CS102-Trial Exam (202526T1) [25 marks]

General Instructions:

1. **You are not allowed to communicate in any way during the test. No mobile phones, earphones, smart watch or other AI / WIFI related devices. Any violation of this instruction will result in a zero score for your exam.**

2. You will perform the exam on your personal laptop. **Your screen will be recorded. IDE or Tools** that enable **AI** pair programming (e.g., GitHub co-pilot) or pair programming (Live Share) are **not allowed**. **Any violation of this instruction will result in a ZERO score.**

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

   Failure to do the following will trigger 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.
   a. Grading will be automated; i.e. no partial credit will be given for code.
   b. Grading will use a different set of test data to detect hardcoding.

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. Ensure that all your Java code can compile without 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.

8. Download the source code and rename the root folder in the zip file to your **Email ID (e.g. `jiagu.wen.2025`)**

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

```
/*
* Name     : Jia Gu Wen
* Email ID: jiagu.wen.2025
*/
```

10. Do **NOT**
    a. change the signature (parameter and return type) of the methods and DO NOT add in unnecessary packages.
    b. rename the file name of the Java files.

11. **You can write additional private (static or instance) helper methods and default access helper classes within the given .java files to solve the questions.**

Question 1: [7 marks] String Processing

(a) [Difficulty: *] Implement a static method named `reorderWordsInSentence` (**`Q1a.java`**). [3 marks]

Given a sentence, reorder the words in such a way that each word is sorted alphabetically by its characters, while maintaining the original word positions in the sentence. You should preserve the spaces between words and treat punctuation marks (like commas, periods, etc.) as part of the word itself. However, the space before the first word, and the space after the last word should be removed.

Examples:

1. `Input: "hello world, this     is great!"`
   `Output: "ehllo dlorw, hist     is aegrt!"`
2. `Input: "java programming is fun."`
   `Output: "aajv agrimnoprg is fun."`
3. `Input: "   quick brown fox    "`
   `Output: "cikqu bnorw fox"`

(b) [Difficulty: **] Implement a static method named `stringToDouble` (**`Q1b.java`**). [4 marks]

Your task is to write a method that converts a given string into a double number. This method should parse the string and return the double representation of the numerical characters found in the string according to the following requirements.

- Requirements:
  - The method should ignore any leading whitespace characters before processing the number.
  - The method should recognize an optional initial plus ('+') or minus ('-') sign to indicate the sign of the number.
  - If more than one consecutive non-numeric characters are present (e.g., "+-" or "-+" or "..") at any position of the input string, the method should return 0.0
  - If the string contains a valid scientific notation (e.g., "1.23e4" or "1.23E4"), the method should correctly interpret it. **'e' / 'E'** represents the exponent base 10, so 1.23e4 = 1.23 x 10^4 = 12300.0
  - Assume the number after '**e' / 'E'** must be an integer.
  - Once a non-numeric character (excluding a decimal point or an **'e' / 'E'** for scientific notation) is encountered, stop parsing and return the number parsed up to that point.
  - The method should handle edge cases like empty strings, strings that are only spaces, or strings that contain non-numeric characters only.

Examples:

1. `stringToDouble(" -123.45abc")` should return -123.45
2. `stringToDouble(" -3.14e-2")` equals to -3.14 x 10^{-2} should return -0.0314
3. `stringToDouble("123.45.67")` should return 123.45
4. `stringToDouble(" 3.14E2")` should return 314.0
5. `stringToDouble(" +42.0abc ")` should return 42.0
6. `stringToDouble("")` should return 0.0 (empty string)
7. `stringToDouble("3.14e+3.1")` should return 0.0 (invalid scientific notation as 3.1 is not an integer)
8. `stringToDouble(" +-123.45")` should return 0.0
9. `stringToDouble("e12345")` should return 0.0 (because it starts with 'e' without a number before it)
10. `stringToDouble("12...345")` should return 0.0

Question 2: [8 marks] File Reading

(a) [Difficulty: *] Implement the static method named `getAverageAge` (**`Q2a.java`**). **[4 marks]**

- takes in 2 parameters:
  - `filename` (type: `String`): This is the name of the input file (e.g. persons.txt)
    - An entry in the file would have the following format
      - <Name>-<Age> e.g. John LEE-28
  - `surname` (type: `String`): This is the surname.

- return
  - `double` which is the value of the average age of the people in the input file where the surname matches with the parameter `surname`
  - Assume that a person's surname in the input file can only either be the first word or the last word in a name. (e.g. John LEE, LEE Teck Leong) and the surname are in uppercase.
  - Assume there will always be a surname for each person in the input file.
  - No rounding required for the return value
  - Returns -1.0 if the file doesn't exists.
  - Returns 0.0 if the file doesn't contain a person of that surname

Refer to Q2a.java for more test cases.

(b) **[Difficulty:*]** Implement the static method named `getTopStudent` (**`Q2b.java`**). **[4 marks]**

- takes in 2 parameters:
  - `filename` (type: `String`): This is the name of the input file. (e.g. students.txt)
    - An entry in the file would have the following format
    - <Name>,<Courses> e.g. John LEE,IS101#4.0-IS102#3.0-IS103#2.5
    - John LEE takes 3 courses (IS101 with GPA of 4.0, IS102 with GPA of 3.0, IS103 with GPA of 2.5)
    - The number of courses are dynamic for each student
  - `courseName` (type: `String`): This is the name of the course. (e.g. IS101)

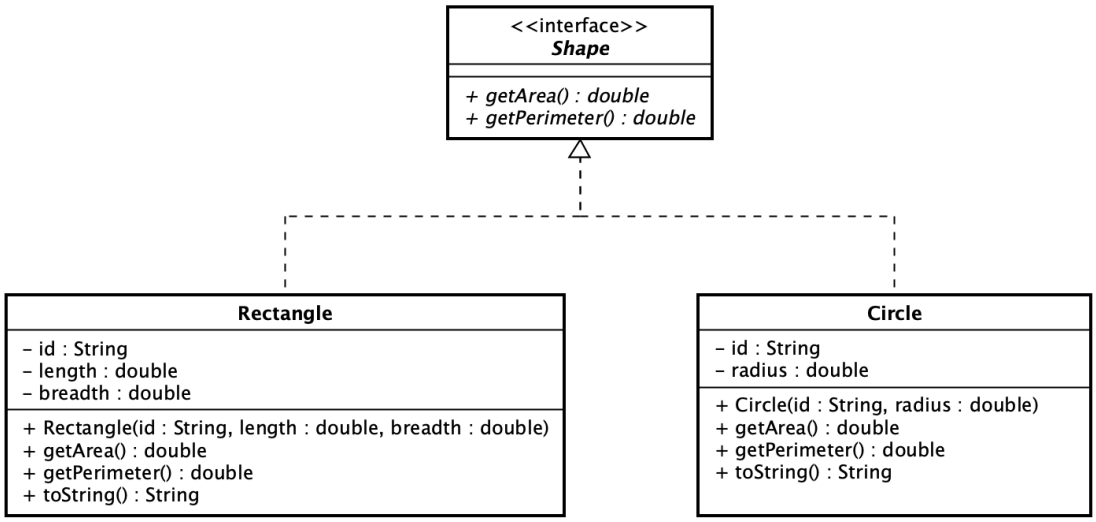
- return
  - `String` which contains the name and gpa of the student and which has the highest GPA where the course matches with the parameter `courseName`
    e.g. John LEE-4.0
  - If there are multiple students having the highest GPA, take the first occurrence.
  - You are given `DataException.class` where `DataException` is a RuntimeException
  - Throws `DataException` if the file doesn't exists.
  - Throws `DataException` if the file doesn't contain a course of that coursename.

Refer to Q2b.java for more test cases.

Question 3: [4 marks] Comparator

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>\Q3\API\index.html** and the class files are inside **<your email ID>\Q3**). Study the API documentation of these classes.

- `Shape, Rectangle, Circle`



[Difficulty:*] In Q3.java, complete the codes in `ShapeComparator.java` which is used in `sortShapes`. **[4 marks]**

- area by ascending order
- perimeter in descending order.

Refer to Q3.java for more test cases.

Question 4: [6 marks] Class Path

Refer to the class diagram in appendix and complete the following:

1. Copy the files(.java, .class) provided in the **resource** folder into their respective folders according to the folder structure (on the next page) and the class diagram in Appendix 1.

   Note:

   a. Please leave a backup copy of your files in the resource folder. We will not provide you with a fresh copy.
   b. Do not reference the resource folder in compile.sh/compile.bat and run.sh/run.bat.

2. Make the relevant changes to the Java source files with reference from the class diagram.

3. Application.java contains the main() method to test the above classes.

4. Write a one-liner in either compile.sh/compile.bat such that the classes are compiled to the **out** folder. Do not write in both compile.sh AND compile.bat.

5. Write a one-liner in either run.sh/run.bat to run the main method in Application.java. Do not write in both run.sh AND run.bat.

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

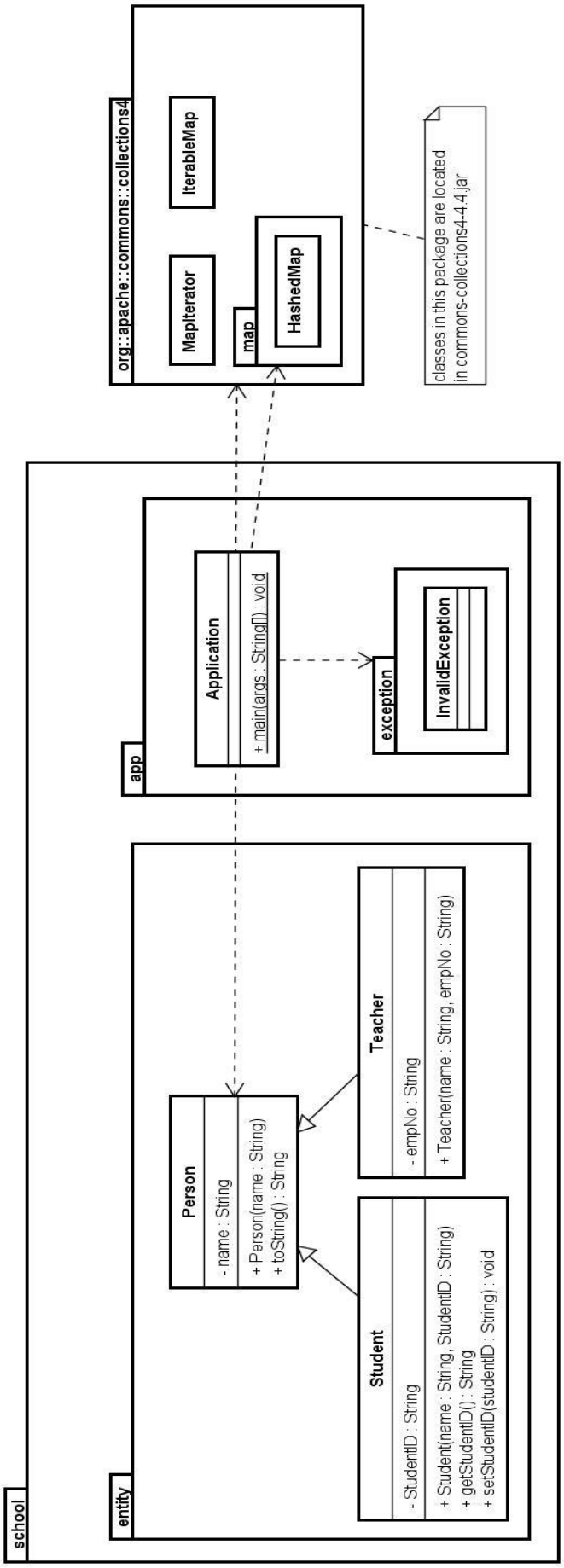
   The directory should look like this:

```
Q4\
 |
 |- out\
 |   |- <your generated files here>
 |
 |- external\
 |   |- apache\
 |       |- commons-collections4-4.4.jar
 |
 |- src\
 |    |-<source files here>
 |
 |- given\
 |    |- <the provided Person.class here>
 |
 |- compile.sh
 |- compile.bat
 |- run.sh
 |- run.bat
```

If `run.bat/run.sh` runs successfully, the following will be displayed on the console

```
2 - Person[name=Peter]
4 - Person[name=Candy]
1 - Person[name=John]
3 - Person[name=Joe]
Number of Persons : 4
```

APPENDIX 1



END