ACS-2947-002/050
Assignment 2
Due by Sunday, March 3, 11:59 PM

Instructions

- Submit your `.java` files (together in a `Assign2.zip` file) via Nexus.
- Include your name and student number as a comment in every file.
- Document the classes using Javadoc notation.
- Include comments as needed and use exception handling where necessary.

PART A – ArrayList (30 marks)

Write a simple text-based version 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.

In 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 *partial* matches to the code (correct color placed in the wrong position, marked `'o'`). The feedback is displayed in a 2 x 2 grid format similar to the board game.

e.g., suppose the code is *black red blue green*

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

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

The player makes guesses until either: a) the player breaks the code (player wins!) or b) 11 guesses are made but did not result in a full match (system wins).

1. Create the generic `ArrayList` class that implements the provided `List` interface (note that `List` extends `Iterable`)
   a. Overload the `add` method: include another `add` method that will have one parameter, an element that adds to the ***end*** of the list.
   b. Make your `ArrayList` dynamic: the array should grow to double its current capacity if it runs out of space and shrink to half its current capacity when the number of elements in the arraylist goes below *N*/4, where N is the current capacity of the array. Modify `add()` and `remove()` methods and include a `resize()` method to support the dynamic structure. Set the default capacity to 4.
   c. **Override** the `equals` method of the Object class so that it checks if the `ArrayList` is equivalent to the given instance. Consider an appropriate definition of equivalence.

2. Create a class named `Peg` with field `colour.` Include any other fields/methods to help with gameplay. Override the `equals` method to return true if the colours match.
3. Write a `Game` class that acts as the code maker and handles the mechanics of the Mastermind game. Include a minimal main method that instantiates the game and invokes the method to play the game.
4. In the main method, also illustrate how the capacity of your array would changes as objects are added and removed. *That is, include lines of code that to add/remove and show that it shrinks or enlarges as elements are removed or added, respectively.*

   Your program should have the following:

   a. An instance of `ArrayList` that holds a set of 4 pegs of which colours are randomly generated. Each peg has a colour of 6 different possibilities (duplicates are allowed, blanks are not).
   b. Another `ArrayList` that holds pegs that represent the player's guess.
   c. A game loop that prompts the user for their guess and determines 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 if each peg of the guess is a match and mark it accordingly. You will need to compare the guess against the code and determine the number of exact and partial matches.
   d. After their 11th guess, if it is not a full match, inform the player that the system won.

Note:

- You may assume that the player knows the valid colours i.e., if it is an invalid colour, the player loses and the game is over.
- Enums are optional (e.g., `Colour`)

Suggestions:

- Display the generated code (e.g., first line of the sample output below) for testing and remove before submitting
- For guess feedback: must be careful to avoid counting any of the pegs twice; 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 partial matches.

Sample output:

[code: white blue yellow green]

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

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

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

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

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

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

PART B – PositionalList (60 marks)

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

1. Create a class called `ArrayPositionalList` (APL) with a nested class `ArrPosition` that implement the provided `PositionalList` and `Position` interfaces, respectively.

2. Demonstrate the use of your APL in a `PartB_Driver` class by doing the following. Create a static method called `removeConsecutiveDuplicates` that removes any consecutive duplicate strings from an array positional list of Strings and returns the number of strings left after the removal. After calling the method, the positional list parameter should contain the same sequence of strings as before but with any consecutive duplicates removed. Illustrate your method using the following sets of strings and display the content of the list after executing the method.

    - harry ron tom tom tom hermione
    - harry harry tom ron mary harry
    - tom ron harry hermione mary
    - mary mary tom james hermione hermione james harry harry harry

NOTE: You **MUST** use a combination/variation of ALL the add and set methods (`AddFirst, AddLast, AddBefore, AddAfter,` and `set`) in creating **one** of the original lists above.

You **MUST NOT** use any other auxiliary data structure e.g., arrays in your implementation of `removeConsecutiveDuplicates`

3. Create a static method called `InsertionSort` that sorts the items of an array positional list of characters based on the insertion sort algorithm. Refer to slides 16 – 20 of L02_ for the array-based algorithm and implementation. Illustrate your sorting method with the following list of characters: S, C, R, A, M, B, L, E, G, A, M, E.

To get you started, a beginning `ArrayPositionalList` implementation with the nested class `ArrPosition` is provided.

- `ArrPosition` 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 generic `element`.
- Fields and constructors are included, array stores `ArrPosition` objects.

a. Add your `size()` and `isEmpty()` methods.
b. Implement the `first()` and `last()` methods: how would you get the first and last elements from the 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 implementation.

Consideration:
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()` method and the array. Instead of simply calling `node.next()` you will find out what `ArrPosition.getIndex()` is, then return the `ArrPosition` at the *next index* of your array.

Note:
- You will need a way to validate and explicitly cast `Position` objects to `ArrPosition` objects in order to use `ArrPosition` methods like `getIndex()`
- The `index` field of the `ArrPosition` class needs to be updated with any methods that require a *shift* in elements.
- Override the `toString` method to display both index and element. For example, the content of the `ArrayPositionalLists` for the last set of strings above will be displayed as in the sample output after executing `removeConsecutiveDuplicates`. This can be useful for testing/debugging.
- The APL is not required to be dynamic.

Suggestions:

- Many methods declare exceptions in their signature: most can be handled in common private utility methods.
- Think about how you can reuse code e.g., for the different add methods
- Test each individual method as you write it.

Sample Output (Showing only the last set of Strings for Question 3)

```
Original positional list:
[0] mary [1] mary [2] tom [3] james [4] hermione [5] hermione
[6] james [7] harry [8] harry [9] harry

Number of entries after call: 6
List with duplicates removed:
[0] mary [1] tom [2] james [3] hermione [4] james [5] harry


Sorted characters using Insertion sort algorithm:
[0] A [1] A [2] B [3] C [4] E [5] E [6] G [7] L [8] M [9] M [10] R
[11] S
```

Submission

Submit your **`Assign2.zip`** file that include all the assignment files (`List.java`, `ArrayList.java`, `Peg.java`, `Game.java`, `PositionalList.java`, `Position.java`, `ArrayPositionalList.java`, `PartB_Driver.java`, any other class used by your code) via **Nexus**.