BML240 Coursework 2019

Submission Deadline, parts one and two (Moodle): 7 May 2019 1pm

Part one, 80%. Report and High Language Programme Code

Table tennis tournament simulator

This is an assessed piece of coursework, it is therefore essential to be completed and handed-in on time. If you are unclear about any aspect of the assignment, including the assessment criteria, please raise this with the module tutor well in advance of the submission deadline. The usual regulations apply to a late submission of work.

The submitted application must be in Java (using Java Intellij Idea IDE) to be marked. You must submit your source code and Java Intellij idea project files with your student ID on it.

The coursework you submit should be your work. If your coursework includes other people's ideas and material, they must be properly referenced or acknowledged. Failing to do so intentionally or unintentionally constitutes plagiarism. The University treats plagiarism seriously.

Table tennis tournament simulator details

This is not a game. This is a simulation of a table tennis tournament using Java to set up and play virtual games by players in a tournament and display the result(s). The program will rely heavily on random number generation and an element of probability will be coded.

The statistics of various elements of the tournament can then be displayed. These could include:

- Tournament winner
- All subsequent positions of players in the tournament
- Games won
- Points won in a game
- Details statistics on average scores, of point per game etc.

BML240 Coursework 2019

The rules / parameters of the game are:

Tournament

- Each tournament can have a number of players entered by at the start of the program. This would be simplest to be 8,16,32,64 etc to allow a 50% elimination per round to achieve a final of the last two remaining players. If no number of players entered the tournament will default to sixty-four.
- On start, the tournament is set up by dividing the field of players in to relevant groupings (draws) to play another player.
- The draw selection can be randomised so is different each time.
- Tournaments are comprised of a number of rounds (of matches).

Matches

- Matches can only have two players (there are no doubles matches)
- Matches are comprised of a number of games
- The number of games could be entered by the user or a default set (best of ten of example)

Games

- Games are comprised of points being won by other player.
- A game is won when one player reaches 11 points, and is two points ahead of the opponent.

Points / shots

- A point is won by one of the player whose opponent, by definition, loses it.
- Each player serves for two points before the opponent serves for two points and so on.
- A point can be won whether a player is serving or receiving.
- A player wins a shot if:
    - It is a legal stroke.
        - It does not bounce on his/her side of the net first (the serve is an exception)

    - It is not 'out' (that is, it bounces on the opponent's side of the table
    - **AND**
    - It is not returned or not returned legally by the opponent.
    - This could be one of the following:
        - The opponent misses the ball completely
        - The opponent hits the balls into the net
        - The opponent hits the ball so it bounces on his/her side of the net before crossing the net
        - The opponent hits the ball out – that is, it does not land on the opponents side of the table.
- For each shot a player receives, they attempt to return it (you can determine the level of details e.g. backhand, forehand stroke etc.
- The ball returned (if not out or in the net) will land on the left or right side of the opponent's table. This may determine whether he or she attempts to return the ball with a backhand or forehand stroke (but is not a direct relationship as the player can move position to play a preferred stroke).
- The above combinations will require some creative programming.
- (Take care not to lose focus on the object-oriented nature of the challenge. You might want to limit the amount of time merely creating complexity if it does not contribute to demonstrating your understanding of object oriented programming.)

2 Assessment Criteria

You should submit your source code and Java Intellij idea project files of your software no later than the mentioned deadline. You should submit electronically to Moodle a .pdf file with your report. The file name should be your Student ID, and should include the following:

- A GUI (graphical user interface) using Swing or JavaFX. If no GUI is used, you may lose the marks allocated for this part of your coursework.

- A UML use case diagram of your order system, UML class hierarchy diagram of your OO application design, and also one UML class diagram (one class of your choice), and one instance diagram;

- A brief description of the application including any assumptions you have made and any limitations in your implementation of the application;

The work must demonstrate (even if just for the purpose of demonstration) an understanding of.

1. The four main concepts of OOP
    a. Encapsulation
    b. Abstraction
    c. Inheritance
    d. Polymorphism
2. Java syntax
3. Java functions
4. Java libraries
5. Programming algorithms

Part one report

The report (a maximum of 1500 words) should address all requirements and the use of OOP principles such as data abstraction, inheritance and polymorphism.

Part 2 Testing. 20%

- A test schedule (no more than one page) and screen shots to evidence the testing and evaluation;
- The source code that you have written as an Appendix (the same code that you used in your demonstration);
- Some sample input and output (screenshots) to demonstrate your application is working;
- The assessment criteria and marks distribution are given in the following table.

Criteria	Report and High Language Programme Code	Testing
70% and above	The student demonstrates excellent knowledge of the principles of object-oriented languages, namely data encapsulation, abstraction, inheritance, and polymorphism. The student is able to extensively apply object-oriented techniques to write software with multiple classes, e.g., enforcing data hiding as much as possible via class encapsulation. The student demonstrates excellent know how in writing programs with graphical user interfaces. The student correctly writes object-oriented programs with complicated exception handling facilities. Excellent supporting documentation and excellent presented comments.	Accurate proof of testing is submitted which includes anticipated and actual test results. Submitted test results demonstrate that all cases were identified and tested. (In the case of combinatorial explosion, a reasonable number of cases were identified and tested.) The submitted program returns the anticipated output for all input tested by the instructor.
60%–69%	The student demonstrates sufficient knowledge in the principles of object oriented languages, namely, data encapsulation, inheritance, and polymorphism. The student is able to sufficiently apply object oriented techniques to write software applications with multiple classes, e.g., enforcing data hiding via class encapsulation. The student demonstrates considerable know-how in writing programs with graphical user interfaces.	Accurate proof of testing is submitted which includes anticipated and actual test results. Submitted test results demonstrate that nearly all cases were identified and tested. (In the case of combinatorial ex- plosion, minimally less than a reason- able number of cases were identified and tested.) The submitted program returns the anticipated output for most (but not all) input tested by the instructor.

	The student correctly writes object- oriented programs with considerable exception handling facilities. Good supporting documentation and good presented comments.	

50%–59%	The student demonstrates reasonable knowledge of the principles of object-oriented languages, namely, data encapsulation, in- heritance, and polymorphism. The student is able to apply object-oriented techniques in some key elements of software applications with multiple classes, e.g., enforcing data hiding via class privacy. The student demonstrates average know how in writing programs with graphical user interfaces. The student correctly writes object-oriented programs with an average amount of exception handling facilities. Satisfactory supporting documentation and satisfactory presented comments.	Accurate proof of testing is submitted which includes anticipated AND/OR actual test results. Submitted test results demonstrate that minimal cases were identified and tested. (In the case of combinatorial explosion, little or no consideration was given to identifying a reasonable number of cases.) The submitted program returns the anticipated output for some (but not all) of the input tested by the instructor.

Criteria	Report and High Language Programme Code	Testing

40%–49%	The student is able to demonstrate some meanings of data encapsulation, inheritance, and polymorphism, and to give simple examples of them. The student can apply some object-oriented techniques to write software applications with multiple classes, e.g., enforcing data hiding via class privacy. The student demonstrates some know-how in writing programs with graphical user interfaces. The student correctly writes object-oriented programs with some exception handling facilities. Basic supporting documentation and basic presented comments.	Accurate proof of testing is submitted which includes anticipated OR actual test results but not both. Submitted test results demonstrate that little consideration was given to identifying a complete set of test cases. The submitted program returns the anticipated output for few of the input tested by the instructor.
Less than 40%	The student is unable to demonstrate the principles of data encapsulation, inheritance, and polymorphism, or to give simple examples. The student cannot apply object oriented techniques to write software applications with multiple classes, e.g., enforcing data hiding via class privacy. The student does not demonstrate any know-how in writing programs with graphical user interfaces. The student cannot write object-oriented programs with any exception handling facilities.	A program solution was not submitted. Therefore, there is no solution to test. Tests results were not submitted. Proof of testing is submitted but includes inaccurate test results.

	No supporting documentation and no presented comments.	