Lab Test 1 [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. Disable the following on your laptop before the test begins: WIFI, Bluetooth, and any other communication devices (e.g. 3G/4G modems).
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.

1. 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.
2. Do not hardcode. We will use different test cases to test and grade your solutions.

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

1. 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.2014, include the following block of comments at the beginning of each source file (.java) you write.

```
/*
 * Name: Sotong Tan
 * Email ID: sotong.tan.2019
 */
```

2. 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.

3. 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 marked.

Question 1 [7 marks]

1. Update `ElectronicBook` according to the class diagram.

   a. The price of an `ElectronicBook` is $\frac{7}{9}$ of a Book's price. Discard all decimal places.

   b. An `ElectronicBook` A is considered the same as `ElectronicBook` B if and only if
      i. The `Book`'s `equals` method returns `true`, AND
      ii. The `expiryDate` of A is the same as `expiryDate` of B.

      Note: The `Book`'s `equals` implementation may change in the future.

   c. An `ElectronicBook` is compared against another in the following order:
      i. the title
      ii. the authors.
         - Sort the list of authors in ascending order before comparing them
         - Compare the names of the first author in both `ElectronicBook`s. If they are equal, compare the 2nd author. Likewise, complete all pairs one by one.
         - As you compare the pairs of authors, if one list is exhausted, that shorter list should appear first.
      iii. the expiry dates.

      It returns a negative integer, zero, or a positive integer if this object is "less than, equal to, or greater than" another specified object.

   d. The toString method returns a String of the format:
      <title> : <expiry date in ddMMyyyy format>

      For example, "`Apple : 12122019`".

The following test classes are given:

1. `ConstructorTest`
2. `CompareToTest`
3. `EqualsTest`
4. `GetPriceInCentsTest`
5. `ToStringTest`

Question 2 [9 marks]

Implement the following methods:
Note : you can assume that `List<Course>` has different courses that are offered.

1. **[2 marks]** In `Utility1.java`, implement `getSMUXSections`: It takes in one parameter, courses (type: `List<Courses>`). It returns a map whereby the key is the `Course`'s name, and the value is a `List` of `Section`s for that course.

2. **[2 marks]** In `Utility2.java`, implement `retrieveLowEnrolmentCourses`: It takes in one parameter, courses (type: `Courses[]`). It returns an array of `Course`s where $\frac{1}{3}$ or more of the sections has enrolment number that is less than half of the section's capacity.

3. **[2 marks]** In `Utility3.java`, implement `removeDisqualifiedSMUXSections`: It takes in one parameter, courses (type: `List<Courses>`). It removes all the SMUX sections from the `List<Courses>` that do not have a `Sponsor`.

4. **[3 marks]** In `Utility4.java`, implement `getShortFall`: It takes in two parameters:
    a. `courses` (type: `List<Courses>`)
    b. `demand`(type: `Map<String, Integer>`). The key is the course name, the value is the number of students who requested for the course.

    This method will check if the course is offered and if there are sufficient vacancies (i.e. capacity - enrolment number) in the courses (type: `List<Course>`) and returns the shortfall required for all courses. The return data type is a Map<String, Integer> where the key is the course name, and the value is the shortfall number for the course.

Question 3 [5 marks]

Minesweeper is a single-player puzzle computer game. The objective of the game is to clear a rectangular board containing hidden bombs without detonating any of them, with the help from clues about the number of neighbouring mines in each field. You are given a 2D char array to represent the game board. The board size is not fixed (i.e. the number of rows and columns may vary).

Before the cell is revealed, each cell is represented either by:

- 'M': Represents a MINE
- 'E': Represents an EMPTY cell, i.e. contains no mine

You are to implement the `click` method that simulates a player's move.

For example, if the board is this:

	0	*1*	*2*	*3*	*4*
0	M	E	E	E	E
1	E	E	M	E	E
2	E	E	E	E	E
3	E	E	E	E	E

1. If the player "clicks" on a square containing a mine, the cell is marked with an 'X' (i.e. the bomb is detonated). For example, if the player's move is (0,0), then the board should be updated to:

	0	*1*	*2*	*3*	*4*
0	X	E	E	E	E
1	E	E	M	E	E
2	E	E	E	E	E
3	E	E	E	E	E

2. If the player "clicks" on a square where no mine is revealed, a digit ('1' - '8') should be displayed in the square, indicating how many adjacent squares contain mines. For example, if the player's move is (1,1), then the board should be updated to:

	0	*1*	*2*	*3*	*4*
0	M	E	E	E	E
1	E	2	M	E	E
2	E	E	E	E	E

3	E	E	E	E	E

3. If the player "clicks" on a square where no mines are adjacent, the square becomes blank (i.e. 'B'), and all adjacent squares will be recursively revealed. For example, if the player's move is (3,0), then the board should be updated to:

Board State (before click):

	0	*1*	*2*	*3*	*4*
0	M	E	E	E	E
1	E	E	M	E	E
2	E	E	E	E	E
3	E	E	E	E	E

Board State (after click):

	0	*1*	*2*	*3*	*4*
0	M	E	E	1	B
1	1	2	M	1	B
2	B	1	1	1	B
3	B	B	B	B	B

Note:

1. You could be given a board of different dimensions (e.g. 4 rows and 10 columns).

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	M	E	E	1	B	B	B	B	B	B
1	1	2	M	1	B	B	B	B	B	B
2	B	1	1	1	B	B	B	B	B	B
3	B	B	B	B	B	B	B	B	B	B

2. The number of bombs on the board varies.

3. You could be given a board where the player has made some moves. For example,

	0	*1*	*2*	*3*	*4*
0	M	E	E	1	B
1	1	2	M	1	B
2	B	1	1	1	B
3	B	B	B	B	B

a. If the player's move is (0, 2), then the board will be updated to

	0	*1*	*2*	*3*	*4*
0	M	E	1	1	B
1	1	2	M	1	B
2	B	1	1	1	B
3	B	B	B	B	B

b. If the player's move is (1, 0), then the click method does nothing.
Reason: The cell has been revealed.

Question 4 [4 marks]

You are given the following directory structure and you are not supposed to reorganize the files.
The `App.java` contains the `main()` method to test the above classes.

1. Organize the source files in the source folder according to the class diagram so that your `compile.bat` and `run.bat` will work correctly.

2. Update the Java source files to include the correct package and import statements with the help of the class diagram.

3. Write a one-liner `compile.bat` such that classes are compiled to folder `output`.

4. Write a one-liner `run.bat` to run the `main` method in `App`.

```
Q4\
 |- data
     |- order.xlsx
 |
 |- jackson-lib
 |   |- jackson-annotations-2.10.0.jar
 |   |- jackson-core-2.9.9.jar
 |   |- jackson-databind-2.9.9.3.jar
 |
 |- poi-lib
 |   |- poi-4.1.1.jar
 |   |- poi-ooxml-4.1.1.jar
 |   |- poi-ooxml-schemas-4.1.1.jar
 |   |- lib\
 |       |- commons-codec-1.13.jar
 |       |- commons-collections4-4.4.jar
 |       |- commons-compress-1.19.jar
 |   |- ooxml-lib\
 |       |- xmlbeans-3.1.0.jar
 |
 |- output
 |   |- <your generated classes here>
 |
 |- src\
 |   |- q4
 |       |- App.java
 |       |- domain\
 |           |- Product.java
 |
 |- compile.bat
 |- run.bat
```

If run.bat runs successfully, the console output is as follows:

```
[ {
  "name" : "orange",
  "quantity" : 3,
  "price" : 0.5
}, {
  "name" : "pear",
  "quantity" : 5,
  "price" : 1.1
} ]
```