420-201-VA | Assignment 1-B | Due by 15 February 2020

Modeling a Course Grade Book

1 Purpose

The purpose of this assignment is to provide practice with one-dimensional arrays of object reference variables as well as arrays of variables of a primitive data type. The context in which you will practice arrays involves writing four classes using OOP principles, focusing on composition and aggregation.

The code for most of the common array operations can be found in the course textbook:

- Printing All Elements of an Array (page 434)
- Summing the Elements of an Array (page 436)
- Finding a Maximum Value in an Array (page 438)
- Copying Array Elements into Another Array (page 441)
- Comparing Arrays of Primitive Data Types (page 445)
- The Sorter Class (page 472)
- Using Arrays as Counters (page 496)
- Exercises 63 (page 515) and 68 (page 518), at the end of chapter 8

2 Your Assignment

You have just started work for the Vanier College Teachers Association (VTCA) and have been asked to write a program to assist teachers with

(1) maintaining a list of student grade records for a course,
(2) basic analysis of the grades, and
(3) computing a student's final numeric grade and final letter grade.

You are told that a student's grade record includes a student's name and number, together with a list of assessment grades for that student.

Your program is to display the results in a table form, showing the names, weights, and grades of all the assessments[1] included in a grade record, together with basic statistics[2] on each assessment.

Having a general understating of the task at hand, you ask the VTCA for a sketch or layout of how the table format should look for a typical list of student grade records. They offer the following sample spreadsheet printout, pointing out that although all numeric grades are printed out with zero decimal places, they are actually floating-point numbers in the range [0, 100].

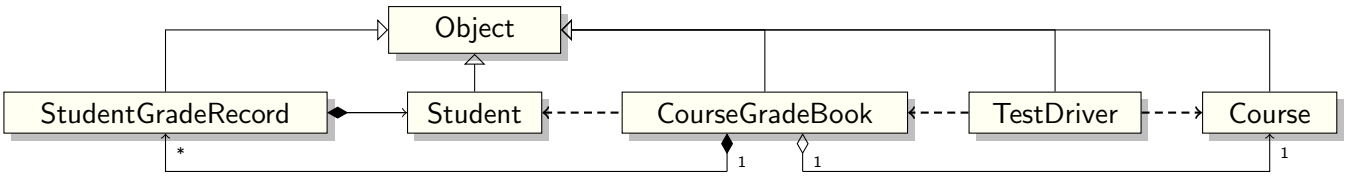
[1] such as homework, assignment, quiz, midterm, final exam, etc.
[2] such as maximum, minimum, average, standard deviation, etc.

```
Course Information
------------------
title       : Programming 2
number      : 420-201-VA
section     : 0001 & 0002
semester    : Winter
year        : 2020
teacher     : Sadegh Ghaderpanah
room        : B-502

                                Student Grade Table
-------------------------------------------------------------------------------------
ID Number  Student Name      A1    A2    A3    A4    A5    A6    A7    A8    fin  grd
-------------------------------------------------------------------------------------
63989996  Caden Olivia       60    66    37    85    87    85    73    79     78   C
75589670  Ezra Colton        93    86    93    88    93    58    35    69     64   D
38977699  Cameron Ryan       59    87    86    99    66    97    78    55     73   C
88936308  Charlotte Aria     80    67    77    63    97    77    89    95     88   B
80970578  Caleb Owen         99    93    90    67    53    79    79    89     82   B
86795367  Lucas Jack         88    86    76    90    95    89    79    77     82   B
89099578  Caleb Charlie      75    87    86    93    78    79    93    55     73   C
75950969  William Wyatt      93    69    93    79    87    77    67    78     77   C
78989969  Mila Charlotte     80    98    99    56    95    89    90    78     84   B
78798089  Elena Aiden        77    87    79   100    88    73    73    88     82   B
-------------------------------------------------------------------------------------
                     Max:    99    98    99   100    97    97    93    95     88   B
                     Min:    59    66    37    56    53    58    35    55     64   D
                 Average:    80    83    82    82    84    80    76    76     78   C
      Standard Deviation:    13    11    16    15    14    10    16    13      6

Legend
-----------------------------------------------
Assessment        Name    Weight/50.0   Weight%
-----------------------------------------------
    A1             hw1           1       2.0%
    A2             hw2           1       2.0%
    A3             hw3           2       4.0%
    A4             hw4           2       4.0%
    A5             hw5           4       8.0%
    A6       midterm-1          10      20.0%
    A7       midterm-2          10      20.0%
    A8      final exam          20      40.0%
-----------------------------------------------
   fin     final grade
   grd    letter grade
```

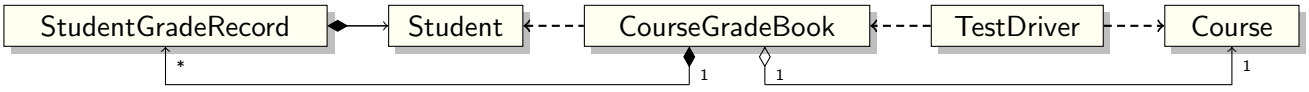
The VTCA feels that hiding the decimal points and the decimal places to their right increases readability. Again, the numeric grades are all floating-point numbers in the range [0, 100].

3 Requirements

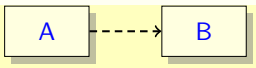
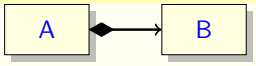
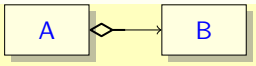
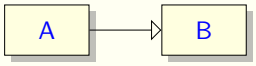
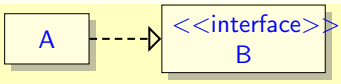
Your program must implement the following class diagram in Java:



Since the `Object` class is the superclass to all other classes in Java, your program will effectively implement the following UML diagram of Java classes:



To interpret the class diagram above, let's first expand the UML notations we listed in A1-A:

Relationship	UML notation	Meaning
Dependency		An `A` uses (depends on) a `B`. Basically, an `A`'s method has a `B` reference, as a local variable, a parameter, or both.
Composition		An `A` is made up of `B`s with lifetime dependency. The `B`s belong to `A`; if `A` is destroyed, its `B`s are destroyed as well.
Aggregation		An `A` is made up of `B`s. The `B`s are shared with `A`.
Inheritance		An `A` *is a* special kind of `B`. `A` specializes `B`. `B` generalizes `A`. `A` has access to all non-private members of `B`.
Interface		An `A` uses (requires) the interface `B`. The `A` class must implement the operations specified in the B interface.

Let us next recall that basically a class `A` can associate with a class `B` if class `A` uses an object reference of type `B`

(i) outside all of `A`'s methods, such as the instance variable `ivar` on line 4,

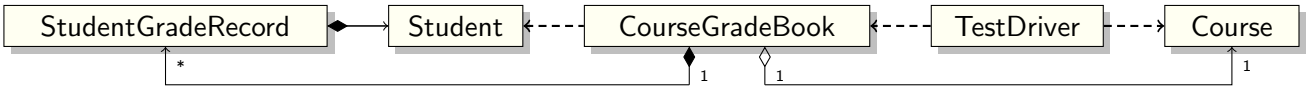
(ii) inside one or more methods of `A`, such as the local variable `local` on line 6 in method `f`, or

(iii) inside the parameter list of one or more methods of `A`, such as the parameter `param` on line 8 in method `g`.

```
1  class B{}
2  class A
3  {
4      private B ivar ;
5      public void f() {
6          B local ;
7      }
8      public int g( B param )
9      {return 1;}
10 }
```

Note that the three association cases listed above implicitly cover cases where a method of `A` returns a reference of type `B`, and where class `A` inherits another class that associates with `B`.

Using the UML notation and observation above, we can describe the association between these classes as follows:



(1) `TestDriver` depends on both `CourseGradeBook` and `Course` for its operations, meaning that one or more of its methods must be using references to `CourseGradeBook` or `Course` as local variables or parameters. Neither `Course` nor `CourseGradeBook` knows about `TestDrive`

(2) `CourseGradeBook` does have an instance variable referencing a `Course` object, but `CourseGradeBook` does not own (create) the `Course` object itself. Rather, a `Course` object is being shared with `CourseGradeBook`. `Course` does not know about `CourseGradeBook`.

(3) `CourseGradeBook` uses `Student`, and it owns zero or more `StudentGradeRecords`. Each `StudentGradeRecord` is owned by exactly one `CourseGradeBook`. `Student` does not know about either of `CourseGradeBook` or `StudentGradeRecord`.

(4) A `StudentGradeRecord` object owns a `Student` object. A `Student` object belongs to (is owned by) exactly one `StudentGradeRecord` object. `StudentGradeRecord` does not know about `CourseGradeBook`.

A description of the classes that you need to implement in this assignment is given below.

For this assignment only, you cannot use Java's `ArrayList` class or any of the other Java classes from the Java library that implements `Collection`.

For this assignment only, you cannot use any method (such as searching, sorting, and copying, etc.) of Java's `Arrays` class.

For future assignments only, always prefer using Java classes from the Java library; for example, Java's `Arrays`, `ArrayList`, and `Collections` classes, as well as Java classes that implement `Collection`.

3.1 Class Course

This class encapsulates a course taken by a student.

Course	The name of this Java class
− name : String	stores the name of *this* course
− number : String	stores the id number of *this* course
− section : String	stores the section of *this* course
− semester : String	stores the semester of *this* course
− year : int	stores the year of *this* course
− teacher : String	stores the teacher of *this* course
− room : String	stores the classroom of *this* course
+ Course(name : String, number : String, section : String, semester : String, year : int, teacher : String, room : String):	Constructs a new course object, initializing it to the specified name, id number, section, semester, year, teacher, and room.
+ Course(course : Course)	copy constructor
+ setName(name : String) : void	sets the name of *this* course
+ setNumber(number : String) : void	sets the id number of *this* course
+ setSection(section : String) : void	sets the section of *this* course
+ setSemester(semester : String): void	sets the semester of *this* course
+ setYear(year : int) : void	sets the year of *this* course
+ setTeacher(teacher : String) : void	sets the teacher of *this* course
+ setRoom(room : String) : void	sets the room of *this* course
+ getName() : String	returns the name of *this* course
+ getNumber() : String	returns the id number of *this* course
+ getSection() : String	returns the section of *this* course
+ getSemester() : String	returns the semester of *this* course
+ getYear() : int	returns the year of *this* course
+ getTeacher() : String	returns the teacher of *this* course
+ getRoom() : String	returns the room of *this* course
+ equals(obj : Object) : boolean	compares *this* and obj for equality
+ toString() : String	returns a string similar to lines 1-9, page 2

3.2 Class Student

Student	The name of this Java class
− name : String	stores the name of *this* Student
− id : int	stores the id number of *this* Student
+ Student(name : String , id : int) :	constructs a new Student object, initializing it to the specified name, id number
+ Student(Student other) :	copy constructor
+ getName() : String	returns the name of *this* student
+ getId() : int	returns the id number of *this* student
+ setName(String name) : void	sets the name of *this* student
+ setId(int id) : void	sets the id number of *this* student
+ equals(obj : Object) : boolean	compares *this* and obj for equality
+ toString() : String	returns a string containing the values of *this* object's private instance variables

3.3 Class StudentGradeRecord

StudentGradeRecord	The name of this Java class
− grades : double[]	stores the grades in *this* StudentGradeRecord
− student : Student	stores the student of *this* StudentGradeRecord
+ StudentGradeRecord(student : Student , grades : double[]) :	constructs a new StudentGradeRecord object, initializing it to the specified student and grades
+ StudentGradeRecord(grdRecord : StudentGradeRecord) :	copy constructor. deep copies from grdRecord
+ computeFinalGrade(assessmentWeights : double[]) : double	computes and returns the final grade of the StudentGradeRecord using the assessment weights
+ getgrade(k : int) : double	Return the k'th assessment grade. Throws an IllegalArgumentException if k is out of bounds
+ setGrade(k : int , grd : double) : void	sets the value of the k'th assessment grade to grd. Throws an IllegalArgumentException if k is out of bounds
+ getAllGrades() : double []	returns a deep copy of the grade list of *this* StudentGradeRecord
+ setAllGrades(allGrades : double []) : void	sets the grade list of *this* StudentGradeRecord to deep copy of allGrades. Throws an IllegalArgumentException if the length of allGrades is different from the number of assessments
+ getStudent() : Student	returns the student of this StudentGradeRecord
+ setStudent(student : Student) : void	sets the student of *this* StudentGradeRecord
+ getNumberOfAssessments() : int	returns the number of assessments
+ computeLetterGrades(fin : double) : static char	converts grade g to a letter grade, which it returns; specifically, returns 'A', 'B', 'C', or 'D' if g is ≥ 90, g is ≥ 80, g is ≥ 70, or g is ≥ 60, respectively; otherwise, returns 'F'
+ equals(obj : Object) : boolean	compares *this* and obj for equality
+ toString() : String	returns a string containing the values of *this* object's private instance variables

Note: In UML, static members are underlined.

3.4 Class CourseGradeBook

CourseGradeBook	name of this Java class; referred below as CGB
– caNames : String[]	stores the course assessment names of *this* CGB
– caWeights : double[]	stores the course assessment weights of *this* CGB
– course : Course	stores the course of *this* CGB
– gradeList : StudentGradesRecord[]	stores a reference to the list of student grade records of *this* CGB. The length of this list must be initialized to 1 for newly created CGB objects.
– gradesRecordCount : int	stores the number of students records added to the gradeList of *this* CGB
+ CourseGradeBook(course : Course, assNames : String[], assWeights : double[]) :	Constructs a new course object, initializing it to the specified course, assessment names and weights. Must throw IllegalArgumentException if assName and assWeights are unequal in length
+ isFull() : boolean	returns gradesRecordCount == gradeList.length
+ addGradeRecord(student : Student, grades : double...) : void	Adds a student record using the specified name and grades, or student and grades. Must double the length of gradeList if it is full. Must throw IllegalArgumentException if grades and caName are of different lengths
– doubleGradeListCapacity() : void	doubles the length of gradeList.
+ findArraySum(numbers : double[]) : double + findArrayMaximum(numbers : double[]) : double + findArrayMinimum(numbers : double[]) : double + findArrayAverage(numbers : double[]) : double + findArrayStandardDev(numbers : double[]) : double	returns the sum, highest, lowest, average of, and standard deviation of all elements of the numbers array, respectively. See section 8.3 of the course textbook for such common array operations and more
+ computeTotalWeight() : double	returns the sum of all course assessment weights
+ getFinalsArray() : double[]	returns an array of the final grades of all student records added to gradeList
+ getAssessmentArray(k : int) : double[]	returns an array of the k'th assessment grades of all student records added to gradeList. Must throw IllegalArgumentException of k is out of bounds
+ findMaxAssessment(k : int) : double + findMinAssessment(k : int) : double + findAvgAssessment(k : int) : double + findStdevAssessment(k : int) : double	return the highest, lowest, average of, and standard deviation of the k'th assessment grade of all student records added to gradeList, respectively. Must throw IllegalArgumentException of k is out of bounds
+ toStringMaxAssessments() : String	return a string similar to line 26, page 2
+ toStringMinAssessments() : String	return a string similar to line 27, page 2
+ toStringAvgAssessments() : String	return a string similar to line 28, page 2
+ toStringStdevAssessments() : String	return a string similar to line 29, page 2
+ toStringAssessmentLegend() : String	return a string similar to lines 31-45, page 2
+ equals(obj : Object) : boolean	compares *this* and obj for equality
+ toString() : String	returns a string similar to lines 11-25
+ CourseGradeBook(cgb : CourseGradeBook)	copy constructor, deep copies from cgb's caNames, caWeights, course, and gradeList.

3.5 Class TestDriver

```
public class GradeBookTestDriver
{
   public static void main(String[] args)
   {

      String title         = "Programming 2";
      String courseNumber  = "420-201-VA";
      String section       = "0001 & 0002";
      String semester      = "Winter";
      int year             = 2020;
      String teacher       = "Sadegh Ghaderpanah";
      String room          = "B-502";

      Course course = new Course(title, courseNumber, section, semester, year, teacher, room);

      String[] assessmentNames =
      {
         "hw1", "hw2", "hw3", "hw4", "hw5", "midterm-1", "midterm-2", "final exam"
      };
      double[] assessmentPoints =
      {
         1,1,2,2,4,10,10,20
      };

      CourseGradeBook cgb = new CourseGradeBook(course, assessmentNames, assessmentPoints);

      cgb.addGradeRecord(new Student("Caden Olivia"  , 63989996), 60, 66, 37, 85, 87, 85, 73, 79);
      cgb.addGradeRecord(new Student("Ezra Colton"   , 75589670), 93, 86, 93, 88, 93, 58, 35, 69);
      cgb.addGradeRecord(new Student("Cameron Ryan"  , 38977699), 59, 87, 86, 99, 66, 97, 78, 55);
      cgb.addGradeRecord(new Student("Charlotte Aria", 88936308), 80, 67, 77, 63, 97, 77, 89, 95);
      cgb.addGradeRecord(new Student("Caleb Owen"    , 80970578), 99, 93, 90, 67, 53, 79, 79, 89);
      cgb.addGradeRecord(new Student("Lucas Jack"    , 86795367), 88, 86, 76, 90, 95, 89, 79, 77);
      cgb.addGradeRecord(new Student("Caleb Charlie" , 89099578), 75, 87, 86, 93, 78, 79, 93, 55);
      cgb.addGradeRecord(new Student("William Wyatt" , 75950969), 93, 69, 93, 79, 87, 77, 67, 78);
      cgb.addGradeRecord(new Student("Mila Charlotte", 78989969), 80, 98, 99, 56, 95, 89, 90, 78);
      cgb.addGradeRecord(new Student("Elena Aiden"   , 78798089), 77, 87, 79, 100, 88, 73, 73, 88);

      System.out.println(course);
      System.out.println(cgb);
      System.out.println(cgb.toStringMaxAssessments());
      System.out.println(cgb.toStringMinAssessments());
      System.out.println(cgb.toStringAvgAssessments());
      System.out.println(cgb.toStringStdevAssessments());
      System.out.println(cgb.toStringAssessmentLegend());
   }
}
```

4 Evaluation Criteria

Evaluation Criteria

Functionality	Ability to perform as required, producing correct output for any set of input data, Proper implementation of all specified requirements, Efficiency	60%
Robustness	Ability to handle input data of wrong type or invalid value	10%
OOP style	Encapsulating only the necessary data inside objects, Information hiding, Proper use of Java constructs and facilities.	10%
Documentation	Description of purpose of program, Javadoc comment style for all methods and fields, comments on non-trivial steps in all methods	10%
Presentation	Format, clarity, completeness of output, user friendly interface	5%
Code readability	Meaningful identifiers, indentation, spacing, localizing variables	5%