Learning Outcomes

By completing this assignment, you will gain skills in:
- Using strings and reading text files in Java,
- Writing some Java classes,
- Using Java arrays, and
- Testing your code.

In this assignment you will create a complete Java program that uses classes to store and manipulate student and course data. You are required to implement 3 classes: Student, Course, and GradeReport, stored in files Student.java, Course.java, and GradeReport.java, respectively.

The Student Class

The Student class encapsulates a student using the private variables:
- studentName: the name of the student (of type String),
- studentEmail: the email of the student (of type String), and
- studentId: the id of the student (of type String).

and the methods:
- A constructor method, Student(), with 3 parameters: the student name, student email, and student id. This method creates an instance (object) of the class Student.
- Getter methods: getName(), getEmail(), and getId() that return the student name, student email, and student id, respectively.
- Setter methods: setName(String), setEmail(String), and setId(String) that saves the student name, student email, and student id, respectively.
- Method toString(): returns a string containing the values of the object’s private variables. This string should be formatted nicely. See the section called Formatting near the bottom of this document.

When a parameter and a variable have the same name, for example within Student.java’s constructor both the private variable private String studentName and the parameter String studentName are visible, then this.studentName refers to the private variable and studentName refers to the parameter.

The data for this class is stored in the file student.txt.
The Course Class

The Course class encapsulates a student taking a course using the private variables:

- `courseName`: the name of the course (of type `String`),
- `courseId`: the id of the course (of type `String`),
- `studentId`: the id of the student (of type `String`), and
- `studentGrade`: the grade of the student (of type `int`).

and the methods:

- A constructor method, `Course()`, with 4 parameters: the course name, course id, student id, and student grade. This method creates an instance of the class Course.
- Getter methods: `getCourseName()`, `getCourseId()`, `getStudentId()`, and `getStudentGrade()` that return the course name, course id, student id, and student grade, respectively.
- Setter methods: `setCourseName()`, `setCourseId()`, `setStudentId()`, and `setStudentGrade()` that saves the course name, course id, student id, and student grade, respectively.
- Method `toString()`: returns a string containing the values of the object’s private variables. This string should be formatted nicely. See the section called Formatting near the bottom of this document.

The data for this class is stored in the file `course.txt`.

The GradeReport Class

The GradeReport class holds information about the students and the courses they are in, using the private variables:

- `studentId`: the id of the student (of type `String`),
- `courseId`: the id of the course (of type `String`),
- `studentCt`: the number of students (of type `int`),
- `courseCt`: the number of courses (of type `int`),
- `studentArray`: an array containing a list of student objects (of type `Student`), and
- `courseArray`: an array containing a list of course objects (of type `Course`).

This class has the following methods:

- A constructor method, `GradeReport()`, with 4 parameters: the array of student objects, array of course objects, number of student objects, and number of course objects. This method creates an instance of the class GradeReport. **Hint:** you can copy an array from a parameter to a private variable using the `clone()` method. For example, `this.studentArray = studentArray.clone();`.
- Method `getGradesOfCourse()`: returns a string containing a list of all students and their corresponding grades for a given course id. This string should be formatted nicely and include the average grade of the course to 2 decimal places. See the section called Formatting near the bottom of this document. **Hint:** you can compare course id’s using the `equals()` method. For example, to compare two strings `A` and `B` you can use the boolean expression `A.equals(B);`.
- Method `getGradesOfStudent()`: returns a string containing a list of all courses and the corresponding grades for a given student id. This string should be formatted nicely and include the average grade of the course to 2 decimal places. See the section called Formatting near the bottom of this document.
The Main Class

The Main class has a static method called main(), which runs all of the code. The files student.txt and course.txt are read and the information is used to create Student and Course objects. These objects are inserted one at a time into the studentArray and courseArray arrays, which in turn are used to create an object of type GradeReport. Methods in this object are then checked using 10 tests. Main.java is provided for you.

The InStringFile Class

The InStringFile class contains code to read text from a file. This process involves reading the file line-by-line, and for each line, reading the text word-by-word. Main.java uses this class when parsing the two input files. InStringFile.java is provided for you.

Testing

Testing is done by running the supplied Main.java. There are 10 tests.

Formatting

The output from getGradesOfStudent(), getGradesOfCourse(), and the toString() methods should be formatted nicely. You can use the new line character (\n), the tab character (\t), string concatenation (+), the format() method, or any other method that you might find. For example:

```java
String s = "Hello\tWorld!\n";
s = s + String.format("The character string %-6s\n", "62") +
    String.format("has integer value %5d\n", 62) +
    String.format("and double value %9.2f\n", 62.0);
System.out.println(s);
```

prints:

Hello   World!
The character string 62
has integer value 62
and double value 62.00

You can google method format() for a full description of how it works.

The output from Student’s toString() should include the student name, email, and student id such as:
Student Name: Andrew
Student Email: ablochha@uwo.ca
Student Id: 1

The output from Course’s toString() should include the course name, course id, student id, and grade such as:

Course Name: JavaFundamentals1
Course Number: 1026
Student Id: 1
Student Grade: 100

Each line of output from GradeReport’s toString() should include the student id, grade, and course id such as:

Student Id: 1, Grade: 100, Course: 1026
Student Id: 1, Grade: 100, Course: 1017

Each line of output from getGradesOfCourse() should include the student name and grade, such as:

Student Name: Andrew the Aligator, Grade: 100
Student Name: Fred, Grade: 73
The Average of the Course: 86.50

Each line of the output of getGradesOfStudent() should include the student name, grade, and course id, such as:

Student Name: Andrew the Aligator, Course: 1026, Grade: 100
Student Name: Andrew the Aligator, Course: 1027, Grade: 100
The Average of the Student: 100.00

Hint: You will need to use a nested loop to find studentName corresponding to studentId for each course.

You do not need to follow exactly the format shown above; as long as you include in your output the correct fields in a readable way and calculate the average grade to 2 decimal places you will get full marks for formatting.

Submission

Submit all your .java files to OWL. Do not put the code inline in the textbox. Do not submit your .class files. If you do this and do not submit your .java files your program cannot be marked. Do not submit a compressed file with your Java classes (.zip, .rar, .gzip, ...). Do not put a "package" command at the top of your Java classes.
Non-Functional Specifications:

1. **Assignments are to be done individually and must be your own work.** Software will be used to detect cheating.

2. Your program does not have to perform exception handing.

3. You can assume that the data in the input files are correct. So no error checking of any type is required.

4. Add comments at the beginning of your classes indicating who the author of the code is and giving a brief description of the class. Add comments to the methods and instance variables.

5. Include comments in each Java file describing key parts of your program. Comments should be grammatically correct, concise, and easy to understand.

6. Use Java coding conventions and good programming techniques. For example:
   - Use Java conventions for naming variables and constants.
   - Write readable code: good indentation, appropriate white spaces, etc., are required.

7. Make sure your code runs using Eclipse’s Java, even if you do not use Eclipse to write your code.

Grading Criteria

- Total Marks: [20]
- Functional Specifications:
  - [1] Student.java
  - [1] Course.java
  - [4] GradeReport.java
  - [10] 10 Tests
- Non-Functional Specifications:
  - [1.5] Meaningful variable names, private instance variables
  - [0.5] Code readability and indentation
  - [1] Code comments
  - [1] Output formatting

Assignment files (Student.java, Course.java, and GradeReport.java) are to be submitted to OWL by 11:55pm on September 27th (a Friday).