Project: Ball Players – Part 2

Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified. Note that there is also an optional Skeleton Code assignment which will indicate level of coverage your tests have achieved (there is no late penalty since the skeleton code assignment is ungraded for this project). The files you submit to skeleton code assignment may be incomplete in the sense that method bodies have at least a return statement if applicable or they may be essentially completed files. To avoid a late penalty for the project, you must submit your completed code files to Web-CAT no later than 11:59 PM on the due date for the completed code assignment. If you are unable to submit to Web-CAT directly or via jGRASP, you should e-mail your project Java files in a zip file to your TA before the deadline. Test files are **not** required for this project. If submitted, you will be able to see your code coverage, but this will not be counted as part of your grade.

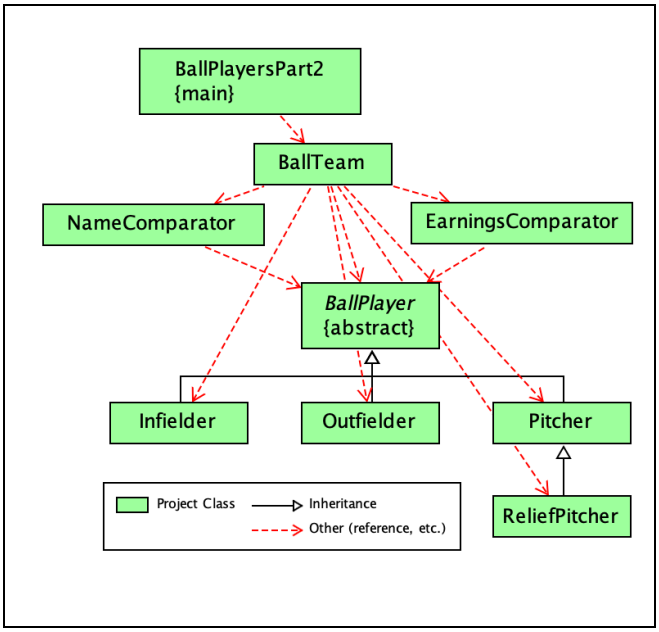
Files to submit to Web-CAT (*test files are optional*):

From Part 1

- BallPlayer.java, BallPlayerTest.java
- Outfielder.java, OutfielderTest.java
- Infielder.java, InfielderTest.java
- Pitcher.java, PitcherTest.java
- ReliefPitcher.java, *ReliefPitcherTest.java*

New in Part 2

- NameComparator.java, NameComparatorTest.java
- EarningsComparator.java, EarningsComparatorTest.java
- BallTeam.java, BallTeamTest.java
- BallPlayersPart2.java, BallPlayersPart2Test.java



Recommendations

You should create new folder for Part 2 and copy your relevant Part 1 source and optional test files to it. You should create a jGRASP project with these files in it, and then add the new source and optional test files as they are created.

Specifications – Use arrays in this project; ArrayLists are not allowed!

Overview: This project is the second of three projects that will involve the pay analysis and reporting for ball players. In Part 1, you developed Java classes that represent categories of ball players including outfielders, infielders, pitchers, and relief pitchers. In Part 2, you will implement four additional classes: (1) NameComparator that implements the Comparator interface, (2) EarningsComparator that implements the Comparator interface, (3) BallTeam that represents a team of ball players and includes several specialized methods to read in data and produce reports, and (4) BallPlayersPart2 which contains the main method for the program. Note that the main method in

BallPlayersPart2 should create a BallTeam object and then invoke the readBallPlayerFile method on the BallTeam object to add the players to the team. You can use BallPlayersPart2 in conjunction with interactions by running the program in a jGRASP canvas (or debugger with a breakpoint) and single stepping until the variables of interest are created. You can then enter interactions in the usual way. In addition to the source files, you may create an *optional* JUnit test file for each class and write one or more test methods to ensure the classes and methods meet the specifications. You should create a jGRASP project upfront and then add the new source and optional test files as they are created. All of your files should be in a single folder.

- **Outfielder, Infielder, Pitcher, and ReliefPitcher**

  Requirements and Design: No changes from the specifications in Part 1.

- **BallPlayer.java**

  Requirements and Design: In addition to the specifications in Part 1, the BallPlayer class should implement the Comparable interface.
  - `compareTo`: Takes a BallPlayer as a parameter and returns an int indicating the results of comparing ball players based on their respective numbers. This method is required since the BallPlayer class implements the Comparable interface for BallPlayer.

- **BallTeam.java**

  Requirements: The BallTeam class provides methods for reading in the data file and generating reports.

  Design: The BallTeam class has fields, a constructor, and methods as outlined below.

(1) **Fields:** All instance fields below should be private; the constants should be public.
(a) *teamName* is the name of the team.
(b) *roster* which is an array that can hold up to 24 BallPlayer objects.
(c) *playerCount* which tracks the number of players in the roster array.
(d *excludedRecords*, which is an array that can hold up to 30 String elements representing records that are read from the data file but not processed. For example, if the file contains 28 records for valid players, at most 24 can added to the *roster* array of BallPlayer objects, so four records would end up in the excludedRecords array. If the file contains 60 records for valid players, the last six would be ignored (i.e., not added to the excludedRecords array).
(e) *excludedCount* which tracks the number of records that have been added to the excludedRecords array.
(f) *MAX_PLAYERS* is public constant (i.e., final static) of type int set to 24.
(g) *MAX_EXCLUDED* is public constant (i.e., final static) of type int set to 30.

(2) **Constructor:** The constructor has no parameters and initializes the fields in BallTeam.

(3) **Methods:** Usually a class provides methods to access and modify each of its instance variables (i.e., getters and setters) along with any other required methods. The methods for BallTeam are described below.

- `getTeamName` returns the String representing the teamName.
- `setTeamName` has no return value, accepts a String, and then assigns it to teamName.
- `getRoster` returns the BallPlayer array representing the roster.
- `setRoster` has no return value, accepts a BallPlayer array, and then assigns it to roster.
- `getPlayerCount` returns the current value of playerCount.
- `setPlayerCount` has no return value, accepts an int, and sets playerCount.
- `getExcludedRecords` returns the String array representing the excludedRecords.
- `setExcludedRecords` has no return value, accepts a String array, and then assigns it to excludedRecords.
- `getExcludedCount` returns the current value of excludedCount.
- `setExcludedCount` has no return value, accepts an int, and sets excludedCount.
- `readBallPlayerFile` has no return value, accepts the data file name as a String, and throws `FileNotFoundException`. This method creates a Scanner object to read in the file and then reads it in line by line. The first line contains the team name and each of the remaining lines contains the data for a player. After reading in the team name and setting the field, the "player" lines should be processed as follows. A player line is read in, a second scanner is created on the line, and the individual values for the player are read in. After the values on the line have been read in, an "appropriate" BallPlayer object created. If there is room on the roster, the player is added to the roster array and the player count is incremented. Any player lines/records read from the file after the limit of *MAX_PLAYERS* players has been reached should be added to the excluded array and its count should be incremented. If excluded array is full, the line/record should just be skipped. The data file is a "comma separated values" file; i.e., if a line contains multiple values, the values are delimited by commas. So when you set up the scanner for the player lines, you need to set the delimiter to use a ",". Each player line in the file begins with a category for the ball player (O, I, P, and R are valid categories for ball players indicating **O**utfielder, **I**nfielder, **P**itcher, and **R**eliefPitcher respectively. The second field in the record is the player's number, followed by the data for the name, position, base salary, bonus adjustment factor, and batting average. The last items correspond to the data needed for the particular category (or subclass) of BallPlayer. If a player line does has an invalid category, add "*** invalid category ***" plus the entire line to the excluded record array. For example:

```
*** invalid category *** L,34,Sammi James,LHP,150000,3.50,.125,5,4,3.85,17
```

The files *ball_player_data.csv* and *ball_player_data2.csv* are available for download from the course web site. Be sure to see the Hints on the last page.

Below are data records in *ball_player_data.csv* (the first line/record containing the team name is followed by player lines/records):

```
Auburn Heavy Hitters
O,32,Pat Jones,RF,150000,1.25,.375,.950
I,23,Jackie Smith,3B,150000,2.50,.275,.850
P,43,Jo Williams,RHP, 150000,3.50,.125,22,4,2.85
L,34,Sammi James,LHP,150000,3.50,.125,5,4,3.85,17
R,34,Sammi James,LHP,150000,3.50,.125,5,4,3.85,17
O, 9,Pat Williams,RF,150000,1.25,.340,.950
```

- `generateReport` returns a String (does not begin with \n) that represents the Team Report, including heading, generated by processing the roster array using the original order as shown below in the example output.
- `generateReportByNumber` returns a String (does not begin with \n) that represents the Team Report (by Player Number), including heading, generated by sorting the roster array using the natural ordering, and processing the roster array as shown below in the example.
- `generateReportByName` returns a String (does not begin with \n) that represents the Team Report (by Name), including heading, generated by sorting the roster array by the player's last name, and processing the roster array as shown below in the example output.
- `generateReportByEarnings` returns a String (does not begin with \n) that represents the Team Report (by Total Earnings), including heading, generated by sorting the roster array by total earnings, and processing the roster array as shown below in the example output.
- `generateExcludedRecordsReport` returns a String (does not begin with \n) that represents the Excluded Records Report, including heading, generated by processing the excludedRecords array as shown below in the example output.

Code and Test: See examples of file reading and sorting (using Arrays.sort) in the lecture notes. The natural sorting order for BallPlayer objects is determined by the compareTo method from the Comparable interface. If *roster* is the variable for the array of BallPlayer objects, it can be sorted with the following statement.

```
BallPlayer[] bp = Arrays.copyOf(roster, playerCount);
Arrays.sort(bp);
```

The sorting order based on name is determined by the NameComparator class which implements the Comparator interface (described below). It can be sorted with the following statement.

```
BallPlayer[] bp = Arrays.copyOf(roster, playerCount);
Arrays.sort(bp, new NameComparator());
```

- **NameComparator.java**

Requirements and Design: The NameComparator class implements the Comparator interface for BallPlayer objects. Hence, it implements the following method.

- `compare(`BallPlayer `p1,` BallPlayer `p2)` compares its two arguments for order and returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second to define the ordering from lowest to highest based on *last name* followed by *first name*. To do this, you will need to extract the *last name* and *first name* from the name field (assume that each BallPlayer has only a first and last name separated by a space and that there are no other spaces in the name). To avoid uppercase and lowercase issues you should make the extracted String values either all uppercase or all lowercase prior to comparing them. Since these are String values, which are

Comparable, you may use the String compareTo method to return the negative integer, zero, or a positive integer.

Note that the *compare* method is the only method in the NameComparator class. An instance of this class will be used as one of the parameters when the Arrays.sort method is used to sort by "name". For an example of a class implementing Comparator, see lecture notes 10B Comparing Objects.

- **EarningsComparator.java**

Requirements and Design: The EarningsComparator class implements the Comparator interface for BallPlayer objects. Hence, it implements the following method.

- `compare(`BallPlayer `p1,` BallPlayer `p2)` compares its two arguments for order and returns a negative integer, zero, or a positive integer as the first argument is before, equal to, or after the second to define the ordering from highest to lowest based on the total earnings of each player.

Note that the *compare* method is the only method in the EarningsComparator class. An instance of this class will be used as one of the parameters when the Arrays.sort method is used to sort by "earnings". For an example of a class implementing Comparator, see lecture notes 10B Comparing Objects.

- **BallPlayersPart2.java**

Requirements and Design: The BallPlayersPart2 class contains the main method as described below.

- `main` - if the length of the args array is 0, prints the following message and ends the program as indicated in below in the example output section:

```
File name expected as command line argument.
Program ending.
```

Otherwise, main get the file name from the command line (i.e., args[0]), creates an instance of BallTeam, and then calls the methods in the BallTeam class to read in the data file and print the five reports as shown in the example output starting on the next page. Note that main will need a throws clause for FileNotFoundException since it will be calling the readFile method in BallTeam, which throws FileNotFoundException.

Example data files can be downloaded from the assignment page in Canvas.

Code and Test: If you have an optional test file for the BallPlayersPart2 class, you should have at least two test methods for the main method. One test method should invoke BallPlayersPart2.main(args) where args is an empty String array, and the other test method should invoke BallPlayersPart2.main(args) where args[0] is the String representing the data file name. Depending on how you implemented the main method, these two methods should cover the code in main. As for the assertion in the test method, since the BallPlayer class has public constant *MAX_PLAYERS*, you could assert that `BallPlayer.MAX_PLAYERS == 24`.

In the first test method, you can invoke main with no command line argument as follows:

```
// if you are checking for args.length == 0
// the following should exercise the code
String[] args2 = {};
BallPlayersPart2.main(args2);
```

In the second test method, you can invoke main as follows with the file name as the first (and only) command line argument:

```
String[] args = {"ball_player_data.csv"};
BallPlayersPart2.main(args);
```

If Web-CAT complains the default constructor for `BallPlayersPart2` has not been covered, you may want to include the following line of code in one of your test methods to exercise the constructor.

```
// to exercise the default constructor
BallPlayersPart2 app = new BallPlayersPart2();
```

Example Output

Output when no file name is provided as a command line argument (i.e., `args.length == 0`):

```
----jGRASP exec: java BallPlayersPart2
File name expected as command line argument.
Program ending.

 ----jGRASP: operation complete.
```

Output when `ball_player_data.csv` is successfully read:

```
----jGRASP exec: java BallPlayersPart2 ball_player_data.csv
---------------------------------------
Team Report for Auburn Heavy Hitters
---------------------------------------

32 Pat Jones (RF) .375
Base Salary: $150,000.00 Bonus Adjustment Factor: 1.25
Total Earnings: $216,796.88 (class Outfielder)

23 Jackie Smith (3B) .275
Base Salary: $150,000.00 Bonus Adjustment Factor: 2.5
Total Earnings: $237,656.25 (class Infielder)

43 Jo Williams (RHP) 22 wins, 4 losses, 2.85 ERA
Base Salary: $150,000.00 Bonus Adjustment Factor: 3.5
Total Earnings: $248,181.82 (class Pitcher)

34 Sammi James (LHP) 5 wins, 4 losses, 17 saves, 3.85 ERA
```

```
Base Salary: $150,000.00 Bonus Adjustment Factor: 3.5
Total Earnings: $214,948.45 (class ReliefPitcher)

 9 Pat Williams (RF) .340
Base Salary: $150,000.00 Bonus Adjustment Factor: 1.25
Total Earnings: $210,562.50 (class Outfielder)

----------------------------------------
Team Report for Auburn Heavy Hitters (by Number)
----------------------------------------
 9 Pat Williams RF .340
23 Jackie Smith 3B .275
32 Pat Jones RF .375
34 Sammi James LHP 5 wins, 4 losses, 17 saves, 3.85 ERA
43 Jo Williams RHP 22 wins, 4 losses, 2.85 ERA

----------------------------------------
Team Report for Auburn Heavy Hitters (by Name)
----------------------------------------
34 Sammi James LHP 5 wins, 4 losses, 17 saves, 3.85 ERA
32 Pat Jones RF .375
23 Jackie Smith 3B .275
43 Jo Williams RHP 22 wins, 4 losses, 2.85 ERA
 9 Pat Williams RF .340

----------------------------------------
Team Report for Auburn Heavy Hitters (by Earnings)
----------------------------------------
$248,181.82 43 Jo Williams RHP 22 wins, 4 losses, 2.85 ERA
$237,656.25 23 Jackie Smith 3B .275
$216,796.88 32 Pat Jones RF .375
$214,948.45 34 Sammi James LHP 5 wins, 4 losses, 17 saves, 3.85 ERA
$210,562.50  9 Pat Williams RF .340

----------------------------------------
Excluded Records Report
----------------------------------------
*** invalid category *** L,34,Sammi James,LHP,150000,3.50,.125,5,4,3.85,17


 ----jGRASP: operation complete.
```

Output when `ball_player_data2.csv` is successfully read:

```
----jGRASP exec: java BallPlayersPart2 ball_player_data2.csv
----------------------------------------
Team Report for My Test Team
----------------------------------------

21 John Doe (RF) .200
Base Salary: $150,000.00 Bonus Adjustment Factor: 2.3
Total Earnings: $217,620.00 (class Outfielder)

11 Tim Dobbs (RF) .350
Base Salary: $140,000.00 Bonus Adjustment Factor: 2.2
Total Earnings: $244,566.00 (class Outfielder)

13 Jim Dobbs (LF) .278
Base Salary: $140,000.00 Bonus Adjustment Factor: 2.0
Total Earnings: $213,948.00 (class Outfielder)

12 Joey Ledet (LF) .325
```

```
Base Salary: $150,000.00 Bonus Adjustment Factor: 2.1
Total Earnings: $248,280.00 (class Outfielder)

14 Sruthi Yalamanchili (CF) .285
Base Salary: $150,000.00 Bonus Adjustment Factor: 1.9
Total Earnings: $226,351.50 (class Outfielder)

15 Kavyashree Krishnappa (CF) .298
Base Salary: $140,000.00 Bonus Adjustment Factor: 1.8
Total Earnings: $209,839.28 (class Outfielder)

29 Sanket Chintapalli (1B) .200
Base Salary: $150,000.00 Bonus Adjustment Factor: 2.3
Total Earnings: $217,620.00 (class Infielder)

17 Jane Doe (1B) .350
Base Salary: $140,000.00 Bonus Adjustment Factor: 2.2
Total Earnings: $244,566.00 (class Infielder)

18 Buddy Bell (2B) .325
Base Salary: $150,000.00 Bonus Adjustment Factor: 2.1
Total Earnings: $248,280.00 (class Infielder)

19 Oscar De La Hoya (2B) .278
Base Salary: $140,000.00 Bonus Adjustment Factor: 2.0
Total Earnings: $213,948.00 (class Infielder)

20 David Umphress (3B) .285
Base Salary: $150,000.00 Bonus Adjustment Factor: 1.9
Total Earnings: $226,351.50 (class Infielder)

10 Mikie Mahtook (3B) .298
Base Salary: $140,000.00 Bonus Adjustment Factor: 1.8
Total Earnings: $209,839.28 (class Infielder)

22 Matty Ott (SS) .200
Base Salary: $150,000.00 Bonus Adjustment Factor: 2.3
Total Earnings: $217,620.00 (class Infielder)

23 Louis Coleman (SS) .350
Base Salary: $140,000.00 Bonus Adjustment Factor: 2.2
Total Earnings: $244,566.00 (class Infielder)

25 Aaron Nola (RHP) 5 wins, 11 losses, 1.3 ERA
Base Salary: $140,000.00 Bonus Adjustment Factor: 2.0
Total Earnings: $110,782.61 (class Pitcher)

26 John Malkovic (RHP) 6 wins, 10 losses, 1.4 ERA
Base Salary: $150,000.00 Bonus Adjustment Factor: 1.9
Total Earnings: $131,000.00 (class Pitcher)

27 Louis Giglio (RHP) 7 wins, 9 losses, 1.5 ERA
Base Salary: $140,000.00 Bonus Adjustment Factor: 1.8
Total Earnings: $131,936.00 (class Pitcher)

28 Gus Malzan (LHP) 8 wins, 8 losses, 1.6 ERA
Base Salary: $150,000.00 Bonus Adjustment Factor: 2.3
Total Earnings: $150,000.00 (class Pitcher)

16 Tim Brando (LHP) 9 wins, 7 losses, 1.7 ERA
Base Salary: $140,000.00 Bonus Adjustment Factor: 2.2
Total Earnings: $149,125.93 (class Pitcher)

30 Todd Strange (LHP) 10 wins, 6 losses, 2 saves, 1.8 ERA
```

```
Base Salary: $150,000.00 Bonus Adjustment Factor: 2.1
Total Earnings: $172,500.00 (class ReliefPitcher)

31 Blake Dean (RHP) 11 wins, 5 losses, 3 saves, 1.9 ERA
Base Salary: $140,000.00 Bonus Adjustment Factor: 2.0
Total Earnings: $168,965.52 (class ReliefPitcher)

32 Brian Wilson (RHP) 12 wins, 4 losses, 4 saves, 2.0 ERA
Base Salary: $150,000.00 Bonus Adjustment Factor: 1.9
Total Earnings: $188,000.00 (class ReliefPitcher)

33 Johnny Manziel (RHP) 13 wins, 3 losses, 2 saves, 2.1 ERA
Base Salary: $140,000.00 Bonus Adjustment Factor: 1.8
Total Earnings: $172,516.13 (class ReliefPitcher)

34 Green Lantern (LHP) 14 wins, 2 losses, 3 saves, 2.2 ERA
Base Salary: $150,000.00 Bonus Adjustment Factor: 2.3
Total Earnings: $203,906.25 (class ReliefPitcher)

--------------------------------------
Team Report for My Test Team (by Number)
--------------------------------------
10 Mikie Mahtook 3B .298
11 Tim Dobbs RF .350
12 Joey Ledet LF .325
13 Jim Dobbs LF .278
14 Sruthi Yalamanchili CF .285
15 Kavyashree Krishnappa CF .298
16 Tim Brando LHP 9 wins, 7 losses, 1.7 ERA
17 Jane Doe 1B .350
18 Buddy Bell 2B .325
19 Oscar De La Hoya 2B .278
20 David Umphress 3B .285
21 John Doe RF .200
22 Matty Ott SS .200
23 Louis Coleman SS .350
25 Aaron Nola RHP 5 wins, 11 losses, 1.3 ERA
26 John Malkovic RHP 6 wins, 10 losses, 1.4 ERA
27 Louis Giglio RHP 7 wins, 9 losses, 1.5 ERA
28 Gus Malzan LHP 8 wins, 8 losses, 1.6 ERA
29 Sanket Chintapalli 1B .200
30 Todd Strange LHP 10 wins, 6 losses, 2 saves, 1.8 ERA
31 Blake Dean RHP 11 wins, 5 losses, 3 saves, 1.9 ERA
32 Brian Wilson RHP 12 wins, 4 losses, 4 saves, 2.0 ERA
33 Johnny Manziel RHP 13 wins, 3 losses, 2 saves, 2.1 ERA
34 Green Lantern LHP 14 wins, 2 losses, 3 saves, 2.2 ERA

--------------------------------------
Team Report for My Test Team (by Name)
--------------------------------------
18 Buddy Bell 2B .325
16 Tim Brando LHP 9 wins, 7 losses, 1.7 ERA
29 Sanket Chintapalli 1B .200
23 Louis Coleman SS .350
19 Oscar De La Hoya 2B .278
31 Blake Dean RHP 11 wins, 5 losses, 3 saves, 1.9 ERA
13 Jim Dobbs LF .278
11 Tim Dobbs RF .350
17 Jane Doe 1B .350
21 John Doe RF .200
27 Louis Giglio RHP 7 wins, 9 losses, 1.5 ERA
15 Kavyashree Krishnappa CF .298
34 Green Lantern LHP 14 wins, 2 losses, 3 saves, 2.2 ERA
12 Joey Ledet LF .325
```

```
10 Mikie Mahtook 3B .298
26 John Malkovic RHP 6 wins, 10 losses, 1.4 ERA
28 Gus Malzan LHP 8 wins, 8 losses, 1.6 ERA
33 Johnny Manziel RHP 13 wins, 3 losses, 2 saves, 2.1 ERA
25 Aaron Nola RHP 5 wins, 11 losses, 1.3 ERA
22 Matty Ott SS .200
30 Todd Strange LHP 10 wins, 6 losses, 2 saves, 1.8 ERA
20 David Umphress 3B .285
32 Brian Wilson RHP 12 wins, 4 losses, 4 saves, 2.0 ERA
14 Sruthi Yalamanchili CF .285

---------------------------------------
Team Report for My Test Team (by Earnings)
---------------------------------------
$248,280.00 12 Joey Ledet LF .325
$248,280.00 18 Buddy Bell 2B .325
$244,566.00 11 Tim Dobbs RF .350
$244,566.00 17 Jane Doe 1B .350
$244,566.00 23 Louis Coleman SS .350
$226,351.50 14 Sruthi Yalamanchili CF .285
$226,351.50 20 David Umphress 3B .285
$217,620.00 21 John Doe RF .200
$217,620.00 29 Sanket Chintapalli 1B .200
$217,620.00 22 Matty Ott SS .200
$213,948.00 13 Jim Dobbs LF .278
$213,948.00 19 Oscar De La Hoya 2B .278
$209,839.28 15 Kavyashree Krishnappa CF .298
$209,839.28 10 Mikie Mahtook 3B .298
$203,906.25 34 Green Lantern LHP 14 wins, 2 losses, 3 saves, 2.2 ERA
$188,000.00 32 Brian Wilson RHP 12 wins, 4 losses, 4 saves, 2.0 ERA
$172,516.13 33 Johnny Manziel RHP 13 wins, 3 losses, 2 saves, 2.1 ERA
$172,500.00 30 Todd Strange LHP 10 wins, 6 losses, 2 saves, 1.8 ERA
$168,965.52 31 Blake Dean RHP 11 wins, 5 losses, 3 saves, 1.9 ERA
$150,000.00 28 Gus Malzan LHP 8 wins, 8 losses, 1.6 ERA
$149,125.93 16 Tim Brando LHP 9 wins, 7 losses, 1.7 ERA
$131,936.00 27 Louis Giglio RHP 7 wins, 9 losses, 1.5 ERA
$131,000.00 26 John Malkovic RHP 6 wins, 10 losses, 1.4 ERA
$110,782.61 25 Aaron Nola RHP 5 wins, 11 losses, 1.3 ERA

---------------------------------------
Excluded Records Report
---------------------------------------
*** invalid category *** H,24,Austin Nola,LHP,150000,2.1,0.325,4,12,1.2
R,35,Bruce Wayne,LHP,140000,2.2,0.35,15,1,2.3,4
R,36,Bill Gates,LHP,150000,2.1,0.325,16,0,2.4,2


 ----jGRASP: operation complete.
```

Hints

1. In the readFile method, if you use a switch statement to determine the category, you should use type char for the switch expression rather than String; that is, each of the case labels should be of type char (e.g., `case 'E':` rather than type String (e.g., `case "E":`). When the switch type is String, the code coverage tool used by Web-CAT fails to detect that the default case is covered. If category is the reference to the String that contains the category code, then the following statement returns the category code as type char.

```
category.charAt(0)
```

2. If a player line has an invalid category, add "*** invalid category ***" plus the entire line to the excluded record array.
   For example, the following record has an invalid category (L).
   ```
   L,34,Sammi James,LHP,150000,3.50,.125,5,4,3.85,17
   ```

   This would result in the following String being added to the excluded record array.
   ```
   *** invalid category *** L,34,Sammi James,LHP,150000,3.50,.125,5,4,3.85,17
   ```