COMP 1020
Lab 8

MATERIAL COVERED

- ArrayLists

Notes:

- The three exercises are independent – they can be done in any order.
- Only one of the three exercises is required.
- Try to complete as many as you can.

Basic ArrayList operations

Most of the basic **`ArrayList`** operations: **`size`**, **`add`**, **`remove`**, **`get`**, and **`indexOf`**, are used in this problem. Generic **`ArrayLists`** (not **`ArrayList<Type>`**) will be used in all of the exercises in this lab.

Begin with the file **`TemplateLab8Bronze.java`**. Complete the method

```
ArrayList extractDuplicates(ArrayList a1, ArrayList a2)
```

which will look for elements that appear in *both* **`a1`** and **`a2`**. When one is found, it should be removed from *both* **`a1`** and **`a2`**. You can assume that neither list will contain a value more than once. An **`ArrayList`** containing all of the elements that were *removed* should be returned as the result of the method.

The output printed by the main method should be:

```
a1 is [45, 12, 98, 34, 6, 42]
a2 is [6, 81, 36, 12, 77, 42]
removed elements: [42, 6, 12]
a1 is now [45, 98, 34]
a2 is now [81, 36, 77]
```

Notes:

1. When you're going through all of the elements of a list, but you're shrinking the list at the same time, unexpected things may happen. Write the loop very carefully. You can't just use the usual **`for`** loop.
2. If you use **`get( )`** to obtain an element from **`a1`** or **`a2`**, its type will be **`Object`**.

Shut up and deal, I'm losing.

1. An **`ArrayList`** is a good way to store a deck of cards (or a partial deck, or a hand). Begin with the file **`TemplateLab8Silver.java`** and then complete the three small methods whose headers appear at the bottom of the file.

2. Complete the method **`ArrayList makeDeck(int numCards)`** which should return a new **`ArrayList`** containing the numbers from **`0`** to **`numCards-1`**. (These numbers will have type **`Integer`**, not **`int`**, but since the two are interchangeable it will make no real difference.)
3. Complete the method **`void shuffle(ArrayList deck)`** which will rearrange the elements of **`deck`** into a random order. The usual technique is: Choose a random card position from **`0`** to **`deck.size()-1`**. Delete the card in that position and put it in position **`0`** instead. From now on, leave that card alone. Choose a random card position from **`1`** to **`deck.size()-1`**. Delete the card in that position. Put it in position **`1`**. And so on for positions **`2..deck.size()-2`**. (There's no need to select a random card to go into position **`deck.size()-1`** because there's only one card left at that point anyway.)
4. Complete the method **`ArrayList[] deal(ArrayList deck, int numHands, int numCards)`** which will deal the indicated number of hands, with the indicated number of cards in each hand, from the top of the deck. The cards that are dealt should be *removed* from the deck. It should return an array of **`ArrayList`**s, where each **`ArrayList`** contains the cards in one hand. It should deal in the normal way, with consecutive cards going into different hands, in a circular manner, as shown in the example below.
5. When the main method is run, it should produce random output similar to the following. A small deck of only 20 cards is used to keep the output small.

```
The new deck is [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
The shuffled deck is [11, 14, 19, 2, 17, 1, 7, 9, 6, 8, 10, 3, 12, 16, 4, 15, 13, 5, 0, 18]
How many hands should be dealt? 3
How many cards in each hand? 4
The hands are:
Hand 0: [11, 2, 7, 8]
Hand 1: [14, 17, 9, 10]
Hand 2: [19, 1, 6, 3]
The remaining deck: [12, 16, 4, 15, 13, 5, 0, 18]
```

Nested ArrayLists

The elements in an **`ArrayList`** may be any type of **`Object`**, including other **`ArrayLists`**. Such "nested lists" are very powerful and flexible, and in fact there are entire languages built around this idea.

1. Begin with the file **`TemplateLab8Gold.java`**. Take a look at the **`main`** method, and run the program and look at the output. This program creates **`ArrayLists`** nested inside other **`ArrayLists`**, to a depth of 4 levels.
2. Complete the method **`ArrayList flatten(ArrayList a)`** at the bottom of the file. This method should accept an **`ArrayList a`**, which may contain other **`ArrayLists`** inside it, nested to any depth. It should create and return a new **`ArrayList`** which is a "flattened" version of the original, which still contains all the data that was present in the original, in the same order, but with the nesting removed (essentially, the **`[]`** should disappear). See the sample output below.
3. The correct output from the `main` method, once **`flatten`** is working, is:

```
The nested ArrayList is: [23, [19, 46], Hello, [45.5, World, [11, 22, [33]]]]
The flattened ArrayList is: [23, 19, 46, Hello, 45.5, World, 11, 22, 33]
```

4. Note: If **`a1`** and **`a2`** are both **`ArrayList`** objects, then **`a1.addAll(a2)`** will add all of the elements of **`a2`** to the end of **`a1`** (a concatenation operation like + for Strings). That would be useful here.