THE PERFECT GAME PLAN: FINAL WRITE UP

Molly Smith
Dr. Arias
CMPT_220_112_17S
8 May, 2017

Abstract. There are many components to being an athlete than just participating in games and practice. The same goes for the coaching staff. What separates elite teams from average ones goes beyond the court; elite teams study statistics, are conscious of diet and sleep, and know their opponents inside and out. They are working to perfect every aspect of their game on and off the court. However, it is easy to get disorganized when receiving all this data and understanding what it means. The Perfect Game Plan is designed to alleviate the stress of keeping things in order and interpreting how the data affects the athlete. It lays out the information for each game in an orderly way to help the team be successful. The application prompts the user to enter which game to view. After the user enters the input, information about the game selected will be neatly displayed. Information includes game stats, high scorer, high rebounder, what the players ate before the game, how many hours of sleep the players got the night before, and the scouting report.

Introduction. The purpose of this application is to layout organized stats so that coaches and their players are able to design a game plan that will lead the team to success. Although the application can be tailored to many different sports, this project focuses solely on basketball. Often times, coaches look at their players' stats (field goal percentage, assist-to-turnover ratio, minutes played, etc.) to see where improvement needs to be made. However, this application will also address the factors off the court that may influence these stats; these factors include amount of sleep the night before a game, pregame meal, amount of recovery days between days, and many more. Having this information nicely laid out in an application will give the user the ability to see if there is a pattern between these factors and a player’s performance. Originally, the project was to implement a database, so that the user could enter, store, and compare data. However, with limited time and knowledge, an application was created instead. The motivation behind the application was to mimic what the database would do without creating the database.

Detailed System Description. The users that will be using this application are members of a sports team, including players, coaches, managers, etc. The coach should be able to update players’ and team stats after a game. However, this application will just be the beginning of this; instead of prompting the user for data for the roster, schedule, games, etc., this data will already be hardcoded. Ideally, the application will be used by the players, as well, to look at their stats, see upcoming events, and see posted messages from the coaches.

The application consists of many classes: the SwingCalendar class, the Roster class, the Schedule class, and a class for each game on the schedule. The application is run through the TestRoster class. The application prompts are run by if/else statements. The system begins by asking the user if they would like to view the calendar. If the user enters “y” then the calendar will appear by calling the SwingCalendar class and proceeds to the next question. If the user enters “n” then the system does not show the calendar, but proceeds to the next question. The next question is whether the user would like to view the roster. The same results as the calendar will appear, except a roster will be displayed, not a calendar. The next question will prompt the user to view the schedule. If the user enters “y” a schedule appears, but if the user enters “n” the system is over. If “y” is entered, the system prompts the user to enter a specific game. On the schedule, each game is given a game number, this is what the user will enter. Each game has its own class, and these classes are put in a switch statement in the TestRoster class. The user enters the game number calling that specific game's class. Once this happens, tables for the game's stats, game leaders (in points and rebounds), what the players ate for pregame and how many hours of sleep they got the night before, and the scouting report are displayed. The system also asks the user if they want to view another game to compare with the first game they entered. Another similar switch statement is created, so the system calls another game class when the user enters the game number.

The data in the roster class, the schedule class, and each individual game classes are displayed using JFrame. JFrame will show the data in a separate frame outside of the console. In the roster class, there is one JFrame consisting of data for individual players' number, last name, first name, position, height, and weight. In the schedule class, there is one JFrame consisting of data for date of the game, the opponent, the location, the time, the results, and the game number. In each individual game class, there are four JFrames. The first JFrame consists of data for the game stats, which include minutes played, field goals made, field goals attempted, field goal percentage, three-pointers made, three-pointers attempted, three-pointers percentage, free throws made, free throws attempted, free throw percentage, offensive rebounds, defensive rebounds, total rebounds, assists, turn overs, blocks, and steals for each player. The second JFrame consists of outside factors such as pregame meal, hours of sleep, and the amount of recovery days between games.

Ideally, the application would prompt the user to enter all the data for the roster class, the schedule class, and the game classes, store the inputted data, and display it in neat tables. However, this would involve making a database, so in replacement all this data was hardcoded into the separate classes. In addition, there is a separate class for each game. This project only accounts for eight games for one season. If all the games for one season were created, there would be approximately thirty separate classes, and if a coach wanted more than one season, the number of classes created would be ridiculous. A solution to this redundancy would to create one game class, and each separate game would be an instance of the class. This would make the programming simpler and more organized when more games and seasons are coded.
*UMLs found on pg. 7-8

Requirements. In the world of sports, there are millions and millions of statistics. Stats are very important to a sports franchise because it reveals a lot about how well a team is doing and how individual players are performing. Teams can learn a lot about their opponent by studying their opponent's stats. Likewise, a coaching staff learns what part of the game their players need improvement in. However, often times teams do not take in account of what affects these stats outside of the gym. It is also very easy for a coach to become disorganized with the amount of stats gathered on a daily basis. The Perfect Game Plan will address these problem of organizing statistical information for sports teams and gathering information about the team that goes beyond the gym. Ideally, it will allow coaches to have their team's stats, calendar, messages, and scouting reports for other teams all in one place. It will also pinpoint any possibility of correlation between a player's stats and any outside factor a coach has any interest in, such as amount of sleep a player gets or what the player eats before they play. The application currently is hardcoded with all the data, but the data is neatly displayed in tables, so the coach has all their information in one place. This makes it easy for a coach to compare stats and outside factors.

Literature Survey. There are many apps that provide statistical organization tools. One app in particular is TeamSnap. TeamSnap is a sports managing tool that organizes a team's schedule, member availability, statistics, calendar, messages, payment, and roster. Something that The Perfect Game Plan offers that TeamSnap does not is the correlation of outside factors and a player's individual statistics.

DakStats is another program dealing with sports and statistics. The program records game, season, and career statistics. It also provides live stats for fans and people following the game online. It prides itself in flexible entry modes, accuracy stat reports, and a short learning curve.

User Manual. The user will open the application. The application will ask the user if he or she would like to view the calendar, the roster, the schedule, and specific games. The user will enter either a "y" or an "n" for the prompts dealing with the calendar, the roster, and the schedule. When the application prompts the user to view a specific game, the user will enter the number corresponding with the game number on the schedule. Separate frames will appear for everything the user wants to see. For example, if the user enters "y" when the system asks for the roster, a roster table will appear. For specific games, four frames will appear for game stats, outside factors, stat leaders, and scouting report. The user can move the frames around to their personal liking to compare the data in each frame.

Conclusion. Ultimately, this application will be an organization tool for keeping track of team and player statistics, calendars and schedules, personal player information, and scouting reports. Ideally, the coach will be able to enter his or her players' statistics for a game, but they will also be able to make correlations for outside factors, such as food, sleep, and recovery. In the program there is a class for the roster, the schedule, the calendar, and individual games. One way to make the program more efficient would to make one game class, and make each specific game an instance of the game class.

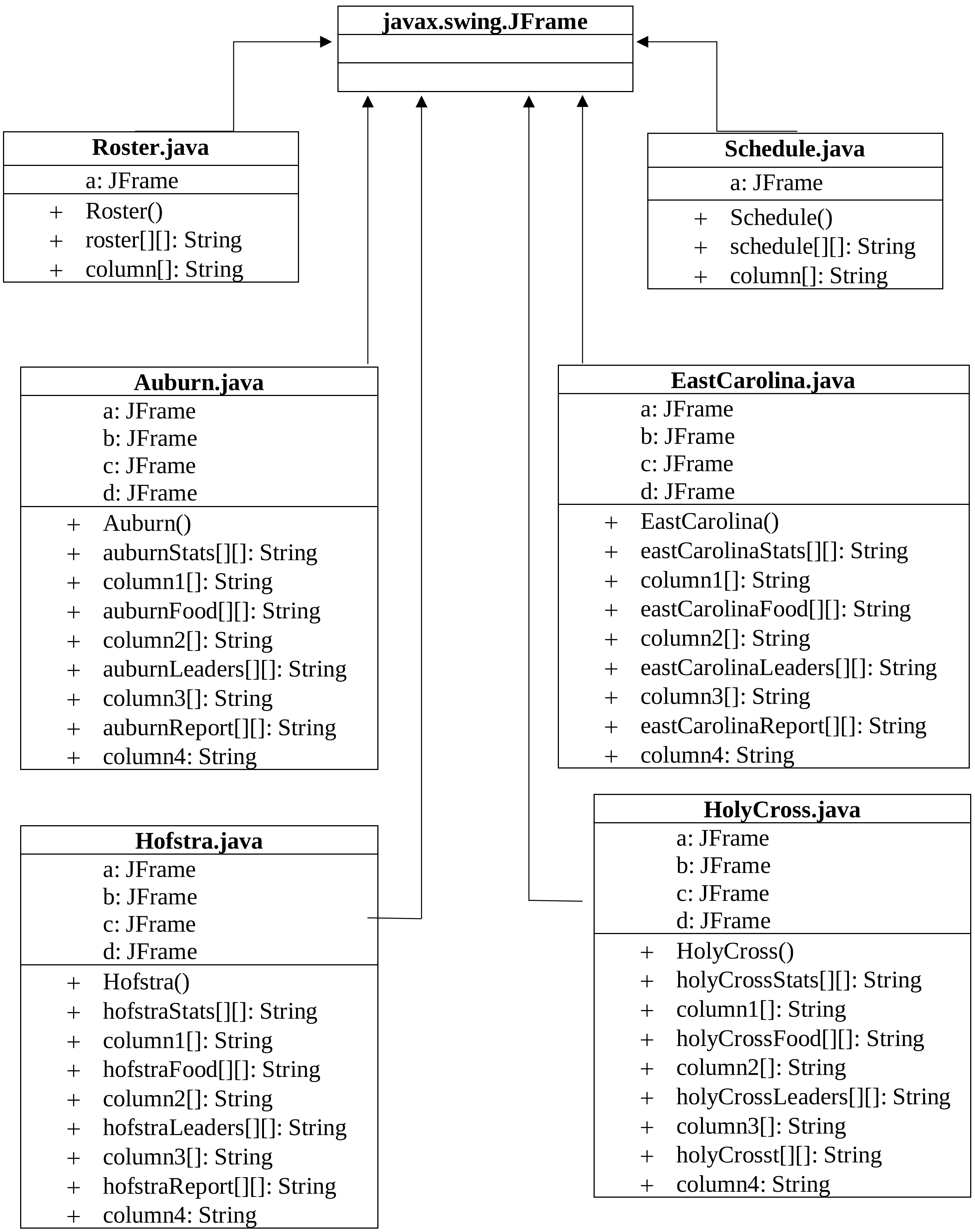
References/Bibliography.

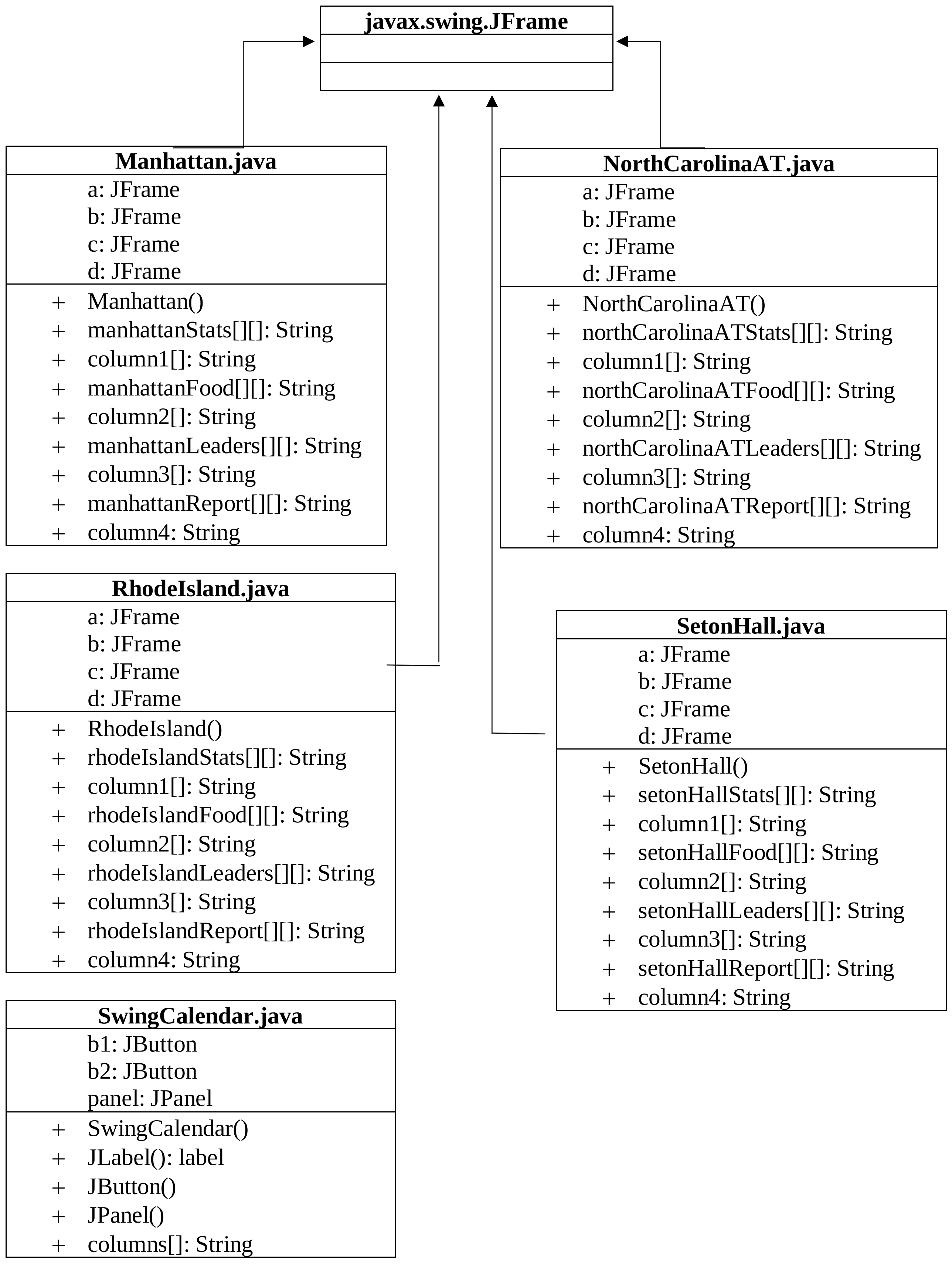
Hubbard, J. R., and Anita Huray. *Data Structures with Java.* Pearson Prentice Hall, Upper Saddle River, NJ, 2004.

Johnson, Richard A. "JAVA DATABASE CONNECTIVITY USING JAVA DB (APACHE DERBY): A TUTORIAL." *International Journal of Information, Business and Management* 8.3 (2016): 295-309. *ProQuest.* Web. 7 Apr. 2017.

Liang, Y. Daniel. *Introduction to Java Programming: Comprehensive Version.* Boston: Pearson, 2015. Print.

Martin, Robert C. *UML for Java Programmers.* Prentice Hall PTR, Upper Saddle River, NJ, 2003.


javax.swing.JFrame
Roster.java
a: JFrame
+ Roster()
+ roster[][]: String
+ column[]: String
Schedule.java
a: JFrame
+ Schedule()
+ schedule[][]: String
+ column[]: String
Auburn.java
a: JFrame
b: JFrame
c: JFrame
d: JFrame
+ Auburn()
+ auburnStats[][]: String
+ column1[]: String
+ auburnFood[][]: String
+ column2[]: String
+ auburnLeaders[][]: String
+ column3[]: String
+ auburnReport[][]: String
+ column4: String
EastCarolina.java
a: JFrame
b: JFrame
c: JFrame
d: JFrame
+ EastCarolina()
+ eastCarolinaStats[][]: String
+ column1[]: String
+ eastCarolinaFood[][]: String
+ column2[]: String
+ eastCarolinaLeaders[][]: String
+ column3[]: String
+ eastCarolinaReport[][]: String
+ column4: String
Hofstra.java
a: JFrame
b: JFrame
c: JFrame
d: JFrame
+ Hofstra()
+ hofstraStats[][]: String
+ column1[]: String
+ hofstraFood[][]: String
+ column2[]: String
+ hofstraLeaders[][]: String
+ column3[]: String
+ hofstraReport[][]: String
+ column4: String
HolyCross.java
a: JFrame
b: JFrame
c: JFrame
d: JFrame
+ HolyCross()
+ holyCrossStats[][]: String
+ column1[]: String
+ holyCrossFood[][]: String
+ column2[]: String
+ holyCrossLeaders[][]: String
+ column3[]: String
+ holyCrosst[][]: String
+ column4: String


javax.swing.JFrame
Manhattan.java
a: JFrame
b: JFrame
c: JFrame
d: JFrame
+ Manhattan()
+ manhattanStats[][]: String
+ column1[]: String
+ manhattanFood[][]: String
+ column2[]: String
+ manhattanLeaders[][]: String
+ column3[]: String
+ manhattanReport[][]: String
+ column4: String
NorthCarolinaAT.java
a: JFrame
b: JFrame
c: JFrame
d: JFrame
+ NorthCarolinaAT()
+ northCarolinaATStats[][]: String
+ column1[]: String
+ northCarolinaATFood[][]: String
+ column2[]: String
+ northCarolinaATLeaders[][]: String
+ column3[]: String
+ northCarolinaATReport[][]: String
+ column4: String
RhodeIsland.java
a: JFrame
b: JFrame
c: JFrame
d: JFrame
+ RhodeIsland()
+ rhodeIslandStats[][]: String
+ column1[]: String
+ rhodeIslandFood[][]: String
+ column2[]: String
+ rhodeIslandLeaders[][]: String
+ column3[]: String
+ rhodeIslandReport[][]: String
+ column4: String
SetonHall.java
a: JFrame
b: JFrame
c: JFrame
d: JFrame
+ SetonHall()
+ setonHallStats[][]: String
+ column1[]: String
+ setonHallFood[][]: String
+ column2[]: String
+ setonHallLeaders[][]: String
+ column3[]: String
+ setonHallReport[][]: String
+ column4: String
SwingCalendar.java
b1: JButton
b2: JButton
panel: JPanel
+ SwingCalendar()
+ JLabel(): label
+ JButton()
+ JPanel()
+ columns[]: String