CSE143X Section #5 Problems

For all problems involving lists and sets, the contents will be displayed using the standard square bracket notation used by toString, as in:

```
[12, 4, 19, 78, 42]
```

List<E> Methods (10.1)

add(value)	appends value at end of list
add(index, value)	inserts given value at given index, shifting subsequent values right
clear()	removes all elements of the list
indexOf(value)	returns first index where given value is found in list (-1 if not found)
get(index)	returns the value at given index
remove(index)	removes/returns value at given index, shifting subsequent values left
set(index, value)	replaces value at given index with given value
size()	returns the number of elements in list
addAll(list)	adds all elements from the given collection to the end of the list
contains(value)	returns true if the given value is found somewhere in this list
remove(value)	finds and removes the given value from this list
removeAll(list)	removes any elements found in the given collection from this list
iterator()	returns an object used to examine the contents of the list

Set<E> Methods (11.2)

add(value)	adds the given value to the set
contains(value)	returns true if the given value is found in the set
remove(value)	removes the given value from the set
clear()	removes all elements of the set
size()	returns the number of elements in the set
isEmpty()	returns true if the set's size is 0
addAll(collection)	adds all elements from the given collection to the set
containsAll(collection)	returns true if set contains every element from given collection
removeAll(collection)	removes any elements found in the given collection from this set
retainAll(collection)	removes any elements not found in the given collection from this set
iterator()	returns an object used to examine contents of the set

Iterator<E> Methods (11.1)

hasNext()	returns true if there are more elements to be read from collection
next()	reads and returns the next element from the collection
remove()	removes the last element returned by next from the collection

1. Write a method called acronymFor that takes a list of strings as a parameter and that returns the corresponding acronym. You form an acronym by combining the capitalized first letter of a series of words. For example, the list [laughing, out, loud] produces the acronym "LOL". The list [Computer, Science and, Engineering] produces the acronym "CSE". You may assume that all of the strings are nonempty. Your method is not allowed to change the list passed to it as a parameter. If passed an empty list, your method should return the empty string. You may construct iterators and strings, but you are not allowed to construct other structured objects.

2. Write a method called switchPairs that switches the order of values in a List of Strings in a pairwise fashion. Your method should switch the order of the first two values, then switch the order of the next two, switch the order of the next two, and so on. For example, if a list stores:

```
[four, score, and, seven, years, ago]
```

your method should switch the first pair (four, score), the second pair (and, seven) and the third pair (years, ago), to yield this list:

```
[score, four, seven, and, ago, years]
```

If there are an odd number of values in the list, the final element is not moved. For example, if the original list had been:

```
[to, be, or, not, to, be, hamlet]
```

It would again switch pairs of values, but the final value (hamlet) would not be moved, yielding this list:

```
[be, to, not, or, be, to, hamlet]
```

3. Write a method stutter that doubles the size of the list by replacing every integer in the list with two of that integer. For example, if the list stores the following sequence of integers when the method is called:

```
[1, 8, 19, 4, 17]
```

It should store the following sequence of integers after stutter is called:

```
[1, 1, 8, 8, 19, 19, 4, 4, 17, 17]
```

4. Write a method called reverse3 that takes a List of integer values as a parameter and that reverses each successive sequence of three values in the list. For example, suppose that a variable called list stores the following sequence of values:

```
[3, 8, 19, 42, 7, 26, 19, -8, 193, 204, 6, -4]
```

and we make the following call:

```
reverse3(list);
```

Afterwards the list should store the following sequence of values:

```
[19, 8, 3, 26, 7, 42, 193, -8, 19, -4, 6, 204]
```

The first sequence of three values (3, 8, 19) has been reversed to be (19, 8, 3). The second sequence of three values (42, 7, 26) has been reversed to be (26, 7, 42). And so on. If the list has extra values that are not part of a sequence of three, those values are unchanged. For example, if the list had instead stored:

```
[3, 8, 19, 42, 7, 26, 19, -8, 193, 204, 6, -4, 99, 2]
```

The result would have been:

```
[19, 8, 3, 26, 7, 42, 193, -8, 19, -4, 6, 204, 99, 2]
```

Notice that the values (99, 2) are unchanged in position because they were not part of a sequence of three values.

5. Write a method hasOdd that takes a set of integers as a parameter and that returns true if the set contains at least one odd integer, false otherwise.

6. Write a method removeEvens that takes a set of integers as a parameter and that removes the even values from the set, returning those values as a new set. The new set should be ordered in increasing numerical order. For example, if a set s1 contains these values:

```
[0, 17, 16, 7, 10, 12, 13, 14]
```

and we make the following call:

```
Set<Integer> s2 = removeEvens(s1);
```

Then after the call s1 and s2 would contain the following values:

```
s1: [17, 7, 13]
s2: [0, 10, 12, 14, 16]
```

7. Write a method containsAll that takes two sets of integers as parameters and that returns true if the first set contains all of the values of the second set and that returns false otherwise. For example, if the two sets are:

```
s1: [17, 16, 7, 10, 12, 13, 14]
s2: [7, 12, 13]
```

then the call containsAll(s1, s2) would return true while the call containsAll(s2, s1) would return false. You are implementing a two-argument alternative to the standard Set method called containsAll, so you are not allowed to call that method to solve this problem. You are also not allowed to construct any structured objects to solve the problem (no set, list, stack, queue, string, etc). Your method should not change either set passed as a parameter.

8. Write a method called equals that takes two sets of integers as parameters and that returns true if the sets are equal. Two sets are considered equal if they store the same values. For example, given sets: s1: [5, 3, 1, 0] s2: [0, 1, 5, 3] s3: [1, 0, 5, 3, 4] The call equals(s1, s2) would return true while the calls equals(s1, s3) and equals(s2, s3) would return false. As in the examples above, you can not assume that the set values are ordered.

   You are implementing a two-argument alternative to the standard Set method called equals, so you are not allowed to call that method or the containsAll method to solve this problem. You may construct iterator objects, but you are also not allowed to construct any structured objects to solve the problem (no set, list, stack, queue, string, etc). Your method should not change either of the sets passed as parameters.

9. Write a method called retainAll that takes two sets of integers as parameters and that removes any values in the first set that are not found in the second set. For example, given sets:

```
s1: [0, 19, 8, 9, 12, 13, 14, 15]
s2: [0, 19, 2, 4, 5, 9, 10, 11]
```

If the following call is made:

```
retainAll(s1, s2);
```

after the call, the sets would store the following values:

```
s1: [0, 19, 9]
s2: [0, 19, 2, 4, 5, 9, 10, 11]
```

You are implementing a two-argument alternative to the standard Set method called retainAll, so you are not allowed to call that method to solve this problem. You are also not allowed to construct any structured objects to solve the problem (no set, list, stack, queue, string, etc) although you can construct iterators. Your method should not change the second set passed as a parameter.