CITS1001 Project 1, Semester 1 2019: sports scheduling

Version 1.10, 6 April 2019. (Added Hints and Tips.)
Check the LMS to ensure that you have the latest version.
If you have any questions about any aspect of the project, submit them to *help1001*.

Project Rules

- Submission deadline: **5pm Friday 12 April 2019**
- Submit via ***cssubmit***
- Value: **10% of CITS1001**
- Project work to be done **individually**
- Project published: 31 March 2019

The project task is to construct a Java program containing your solution to the following problem. You must submit your program via *cssubmit*. No other method of submission is allowed.

You are expected to have read and understood the UWA Policy on Academic Conduct. In accordance with this policy, you may discuss with other students the general principles required to understand this project, but the work you submit must be the result of your own effort.

You must submit your project before the submission deadline above. UWA's Policy on Late Submission will apply.

Project Overview

A common format for sports leagues with *n* teams is "all-play-all", where each team plays *n-1* matches, one against each of the other teams. A second requirement is that each team plays half of its matches at home and the others at their opponents' venues. (Although obviously a perfect home/away balance is not possible for even *n*.)

Deriving a schedule for these matches is a non-trivial task if *n* is large, and many algorithms have been described. One is the so-called "polygon construction algorithm", described as Construction 1 on Page 5 of this paper (you do not need to read or understand this paper to do the project!). Given e.g. six teams T1, T2, ..., T6, this algorithm might produce a schedule like the following.

Round 1	Round 2	Round 3	Round 4	Round 5
T1 vs. T2	T6 vs. T1	T1 vs. T5	T4 vs. T1	T1 vs. T3

T3 vs. T6	T2 vs. T5	T6 vs. T4	T5 vs. T3	T4 vs. T2
T4 vs. T5	T3 vs. T4	T2 vs. T3	T6 vs. T2	T5 vs. T6

The basic algorithm works only for even numbers of teams, but it can be massaged to also work for odd numbers of teams, giving a schedule like the following for five teams.

Round 1	Round 2	Round 3	Round 4	Round 5
T2 vs. T5	T1 vs. T4	T5 vs. T3	T4 vs. T2	T3 vs. T1
T3 vs. T4	T2 vs. T3	T1 vs. T2	T5 vs. T1	T4 vs. T5

In this project you will write an implementation of the polygon construction algorithm for sports scheduling.

The Polygon Construction Algorithm

The basic algorithm is very simple. Consider for example six teams as above.

T1, T2, T3, T4, T5, T6

Clearly there will be three matches in each round.

Round 1

The first round is constructed as follows.

- The first team is at home to the second team: T1 vs. T2
- The third team is at home to the last team: T3 vs. T6
- The next team from the front (i.e. the fourth) is at home to the next team from the back (i.e. the fifth): T4 vs. T5

We then *rotate* the teams; we move the last team up to the second position on the list.

T1, T6, T2, T3, T4, T5

You can see that T6 has been moved up to the position after T1. The first team on the list is never affected by rotation.

Round 2

For the second round, we use exactly the same pairings as before.

- The first team plays the second team: T1 vs. T6
- The third team plays the last team: T2 vs. T5
- The next team from the front (i.e. the fourth) plays the next team from the back (i.e. the fifth): T3 vs. T4

Then we rotate again; this time T5 (i.e. the last team) is moved up to the second position on the list.

T1, T5, T6, T2, T3, T4

Subsequent rounds

If we do these two operations (pairing and rotating) *n-1* times, we get the entire schedule. I recommend that you

- try it yourself for the last three rounds, and check your answer above;
- try generating the first round for ten teams T1, T2, ..., T10. You should get these five matches.

  T1 vs. T2
  T3 vs. T10
  T4 vs. T9
  T5 vs. T8
  T6 vs. T7

One complication

You may have noticed that the process as described so far leaves one problem: the first team (here T1) plays all of its matches at home. We can solve this by having the first team alternate its matches home and away. So in the first, third, fifth, *etc.* rounds, T1 should play at home; and in the second, fourth, sixth, *etc.* rounds, it should play away. You can see this in the schedule above for six teams.

With this modification, the basic polygon construction algorithm is complete.

Odd numbers of teams

The basic algorithm works only for even numbers of teams. To apply it for odd numbers of teams though is easy. Consider for example five teams as above.

T1, T2, T3, T4, T5

There are three steps.

1. Add a *dummy team* at the front of the list, giving

   dummy, T1, T2, T3, T4, T5

2. Apply the algorithm for even numbers in *exactly* the same way as before.
3. Delete the match in each round that features the dummy team. Note that because the dummy team was added at the front of the list, in every round its match will be the first one.

Again, try this yourself with five teams and check your answer above, to ensure that you understand the process.

If you have any questions about any aspect of the algorithm, submit them to *help1001*.

Project Materials

Download the folder *Project1.zip* from the LMS Project 1 page. This folder contains the following.

1. Skeletons for three Java classes.

   - `Match` represents a match between two teams in a league.
   - `Round` represents one round of matches in a schedule.
   - `League` represents the complete schedule of matches for a league.

   Collectively, these classes implement the polygon construction algorithm. You are required to complete the constructors and methods for all three classes. Where the body of a constructor or method contains a comment `COMPLETE THIS`, delete the body and replace it with code which implements the required functionality, as described in the associated program comment. The numbers 1-19 in these comments describe a suggested order for you to perform these tasks. (Don't worry, most of these methods are quite simple!)

2. A fully-written class `FileIO`, which will help you to perform file input operations in the second `League` constructor. You do not need to change or submit this class.

3. Three JUnit test classes, which are provided for you to check your code. Note that the test cases are not complete; a method that gets all green ticks is not guaranteed to be completely correct. Additional test cases will be used for marking and it is your responsibility to thoroughly test your code.

4. Three input files **.txt*, for use with the second `League` constructor.

5. Seven output files **.out.txt*, showing the required output in various cases.

Project Management Tips

Before starting the project, students are expected to have

- studied the lectures and Chapters 1-4 of the text,
- completed the assigned labs during Weeks 2-5,
- read and understood the whole of this project description,
- read and understood all relevant UWA policies.

Submit any questions about any of this to *help1001*.

It is recommended that you tackle the project tasks in the order indicated; that you compile frequently; and that you test and run the code after completing each method, to ensure your code behaves as you think it ought, and does not fail any tests. You can gain good marks even if you do not complete all the methods, so long as the code you have written compiles and runs correctly. But if you submit a large body of code that does not compile or that crashes, then few marks can be awarded.

Hints and tips about the various methods may be uploaded here from time to time. Whenever that happens, the document version number will be updated.

1. Make sure that you understand exactly what `makeRound` is meant to do. Given a list of *n* teams (*n* even), it sets the field variable to hold the *n/2* matches for those teams. The matches should be in the order described in the Javadoc comment. When the method finishes, the teams list should be the same as when it started.
2. Make sure you know exactly what `testmakeRound` does. It calls `makeRound` twice, and checks the two field variables after each one.
3. Make sure that a method doesn't re-implement functionality that is defined in another method. For example, if one of your methods needs to rotate the team list, it shouldn't do that itself, it should call `rotateTeams`.
4. A new version of the `RoundTest` class has been posted on the project page.

Project Submission

Submit your three completed **.java source files** by the submission deadline above via *cssubmit*. Do not submit anything else. No other method of submission is allowed.

The file names, class names, and method signatures in your submitted code must match the original specifications exactly. The marking process is partly automated, and any changes you make which complicate this process will be penalised. If your code cannot be compiled with the original JUnit test cases provided, your submission will be penalised. It is ok to add other methods if you like, so long as you do not change the signatures of existing methods.

Common mistakes are to submit .class files in place of .java files, or to submit the test classes in place of your code. If you do one of these (or something similar), you will be notified by email as soon as we become aware, but you will be due for any applicable late penalty up to the time you re-submit. *cssubmit* makes it easy to check your files after you have submitted them - do it!

Project Assessment

Your submission will be assessed on

- completeness: how many of the methods you have written;
- correctness: whether your methods implement the specifications exactly;
- clarity: whether your code is clear to read and well-constructed.

JUnit testing classes are provided to help you check your program before submission, but the correctness of your program remains your responsibility.

Completeness and Correctness guidelines (/34)

The marks available for each constructor and method are as follows. These numbers give some indication of how difficult we believe each component is.

`Match:` constructor (1), `getHome` (1), `getAway` (1), `getDisplayValue` (1)

`Round`: constructor (1), `getMatches` (1), `addMatch` (1), `deleteMatch` (1), `getDisplayValue` (2), `makeRound` (4)

`League`: `getTeams` (1), `getFixtures` (1), constructor(`int noOfTeams`) (3), `addRound` (1), `rotateTeams` (2), `getDisplayValue` (3), `makeEvenSchedule` (4), `makeSchedule` (3), constructor(`String filename`) (2)

Methods will be assessed independently, so e.g. even if your `rotateTeams` is faulty, other methods that use `rotateTeams` could still get full marks.

Completeness and correctness will be evaluated (partly) automatically using JUnit tests. You should, therefore, try to ensure the following.

- The code passes the tests as per the specification.
- Validation should be **exactly** what is required by the assignment specification.
- Incorrectly rejecting a parameter is incorrect and will be penalized, so do not add extra validation that is not asked for.
- Do not print things to `System.out` or `System.err` that are not asked for. Use the BlueJ debugger rather than `print/ln` statements for debugging your code.
- Do not change the given file names or class names, or modify the signatures of existing methods.

Clarity guidelines (/6)

- Ensure you fill in the @author and @version fields in the header comments with your details for each submitted class.
- All code should be neatly laid out and indented, with lines no longer than 80 characters.
- Do not add fields to the classes.
- Variables should be given appropriate names.
- Select appropriate programming constructs for the method implementations.
- Keep code as simple as possible for the job it is required to do.
- If the logic you have used in a method is particularly complex, a brief comment should be added explaining the strategy you have adopted; but otherwise, code should not be commented unnecessarily.

Help!

The quickest way to get help with the project (unless you are sitting in a lab session!) is via *help1001*. You can ask questions, browse previous questions and answers, and discuss all sorts of topics with both staff and other students.

Please read and follow these guidelines.

- Do not post project code on *help1001*. For obvious reasons, this behaviour undermines the entire assessment process: as such you will be liable for punishment

under the University's Policy on Academic Conduct, even if the behaviour is unintentional.
- Before you start a new topic, check first to see if your question has already been asked and answered in an earlier topic. This will be the fastest way to get an answer!
- When you do start a new topic, give it a meaningful subject. This will help all students to search and to make use of information posted on the page.
- Feel free to post answers to other students' questions. And don't be afraid to rely on other students' answers: we read everything on *help1001* and we will correct anything inaccurate or incomplete that may be posted from time to time.
- Be civil. Speak to others as you would want them to speak to you. Remember that when you post anonymously, this just means that your name is not displayed on the page; your identity is still recorded on our systems. Poor behaviour will not be tolerated.
- Do not post project code on *help1001*, or anywhere else online. This one is worth saying twice.

If you have a question of a personal nature, do not post to *help1001*: please email Lyndon instead.

Project Extension

An extension to the project is available by which you can earn extra marks in the assessment.

- Up to 4 marks can be earned this way, which can replace marks lost in the main assessment.
- It is not possible to be awarded a total mark greater than 40 for the project.
- Late submissions will not be awarded any extension marks.
- If you attempt the extension, submit in the normal way and email Lyndon before the project deadline to let me know. No extension marks will be awarded without such notification.

The best project that earns 4/4 extension marks will be awarded a prize*.

Extension details

Another consideration in sports scheduling is for teams to alternate home and away matches as much as possible, mostly for the benefit of their fans. Define a *break* in a team's schedule as an occasion where it plays two consecutive home or away matches. It has been proven that an optimal schedule in this regard for leagues with an even number of teams has two teams with no breaks, and every other team with exactly one break. (For purposes of fairness, sometimes a schedule is preferred which has exactly one break for every team.)

For example, this is an optimal schedule for six teams T1, T2, ..., T6.

	Round 1	Round 2	Round 3	Round 4	Round 5

T1 vs. T2	T4 vs. T1	T1 vs. T6	T3 vs. T1	T1 vs. T5
T3 vs. T6	T5 vs. T3	T2 vs. T5	T4 vs. T2	T6 vs. T4
T4 vs. T5	T6 vs. T2	T3 vs. T4	T5 vs. T6	T2 vs. T3

T1 and T5 have no breaks, and the other teams have one break each.

Define an extra method in the `League` class with the following definition.

```
/**
 * Makes an optimal schedule for teams, if there are an even number of them.
 * Generate teams.size()-1 rounds, with minimal breaks in the individual teams'
schedules.
 * You may assume that teams.size() >= 2 and teams.size() is even.
*/
public void makeOptimalEvenSchedule()
```

Feel free to define other methods too if you like, but ***DO NOT*** compromise the method signatures or specifications given to you in the class skeletons. If you do this, you will lose marks in the main assessment.

There is some discussion of this issue in Section 3 on Pages 9-10 of this paper. Feel free to use this or other sources if you attempt the extension; you should include credit for any source that you use in your work.

---

*All prizes are awarded or not solely at Lyndon's discretion.

Department of Computer Science & Software Engineering