Project 10: Softball Players – Part 2

Mon 4/18 Update:
See highlighted text on p. 3

Due: Skeleton Code (ungraded – checks class names, method names, parameters, and return types)
Completed Code – **Thursday, April 21, 2016 by 11:59 p.m.**

Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified above (see the Lab Guidelines for information on submitting project files). You may submit your skeleton code files until the project due date but should try to do this by Friday (there is no late penalty since this is ungraded). You must submit your completed code files to Web-CAT before 11:59 PM on the due date for the completed code to avoid a late penalty for the project. You may submit your completed code up to 24 hours after the due date, but there is a late penalty of 15 points. No projects will be accepted after the one-day late period. If you are unable to submit via Web-CAT, you should e-mail your project Java files in a zip file to your lab instructor before the deadline. The Completed Code will be tested against your test methods in your JUnit test files and against the usual correctness tests. The grade will be determined, in part, by the tests that you pass or fail and the level of coverage attained in your Java source files by your test methods.

Files to submit to Web-CAT:

From Project 9

- SoftballPlayer.java
- Outfielder.java, OutfielderTest.java
- Infielder.java, InfielderTest.java
- Pitcher.java, PitcherTest.java
- ReliefPitcher.java, ReliefPitcherTest.java

New in Project 10

- NameComparator.java, NameComparatorTest.java
- RatingComparator.java, RatingComparatorTest.java
- SoftballTeam.java, SoftballTeamTest.java
- SoftballPlayersPart2.java, SoftballPlayersPart2Test.java

Recommendations

You should create new folder for Project 10 and copy your relevant Project 9 source and test files to it. You should create a jGRASP project and add the class and test files as they are created.

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

Overview: This project is Part 2 of three that will involve the rating and reporting for softball players. In Part 1, you developed Java classes that represent categories of softball 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) RatingComparator that implements the Comparator interface, (3) SoftballTeam that represents a team of softball players and includes several specialized methods, and (4) SoftballPlayersPart2 which contains the main method for the program. Note that the main method in SoftballPlayersPart2 should create a SoftballTeam object and then call the readPlayerFile on the SoftballTeam object, which will add softball players to

the team as the data is read in from a file. You can use SoftballPlayersPart2 in conjunction with interactions by running the program in the 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 will create a 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 source and 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 Project 9.

- **SoftballPlayer.java**

  Requirements and Design: In addition to the specifications in Project 9, the SoftballPlayer class should implement the Comparable interface for SoftballPlayer, which means the following method must be implemented in SoftballPlayer.
  - `compareTo`: Takes a SoftballPlayer as a parameter and returns an int indicating the results of comparing the two softball players based on their respective numbers.

- **SoftballTeam.java**

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

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

(1) **Fields:** All fields below should be private.
(a) *teamName* is the name of the team.
(b) *roster* which is an array that can hold up to 24 SoftballPlayer objects.
(c) *playerCount* is a field 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, at most 24 can added to the *roster* array of SoftballPlayer objects, so four records would end up in the excludedRecords array. If the file contains 60 records (24 players and 30 excluded records), the last six would be ignored (i.e., not added to the excludedRecords array).
(e) *excludedCount* is a field which tracks the number of records that have been added to the excludedRecords array.
(f) teamCount is a static field which tracts the number of SoftballTeam objects created.
(g) *MAX_PLAYERS* is public constant (i.e., final static) of type int set to 24.
(h) *MAX_EXCLUDED* is public constant (i.e., final static) of type int set to 30.

(2) **Constructor:** The constructor has no parameters, initializes the instance fields in SoftballTeam, and increments the static field teamCount.

(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 SoftballTeam 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 SoftballPlayer array representing the roster.
- `setRoster` has no return value, accepts a SoftballPlayer array, and then assigns it to roster.
- `getPlayerCount` returns the current value of playerCount.
- `setPlayerCount` has no return value, accepts an int, and assigns it to 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 to it.
- `getTeamCount` is a static method that returns the current value of teamCount.
- `resetTeamCount` is a static method that has no return value and sets teamCount to 0.
- `readPlayerFile` has no return value and accepts the data file name as a String. Remember to include the `throws IOException` clause in the method declaration. 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, 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" SoftballPlayer 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 with appropriate prefix message (`Maximum player count of 24 exceeded for:`) 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 "," by calling the `useDelimiter(",")` method on the Scanner object. Each player line in the file begins with a category for the softball player (O, I, P, and R are valid categories for softball 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, specialization factor, and batting average. The last items correspond to the data needed for the particular category (or subclass) of SoftballPlayer. A record with an invalid category should be added to the excluded array with appropriate prefix message (`*** invalid category ***`) and its count should be incremented. The file *softball_player_data.csv* is available for download from the course web site. Below are example data records (the first line/record containing the team name is followed by player lines/records):

```
Auburn Stars
O,32,Pat Jones,RF,1.0,.375,.950
I,23,Jackie Smith,3B,1.25,.275,.850
P,43,Jo Williams,RHP,2.0,.125,22,4,2.85
```

```
R,34,Sammi James,LHP,2.0,.125,5,4,3.85,17
L,23,Gayle Adams,2B,1.25,.225,.875
O, 9,Pat Williams,RF,1.0,.480,.950
```

- `generateReport` processes the roster array using the original order from the file to produce the Team Report which is returned as a String. See the example output below.
- `generateReportByNumber` sorts the roster array using the natural ordering, and processes the roster array to produce the Team Report (by Player Number) which is returned as a String. See the example output below.
- `generateReportByName` sorts the roster array by name, and processes the roster array to produce the Team Report (by Name) which is returned as a String. See the example output below.
- `generateReportByRating` sorts the roster array by rating, and processes the roster array to produce the Team Report (by Rating) which is returned as a String. See the example output below.
- `generateExcludedRecordsReport` processes the excludedRecords array to produce the Excluded Records Report which is returned as a String. See the example output below.

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

```
SoftballPlayer[] 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.

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

- **NameComparator.java**

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

- `compare(SoftballPlayer p1, SoftballPlayer p2)` that defines 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 the first name is separated from the last name with a space and that there may be other spaces in the last name; e.g., Gigi de la Hoya). To avoid uppercase and lowercase issues you should make the extracted String values either all uppercase or all lowercase prior to comparing them.

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 class notes 10B Comparing Objects.

- **RatingComparator .java**

  Requirements and Design: The RatingComparator class implements the Comparator interface for SoftballPlayer objects. Hence, it implements the following method.
  - `compare(SoftballPlayer p1, SoftballPlayer p2)` that defines the ordering from highest to lowest based on the rating of each player.

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

- **SoftballPlayersPart2.java**

  Requirements and Design: The SoftballPlayersPart2 class has only a main method as described below.
  - `main` reads in the file name as the first argument, args[0], of the command line arguments, creates an instance of SoftballTeam, and then calls the readPlayerFile method in the SoftballTeam class to read in the data file and generate the five reports as shown in the output examples beginning on page 7. If no command line argument is provided, the program should indicate this and end as shown in the first example output on page 7. Example data files, *softball_player_data.csv* and *softball_player_data2.csv* can be downloaded from the Lab assignment web page.

  Code and Test: In your JUnit test file for the SoftballPlayersPart2 class, you should have at least two test methods for the main method. One test method should invoke SoftballPlayersPart2.main(args) where args is an empty String array, and the other test method should invoke SoftballPlayersPart2.main(args) where args[0] is the String representing the data file name. Depending on how you implemented the main method, these two test methods should cover the code in main. In each test method, you should reset *teamCount*, call your main method, then assert that `getTeamCount()` is zero for the test with no file name and one for the test with *softball_player_data.csv* as the file name. The following statements accomplish the test.

```
SoftballTeam.resetTeamCount(0);
SoftballPlayersPart2.main(null);
Assert.assertEquals("Team count should be 0. ",
                    0, SoftballTeam.getTeamCount());
```

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

```
// If you are checking for args.length == 0
// in SoftballPlayersPart2, the following should exercise
// the code for true.
String[] args1 = {};  // an empty String[]
SoftballPlayersPart2.main(args1);
Assert.assertEquals("Team count should be 0. ",
                    0, SoftballTeam.getTeamCount());
```

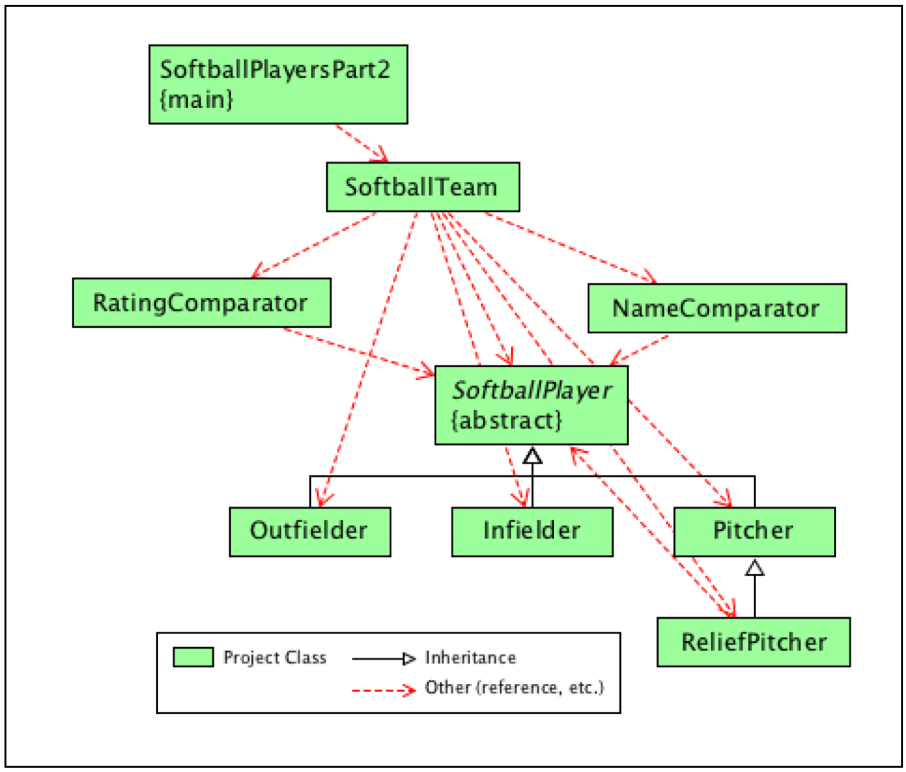
In the second test method, you can invoke main as follows with the file name as the first (and only) command line argument:

```
String[] args2 = {"softball_player_data.csv"};
// element args2[0] is the file name
SoftballPlayersPart2.main(args2);
Assert.assertEquals("Team count should be 1. ",
                    1, SoftballTeam.getTeamCount());
```

If Web-CAT complains that the default constructor for `SoftballPlayersPart2` has not been covered, you should include the following line of code in one of your test methods.

```
// to exercise the default constructor
SoftballPlayersPart2 app = new SoftballPlayersPart2();
```

UML Class Diagram



Notes:

1. Player data – Since player numbers are of type String, single digit numbers should have a leading space which should not be trimmed. For example, see player number 9 in the *softball_player_data.csv* file. This this is important if a String compare is used to determine order.
2. You should test your program using both data files provided (*softball_player_data.csv* and *softball_player_data2.csv*).

Example Output when file name is missing as command line argument

```
 ----jGRASP exec: java SoftballPlayersPart2
File name expected as command line argument.
Program ending.

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

Example Output for *softball_player_data.csv*

```
 ----jGRASP exec: java SoftballPlayersPart2 softball_player_data.csv

----------------------------------------
Team Report for Auburn Stars
----------------------------------------

32 Pat Jones (RF) .375
Specialization Factor: 1.0 (class Outfielder) Rating: 3.562

23 Jackie Smith (3B) .275
Specialization Factor: 1.25 (class Infielder) Rating: 2.922

43 Jo Williams (RHP) 22 wins, 4 losses, 2.85 ERA
Specialization Factor: 2.0 (class Pitcher) Rating: 3.740

34 Sammi James (LHP) 5 wins, 4 losses, 17 saves, 3.85 ERA
Specialization Factor: 2.0 (class ReliefPitcher) Rating: 2.474

 9 Pat Williams (RF) .480
Specialization Factor: 1.0 (class Outfielder) Rating: 4.560

----------------------------------------
Team Report for Auburn Stars (by Number)
----------------------------------------
 9 Pat Williams RF .480
```

```
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 Stars (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 .480

---------------------------------------
Team Report for Auburn Stars (by Rating)
---------------------------------------
4.56  9 Pat Williams RF .480
3.74 43 Jo Williams RHP 22 wins, 4 losses, 2.85 ERA
3.56 32 Pat Jones RF .375
2.92 23 Jackie Smith 3B .275
2.47 34 Sammi James LHP 5 wins, 4 losses, 17 saves, 3.85 ERA

---------------------------------------
Excluded Records Report
---------------------------------------
*** invalid category *** L,23,Gayle Adams,2B,1.25,.225,.875

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

Example Output for *softball_player_data2.csv*

```
 ----jGRASP exec: java SoftballPlayersPart2 softball_player_data2.csv

---------------------------------------
Team Report for My Bigger Team
---------------------------------------

21 Jodi Doe (RF) .305
Specialization Factor: 1.0 (class Outfielder) Rating: 2.989

11 Tina Dobbs (RF) .350
Specialization Factor: 1.0 (class Outfielder) Rating: 3.395

13 Nina Dobbs (LF) .478
Specialization Factor: 1.0 (class Outfielder) Rating: 4.541

12 Poppi Ledet (LF) .325
```

```
Specialization Factor: 1.0 (class Outfielder) Rating: 3.120

14 Sruthi Yalamanchili (CF) .285
Specialization Factor: 1.0 (class Outfielder) Rating: 2.679

15 Kavya Krishnappa (CF) .298
Specialization Factor: 1.0 (class Outfielder) Rating: 2.771

29 Sandy Chintapalli (1B) .265
Specialization Factor: 1.25 (class Infielder) Rating: 2.915

17 Janie Doe (1B) .350
Specialization Factor: 1.25 (class Infielder) Rating: 4.244

18 Buddy Bell (2B) .325
Specialization Factor: 1.25 (class Infielder) Rating: 3.494

19 Gigi de la Hoya (2B) .278
Specialization Factor: 1.25 (class Infielder) Rating: 3.301

20 Daisy Doalot (3B) .285
Specialization Factor: 1.25 (class Infielder) Rating: 3.349

10 Mikie Mahtook (3B) .298
Specialization Factor: 1.25 (class Infielder) Rating: 3.464

22 Matty Ott (SS) .298
Specialization Factor: 1.25 (class Infielder) Rating: 3.278

23 Leah Coleman (SS) .350
Specialization Factor: 1.25 (class Infielder) Rating: 4.244

25 Erin Noland (RHP) 5 wins, 11 losses, 4.3 ERA
Specialization Factor: 2.0 (class Pitcher) Rating: -.906

26 Jackie Malkovic (RHP) 6 wins, 10 losses, 5.4 ERA
Specialization Factor: 2.0 (class Pitcher) Rating: -.500

27 Lois Gibson (RHP) 8 wins, 7 losses, 3.5 ERA
Specialization Factor: 2.0 (class Pitcher) Rating: .178

28 Gina Malika (LHP) 7 wins, 8 losses, 1.6 ERA
Specialization Factor: 2.0 (class Pitcher) Rating: -.308

16 Tika Brando (LHP) 9 wins, 7 losses, 1.7 ERA
Specialization Factor: 2.0 (class Pitcher) Rating: .593

30 Belinda Striker (LHP) 10 wins, 6 losses, 10 saves, 1.8 ERA
Specialization Factor: 2.0 (class ReliefPitcher) Rating: 3.333

31 Lilly Dean (RHP) 11 wins, 5 losses, 3 saves, 1.9 ERA
Specialization Factor: 2.0 (class ReliefPitcher) Rating: 2.069

32 Briana Wilson (RHP) 4 wins, 4 losses, 14 saves, 2.0 ERA
Specialization Factor: 2.0 (class ReliefPitcher) Rating: 3.111

33 Janine Mason (RHP) 5 wins, 3 losses, 12 saves, 2.1 ERA
```

```
Specialization Factor: 2.0 (class ReliefPitcher) Rating: 3.011

34 Green Lantern (LHP) 14 wins, 2 losses, 3 saves, 2.2 ERA
Specialization Factor: 2.0 (class ReliefPitcher) Rating: 3.125

---------------------------------------
Team Report for My Bigger Team (by Number)
---------------------------------------
10 Mikie Mahtook 3B .298
11 Tina Dobbs RF .350
12 Poppi Ledet LF .325
13 Nina Dobbs LF .478
14 Sruthi Yalamanchili CF .285
15 Kavya Krishnappa CF .298
16 Tika Brando LHP 9 wins, 7 losses, 1.7 ERA
17 Janie Doe 1B .350
18 Buddy Bell 2B .325
19 Gigi de la Hoya 2B .278
20 Daisy Doalot 3B .285
21 Jodi Doe RF .305
22 Matty Ott SS .298
23 Leah Coleman SS .350
25 Erin Noland RHP 5 wins, 11 losses, 4.3 ERA
26 Jackie Malkovic RHP 6 wins, 10 losses, 5.4 ERA
27 Lois Gibson RHP 8 wins, 7 losses, 3.5 ERA
28 Gina Malika LHP 7 wins, 8 losses, 1.6 ERA
29 Sandy Chintapalli 1B .265
30 Belinda Striker LHP 10 wins, 6 losses, 10 saves, 1.8 ERA
31 Lilly Dean RHP 11 wins, 5 losses, 3 saves, 1.9 ERA
32 Briana Wilson RHP 4 wins, 4 losses, 14 saves, 2.0 ERA
33 Janine Mason RHP 5 wins, 3 losses, 12 saves, 2.1 ERA
34 Green Lantern LHP 14 wins, 2 losses, 3 saves, 2.2 ERA

---------------------------------------
Team Report for My Bigger Team (by Name)
---------------------------------------
18 Buddy Bell 2B .325
16 Tika Brando LHP 9 wins, 7 losses, 1.7 ERA
29 Sandy Chintapalli 1B .265
23 Leah Coleman SS .350
19 Gigi de la Hoya 2B .278
31 Lilly Dean RHP 11 wins, 5 losses, 3 saves, 1.9 ERA
20 Daisy Doalot 3B .285
13 Nina Dobbs LF .478
11 Tina Dobbs RF .350
17 Janie Doe 1B .350
21 Jodi Doe RF .305
27 Lois Gibson RHP 8 wins, 7 losses, 3.5 ERA
15 Kavya Krishnappa CF .298
34 Green Lantern LHP 14 wins, 2 losses, 3 saves, 2.2 ERA
12 Poppi Ledet LF .325
10 Mikie Mahtook 3B .298
28 Gina Malika LHP 7 wins, 8 losses, 1.6 ERA
26 Jackie Malkovic RHP 6 wins, 10 losses, 5.4 ERA
33 Janine Mason RHP 5 wins, 3 losses, 12 saves, 2.1 ERA
25 Erin Noland RHP 5 wins, 11 losses, 4.3 ERA
22 Matty Ott SS .298
```

```
30 Belinda Striker LHP 10 wins, 6 losses, 10 saves, 1.8 ERA
32 Briana Wilson RHP 4 wins, 4 losses, 14 saves, 2.0 ERA
14 Sruthi Yalamanchili CF .285

---------------------------------------
Team Report for My Bigger Team (by Rating)
---------------------------------------
4.54 13 Nina Dobbs LF .478
4.24 17 Janie Doe 1B .350
4.24 23 Leah Coleman SS .350
3.49 18 Buddy Bell 2B .325
3.46 10 Mikie Mahtook 3B .298
3.39 11 Tina Dobbs RF .350
3.35 20 Daisy Doalot 3B .285
3.33 30 Belinda Striker LHP 10 wins, 6 losses, 10 saves, 1.8 ERA
3.30 19 Gigi de la Hoya 2B .278
3.28 22 Matty Ott SS .298
3.12 12 Poppi Ledet LF .325
3.12 34 Green Lantern LHP 14 wins, 2 losses, 3 saves, 2.2 ERA
3.11 32 Briana Wilson RHP 4 wins, 4 losses, 14 saves, 2.0 ERA
3.01 33 Janine Mason RHP 5 wins, 3 losses, 12 saves, 2.1 ERA
2.99 21 Jodi Doe RF .305
2.92 29 Sandy Chintapalli 1B .265
2.77 15 Kavya Krishnappa CF .298
2.68 14 Sruthi Yalamanchili CF .285
2.07 31 Lilly Dean RHP 11 wins, 5 losses, 3 saves, 1.9 ERA
0.59 16 Tika Brando LHP 9 wins, 7 losses, 1.7 ERA
0.18 27 Lois Gibson RHP 8 wins, 7 losses, 3.5 ERA
-0.31 28 Gina Malika LHP 7 wins, 8 losses, 1.6 ERA
-0.50 26 Jackie Malkovic RHP 6 wins, 10 losses, 5.4 ERA
-0.91 25 Erin Noland RHP 5 wins, 11 losses, 4.3 ERA

---------------------------------------
Excluded Records Report
---------------------------------------
*** invalid category *** H,24,Nola Austin,LHP,2.0,0.225,4,12,1.2
Maximum player count of 24 exceeded for: R,35,Bruce Wayne,LHP,2.0,0.175,15,1,2.3,4
Maximum player count of 24 exceeded for: R,36,Billie Gates,LHP,2.0,0.325,16,0,2.4,2

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