2023-01-09

(This is a ISO8601 formatted date. This is the best format because it sorts well!)

7.3: Traversing `ArrayLists` with Loops

Traversing `ArrayLists`

- `ArrayLists` support the same mechanisms you used when traversing Arrays - `while`, `for`, `for-each` - with the following differences

Operation	**`Array`**	**`ArrayList`**

length/size	`Array.length`
	`(property)`
		`ArrayList.size()`
		`(method)`

read	`value = array[index];`	`value = arrayList.get(index);`
write	`array[index] = value;`	`arrayList.set(index, value);`

Traversing ArrayLists

Array - for loop

```
Integer[] array = {1, 2, 3, 4, 5};
for (int idx = 0 ; idx <  array.length ; idx++) {
  int value = array[idx];
  System.out.println(value);
  array[idx] = value + 1;
}
```

ArrayList - for loop

```
Integer[] array = {1, 2, 3, 4, 5};
ArrayList<Integer> arrayList = new ArrayList<Integer>(Arrays.asList(array));
for (int idx = 0 ; idx <  arrayList.size() ; idx++) {
  int value = arrayList.get(idx);
  System.out.println(value);
  arrayList.set(idx, value + 1);
}
```

Traversing ArrayLists

Array & ArrayList - for loop **ArrayIndexOutOfBoundsException**
This exception will be thrown if you try to access the item at an index less than 0 or greater than the number of items in the Array or ArrayList

Traversing `ArrayLists`

Array - for-each loop

```java
Integer[] array = {1, 2, 3, 4, 5};
for (Integer value : array) {
  System.out.println(value);
}
```

ArrayList - for-each loop

```java
Integer[] array = {1, 2, 3, 4, 5};
ArrayList<Integer> arrayList = new ArrayList<Integer>(Arrays.asList(array));
for (Integer value : arrayList) {
  System.out.println(value);
}
```

Traversing `ArrayLists`

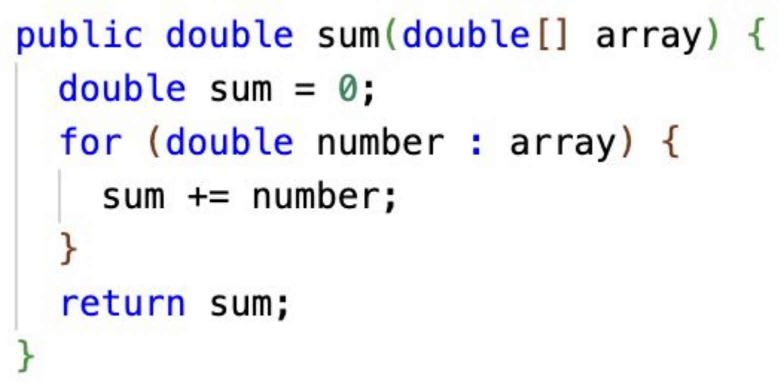
ArrayList for-each loop **ConcurrentModificationException**
This exception will be thrown if you try to add or remove items from an ArrayList while traversing that ArrayList with a for-each loop

7.4: ArrayList Algorithms

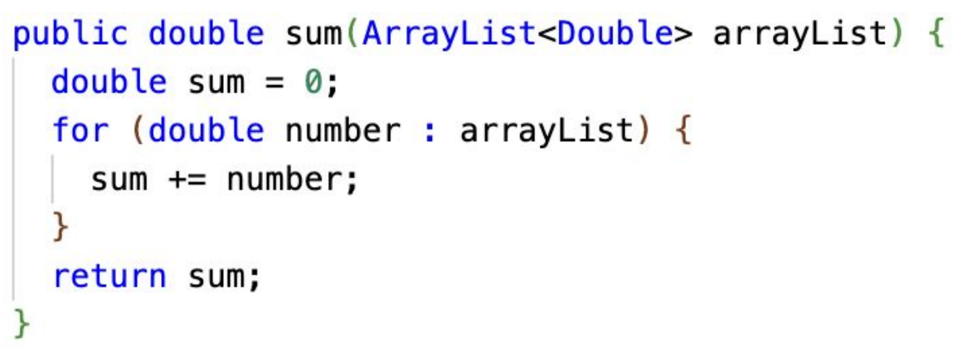
ArrayList Algorithms

A lot of the algorithms we learned for arrays can be applied directly to ArrayList.

Sometimes the code is almost exactly the same!



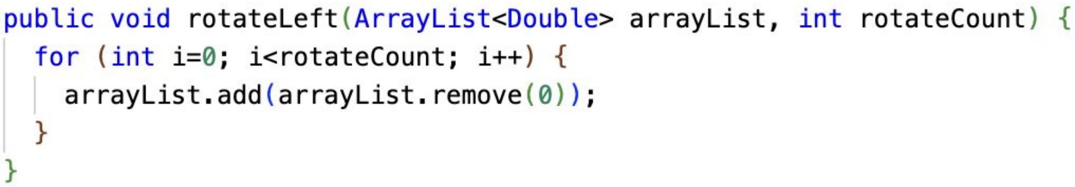
```java
public double sum(double[] array) {
    double sum = 0;
    for (double number : array) {
        sum += number;
    }
    return sum;
}
```



```java
public double sum(ArrayList<Double> arrayList) {
    double sum = 0;
    for (double number : arrayList) {
        sum += number;
    }
    return sum;
}
```

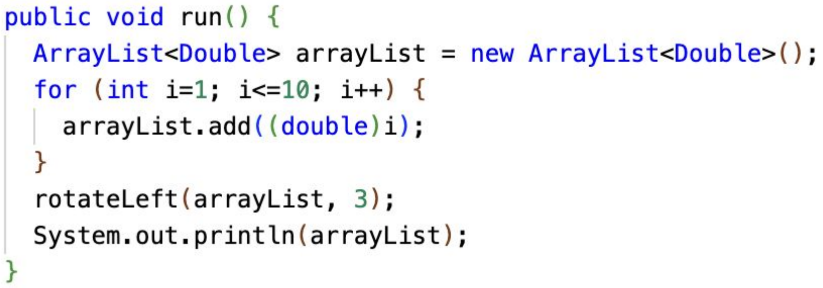
ArrayList Algorithms

Some things are easier with ArrayList because ArrayList has more built-in functionality. Remember rotating an array? Super easy with ArrayList.

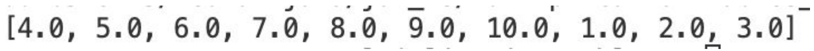


```
public void rotateLeft(ArrayList<Double> arrayList, int rotateCount) {
  for (int i=0; i<rotateCount; i++) {
    arrayList.add(arrayList.remove(0));
  }
}
```

But some things are harder... no equivalent to Array literals, for one.



```
public void run() {
  ArrayList<Double> arrayList = new ArrayList<Double>();
  for (int i=1; i<=10; i++) {
    arrayList.add((double)i);
  }
  rotateLeft(arrayList, 3);
  System.out.println(arrayList);
}
```



```
[4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 1.0, 2.0, 3.0]
```

List vs ArrayList

Sometimes you may see code like List<Double> instead of ArrayList<Double>. What's the difference?

ArrayList is part of Java's Collections library. Collections are grouped into kinds: Lists, Maps, Sets, etc. ArrayList is a **subclass** of List. But there are other kinds of List. For instance, Arrays.asList returns a List, but it's NOT an ArrayList, it's a fixed-size list:

```
import java.utils.List; import java.utils.Arrays;
...
 public void run() {
   List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6);
   System.out.println(list);
   list.add(7); // <- Blows up with UnsupportedOperationException
 }
```

Array vs ArrayList: How to choose?

When solving a problem, you may need to choose Array or ArrayList. ArrayList has more power, but it's built on arrays, so it adds some overhead.

Array	ArrayList
● I know exactly how many "slots" I need up front. ● I want to use the slots without having to "add" them all first. ● I want to use Array literals or other features not available on ArrayList. ● I don't need the extra functionality offered by ArrayList for this use case. ● I need every last bit of performance. ○ Avoid method call overhead ○ Avoid wrapper type overhead	● The number of slots I need is not known up front. ● The number of slots I need may grow and shrink over time. ● I need to insert or remove elements or use other methods ArrayList has not available on Arrays. ● I want to pass this data to some code that expects a Java Collection.

CSAwesome 7.4.2: ClimbingClub

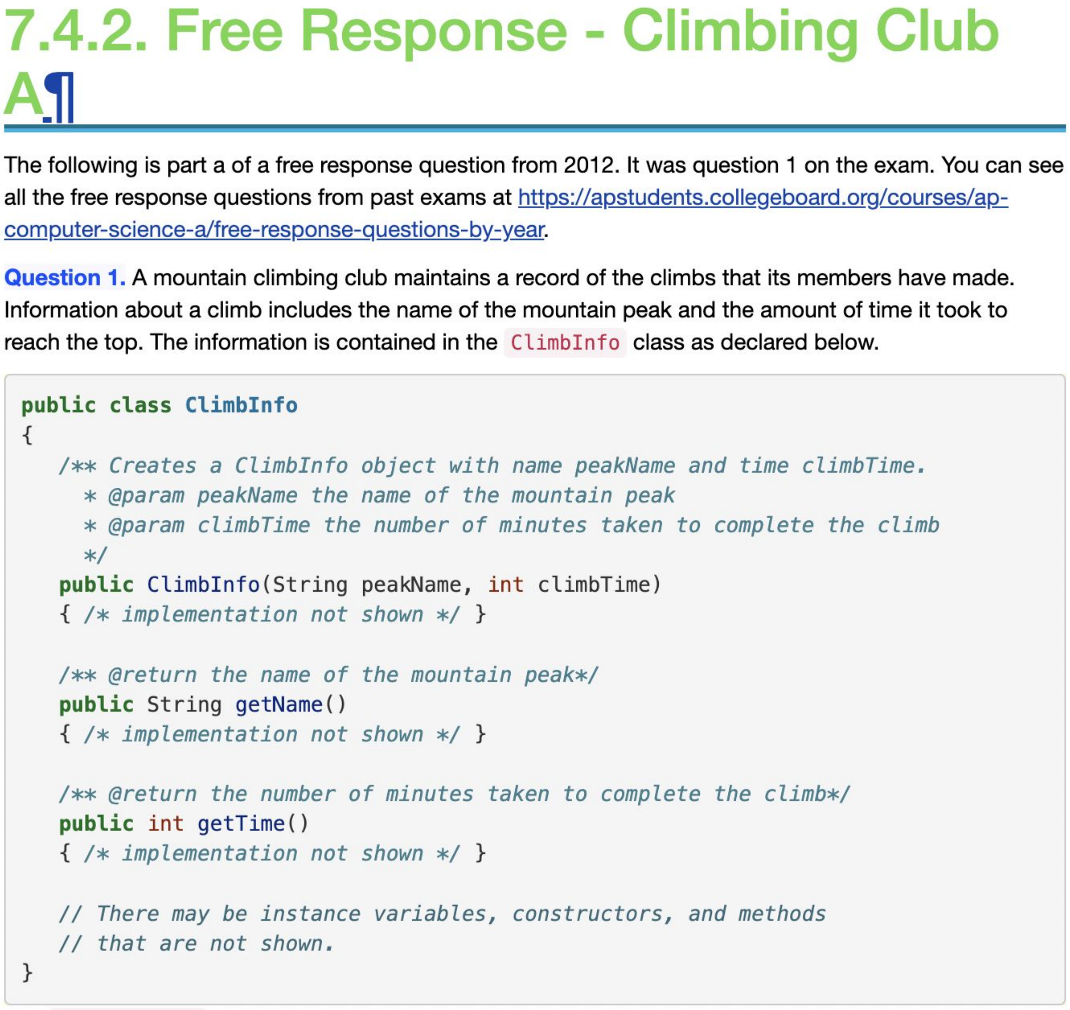
Much of CSAwesome 7.4 is practice FRQs for the AP Exam. These are good to work through!

Let's work through one of them together. Write your own code, then we'll show a solution and discuss.

However, let's do it using this Replit so that you can get the exercise of writing the code in part (c), not just checking the code that they present.

Replit: ClimbingClub

The solutions are in this slide deck but don't peek! Work through it yourself!



7.4.2. Free Response - Climbing Club A¶

The following is part a of a free response question from 2012. It was question 1 on the exam. You can see all the free response questions from past exams at https://apstudents.collegeboard.org/courses/ap-computer-science-a/free-response-questions-by-year.

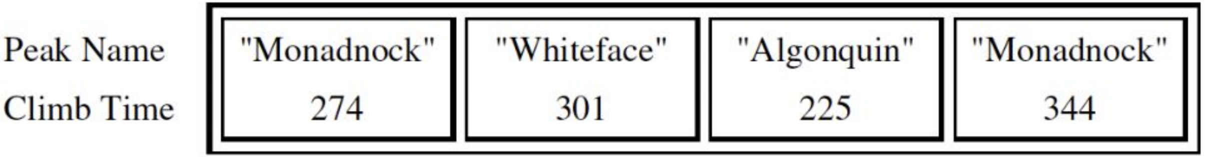
Question 1. A mountain climbing club maintains a record of the climbs that its members have made. Information about a climb includes the name of the mountain peak and the amount of time it took to reach the top. The information is contained in the `ClimbInfo` class as declared below.

```
public class ClimbInfo
{
   /** Creates a ClimbInfo object with name peakName and time climbTime.
     * @param peakName the name of the mountain peak
     * @param climbTime the number of minutes taken to complete the climb
     */
   public ClimbInfo(String peakName, int climbTime)
   { /* implementation not shown */ }

   /** @return the name of the mountain peak*/
   public String getName()
   { /* implementation not shown */ }

   /** @return the number of minutes taken to complete the climb*/
   public int getTime()
   { /* implementation not shown */ }

   // There may be instance variables, constructors, and methods
   // that are not shown.
}
```

Part a. Write an implementation of the `ClimbingClub` method `addClimb` that stores the `ClimbInfo` objects in the order they were added. This implementation of `addClimb` should create a new `ClimbInfo` object with the given name and time. It appends a reference to that object to the end of climbList. For example, consider the following code segment.

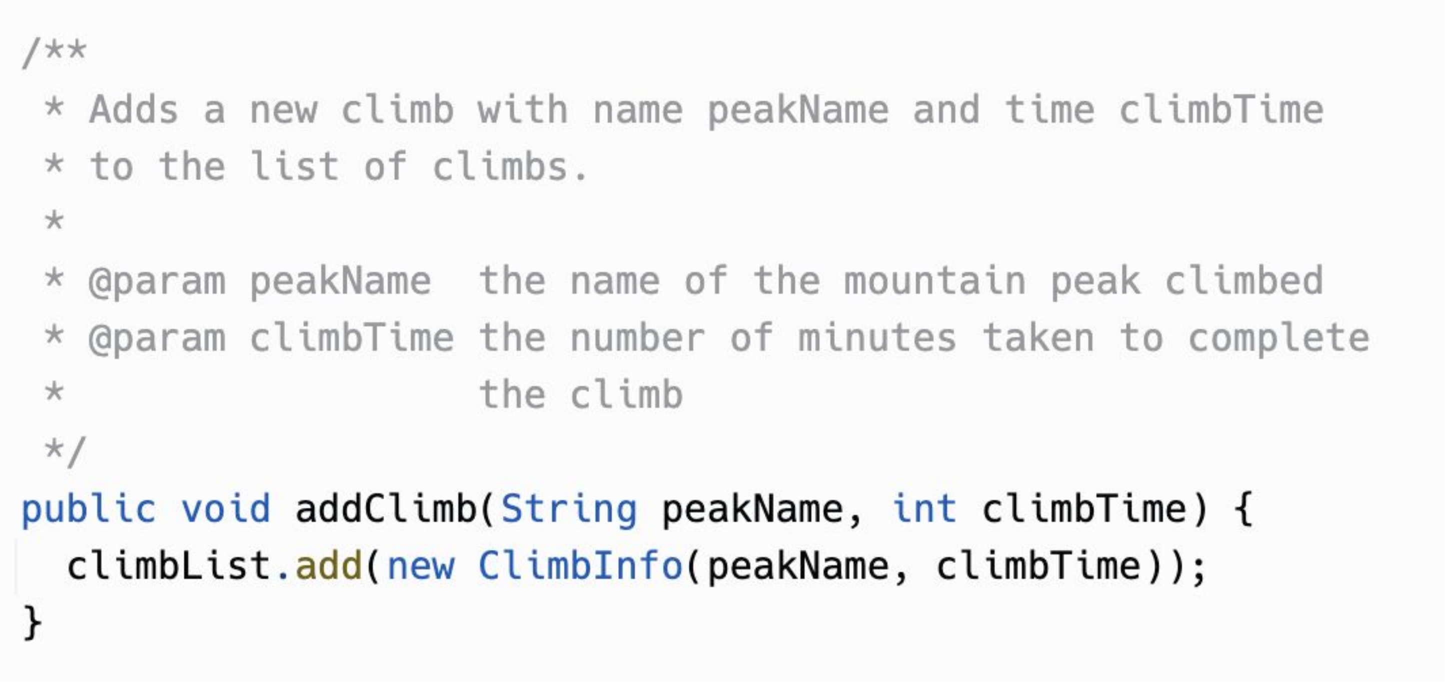
```
ClimbingClub hikerClub = new ClimbingClub();
hikerClub.addClimb("Monadnock", 274);
hikerClub.addClimb("Whiteface", 301);
hikerClub.addClimb("Algonquin", 225);
hikerClub.addClimb("Monadnock", 344);
```

When the code segment has completed executing, the instance variable `climbList` would contain the following entries.

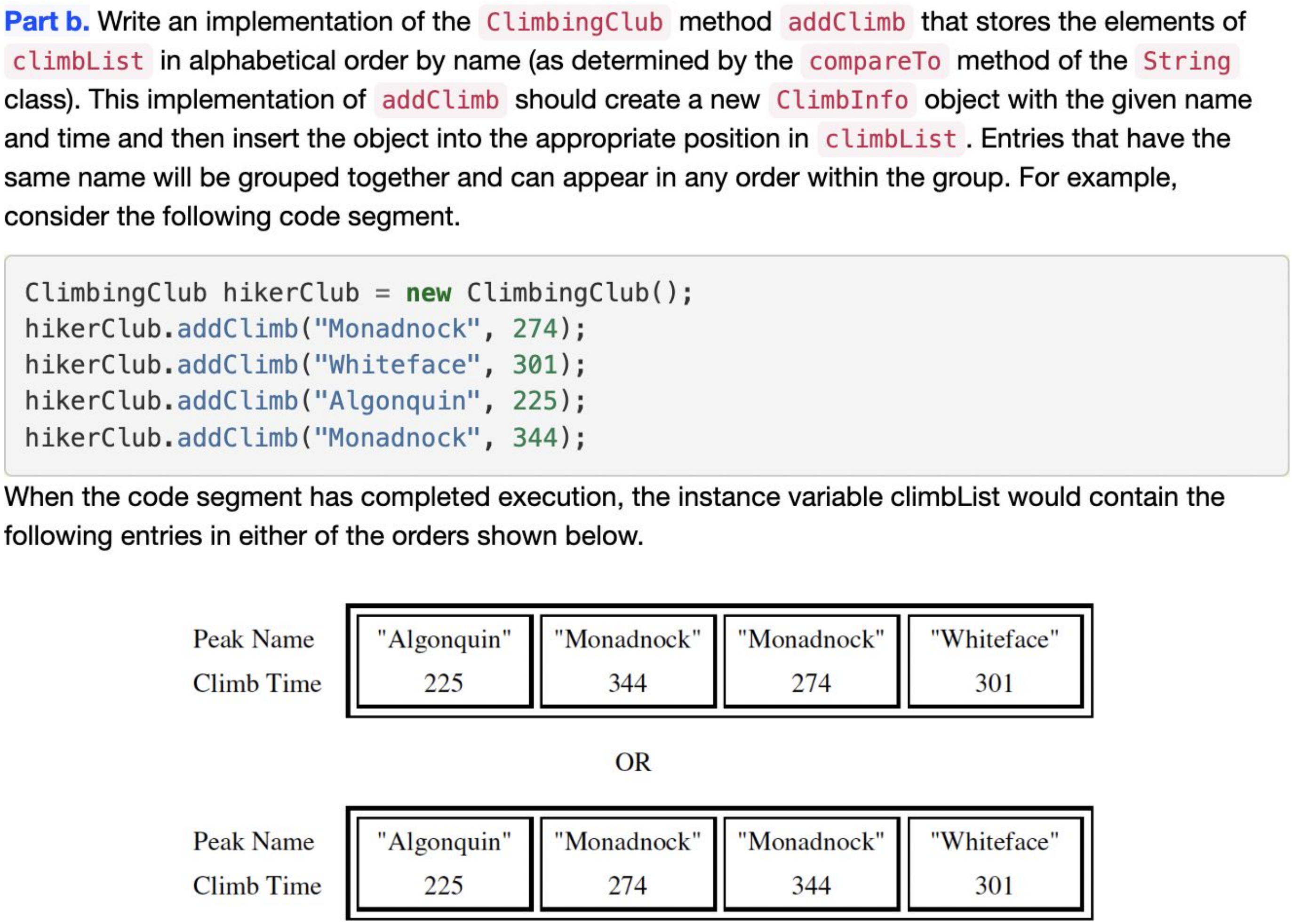


Peak Name	"Monadnock"	"Whiteface"	"Algonquin"	"Monadnock"
Climb Time	274	301	225	344

Part (a) solution



```java
/**
 * Adds a new climb with name peakName and time climbTime
 * to the list of climbs.
 *
 * @param peakName  the name of the mountain peak climbed
 * @param climbTime the number of minutes taken to complete
 *                  the climb
 */
public void addClimb(String peakName, int climbTime) {
  climbList.add(new ClimbInfo(peakName, climbTime));
}
```



Part b. Write an implementation of the `ClimbingClub` method `addClimb` that stores the elements of `climbList` in alphabetical order by name (as determined by the `compareTo` method of the `String` class). This implementation of `addClimb` should create a new `ClimbInfo` object with the given name and time and then insert the object into the appropriate position in `climbList`. Entries that have the same name will be grouped together and can appear in any order within the group. For example, consider the following code segment.

```
ClimbingClub hikerClub = new ClimbingClub();
hikerClub.addClimb("Monadnock", 274);
hikerClub.addClimb("Whiteface", 301);
hikerClub.addClimb("Algonquin", 225);
hikerClub.addClimb("Monadnock", 344);
```

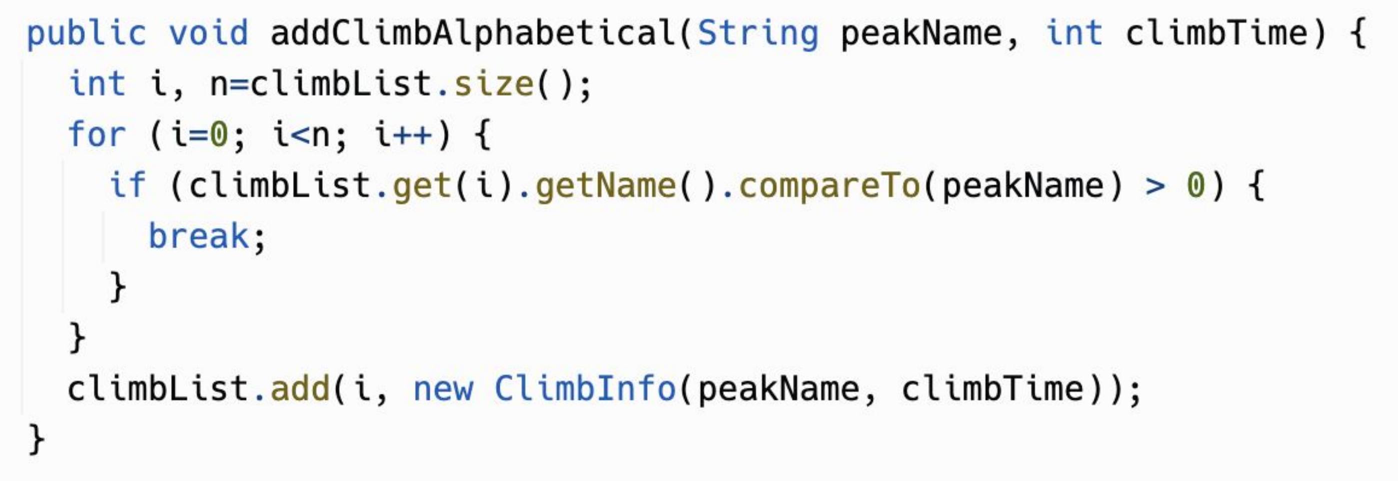
When the code segment has completed execution, the instance variable climbList would contain the following entries in either of the orders shown below.

Peak Name	"Algonquin"	"Monadnock"	"Monadnock"	"Whiteface"
Climb Time	225	344	274	301

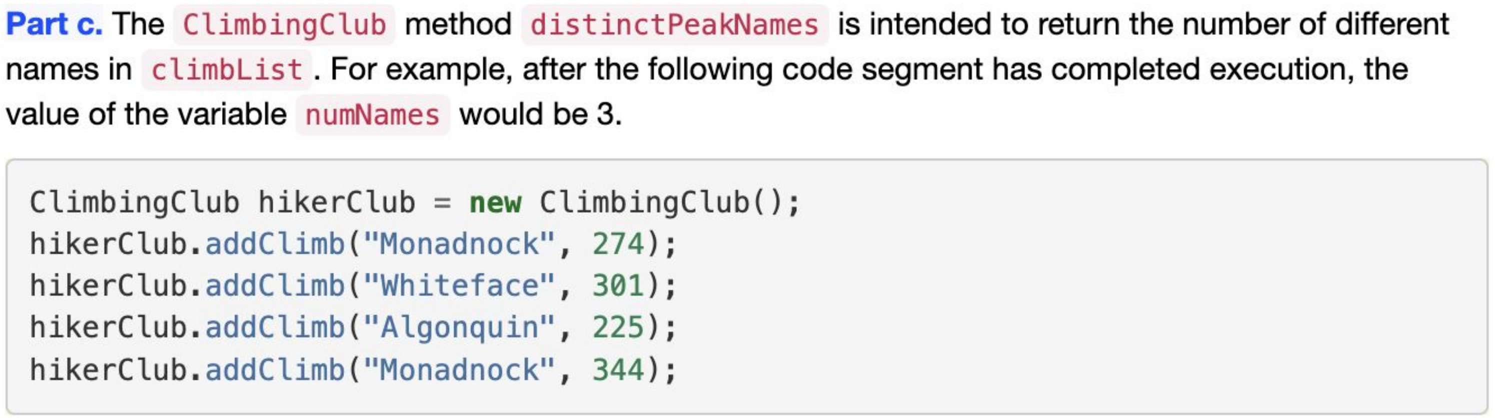
OR

Peak Name	"Algonquin"	"Monadnock"	"Monadnock"	"Whiteface"
Climb Time	225	274	344	301

Part (b) solution



```java
public void addClimbAlphabetical(String peakName, int climbTime) {
  int i, n=climbList.size();
  for (i=0; i<n; i++) {
    if (climbList.get(i).getName().compareTo(peakName) > 0) {
      break;
    }
  }
  climbList.add(i, new ClimbInfo(peakName, climbTime));
}
```

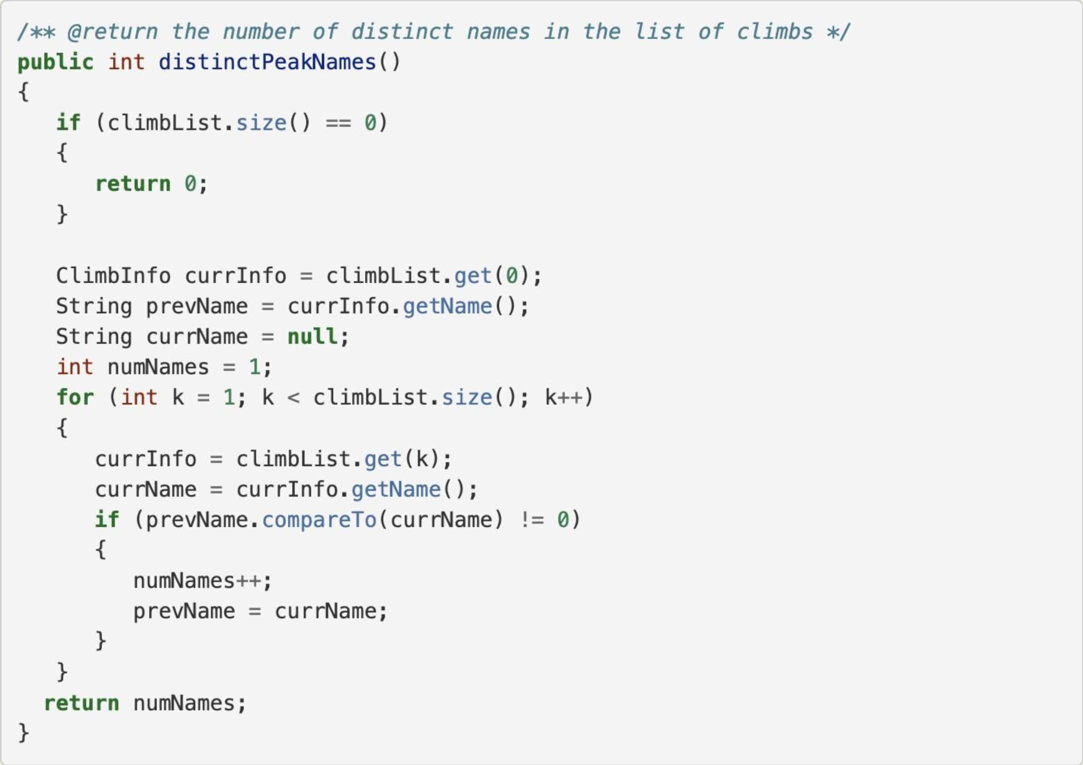


Part c. The `ClimbingClub` method `distinctPeakNames` is intended to return the number of different names in `climbList`. For example, after the following code segment has completed execution, the value of the variable `numNames` would be 3.

```
ClimbingClub hikerClub = new ClimbingClub();
hikerClub.addClimb("Monadnock", 274);
hikerClub.addClimb("Whiteface", 301);
hikerClub.addClimb("Algonquin", 225);
hikerClub.addClimb("Monadnock", 344);
```

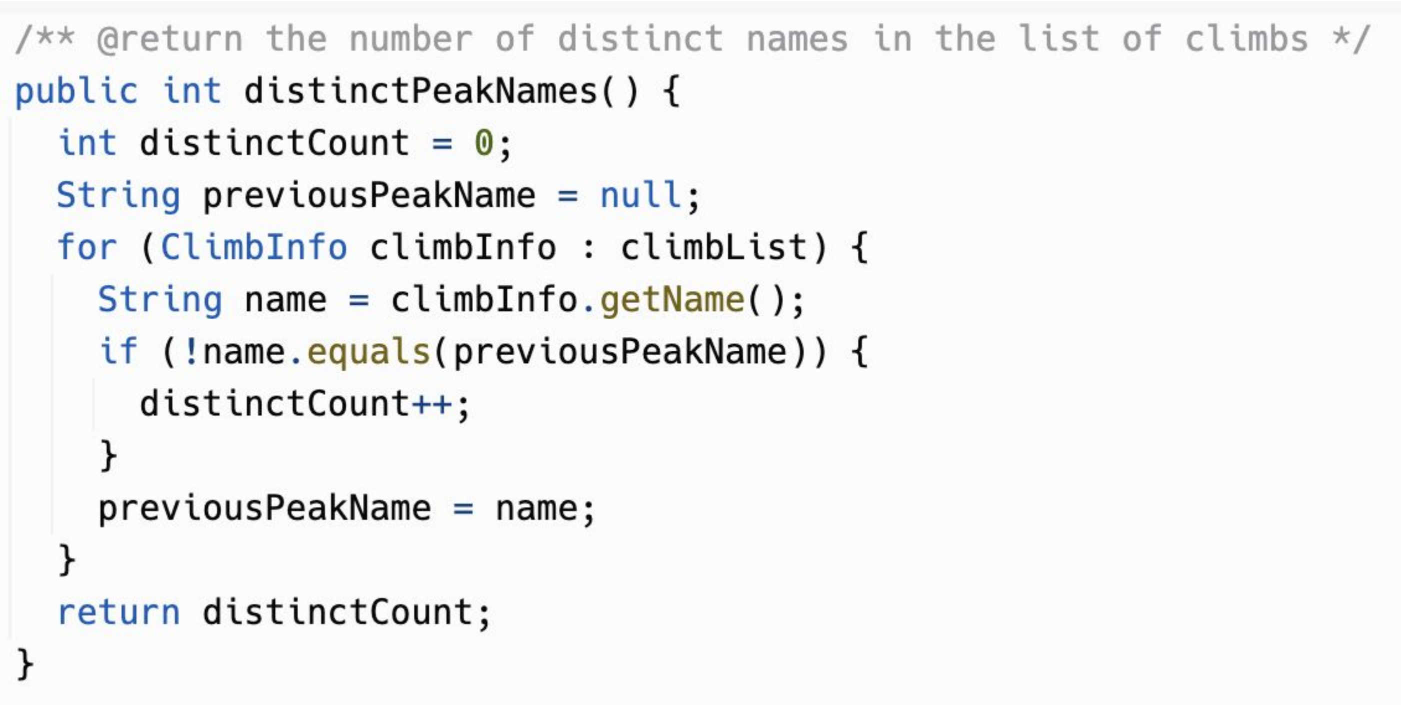
The AP Exam FRQ actually GIVES you a pre-written method for this, and asks you some questions about it. Instead, let's try to write the method. Think about the ArrayList algorithms we learned back in Section 6.4.

Part (c) solution from the AP exam



```java
/** @return the number of distinct names in the list of climbs */
public int distinctPeakNames()
{
   if (climbList.size() == 0)
   {
      return 0;
   }

   ClimbInfo currInfo = climbList.get(0);
   String prevName = currInfo.getName();
   String currName = null;
   int numNames = 1;
   for (int k = 1; k < climbList.size(); k++)
   {
      currInfo = climbList.get(k);
      currName = currInfo.getName();
      if (prevName.compareTo(currName) != 0)
      {
         numNames++;
         prevName = currName;
      }
   }
  return numNames;
}
```

Part (c) solution



```
/** @return the number of distinct names in the list of climbs */
public int distinctPeakNames() {
  int distinctCount = 0;
  String previousPeakName = null;
  for (ClimbInfo climbInfo : climbList) {
    String name = climbInfo.getName();
    if (!name.equals(previousPeakName)) {
      distinctCount++;
    }
    previousPeakName = name;
  }
  return distinctCount;
}
```

Practice on your own

There are several other FRQs in CSAwesome 7.4.

With the remaining time, start working through one of the others.

These FRQs are directly from past AP Exams, so they're good practice, but they're also good applied programming practice in general.