AP® Computer Science A
2008 Scoring Guidelines
**Question 1: Flight List**

<table>
<thead>
<tr>
<th>Part A:</th>
<th>getDuration</th>
<th>4 points</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>handle empty case</td>
<td></td>
</tr>
<tr>
<td>+1/2</td>
<td>check if flights is empty</td>
<td></td>
</tr>
<tr>
<td>+1/2</td>
<td>return 0 if empty</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>access start time</td>
<td></td>
</tr>
<tr>
<td>+1/2</td>
<td>access flights.get(0)</td>
<td></td>
</tr>
<tr>
<td>+1/2</td>
<td>correctly call getDepartureTime on a flight</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>access end time</td>
<td></td>
</tr>
<tr>
<td>+1/2</td>
<td>access flights.get(flights.size()-1)</td>
<td></td>
</tr>
<tr>
<td>+1/2</td>
<td>correctly call getArrivalTime on a flight</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>calculate and return duration</td>
<td></td>
</tr>
<tr>
<td>+1/2</td>
<td>call minutesUntil using Time objects</td>
<td></td>
</tr>
<tr>
<td>+1/2</td>
<td>return correct duration (using minutesUntil)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Part B:</th>
<th>getShortestLayover</th>
<th>5 points</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>handle case with 0 or 1 flight</td>
<td></td>
</tr>
<tr>
<td>+1/2</td>
<td>check if flights.size() &lt; 2</td>
<td></td>
</tr>
<tr>
<td>+1/2</td>
<td>return -1 in that case</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>traverse flights</td>
<td></td>
</tr>
<tr>
<td>+1/2</td>
<td>correctly access an element of flights (in context of loop)</td>
<td></td>
</tr>
<tr>
<td>+1/2</td>
<td>access all elements of flights (lose this if index out-of-bounds)</td>
<td></td>
</tr>
<tr>
<td>+2 1/2</td>
<td>find shortest layover (in context of loop)</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>get layover time between successive flights (using minutesUntil)</td>
<td></td>
</tr>
<tr>
<td>+1/2</td>
<td>compare layover time with some previous layover</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>correctly identify shortest layover</td>
<td></td>
</tr>
<tr>
<td>+1/2</td>
<td>return shortest layover</td>
<td></td>
</tr>
</tbody>
</table>
Question 2: String Coder

**Part A:** decodeString  
4 1/2 points

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

+2 retrieve substrings from masterString
+1/2 correctly call getStart() and getLength() on accessed part
+1 1/2 extract a substring from masterString
  +1/2 masterString.substring(X,Y)
  +1 extract correct substring

+1 1/2 build and return decoded string
  +1 correctly build string from substrings of masterString
  +1/2 return built string

**Part B:** encodeString  
4 1/2 points

+1/2 construct an ArrayList<StringPart> (must assign to a variable, generic okay)

+3 1/2 find, collect string parts, and build list (in context of loop)
  +1 findPart(X), where X is word or a substring of word
  +1 calls to findPart involve progressively smaller suffixes of word
  +1/2 add found string part to ArrayList of string parts
  +1 build correct list of string parts (must have used findPart)

+1/2 return ArrayList of string parts
Question 3: Opossum Critter (GridWorld)

Part A: processActors 6 points

+1/2 initialize friend/foe counter(s)

+2 1/2 loop and identify actors
   +1 traverse actors
     +1/2 correctly access an element of actors (in context of loop)
     +1/2 access all elements of actors (lose this if index out-of-bounds)
   +1 1/2 identify actor category and update counters (in context of loop)
     +1/2 call isFriend(nextActorFromList)
     +1/2 call isFoe(nextActorFromList)
     +1/2 update counters appropriately in both cases

+3 update OpossumCritter state
   +1 correctly identify whether to play dead
   +1 appropriate result if playing dead
     +1/2 setColor(Color.BLACK)
     +1/2 numStepsDead++
   +1 appropriate result if normal
     +1/2 setColor(Color.ORANGE)
     +1/2 numStepsDead = 0

Part B: selectMoveLocation 3 points

+1 determine appropriate case (using == with Color is okay)
   +1/2 correctly identify one case (dead, playing dead, normal)
   +1/2 correctly identify all three cases

+2 appropriate return values
   +1/2 return null if really dead
   +1/2 return current location if playing dead
   +1 return super.selectMoveLocation(locs) otherwise
     +1/2 super.selectMoveLocation(locs)
     +1/2 return value from call

Usage:
-1 if violate postconditions (e.g., removeSelfFromGrid())
-1 for BLACK or “Black” instead of Color.BLACK
-1/2 for call to (nonexistent) default Location constructor
Question 4: Checker Objects (Design)

**Part A:** SubstringChecker 4 points

+1/2 class SubstringChecker implements Checker

+1/2 declare private instance variable of type String

+1 constructor
  +1/2 SubstringChecker(String goalString)
  +1/2 initialize instance variable to parameter

+2 accept method
  +1/2 public boolean accept(String text)
  +1 1/2 determine whether to accept
    +1/2 attempt to find instance variable in text
      (either call indexOf, contains, or compare with substrings)
    +1 return correct boolean value in all cases

**Part B:** AndChecker 4 points

+1/2 class AndChecker implements Checker

+1/2 declare private instance variable(s) capable of storing two Checker objects

+1 constructor
  +1/2 AndChecker(Checker c1, Checker c2)
  +1/2 initialize instance variable(s) to parameters

+2 accept method
  +1/2 public boolean accept(String text)
  +1 1/2 determine whether to accept
    +1/2 attempt to call accept(text) on both stored Checkers
    +1 return correct boolean value in all cases

**Part C:** yummyChecker 1 point

+1 correctly assign yummyChecker