CSCI 1101 - 2017
Assignment 4

Complete the assigned tasks and submit a ZIP file containing your finished source code (these are the .JAVA files) on Brightspace.

Your assignment **must be your own work**. You may ask questions of your instructor, course TAs, or Learning Centre TAs. If you discuss the ideas in this assignment with your classmates, **it is your responsibility to make sure you are not plagiarizing** – take no notes during the discussion, wait 30 minutes, and whatever you remember is safe to be used in your own code.

Submission Deadlines (firm):
Due: August 3, by 12:00pm (noon)
Late submissions **CANNOT BE ACCEPTED!**

Submissions outside of these given deadlines are **not** accepted.

Before submitting, check that:

1. Your code is properly formatted, includes reasonable comments (including the header comment), and is properly tested for the given problem.
2. Your submission ZIP file contains your **source code** .JAVA files, not your compiled .CLASS files.
3. Your code **compiles** and can be run – even if your program does not do everything perfectly, make sure it compiles before you submit.

Header Comments

Your code should now include header comments for **all** of your class (.java) files. The comment should include the lab/assignment number, the course (CSCI 1101), the name of your program and a short description of the entire class, the date, your name and Banner ID, and a declaration that matches the first page of this document (e.g., whether you received help). See the example below for what a header comment should include:

```
/*Lab1, Question 1 CSCI 1101
   Student.java holds information about a student at Dalhousie in CSCI1101 and
   their grades
   June 29, 2015
   John Smith B00112345
   This is entirely my own work. */

public class Student {
//rest of Code
```

If applicable, your demo class should then also have a similar header:

```
/*Lab1, Question 1  - demo class CSCI 1101
   StudentDemo.java is a demo program for the Student class. It creates student
   objects, and compares different students.
   June 29, 2015
   John Smith B00112345
   I received help with creating Student objects from my TA but the rest is my
   own work. */

public class StudentDemo {
//rest of Code
```

Question 1
We will re-visit the "War" game you implemented in Assignment 3. This time, we will make use of our new knowledge of data structures and see if this can make our code more streamlined and efficient!

Minimum files to submit: LinkedList.java, Node.java, Card.java, Deck.java, Pile.java, Demo.java

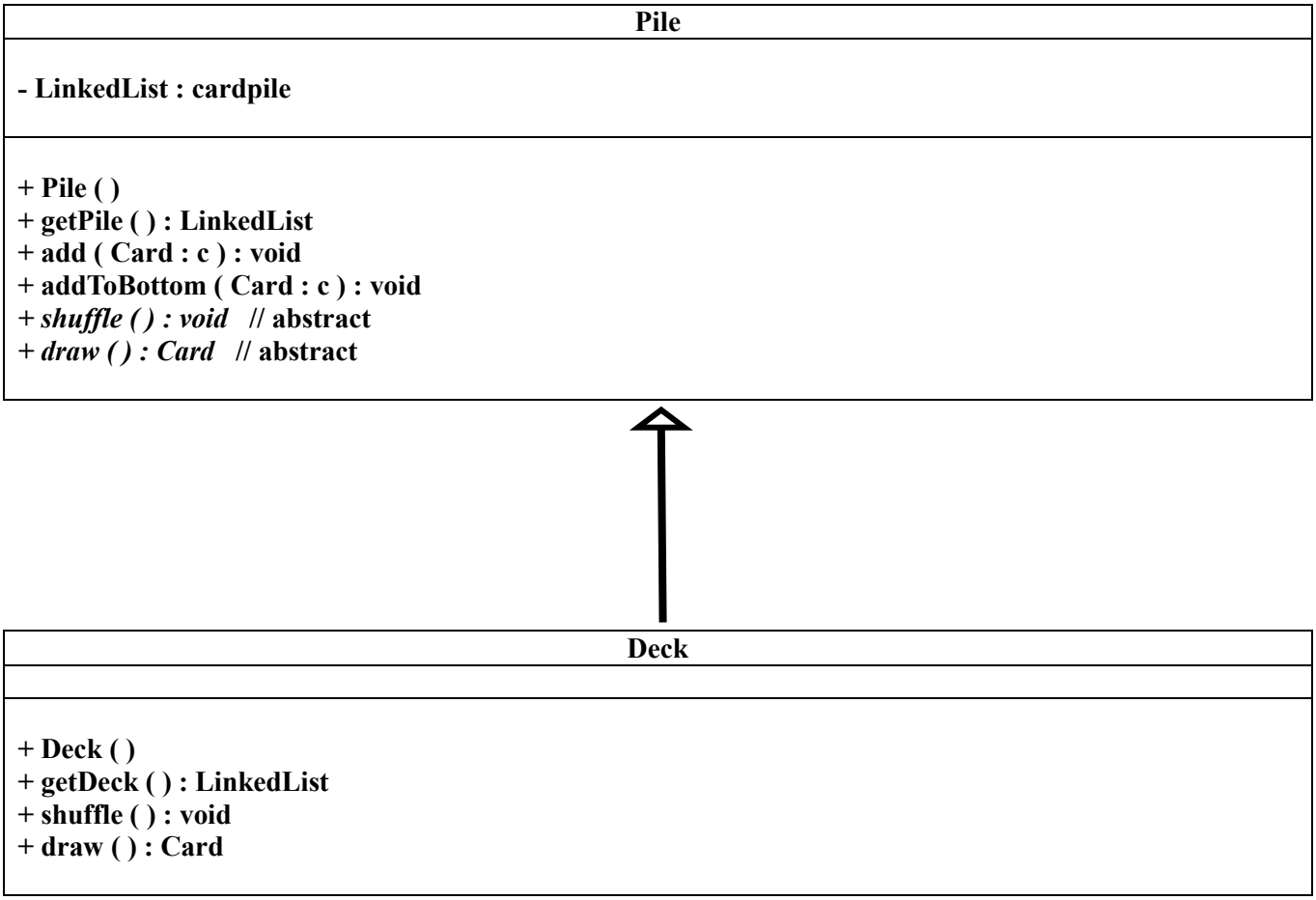
You will be using similar LinkedList and Node implementations as we have seen in previous labs and in class, but you must update these to work with objects of class Card instead of String objects. As a summary, your LinkedList and Node classes should adhere to the following UML descriptions:

LinkedList
- head : Node **- count : int**
+ LinkedList ( ) **+ addToFront(Card : d) : void** **+ size ( ) : int** **+ isEmpty ( ) : boolean** **+ clear ( ) : void** **+ getFrontData ( ) : Card** **+ getFrontNode ( ) : Node** **+ enumerate ( ) : void** **+ removeFront ( ) : void** **+ addToEnd (Card : c) : void** **+ removeLast ( ) : void** **+ contains (Card : c) : int** **+ add (int : i , Card c) : void** **+ remove (int : i) : void** **+ getNode (int : i) : Node** **+ addAll (LinkedList : l) : void**

Node
- data : Card **- next : Node**
+ Node (Card : c , Node : n) **+ toString ( ) : String** **+ get/set methods**

There should not be any extra methods or variables within these classes.

Next, you may choose to modify **or** re-write your own code for the Card and Deck classes. These two classes should now work with the LinkedList and Node classes defined above. Additionally, the Deck class should now use the abstract class "Pile" - the file Pile.java is included in this assignment hand-out. A Pile is intended to represent an arbitrary pile of cards, whether in a deck, or a player's hand, or a discard pile, or something else. It tracks an ordered sequence of Cards and provides a few rudimentary methods. In particular, it demands that its subclasses implement shuffle() and draw() methods. The relationship between Pile and Deck is given in the UML diagram below:



Parts of the implementations of draw() and shuffle() within your class Deck may be similar (or even identical) to methods you wrote for the previous assignment.

Your Demo class will execute a game of War as before (shuffling Cards, dealing to players, flipping in sequence, etc.) but this time it should work with the classes defined above. Portions of your Demo class may be similar (or even identical) to what you wrote for the previous assignment. Please see the Assignment 3 handout for an example of the output formatting and to recall the exact sequence of processing.

Bonus: This component is not required, but completion will give you bonus marks on this assignment. Write a class PlayerHand that extends the Pile class, and use this to represent each player's hand (or pile of cards) during a game of War. Your implementation may define any extra methods or variables you deem appropriate. By using

this class, you will be "moving" Card objects around between a Deck and two PlayerHands, so be aware of if/when you want to use shallow versus deep copies of objects.

Exercise 2:

Question 2

We will also re-visit the "Connect4" game you implemented in Assignment 2. This time, there will only be one human player, and a computer AI (Artificial Intelligence) will take the role of the second player. You will design and implement this AI to play the game of Connect4.

Minimum files to submit: Chip.java, Board.java, Connect4Board.java, Play.java

These four classes will function roughly the same as in Assignment 2, but the details of this are up to you. You may modify any of your own code for these classes to reflect your increased understanding (e.g., you may have an array that you would like to replace with an ArrayList) or improve functionality. Additionally, you may define any new classes, methods, or variables that you deem appropriate, so long as they are properly documented in your comments.

The biggest change will be within the Play class, since now one of the players is computer-controlled. Your AI should be able to play the game in a simple way, and at minimum should check for winning and blocking moves for each of its turns. Your AI player does not need to plan beyond its current turn.

Bonus: This component is not required, but completion will give you bonus marks on this assignment. In the final (winning/losing/tying) round of each game of Connect4, print the game board status and the winner notification to a text file as well as to the standard user output. The filename should be the same for each game, should end in ".txt", and its contents should be over-written with each game played.