Quiz1

Tuesday, October 2, 2018 3:26 PM

CSE143 Section #2 Problems

For problems 1-3, you are writing a method for the ArrayIntList class:

```
public class ArrayIntList {
    private int[] elementData;
    private int size;

    <methods>
}
```

Unless otherwise noted, assume that you may not call any other methods of the class in solving the problem.

1. Write a method isPairwiseSorted that returns whether or not a list of integers is pairwise sorted (true if it is, false otherwise). A list is considered pairwise sorted if each successive pair of numbers is in sorted (non-decreasing) order. For example, if a variable called list stores the following sequence of values:

   [3, 8, 2, 5, 19, 24, -3, 0, 4, 4, 8, 205, 42]

   then the following call:

   list.isPairwiseSorted()

   should return the value true because the successive pairs of this list are all sorted: (3, 8), (2, 5), (19, 24), (-3, 0), (4, 4), (8, 205). Notice that the extra value 42 at the end had no effect on the result because it is not part of a pair. If the list had instead stored the following:

   [1, 9, 3, 17, 4, 28, -5, -3, 0, 42, 308, 409, 19, 17, 2, 4]

   then the method should return the value false because the pair [19, 17] is not in sorted order. If a list is so short that it has no pairs, then it is considered to be pairwise sorted.

2. Write a method called mirror that doubles the size of a list of integers by appending the mirror image of the original sequence to the end of the list. The mirror image is the same sequence of values in reverse order. For example, if a variable called list stores this sequence of values:

   [1, 3, 2, 7]

   and we make the following call:

   list.mirror();

   then it should store the following values after the call:

   [1, 3, 2, 7, 7, 2, 3, 1]

   Notice that it has been doubled in size by having the original sequence appearing in reverse order at the end of the list. You may not make assumptions about how many elements are in the list although you may assume that the array has sufficient capacity to store the new list.

3. Write a method called fromCounts that converts an ArrayIntList of counts into a new ArrayIntList of values that the method returns. Assume that the ArrayIntList that is called stores a sequence of integer pairs that each indicate a count and a number. For example, suppose that an ArrayIntList called list stores the following sequence of values:

   [5, 2, 2, -5, 4, 3, 2, 4, 1, 1, 1, 0, 2, 17]

   This sequence of pairs indicates that you have 5 occurrences of 2, followed by two occurrences of -5, followed by 4 occurrences of 3, and so on. If we make the following call:

   ```
   ArrayIntList list2 = list.fromCounts();
   ```

   Then the variable list2 should store the following sequence of values:

   [2, 2, 2, 2, 2, -5, -5, 3, 3, 3, 3, 4, 4, 1, 0, 17, 17]

   Assume that the ArrayIntList that is called stores a legal sequence of pairs (which means it will always have an even size) and that the default constructor for ArrayIntList will construct an array of sufficient capacity to store the result. Your method should not change the original list. If the sequence of pairs is empty, the result should be an empty list.

4. Below is a "bad" version of the ArrayIntList. It has mostly the same functionality as the version discussed in lecture, but it has poor style.

to store the result. Your method should not change the original list. If the sequence of pairs is empty, the result should be an empty list.

4. Below is a "bad" version of the ArrayIntList. It has mostly the same functionality as the version discussed in lecture, but it has poor style. Approximately half of the points for each programming assignment in this class will be devoted to style issues, so it is important to understand style issues. What is bad about this version?

```
// Stuart Reges (my name is Elroy Jetson)
// This is the ArrayIntList class.

import java.util.*;

public class ArrayIntList {
    int[] elementData; // element data
    int size;          // size
    int capacity;      // capacity
    public static int defaultCapacity = 100;

    public ArrayIntList() {
        elementData = new int[100];
        size = 0;
        capacity = 100;
    }

    // capacity should be not be negative
    public ArrayIntList(int capacity) {
        if (capacity < 0) {
            throw new IllegalArgumentException();
        } else {
            elementData = new int[capacity];
            size = 0;
            this.capacity = capacity;
        }
    }

    public int size() {
        return size;
    }
```

useless comment
(useless capacity)
private static final.
DEFAULT_CAPACITY
dc.

```
    // pre : 0 <= index < size() (throws exception if not)
    public int get(int index) {
        if (index < 0 || index >= size) {
            throw new IndexOutOfBoundsException();
        }
        for (int i = 0; i < size; i++) {
            if (i == index) {
                return elementData[i];
            }
        }
        return 0;
    }

    public String toString() {
        if(size==0) {
            return"[]";
        } else {
            String result="["+elementData[0];
            for(int i=1;i<size;i++) {
                result+=", "+elementData[i];
            }
            return result+"]";
        }
    }

    // uses a for loop to find out if the array has the given value
    public boolean contains(int value) {
        int count = 0;
        for (int i = size - 1; i >= 0; i--) {
            if (elementData[i] == value) {
                count++;
            }
        }
        if (count == 0) {
            return false;
        } else {
            return true;
        }
    }

    // returns the position of the given value in the array, only checking up
    // to the current size
    public int indexOf(int value) {
        int index = -1;
        for (int i = size - 1; i >= 0; i--) {
            if (elementData[i] == value) {
                index = i;
            }
        }
        if (elementData[index] == value) {
            return index;
        } else {
            return -1;
        }
    }
```

No need to count.

boolean zen

return when find a first 1.

return index;

```
    // pre : size() < capacity (throws IllegalStateException if not)
    // post: appends the value to the end of the list
    public void add(int value) {
        if (size > capacity - 1) {
            throw new IllegalStateException();
        }
        elementData[size] = value;
        size++;
    }

    // pre : size() < capacity (throws IllegalStateException if not) &&
    //       0 <= index <= size() (throws IndexOutOfBoundsException if not)
    // post: inserts the value at the index
    public void add(int index, int value) {
        for (int i = size; i >= index; i--) {
            if (index < 0 || index > size) {
                throw new IndexOutOfBoundsException();
            } else if (size > capacity - 1) {
                throw new IllegalStateException();
            } else if (i > index) {
                elementData[i] = elementData[i - 1];
            } else {
                elementData[i] = value;
                size++;
            }
        }
    }

    // post: returns the capacity of the list
    public int capacity() {
        return capacity;
    }

    // pre : 0 <= index < size() (throws IndexOutOfBoundsException if not)
    // post: removes value at the index and decreases the size by 1
    public void remove(int index) {
        if (index < 0 || index >= size) {
            throw new IndexOutOfBoundsException();
        }
        for (int i = index; i < size - 1; i++) {
            elementData[i] = elementData[i + 1];
        }
        size--;
    }

    // appends values to the end of the list
    public void addAll(ArrayIntList other) {
        int[] data = Arrays.copyOf(elementData, size + other.size);
        while (other.size() > 0) {
            data[size++] = other.elementData[0];
            other.remove(0);
        }
        elementData = data;
    }
}
```

- In Comment:

Behavior.

-Exceptions.
  - when it's thrown.
  - name included

-Returns.

-Parameters

Solution to CSE143 Section #2 Problems

1. One possible solution appears below.

```
public boolean isPairwiseSorted() {
    for (int i = 0; i < size - 1; i += 2) {
        if (elementData[i] > elementData[i + 1]) {
            return false;
        }
    }
    return true;
}
```

2. One possible solution appears below.

```
public void mirror() {
    int last = 2 * size - 1;
    for (int i = 0; i < size; i++) {
        elementData[last - i] = elementData[i];
    }
    size *= 2;
}
```

3. One possible solution appears below.

```
public ArrayIntList fromCounts() {
    ArrayIntList result = new ArrayIntList();
    int size2 = 0;
    for (int i = 0; i < size; i += 2) {
        for (int j = 0; j < elementData[i]; j++) {
            result.elementData[size2] = elementData[i + 1];
            size2++;
        }
    }
    result.size = size2;
    return result;
}
```

4. Below is a list of style problems with the bad ArrayIntList:

class comment: Don't include Stuart's name, include just your name. It would also be helpful to include the date and your section or TA's name. The class comment is meaningless. Make some kind of attempt to describe what the class is used for, as in "Class ArrayIntList can be used to store a list of integers."

fields: Fields should be declared private. The field comments are useless because they are repeating the names of the fields. Only include comments if you can provide something beyond the field name. The field called "capacity" is not needed because it has the same value as elementData.length.

class constant: It is improperly declared because it is missing "final". It is also improperly named because the convention for constants is to use all uppercase letters and underscore characters, as in DEFAULT_CAPACITY.

first constructor: It should have a comment and it should be using the "this(...)" notation to call the other constructor. It does not use the class constant as it should.

second constructor: Comment doesn't mention the fact that it throws an IllegalArgumentException when capacity is negative and the comment should describe more about what it does. The use of if/else is not appropriate. The convention in Java is to throw exceptions with an if and then to have the standard code follow without being inside an else.

size method: It has no comment.

get method: Comment doesn't mention what kind of exception is thrown and doesn't describe what it does. The for loop is not needed and makes the method extremely inefficient, basically negating the benefit of using an array (the random-access aspect of the array).

toString method: No comment. Spacing is terrible. Introduce spaces to make your code more readable. The indentation is also off on many lines.

contains method: The comment has implementation details, discussing the use of a for loop and the fact that it is searching an array. This method is also highly redundant. It should call indexOf. The if/else after the loop violates boolean zen (can simply return (count > 0)).

indexOf method: The comment has implementation details, talking about the array and the size fields. It also does not describe significant behavior: the fact that the first occurrence is returned and that a -1 is returned if not found. The implementation is horrible. It uses a variable called index that is not needed and then there is a redundant test after the loop. Even if you are going to use this index variable, then initialize it to -1 and return it after the loop instead of having the same test both inside and outside the loop.

first add method: This is a good method.

second add method: The exception comments are good, but the description of what it does is incomplete. It doesn't say what happens to the old value at the given index. It shifts subsequent values to the right, which should be described in the comment so that the client knows what it does. Also, the loop structure is very bad. Code that is executed once either before or after the repeated operation of a loop should appear outside the loop. In particular, The exception checking should occur before the loop and the "else" part that stores the value and increments size should appear after the loop. Because the exception testing is inside the loop, it never throws an exception for index values greater than the size of the list.

capacity method: You are not allowed to add extra public functions to a class that weren't part of the specification. You can add private methods that are part of the implementation, but not public methods.

remove method: The comment shouldn't discuss the implementation detail of decreasing the size and it should mention what happens to the list when the value is removed. The subsequent values are shifted to the left.

addAll method: The comment does not mention the parameter (i.e., that the values being appended come from the other ArrayIntList). It constructs a new array, which is not necessary, and the new array has a different capacity than the original. It also destroys the data in the other list.

overall: The lines of code to check for illegal indexes in get and remove are redundant. It would have been good to introduce a private method to eliminate the redundancy.