AP® COMPUTER SCIENCE AB
2006 SCORING GUIDELINES

Question 1: Thesaurus

Part A: `addSynonym` **4 points**

+1/2 correctly check if `word` is already stored in `wordMap`

+2 new word
- **+1/2** correctly create a new set (either `TreeSet` or `HashSet`)
- **+1/2** correctly add `syn` to set
- **+ 1** add new entry to `wordMap`
  - **+1/2** attempt (`wordMap.put(word,syn)` OK)
  - **+1/2** correct

+1 1/2 existing word
- **+ 1** access the set of synonyms
  - **+1/2** attempt (must access element of collection)
  - **+1/2** correct
- **+1/2** add `syn` to the set of synonyms

Part B: `removeSynonym` **5 points**

+1/2 correctly create a new set (either `TreeSet` or `HashSet`)

+1 1/2 iterate over all words in `wordMap`
- **+1/2** attempt to iterate over words in map
- **+1/2** get `keySet`
- **+1/2** correctly access each word in `keySet`

+2 1/2 process words (in context of loop)
- **+ 1** access the set of synonyms
  - **+1/2** attempt (must access element of collection)
  - **+1/2** correct
- **+1/2** check whether the set contains `syn`
- **+1/2** remove `syn` from set of synonyms in correct context of check
- **+1/2** add word to set of affected words in correct context of check

+1/2 return set of affected words (without destroying the `keySet`)

AP® COMPUTER SCIENCE A/AB
2006 GENERAL USAGE

Most common usage errors are addressed specifically in rubrics with points deducted in a manner other than indicated on this sheet. The rubric takes precedence.

Usage points can only be deducted if the part where it occurs has earned credit.

A usage error that occurs once when the same usage is correct two or more times can be regarded as an oversight and not penalized. If the usage error is the only instance, one of two, or occurs two or more times, then it should be penalized.

A particular usage error should be penalized only once in a problem, even if it occurs on different parts of a problem.

Nonpenalized Errors

spelling/case discrepancies*

local variable not declared when any other variables are declared in some part

default constructor called without parens; for example, `new Fish;`

use keyword as identifier

`[r,c]`, `(r)(c)` or `(r,c)` instead of `[r][c]`

= instead of == (and vice versa)

`length`/`size` confusion for array, `String`, and `ArrayList`, with or without `()`

`private` qualifier on local variable

extraneous code with no side-effect, for example a check for precondition

common mathematical symbols for operators (x • ÷ ≤ ≥ < > ≠)

missing { } where indentation clearly conveys intent

missing `( )` on method call or around `if/while` conditions

missing `;`s

missing "new" for constructor call once, when others are present in some part

missing downcast from collection

missing `int` cast when needed

missing `public` on class or constructor header

Minor Errors (1/2 point)

confused identifier (e.g., `len` for `length` or `left()` for `getLeft()`)

no local variables declared

`new` never used for constructor calls

`void` method or constructor returns a value

modifying a constant (`final`)

use `equals` or `compareTo` method on primitives, for example
`int x; …x.equals(val)`

`[]` – `get` confusion if access not tested in rubric

assignment dyslexia, for example, `x + 3 = y;` for `y = x + 3;`

`super(method())` instead of `super.method()`

formal parameter syntax (with type) in method call, e.g., `a = method(int x)`

missing `public` from method header when required

"false"/"true" or 0/1 for boolean values

"null" for `null`

Major Errors (1 point)

extraneous code which causes side-effect, for example, information written to output

use interface or class name instead of variable identifier, for example `Simulation.step()` instead of `sim.step()`

`aMethod(obj)` instead of `obj.aMethod()`

use of object reference that is incorrect, for example, use of `f.move()` inside method of `Fish` class

use private data or method when not accessible

destruction of data structure (e.g., by using root reference to a `TreeNode` for traversal of the tree)

use class name in place of `super` either in constructor or in method call

**Note: Spelling and case discrepancies for identifiers fall under the "nonpenalized" category as long as the correction can be unambiguously inferred from context. For example, "Queu" instead of "Queue". Likewise, if a student declares "Fish fish;", then uses Fish.move() instead of fish.move(), the context allows for the reader to assume the object instead of the class.*

AP® COMPUTER SCIENCE AB
2006 CANONICAL SOLUTIONS

Question 1: Thesaurus

PART A:

```
public void addSynonym(String word, String syn)
{
  if (!wordMap.containsKey(word))
  {
    Set synonyms = new TreeSet();
    synonyms.add(syn);
    wordMap.put(word, synonyms);
  }
  else
  {
    Set synonyms = (Set)wordMap.get(word);
    synonyms.add(syn);
  }
}
```

PART B:

```
public Set removeSynonym(String syn)
{
  Set affectedWords = new TreeSet();

  Set allWords = wordMap.keySet();
  Iterator iter = allWords.iterator();
  while (iter.hasNext())
  {
    String nextWord = (String)iter.next();
    Set synonyms = (Set)wordMap.get(nextWord);
    if (synonyms.remove(syn))
    {
      affectedWords.add(nextWord);
    }
  }

  return affectedWords;
}
```

AB1A,

(a) Write the `Thesaurus` method `addSynonym`. If `word` is already a key in `wordMap`, `syn` is added to the set associated with `word`. Otherwise, a new entry is added to `wordMap` with `word` as the key and a set containing `syn` as the associated value.

For example, assume that the `Thesaurus` object `myThesaurus` has been declared and initialized with the values shown at the beginning of this question. The following code segment contains two calls to `addSynonym`.

```
myThesaurus.addSynonym("wonderful", "magnificent");
myThesaurus.addSynonym("awesome", "wonderful");
```

After the code segment has executed, the contents of `myThesaurus` are as shown below.

Word	Synonym Set
excellent	{brilliant, great, outstanding, tremendous}
super	{excellent, fantastic, great, wonderful}
wonderful	{amazing, brilliant, fantastic, great, magnificent}
awesome	{wonderful}

Complete method `addSynonym` below.

```
// adds syn to the set of synonyms associated with word
// in this Thesaurus;
// if word is not a key in this Thesaurus, adds an
// entry with word as key and a set containing syn
// as the associated value
public void addSynonym(String word, String syn)
{
   Set s = (set) wordmap.get(word);
   if (s == null)
   {
      wordmap.put(word, s.add(syn));
   }
   else
   {
      s.add(syn);
      wordmap.put(word, s);
   }
}
```

Part (b) begins on page 6.

AB1 A2

Complete method removeSynonym below.

```
// removes the word syn from each synonym set in this Thesaurus;
// returns a set of the words (keys) whose associated
// synonym sets originally contained the word syn;
// if syn was not contained in any synonym set, returns an empty set.
public Set removeSynonym(String syn)
{
Set s = wordmap.keySet();
Set t = new HashSet();
Iterator itr = s.iterator();

while(itr.hasNext())
{
    String x = (String) itr.next();
    Set a = wordmap.get(x);
    Iterator itr2 = a.iterator();
    while(itr2.hasNext())
    {
      if (syn.equals((String) itr2.next()))
      {
        itr2.remove();
        t.add(x);
      }
    }

}

return t;

}
```

AB1 B1

(a) Write the `Thesaurus` method `addSynonym`. If `word` is already a key in `wordMap`, `syn` is added to the set associated with `word`. Otherwise, a new entry is added to `wordMap` with `word` as the key and a set containing `syn` as the associated value.

For example, assume that the `Thesaurus` object `myThesaurus` has been declared and initialized with the values shown at the beginning of this question. The following code segment contains two calls to `addSynonym`.

```
myThesaurus.addSynonym("wonderful", "magnificent");
myThesaurus.addSynonym("awesome", "wonderful");
```

After the code segment has executed, the contents of `myThesaurus` are as shown below.

Word	Synonym Set
excellent	{brilliant, great, outstanding, tremendous}
super	{excellent, fantastic, great, wonderful}
wonderful	{amazing, brilliant, fantastic, great, magnificent}
awesome	{wonderful}

Complete method `addSynonym` below.

```
// adds syn to the set of synonyms associated with word
// in this Thesaurus;
// if word is not a key in this Thesaurus, adds an
// entry with word as key and a set containing syn
// as the associated value
public void addSynonym(String word, String syn)
{
  if(!wordMap.contains(word))
    wordMap.put(word,(new HashSet()).add(syn));
  else
    {
     Set s = wordMap.get(word);
     wordMap.put(word, s.add(syn));
    }
}
```

Part (b) begins on page 6.

AB1B2

Complete method `removeSynonym` below.

```
// removes the word syn from each synonym set in this Thesaurus;
// returns a set of the words (keys) whose associated
// synonym sets originally contained the word syn;
// if syn was not contained in any synonym set, returns an empty set
public Set removeSynonym(String syn)
{ Set s = new HashSet();
  Iterator it = wordMap.keySet().iterator();
  while (it.hasNext())
  {
    String key = it.next();
    if (syn.equals((String)(wordMap.get(key))))
    {
      s.add(key);
      Set s1 = word.get(key);
      s1.remove(syn);
      word.put(key, s1);
    }
  }
  return s;
}
```

GO ON TO THE NEXT PAGE.

AB1 C1

(a) Write the `Thesaurus` method `addSynonym`. If `word` is already a key in `wordMap`, `syn` is added to the set associated with `word`. Otherwise, a new entry is added to `wordMap` with `word` as the key and a set containing `syn` as the associated value.

For example, assume that the `Thesaurus` object `myThesaurus` has been declared and initialized with the values shown at the beginning of this question. The following code segment contains two calls to `addSynonym`.

```
myThesaurus.addSynonym("wonderful", "magnificent");
myThesaurus.addSynonym("awesome", "wonderful");
```

After the code segment has executed, the contents of `myThesaurus` are as shown below.

Word	Synonym Set
excellent	{brilliant, great, outstanding, tremendous}
super	{excellent, fantastic, great, wonderful}
wonderful	{amazing, brilliant, fantastic, great, magnificent}
awesome	{wonderful}

Complete method `addSynonym` below.

```
// adds syn to the set of synonyms associated with word
// in this Thesaurus;
// if word is not a key in this Thesaurus, adds an
// entry with word as key and a set containing syn
// as the associated value
public void addSynonym(String word, String syn)
{
  if (myThesaurus.containsKey(word))
  {
    Set synset = mythesaurus.get(word);
    Synset.add(syn);
  }
  else
  {
    myThesaurus.put(word, syn);
  }
}
```

Part (b) begins on page 6.

AB1 C2

Complete method `removeSynonym` below.

```
// removes the word syn from each synonym set in this Thesaurus;
// returns a set of the words (keys) whose associated
// synonym sets originally contained the word syn;
// if syn was not contained in any synonym set, returns an empty set
public Set removeSynonym(String syn)
{
Set allwords = myThesaurus.keyset();
Set affwords = new Set();
Iterator iter = new iterator().
while (allwords.iter.hasNext()
{
Object viewWord = allwords.iter.next;
set synset = myThesaurus.get(viewWord);
if (synset.contains (syn))
{ synset.remove(syn);
affwords.add(viewWord);
}
```

AP® COMPUTER SCIENCE AB
2006 SCORING COMMENTARY

Question 1

Overview

This question focused on manipulating `Sets` and `Maps` in order to provide the functionality of a thesaurus. The skeleton of a `Thesaurus` class was provided, with a private data field of type `Map` (mapping words to `Sets` of synonyms). In part (a) students were required to complete the `addSynonym` method, which took a word/synonym pair and added a corresponding entry into the `Map`. This involved checking whether an entry for that word already existed, initializing a new entry if necessary, accessing the current `Set` of synonyms associated with the word, and adding the new synonym to that `Set`. In part (b) students were required to complete the `removeSynonym` method, which took a synonym and removed every entry containing that synonym from the `Map`. This involved iterating through all words in the `Map`, accessing each corresponding synonym `Set`, testing whether that `Set` contained the desired synonym, and removing each matching synonym. In solving this problem, students demonstrated a solid understanding of the `Map` and `Set` collections and effectively utilized these collections in organizing and accessing data.

Sample: AB1A
Score: 8

In part (a) the student checks for the word by attempting to get the entry for `word` and checking for `null`, earning a ½ point. In the case of a new word, the student does not create a new set and so lost the new set ½ point. The student adds the synonym correctly to a set, but the add new entry correctness ½ point was lost because the student uses the return value for set `add` (a `boolean`) as the second argument to `put`. The student handles existing words correctly, earning all three existing word ½ points.

In part (b) the code is completely correct. The student creates an iterator based on the keyset and uses it in a loop to access each entry correctly, earning the three iterate ½ points. The set of synonyms is accessed correctly, which earned the two access set ½ points. The student uses an iterator over the synonym set to check for and remove `syn` from each entry where it occurred in the value. This loop earned the check synonym and remove synonym ½ points. The affected words set is also created, built, and returned correctly, earning the corresponding ½ points.

Sample: AB1B
Score: 6

In part (a) the student uses `contains` rather than `containskey` to check for the word, which lost the word check ½ point. In the case of a new word, the student creates a new set and adds the synonym correctly, which earned the new set and add synonym ½ points. The add new entry correctness ½ point was lost because the student uses the return value for set `add` (a `boolean`) as the second argument to `put`. In the case of an existing word, the student accesses the set correctly, earning the two access set ½ points. The student again uses the return value for set `add` (a `boolean`) as the second argument to `put`, which lost the add synonym ½ point for existing word.

In part (b) a ½ point usage deduction was assessed for the use of `word` rather than `wordMap` in two places. The student creates an iterator based on the keyset and uses it in a loop to access each entry correctly, earning the three iterate ½ points. The set of synonyms is accessed but then cast as a string, which lost the access set correct ½ point. The student uses `equal` rather than `contains` to check for the synonym, losing the check synonym ½ point. The student accesses the set again, this time correctly, and then removes the synonym, which earned the remove synonym ½ point. The affected words set is also created, built, and returned correctly, earning the new set, add word, and return ½ points.

AP® COMPUTER SCIENCE AB
2006 SCORING COMMENTARY

Question 1 (continued)

Sample: AB1C
Score: 2

In part (a) the student uses `myThesaurus`, an undeclared variable, throughout the code, which lost all ½ points except add synonym in the case of an existing word. The student earned the add synonym ½ point for adding the synonym correctly to what is assumed to be the retrieved set of synonyms.

In part (b) the student correctly checks a set to determine if it contains the synonym and if so removes `syn`, earning the check synonym and remove synonym ½ points. The student also adds the affected word to a set, which earned the add word ½ point. The student again uses `myThesaurus` throughout the code, losing all remaining ½ points in this section.