ArrayList and Vector

A brief recap

- Both ArrayList and Vector are classes in Java that implement the List interface and provide dynamic arrays that can be resized at runtime. They have similar functionality, but there are some differences:
- ArrayList is part of the Java Collections Framework introduced in Java 1.2, while Vector is a legacy class that was introduced in Java 1.0.
- Vector methods are synchronized, which means they are thread-safe, but can be slower than ArrayList when used in a single-threaded environment. ArrayList methods are not synchronized, but you can synchronize them manually using external synchronization.
- Vector can grow or shrink by specifying the increment size or the new size explicitly, whereas ArrayList automatically resizes itself when necessary.
- It is possible to store a primitive value directly in an array while ArrayList can only hold references. Therefore you need to use wrapper classes for primitive types (Integer for int, Character for char etc.)
- There is no special syntax for a multi-dimensional ArrayList, you can create it via nesting (e.g. ArrayList<ArrayList>).

Critical Thinking

- The following code is supposed to store numbers entered by user in memory until the user enters 0. Which statement should be put in place of the comment?

```java
public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    ArrayList<Integer> nums = new ArrayList<>();
    int cur;
    do {
        cur = sc.nextInt();
        // Here
    } while(cur != 0);
}
```

a) `nums[i] = cur;`

b) `nums.add(cur);`

c) `nums = nums.add(cur);`

d) `nums.append(cur)`

answer: b

- What will be the output of the following program?

```
public static void main(String[] args){
    ArrayList<Integer> nums = {2,3,5};
    System.out.println(nums.get(0));
}
```

a) 0

b) 2

c) Runtime Error

d) Compiler Error

answer: d

- What will be the output of the following program?

```
public static void main(String[] args){
    ArrayList<Integer> nums = new ArrayList<>();
    nums.add(3);
    nums.add(5);
    nums.add(nums.get(0));
    System.out.println(nums);
}
```

a) [3, 5, 3]

b) [3, 3, 5]

c) [3, 5]

d) Something like [I@1c4af82c

answer: a

- In which of the following scenarios we need a dynamically resizable storage?

a) Storing integer values received from user where the first value entered is assumed to be the number of values following.

b) Storing integer values provided in the source code.

c) Storing integer values which were accessed through an array parameter in a function.

d) Storing integer values received from user until user enters a specific value.

answer: d

- What will be the output of the following program?

```
public static void main(String[] args){
    Vector<Integer> v = new Vector<>(0,10);
    v.add(4);
    System.out.println(v.capacity() - v.size());
}
```

a) 1

b) 0

c) 9

d) 4

answer: c

- What will be the output of the following program?

```
public static void main(String[] args){
    ArrayList<Integer> arr = new ArrayList<>();
    arr.add(4);
    System.out.println(arr.capacity() - arr.size());
}
```

a) 1

b) Compiler Error

c) 4

d) 9

answer: b

Practice

- Write a method which takes two ArrayList of integers and returns their intersection (the elements that appear in both lists) in the order they appear in the first list.

Main method:

```
public static void main(String[] args){
    ArrayList<Integer> a = new ArrayList<>();
    ArrayList<Integer> b = new ArrayList<>();
    for(int i=0; i<10; i++){
        a.add((int)(Math.random()*10));
        b.add((int)(Math.random()*10));
    }
    System.out.println(a);
    System.out.println(b);
    System.out.println(intersection(a,b));
}
```

solution:

```
static ArrayList<Integer> intersection( ArrayList<Integer> a, ArrayList<Integer> b){
    ArrayList<Integer> res = new ArrayList<>();
    for(int i=0; i<a.size(); i++)
    if(b.contains(a.get(i)))
        res.add(a.get(i));
    return res;
}
```

- Write a method which takes an ArrayList of integers and removes any duplicate elements in it.

Main method:

```
public static void main(String[] args){
    ArrayList<Integer> a = new ArrayList<>();
    for(int i=0; i<10; i++)
        a.add((int)(Math.random()*10));
    System.out.println(a);
    uniqify(a);
    System.out.println(a);
}
```

solution1:

```
static void uniqify(ArrayList<Integer> a) {
    for(int i=0; i<a.size(); i++)
        if(a.lastIndexOf(a.get(i)) != i)
            a.set(i, null);
    while(a.remove(null));
}
```

solution2:

```
static void uniqify(ArrayList<Integer> a) {
    for(int i=0; i<a.size(); i++)
        if(a.lastIndexOf(a.get(i)) != i)
            a.remove(i--);
}
```

- Write a function which takes an ArrayList of characters and removes all 'e' characters in it without changing the order of other elements.(removeAll() method is not allowed)

Main method:

```
public static void main(String[] args){
    ArrayList<Character> arl = new ArrayList<>();
    for(int i=0; i<10; i++)
        arl.add((char)
    Math.round(Math.random()*3+100) );
    System.out.println(arl);
    cleanse(arl);
    System.out.println(arl);
}
```

solution:

```
static void cleanse(ArrayList<Character> arl) {
    while(arl.remove(Character.valueOf('e')));
}
```

- Write a function which takes a positive integer and returns all its positive divisors.

example

- divisors(20) = [1, 2, 4, 5, 10, 20]
- divisors(4) = [1, 2, 4]

Main method:

```
public static void main(String[] args){
    System.out.println(divisors(20));
    System.out.println(divisors(4));
}
```

solution:

```
static ArrayList<Integer> divisors(int n) {
    ArrayList<Integer> res = new ArrayList<>();
    for(int i=1; i<=n; i++)
        if(n%i==0)
            res.add(i);
    return res;
}
```

- Write a function which takes an array of integers and returns an ArrayList of integers such that for each element in the array a fair coin is tossed to decide whether or not to include it in the ArrayList.

  example

  - randomSubset([2, 3, 5, 6]) returns [2, 6] (maybe)
  - randomSubset([2, 3, 5, 6]) returns [] (maybe)

Main method:

```
public static void main(String[] args){
    int[] arr = {2, 3, 5, 6};
    System.out.println(randomPick(arr));
}
```

solution:

```
static ArrayList<Integer> randomPick(int[] arr) {
    ArrayList<Integer> res = new ArrayList<>();
    for(int elem: arr)
        if(Math.random() < 0.5)
            res.add(elem);
    return res;
}
```

- Write a function which takes an ArrayList of integers and a rotation amount (integer) and rotates the ArrayList by the amount given.

  example

  - rotate([3,1,5,7], 0) –> [3,1,5,7]
  - rotate([2,7,4,4,4], 2) –> [4,4,2,7,4]

- rotate([2,5,7,2,1,3], -1) –> [5,7,2,1,3,2]

Main method:

```
public static void main(String[] args){
    ArrayList<Integer> arl = new ArrayList<>();
    for(int i=0; i<10; i++)
        arl.add((int)(Math.random()*10));
    System.out.println(arl);
    rotate(arl, 2);
    System.out.println(arl);
}
```

solution1:

```
static void rotate(ArrayList<Integer> arl, int k) {
    if(k < 0)
        for(int i=0; i<-k; i++)
            arl.add(arl.remove(0));
    else
        for(int i=0; i<k; i++)
            arl.add(0, arl.remove(arl.size()-1));
}
```

solution2:

```
static void rotate(ArrayList<Integer> arl, int k) {
// This is also OK but harder to understand
    for(int i=0; i<-k; i++)
        arl.add(arl.remove(0));
    for(int i=0; i<k; i++)
        arl.add(0, arl.remove(arl.size()-1));
}
```

- Write a program which takes integers from user until user enters a 0 and displays them in ascending order (0 won't be displayed).

solution:

```
public static void main(String[] args){
    Scanner input = new Scanner(System.in);
    // get numbers
    ArrayList<Integer> arl = new ArrayList<>();
    int next;
    while((next = input.nextInt()) != 0)
        arl.add(next);
    // sort them
    Collections.sort(arl);
    // display
    for(int num: arl)
        System.out.println(num);
}
```

Project

1. Write a function which takes an ArrayList of Boolean and fills its last half with null references. Note that return-type of your function must be void. You can assume that the size of the parameter list is always even.

Main method:

```
public static void main(String[] args){
    ArrayList<Boolean> arl = new ArrayList<>();
    for(int i=0; i<10; i++)
        arl.add(Math.random() < 0.5);
    System.out.println(arl);
    fill(arl);
    System.out.println(arl);
}
```

solution:

```
static void fill(ArrayList<Boolean> arl) {
    for(int i=arl.size()/2; i<arl.size(); i++)
        arl.set(i, null);
}
```

2. Write a function which takes an ArrayList of Strings words and adds just enough strings at the end of it to ensure that the following condition is satisfied: “For every string w in words the reverse of w is also in words.” If the condition is already satisfied, no action is needed.

   example

   - complete([“aa”, “aca”, “ba”, “ab”]) would not change anything.
   - complete([“ab”, “qe”, “eq”]) appends “ba”.

Main method:

```
public static void main(String[] args){
    ArrayList<String> arl = new ArrayList<>();
    arl.add("ab");
    arl.add("qe");
    arl.add("eq");
    System.out.println(arl);
    complete(arl);
    System.out.println(arl);
}
```

solution:

```
public static void complete(ArrayList<String> words) {
    int size = words.size();
    for (int i = 0; i < size; i++) {
        String word = words.get(i);
        String reverse = new StringBuilder(word).reverse().toString();
        if (!words.contains(reverse)) {
            words.add(reverse);
            size++;
            }
        }
    }
```

3. Write a function which takes an ArrayList of Strings words and removes just enough strings from it to ensure that the following condition is satisfied: "For every string w in words the reverse of w is also in words." If the condition is already satisfied, no action is needed.

   example

   - remove(["aa", "aca", "ba", "ab"]) would not change anything.
   - remove (["ab", "qe", "eq"]) removes "ab".

Main method:

```
public static void main(String[] args){
    ArrayList<String> arl = new ArrayList<>();
    arl.add("aa");
    arl.add("aca");
    arl.add("ba");
    arl.add("ab");
    System.out.println(arl);
    remove(arl);
    System.out.println(arl);
}
```

solution:

```
public static void remove(ArrayList<String> words) {
    int size = words.size();
    for (int i = 0; i < size; i++) {
    // for (int i = size - 1; i >= 0; i--) { // if using this remove "i--" at end
        String word = words.get(i);
        String reverse = new StringBuilder(word).reverse().toString();
        if (!words.contains(reverse)) {
            words.remove(i);
            size--;
            i--;
        }
    }
}
```

4. Write a Java method that takes two ArrayLists of integers as input and returns a new ArrayList that contains only the integers that appear in both input lists, in the order they first appear in the first input list.

   example

   - common([1, 2, 3, 4, 5], [2, 4, 6, 8, 10]) returns [2, 4]
   - common([1, 2, 3, 4, 5], [6, 7, 8, 9, 10]) returns []

Main method:

```
public static void main(String[] args){
    ArrayList<Integer> arl1 = new ArrayList<>();
    ArrayList<Integer> arl2 = new ArrayList<>();
    ArrayList<Integer> arl3 = new ArrayList<>();
    for(int i=0; i<5; i++){
        arl1.add(i+1);
        arl2.add(i+6);
        arl3.add(i+3);
    }
    System.out.println(arl1);
    System.out.println(arl2);
    System.out.println(arl3);
    System.out.println(common(arl1, arl2));
    System.out.println(common(arl1, arl3));
}
```

solution:

```
public static ArrayList<Integer> common(ArrayList<Integer> arl1, ArrayList<Integer> 
    ArrayList<Integer> res = new ArrayList<>();
    for (int elem : arl1) {
        if (arl2.contains(elem)) {
            res.add(elem);
        }
    }
    return res;
}
```

Extra

- Write a function which takes a string representing a large corpus of English (a text basically) and returns an ArrayList of Strings consisting of unique English words that appear at least once in this corpus, sorted alphabetically.

getWords("Once upon a time, there was a queen ruling a cold land. She had a big castle in which there were bright gardens as well as dark dungeons.") returns

[a, as, big, bright, castle, cold, dark, dungeons, gardens, had, in, land, once, quein if statementen, ruling, she, there, time, upon, was, well, were, which]

Main method:

```
public static void main(String[] args){
    String text = "Once upon a time, there was a queen ruling a cold land. She had a
    System.out.println(getWords(text));
}
```

solution1:

```
public static ArrayList<String> getWords(String text) {
    ArrayList<String> words = new ArrayList<>();
    String[] split = text.split("[^a-zA-Z]+");
    for (String word : split) {
        if (!words.contains(word)) {
            words.add(word);
        }
    }
    // collections.sort, is case sensitive, so it will sort A-Z, a-z
    Collections.sort(words);
    return words;
}
```

solution2:

```
public static ArrayList<String> getWords(String text) {
    // Split the text into words
    String[] words = text.split("[^a-zA-Z]+");

    // Create a set to store unique words
    Set<String> uniqueWords = new HashSet<>();

    // Add each word to the set
    for (String word : words) {
        if (!word.isEmpty()) { // Ignore empty strings
            uniqueWords.add(word.toLowerCase());
        }
    }

    // Sort the words alphabetically
    ArrayList<String> sortedWords = new ArrayList<>(uniqueWords);
    Collections.sort(sortedWords);

    return sortedWords;
}
```

- Write a Java method that takes an ArrayList of integers as input and returns a new ArrayList that contains the minimum number of swaps required to sort the input list in non-decreasing order.

Main method:

```
public static void main(String[] args) {
    ArrayList<Integer> arr = new ArrayList<>();
    arr.add(4);
    arr.add(3);
    arr.add(1);
    arr.add(2);
    arr.add(5);

    ArrayList<Integer> swapCounts = minimumSwaps(arr);

    System.out.println(swapCounts);
}
```

solution1:

```
public static ArrayList<Integer> minimumSwaps(ArrayList<Integer> arr) {
    ArrayList<Integer> sortedArr = new ArrayList<>(arr);
    Collections.sort(sortedArr);
    Map<Integer, Integer> valueToIndexMap = new HashMap<>();
    for (int i = 0; i < arr.size(); i++) {
        valueToIndexMap.put(arr.get(i), i);
    }

    ArrayList<Integer> swapCounts = new ArrayList<>();
    boolean[] visited = new boolean[arr.size()];

    for (int i = 0; i < arr.size(); i++) {
        if (visited[i] || valueToIndexMap.get(sortedArr.get(i)) == i) {
            continue;
        }

        int j = i;
        int cycleSize = 0;
        while (!visited[j]) {
            visited[j] = true;
            j = valueToIndexMap.get(sortedArr.get(j));
            cycleSize++;
        }

        if (cycleSize > 0) {
            swapCounts.add(cycleSize - 1);
        }
    }

    return swapCounts;
}
```

This method uses an approach that is based on graph theory to find the minimum number of swaps required to sort the input list. The basic idea is to treat the input list as a graph, where each element in the list is a node, and each swap is an edge between two nodes. We can then use graph algorithms to find the minimum number of swaps required to sort the graph.

The implementation first creates a sorted version of the input list, and a map that maps each value in the input list to its index in the list. It then loops through the input list and finds the cycles in the graph. For each cycle, it computes the size of the cycle and adds the size minus one (the number of swaps required to sort the cycle) to an ArrayList of swap counts.

Finally, the method returns the ArrayList of swap counts, which contains the minimum number of swaps required to sort the input list in non-decreasing order.