COMPSCI 101 – Semester Two, 2013

Assignment Two — Dunrobin Cycling Competitions

Due: 4:00pm, Thursday 3rd October

Worth: 5%

Introduction

In this assignment, you will develop a program which manages cycling competitions. Each competition consists of multiple races with multiple cyclists in each race. The winners and runners-up all receive points, and the cyclist with the most points wins the competition.

Learning goals

This assignment is designed to help you learn a number of concepts and practice a number of skills. In particular:

- defining classes
- managing arrays of objects

This program manages a database of races and competitors. The first three stages are the development of the `MyDate`, `Competitor` and `Race` classes. In the last stages you need to complete the program which manages the competition database.

Stage 1 – Define the `MyDate` class

Initially, to develop the three classes, you will be working in the `Stages1To3` folder. The files you will find in this folder are:

`MyDate.java`

`Competitor.java`

`Race.java`

`TestA2.java`

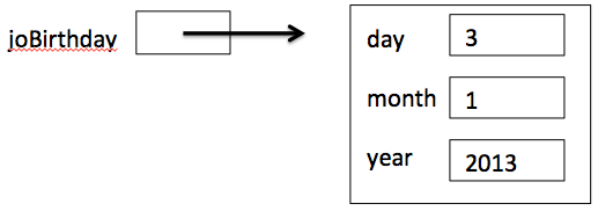
`TestA2Classes.java`

Once you have defined the `MyDate`, `Competitor` and `Race` classes you can do a preliminary check of these classes by running the `TestA2` application.

A `MyDate` instance is used in the Assignment 2 program to store a calendar date. A `MyDate` object can be declared and created using code similar to the following:

```
MyDate joBirthday = new MyDate(3, 1, 2013);
```

and the above MyDate instance can be visualised in the following way:



The `toString()` method of the `MyDate` class returns a String made up of the day number, month number and year number separated by the "/" character. For example, the following code:

```
System.out.println( joBirthday.toString() );
```

prints:

```
3/1/2013
```

Below is the skeleton for the `MyDate` class.

```
public class MyDate {
   private int day;
   private int month;
   private int year;

   public MyDate(int day, int month, int year) { … }
   public String toString() { … }
}
```

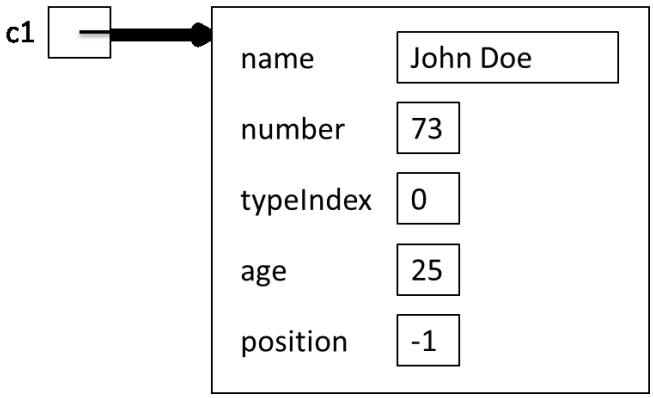
You need to complete the constructor and `toString()` methods for this class.

Stage 2 – Define the `Competitor` class

A `Competitor` instance is used in the Assignment 2 program to store a single competitor in a race. A `Competitor` object can be declared and created using code similar to the following:

```
String name = "John Doe";
int number = 73;
int competitorType = Competitor.A_GRADE;
int age = 25;
Competitor c1 = new Competitor(name, number, competitorType, age);
```

and the above `Competitor` instance can be visualised in the following way:



Below is the skeleton for the `Competitor` class.

```
public class Competitor {
   public static final String[] COMPETITOR_DESCRIPTIONS = {
                                        "A Grade", "B Grade", "Senior"};
   public static final int A_GRADE = 0;
   public static final int B_GRADE = 1;
   public static final int SENIOR = 2;

   private String name;
   private int number;
   private int typeIndex;
   private int age;
   private int position;

   public Competitor(String name, int number, int typeIndex,
                                                        int age) { … }

   public String getName() { … }
   public int getNumber() { … }
   public int getTypeIndex() { … }
   public int getAge() { … }
   public int getPosition() { … }
   public void setPosition(int newPosition) { … }
   public String toString() { … }
}
```

Notes on the `Competitor` class

There are three kinds of competitors and the `typeIndex` instance variable stores an `int` indicating which type of competitor it is. The three types of competitors are:

A Grade – `typeIndex` is 0 (given by the constant, `A_GRADE`).

B Grade – `typeIndex` is 1 (given by the constant, `B_GRADE`).

and

Senior – `typeIndex` is 2 (given by the constant, `SENIOR`).

You will need to complete the constructor method, the five accessor methods (`getName()`, `getNumber()`, `getTypeIndex()`, `getAge()` & `getPosition()`), the mutator method (`setPosition()`) and the `toString()` method.

The constructor of the `Competition` class initialises the five instance variables. The parameters `name`, `number`, `typeIndex` and `age` are all passed to the constructor. The `position` instance variable is initially set to -1 in the constructor and later updated by the setter method (`setPosition`). A `position` of -1 indicates the competitor has not finished the race. Other allowed values are 1, 2, 3 (for first, second or third) or zero (finished outside first three).

The `toString()` method of the `Competitor` class returns a `String` describing the competitor. This starts with the number of the competitor, a comma and the name of the competitor, followed by square brackets "[ ]" containing the grade and age separated by a "-". If the position is not -1 it is then followed by the literal ": " (colon space) then a `String` with the position. For 1 the position will be the literal "First", for 2 the position will be the literal "Second", for 3 the position will be the literal "Three", otherwise it will be the literal "Finished". Five example `String`s returned by the `toString()` method of the `Competitor` class are shown below:

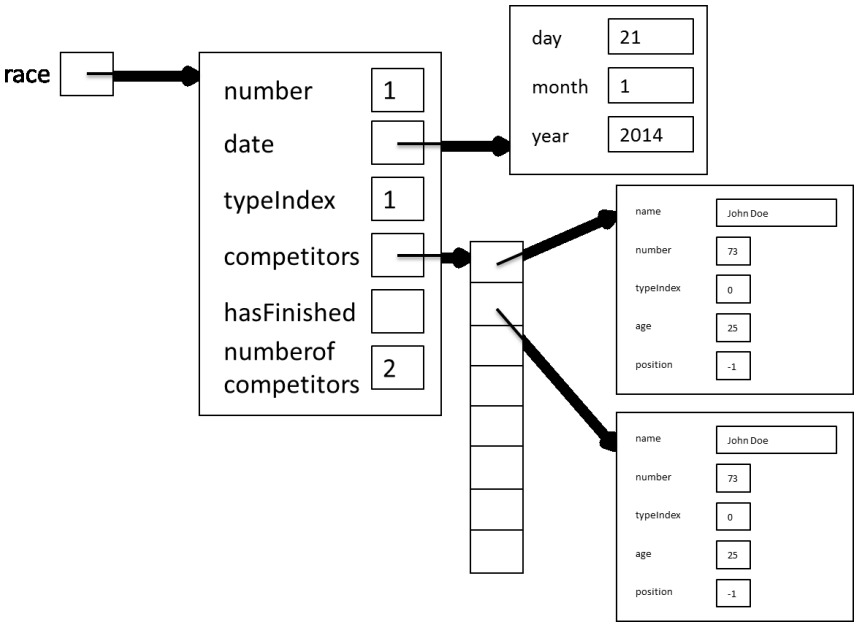
```
73, John Doe [A Grade-25]: First
32, Joe Ra [A Grade-22]: Second
105, Jane Mee [Senior-45]: Third
42, Jake Farr [B Grade-29]: Finished
86, James Seo [Senior-51]
```

Stage 3 – Define the `Race` class

A `Race` instance is used in the Assignment 2 program to store the details about a race. A `Race` instance can be declared and created using code similar to the following:

```
int number = 1;
int raceType = Race.SPRINT;
int date = new MyDate(21, 1, 2014);
Race race = new Race(number, raceType, date);
```

and the above `Race` instance (after two `Competitors` have been added) can be visualised in the following way:



Below is the skeleton for the `Race` class.

```
public class Race {
   public static final String[] RACE_DESCRIPTIONS = {"Sprint",
                                "Distance", "Eliminator", "Keirin"};
   public static final int SPRINT = 0;
   public static final int DISTANCE = 1;
   public static final int ELIMINATOR = 2;
   public static final int KEIRIN = 3;
   public static final int MAX_COMPETITORS = 8;

   private int number;
   private int typeIndex;
   private MyDate date;
   private boolean hasFinished;
   private Competitor[] competitors;
   private int numberOfCompetitors;

   public Race(int number, int typeIndex, MyDate date) { … }

   public int getNumber() { … }
   public boolean getHasFinished() { … }
   public int getTypeIndex() { … }
   public MyDate getDate() { … }
   public Competitor getCompetitor(int number) { … }
   public void finishRace(int first, int second, int third) { … }
   public boolean addCompetitor(Competitor competitor) { … }
   public String toString() { … }
}
```

Notes on the `Race` class

The constructor method for the `Race` class initialises the six instance variables: `number`, `typeIndex` and `date` are passed in as parameters, `hasFinished` is set to `false`, `numberOfCompetitors` is set to 0 and the `competitors` array should be big enough to hold eight competitors (use the `MAX_COMPETITORS` constant provided in the `Race` class).

The `getHasFinished()`, `getTypeIndex()` and `getDate()` accessor methods return the values in the underlying instance variables (`hasFinished`, `date` and `typeIndex`).

The `getCompetitor()` accessor method returns the competitor whose number matches the number passed in as a parameter. If there is no competitor with a matching number this method returns `null`.

The `finishRace()` instance method marks the race as finished. This involves setting the `hasFinished` variable to `true`, then looping through all the competitors in the race and setting their position. If the competitor's number matches the number in the parameter `first` their position is set to 1, if their number matches the parameter `second` it is set to 2 and their number matches the parameter `third` it is set to 3. Otherwise their position is set to zero.

The `addCompetitor()` instance method will first check if there is space for another competitor (i.e. `numberOfCompetitors` is less than `MAX_COMPETITORS`). If there is space the competitor will be added in the next available slot and the `numberOfCompetitors` variable incremented by one. The method returns `true` if the competitor was added (i.e. there was space) otherwise it returns `false`.

The `toString()` instance method returns a `String` describing the race. The first part of the `String` has the race number, a comma and race type (given by the `RACE_DESCRIPTIONS String` constant) followed by the date inside square brackets. If the race has not finished the initial `String` is followed by the literal `": Race not finished"`. Otherwise it is followed by three lines with the names of the first, second and third place winners. Each line is prefixed by five spaces and the position (e.g. 1st, 2nd or 3rd). If the position has not been set then the string literal `"n/a"` should be returned instead of the person's name.

Three example `String`s returned by the `toString()` method of the `Race` class are shown below:

```
1, Sprint [21/1/2014]: Race not finished
```

```
2, Distance [20/1/2014]
 1st: John Doe
 2nd: Joe Ra
 3rd: Jane Mee
```

```
3, Keirin [22/2/2014]
 1st: Joe Ra
 2nd: n/a
 3rd: n/a
```

Now that you have defined the three classes, you can do a preliminary check of your classes by compiling and running the `TestA2` application. The output when you run the `TestA2` application should be:

```
Preliminary testing of the MyDate class
3/12/2013
23/4/2013

Preliminary testing of the Competitor class
73, John Doe [A Grade-25]: First
84, Joe Ra [A Grade-22]: Second
16, Jane Mee [Senior-45]: Third
108, Jake Farr [B Grade-29]: Finished
34, James Seo [Senior-51]

Preliminary testing of the Race class
1, Sprint [21/1/2014]: Race not finished
2, Distance [20/1/2014]
 1st: John Doe
 2nd: Joe Ra
 3rd: Jane Mee
3, Eliminator [20/1/2014]
 1st: James Seo
 2nd: n/a
 3rd: n/a
```

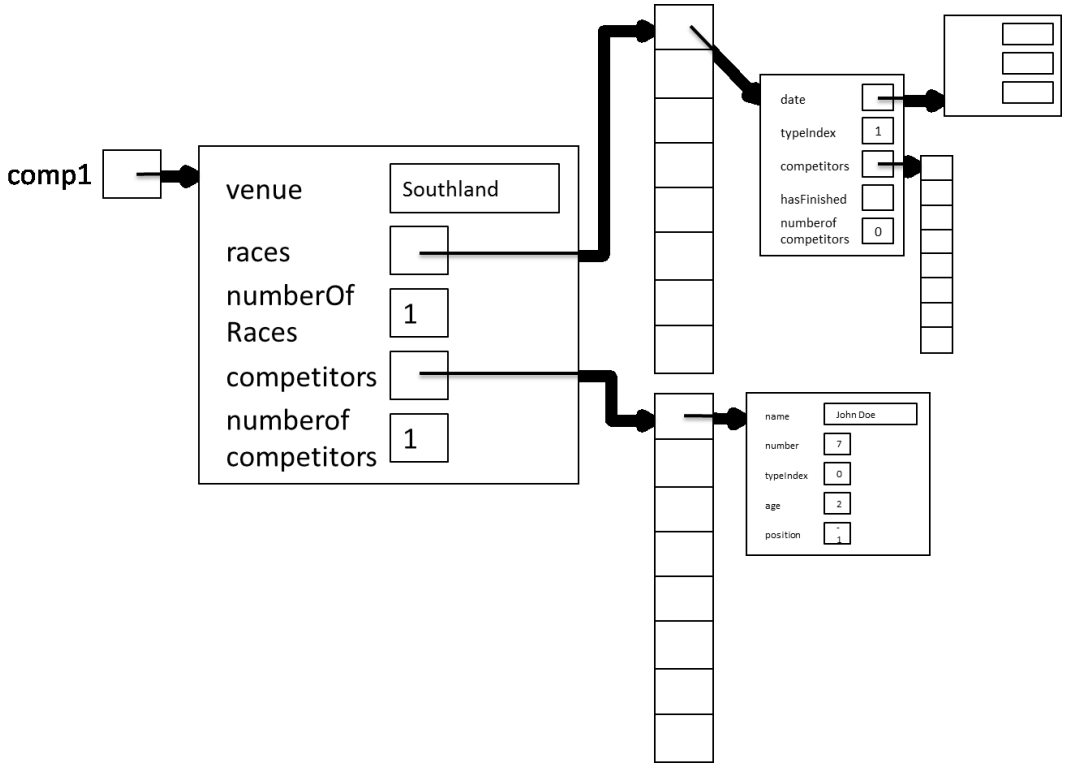
Now that you have tested your three classes, make a copy of your three classes and put a copy of the three classes in the `A2Stages4And5` folder. Now you are ready to complete the last stages of the assignment. This part of the assignment manages the competition database.

Stage 4: Complete the `Competition` class

This class contains all the race details for a single competition. A basic skeleton has been provided: you will need to finish the incomplete sections. A `Competition` instance can be declared and created using code similar to the following:

```
String venue = "Southland";
Competition comp1 = new Competition(venue);
MyDate date = new MyDate(1, 2, 2014);
comp1.addRace(Race.Sprint, date);
comp1.addCompetitor(32, "Joe Ra", 22 Competitor.A_GRADE);
```

and the above `Competition` instance can be visualised in the following way:



Below is the skeleton for the `Competition` class.

```
public class Competition {
   public static final int MAX_RACES = 50;
   public static final int MAX_COMPETITORS = 50;
   private String venue;
   private Race[] races;
   private int numberOfRaces;
   private Competitor[] competitors;
   private int numberOfCompetitors;

   public Competition(String venue) { … }

   public String getVenue() { … }
   public Competitor[] getCompetitors() { … }
   public Race[] getRaces() { … }
   public Race addRace(int typeIndex, MyDate date) { … }
   public Competitor addCompetitor(int number, String name, int age,
                                                  int typeIndex) { … }
   public Race getRace(int number) { … }
   public Competitor getCompetitor(int number) { … }
   public double calculateFinalScore(int number) { … }
   public boolean addCompetitorToRace(int competitor, int race)
                                                                { … }
   public String getTop10Competitors() { … }
   public String toString() { … }
}
```

Notes on the `Competition` class

The following methods have been implemented for you and you should not modify them:

- `calculateFinalScore()`
- `addCompetitorToRace()`
- `getTop10Competitors()`
- `toString()`

You will need to implement the constructor method. This sets the `venue` instance field to the received venue argument. `numberOfRaces` and `numberOfCompetitors` should both be set to zero. `races` and `competitors` should be set constructed as arrays of the relevant class (use the pre-defined constants `MAX_RACES` and `MAX_COMPETITORS`).

You will need to implement the following methods:

`getVenue()` – this method should return the venue.

`getCompetitors()` – this method should return a new array containing the competitors for the competition. It will construct a new array of the `Competitor` class with the length of `numberOfCompetitors` and then copy over all the instances from the `competitors` array. The new array will then be returned.

`getRaces()` – this method should return a new array containing the races for the competition. It will construct a new array of the `Race` class with the length of `numberOfRaces` and then copy over all the instances from the `races` array. The new array will then be returned.

`addRace()` – this method should check if there is a race slot available. If so it will generate a new `Race` instance using the `type` and `date` values from the parameters, add it to the `races` array and increase the `numberOfRaces` variable by one. If the race was added this method will return the race, otherwise it will return `null`.

`addCompetitor()` – this method should check if there is a competitor slot available. If so it will generate a new `Competitor` instance with the `number`, `name`, `type` and `age`, add it to the `competitors` array and increase the `numberOfCompetitors` variable by one. If the competitor was added this method will return the competitor, otherwise it will return `null`.

`getRace()` – this method should search through the `races` array and return the race with the matching number. If no matching race is found, then `null` should be returned.

`getCompetitor()` – this method should search through the `competitors` array and return the competition with the matching number. If no matching competition is found, then `null` should be returned.

The A2Program

This program manages the competition database and the user interaction with the database. When the program first runs it will display the following menu:

```
Finished reading file - 3 venues loaded
============================
Welcome to Dunrobin Cycling Competitions - by abc1234
============================
Please select an options:
1: Choose competition
2: Display races
3: Add competitor to race
4: Record race result
5: Display competition results
0: Exit

Enter selection:
```

To initially fill the database with a competition, race and competitors, the information from the file `"Competitions.txt"` (part of this file is shown below) is initially loaded into the program. **DO NOT** modify the contents of the `"Competitions.txt"` file.

```
V:Southland
C:32,Joe Ra,0,22
C:73,John Doe,0,25
C:105,Jane Mee,2,45
…
R:1,0,12,1,2014
R:2,2,12,1,2014
R:3,1,13,1,2014
…
```

Stage 5 – Complete the A2Progam class

IMPORTANT: First complete the `displayWelcome()` method

This method in the `A2Program` class displays the `String "Welcome to Dunrobin Cycling Competitions – by "` followed by **YOUR** UPI followed by a blank line. An output similar to the following is printed by this method:

```
==========================
Welcome to Dunrobin Cycling Competitions - by abcd001
==========================
```

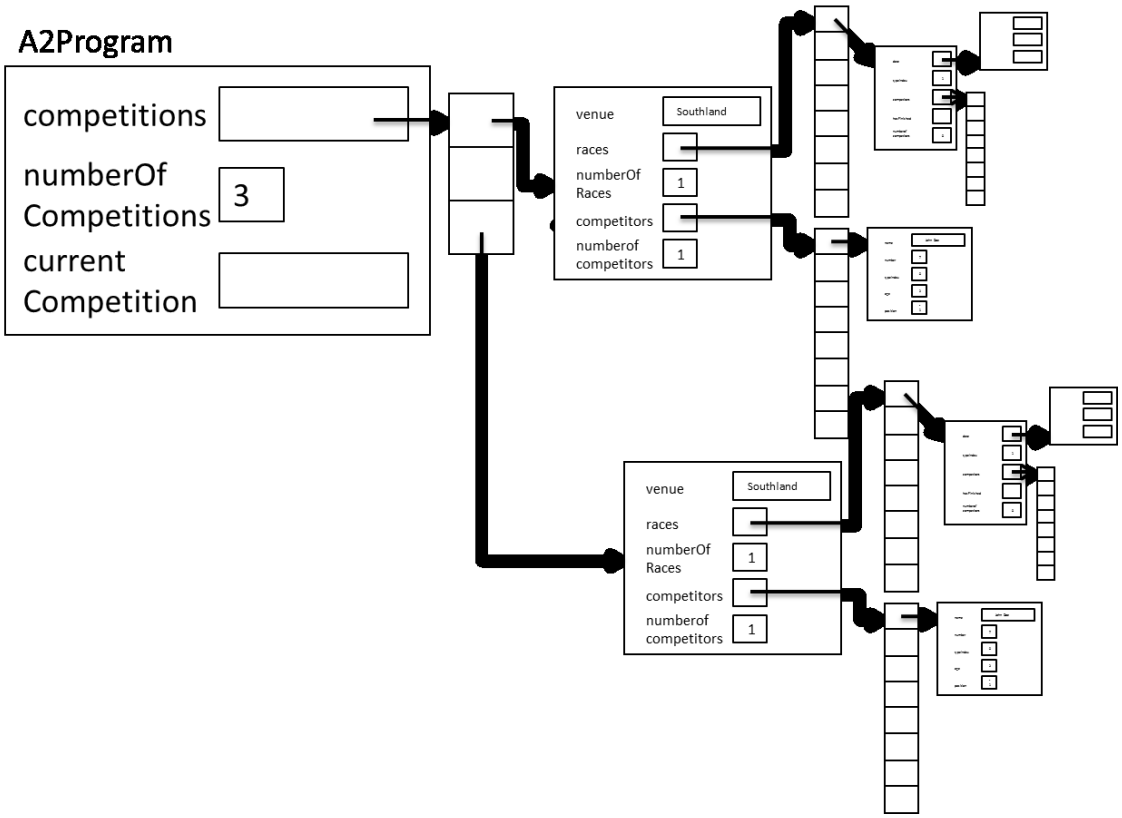
The code in the `A2Program` class has been partly completed but there are two helper methods which need to be completed by you. You will not need to modify any of the other code, but you may find it useful to understand what is happening in it.

The competition information in the `A2Program` class is held in a partly filled array of `Competition` objects (the `competitions` array). This class also has several constants (representing the user options) and three instance variables. One of these instance variables is the current competition. The other two instance variables are relevant to the code which you are required to write:

```
//the array of Competitions
private Competition[] competition;

//the number of competitions currently in the array
private int numberOfCompetitions;
```

The relevant instance variables in the `A2Program` class can be visualized as follows:



Below is part of the skeleton for the `A2Program` class.

```
public class A2Program {
   private Competition[] competitions;
   private int numberOfCompetitions;
   private Competition currentCompetition;

   public void start() { … }
   private void displayWelcome() { … }
   private void displayMenu() { … }
   private void changeCompetition() { … }
   private void displayCompetitions() { … }
   private void displayRaces(Competition current) { … }
   private void displayResults(Competition current) { … }
   private void recordRaceResult(Competition current) { … }
   private void addCompetitorToRace(Competition current) { … }
   private int getNumberFromKeyboard(String prompt) { … }
   private void readFromFile() { … }
}
```

Notes on the two methods which need to completed

`private void displayCompetitions() { … }`

This method will loop through all the competitions and display the index number for the competition and the venue. This will display output like the following:

```
0: Southland
1: Northland
2: Eastland
```

`private void displayRaces(Competition current) { … }`

If a competition has been selected it will display all the races for the competition. First it will call `getVenue()` on the current competition and display the venue. Then it will call the `getRaces()` method on the current competition to get an array of `Race` instances and display each race using the `toString()` method (you will need to use the length field on the array). If a competition has not been selected it will display an error message.

This will display output like the following when a competition has been selected (this assumes you have chosen the first competition):

```
Venue: Southland
1, Keirin [12/1/2013]: Race not finished
2, Sprint [12/1/2013]: Race not finished
3, Eliminator [12/1/2013]: Race not finished
4, Sprint [13/1/2013]: Race not finished
5, Eliminator [13/1/2013]: Race not finished
6, Eliminator [14/1/2013]: Race not finished
7, Keirin [14/1/2013]: Race not finished
```

This will display output like the following otherwise:

```
Please choose a competition before trying to display the
races.
```

You can now fully test that your program works as it should.

Full output

When you have made all your changes, do a quick run through of the program in Dr. Java. You can see the full output at

http://www.cs.auckland.ac.nz/courses/compsci101s2c/assignments/2013/A2/FullOutput.txt

Academic honesty (this is important!)

This assignment is an **assessed piece of coursework**, and it is essential that the work you submit reflects what you are capable of doing. You must not copy any source code for this assignment and submit it as your own work. You must also not allow anyone to copy your work. All submissions for this assignment will be checked, and

any cases of copying/plagiarism will be dealt with severely. We really hope there are no issues this semester in CompSci 101, so please be sensible!

Ask yourself:

"have I written the source code for this assignment myself?"

If the answer is "no", then please talk to us before the assignments are marked.

Ask yourself:

"Have I given ***anyone*** access to the source code that I have written for this assignment?"

If the answer is "yes", then please talk to us before the assignments are marked.

Once the assignments have been marked it is too late.

There is more information regarding The University of Auckland's policies on academic honesty and plagiarism here:

http://www.auckland.ac.nz/uoa/home/about/teaching-learning/honesty

Submission

You have worked hard on this assignment, and we want to make sure that you are rewarded for your effort.

You should submit **EIGHT SOURCE** files for this assignment:

- A2Program.java
- A2.java
- Keyboard.java
- MyDate.java
- Competitor.java
- Race.java
- Competition.java
- Competitions.txt

Do NOT submit your .class files.

IMPORTANT: **check that the code you submit compiles.** If your code has syntax errors of any kind, the compiler will tell you what they are – correct all syntax errors before submitting your code.

It will be much better for you to submit code that compiles but is not functionally complete than to submit code which does not compile.

Submit your file using the Web Drop Box (https://adb.auckland.ac.nz/). This link is available on the CompSci 101 website (Assignments page).

Marking

Style – 10 marks

Comment at the top of each class (containing your name, upi, date and a brief description of the class), good variable names, correct indentation, uses the constants and the helper methods provided.

Correctness – 90 marks

Stage 1 – MyDate class:	5 marks
Stage 2 – Competitor class:	20 marks
Stage 3 – Race class:	25 marks
Stage 4 – Competition class:	25 marks
Stage 5 –A2Program class:	15 marks