ACS-2947-050
Assignment #2
Due by Friday October 26 at 11:59 pm

- Submit your `.java` files to 2947L-070@acs.uwinnipeg.ca or 2947L-071@acs.uwinnipeg.ca
- Include your name and student number in each file as a comment
    - Document the `Peg` class using Javadoc notation
    - Include comments as needed
    - Use appropriate exception handling where necessary

PART A

Write a simple text-based simulation of the classic board game Mastermind, where a single player is the code breaker and the system is the code maker. The system selects a code of four coloured pegs and the player tries to guess the secret code.

Each round, the player makes a guess, and the system tells the player how many pegs of the guess were *exact* matches to the code (correct in both color and position, marked `'x'`), and how many colours were *non-exact* matches to the code (correct color placed in the wrong position, marked `'o'`). The feedback is displayed in a 2x2 grid format similar to the board game.

e.g. suppose the code were: red black blue green

```
Guess #1:
red blue green yellow
x o
o -
```

The feedback shows that there is one exact match and 2 non-exact matches. Notice that this configuration does not indicate which pegs are exact matches.

The player makes guesses until a) the guess matches the code (player wins!) or b) 10 guesses are made but no match (system wins).

1. Create the generic `ArrayList` class that implements the `List` interface.
    a. Override the `equals` method that checks if the `ArrayList` is equivalent to the given instance.
    b. Add an add method that adds a new element by appending it to the end of the list.
2. Create a class named `Peg`. Each peg should hold a colour and an indication of whether the peg is a match or not.
3. Write a driver class that acts as the code maker for the Mastermind game.

Your program should do the following:

a. Have an `ArrayList` that holds a set of 4 pegs whose colours are randomly generated and defaulted to no-match. Each peg has a colour of 6 possibilities, duplicates are allowed.
b. Have another ArrayList that holds pegs that represents the player's guess
c. Determine if the 2 ArrayLists are equal:
    i. if so, notify the player and end the game
    ii. If not, provide the user feedback on their guess:
        - Determine whether or not each peg of the guess is a direct match, and mark it accordingly
            - You will need to compare the guess against the code, and determine the number of exact and non-exact matches. However, *you must be very careful to avoid counting any of the pegs twice*.
            - Suggestion: make at least two passes to compare the guess and the code. In the first pass, look for exact matches and in the second pass, look for non-exact matches. Consider using an array of characters to mark the match types (`x`, `o` or `-`) and eventually display as feedback.

f. After their 10th guess, if it is not a match, inform the player that the system won.

Sample output
[code: white blue yellow green]

```
System:     Guess #1:
Player:     blue blue blue blue
System:     x -
            - -

            Guess #2:
Player:     blue red red red
System:     o -
            - -

            Guess #3:
Player:     yellow blue yellow yellow
System:     x x
            - -
```

```
        Guess #4:
Player: green blue yellow green
System  x x
        x -

        Guess #5:
Player: green blue yellow black
System  x x
        o -

        Guess #6:
Player: white blue yellow green
System: You cracked the code!
```

PART B

Implement a Positional List using an ***array***. Refer to [page 281]() in your textbook.

1. Create 2 classes called `ArrayPositionalList` with a nested `ArrPos` that implement the [`PositionalList`]() and [`Position`]() interfaces respectively.

2. Demonstrate the use of your implementation with the driver code you wrote for Lab 4 question 1. You may copy/paste from the sample solution provided in Teams.

To get you started, a beginning `ArrayPositionalList` implementation with the nested `ArrPos` class can be found [here](). It implements the `Position` interface (just like `Node` in a *linked* positional list). Note that there is no next or prev, but only an integer `index` and `getElement()`

a) Declare your fields and create your constructors (it may help to look at `ArrayList`). Your array will store `ArrPos` objects.
b) Add your `size()` and `isEmpty()` methods
c) Implement the `first()` and `last()` methods: how would you get the first and last elements from an array? This should form a basis of how to move from linked to array. From there you can start thinking about how to convert all the methods from linked-based to array-based.

To consider: with `LinkedPositionalList`, you get the previous and next positions through the node (which is a `Position`) and `getNext()` and `getPrev()` methods. With `ArrayPositionalList` you get the next and previous through the `Position` as well, but with the `getIndex()` and the array. Instead of simply calling `node.next()` you will find out what `arrpos.getIndex()` is, then return the `ArrPos` at the *next index* of your array.

Note that:

- you will need a way to validate and explicitly cast `Position` objects to `ArrPos` objects in order to use `ArrPos` methods like `getIndex()`
- many methods declare exceptions in its signature: most can be handled in common private utility methods
- the `ArrPos` field `index` needs to be updated with any methods that require a *shift* in elements

Suggestion:

- Override the toString method to display both index and element i.e. an ArrayPositionalList populated with integers 1-5

  ```
  [0] 1 [1] 2 [2] 3 [3] 4 [4] 5
  ```

  - Can be helpful for testing/debugging
- Test each individual method as you write it