COP3022: Java Programming

Programming Project # 3

See Assignment for due date

Project Outcomes:

Use Java to

1. Implement an abstract class in Java to produce reusable classes.
2. Develop a program with several classes.
3. Develop a program using an Array of object references.
4. Use method calls to perform logic steps involved in calculations and comparison.
5. Implement a solution involving Inheritance and polymorphism.

Prep Readings:

zyBooks 1 - 8. Instructor Notes.

Background Information:

1. Fantasy Football is played by millions of people, mainly online. This project will simulate a simple Fantasy Football team, that includes players and the team.
2. This project will store player information including some statics that can be used in the game.
3. The statics will be based on the current season.
4. Remember part of being a strong programmer is being flexible enough to learn about new domains, you don't have to play fantasy football or even understand football to write application in this domain. However, if you like to get a little better understand of it you can go to https://en.wikipedia.org/wiki/Fantasy_football_(American).

Project Requirements:

1. The project will consist of 6 classes as follows:
    1. Abstract **FootballPlayer** – hold information related to all players

    2. Specific Players classes designation by position.
        2.1. Subclass of **FootBallPlayer**
        2.2. Named based on the player's position such as **QuarterBack**, **RunningBack** and **DefensiveBack**. (could be an enum)
        2.3. Number of games played this season – used to generate statistics.

    3. A **FantasyFootBallTeam** class that contains a collection of **FootBallPlayer**s.

    4. A **FantasyFootBallTester** class that will create the objects and test all the classes methods.

2. **FootBallPlayer** Class – Abstract Class

    1. Instance variables include:
        1.1. Players name.
        1.2. Position – in this project it either quarterback, runningback or defensiveback.
        1.3. NFL Team (a **String**)
        1.4. Number of Games played this season.

    2. Constructors
        2.1. Default constructor, sets all instance field to a default value (empty String or 0, depending on the type)
        2.2. Parameterized constructor – sets all instance variables to the parameterized value (hint use getters for data integrity and ease of testing)

    3. Methods
        3.1. Abstract **playersRating** method that returns an integer that is calculates based on their statistics. (Specific formula is outlined in the subclasses below)

3.2. **compareTo** method

3.2.1. Similar to the compareTo method prevalent in the API.

3.2.2. Returns a positive int if the calling player's rating is higher than the parameter's player's rating

3.2.3. Returns a negative int if the calling player's rating is lower than the parameter's player's rating

3.2.4. Return zero if their ratings are equal.

3.3. Getters and Setters for all instance variables.

3.4. **toString** method

3.4.1. returns a String representation of the object
In the format *Name: Phil Sims, Position: Quarter Back, NFL Team: Giants*

3. **Quarterback** Class – subclass of **FootBallPlayer**

1. Stores statistics specific to Quarterbacks

2. Instance variables (all ints)

2.1. Pass Attempts

2.2. Pass Completed

2.3. Touchdowns Passing

2.4. Total Yards Passing

3. Constructors

3.1. Default constructor, sets all instance field to a default value. Remember to pass appropriate values to super class.

3.2. Parameterized constructor – sets all instance variables to the parameterized value (hint use setter methods for data integrity and ease of testing)

4. Methods

4.1. double completionPercentage()

Formula: passes completed / pass Attempts

4.2. double averagePassingYardsPerGame()

Formula: total yards passing/games Played

4.3. double averageTouchDownsPerGame()

Formula: touch Downs Passing/games Played

4.4. Getters and Setters for all instance variables.

4.5. **playerRating**

4.5.1. overwrite of abstract method

4.5.2. Calculation is the sum of:

*Average Touch Downs Per Game + (Completion Percentage * 100) + (Average Passing Yards Per Game/5);*

4.6. **toString** (hint use the super classes toString)

4.6.1. returns a String representation of the object in the format:

Name: Phil Sims, Position: Quarter Back, NFL Team: Giants Completion Percentage: 0.25, Average Passing Yards Per Game: 500.00 Average Touch Downs Per Game: 1.50, Player's Rating: 127

4. **RunningBack** and **DefensiveBack** are similar to the QB but have slightly different instance variables and statistic methods.

5. **RunningBack** Class
    1. Similar to the **QuarterBack** class except for slightly different instance variables and statistical methods.
    2. Instance Variables (all ints)
        2.1. Running Attempts
        2.2. Total running yards
        2.3. Touchdowns
    3. Constructors
        3.1. Default constructor, sets all instance field to a default value. Remember to pass appropriate values to super class.
        3.2. Parameterized constructor – sets all instance variables to the parameterized value (hint use setter methods for data integrity and ease of testing)
    4. Methods
        4.1. Statistical method all return a double
            4.1.1. averageYardsPerGame()
            4.1.2. averageYardsPerAttempt()
            4.1.3. averageTouchDownsPerGame()

4.2. **playerRating** is sum of averageTouchDownsPerGame() + averageYardsPerAttempt()+ (averageYardsPerGame()/5);

4.3. Getters and Setters for all instance variables.

4.4. toString similar to QuarterBack but with RunningBack info.

6. **DefensiveBack** Class
    1. Similar to the **QuarterBack** class except for slightly different instance variables and statistical methods.
    2. Instance Variables (all ints)
        2.1. Tackles
        2.2. Interceptions
        2.3. Forced Fumbles
    3. Constructors
        3.1. Default constructor, sets all instance field to a default value. Remember to pass appropriate values to super class.
        3.2. Parameterized constructor – sets all instance variables to the parameterized value (hint use setter methods for data integrity and ease of testing)
    4. Methods
        4.1. Statistical method all return a double
            4.1.1. averageTacklesPerGame()
            4.1.2. averageInterceptionsPerGame()
            4.1.3. averageForcedFumblesPerGame()
        4.2. **playerRating** is sum of (averageTacklesPerGame() + averageInterceptionsPerGame() + (averageForcedFumblesPerGame()/5)) * 10;
    5. Accessor (getters) and Mutator (setters) for all instance variables.
    6. toString similar to QuarterBack but with DefensiveBack info

7. **FantasyFootballTeam** Class
    1. Stores a collection (Array) of FootBallPlayer objects and performs basic search and toString operations
    2. Instance variables
        2.1. Team Name
        2.2. Team Owner
        2.3. Array of FootballPlayer objects
        2.4. Variable to handle a partially filled array.
    3. Parameterized constructor
        3.1. takes in a team name, owner and size of the array
        3.2. creates the array.
        3.3. sets the partially filled array value as appropriate
    4. Methods
        4.1. addPlayer
            4.1.1. Takes in a **FootBallPlayer** objects and add it to the Array.
            4.1.2. Ensures the parameter is not null and the array is not full. (Produces Error message to console)
        4.2. findPlayerbyPosition
            4.2.1. search the array for all players at a particular position
            4.2.2. returns a string of all the player at that position or a string stating no players at that position. Such as

            No player at QuarterBack found.

        4.3. **toString** – a String representation of the object such as

Team Name: Pace Palominos Owner: Bernd OK

Name: Phil Sims, Position: Quarter Back, NFL Team: Giants
Completion Percentage: 0.25, Average Passing Yards Per Game: 500.00
Average Touch Downs Per Game: 1.50, Player's Rating: 127

Name: Jim Brown, Position: Running Back, NFL Team: Browns
Running Yards Per Game: 104.17, Running Yards Per Attempt: 9.77
Average Touch Downs Per Game: 0.67, Player's Rating: 31

Name: Spider Lockart, Position: Defensive Back, NFL Team: Giants
Tackles Per Game: 1.43, Interceptions Per per Game: 0.36
Forced Fumbles Per Game: 0.21, Player's Rating 18

6. **FantasyFootBallTester** class
    1. Create at least two object per type of player
        1.1. Two QuarterBack, Running Back and Defensive Back objects.
        1.2. Hard code these object, No user or file input.
    2. Create at least two **FantasyFootBallTeam** objects
    3. Add at least one player per position to each team
    4. Compares players to completely test the **compareTo** method.
    5. Test all methods in all the class directly or indirectly.
    6. Indirect testing
        6.1. Calls a method that calls others methods
        6.2. Example the playerRating method would call all the statistical methods or the toString would call all the accessor method.

7. Sample run – Example run only, use as a guide not a solution.

Team Name: UWF Agros Owner: Mr. Pinto

Name: Phil Sims, Position: Quarter Back, NFL Team: Giants
Completion Percentage: 0.25, Average Passing Yards Per Game: 500.00
Average Touch Downs Per Game: 1.50, Player's Rating: 127

Name: Jim Brown, Position: Running Back, NFL Team: Browns
Running Yards Per Game: 104.17, Running Yards Per Attempt: 9.77
Average Touch Downs Per Game: 0.67, Player's Rating: 31

Name: Spider Lockart, Position: Defensive Back, NFL Team: Giants
Tackles Per Game: 1.43, Interceptions Per per Game: 0.36
Forced Fumbles Per Game: 0.21, Player's Rating 18

Team Name: Cantonment Stompers Owner: Mrs. Pinto

Name: Steve Young, Position: Quarter Back, NFL Team: 49ers
Completion Percentage: 0.25, Average Passing Yards Per Game: 375.00
Average Touch Downs Per Game: 1.50, Player's Rating: 102

Name: Saquon Barkley, Position: Running Back, NFL Team: Giants
Running Yards Per Game: 104.50, Running Yards Per Attempt: 11.40
Average Touch Downs Per Game: 0.67, Player's Rating: 33

Name: Aqib Talib, Position: Defensive Back, NFL Team: Rams
Tackles Per Game: 2.14, Interceptions Per per Game: 0.50
Forced Fumbles Per Game: 0.43, Player's Rating 27

Testing Finding player by position
Phil Sims

Testing comparing two players rating
Comparing Phil Sims with Steve Young
Phil Sims has a higher rating;

8. Create UML Class Diagram for the final version of your project. The diagram should include:
    1. All instance variables, including type and access specifier (+, -);
    2. All methods, including parameter list, return type and access specifier (+, -);
    3. Include Generalization and Aggregation where appropriate.
    4. The FootBallPlayerTester does not need to be included.
    5. Refer to the UML Distilled pdf on the content page as a reference for creating class diagrams.
    6. A Link to a drawing program, Dia, is also posted on the content page.

Submission Requirements:

Your project must be submitted using the instructions below. Any submissions that do not follow the stated requirements will not be graded.

1. You should have 7 files for this assignment:
    1. FootballPlayer.java
    2. QuarterBack.java
    3. RunningBack.java
    4. DefensiveBack.java
    5. FantasyFootballTeam.java
    6. FantasyFootballTester.java
    7. Simply UML Class diagram of your 5 classes, do not include the FantasyFootballTester. (Dia file or image file , jpg, gif, pdf etc)
    8. The javadoc files for all the classes except the tester.(Do not turn in)
2. Remember to compile and run your program one last time before you submit it. If your program will not compile, the graders will not be responsible for trying to test it.
3. Follow the submission requirements posted on Canvas.

Important Notes:

1. Projects will be graded on whether they correctly solve the problem, and whether they adhere to good programming practices.
2. Projects must be submitted by the time specified on the due date. Projects submitted after that time will get a grade of zero.
3. Please review UWFs academic conduct policy. Note that viewing another student's solution, whether in whole or in part, is considered academic dishonesty. Also note that submitting code obtained through the Internet or other sources, whether in whole or in part, is considered academic dishonesty. All programs submitted will be reviewed for evidence of academic dishonesty, and all violations will be handled accordingly.