/\*\*

\* AbstractFantasyTeam.java

\*

\* The AbstractFantasyTeam is an abstract class from which can be derived for different

\* kinds of BasketballPlayer lists.

\*

\* @author Jonathan Molina

\* @version Project 2 - Fantasy Basketball Draft; 18 October 2013

\*/

import java.util.ArrayList;

public abstract class AbstractFantasyTeam extends ArrayList<BasketballPlayer>

{

/\* Purposely empty. \*/

}

/\*\*

\* List of all players to choose from.

\*

\* @author Jonathan Molina

\* @version Project 2 - Fantasy Basketball Draft; 18 October 2013

\*/

import java.util.ArrayList;

public class BasketballPlayerPool extends AbstractFantasyTeam

{

/\*\*

\* Default constructor that initializes the list of BasketballPlayers with the

\* hard-coded BasketballPlayer information.

\*/

public BasketballPlayerPool()

{

/\*\*

\* Easetern Conference

\* \*/

/\* Atlantic Division \*/

addBrooklynNets();

addPhiladelphia76ers();

/\* Southeast Division \*/

addMiamiHeat();

addCharlotteBobcats();

/\*\*

\* Western Conference

\* \*/

/\* Northwest Division \*/

addOklahomaCityThunder();

addMinnesotaTimberwolves();

/\* Pacifc Division \*/

addGoldenStateWarriors();

addSacramentoKings();

}

/\*\*

\* getPlayer(int)

\*

\* Outputs a deep copy of the BasketballPlayer at the specific index.

\*

\* @param index - Position of the BasketballPlayer in the ArrayList.

\* @return BasketballPlayer object at that given index (deep copy).

\*/

public BasketballPlayer getPlayer(int index)

{

return new BasketballPlayer(this.get(index));

}

/\*\*

\* getPlayer(String)

\*

\* Only called if the player is in the list. Searches for the BasketballPlayer

\* and outputs a deep copy of it.

\*

\* @param name - BasketballPlayer's name. Used to search through list.

\* @return BasketballPlayer object with the same name (deep copy).

\*/

// Only called if player in list in order to check if valid pick

public BasketballPlayer getPlayer(String name)

{

int index = 0;

for (int cycle = 0; cycle < this.size(); cycle++)

{

if (this.get(cycle).getPlayerName().toUpperCase().equals(name.toUpperCase()))

{

index = cycle;

}

}

return new BasketballPlayer(this.get(index));

}

/\*\*

\* removePlayer

\*

\* Removes BasketballPlayer with the same name from the list.

\* Assumes the player is in the list.

\*

\* @param name - Name of the BasketballPlayer.

\*/

public void removePlayer(String name)

{

int index = 0;

for (int cycle = 0; cycle < this.size(); cycle++)

{

/\* Check to see if the name searching for. \*/

if (this.get(cycle).getPlayerName().toUpperCase().equals(name.toUpperCase()))

{

index = cycle;

}

}

this.remove(index);

}

/\*\*

\* containsPlayer

\*

\* Searches for a player in the list with the given name

\* and returns true if fond.

\*

\* @param name - Name of the BasketballPlayer.

\* @return True if player found.

\*/

public boolean containsPlayer(String name)

{

boolean contains = false;

for (int cycle = 0; cycle < this.size(); cycle++)

{

if (this.get(cycle).getPlayerName().toUpperCase().equals(name.toUpperCase()))

{

contains = true;

}

}

return contains;

}

/\*\*

\* EASTERN CONFERENCE

\*/

/\* Atlantic Division \*/

/\*\*

\* addBrooklynNets

\*

\* Adds the BasketballPlayers to the list.

\*/

private void addBrooklynNets()

{

Team nets = new Team("Nets", "Brooklyn", "Eastern", "Atlantic", 9);

this.add(new BasketballPlayer("Deron Williams", nets, 8, "PG", 20.4, 2.8, 0.440, 0.860, 2.2, 3.0, 7.7));

this.add(new BasketballPlayer("Brook Lopez", nets, 11, "C", 19.4, 1.8, 0.521, 0.758, 0.0, 6.9, 0.6));

}

/\*\*

\* addPhiladelphia76ers

\*

\* Adds the BasketballPlayers to the list.

\*/

private void addPhiladelphia76ers()

{

Team seventySixers = new Team("76ers", "Philadelphia", "Eastern", "Atlantic", 26);

this.add(new BasketballPlayer("Jrue Holiday", seventySixers, 11, "PG", 17.7, 3.7, 0.431, 0.752, 2.0, 4.2, 8.0));

this.add(new BasketballPlayer("Thaddeus Young", seventySixers, 21, "SF", 14.8, 1.2, 0.531, 0.574, 0.1, 7.5, 1.6));

}

/\* Southeast Division \*/

/\*\*

\* addMiamiHeat

\*

\* Adds the BasketballPlayers to the list.

\*/

private void addMiamiHeat()

{

Team heat = new Team("Heat", "Miami", "Eastern", "Southeast", 1);

this.add(new BasketballPlayer("LeBron James", heat, 6, "SF", 26.8, 3.0, 0.565, 0.753, 2.4, 8.0, 7.3));

this.add(new BasketballPlayer("Chris Bosh", heat, 1, "PF", 19.5, 2.1, 0.496, 0.799, 0.1, 8.9, 2.1));

}

/\*\*

\* addCharlotteBobcats

\*

\* Adds the BasketballPlayers to the list.

\*/

private void addCharlotteBobcats()

{

Team bobcats = new Team("Bobcats", "Charlotte", "Eastern", "Southeast", 28);

this.add(new BasketballPlayer("Kemba Walker", bobcats, 15, "PG", 17.7, 2.4, 0.423, 0.798, 3.0, 3.5, 5.7));

this.add(new BasketballPlayer("Gerald Henderson", bobcats, 9, "SG", 15.5, 1.6, 0.447, 0.824, 1.0, 3.7, 2.6));

}

/\*\*

\* WESTERN CONFERENCE

\*/

/\* Northwest Division \*/

/\*\*

\* addOklahomaCityThunder

\*

\* Adds the BasketballPlayers to the list.

\*/

private void addOklahomaCityThunder()

{

Team thunder = new Team("Thunder", "Oklahoma City", "Western", "Northwest", 2);

this.add(new BasketballPlayer("Kevin Durant", thunder, 35, "SF", 30.8, 3.5, 0.510, 0.905, 3.2, 7.9, 4.6));

this.add(new BasketballPlayer("Kendrick Perkins", thunder, 5, "C", 4.2, 1.4, 0.457, 0.611, 0.0, 6.0, 1.4));

}

/\*\*

\* addMinnesotaTimberwolves

\*

\* Adds the BasketballPlayers to the list.

\*/

private void addMinnesotaTimberwolves()

{

Team timberwolves = new Team("Timberwolves", "Minnesota", "Western", "Northwest", 25);

this.add(new BasketballPlayer("Kevin Love", timberwolves, 42, "PF", 18.3, 2.2, 0.352, 0.704, 1.8, 14.0, 2.3));

this.add(new BasketballPlayer("Ricky Rubio", timberwolves, 9, "PG", 10.7, 3.0, 0.360, 0.800, 1.4, 4.0, 7.3));

}

/\* Pacifc Division \*/

/\*\*

\* addSacramentoKings

\*

\* Adds the BasketballPlayers to the list.

\*/

private void addSacramentoKings()

{

Team kings = new Team("Kings", "Sacramento", "Western", "Pacific", 13);

this.add(new BasketballPlayer("Marcus Thorton", kings, 23, "SG", 12.7, 0.9, 0.429, 0.881, 3.7, 2.5, 1.3));

this.add(new BasketballPlayer("Jason Thompson", kings, 34, "PF", 10.5, 1.2, 0.502, 0.694, 0.0, 6.7, 1.0));

}

/\*\*

\* addGoldenStateWarriors

\*

\* Adds the BasketballPlayers to the list.

\*/

private void addGoldenStateWarriors()

{

Team warriors = new Team("Warrios", "Golden State", "Western", "Pacific", 10);

this.add(new BasketballPlayer("Klay Thompson", warriors, 11, "SG", 16.6, 1.9, 0.429, 0.881, 3.7, 2.5, 1.3));

this.add(new BasketballPlayer("David Lee", warriors, 10, "C", 18.5, 2.6, 0.502, 0.694, 0.0, 1.2, 3.5));

}

}

/\*\*

\* FantasyTeam.java

\*

\* The Team class represents a sports team that includes the team name and

\* which conference and division the team is in.

\*

\* @author Jonathan Molina

\* @version Project 2 - Fantasy Basketball Draft; 18 October 2013

\*/

import java.util.ArrayList;

public class FantasyTeam extends AbstractFantasyTeam

{

//private ArrayList<BasketballPlayer> fantasyTeam = new ArrayList<BasketballPlayer>();

/\*\*

\* Default constructor for FantasyTeam objects.

\*/

public FantasyTeam()

{}

/\*\*

\* Constructor for FantasyTeam and adds the BasketballPlayer to the list.

\*

\* @param newPlayer - Adds a new BasketballPlayer to the list.

\*/

public FantasyTeam(BasketballPlayer newPlayer)

{

this.add(newPlayer);

}

/\*\*

\* getTeam

\*

\* Query method for a deep copy of the ArrayList of BasketballPlayers.

\*

\* @return The ArrayList of BasketballPlayers (deep copy).

\*/

public ArrayList<BasketballPlayer> getTeam()

{

/\* Create new ArrayList of BasketballPlayers, ready for copying \*/

ArrayList<BasketballPlayer> deepCopyTeam = new ArrayList<BasketballPlayer>();

int cycle = 0; // Variable to help cycle through list.

/\* Cycle through list \*/

while (cycle < this.size())

{

/\* Calls deep copy constructor. \*/

deepCopyTeam.add(new BasketballPlayer(this.get(cycle)));

cycle++;

}

return deepCopyTeam;

}

/\*\*

\* addPlayer

\*

\* Adds a player to the Fantasy Team and returns true if successful.

\*

\* @param player - BasketballPlayer object being added to the team.

\* @return True if player added successful.

\*/

public boolean addPlayer(BasketballPlayer newPlayer)

{

boolean success = false;

/\* If no players on the list, no need to check. \*/

if (this.size() == 0)

{

this.add(newPlayer);

success = true;

}

/\* If there are other players on the team, check positions. \*/

else if (checkValid(newPlayer))

{

this.add(newPlayer);

success = true;

}

return success;

}

/\*\*

\* checkValid

\*

\* Checks if position different than the others. True if it is valid.

\*

\* @param player - BasketballPlayer object being added to the team.

\* @return True if valid move.

\*/

public boolean checkValid(BasketballPlayer player)

{

boolean valid = true, check;

for (int cycle = 0; cycle < this.size(); cycle++)

{

/\* Call method to see if same position \*/

check = this.get(cycle).isSamePosition(player);

if (check)

{

valid = false;

}

}

return valid;

}

/\*\*

\* size

\*

\* Query method for the size of the team.

\*

\* @return The size of the team.

\*/

public int getSize()

{

return this.size();

}

/\*\*

\* toString

\*

\* Overrides toString method to append the players on the team

\* and their position together.

\*

\* @return String of the players on the team and their positions.

\*/

public String toString()

{

String players = "";

for (int cycle = 0; cycle < this.size(); cycle++)

{

/\* PlayerName (Position) \*/

players += (this.get(cycle).getPlayerName() + " (" + this.get(cycle).getPlayerPosition()) + ") ";

}

return players;

}

}

/\*\*

\* Player.java

\*

\* A Player class that represents a generic sports player that includes

\* his or her player name and team information.

\*

\* @author Jonathan Molina

\* @version Project 2 - Fantasy Basketball Draft; 18 October 2013

\*/

public class Player

{

private String playerName; // The name of the player.

private Team team; // Information about the team.

/\*\*

\* Default constructor for objects of class Player.

\*/

public Player()

{}

/\*\*

\* Constructor for objects of class Player that initializes Player

\* with the given player name and Team object.

\*

\* @param pName - Name of the player.

\* @param team - Team object, contains team information.

\*/

public Player(String pName, Team tTeam)

{

playerName = pName;

team = tTeam;

}

/\*\*

\* Query method for the name of the player.

\*

\* @return The name of the player.

\*/

public String getPlayerName()

{

return playerName;

}

/\*\*

\* Query method for the player's team.

\*

\* @return The player's team.

\*/

public Team getTeam()

{

return team;

}

/\*\*

\* Overrides toString method to show the Player's name and team.

\*

\* @return String of "Name, City Team\n".

\*/

public String toString()

{

return playerName + ", " + team.getCity() + " " + team.getTeam() + "\n";

}

}

/\*\*

\* BasketballPlayer.java

\*

\* A BasketballPlayer class extends the Player class. Represents a basketball

\* that includes his or her player number, position, and statistics.

\*

\* @author Jonathan Molina

\* @version Project 2 - Fantasy Basketball Draft; 18 October 2013

\*/

public class BasketballPlayer extends Player

{

private int number; // The player's number.

private String position; // The player's position on the team.

private BasketballStats stats; // The player's basketball satistics.

/\*\*

\* Default constructor for objects of class BasketballPlayer.

\*/

public BasketballPlayer() {}

/\*\*

\* Constructor for objects of class BasketballPlayer that initializes BasketballPlayer

\* with: Basketballplayer's name, team information, number, position, and stats.

\*

\* @param name - Basketballplayer's name.

\* @param team - Basketballplayer's team information.

\* @param pNumber - BasketballPlayer's number.

\* @param pPosition - BasketballPlayer's positon.

\* @param pStats - BasketballPlayer's personal statistics.

\*/

public BasketballPlayer(String name, Team team, int pNumber, String pPosition, BasketballStats pStats)

{

super(name, team);

number = pNumber;

position = pPosition;

stats = pStats;

}

/\*\*

\* Constructor for objects of class BasketballPlayer that initializes BasketballPlayer

\* with: Basketballplayer's name, team information, number, position, and individual stats.

\*

\* @param name - Basketballplayer's name.

\* @param team - Basketballplayer's team information.

\* @param pNumber - BasketballPlayer's number.

\* @param pPosition - BasketballPlayer's positon.

\* @param pStats - BasketballPlayer's personal statistics.

\* @param avgPts - Average points.

\* @param avgTO - Average turnovers.

\* @param ftPct - Free throw percentage.

\* @param avg3pt - Average number of 3 pointers per game.

\* @param avgREB - Average number of rebounds per game.

\* @param avgAST - Average number of assists per game.

\*/

public BasketballPlayer(String name, Team team, int pNumber, String pPosition, double avgPts, double avgTO,

double fgPCT, double ftPCT, double avg3PT, double avgReb, double avgAst)

{

super(name, team);

number = pNumber;

position = pPosition;

stats = new BasketballStats(avgPts, avgTO, fgPCT, ftPCT, avg3PT, avgReb, avgAst);

}

/\*\*

\* Constructor used to make deep copies of BasketballPlayer objects.

\*

\* @param player - Basketball player.

\*/

public BasketballPlayer(BasketballPlayer player)

{

super(player.getPlayerName(), player.getTeam());

number = player.number;

position = player.position;

stats = player.stats;

}

/\*\*

\* getPlayerNumber

\*

\* Query method for the BasketballPlayer's number.

\*

\* @return The number of the BasketballPlayer.

\*/

public int getPlayerNumber()

{

return number;

}

/\*\*

\* getPlayerPosition

\*

\* Query method for the BasketballPlayer's position.

\*

\* @return The position of the BasketballPlayer.

\*/

public String getPlayerPosition()

{

return position;

}

/\*\*

\* getBBallStats

\*

\* Query method for the BasketballPlayer's personal statistics.

\*

\* @return The stats of the BasketballPlayer.

\*/

public BasketballStats getBBallStats()

{

return stats;

}

/\*\*

\* isSamePosition

\*

\* Checks if two players have the same basketball position.

\*

\* @return True if same position.

\*/

public boolean isSamePosition(BasketballPlayer player)

{

return position.equals(player.position);

}

/\*\*

\* toString

\*

\* Overrides toString method to add the basketball player's position and stats.

\*

\* @return String of the Basketball Player's position and stats.

\*/

public String toString()

{

return super.toString() + "Position: " + position + "\nStats: " + stats.toString() + "\n";

}

}

/\*\*

\* Team.java

\*

\* The Team class represents a sports team that includes the team name and

\* which conference and division the team is in.

\*

\* @author Jonathan Molina

\* @version Project 2 - Fantasy Basketball Draft; 18 October 2013

\*/

public class Team

{

private String name; // Team's name.

private String city; // Team's city.

private String conference; // Team's conference.

private String division; // Team's division.

private int rank; // Team's rank.

/\*\*

\* Constructor for objects of class Team that initializes the team name.

\*

\* @param teamName - Team's name.

\* @param cityName - Team's city.

\* @param confName - Team's conference.

\* @param divName - Team's division.

\* @param rankNum - Team's rank.

\*/

public Team(String teamName, String cityName, String confName, String divName, int rankNum)

{

name = teamName;

city = cityName;

conference = confName;

division = divName;

rank = rankNum;

}

/\*\*

\* getTeam

\*

\* Query method for the name of the team.

\*

\* @return The name of the team.

\*/

public String getTeam()

{

return name;

}

/\*\*

\* getCity

\*

\* Query method for the name of the city.

\*

\* @return The name of the city.

\*/

public String getCity()

{

return city;

}

/\*\*

\* getConference

\*

\* Query method for the name of the conference.

\*

\* @return The name of the conference.

\*/

public String getConference()

{

return conference;

}

/\*\*

\* getDivision

\*

\* Query method for the name of the division.

\*

\* @return The name of the division.

\*/

public String getDivision()

{

return division;

}

/\*\*

\* getRank

\*

\* Query method for the rank of the team

\*

\* @return The team's rank.

\*/

public int getRank()

{

return rank;

}

}

/\*\*

\* SportsStats.java

\*

\* A SportsStats class contains common sports statistics to a player.

\* Includes average points and average turnovers.

\*

\* @author Jonathan Molina

\* @version Project 2 - Fantasy Basketball Draft; 18 October 2013

\*/

public class SportsStats

{

private double avgPts; // Average points.

private double avgTO; // Average turnovers.

/\*\*

\* Constructor for SportsStats objects that initializes avgPts and avgTO.

\*

\* @param pts - Average points.

\* @param to - Average turnovers.

\*/

public SportsStats(double pts, double to)

{

avgPts = pts;

avgTO = to;

}

/\*\*

\* setPts

\*

\* Set the avgPts instance.

\*

\* @param pts - Average points.

\*/

public void setPts(double pts)

{

avgPts = pts;

}

/\*\*

\* setTO

\*

\* Set the avgTO instance.

\*

\* @param to - Average turnovers.

\*/

public void setTO(double to)

{

avgTO = to;

}

/\*\*

\* getPts

\*

\* Query method for the avgPts.

\*

\* @return The points average.

\*/

public double getPts()

{

return avgPts;

}

/\*\*

\* getAvgTO

\*

\* Query method for the avgTO.

\*

\* @return The turnover average.

\*/

public double getAvgTO()

{

return avgTO;

}

/\*\*

\* toString

\*

\* Overrides toString method to show average points and turnovers.

\*

\* @return String of the average poitns and turnovers.

\*/

public String toString()

{

return "Avg PTS: " + avgPts + " TO: " + avgTO;

}

}

/\*\*

\* BasketballStats.java

\*

\* A BasketballStats class extends SportsStats and adds specific basketball statistics.

\* Includes FG%, FT%, 3PT Avg, Reb Avg, Ast Avg, Stl Avg, and Blk Avg.

\*

\* @author Jonathan Molina

\* @version 18 October 2013

\*/

public class BasketballStats extends SportsStats

{

private double fgPCT; // Field goal percentage.

private double ftPCT; // Free throw percentage.

private double avg3PT; // Average number of 3 pointers per game.

private double avgReb; // Average number of rebounds per game.

private double avgAst; // Average number of assists per game.

/\*\*

\* Constructor for BasketballStats objects that initializes all instance variables

\* including its super class's instance variables.

\*

\* @param avPts - Average points.

\* @param avgTO - Average turnovers.

\* @param fgPct - Field goal percentage

\* @param ftPct - Free throw percentage.

\* @param avg3pt - Average number of 3 pointers per game.

\* @param avgREB - Average number of rebounds per game.

\* @param avgAST - Average number of assists per game.

\*/

public BasketballStats(double avgPts, double avgTO, double fgPct, double ftPct,

double avg3pt, double avgREB, double avgAST)

{

super(avgPts, avgTO);

fgPCT = fgPct;

ftPCT = ftPct;

avg3PT = avg3pt;

avgReb = avgREB;

avgAst = avgAST;

}

/\*\*

\* getFG

\*

\* Query method for the field goal percentage.

\*

\* @return The field goal percentage.

\*/

public double getFG()

{

return fgPCT;

}

/\*\*

\* getFT

\*

\* Query method for the free throw percentage.

\*

\* @return The free throw percentage.

\*/

public double getFT()

{

return fgPCT;

}

/\*\*

\* get3PT

\*

\* Query method for average number of 3 pointers per game.

\*

\* @return The average number of 3 pointers per game.

\*/

public double get3PT()

{

return avg3PT;

}

/\*\*

\* getREB

\*

\* Query method for the average number of rebounds per game.

\*

\* @return The average number of rebounds per game.

\*/

public double getREB()

{

return avgReb;

}

/\*\*

\* getAST

\*

\* Query method for the average number of assists per game.

\*

\* @return The average number of assists per game.

\*/

public double getAST()

{

return avgAst;

}

/\*\*

\* toString

\*

\* Overrides toString method to show the stats all in one line.

\*

\* @return String of Basketball Stats on one line.

\*/

public String toString()

{

String out = super.toString() + " FG %: " + fgPCT + " FT %: " + ftPCT + " Avg 3PT: " + avg3PT

+ " Avg REB: " + avgReb + " Avg AST: " + avgAst;

return out;

}

}

/\*\*

\* FantasyPlayer.java

\*

\* The FantasyPlayer represents the players drafting BasketballPlayers using the FantasyTeam class.

\*

\* @author Jonathan Molina

\* @version Project 2 - Fantasy Basketball Draft; 18 October 2013

\*/

public class FantasyPlayer

{

private String fantasyPlayerName; // The Fantasy Player's name.

private FantasyTeam fantasyTeam = new FantasyTeam(); // Refers to the Fantasy Player's Fantasy Team of Basketball Players.

/\*\*

\* Constructor for FantasyPlayer objects and initializes the player's name.

\*

\* @param name - Fantasy Players name.

\*/

public FantasyPlayer(String name)

{

fantasyPlayerName = name;

}

/\*\*

\* Adds a player to the Fantasy Team and returns true if successful.

\*

\* @param player - BasketballPlayer object being added to the team.

\* @return True if player added successful.

\*/

public boolean addPlayer(BasketballPlayer player)

{

return fantasyTeam.addPlayer(player);

}

/\*\*

\* size

\*

\* Query method for the size of the team.

\*

\* @return The size of the team.

\*/

public int size()

{

return fantasyTeam.getSize();

}

/\*\*

\* getName

\*

\* Query method for the name of the Fantasy Player.

\*

\* @return The name of the Fantasy Player.

\*/

public String getName()

{

return fantasyPlayerName;

}

/\*\*

\* toString

\*

\* Overrides toString method to show the Fantasy Player's name and the

\* Basketball Players on his team.

\*

\* @return String of Fantasy Player's name and the team.

\*/

public String toString()

{

return fantasyPlayerName + "'s Team: " + fantasyTeam.toString();

}

}

/\*\*

\* FantasyBasketballConsole.java

\*

\* The FantasyBasketballConsole runs the Fantasy Basketball Draft interface.

\*

\* @author Jonathan Molina

\* @version Project 2 - Fantasy Basketball Draft; 18 October 2013

\*/

import java.util.Scanner;

import java.io.IOException;

public class FantasyBasketballConsole

{

/\*\*

\* Runs the Fantasy Basketball Draft program.

\*

\* @throws IOException - Writing out to a text file.

\*/

public static void main(String[] args) throws IOException

{

/\* Create FantasyBasketballApp object in order to run the game. \*/

FantasyBasketballApp draft = new FantasyBasketballApp(new Scanner(System.in));

/\* Run the Fantasy Basketball Draft \*/

draft.run();

}

}

/\*\*

\* The underlying code for the whole Fantasy Basketball Draft.

\*

\* @author Jonathan Molina

\* @version Project 2 - Fantasy Basketball Draft; 18 October 2013

\*/

import java.util.Scanner;

import java.util.ArrayList;

import java.io.PrintWriter;

import java.io.IOException;

public class FantasyBasketballApp

{

private Scanner in; // Scanner to scan user input.

private int numFantasyPlayers; // Number of Fantasy Players.

private int numRounds; // Number of draft rounds.

private BasketballPlayerPool basketballPlayers = new BasketballPlayerPool(); // BasketballPlayer selection.

private ArrayList<FantasyPlayer> fantasyPlayers = new ArrayList<FantasyPlayer>(); // List of Fantasy Players.

public static final int kDEFAULT\_ROUNDS = 5;

/\*\*

\* Constructor for objects of class FantasyBasketballApp.

\*

\* @param input - Initialize scanner to scan user input.

\*/

public FantasyBasketballApp(Scanner input)

{

in = input;

}

/\*\*

\* run

\*

\* Runs the whole Fantasy Basketball Draft.

\*

\* @throws IOException - Writing out to a text file.

\*/

public void run() throws IOException

{

/\* Welcome screen \*/

System.out.println("\nWelcome to the Fantasy Basketball Draft! This is where you"

+ "\npractice drafting for the upcoming NBA Fantasy Basketball Season."

+ "\n\nHope you kept up with the preseason and good luck!\n");

numFantasyPlayers = getNumPlayers(); // Initializes number of Fantasy Players.

System.out.println("\nThere are 5 rounds of drafting.");

numRounds = kDEFAULT\_ROUNDS; // Initializes number of draft rounds.

/\* Prompt each Fantasy Player to enter names. \*/

for (int cycle = 0; cycle < numFantasyPlayers; cycle++)

{

System.out.print("\nName of Fantasy Player "+ (fantasyPlayers.size() + 1) +" (First and/or Last Name) : ");

fantasyPlayers.add(new FantasyPlayer(in.nextLine())); // Initialize the list of fantasy players.

}

/\* If players not ready to begin the draft \*/

if (!startDraft())

{

System.out.println("\nYou don't want to start the draft yet? Run me again when you guys are ready!\n\n");

}

/\* Otherwise, run the draft. \*/

else

{

runDraft();

System.out.println("\n\nThank you for participating!\n");

}

}

/\*\*

\* reverse

\*

\* Reverse the order of FantasyPlayers.

\*

\* @param fantasyPlayers - The list of FantasyPlayers to be reversed.

\*/

public void reverse(ArrayList<FantasyPlayer> fantasyPlayers)

{

FantasyPlayer temp;

/\* For even number of players \*/

if (fantasyPlayers.size() % 2 == 0)

{

for (int cycle = 0, reverse = fantasyPlayers.size() - 1; cycle < reverse; cycle++, reverse--)

{

temp = fantasyPlayers.get(cycle); // Store the current Fantasy Player in a temporary location.

fantasyPlayers.set(cycle, fantasyPlayers.get(reverse)); // Replace the current item with its corresponding reversed component.

fantasyPlayers.set(reverse, temp); // Store the stored Fantasy Player at its corresponding reversed index.

}

}

/\* For odd number of players \*/

else

{

for (int cycle = 0, reverse = fantasyPlayers.size() - 1; cycle != reverse; cycle++, reverse--)

{

temp = fantasyPlayers.get(cycle); // Store the current Fantasy Player in a temporary location.

fantasyPlayers.set(cycle, fantasyPlayers.get(reverse)); // Replace the current item with its corresponding reversed component.

fantasyPlayers.set(reverse, temp); // Store the stored Fantasy Player at its corresponding reversed index.

}

}

}

/\*\*

\* getNumPlayers

\*

\* Prompt the user for the number of FantasyPlayers.

\*

\* @return Number of FantasyPlayers about to play.

\*/

public int getNumPlayers()

{

Scanner in2 = new Scanner(System.in); // New scanner to scan in input.

int numPlayers; // Variable to store number of players.

System.out.print("\nEnter the number of players (between 2 and 3 inclusive): ");

numPlayers = in2.nextInt();

/\* Prompt user again if user input out of range. \*/

while (numPlayers < 2 || numPlayers > 3)

{

System.out.println("Number of players invalid.");

System.out.print("Enter the number of players (between 2 and 3 inclusive): ");

/\* Prompts user again if inputting anything else other than an integer. \*/

while (!in2.hasNextInt())

{

System.out.print("Enter the number (integer) of the players (range from 2 to 3): ");

in2.next();

}

numPlayers = in2.nextInt(); // Stores inputted integer.

}

return numPlayers;

}

/\*\*

\* startDraft

\*

\* Determines whether the users are ready to start the draft.

\*

\* @return True if the users are ready to begin the draft.

\*/

public boolean startDraft()

{

System.out.print("\n\nAre you guys ready to start the draft? (Y/N) ");

String response = in.nextLine().toUpperCase();

boolean startDraft;

if (response.equals("Y"))

{

startDraft = true;

}

else

{

startDraft = false;

}

return startDraft;

}

/\*\*

\* runDraft

\*

\* Runs the draft round by round. Displays results after and prompts

\* user if text files of the results are desired.

\*

\* @throws IOException - Writing out to a text file.

\*/

public void runDraft() throws IOException

{

int roundCount = 0; // Keeps tracks of the number of rounds.

/\* If there are still rounds left \*/

while (roundCount < numRounds)

{

/\* Each Fantasy Player takes a turn choosing a Basketball Player. \*/

for (int cycle = 0; cycle < fantasyPlayers.size(); cycle++)

{

displayMainMenu(cycle);

}

reverse(fantasyPlayers); // Reverse the order of picks after each round.

roundCount++; // Increase to move on to the next round.

/\* Indicate to the users that the round has ended. \*/

System.out.println("\n---------- END OF ROUND " + roundCount + " ----------");

}

/\* Indicate to the users that the draft is over. \*/

System.out.println("\n\n\*\*\*\*\*\*\*\*\*\* END OF DRAFT \*\*\*\*\*\*\*\*\*\*\n\n");

System.out.println("\n\*\*\*\*\*\*\*\*\*\* Results \*\*\*\*\*\*\*\*\*\*\n");

reverse(fantasyPlayers); // Reverse list one more time for original order.

displayFantasyTeams(); // Display the results after drafting.

System.out.println();

writeToFile(); // Prompts user if text file of the results are desired.

}

/\*\*

\* writeToFile

\*

\* Prompts the user if text file of the results are wanted.

\* User enters the file name.

\*

\* @throws IOException - Writing out to a text file.

\*/

public void writeToFile() throws IOException

{

System.out.print("Would you like to save the results onto a text file? (Y/N) ");

String response = in.nextLine().toUpperCase();

boolean write = false;

if (response.equals("Y"))

{

write = true;

}

if (write)

{

System.out.print("Enter the file name (without .txt extension): ");

writeDraftToFile(in.next() + ".txt");

}

}

/\*\*

\* displayFantasyTeams

\*

\* Prints out the Fantasy Player's name and Basketball Players on his/her

\* team with the position of each player also.

\*

\*/

public void displayFantasyTeams()

{

for (int cycle = 0; cycle < fantasyPlayers.size(); cycle++)

{

/\* If no players on team, indicate to users with '(None)' \*/

if (fantasyPlayers.get(cycle).size() == 0)

{

System.out.println(fantasyPlayers.get(cycle).getName() + "'s Team: (None)");

}

/\* Otherwise, begin to print out normally. \*/

else

{

System.out.println(fantasyPlayers.get(cycle).toString());

}

}

}

/\*\*

\* displayAllPlayers

\*

\* Displays all available BasketballPlayers left to be chose.

\*

\* @param all - If true, shows all players. If false, shows 5 or under.

\*/

public void displayAllPlayers(boolean all)

{

BasketballPlayer display; // Temporary BasketballPlayer object for easy method declaration.

/\* If user wants, show all players available. \*/

if (all)

{

System.out.println("\n\n\nPlayers Available (all " + basketballPlayers.size() + "): \n");

/\* Cycle through the whole list to print out. \*/

for (int cycle = 0; cycle < basketballPlayers.size(); cycle++)

{

display = basketballPlayers.getPlayer(cycle);

System.out.println(display.toString());

}

}

/\* Otherwise, show the first 5 BasketballPlayers in the list. \*/

else if (basketballPlayers.size() > 5)

{

System.out.println("\n\n\nPlayers Available (showing 5 of " + basketballPlayers.size() + "): \n");

/\* Cycle through first 5 BasketballPlayers to display. \*/

for (int cycle = 0; cycle < 5; cycle++)

{

display = basketballPlayers.getPlayer(cycle);

System.out.println(display.toString());

}

}

/\* If less than 5 players availabe, then show the rest. \*/

else

{

System.out.println("\n\n\nPlayers Available (" + basketballPlayers.size() + " players left): \n");

/\* Cycle through the rest of the list to display the last few BasketballPlayers availabe. \*/

for (int cycle = 0; cycle < basketballPlayers.size(); cycle++)

{

display = basketballPlayers.getPlayer(cycle);

System.out.println(display.toString());

}

}

}

/\*\*

\* enterPlayer

\*

\* Prompts the Fantasy Player for which BasketballPlayer to add to their Fantasy Team.

\*

\* @param fantasyPlayer - The turn Fantasy Player.

\*/

public void enterPlayer(FantasyPlayer fantasyPlayer)

{

/\* Prompt the user to enter a player name, to view all players, or the first 5 players. \*/

System.out.print("\nEnter player (first and last name), (a) for All Players, (b) to show first 5 players: ");

String playerName = in.nextLine().toUpperCase(); // Scan in user input.

/\* If wanting to see all players. \*/

if (playerName.equals("A"))

{

displayAllPlayers(true);

displayFantasyTeams();

System.out.println("\nFantasy Player " + fantasyPlayer.getName() + "'s turn to pick!");

enterPlayer(fantasyPlayer);

}

/\* Or wanting to see the first 5 players. \*/

else if (playerName.equals("B"))

{

displayAllPlayers(false);

displayFantasyTeams();

System.out.println("\nFantasy Player " + fantasyPlayer.getName() + "'s turn to pick!");

enterPlayer(fantasyPlayer);

}

/\* If the player entered a BasketballPlayer in the list. \*/

else if (basketballPlayers.containsPlayer(playerName)) // Checks to see if list still has the player.

{

BasketballPlayer player = basketballPlayers.getPlayer(playerName); // gets copy of basketball player

boolean valid = fantasyPlayer.addPlayer(player); // Checks if valid. If true, already added to fantasy team.

/\* Added player to the Fantasy Team successful? \*/

if (valid)

{

basketballPlayers.removePlayer(playerName);

}

/\* Otherwise, position is already taken by another player. \*/

else

{

System.out.println("\n" + fantasyPlayer.toString());

System.out.println("\nThe position " + player.getPlayerPosition() + " is already taken. Please enter a different player"

+ " with a different position.");

enterPlayer(fantasyPlayer);

}

}

/\* Otherwise, the user did not input an existing BasketballPlayer. \*/

else

{

System.out.println("\n'" + playerName + "' Is not available on the roster. Please choose another player.");

enterPlayer(fantasyPlayer); // Prompts the turn Fantasy Player again.

}

}

/\*\*

\* displayMainMenu

\*

\* Displays the main menu where it shows the turn Fantasy Player, the BasketballPlayers on

\* each team, and the first 5 BasketballPlayers available.

\*

\* @param currentPlayer - The index of the current Fantasy Player.

\*/

public void displayMainMenu(int currentPlayer)

{

/\* FantasyPlayer object to retrieve current Fantasy Player for easy method calling. \*/

FantasyPlayer currentFantasyPlayer = fantasyPlayers.get(currentPlayer);

System.out.println("\n\n\*\*\*\*\*\*\*\*\*\* MAIN MENU \*\*\*\*\*\*\*\*\*\*\n\n");

/\* Display the current BasketballPlayers on each team. \*/

displayFantasyTeams();

/\* Indicate the current turn Fantasy Player. \*/

System.out.println("\nFantasy Player " + currentFantasyPlayer.getName() + "'s turn to pick!");

/\* Display the first 5 BasketballPlayers Avaiblable. \*/

displayAllPlayers(false);

/\* Prompt the user to either add a BasketballPlayer or to view all the available players left. \*/

System.out.print("\nEnter (p) to add a player name to your team or (a) to view all players: ");

String input = in.nextLine().toUpperCase(); // Scan in user input.

/\* User wants to add a BasketballPlayer to their team. \*/

if (input.equals("P"))

{

/\* Prompt the turn Fantasy Player to pick a player \*/

enterPlayer(currentFantasyPlayer);

}

/\* User wants to view all BasketballPlayers available. \*/

else if (input.equals("A"))

{

/\* Display all players. \*/

displayAllPlayers(true);

/\* Display the current players in each team. \*/

displayFantasyTeams();

/\* Prompt the turn Fantasy Player to pick a player \*/

System.out.println("\nFantasy Player " + currentFantasyPlayer.getName() + "'s turn to pick!");

/\* Prompt the turn Fantasy Player to pick a player \*/

enterPlayer(currentFantasyPlayer);

}

/\* Otherwise, the user inputted something invalid. Back to the main menu. \*/

else

{

System.out.println("\nInvalid input.");

/\* Back to the main menu. \*/

displayMainMenu(currentPlayer);

}

}

/\*\*

\* writeDraftToFile

\*

\* Creats a text file containing all the Fantasy Player's Teams.

\*

\* @param fileName - Name of the text file being created.

\*/

public void writeDraftToFile(String fileName) throws IOException

{

PrintWriter out = new PrintWriter(fileName); // New PrintWrite Object to write data to a file.

out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Draft Results \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n\n"); // Title of the results text file.

for (int cycle = 0; cycle < fantasyPlayers.size(); cycle++)

{

/\* Write the Fantasy Player's name and the players on their team. \*/

out.println(fantasyPlayers.get(cycle).toString());

}

/\* Close the file stream \*/

out.close();

}

}