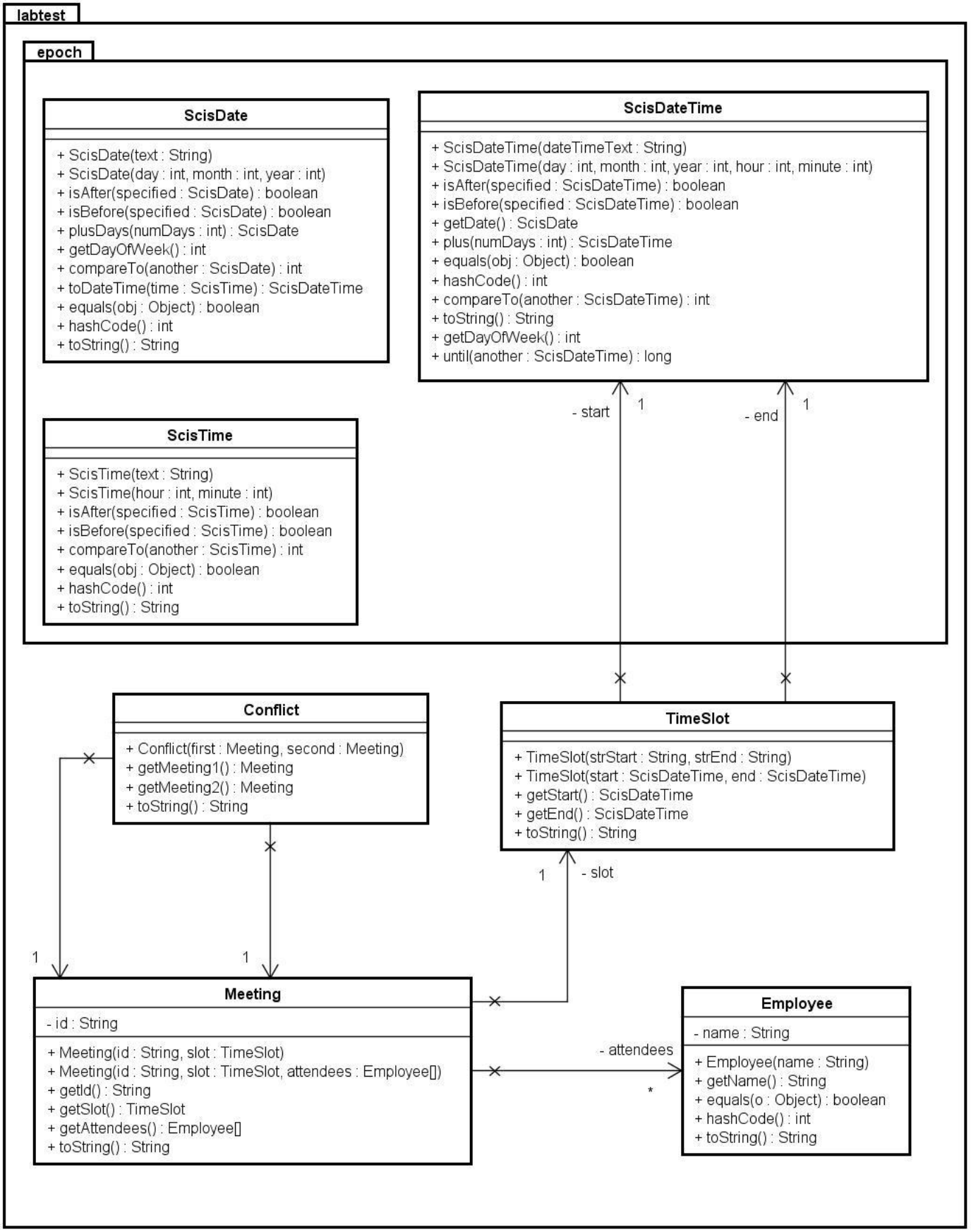
b. `workStart` (type: `ScisTime`): This is the start of a work day,

c. `workEnd` (type: `ScisTime`): This is the end of a work day. `workStart` and `workEnd` will be on the same work day.

This method returns the **free** (not busy) `Timeslot`s of the employee on the `targetDate` between `workStart` and `workEnd`. The result is sorted by the `TimeSlot`'s `start` time in ascending order.

Note: You can assume all the variables are not `null`.

APPENDIX


labtest
epoch
ScisDate
+ ScisDate(text : String)
+ ScisDate(day : int, month : int, year : int)
+ isAfter(specified : ScisDate) : boolean
+ isBefore(specified : ScisDate) : boolean
+ plusDays(numDays : int) : ScisDate
+ getDayOfWeek() : int
+ compareTo(another : ScisDate) : int
+ toDateTime(time : ScisTime) : ScisDateTime
+ equals(obj : Object) : boolean
+ hashCode() : int
+ toString() : String
ScisDateTime
+ ScisDateTime(dateTimeText : String)
+ ScisDateTime(day : int, month : int, year : int, hour : int, minute : int)
+ isAfter(specified : ScisDateTime) : boolean
+ isBefore(specified : ScisDateTime) : boolean
+ getDate() : ScisDate
+ plus(numDays : int) : ScisDateTime
+ equals(obj : Object) : boolean
+ hashCode() : int
+ compareTo(another : ScisDateTime) : int
+ toString() : String
+ getDayOfWeek() : int
+ until(another : ScisDateTime) : long
ScisTime
+ ScisTime(text : String)
+ ScisTime(hour : int, minute : int)
+ isAfter(specified : ScisTime) : boolean
+ isBefore(specified : ScisTime) : boolean
+ compareTo(another : ScisTime) : int
+ equals(obj : Object) : boolean
+ hashCode() : int
+ toString() : String
- start
1
- end
1
Conflict
+ Conflict(first : Meeting, second : Meeting)
+ getMeeting1() : Meeting
+ getMeeting2() : Meeting
+ toString() : String
TimeSlot
+ TimeSlot(strStart : String, strEnd : String)
+ TimeSlot(start : ScisDateTime, end : ScisDateTime)
+ getStart() : ScisDateTime
+ getEnd() : ScisDateTime
+ toString() : String
1
- slot
1
1
Meeting
- id : String
+ Meeting(id : String, slot : TimeSlot)
+ Meeting(id : String, slot : TimeSlot, attendees : Employee[])
+ getId() : String
+ getSlot() : TimeSlot
+ getAttendees() : Employee[]
+ toString() : String
- attendees
*
Employee
- name : String
+ Employee(name : String)
+ getName() : String
+ equals(o : Object) : boolean
+ hashCode() : int
+ toString() : String