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. 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 up to 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.2020, include the following block of comments at the beginning of each source file (.java) you write.

```
/*
 * Name: Sotong Tan
 * Email ID: sotong.tan.2020
 */
```

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 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.

Question 1 [8 marks] [Difficulty: *]

Implement a class called `WebColor`. The class consists of

1. 3 attributes:
    a. `red` (type: `int`): this is a number between 0 (inclusive) to 255 (inclusive).
    b. `green` (type: `int`): this is a number between 0 (inclusive) to 255 (inclusive).
    c. `blue` (type: `int`): this is a number between 0 (inclusive) to 255 (inclusive).
2. 2 constructors:
    a. the first constructor takes in 3 **integer** attributes and initializes the 3 attributes.
    b. the second constructor takes in a hexadecimal **string** and initializes the 3 attributes. It starts with a pound sign ('#') followed by 6 hexadecimals ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', and 'F'). The first 2 letters/numbers refer to the red color value, the next two refer to green color values, and the last two refer to blue. For example, '#FFEE00' where 'FF' is used to represent red, 'EE' is used to represent green and '00' is used to represent blue.

       The color values are defined in values between 00(inclusive) and FF(inclusive) which is the hexadecimal (**base 16**) of 0 to 255.
       Numbers are used when the value is 1-9. Letters are used when the value is higher than 9.
       For example,
        - A represents 10
        - B represents 11
        - C represents 12
        - D represents 13
        - E represents 14
        - F represents 15

       If the constructor is invoked with a hexadecimal string '#FFEE00', it should initialize red to 255 ('FF'), green to 238 ('EE') and blue to 0 ('00'). 'FF' is said to be the color value of red, 'EE' is said to be the color value of blue.

       To convert a hexadecimal string to numerical value in base 10, you can use the `parseInt` method (of the **java.lang.Integer** class) passing to the method the corresponding color value(e.g. `'FF'`) and `16` (hexadecimal's base is 16).

       Note that if the input parameter values given to the constructor are not valid values, it throws **IllegalArgumentException**. Refer to the Test Cases provided to you for details.

3. 3 methods:
    a. `add`: This takes in another `WebColor` object and combines the two colors using the following formula:

$$newRed = \frac{red_1 * red_2}{255}$$

$$newGreen = \frac{green_1 * green_2}{255}$$

$$newBlue = \frac{blue_1 * blue_2}{255}$$

This method returns a new `WebColor` object.

b. `toHexString:` returns the hexadecimal string representation of the color. You can use the `toHexString` method (of the `java.lang.Integer`) class.

c. `toString`: returns the string representation in the format `"<red>:<green>:<blue>"`. For example, `"255:238:0"`.

Use `WebColorTest` to test your code.

Question 2 [8 marks]

You are given the following:

1. **Employee.class**
2. **Project.class**
3. **Administrator.class**

1. **[Difficulty: **]** Implement `ProjectManager` according to the class diagram.

   a. Refer to the class diagram and implement constructors, getters.

   b. Bonus is calculated based on 1% of the worth of all completed projects handled by this `ProjectManager`. Observe the attributes `worth` and `completed` in the class diagram for Project.

   c. Methods `addProject` and `removeProject` are used to add / remove a project to the list of projects handled by the `ProjectManager` .

   d. Method `getProjectWorth` returns an int value computed by summing up the "worth" amount of all completed projects handled by the Project Manager.

   e. Implement `toString()` to return the string representation of Project Manager in the format
   ```
   Name: <name>, type: Project Manager, JoinedDate <year><month>,
   projects worth <completed projects amount>
   ```

   f. Two `ProjectManager`s are said to be equal only if

      i. The `Employee`'s `equals` method returns `true`, AND

      ii. The `Project`s handled by both are the same. Two `Project`s are considered equal if the `equals` method returns `true`.

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

   Use `ConstructorTest`, `GetSetTest` and `EqualsTest` to test your code.

2. **[Difficulty: ***]** Implement `EmployeeComparator` such that it compares two `Employees` by the conditions in the following order:

   a. class type in the following order (`ProjectManager, Administrator)`.

   b. the projects' worth in **descending** order if both are `ProjectManagers` (1000, 500, .. , 0).

   c. employees' name in **ascending** order (e.g. 'a' before 'b', 'ab' before 'ac') .

   d. joinedDate in **ascending** order (Jan before Feb, …, Nov before Dec, and 1982 before 1983, 2019 before 2020).

   Use `ComparatorTest` to test your code.

Question 3 [6 marks] [Difficulty: **]

In a social network, friendships are represented as a directed graph. Given the persons A, B and C, and the following relationships:

1. A considers B as a friend, and
2. B considers A as a friend, and
3. B considers C as a friend

Note: A and B have mutual friendship whereas B and C do not, because C does not consider B as his friend and hence friendship is considered to be unidirectional.

We can represent this as an adjacency matrix. It is a 2-dimensional array with the same number of rows and columns.

	A	**B**	**C**
A	0	1	0
B	1	0	1
C	0	0	0

Each row represents the friends considered by the person. The highlighted second row represents the relationships listings numbered 2. (B considers A as a friend) & 3. (B considers C as a friend) above. If B considers the person as a friend, the value in the cell will be a 1, otherwise 0.

All intersections (A & A, B & B, C & C) will have the value of 0.

In the code, we will use an additional array to store the names.

```
String[] names = {"A", "B", "C"};
int[][] friends = { {0, 1, 0},
                    {1, 0, 1},
                    {0, 0, 0} };
```

- The row corresponding to the index of a person in the `names` array represents the person and who he considers as friends.
- The column corresponding to the index of a person in the `names` array represents whether he is considered as a friend by the person in the corresponding row.
- For example,
  - Row with index 1 represents B's friendship (B considers A & C as friends).
  - Column with index 1 shows that B is considered by A as a friend.

If we were to indicate acquaintances of 5 persons A, B, C, D, E. you would be given a 5 x 5 matrix.

Implement the following static methods in `MatrixRepresentation` class. You can test the methods using the class `MatrixTest` class

1. **[2 marks]** Implement `countFriends`**:** It takes in three parameters:
    a. `ary` (type: `int[][]`): the adjacency matrix.
    b. `names` (type: `String[]`): the names of all the persons, and the index used as the row and column in the adjacency matrix, `ary`.
    c. `target`(type: `String`): the person whose friends we want the method to count.

    It returns an int value that counts the number of friends the parameter `target` has from `ary` (the adjacency matrix).

2. **[2 marks]** Implement `commonFriends`**:** It takes in four parameters:
    a. `ary` (type: `int[][]`): the adjacency matrix.
    b. `names` (type: `String[]`): the names of all the persons, and the index used as the row and column in the adjacency matrix, `ary`.
    c. `person1` (type: `String`)
    d. `person2` (type: `String`)

    The method returns an array (type: `String[]`) representing the friends who are common friends of the given input parameters, `person1` and `person2` from the input array `ary`. If there are no common friends, this method returns an empty array.

3. **[2 marks]** Implement `getSociableStats`**:** It takes two parameters:
    a. `ary` (type: `int[][]`): the adjacency matrix.
    b. `names` (type: `String[]`): the names of all the persons, and the index used as the row and column in the adjacency matrix, `ary`.

    This method returns a data type `Map<Integer, List<String>>` where
    - the **key** is the number of friends a person has, and
    - the **value** is the list of all the persons' names with the same number of friends.

Question 4 [3 marks] [Difficulty: **]

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

```
Q4\
 |- lib\
 |   |- log4j\
 |   |     |- log4j-api-2.14.0.jar
 |   |     |- log4j-core-2.14.0.jar
 |   |     |- log4j-iostreams-2.14.0.jar
 |   |     |- log4j-slf4j-impl-2.14.0.jar
 |   |
 |   |- thymeleaf\
 |   |     |- attoparser-2.0.4.RELEASE.jar
 |   |     |- ognl-3.1.12.jar
 |   |     |- thymeleaf-3.0.9.RELEASE.jar
 |   |     |- javassist-3.20.0-GA.jar
 |   |     |- slf4j-api-1.7.25.jar
 |   |     |- unbescape-1.1.5.RELEASE.jar
 |
 |- conf\
 |   |- log4j2.xml
 |
 |- generated\
 |   |- <your generated classes here>
 |- templates\
 |    |- template.html
 |- src\
 |    |- ...
 |- compile.bat
 |- run.bat
```

Note:

1. The Thymeleaf template engine has a dependency on the log4j framework(i.e. jars in the log4j folder).

2. The configuration file(`log4j2.xml`) in the `conf` folder **needs to be on the classpath during execution**.

3. If the dependencies (stated in #1 and #2) are missing, the application will show the following warning message during execution:

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details
```

4. The jar files in the `thymeleaf` folder contain the classes of the Thymeleaf template engine.

Instructions:

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 `generated`. If you get any warning while compiling, add the suggested option in compile.bat.

   Note: Do not include unnecessary folders/jars in the sourcepath/classpath.

```
warning: Implicitly compiled files were not subject to annotation processing.
  Use -proc:none to disable annotation processing or -implicit to specify a policy
for implicit compilation.
```

   For example, to include this option while compiling `Test.java`:

```
javac -proc:none Test.java
```

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

   Note: Do not include unnecessary folders/jars in the sourcepath/classpath.