2-D Ragged Structures using ArrayLists and Arrays

You will implement the classes **`AlphaList`**V1 and **`AlphaListV2`**. You should be able to do all of **`AlphaList`**V1 on Monday, and start **`AlphaListV2`** as well. Wednesday will be to complete **`AlphaListV2`** and discuss the pros and cons of the two approaches.

An **`AlphaListV1`** object contains a two-dimensional ArrayList of ArrayList<**`String>`** references, called **`list`** (which is public just to make the JUnit cases for the submit server easier for me to write so please leave this variable public).

An **`AlphaListV2`** object contains a two-dimensional array of arrays of **`String`** references, called **`list`** (which is also public to make the JUnit cases for the submit server easier for me to write so please leave this variable public too).

Each class will be used to store a two-dimensional ragged structure of **`String`** references. The different will obviously be in the data types used behind the scenes and your interactions with them.

In both versions, there will be exactly 26 rows to the structure. **The strings that are added to this structure will contain only capital letters.** The first row will contain only strings that start with 'A'. The second row will contain only Strings that start with 'B', etc.

1. Write a constructor that takes no arguments. The constructor for V1 instantiates the two-dimensional structure so it has 26 "rows" and starts off which each row pointing at an empty ArrayList<String>. The constructor for V2 instantiates the two-dimensional ragged array so that it has 26 "rows" and starts off which each row pointing to a **`String`** array of size 0.
2. Write a **`public void`** method called **`insert`** that takes one argument, a **`String`**. The insert method will insert this parameter into the correct row of the structure, at the end of that row. To do this, you'll need to increase the size of that row by one. You can use the **`charAt`** method to determine the first character of the **`String`** and if you explicitly cast a **`char`** into an **`int`**, it will yield its ASCII value. If you subtract the ASCII value for 'A' from the ASCII value of that first character, you'll know which row should have the **`String`** added to it. Note that the difference between V1 and V2 will be the work involved in doing the increase in row size.
3. Write a **`public`** method called **`count`** that returns an **`int`**. This method will return the total number of strings that are stored in the structure.
4. Write a **`public void`** method called **`remove`** that takes one argument, a **`String`**. This method will remove the parameter from the row of the structure where it's found (and you'll need to decrease the size of that row by one). **You may assume that you will only be asked to remove a `String` that is already contained in the structure.** Again, the biggest difference between V1 and V2 will be the work involved in doing the increase in row size.