OoL #3: Algorithm Refinement

For this Out of Lab, we'll be focusing on algorithms and alternate ways of doing things. I'll be posting a possible OoL #2 solution to Moodle on Monday for you to examine. You may use that possible solution OR your own OoL #2 solution as a starting point for OoL #3. The specification for OoL #3 will be broken into 7 parts: Update, Recursion, Sorting, Analysis, Simple AI, Analysis II.

YOU MAY WORK IN GROUP UP TO SIZE 4: Each group member MUST be the PRIMARY author (and note in comments **AND** TLDDR) for AT LEAST 1 of these code sections:

-- Recursion Section

-- Manual Sort subsection

-- Comparator Implementation subsection

-- Major Event Update subsection

UPDATE SECTION:

- Major Event Update:
  - In the Olympics, entry in some Events are based on performance in previous Events (Running Heats → Semi-Finals → Finals …. Or Gymnastics Single Events are based on scores during the team competition to give a few examples). You may assume that for each Event which gets Athletes from previous events will always pick the top 5 from that event (excluding cheaters but including "fainters" if there aren't enough "non-fainters"). Modify your code to allow for this behavior.
  - **TLDDR:** As there are several ways to do this make sure you explain and justify why your group chose to implement this requirement the way you did.
  - Event input file will now have an additional bit of information and will look like the below line:
    - *Name,popularity,venueType,list_of_Event_names_which_this_event_uses_to_draw_Atheletes_from*
  - Events will draw Athletes from 0 – 10 other events
- Minor Event Update:
  - Needs a compareTo() which uses popularity of event
- Fan Update:
  - Needs a compareTo() which uses happiness
  - Happiness is calculated by:
    - Happiness = (Events_Attended * 10) + (money_in_wallet) – (Events_attend_refunded * 20) – (Events_Wanted_to_attend_but_venue_was_full * 40)
  - You may want/need to add attributes to fan to help with this

- Olympic Summary (the output after all events are completed) Update:
  - Print “Average Fan Happiness”
  - Add “Average Fan Happiness” to the success metrics. Include in your TLDDR what your cut off for “happy fans” is.
  - Add a print of the top 10 happiest Fans and top 10 unhappiest fans
  - Add a print of the top 10 highest endorsed Athletes
  - Add a print of the top 10 most attended events
  - Add a print of the top 10 least attended events

RECURSION SECTION: You must include a driver method which uses **recursion** to build a List of Lists (can uses arrays, ArrayLists, or any other 1D list structure in java). This List of Lists should be configured so that there is a list of Athlete Lists, where the sub-lists are:

- Athletes who cheated
- Athletes who didn’t cheat
- Athletes with gold medals
- Athletes with silver medals
- Athletes with bronze medals
- Athletes with at least 1 medal
- Athletes with no medals
- Athletes who prefer Aquatic
- Athletes who prefer Track
- Athletes who prefer Gym
- Athletes who prefer Outdoor

This method should be called prior to the printing of the “End of Olympics” summary data.

SORTING SECTION: 3 sorts will be used in OoL #3

- Comparable Implementation:
  - Prior to the start of the Olympics in the Driver, print out the lists of Events, Venues and Athletes sorted by their compareTo() method. (Basically, make the object implement Comparable and then use Arrays.sort()).
  - After all events are completed, sort your list of all Fans using Comparable and Arrays.sort() to help with the Top 10 happiest and Top 10 unhappiest fan info in the Olympic Summary.
- Comparator Implementation:
  - In Athlete, create Comparators for:
    - Sorting by number of gold medals.
    - Sorting by number of silver medals.
    - Sorting by number of bronze medals.
    - Total metal count
  - For all of these Comparators, break ties by (in order):
    - Skill
    - Total Endorsements

        - Total Medals (except where that WAS the sort)
        - ID number
    - Use these Comparators to sort the sub-lists of: Athletes with gold medals, Athletes with silver medals, Athletes with bronze medals, and Athletes with at least 1 medal generated by the recursive method.
    - After sorting, use those same lists to print the 4 "Top 10" lists required by the Olympic Summary
- Manual Sort:
    - Create method which sorts Athletes by endorsements. You must manually implement one of the 5 sorts we talked about in class (Quick, Bogo, Bubble, Insert, or Selection) to do this. Use this method to help with determine the Top 10 most endorsed Athletes for the Olympic Summary.
    - Create a method which sorts Events by attendance. You must manually implement one of the 5 sorts we talked about in class (Quick, Bogo, Bubble, Insert, or Selection) to do this. **YOU MAY NOT USE THE SAME SORT USED FOR ATHLETE ENDORSEMENTS!** Use this method to help with determine the Top 10 lists for most/least attended Events in the Olympic Summary.

ANALYSIS SECTION: Run your code enough times with the same input files to a get picture for how things play out. Think about and answer the following questions in your TLDDR. Make sure to include your thoughts on WHY those answers are what they are. (Include your number of runs and final result data which led you to make your claims as text files in your jar):

- Do the same Athletes tend to be the most/least successful?
- Does skill seem dominate the results? Or do the Random Number Generator, Cheating, Stamina or other factors play a bigger role?
- Do certain Events seem to be canceled more often?
- Do the Events attendance seem to be consistent between different runs of your program?
- Based on the "success" metrics, do the same levels of success for each metric tend to get produced?

SIMPLE AI SECTION: Based on what you learned in the Analysis Section (and any other thoughts you might have), attempt to add new code to your program to try and meet each of the below goals. You may wish to target 1 goal at a time to decrease the number of variables being manipulated at once.

- Athlete fainting is minimized via Athletes being better at determining if they are too tired to enter an event.
- Adjust score calculation to reflect better what factors you think should dictate Athlete success. (Make sure to explain your reasoning in your TLDDR!)
- Use dynamic Event pricing (and update Fan decision logic to account for this) to change fan behavior to:
    - Increase attendance at "unpopular" events
    - Decrease number of fans who aren't able to buy tickets to sold-out events to which they wanted to attend.

ANALYSIS II SECTION: Run your code enough times to get a feel for how your changes helped/hindered the goalthe current rules work and answer the below questions (include your number of runs and final result data which led you to make your claims as text files in your jar). NOTE, you MUST repeat this section and the Simple AI section at least 2 times. You may repeat as many additional times as you want. Just keep adding sections in your TLDDR as you go. I'm not expecting you to get to "perfect" data (whatever that means), I'm just interested in your logic, discussion and implementation towards the specific goals. In other words, I'm focusing more on your thought process and ability to write code which is CONSISTENT with your thought process. As you go, include in your TLDDR:

- What code changes you tried
- Why you tried those code changes
- What happened (outcome of the changes)
- What each outcome indicated

Grading:

• Analysis Section	10 points
• Simple AI Section and Analysis II Section	25 points*
• Comparable Implementation	10 points
• Minor Event Update	5 points
• Olympic Summary Update	10 points
• Fan Update	10 points
• JavaDoc	5 points**
• Your Primary Author Section	25 points

*With quality repetition and discussion of Simple AI and Analysis II Sections above and beyond my required expectations, you could earn up to 10 points extra credit.

**The 5 points are for JavaDocing the NEW methods. If you still have non-JavaDoc'd methods/classes from existing code, I may deduct up to 10 points for that.