Week9Lab – 10 pts

Pre-lab questions

1. Show the contents of a queue after the following operations are performed. Assume the queue is initially empty:

```
enqueue(5);
enqueue(10);
enqueue(15);
dequeue();
dequeue();
enqueue(20);
enqueue(25);
dequeue();
enqueue(30);
enqueue(35);
enqueue(50);
dequeue();
```

50 → 35 → 30 → 25

➔ Right is the front of the queue and left is the rear of the queue

2. Show the contents of a stack after the following operations are performed. Assume the stack is initially empty:

```
push(5);
push(10);
push(15);
pop();
pop();
push(20);
push(25);
pop();
push(30);
push(35);
push(50);
pop();
```

5 → 20 → 30 → 35

➔ Right is the top of the stack and left is the bottom of the stack

Choose one of the following to develop into a program that will a linked list. Once chosen, do the following:

Understand the problem (restate in your own words, make any assumptions clear):

UML diagrams of the classes you will code, including the one with main:

Pseudocode of any non-trivial methods in each class (no pseudocode needed for basic setters and getters or no args constructors):

Name of files (.java) submitted:

Rummy Hand

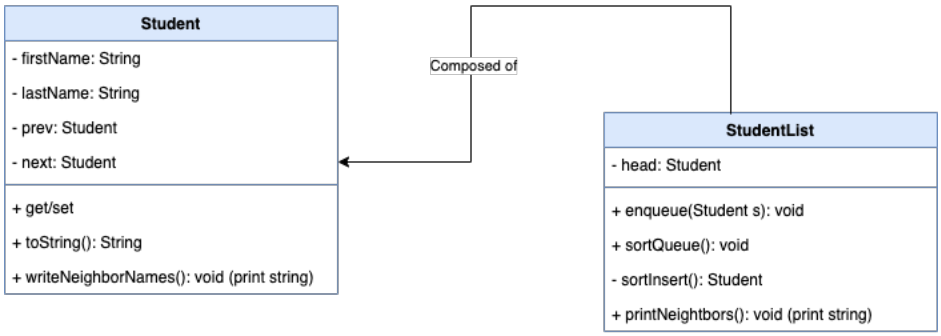
Create a basic playing card class that represents a typical poker deck card with suit and rank (no Jokers). Populate an array or arrayList with the 52 cards (hint, use a nested loop). Simulate a hand by dealing 5 cards randomly from the deck into a hand. Make the hand a linked list. Provide a method to sort the linked list by rank then suit (A of Spades, 3 of Diamonds, 3 of Spades, 8 of Hearts, King of Diamonds) and another method that sorts the list by suit then rank (3 of Diamonds, King of Diamonds, 8 of Hearts, A of Spades, 3 of Spades). Provide a method that will find if 3 or more cards in the sorted list are sequential (like 2, 3 and 4 of the same suit) or matching (3 of the same rank like three ten's) and pull them out to make a separate list.

Kindergarteners at Fire Drill

One way to keep track of lots of kids in a file drill or other emergency is to have each child know who is before them and behind them in a line. Simulate this behavior with a Student class that includes last name and first name. Create a doubly linked list of students so each student "knows" who is before them and behind them. Sort the list based on last name alphabetical order then have each student object write out the name of the person in front of them and the name of the person behind them.

We need to make a student data class that stores the first and last name of each student along with references to the previous and next student in line. We then need to make another helper class that creates the reference for the head of the queue, allows us to sort the queue by last name, and traverse through the line recording the student neighbors.

Files submitted: Student.java, StudentQueue.java, StudentQueueTester.java




```
def writeNeighborNames()
    if prev is is not null then
       Printf("The student before %s is %s\n", this.toString() ,prev);
     else
       Printf("%s is the first student\n", this.toString());

     if next is is not null then
       Printf("The student after %s is %s\n", this.toString() ,next);
     else
       Printf("%s is the last student\n", this.toString());

  def enqueue(Student s)
    if head is null then
       head = s;

    else
       Student currStudent = head;
       while currStudent.getNext() is not null
          currStudent = currStudent.getNext();

       currStudent.setNext(s);
```

```
        s.setPrev(currStudent);



def sortQueue()

    Student sorted = null;
    Student curr = head;

    while curr is not null
        Student nextStudent = curr.getNext();
        curr.setNext(null);
        curr.setPrev(null);
        sorted = sortedInsert(sorted, curr);
        curr = nextStudent;
    head = sorted;

def sortedInsert(Student headRefStudent, Student newStudent)

    Student curr;

    // Case where Queue is empty
    if headRefStudent == null then
        headRefStudent = newStudent;



    // Case where the student needs to be at the beginning of the Queue
    else if headRefStudent.getLastName().compareToIgnoreCase(newStudent.getLastName()) >= 0 then
```

```
        newStudent.setNext(headRefStudent);
        newStudent.getNext().setPrev(newStudent);
        headRefStudent = newStudent;

    else
        curr = headRefStudent;


        // Move to the reference after which the new student is to be inserted
        while curr.getNext() is not null &&
curr.getNext().getLastName().compareToIgnoreCase(newStudent.getLastName()) < 0 do
            curr = curr.getNext();


        // Set the appropriate links
        newStudent.setNext(curr.getNext());

        // Case where the new new node is not inserted at the end of the Queue
        if curr.getNext() is not null then
           newStudent.getNext().setPrev(curr.getNext());


        curr.setNext(newStudent);
        newStudent.setPrev(curr);

    return headRefStudent;


def printNeighbors()
```

```
Student currStudent = head;
while currStudent is not null do
   currStudent.writeNeighborNames();
   System.out.println();
   currStudent = currStudent.getNext();
```

Conga Line

If you've ever been to an American wedding reception or prom, you probably saw or participated in a Conga Line. This is a line dance where one person is at the head and as they dance around the room, other dancers join up by holding on to the hips of the person in front of them. This is definitely a linked list! Simulate a Conga Line with Dancer objects in a linked list with a list header that knows the first and last dancers in the line. Add dancers to the line, with each being added to the end of the line. Make sure Dancer has a toString() method so you can print the line and see that the order is correct.