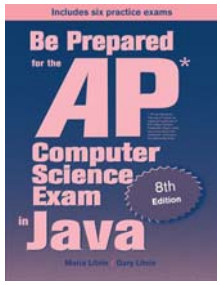

Eighth Edition

Be Prepared
for the
AP
Computer Science Exam in Java

Chapter 6: Annotated Solutions to Past Free-Response Questions

2021

Maria Litvin
Phillips Academy, Andover, Massachusetts

Gary Litvin
Skylight Publishing, Andover, Massachusetts

Skylight Publishing
Andover, Massachusetts

Library of Congress Control Number: 2021950662

ISBN 978-0-9972528-7-3

Skylight Publishing
9 Bartlet Street, Suite 70
Andover, MA 01810

web: www.skylit.com
e-mail: sales@skylit.com
support@skylit.com

www.skylit.com/beprepared/x2021all.zip contains complete Java code, including solutions and test programs for runnable projects.

The free-response questions for this exam are posted on apstudent.collegeboard.org and, for teachers, on AP Central:

- For students: `apstudents.collegeboard.org`
- For teachers: `apcentral.collegeboard.org`

Scoring guidelines for teachers are usually posted over the summer.

Question 1

Part (a)

```
public int scoreGuess(String guess)
{
  int score = 0;
  int n = guess.length();
  for (int i = 0; i <= secret.length() - n; i++) [1]
    if (guess.equals(secret.substring(i, i+n))) [2]
      score++;
  return score * n*n;
}
```

Notes:

1. Trace your code for a simple case to avoid "fencepost" errors. Consider, for example, the case when `guess` and `secret` have the same length (`secret.length() == n`). In that case, the code in the `for` loop should run once.

2. Always use `equals` when comparing strings, not ==.

Part (b)

```
public String findBetterGuess(String guess1, String guess2)
{
  int score1 = scoreGuess(guess1);
  int score2 = scoreGuess(guess2);
  if (score1 > score2 || score1 == score2 &&
                           guess1.compareTo(guess2) > 0) [1]
    return guess1;
  else
    return guess2;
 }
```

Notes:

1. A common mistake would be to write:

```
if (score1 > score2 || guess1.compareTo(guess2) > 0)...
```

Question 2

```
public class CombinedTable
{
  private SingleTable table1;
  private SingleTable table2;

  public CombinedTable(SingleTable t1, SingleTable t2)
  {
    table1 = t1;
    table2 = t2;
  }

  public boolean canSeat(int people) [1]
  {
    return table1.getNumSeats() + table2.getNumSeats() - 2 >= people; [2]
  }

  public double getDesirability()
  {
    double avg = (table1.getViewQuality() +
                                  table2.getViewQuality()) / 2;
    if (table1.getHeight() == table2.getHeight())
      return avg;
    else
      return avg - 10;

  }
```

Notes:

1. The given examples indicate that `canSeat` must be a `boolean` method.

2. Avoid the redundant and verbose version

```
if table1.getNumSeats() + table2.getNumSeats() - 2 >= people
  return true;
else
  return false;
```

Question 3

Part (a)

```
public void addMembers(String[] names, int gradYear)
{
  for (String name : names)
    memberList.add(new MemberInfo(name, gradYear, true)); [1]
}
```

Notes:

1. You need to create a new `MemberInfo` object before adding it to `memberList`. Use the constructor specified in the `MemberInfo` class; do not forget to use the `new` operator.

Part (b)

```
public ArrayList<MemberInfo> removeMembers(int year)
{
  ArrayList<MemberInfo> newList = new ArrayList<MemberInfo>();
  for (int i = memberList.size() - 1; i >= 0; i--) [1]
  {
    MemberInfo member = memberList.get(i);
    if (member.getGradYear() <= year)
    {
      memberList.remove(i);
      if (member.inGoodStanding())
        newList.add(member);
    }
  }
  return newList;
}
```

Notes:

1. Removing elements from a list is one of the standard algorithms. One way to do it correctly is to traverse the list backwards, as shown above. The question states that the order of elements in the returned list does not matter, so going backwards doesn't cause any problems. You can also traverse the list forward, but you have to be careful then not to increment the index when an element is removed, because the indices of the subsequent elements shift down by one. For example:

```
int i = 0;
while (i < memberList.size())
{
  MemberInfo member = memberList.get(i);
```

Continued

```
    if (member.getGradYear() <= year)
    {
      memberList.remove(i);
      if (member.inGoodStanding())
        newList.add(member);
    }
    else
      i++;
  }
```

You might be tempted to use a `for` loop and subtract 1 from `i` inside it when an element is removed. This works, but we think it is bad style to change the loop control variable inside a `for` loop.

Question 4

Part (a)

```
public static boolean isNonZeroRow(int[][] array2D, int r)
{
   for (int x : array2D[r]) [1]
      if (x == 0)
         return false;
   return true; [2]
}
```

Notes:

1. Recall that in Java a 2D array is an array of 1D arrays, its rows.

2.
```
      if (x == 0)
         return false;
      else
         return true;
```
would be a mistake.

Part (b)

```
public static int[][] resize(int[][] array2D)
{
   int numRows = numNonZeroRows(array2D); [1]
   int numCols = array2D[0].length;
   int[][] smaller = new int[numRows][numCols];

   int k = 0; [2]
   for (int r = 0; r < array2D.length; r++)
   {
      if (isNonZeroRow(array2D, r)) [1]
      {
         for (int c  = 0; c < numCols; c++)
            smaller[k][c] = array2D[r][c];
         k++;
      }
   }
   return smaller;
} [3]
```

Notes:

1. To receive full credit, it is important to use the methods specified in the `ArrayResizer` class, even if you skipped Part (a).

Continued

2. `k` is the index of the row in `smaller` that is being filled with values; we need to keep track of it.

3. In the above solution we truthfully allocated a new 2D array and copied the values from `array2D` into it. It is possible to write a "working" solution in which we allocate only an array of references to rows and set these references to point to certain rows of `array2D`:

```
public static int[][] resize(int[][] array2D)
{
   int numRows = numNonZeroRows(array2D);
   int[][] smaller = new int[numRows][];
                             // Array of references to rows
   int k = 0;
   for (int r = 0; r < array2D.length; r++)
   {
      if (isNonZeroRow(array2D, r))
      {
         // Set smaller[k] to the reference to a row in array2D:
         smaller[k] = array2D[r];
         k++;
      }
   }
   return smaller;
}
```

At first glance, this works as specified: if we display `smaller` after calling `resize` we get

```
1 3 2
4 5 6
```

However, here we have two arrays of rows, `arr` and `smaller`, that refer to the same objects — a situation called *aliasing*. Aliasing leads to subtle bugs that are hard to catch. If we modify the values in `arr` after calling `resize`, for example,

```
arr[1][1] = 0;
```

the values in `smaller` will change, too:

```
1 0 2
4 5 6
```

The Part (b) description says: "Write the method `resize`, which returns a new two-dimensional array containing only rows from `array2D` with all non-zero values." One might argue that we did. We don't know at this point whether solutions with aliasing will receive full credit.