Problem Description:
In this assignment, you are going to generate a transcript for students. Information about the students, the courses that they have taken, and the grades that they have received for different activities of a course are stored in a text file. Your program reads the text file and generates a transcript and write it to another text file.

First let us have a look at the format of the text file. The text file contains several rows, each dedicated to one course for one student. In each line you see:

See the next page

`Course code, course credit, student number, [P|E] a number that shows the weight of the assessment (a number that shows the grade received for this assessment), …., student name.`

In this format P stands for practical, which is associated to any types of assessment such as lab activity, exercises, homework, worksheets and so on. E stands for Exam.

For example, this line

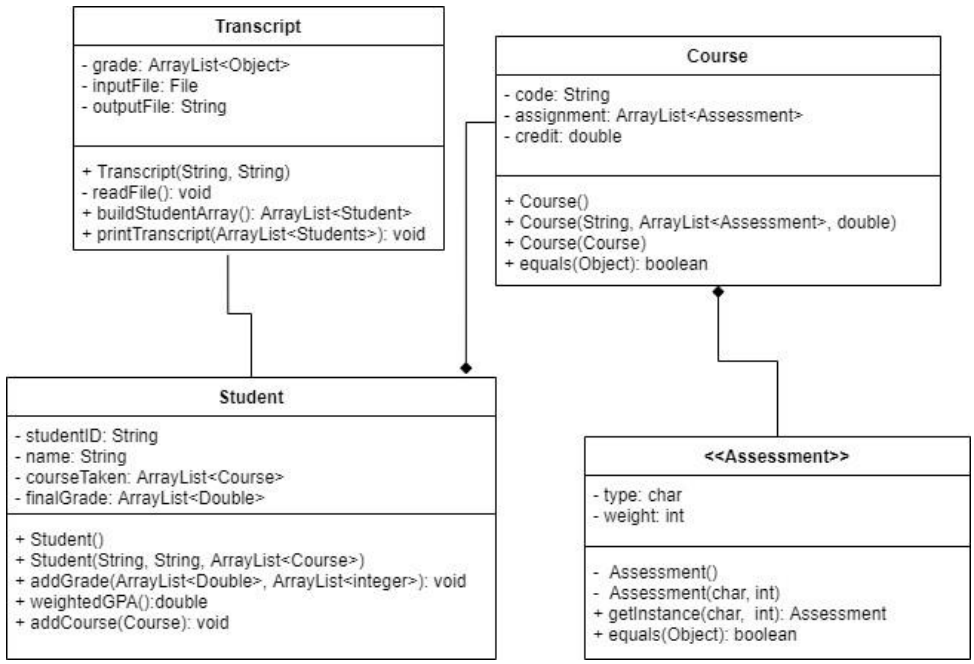
`EECS2030,3,1000,P10(90),P10(80),P30(60),E15(60),E15(44),E20(80),John`

explains that John, whose student number is 1000 has taken EECS2030, which is a three credits course. This course has three practical activities, which worth 10, 10 and 30 percent of the total mark, respectively. Also, this course has three exams, which worth 15, 15 and 20 percent of the final mark, respectively. Please note that the total of the weights must always be 100. The numbers that are in parenthesis shows the grade that John has received out of 100 for each assessment.

Please have a look at the text file (input.txt) that came with the starter code, to better understand the format of the file.

Task 1:

In this assignment, we are not going to tell you what to do step by step. Instead we ask you to implement our design that can be seen in the UML diagram below. In this diagram, you see all the classes, their attributes, methods, and relationship with each other, that you need to develop.



Please note that

- you can find the explanation about each class and its methods in the glossary at the final page of this document.
- not all the required methods can be seen in this diagram (and also in the glossary). Specifically speaking, I have removed all the accessors and mutators to let you decide, which class requires one.
- All the methods that are seen in the UML must be implemented.
- In case you needed to add helper methods to make your code more readable, you are welcome to do so. However, make sure that your helper methods are defined `private`.
- You should not add any attributes to the classes.

- You should not change any method signature.
- `readFile()` and the constructor of `Transcript` have already been implemented for you.
- Encapsulation should be applied to all the classes.
- Two students do not have the same student ID, and two different courses do not have the same code. This means you do not need to verify the correctness of the text file for these cases.
- `courseTaken` and `finalGrades` are parallel arrays. Two (or more) arrays are called parallel if A) they have the same number of elements, B) items in the same indexes are related. In our case, `finalGrade` at index *i*, shows the grade that the student received for the `courseTaken` at index *i*.
- `weightedGPA` and `addGrade` should round the grade to one decimal place.
- No test case is provided. It is your job to test each method and the whole program.
- You can import any java package that you require.
- No Documentation is provided. You are required to document the code completely. I have only provided some documentation for some of the methods, whose functionality were less obvious in the glossary.
- To compute the GPA, the following table should help you. `GPA = Average (grade point x course credit)`

Grade Point	Grade
9	90-100
8	80-89.99
7	75-79.99
6	70-74.99
5	65-69.99
4	60-64.99
3	55-59.99
2	50-54.99
1	47-49.99
0	Otherwise

Output:

The output should follow the format that is explained below.

```
John    1000
--------------------
EECS2030    66.6
EECS2012    64.5
--------------------
GPA: 4.4

Jane    2000
--------------------
EECS2030    76.4
EECS2031    82.1
EECS2012    71.3
EECS3033    63.0
--------------------
GPA: 6.1
```

As you see in the first line, first comes the name of the student followed by a tab (i.e. "\t") followed by the student number.

Next line contains 20 dashes.

Then in each line comes the course code followed by a tab and the grade that the student achieved for it.

Next line contains 20 dashes.
Next line the word GPA: is followed by the GPA that is rounded by one decimal place. There is a space between "GPA:" and the value of the GPA.
This pattern is continued for the rest of the students in the file.

Task 2: Exception Handling

It is time to handle the exceptions. It is possible that the total weight of the assessment in a row of the file does not add up to 100. Also, it is possible that the total grade of a student is more than 100. These two cases are wrong and therefore, an exception should be thrown. You should write your own exception that is called `InvalidTotalException`, that can be used for both the flawed situations.

This exception should be handled in the method called `addGrade`.

See the next page

Glossary:

The list of all classes, methods and instance variables are written here for two purposes. First, to make sure you do not lose any mark because of a typo. Second, the purpose of each component is described here. This list is sorted alphabetically.

- `addCourse`: get a course object as an input and add it to `courseTaken`.
- `addGrade`: This method gets an array list of the grades and their weights, computes the true value of the grade based on its weight and add it to finalGrade attribute. In case the sum of the weight was not 100, or the sum of grade was greater 100, it throws `InvalidTotalException`, which is an exception that should be defined by you.
- `assignment`: It is an attribute for the `Course` class.
- `Assessment`: it is a class name.
- `buildStudentArray`: This method creates and returns an *ArrayList*, whose element is an object of class `Student`. The object at each element is created by aggregating ALL the information, that is found for one student in the `grade` Arraylist of class `Transcript`. (i.e. if the text file contains information about 9 students, then the array list will have 9 elements.
- `code`: It is an attribute for the `Course` class.
- `Course`: This is the class that defines a course.
- `courseTaken`: It is an attribute for the `Student` class.
- `credit`: It is an attribute for the `Course` class.
- `equals`: it is an overridden method for Object's equals() method that returns true, if all the instance variables of two objects have the same value.
- `finalGrade`: It is an attribute for the `Student` class.
- `getInstance`: This is a static factory method for class `Assessment`.
- `grade`: It is an attribute for the `Transcript` class.
- `inputFile`: It is an attribute for the `Transcript` class.
- `InvalidTotalException`: This is the exception that is thrown if the total weight of the assessments does not add up to 100, or of the total grade of a student is more than 100.
- `Name`: It is an attribute for the `Student` class.
- `outputFile`: It is an attribute for the `Transcript` class.
- printTranscript: This is the method that prints the transcript to the given file (i.e. `outputFile` attribute)
- `readFile`: It is a method that reads the input file and stores each line of the file in the `grade` attribute of class `Transcript`.
- `Student`: It is a class that defines a student.
- `studentID`: It is an attribute for the `Student` class.
- `Transcript`: It is a class that reads a file containing the students' information, and generates a transcript for all the students.
- `type`: It is an attribute for the `Assessment` class.
- `weight`: It is an attribute for the `Assessment` class.
- `weightedGPA`: It is the method that computes the GPA.