AP® COMPUTER SCIENCE AB
2008 SCORING GUIDELINES

Question 1: Anagram Set

Part A: constructor **4 points**

+1/2 `groups = new HashMap<String, HashSet<String>>();`

+1 traverse `words`
- **+1/2** correctly access an element of `words` (in context of loop)
- **+1/2** access all elements of `words` (lose this if index out-of-bounds)

+2 1/2 store anagram sets in `groups` (in context of loop)
- **+1/2** `createKeyString(`*accessedWord*`)`
- **+1** correct if keyString not already stored
- **+1** correct if keyString already stored

Part B: findLargestSets **5 points**

+1/2 construct a `HashSet<HashSet<String>>` object (must assign to a variable)

+1 1/2 iterate through anagram sets
- **+1** correctly access an anagram set (in context of loop)
- **+1/2** access all anagram sets (loop/iteration structure is correct)

+2 1/2 find largest sets
- **+1/2** determine size of anagram set
- **+1** update largest (in context of loop)
  - **+1/2** compare size with size of some previous set
  - **+1/2** add to set of sets if size >= largest so far
- **+1** correctly construct set of largest sets

+1/2 return set of largest sets

AP® COMPUTER SCIENCE AB
2008 CANONICAL SOLUTIONS

Question 1: Anagram Set

PART A:

```
public AnagramGrouper(List<String> words)
{
  groups = new HashMap<String, HashSet<String>>();

  for (String str : words)
  {
    String key = createKeyString(str);
    HashSet<String> wordSet = groups.get(key);
    if (wordSet == null)
    {
      wordSet = new HashSet<String>();
      groups.put(key, wordSet);
    }
    wordSet.add(str);
  }
}
```

PART B:

```
public HashSet<HashSet<String>> findLargestSets()
{
  HashSet<HashSet<String>> largest = new HashSet<HashSet<String>>();
  int largestSize = 0;

  for (String key : groups.keySet())
  {
    HashSet<String> anaSet = groups.get(key);
    if (anaSet.size() > largestSize)
    {
      largest = new HashSet<HashSet<String>>();
      largestSize = anaSet.size();
    }
    if (anaSet.size() == largestSize)
    {
      largest.add(anaSet);
    }
  }
  return largest;
}
```

AB1a,

(a) Write the `AnagramGrouper` constructor. The constructor takes a list of words and constructs a map in which each key is a key string and the associated value is the set of words that each have that key string. The map should contain all the anagram sets that are generated from the list of words.

Complete the `AnagramGrouper` constructor below.

```
/** Constructs a map from words in which the keys are key strings and the
 *  value associated with a key string is the set of anagrams having that key string.
 *  Postcondition: each entry of words is contained in an anagram set
 *  @param words a list of strings to be grouped into anagram sets
 *         Precondition: words.size() > 0
 */
public AnagramGrouper(List<String> words)
{
    groups = new HashMap<String, HashSet<String>>();
    for (String s : words)
    {
        String key = createKeyString(s);
        if (groups.containsKey(key))
            groups.get(key).add(s);
        else
        {
            groups.put(key, new HashSet<String>());
            groups.get(key).add(s);
        }
    }
}
```

GO ON TO THE NEXT PAGE.

AB1a2

(b) Write the `AnagramGrouper` method `findLargestSets`. This method analyzes the instance variable `groups` and returns a set containing the largest anagram set(s); that is, the set(s) with the most elements. In the example shown at the beginning of the question, the method would return a set containing the sets {introduces, reductions, discounter} and {retains, retinas, nastier}.

Complete method `findLargestSets` below.

```
/** @return a set of all anagram sets of largest size in this AnagramGrouper
 */
public HashSet<HashSet<String>> findLargestSets()
{
    HashSet<HashSet<String>> largest = new HashSet<HashSet<String>>();
    int maxSize = 1;
    for(String s: groups.keySet())
    {
        int groupSize = groups.get(s).size();
        if(groupSize > maxSize)
            maxSize = groupSize;
    }
    for(String s: groups.keySet())
    {
        HashSet<String> anagramSet = groups.get(s);
        if(anagramSet.size() == maxSize)
            largest.add(anagramSet);
    }
    return largest;
}
```

GO ON TO THE NEXT PAGE.

AB1b

(a) Write the `AnagramGrouper` constructor. The constructor takes a list of words and constructs a map in which each key is a key string and the associated value is the set of words that each have that key string. The map should contain all the anagram sets that are generated from the list of words.

Complete the `AnagramGrouper` constructor below.

```
/** Constructs a map from words in which the keys are key strings and the
 *  value associated with a key string is the set of anagrams having that key string.
 *  Postcondition: each entry of words is contained in an anagram set
 *  @param words a list of strings to be grouped into anagram sets
 *         Precondition: words.size() > 0
 */
public AnagramGrouper(List<String> words){
    for (String s : words){
        if(groups.containsKey(s.createKeyString())){
        ((Set) groups.get(s.createKeyString())).add(s);
        }
    }
}
```

GO ON TO THE NEXT PAGE.

AB1 b 2

(b) Write the `AnagramGrouper` method `findLargestSets`. This method analyzes the instance variable `groups` and returns a set containing the largest anagram set(s); that is, the set(s) with the most elements. In the example shown at the beginning of the question, the method would return a set containing the sets {introduces, reductions, discounter} and {retains, retinas, nastier}.

Complete method `findLargestSets` below.

```
/** @return a set of all anagram sets of largest size in this AnagramGrouper
 */
public HashSet<HashSet<String>> findLargestSets()
{
    HashSet<HashSet<String>> largestSets = new HashSet<HashSet<String>>();
    int maxSize = 0;
    for (String s : groups.keySet())
    {
        if (((Set)groups.get(s)).size() > maxSize)
        {
            largestSets = new HashSet<HashSet<String>>();
            largestSets.add(s);
        }
        else if (((Set)groups.get(s)).size() == maxSize)
        {
            largestSets.add(s);
        }
    }
    return largestSets;
}
```

GO ON TO THE NEXT PAGE.

ABIC1

(a) Write the `AnagramGrouper` constructor. The constructor takes a list of words and constructs a map in which each key is a key string and the associated value is the set of words that each have that key string. The map should contain all the anagram sets that are generated from the list of words.

Complete the `AnagramGrouper` constructor below.

```
/** Constructs a map from words in which the keys are key strings and the
 *  value associated with a key string is the set of anagrams having that key string.
 *  Postcondition: each entry of words is contained in an anagram set
 *  @param words a list of strings to be grouped into anagram sets
 *         Precondition: words.size() > 0
 */
public AnagramGrouper(List<String> words){
groups = new HashMap<String, HashSet<String>>();
for(int a=0; a <= words.size(); a++){
String ks = createKeyString(words.get(a));
if(!groups.containsKey(ks))
groups.put(ks, words.get(a));
}
}
}
```

GO ON TO THE NEXT PAGE.

ABIC2

(b) Write the `AnagramGrouper` method `findLargestSets`. This method analyzes the instance variable `groups` and returns a set containing the largest anagram set(s); that is, the set(s) with the most elements. In the example shown at the beginning of the question, the method would return a set containing the sets {introduces, reductions, discounter} and {retains, retinas, nastier}.

Complete method `findLargestSets` below.

```
/** @return a set of all anagram sets of largest size in this AnagramGrouper
 */
public HashSet<HashSet<String>> findLargestSets()
```

```
{
HashSet<HashSet<String>> lar;        int b;
lar = new HashSet<HashSet<String>>;  int x;
while(groups.keySet().iterator().hasNext){
x=groups.keySet().iterator
}
return lar;
```

GO ON TO THE NEXT PAGE.

AP® COMPUTER SCIENCE AB
2008 SCORING COMMENTARY

Question 1

Overview

This question focused on data structure design and use, and the application of basic algorithms. A data structure for storing sets of word anagrams was described, using a specific key string as the map key for each set. In part (a) students were required to implement the constructor for the `AnagramGrouper` class, which takes a list of words as a `parameter and builds the corresponding map instance variable. This involved traversing the list, creating the associated key strings (using a provided method), and adding the words to their associated sets (adding new sets to the map when necessary). In part (b) students were required to implement the `findLargestSets` method of the class, which identified all anagram sets of the maximum size. This involved traversing the map, identifying the largest sets, and collecting them in a `HashSet`.

Sample: AB1a
Score: 9

In part (a) the student's solution is completely correct. The student uses a for-each loop to access all the strings in `words`. There is a correct call to `createKeyString()`. When the key string `key` is already in `groups`, the student gets the associated `HashSet` and adds the string `s` to it. When the key string `key` is not already in `groups`, the student creates an empty `HashSet` and puts it in `groups`, then adds `s` to it.

In part (b) the student's solution is completely correct. The student constructs a `HashSet<HashSet<String>>` object and assigns it to the variable `largest`. The student correctly uses a for-each loop to access the anagram sets in `groups`. There is a correct determination of each anagram set's size and a correct comparison with the previous largest size. In the second for-each loop, when an anagram set is the same size as the largest set, it is correctly added to the set `largest`. Finally, the student returns the set of sets, `largest`.

Sample: AB1b
Score: 6

In part (a) the student fails to initialize the instance variable `groups`. The student correctly uses a for-each loop to access all the strings in `words`, thus earning both traverse ½ points. The student lost ½ point for incorrectly calling `s.createKeyString()`. The student does not handle the case where the key string is not in `groups`, so 1 point was lost. The student does handle the case when the key string is in `groups`, so 1 point was earned.

In part (b) the student constructs a `HashSet<HashSet<String>>` object and assigns it to the variable `largestSets`. The student correctly uses a for-each loop to access the anagram sets in `groups`. The student correctly determines the size of each `HashSet`. The comparison with `maxSize` is enough to earn the ½ comparison point. When a new largest set size is found, the student creates a new set of sets, assigns it to `largestSets`, and adds the current anagram set to `largestSets`. However, the student fails to update `maxSize`, so the 1 point for correctly constructing the set of sets was lost. Finally, the student earned the ½ point for returning `largestSets`.

AP® COMPUTER SCIENCE AB
2008 SCORING COMMENTARY

Question 1 (continued)

Sample: AB1c
Score: 2

In part (a) the student earned the ½ point for correctly initializing `groups`. The student correctly accesses one `String` in `words`; however, the access-all ½ point was lost because of the incorrect termination test (`a <= words.size()`). There is a correct call to `createKeyString()`. The call to `groups.put()` incorrectly uses a `String` instead of a `HashSet` as the second parameter, so both points for storing an anagram set in `groups` were lost.

In part (b) the student constructs a `HashSet<HashSet<String>>` object and assigns it to the variable `lar`. There is no code to iterate through the anagram sets or to find the largest anagram set. Because the response never adds any sets to `lar`, the student lost the ½ return point.