**APCS:: Basketball Statistics Program Activity**

This assignment is LARGE. It will involve some very serious programming, but in the end you will have a real piece of software that you have created.

**Abstract:**
The goal of the project is develop a command-line based program that helps keep statistics for a basketball game. The user interface will be reminiscent of a unix shell. Once running, your program will accept a command, process it and wait for the next command.

We want to keep track of common statistics for teams and players: field goals attempted/made (fga/fgm), rebounds (r), assists (a), free throws (ft) etc. Thus a command issued to your program might look like this:

$> fga H 21 2

Which might mean: “field goal attempted by player #21 on the Home team for 2 points”

To indicate a miss the line might be:

$> fga H 21 0

You will develop your own command and argument structure. Ultimately, your program and commands should be succinct, clear and fast enough that you could keep the statistics for a live game...guess how you’ll turn it in 😊. Your program will also have to be able to produce various kinds of reports once the game is over – individual player statistics, game box score, game event log, etc.

This assignment really has three parts:

Part I: Design the data/class structures, and implement a minimal number of them that can sufficiently represent the final design. Spend time thinking about the design, but resist the urge to over-design. You can’t think of everything ahead of time, so don’t try. But do get a feel for the overall structure. Then write code and test it.

Part II: Design a user interface that works with the minimal set of structures in order to test the program design for robustness.

Part III: Modify, enhance, build and expand. Once the structures are in place, and if they’ve been designed properly, you should be able to add features to it with relative ease. You may also expose some design flaws that will need to be fixed. It is likely that this is where you’ll spend your most time.
Part I: THE BIG PICTURE
I know you’re already dreaming up the user interface, but hold your horses. We have to design the structures upon which this program will be built. For this part I’m requiring that you start with three classes:

1. Player – represents a player on a team and their associated statistics; also provides methods for player-level stats: e.g fgpct() which would return \((\text{fga}/\text{fgm})\times 100\).

2. Team/Roster - maintains an array of players on some team, and provides other team-specific data and methods.

3. Game – maintains a record of the game; the teams involved, history of events, etc.

1. Start by designing the Player class and test it out in BlueJ. The player should have a number of fields for storing statistics about a particular game. It should also provide methods for accessing those stats and modifying them where appropriate.

2. Then make the Team class which keeps an array of players. The team class is really the program’s path to access certain players. For example it might have a method:

   ```java
   public Player getPlayer(int playerNum)
   ``

   Which would return the Player object for the player on the team with the given number. Or another method:

   ```java
   public int fga(int playerNum, int pts)
   ``

   Which modifies the field goal statistics for the player with number playerNum accordingly.

3. The Game class should be able to manage all of the “transactions” that could occur during a game. This class is really the interface to access team-level information. When an event happens the game class should access the appropriate team to update team and player statistics appropriately, as well as record a game log.

Finally, you’ll want to modify these classes so that they can be constructed from plain-text files (so that you don’t have to enter all the data by hand ahead of time). That is, imagine separate text files that keep a team’s roster. When the game class is constructed it should be able to accept two filenames that

Part I : Start Writing and building the initial structure
1. **Write the class "Player"**
   - The Player class should have private fields for name, the player's number, field goals attempted (fga), field goals made (fgm), and points scored (pts).
   - The class's constructor should accept the player's name and number as arguments, and initialize all the other other vars to appropriate values.
   - Write getter methods for each of the fields.
   - The class should have public methods:
     - `public void fga(int points)`
       which does the following:
       1. increments the field goal attempts for the player
       2. if 'points' is greater than zero
          increment field goals made
          add 'points' to the player's total points.
     - `public String toString()`
       Which provides an appropriate string-ification of the player - perhaps just their name, number and points scored.
   - Test this class on the object bench before moving on.

2. **Write the class "Team"**
   - The Team should have private fields for a team name, an array of players, and points.
   - The class's constructor should accept the team's name and the size of the roster as arguments. The constructor should initialize the array with the size given to the constructor.
   - Add getter methods for the team names and points.
   - The team should provide public methods:
     - `public void addPlayer(Player p)`
       which adds a player to the list of players on the team.
     - `public void print()`
       which prints out the list of players currently on the team.
     - `public void fga(String playerNum, int points)`
       which should locate the player with the given number in the list of players and call its fga command with the 'points' given.
• Test the team class on the Object Bench. You will probably want to create and save an Object bench with two different teams, each with a few players.

3. Write the class "Game"
• The game class should maintain private references to two teams which it should distinguish as home and visitor teams. The class should also have fields to keep track of the date and location where the game was played.

• The Game constructor should accept two Team objects and set the private fields to those teams. You can set the date and location fields to hard-coded values for the time being, or you can accept them as arguments to your constructor.

• The game class should have the public methods:

  public void print()
which should display display the rosters of both teams (simply by calling the print method for each of the two teams).

  public void fga(String HorV, String playerName, int points)
which assumes that the first argument is a single character string “H” or “V” (for “Home” or “Visitor” and that the second argument is a valid player number on the specified team. The method should call the fga command on the proper team.

4. Start the interface

   Okay, we need to make a unix-like shell. That’s easy using what you already know. Write an endless loop (I find that while(true) pretty much gets the job done) that shows some kind of prompt and waits for input. Like so:

   Scanner keyboard = new Scanner(System.in);
   String input;
   while(1)
     {
     System.out.print(">”);
     input = keyboard.nextLine();
     }

   Scanner does have methods for reading parts of lines, or data that’s to be interpreted a type other than String, but we’re not going to use that here. Our interface will simply read the whole line – we’ll leave parsing the line and checking its validity to a different method (and eventually a different class).
4.1 Write a loop that reads a line of text from the user and prints it back to the terminal. **Add a way to end the loop.** This is a simple check to make sure you can get input from the keyboard. When you’re done your output should look something like:

Welcome to Baker’s Basketball Stats Program (“BBSP”)

bbsp $> Hello there how are you?
You entered: “Hello there how are you?”

bbsp $> This is a test of reading strings.
You entered: “This is a test of reading strings.”

bbsp $> game over
You entered: “game over”
Exiting...

4.2 Parse each “token” in the string.
Our commands are going to come in the form:

```
  cmd arg1 arg2...argN
```

In order to process this we need to break up the long string, or “tokenize” it, into its separate words, so we can more easily call the proper method.

There are several ways to do this. We’ll take it in two steps so you can understand the benefit of a slightly more complicated but robust solution.

Step one: “split” the string.
The String class has a method called “split” which will break the string into an array of substrings given some delimiter.

**EXAMPLE:**

```
String foo = “This is a string”;
String[] foo2 = foo.split(“ “);
```

foo2 is now an array of sub-strings from “foo” split by a blank space (“ “). If you were to inspect foo2 you’d see:

```
foo2[0] = “This”
foo2[1] = “is”
foo2[2] = “a”
foo2[3] = “string”
```

Now modify your interface to use the split method to split the input into an array – and use a loop to print out all the values in the array. When you’re done your terminal session should look something like this:

Welcome to Baker’s Basketball Stats Program (“BBSP”)

bbsp $> Hello there how are you?
You entered:
0: Hello
1: there
2: how
3: are
4: you?

bbsp $> This is a test of reading strings.
You entered:
0: This
1: is
2: a
3: test
4: of
5: reading
6: strings.

bbsp $> game over
You entered:
0: game
1: over
Exiting...

Okay great, now we can parse the strings. You’re ready to implement some commands.

4.3 Before you can test out a command, you need to hard-code the construction of some teams into the user interface before you present the prompt. Something simple like:

```java
Team teamA = new Team("FooBars", 12);
Team teamB = new Team("BlahBots", 12);

teamA.addPlayer(new Player("Baker", 14));
...etc.
teamB.addPlayer(new Player("PlayerB1", 1));
...etc.

Game G  = new Game(teamA, teamB);
```

We’ll eventually remove this code from the user interface. It’s just temporary so we can get on with testing.

5. **implement your first command**

We want to be able to type something like the following at the command prompt:

bbsp $> fga H 14 2

This should update the field goal statistics and points for the player with number 14 on the Home team. Your code should identify the first token as “fga” and call the fga
method on the Game class. Notice that the rest of the arguments are conveniently what you need to pass to the Game class’s fga command.

5.1 Making ints out of Strings
In some cases you need to convert an input String into another datatype like, say, an int. Here’s the code:  
```java
int a = Integer.parseInt(someStr);
```
Where `someStr` is a String that does actually contain characters that represent an int.

You can now move on to Lab10 part 1.