CS102 Lab Test 1 [20 marks]

General Instructions:

1. **You are not allowed to communicate in any way during the test. Any violation of this instruction will result in a zero score for your lab test.**

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

3. You can refer to any files on your laptop.

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

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. 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.

7. Tools that enable AI pair programming (e.g., GitHub co-pilot) or pair programming (Live Share) are not allowed.

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.2022`**)**

10. **Include your name as author** in the comments of all your submitted source files. For example, if your registered name is "Sotong TAN", include the following block of comments at the beginning of each source file (.java) you write.

```
/*
 * Name: Sotong Tan
 */
```

11. DO NOT change the signature (parameter and return type) of the methods and DO NOT add in unnecessary packages.

12. You can add additional private static methods within the given .java files to solve the questions.

Question 1 [12 marks]

In **the folder <your email ID>\Q1\**. Insert your name and email ID to the top of

1. Q1a.java,
2. Q1b.java,
3. Q1c.java,
4. Q1d.java,
5. Q1e.java

For all parts in Q1.java, assume

- **`inputs/input` will not be null and empty ArrayList**
- **Strings in `inputs/input` will contain non null words**
- **Strings in `inputs/input` will only contain empty String or single words. (For e.g. it will not include "ice cream" as <u>one</u> word in `inputs`.)**

(a) **[Difficulty : *]** In Q1a.java, implement the static method named `getSumOfAsciiValue`. **[2 marks]**

- takes in 1 parameter:
  - o `inputs` (type: `ArrayList<String>`): .
- return
  - o `ArrayList<Integer>` containing the sum of ASCII values of the characters for each String in `inputs`.
  - o 'A' and 'a' has different Ascii values and the sum of ASCII values should only include the **alphanumeric characters** only, i.e. ('A' - 'Z', 'a' - 'z' and '0' - '9').
  - o 'A' has the ASCII value of 65, 'a' has the ASCII value of 97. '0' has the value of 48.

For example, if `inputs` is `["a", "b", "ab", "a@b+!!"]` , the return value is

`[97, 98, 195, 195]`

Note:

1.	"a":	'a' is 97.
2.	"b":	'b' is 98.
3.	"ab":	97 ('a') + 98 ('b') = 195.
4.	"a@b+!!":	97 ('a') + 98 ('b') = 195.

'@', '+', '!' are excluded as they are non-alphanumeric characters.

Refer to Q1a.java for more test cases.

(b) **[Difficulty : *]** In Q1b.java, implement the static method named `getLongestPalindromeWord`.
[2 marks]

- takes in 1 parameter:
    - o `inputs` (type: `ArrayList<String>`)
- return
    - o the longest palindrome string in `inputs`. A palindrome is a word whereby the sequence of characters are the same when it's read forward or backward.
    - o If there are multiple longest Palindrome strings, choose the one that appears first in `inputs`.
    - o If there is no palindrome, return `null`.

For example,
if `inputs` is `["Madam", "wow", "success", "kayak"]`, the return value is `"Madam"`
Refer to Q1b.java for more test cases.

(c) [Difficulty : *] In Q1c.java, implement the static method named `getMostCommonVowel`.
[2 marks]

- takes in 1 parameter:
    - o `inputs` (type: `ArrayList<String>`)
- return
    - o `ArrayList<Character>` containing the **vowel(s) (in any order)** that appears in the most number of words.
    - o Vowels refer to the 5 alphabets (a/A, e/E, i/I, o/O, u/U).
    - o If there is no word with vowels, return an empty list.

For example, if `inputs` is `["Abate", "Facet", "Pen", "idiom"]`, the return value is `[e]`.

Note:

1. `'a'` appears in 2 words: `["`**`A`**`b`**`a`**`te", "F`**`a`**`cet", "Pen", "idiom"]`.
2. `'e'` appears in 3 words: `["Abat`**`e`**`", "Fac`**`e`**`t", "P`**`e`**`n", "idiom"]`.
3. `'i'` appears in 1 word: `["idiom"]`.
4. `'o'` appears in 1 word. `["idiom"]`.
5. `'u'` appears in 0 words.

Refer to Q1c.java for more test cases.

(d) **[Difficulty : **]** In Q1d.java, implement the static method named `getIncreasingCharWords`. **[3 marks]**

- takes in 1 parameter:
  - o `inputs` (type: `ArrayList<String>`)
- return
  - o `ArrayList<String>` containing the words. Every character within the word is the same or a later alphabet than the previous character from the alphabetical sequence (**case-insensitive**). 'a' (or 'A') < 'b' (or 'B') < … < 'y' (or 'Y') < 'z' (or 'Z').
  - o You can assume that Strings in `inputs` will only contain alphabets.
  - o If there is no such word, return an empty list.

For example,
if `inputs` is `["access", "AcT", "bUy", "Bird"]`, the return value is `["access", "AcT", "bUy"]`.

Note:

1. "access" fulfills the criteria because 'a' <= 'c' <= 'c' <= 'e' <= 's' <= 's'.
2. "Act" fulfills the criteria because 'A' <= 'c' <= 't'.
3. "bUy" fulfills the criteria because 'b' <= 'U' <= 'y'.
4. "Bird" does *not* fulfills the criteria because **'r' >= 'd'**.

Refer to Q1d.java for more test cases.

(e) **[Difficulty : ***]** In Q1e.java, implement the static method named `getTermGPA`. **[3 marks]**

- takes in 1 parameter:
  - o `input` (type: `String`): This is a string of the following format <letter1><unit1><letter2><unit2> … <letterN><unitN> where the
  - ● letter is the grade letter obtained by the student. For example, A+, A, A-
  - ● unit is the number of course units.

  For example, if the input is "A+0.5ABC2", then the student
  1. took 4 courses, and
  2. scored A+ for 1 course of half (0.5) unit,
     Note: if unit value is missing, it is assumed to be of 1 course unit.
  3. scored A and B for 2 courses each of 1 unit,
  4. C for 1 course of 2 units.

- returns
  - o a `double` containing the termGPA of the student. The term GPA is the average grade point over all the courses the student has taken this term. It can be computed using the following formula: In the example above, the student termGPA is approximately 2.922222. Do not perform any rounding of values.

$$\frac{letter_1 \times unit_1 + letter_2 \times unit_2 + \dots\dots\dots\dots + letter_n \times unit_n}{unit_1 + unit_2 + \dots\dots\dots\dots + unit_n}$$

You are given the following:

Grade letter	**Grade Point**
A+	4.3
A	4
A-	3.7
B+	3.3
B	3.0
B-	2.7
C+	2.3
C	2
C-	1.7
D+	1.3
D	1.0
F	0

Refer to Q1e.java for more test cases.

Note: **DO NOT HARDCODE** your methods. We will test your implemented methods using a different `main` method in which different values will be passed into these methods.

IMPORTANT: Keep **Q1a.java, Q1b.java, Q1c.java, Q1d.java and Q1e.java** in the folder <your email ID>\Q1
Only these files will be marked

Question 2 **[8 marks]**

Find the files **Q2a.java, Q2b.java, Q2c.java** in the folder **<your email ID>\Q2\**. Insert your name and email ID to the top of both files.

You are given the API documentation of the following Java classes and their byte code (.class) files.
(The API documentation is located at the folder **<your email ID>\API\** and the class files are inside **<your email ID>\Q2**). Study the API documentation of these classes.

- `Person`
- `Student`
- `School`

Refer to the class diagram for the input parameters and return value and implement the following static methods in **Q2.java**.

(a) **[Difficulty : *]** In Q2a.java, implement the static method named `enrol` **[2 marks]**
- The method will add the student to the first school (having a smaller index in the ArrayList of schools) with one or more vacancies in school.
- Once a student is enrolled into the school, the number of vacancies for the school will be reduced by 1.

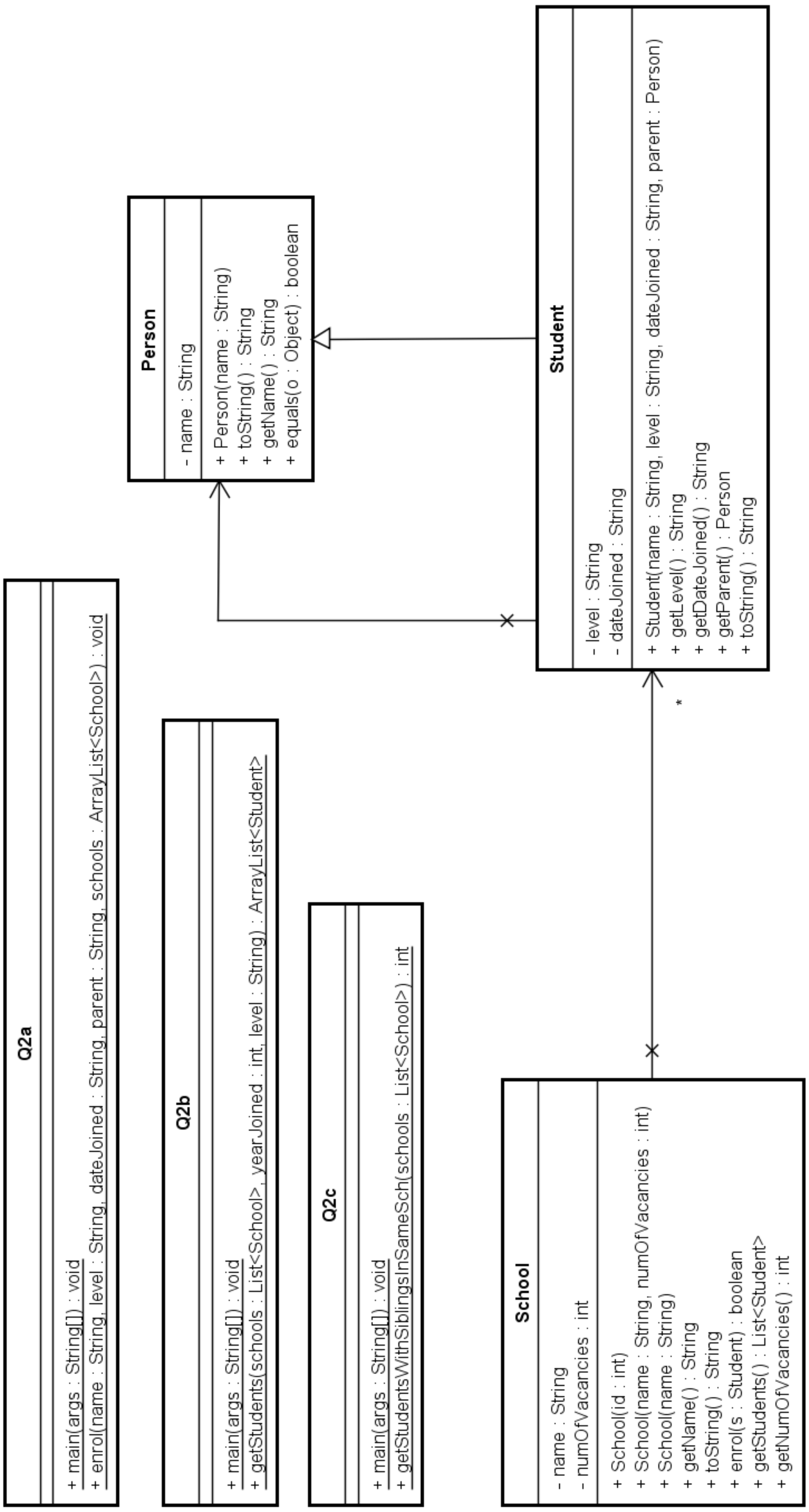
(b) **[Difficulty : **]** In Q2b.java, implement the static method named `getStudents` **[3 marks]**
- The method will return the students **(in any order)** that match the criteria based on the year joined and level.
- If level is not specified (i.e. null), it will return the students that match the criteria based on the year joined.

(c) **[Difficulty : **]** In Q2c.java, Implement the static method named `getStudentsWithSiblingsInSameSch` **[3 marks]**
- The method will return the number of students whereby the student has sibling(s) in the same school (regardless of when the student joined the school)
- A student is considered a sibling with another student if they have the same Parent.
- If student A has a sibling (student B) in the same school, then student B has a sibling (student A) in the same school.

Note: **DO NOT HARDCODE** your methods. We will test your implemented methods using a different `main` method in which different values will be passed into these methods.

IMPORTANT: Keep **Q2a.java, Q2b.java, Q2c.java,**
in the folder <your email ID>\Q2
Only these files will be marked.

APPENDIX A


Person
- name : String
+ Person(name : String)
+ toString() : String
+ getName() : String
+ equals(o : Object) : boolean
Student
- level : String
- dateJoined : String
+ Student(name : String, level : String, dateJoined : String, parent : Person)
+ getLevel() : String
+ getDateJoined() : String
+ getParent() : Person
+ toString() : String
*
School
- name : String
- numOfVacancies : int
+ School(id : int)
+ School(name : String, numOfVacancies : int)
+ School(name : String)
+ getName() : String
+ toString() : String
+ enrol(s : Student) : boolean
+ getStudents() : List<Student>
+ getNumOfVacancies() : int
Q2a
+ main(args : String[]) : void
+ enrol(name : String, level : String, dateJoined : String, parent : String, schools : ArrayList<School>) : void
Q2b
+ main(args : String[]) : void
+ getStudents(schools : List<School>, yearJoined : int, level : String) : ArrayList<Student>
Q2c
+ main(args : String[]) : void
+ getStudentsWithSiblingsInSameSch(schools : List<School>) : int