CS 143 Week 3 Programming Lab Specifications

Checkpoint Due 11:59PM Friday July 20
Lab Due 11:59PM April Monday July 23

Overview: Pretend that Skagit Valley College holds a Red-N-White Day competition each spring quarter for all students and staff. There are two events: a cow-chip toss and a one-legged race. The results of the 2016 contest are contained in a file named `contest2016.dat`. You will write a program in a file named `ContestLab.java`. The outer `ContestLab` class will contain a nested inner `ContestantRecord` class defined as follows:

```
private static class ContestantRecord {
   private String contestant;   //name of contestant
   private double distance;     //toss distance in feet
   private double time;         //race time in seconds

  //methods as needed to satisfy program specifications will go here
}
```

In other words, the `ContestantRecord` class is effectively a static field of your `ContestLab` class. Recall that `static` items describe an entity that belongs to the entire class, whereas non-static items belong to each *instance* (object) of the class. In this case, there is a single static "blueprint" for `ContestantRecord` objects inside the `ContestLab` program—*but*, there may be multiple `ContestantRecord` *objects* created from that single blueprint. The benefit of nesting an inner class within an outer class is the convenience of having fewer source-code files to manage. Also, since the `ContestantRecord` class is a field of the `ContestLab` class, the `ContestLab` class has direct access to all the private data in the inner class, without the need for "get" and "set" methods.

The `ContestLab` program will read the data from the `contest2016.dat` file into a *parameterized* Java `ArrayList<ContestantRecord>` object. The program then uses the various JCF sort, search and shuffle methods associated with the `ArrayList` class to produce various reports. There's a catch, however: The JCF methods "know" how to search and sort built-in `Integer`, `Double`, `String`, etc. data. They don't "know" how to search and sort `ContestantRecord` objects. The `ContestLab` program will tell them how, by defining an appropriate `compareTo()` method, `equals()` method, and *comparator* classes.

Style Note: Your methods must remain within the 25-line limit set in the *Good Style Specifications* document posted in the *Quick Links Module* on the course website. No other `ContestLab` class fields may be declared except the `ContestantRecord` class and a `Scanner` object connected to `System.in`. This requires you to use appropriate parameter passing when defining and calling methods in your program.

Stage 1 Lab 3 Checkpoint Specifications: At this stage of Lab 3 you will input the data from the `contest2016.dat` file into an `ArrayList<ContestantRecord>` object. You will confirm that it works properly by displaying the contents of the resulting `ArrayList` object.

- The `contest2016.dat` file should be stored in the same folder as the `ContestLab` program.
- Inputting data from a file using the `Scanner` class was presented in BJP Chapter 6.

- You should have `throws IOException` clauses in any method that directly or indirectly processes the file. The `IOException` class is a base/super class for the `FileNotFoundException` sub/derived class, so the `IOException` is more general, and easier to type, than the `FileNotFoundException` that was introduced in BJP Chapter 6.

The data for each contestant is contained on a pair of lines in the `contest2016.dat` file as follows:
- The first line of each pair contains the contestant's name (last name followed by initial of first name.)
- The second line of each pair contains the feet of toss distance and the seconds of race time.

Here are a few of the line pairs in the file:
```
ALI Z
20.9  66.3
CHEN R
22.1  52.9
. . .
FREITAS B
22.4  57
LONG T
19    55
```

Your Stage-1 `ContestLab` checkpoint code should implement the following:

- Declare an inner `ContestantRecord` class within your outer `ContestLab` class. The `Lab3Files` folder posted with this assignment contains a file named `RectanglePlay.java` that has an example of nesting an inner class within an outer class.
- Define a `toString()` method for the `ContestantRecord` class that returns the concatenation of the contestant's name, with a space, with their toss distance, with a space, with their race time, followed by a "\n" character. The `ArrayList.toString()` method will then be able to use your `ContestantRecord.toString()` method to display each element in the list. REMINDER: The `toString()` method was introduced in BJP 8.2.
- Declare, initialize, fill and display an `ArrayList` of `ContestantRecord` objects using logic like that in the Java code segment below Table 10.1 in BJP 10.1. That logic must be tweaked as follows:
    - You will need to construct a `new ContestantRecord` object using the default constructor provided by the Java compiler before reading the next 2 lines of data from the file and placing their values into the corresponding fields of the `ContestantRecord` object that you just constructed.
    - REMINDER from BJP Chapter 6: Depending on how you read each pair of lines from the file, you may need to consume the '\n' that followed race time in the file stream before performing another `nextLine()` operation.

When your program runs at this stage it should generate the following console screen output:
```
[ALI Z 20.9 66.3
, CHEN R 22.1 52.9
, GARZA H 22.3 66.0
, HO V 19.3 69.5
, IVERSON R 24.2 51.7
, MAIDEN G 24.3 68.9
, NGUYEN N 20.3 51.2
, OLSON B 20.2 54.6
, SHIN M 23.4 57.5
, WALLIN C 18.7 69.8
, WHITEMARSH A 22.1 64.3
, ZIEGLER N 19.9 63.0
, CONLEY D 22.3 53.2
, DONOHUE R 23.7 50.5
, FELICIANO A 16.3 68.3
, FREITAS B 22.4 57.0
, LONG T 19.0 55.0
]
```

End Stage 1 Lab 3 Checkpoint

Stage 2 Specifications: Write a `static void` `displayReport()` method for the `ContestLab` class that accepts a `String` and an `ArrayList<ContestantRecord>` parameter when it is called. The `String` parameter is the title of the report. Add code to the `ContestLab` program that calls the `displayReport()` method appropriately so as to produce the following sample I/O session:

```
[ALI Z 20.9 66.3
, CHEN R 22.1 52.9
, GARZA H 22.3 66.0
. . .
, FREITAS B 22.4 57.0
, LONG T 19.0 55.0
]
                         Original Data
       ==================================================
          Contestants                      Ft       Secs
       ==================================================
          ALI Z                         20.90     66.30
          CHEN R                        22.10     52.90
          GARZA H                       22.30     66.00
          HO V                          19.30     69.50
          IVERSON R                     24.20     51.70
          MAIDEN G                      24.30     68.90
          NGUYEN N                      20.30     51.20
          OLSON B                       20.20     54.60
          SHIN M                        23.40     57.50
          WALLIN C                      18.70     69.80
          WHITEMARSH A                  22.10     64.30
          ZIEGLER N                     19.90     63.00
          CONLEY D                      22.30     53.20
          DONOHUE R                     23.70     50.50
          FELICIANO A                   16.30     68.30
          FREITAS B                     22.40     57.00
          LONG T                        19.00     55.00
```

The `displayReport()` method produces a report such that

- The report is 50 characters wide and indented as shown above
- The method centers the title it was passed over the report
- The `ArrayList` data is displayed in a report formatted as shown above. HINT: You will need to use `printf()` to control the formatting of the report, and the `get()` method of the `ArrayList` class inside a `for`-loop. You may use either a counting `for`-loop or the `for-each` loop described in the *Using the `For-Each` Loop with `ArrayLists`* section of BJP 10.1.

Stage 3 Specifications: Call the `shuffle()` method of the `Collections` class described in Table 11.3 of BJP 11.1 to randomly rearrange the `ContestantRecord` objects in the `ArrayList`. Then call the `displayReport()` method with the title “Shuffled Data”. A sample I/O session is shown below.

```
[ALI Z 20.9 66.3
. . .
]
                         Original Data
       ==================================================
          Contestants                     Ft       Secs
       ==================================================
          ALI Z                         20.90     66.30
         . . .
          LONG T                        19.00     55.00

                         Shuffled Data
       ==================================================
          Contestants                     Ft       Secs
       ==================================================
          OLSON B                       20.20     54.60
          FELICIANO A                   16.30     68.30
          ALI Z                         20.90     66.30
          IVERSON R                     24.20     51.70
          GARZA H                       22.30     66.00
          WHITEMARSH A                  22.10     64.30
          CONLEY D                      22.30     53.20
          SHIN M                        23.40     57.50
          LONG T                        19.00     55.00
          MAIDEN G                      24.30     68.90
          WALLIN C                      18.70     69.80
          CHEN R                        22.10     52.90
          FREITAS B                     22.40     57.00
          ZIEGLER N                     19.90     63.00
          NGUYEN N                      20.30     51.20
          DONOHUE R                     23.70     50.50
          HO V                          19.30     69.50
```

NOTE: Due to the random nature of the shuffle algorithm, the order of the data in the *Shuffled Data* report will vary from execution to execution.

Sorting Overview: We are going to want to be able to produce the following reports:

- A report that is sorted alphabetically by contestant name
- A report that is sorted from best toss distance to worst; that is, furthest distance to shortest.
- A report that is sorted from the best race time to worst; that is, from quickest time to slowest.

The `Collections` class has two different static `sort()` methods:

- `Collections.sort(list)` as presented in Chapter 10 and the beginning of Chapter 13

  This single-parameter `sort()` method calls the `compareTo()` method of the class of object that you are sorting. We will implement the `Comparable` interface for the `ContestantRecord` class to allow us to use this `sort()` method to sort the `ArrayList` object alphabetically by contestant name.

- `Collections.sort(list, comparator)` as discussed in BJP 13.1

  This two-parameter `sort()` method requires that you define a class that implements the `Comparator` interface by implementing the `compare()` method for the class of objects that are to be sorted. We can have as many of these classes as desired to order our data in as many ways as we wish. We will create two classes that implement the `Comparator` interface. One will enable sorting by race times, the other will support sorting by toss distances.

Stage 4 Specifications: Modify the `ContestantRecord` class so that it *implements* the `Comparable` interface, as discussed in BJP 10.2. This means that you must define the `compareTo()` method for the `ContestantRecord` class. Doing so correctly will ensure that the one-parameter `Collections.sort()` method can sort `ContestantRecord` objects alphabetically by the contestant's name. Keep the following in mind:

- If `c1` and `c2` are `ContestantRecord` objects.
  `c1.compareTo(c2)` returns a negative value if c1 < c2 (that is c1 is alphabetically less than c2)
  `c1.compareTo(c2)` returns 0 if c1 == c2 (that is c1 and c2 are the same name)
  `c1.compareTo(c2)` returns a positive value if c1 > c2 (that is c1 is alphabetically greater than c2)
  It is the `Collections.sort()` method that will be calling your `ContestantRecord`'s `compareTo()` method as it tries to sort your contestant objects alphabetically.
  REMINDER: You can't use <, ==, or > operators on `String` objects, so see the next bullet. . .

- ***Don't reinvent the wheel.*** The `String` class has already defined a `compareTo()` method. Your `compareTo()` method should invoke the `String` class's `compareTo()` method appropriately. If you do this efficiently, the `compareTo()` method of your `ContestantRecord` class will contain only a single line of code.

- Assume that the names can be any combination of all uppercase letters, all lowercase letters, or some mix of both. "Jo", "JO" and "jo" should all be treated like they are the same name. This requires that you compare the `toLowerCase()` (or `toUpperCase()`) values of the two `ContestantRecord` object names. These `String` methods were introduced in Table 3.3 in BJP 3.3. ***Do not change the values of the names.***

Once you have implemented the `Comparable` interface for the `ContestantRecord` class, call the one-parameter `Collections.sort()` method and display a "Contest Results Sorted by Name" report to see if it worked. Your program should now be able to produce an I/O session such as the following:

```
[ALI Z 20.9 66.3
. . .
]
                        Original Data
         . . .
          LONG T                        19.00     55.00

                        Shuffled Data
         . . .
          MAIDEN G                      24.30     68.90

                Contest Results Sorted by Name
       ==================================================
          Contestants                     Ft       Secs
       ==================================================
          ALI Z                         20.90     66.30
          CHEN R                        22.10     52.90
          CONLEY D                      22.30     53.20
          DONOHUE R                     23.70     50.50
          FELICIANO A                   16.30     68.30
          FREITAS B                     22.40     57.00
          GARZA H                       22.30     66.00
          HO V                          19.30     69.50
          IVERSON R                     24.20     51.70
          LONG T                        19.00     55.00
          MAIDEN G                      24.30     68.90
          NGUYEN N                      20.30     51.20
          OLSON B                       20.20     54.60
          SHIN M                        23.40     57.50
          WALLIN C                      18.70     69.80
          WHITEMARSH A                  22.10     64.30
          ZIEGLER N                     19.90     63.00
```

Stage 5 Specifications: Create a nested inner `public static` `RaceComparator` class, such as those presented in the *Custom Ordering with Comparators* section of BJP 13.1, that will be used by the two-parameter `Collections.sort()` method to order `ContestantRecord` objects by race time. Keep the following in mind:

- Define the `compare()` method for the `RaceComparator` class as follows:
  Assume `c1` and `c2` are `ContestantRecord` objects.
  `compare(c1,c2)` returns negative value if c1 < c2
  (that is c1's time is faster and should precede c2 in ordering)
  `compare(c1,c2)` returns 0 if c1 == c2 (that is c1 and c2 have the same time score)
  `compareTo(c1,c2)` returns positive value if c1 > c2 (that is c1's time is worse than c2's time)

Once you have implemented the `RaceComparator` class, call the 2-parameter `Collections.sort()` method and display a "Race Results" report. Your program should now be able to produce an I/O session such as the following:

```
[ALI Z 20.9 66.3
. . .
, LONG T 19.0 55.0
]
                         Original Data
         . . .
          LONG T                        19.00     55.00

                         Shuffled Data
         . . .
          MAIDEN G                      24.30     68.90

                 Contest Results Sorted by Name
         . . .
          ZIEGLER N                     19.90     63.00

                          Race Results
       ==================================================
          Contestants                     Ft       Secs
       ==================================================
          DONOHUE R                     23.70     50.50
          NGUYEN N                      20.30     51.20
          IVERSON R                     24.20     51.70
          CHEN R                        22.10     52.90
          CONLEY D                      22.30     53.20
          OLSON B                      20.20     54.60
          LONG T                       19.00     55.00
          FREITAS B                    22.40     57.00
          SHIN M                       23.40     57.50
          ZIEGLER N                    19.90     63.00
          WHITEMARSH A                 22.10     64.30
          GARZA H                      22.30     66.00
          ALI Z                        20.90     66.30
          FELICIANO A                  16.30     68.30
          MAIDEN G                     24.30     68.90
          HO V                         19.30     69.50
          WALLIN C                     18.70     69.80
```

Stage 6 Specifications: Create a nested inner `public static TossComparator` class that will be used by the two-parameter `Collections.sort()` method to order `ContestantRecord` objects by toss distance. Then produce a "Results by Toss Distance" report. Your program should now be able to produce an I/O session such as the following:

```
[ALI Z 20.9 66.3
. . .
, LONG T 19.0 55.0
]
                         Original Data
         . . .
          LONG T                       19.00     55.00

                         Shuffled Data
          . . .
          FREITAS B                    22.40     57.00

                 Contest Results Sorted by Name
         . . .
          ZIEGLER N                    19.90     63.00

                          Race Results
          WALLIN C                     18.70     69.80

                    Results by Toss Distance
       ==================================================
          Contestants                    Ft       Secs
       ==================================================
          MAIDEN G                     24.30     68.90
          IVERSON R                    24.20     51.70
          DONOHUE R                    23.70     50.50
          SHIN M                       23.40     57.50
          FREITAS B                    22.40     57.00
          CONLEY D                     22.30     53.20
          GARZA H                      22.30     66.00
          CHEN R                       22.10     52.90
          WHITEMARSH A                 22.10     64.30
          ALI Z                        20.90     66.30
          NGUYEN N                     20.30     51.20
          OLSON B                      20.20     54.60
          ZIEGLER N                    19.90     63.00
          HO V                         19.30     69.50
          LONG T                       19.00     55.00
          WALLIN C                     18.70     69.80
          FELICIANO A                  16.30     68.30
```

Searching Overview: We search lists for target values. In this lab, we search the `ArrayList` for a target name. We will want to know if the target name is in the list, and if so, where in the list it is—that is, what is its index?

The `indexOf()` method of the `ArrayList` class described in Table 10.2 of BJP 10.1 uses a linear search to perform this operation. It uses the `equals()` method discussed in Chapter 9 to perform the search, *not* the `compare()` or `compareTo()` methods discussed earlier.

If the list is sorted on the field of the target item we are looking for, we can speed things up by using a `Collections.binarySearch()` method described in Table 13.2 in BJP 13.1. The two-parameter method uses the `compareTo()` method of the class of the target item it is searching for.

The search methods will want `ContestantRecord` target objects to search for. We'll be searching for names—which are `String` objects. We will therefore have to create `ContestantRecord` target objects to hold the target name we are looking for in a manner such as the following:

```
ContestantRecord targetRec;
do {
   System.out.print("Enter target name to search for (or QUIT): ");
   targetRec = new ContestantRecord();
   targetRec.contestant = console.nextLine();
//call desired search method to seek targetRec
}while(!targetRec.contestant.equalsIgnoreCase("QUIT"));
```

Stage 7 Specifications: Support the `ArrayList indexOf()` linear search method by defining the `equals()` method for the `ContestantRecord` class appropriately. We will consider two `ContestantRecord` objects to be equal it the contestant's names are the same. In other words, we are treating the names like key fields that uniquely distinguish one contestant from another. Keep the following issues in mind:

- The parameter should be of type `Object` as presented in the *The `equals` Method* section of BJP 9.2.
- If the parameter isn't a `ContestantRecord` object, return `false`.
- Don't reinvent the wheel: If the parameter *is* a `ContestantRecord` object, return the result of calling the `String` class's `equalsIgnoreCase()` method.

After you have implemented the `equals()` method of the `ContestantRecord` class, call the `indexOf()` method of the `ArrayList` class to see if everything is working properly. Your program should now be able to produce an I/O session such as the following:

```
ALI Z 20.9 66.3
. . .
, LONG T 19.0 55.0
]
                         Original Data
         . . .
          LONG T                      19.00     55.00

                         Shuffled Data
         . . .
          ALI Z                       20.90     66.30

                 Contest Results Sorted by Name
          ALI Z                         20.90     66.30
         . . .
          ZIEGLER N                     19.90     63.00

                         Race Results
         . . .
          WALLIN C                      18.70     69.80

                     Results by Toss Distance
         . . .
          FELICIANO A                   16.30     68.30

Enter target name to search for (or QUIT): shin m
indexOf() shows SHIN M had toss of 23.4 ft and race time of 57.5 secs.
Enter target name to search for (or QUIT): ALI Z
indexOf() shows ALI Z had toss of 20.9 ft and race time of 66.3 secs.
Enter target name to search for (or QUIT): xyz
indexOf() says xyz NOT in list.
Enter target name to search for (or QUIT): quit
indexOf() says quit NOT in list.
```

Stage 8 Specifications: Sort the `ArrayList` by contestant name and then call the two-parameter `binarySearch()` method of the `Collections` class shown in Table 11.3 of section 11.1 or Table 13.1 in section 13.1. Your program should now be able to produce an I/O session such as the following:

```
[ALI Z 20.9 66.3
, CHEN R 22.1 52.9
. . .
, FREITAS B 22.4 57.0
, LONG T 19.0 55.0
]

                         Original Data
       ==================================================
          Contestants                     Ft      Secs
       ==================================================
          ALI Z                         20.90     66.30
      . . .
          LONG T                        19.00     55.00

                         Shuffled Data
       ==================================================
          Contestants                     Ft      Secs
       ==================================================
          IVERSON R                     24.20     51.70
       . . .
          CHEN R                        22.10     52.90

                 Contest Results Sorted by Name
       ==================================================
          Contestants                     Ft      Secs
       ==================================================
          ALI Z                         20.90     66.30
       . . .
          ZIEGLER N                     19.90     63.00

                          Race Results
```

```
        ==================================================
           Contestants                  Ft      Secs
        ==================================================
           DONOHUE R                  23.70     50.50
       . . .
           WALLIN C                   18.70     69.80

                     Results by Toss Distance
        ==================================================
           Contestants                  Ft      Secs
        ==================================================
           MAIDEN G                   24.30     68.90
        . . .
           FELICIANO A                16.30     68.30

Enter target name to search for (or QUIT): ho v
indexOf() shows HO V had toss of 19.3 ft and race time of 69.5 secs.
binarySearch() method shows HO V had toss of 19.3 ft and race time of 69.5 secs.
Enter target name to search for (or QUIT): ALI Z
indexOf() shows ALI Z had toss of 20.9 ft and race time of 66.3 secs.
binarySearch() method shows ALI Z had toss of 20.9 ft and race time of 66.3 secs.
Enter target name to search for (or QUIT): quit
indexOf() says quit NOT in list.
binarySearch() method says quit NOT in list.
```

Style Requirements: Be sure to compare your code against the *Good Style Requirements* document posted in the Quick Links Module on the Canvas website for the course.