Lab Test [25 marks] 1 hour 30 mins

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 file 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
 */
```

Question 1 [13 marks]

Refer to the class diagram in the appendix and implement the following classes accordingly.

You should only be writing codes in `Utility.java, Person.java, Youth.java, Adult.java, InvalidInfoException.java`. You can write additional helper methods in `Utility.java` if required. Do <u>not</u> modify `App.java`.

A. **[4 marks]** Implement `Person`.

- `toString()` returns a String in the following format
  - name=<name>, age=<age>, medicalCondition=<medicalCondition>, shotType=<shotType>
  - e.g. name=Jane Lim, age=70, medicalCondition=[High Blood, Diabetes], shotType=S

- **[Difficulty: ***]** `compareTo(Person anotherPerson)` is used to sort the Person objects based on the following orders:
  - `shotType` in the following order (F,S,B) **.**
    It has 3 possible values:
    1. 'F' refers to the first vaccine shot.
    2. 'S' refers to the second vaccine shot.
    3. 'B' refers to the first booster shot.
  - number of `medicalCondition` in **descending** order

B. **[1 mark]** Implement `Youth`.

- `toString()` returns a String in the following format
  - Youth [name=<name>, age=<age>, medicalCondition=<medicalCondition>, shotType=<shotType>, school=<school>]
  - e.g. Youth [name=Angie Lau, age=10, medicalCondition=null, shotType=S, school=Princess Secondary School]

C. **[1 mark]** Implement `Adult`.

- `toString()` returns a String in the following format
  - Adult [name=<name>, age=<age>, medicalCondition=<medicalCondition>, shotType=<shotType>, isElderly=<isElderly>]
  - e.g. Adult [name=Ben Tan, age=45, medicalCondition=null, shotType=B, isElderly=false]

D. **[1 mark]** Implement `InvalidInfoException` which is a checked Exception.

E. **[6 marks]** Implement `load(String filename)` in `Utility.java` with the following requirements.

- Reads a text file containing the vaccination records of individuals and stores them in a `Map<String, List<Person>>` where `String` refers to a specific vaccination centre.

- There's a header in each file to describe the format of information stored. The header row will always be available in all test files. The order of the fields will not change.

  ```
  Name:Age:MedicalCondition:School:VaccineCentre:Vaccine:ShotType
  ```

  ":School" is a field that applies only to Youths.

- A person is considered:
  - a Youth if the person is 18 years and below.
  - an Adult if the person is above 18. For Adults that are above 65 years and above, the `isElderly` attribute will be set to true.

  For a Youth, there are 7 fields (including "school" field). For example,

  ```
  Xavier Lee:16:NA:Prince Secondary School:Queenstown CC:Pfizer:S
  ```

  For an Adult, there are 6 fields. There is no "school" field. For example,

  ```
  Ben Tan:45:NA:Bukit Merah CC:Moderna:B
  Jane Lim:70:High Blood,Diabetes:Tiong Bahru CC:Pfizer:S
  ```

  Note: Between MedicalCondition and VaccineCenter fields, there is 1 colon (NOT 2).

- If there is no medical condition, the value NA is given in the text file. If there are more than one medical conditions, the conditions are separated by a comma.

- It will then add the relevant person (Youth or Adult) to the map based on the vaccination centre.

- This method throws `InvalidInfoException`
  - if the input file is missing, or
  - when it **first** encounters a row that contains less than
    - 6 fields for adult records.
    - 7 fields for youth records.

Use `App` to test your code. Refer to the provided output and the exception messages to be displayed below.

WARNING : You must not *modify* App to achieve the given output!

```
*** Test File : "testdata0.txt" ***
 Actual : Exception caught - Missing file - testdata0.txt
 Result : Passed.

*** Test File : "testdata2.txt" ***
 Actual : Exception caught - Missing information at Line 2 - Xavier Lee
 Result : Passed.

*** Test File : "testdata1.txt" ***
  Actual : 3 vaccineCentres loaded
  Result : Passed

------------------------------------------------------------------------
Vaccine Centre : Queenstown CC (4 persons)
Adult [name=John Neo, age=35, medicalCondition=[Hypertension], shotType=F,
isElderly=false]
Youth [name=Shirley Low, age=16, medicalCondition=[High Blood], shotType=S,
school=King Secondary School]
Youth [name=Xavier Chris Lee, age=17, medicalCondition=null, shotType=S,
school=Prince Secondary School]
Adult [name=Jack Yeo, age=25, medicalCondition=null, shotType=B, isElderly=false]
------------------------------------------------------------------------
Vaccine Centre : Tiong Bahru CC (3 persons)
Adult [name=Charles Yang, age=65, medicalCondition=null, shotType=F,
isElderly=true]
Adult [name=Jane Lim, age=70, medicalCondition=[High Blood, Diabetes,
Hypertension], shotType=S, isElderly=true]
Adult [name=Fabian Yong, age=25, medicalCondition=[Diabetes], shotType=S,
isElderly=false]
------------------------------------------------------------------------
Vaccine Centre : Bukit Merah CC (2 persons)
Youth [name=Angie Lau, age=10, medicalCondition=null, shotType=S, school=Princess
Secondary School]
Adult [name=Ben Tan, age=45, medicalCondition=null, shotType=B, isElderly=false]
```

Question 2 [4 marks]

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 2.

   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:

```
q2\
 |
 |- 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
```

Question 3 [8 marks]

You are given the class diagram in the appendix. You can write additional helper methods if required.

Implement the following static methods.

1. **[1 mark]** Implement `getCommonCourses` (`Utility1.java`). It takes in two parameters:
   a. `courseList1` (type: `List<Course>`): the list of courses taken by the first student. This attribute is non-null and is not sorted in any order.
   b. `courseList2` (type: `List<Course>`): the list of courses taken by the second student. This attribute is non-null and is not sorted in any order.

   It returns the list of unique course code (e.g. IS111) of the common courses taken by both students taken in the same or different academic term. The result should be sorted in ascending alphabetical order of the course codes.

2. **[2 marks]** Implement `getListOfPartnerExchangeUniversities` (`Utility2.java`). It takes in one parameter:
   a. `courseList` (type: `List<Course>`): the list of courses taken by all students. This attribute is non-null and is not sorted in any order.

   It returns a map where the
   a. key is the country where the university is located.
   b. value is the list of universities found in the country. **The result need not be sorted in any order.**

3. **[2 marks]** Implement `getLOATerms` (`Utility3.java`). It takes in one parameter:
   a. `cList` (type: `List<Course>`): the list of courses taken by a student taken to date. This attribute is non-null, contains at least 1 element and is not sorted in any order.

   The term attribute for the Course class is in the format AY<4 digit year><2 digit year>T<1 or 2>. For example, "AY202021T1" refers to the academic term 2020-21 term 1.

   There are only 2 regular terms in a year (term 1 and term 2).

   A student can start his first academic term either on term 1 or term 2.

   It returns the LOA terms (i.e. the term where no course is taken) between the first term and the last active term in `cList`. The terms should be sorted in ascending order.

4. **[3 marks]** Implement `getCoursesTakenInSameYearAndSemesterOfStudy`(`Utility4.java`). It takes in two parameters:
    a. `courseList1` (type: `List<Course>`): the list of courses taken by the first student. This attribute is non-null and is not sorted in any order.
    b. `courseList2` (type: `List<Course>`): the list of courses taken by the second student. This attribute is non-null and is not sorted in any order.

    It returns the code of the courses that both students take in the same year(first, second, third or fourth year of study) and semester of study (semester 1 or 2) sorted in ascending order. See explanation below. You can assume that both students did not take any Leave of Absence (LOA).

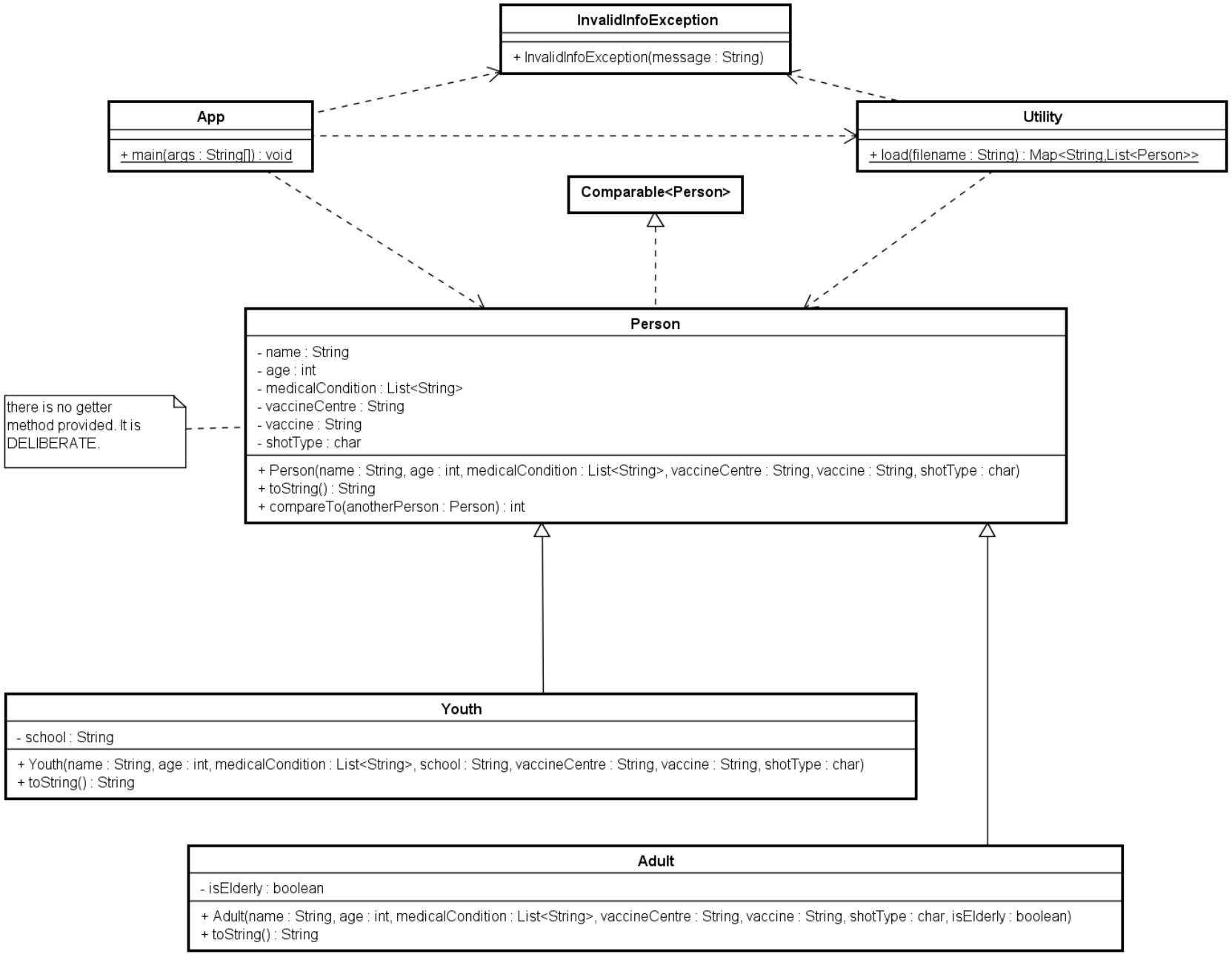
    For example,

Academic Term	**Student A**	**Student B**
AY202021T1	**IS111** (Year 1 of study, Semester 1)	
AY202021T2	IS112 (Year 1 of study, Semester 2)	**IS111**, IS112 (Year 1 of study, Semester 1)
AY202122T1	**IS113** (Year 2 of study, Semester 1)	IS114 (Year 1 of study, Semester 2)
AY202122T2	IS114 (Year 2 of study, Semester 2)	**IS113** (Year 2 of study, Semester 1)

    Both students did
    - IS111 in Year 1 of study, Semester 1 (even though at different academic terms AY202021T1 and AY202021T2)
    - IS113 in Year 2 of study, Semester 2 (even though at different academic terms AY202122T1 and AY202122T2)

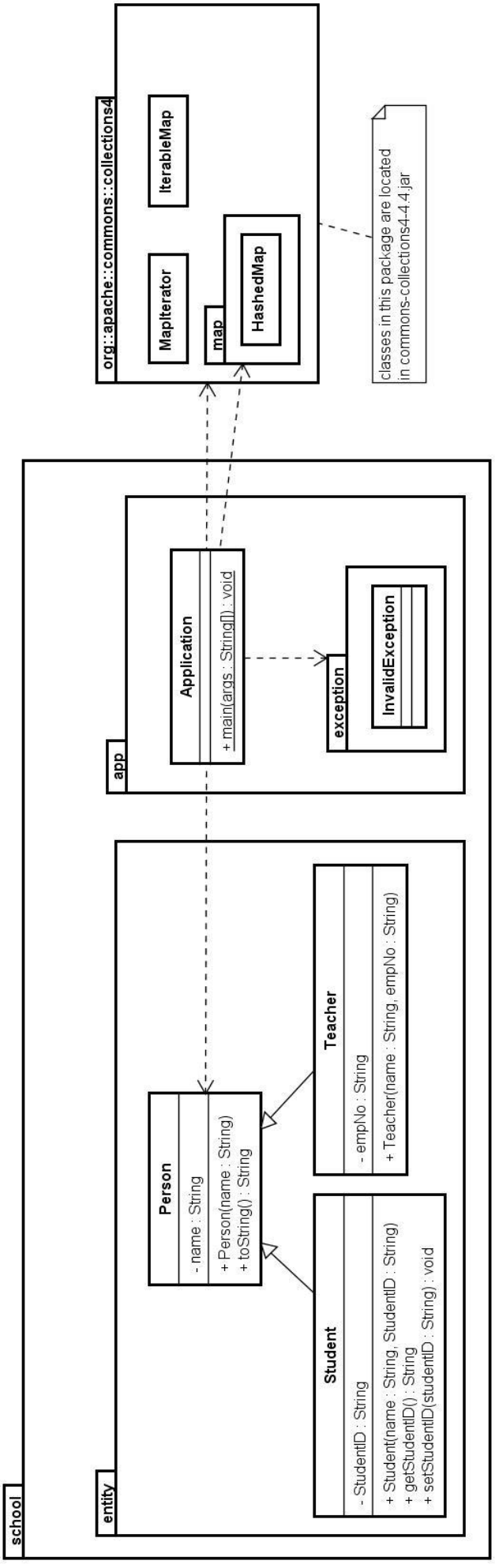
    Thus, IS111 and IS113 are included in the result.

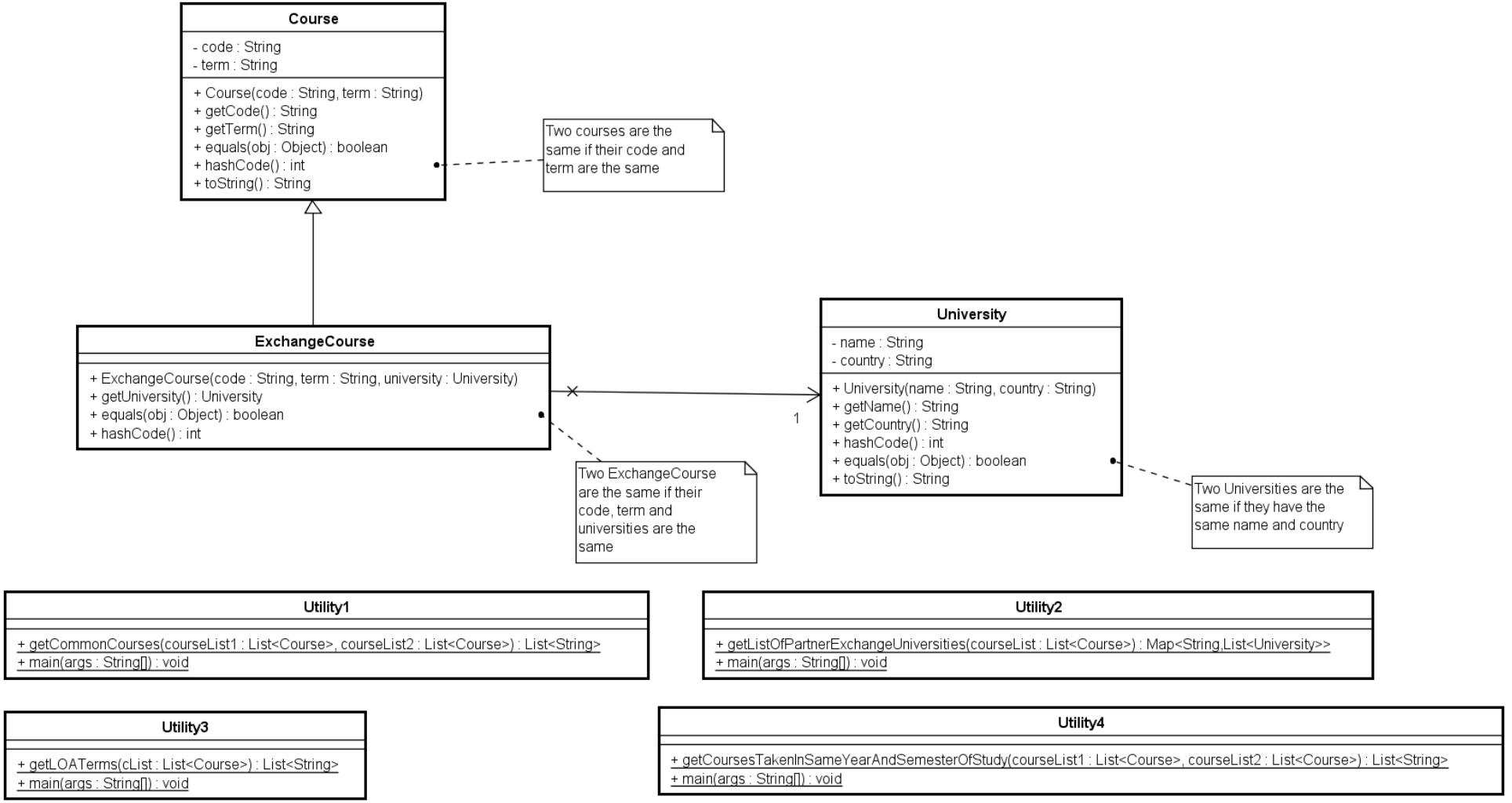
    Both students did IS112 in the same academic term (AY202021T2) but student A did it in his Year 1 of study, Term 2 whereas student B did it in his Year 1 of study, Term 1. Thus, IS112 is not included in the result.


InvalidInfoException
+ InvalidInfoException(message : String)
App
+ main(args : String[]) : void
Utility
+ load(filename : String) : Map<String,List<Person>>
Comparable<Person>
Person
- name : String
- age : int
- medicalCondition : List<String>
- vaccineCentre : String
- vaccine : String
- shotType : char
+ Person(name : String, age : int, medicalCondition : List<String>, vaccineCentre : String, vaccine : String, shotType : char)
+ toString() : String
+ compareTo(anotherPerson : Person) : int
there is no getter method provided. It is DELIBERATE.
Youth
- school : String
+ Youth(name : String, age : int, medicalCondition : List<String>, school : String, vaccineCentre : String, vaccine : String, shotType : char)
+ toString() : String
Adult
- isElderly : boolean
+ Adult(name : String, age : int, medicalCondition : List<String>, vaccineCentre : String, vaccine : String, shotType : char, isElderly : boolean)
+ toString() : String

APPENDIX 1

APPENDIX 2


org::apache::commons::collections4
IterableMap
MapIterator
map
HashedMap
classes in this package are located in commons-collections4-4.4.jar
school
app
Application
+ main(args : String[]) : void
exception
InvalidException
entity
Person
- name : String
+ Person(name : String)
+ toString() : String
Teacher
- empNo : String
+ Teacher(name : String, empNo : String)
Student
- StudentID : String
+ Student(name : String, StudentID : String)
+ getStudentID() : String
+ setStudentID(studentID : String) : void


Course
- code : String
- term : String
+ Course(code : String, term : String)
+ getCode() : String
+ getTerm() : String
+ equals(obj : Object) : boolean
+ hashCode() : int
+ toString() : String
Two courses are the same if their code and term are the same
ExchangeCourse
+ ExchangeCourse(code : String, term : String, university : University)
+ getUniversity() : University
+ equals(obj : Object) : boolean
+ hashCode() : int
Two ExchangeCourse are the same if their code, term and universities are the same
1
University
- name : String
- country : String
+ University(name : String, country : String)
+ getName() : String
+ getCountry() : String
+ hashCode() : int
+ equals(obj : Object) : boolean
+ toString() : String
Two Universities are the same if they have the same name and country
Utility1
+ getCommonCourses(courseList1 : List<Course>, courseList2 : List<Course>) : List<String>
+ main(args : String[]) : void
Utility2
+ getListOfPartnerExchangeUniversities(courseList : List<Course>) : Map<String,List<University>>
+ main(args : String[]) : void
Utility3
+ getLOATerms(cList : List<Course>) : List<String>
+ main(args : String[]) : void
Utility4
+ getCoursesTakenInSameYearAndSemesterOfStudy(courseList1 : List<Course>, courseList2 : List<Course>) : List<String>
+ main(args : String[]) : void