COSC 2P03 – Assignment 1

This assignment deals with data structures. Specifically, you'll be providing an implementation for a defined ADT. You'll also be reacquainting yourself with generics and packages.

Pool Queue:
The Macguffin games have been very popular lately. People from far and wide have been drawn to the games; some to play with friends, and some to simply play. As you, of course, know, the Macguffin games can be played individually, as a pair, as a trio, or even as a group of four.
Unfortunately, as you also know, the waiting lines can often get quite large for the games. What's more, considering the sheer number of players, and the variety of group sizes, it can be problematic to fairly accommodate people in an appropriate order.

Thus, the concept of a "Pool Queue" was created, to accommodate lines of people specifically for the Macguffin Games. A Pool Queue is pretty simple:

- Players are added in groups of one, two, three, or four.
  - The sequence of players *within* a group is not important.
- Players who are added in a group may not be split from that group.
- A Macguffin Game may be comprised of one or more groups of players.
  - If multiple groups are used, then they are merged into a single group.
- Older groups within the structure must *always* take precedence over newer groups.
  - For example, if the first group entered ***can*** be included in the requested group size, then it ***must*** be included. Similarly, the second group has priority over the successive groups.

See the last page for an example.

Programming Task:
You have been provided with an interface for a `PoolQueue`. It defines the minimum operations for the list. Your primary task is to create a class, `ConcretePoolQueue`, that implements the interface. Note that it should throw an `InsufficientElementsException` if a remove is attempted that can't be satisfied.
You are free to implement the class in whatever fashion you desire (within reason). For example, though a linked implementation is recommended (because it's much easier), a contiguous implementation is also allowed. Outside of completely bizarre design choices[1], you aren't required to concern yourself with things like optimal efficiency, or adhering to any specific design style. However, **refer to the interface for other requirements**.
You've also been provided with a sample test program, but feel free to make your own as well (you may, for example, wish to more rigorously test your implementation).

Writeup:
In addition to the sample output and printed code (see below), you must also include a **typed** writeup.
In it, discuss the Big-O complexities of *your* implementations for `addTriple`, `removeTriple`, and `hasTriple`, and considerations for complexities of the generalized add/remove.
Include brief explanations for each.

Submission:
Create `.pdf` output of sample executions of your program. **Zip** those, along with a `.pdf` of your writeup, and all of your `.java` source code, and submit through Sakai. Include the *provided* `.java` files **unedited**.

Lastly: remember that we have consultation time precisely for things like this.

1 For example, if you implement something resembling a roulette wheel, or a simulation of a trebuchet, or random dice...

The basic premise is simple. Assuming groups enter from the right and exit to the left:

```
  D
 BEG
ACFH
```

Suppose you wanted a group of 4. There are two possible patterns: ADEF, and BCGH.
In this case, only ADEF is considered valid, because A has the highest priority (and thus must be included if possible). Even though BC does have a higher priority than DEF, that doesn't matter; A trumps. Similarly, if a triple had been requested instead, even though there would be three potential combinations (ABC, DEF, and AGH), only ABC is valid.
(Note: as mentioned above, you can consider the result of ABC to be equivalent to ACB, BAC, BCA, CAB, and CBA)

```
---Sample Execution---
1. Add One        2. Add Two        3. Add Three      4. Add Four
5. Remove One     6. Remove Two     7. Remove Three   8. Remove Four
                  9. Test Options   0. Quit
1
Enter Name: Abby
:-                Added [Abby]

1. Add One        2. Add Two        3. Add Three      4. Add Four
5. Remove One     6. Remove Two     7. Remove Three   8. Remove Four
                  9. Test Options   0. Quit
2
First: Bobby
Second: Claire
:-                Added [Bobby,Claire]

1. Add One        2. Add Two        3. Add Three      4. Add Four
5. Remove One     6. Remove Two     7. Remove Three   8. Remove Four
                  9. Test Options   0. Quit
3
First: Darren
Second: Edgar
Third: Felix
:-                Added [Darren,Edgar,Felix]

1. Add One        2. Add Two        3. Add Three      4. Add Four
5. Remove One     6. Remove Two     7. Remove Three   8. Remove Four
                  9. Test Options   0. Quit
2
First: Gilligan
Second: Harry
:-                Added [Gilligan,Harry]

1. Add One        2. Add Two        3. Add Three      4. Add Four
5. Remove One     6. Remove Two     7. Remove Three   8. Remove Four
                  9. Test Options   0. Quit
4
First: Ilona
Second: Jill
Third: Kevin
Fourth: Larry
:-                Added [Ilona,Jill,Kevin,Larry]

1. Add One        2. Add Two        3. Add Three      4. Add Four
5. Remove One     6. Remove Two     7. Remove Three   8. Remove Four
                  9. Test Options   0. Quit
9
One:              true
Two:              true
Three:            true
```

```
Four:             true
Count:            12

1. Add One        2. Add Two        3. Add Three      4. Add Four
5. Remove One     6. Remove Two     7. Remove Three   8. Remove Four
                  9. Test Options   0. Quit
8
:>                [Abby,Felix,Edgar,Darren]

1. Add One        2. Add Two        3. Add Three      4. Add Four
5. Remove One     6. Remove Two     7. Remove Three   8. Remove Four
                  9. Test Options   0. Quit
9
One:              false
Two:              true
Three:            false
Four:             true
Count:            8

1. Add One        2. Add Two        3. Add Three      4. Add Four
5. Remove One     6. Remove Two     7. Remove Three   8. Remove Four
                  9. Test Options   0. Quit
8
:>                [Claire,Bobby,Harry,Gilligan]

1. Add One        2. Add Two        3. Add Three      4. Add Four
5. Remove One     6. Remove Two     7. Remove Three   8. Remove Four
                  9. Test Options   0. Quit
8
:>                [Larry,Kevin,Jill,Ilona]

1. Add One        2. Add Two        3. Add Three      4. Add Four
5. Remove One     6. Remove Two     7. Remove Three   8. Remove Four
                  9. Test Options   0. Quit
9
One:              false
Two:              false
Three:            false
Four:             false
Count:            0

1. Add One        2. Add Two        3. Add Three      4. Add Four
5. Remove One     6. Remove Two     7. Remove Three   8. Remove Four
                  9. Test Options   0. Quit
0
```