3. A high school club maintains information about its members in a `MemberInfo` object. A `MemberInfo` object stores a club member's name, year of graduation, and whether or not the club member is in *good standing*. A member who is in good standing has fulfilled all the responsibilities of club membership.

   A partial declaration of the `MemberInfo` class is shown below.

```
public class MemberInfo
{
   /** Constructs a MemberInfo object for the club member with name name,
    *  graduation year gradYear, and standing hasGoodStanding.
    */
   public MemberInfo(String name, int gradYear, boolean hasGoodStanding)
   {  /* implementation not shown */  }

   /** Returns the graduation year of the club member. */
   public int getGradYear()
   {  /* implementation not shown */  }

   /** Returns true if the member is in good standing and false otherwise. */
   public boolean inGoodStanding()
   {  /* implementation not shown */  }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

   The `ClubMembers` class maintains a list of current club members. The declaration of the `ClubMembers` class is shown below.

```
public class ClubMembers
{
   private ArrayList<MemberInfo> memberList;

   /** Adds new club members to memberList, as described in part (a).
    *  Precondition: names is a non-empty array.
    */
   public void addMembers(String[] names, int gradYear)
   {  /* to be implemented in part (a) */  }

   /** Removes members who have graduated and returns a list of members who have graduated
    *  and are in good standing, as described in part (b).
    */
   public ArrayList<MemberInfo> removeMembers(int year)
   {  /* to be implemented in part (b) */  }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

**GO ON TO THE NEXT PAGE.**

(a) Write the `ClubMembers` method `addMembers`, which takes two parameters. The first parameter is a `String` array containing the names of new club members to be added. The second parameter is the graduation year of all the new club members. The method adds the new members to the `memberList` instance variable. The names can be added in any order. All members added are initially in good standing and share the same graduation year, `gradYear`.

Complete the `addMembers` method.

```
/** Adds new club members to memberList, as described in part (a).
 *  Precondition: names is a non-empty array.
 */
public void addMembers(String[] names, int gradYear)
```

---

**Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.**

Complete the `removeMembers` method.

```
/** Removes members who have graduated and returns a list of members who have graduated and are
 *  in good standing, as described in part (b).
 */
public ArrayList<MemberInfo> removeMembers(int year)
```

---

**Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.**

Class information for this question

```
public class MemberInfo

public MemberInfo(String name, int gradYear, boolean hasGoodStanding)
public int getGradYear()
public boolean inGoodStanding()

public class ClubMembers

private ArrayList<MemberInfo> memberList

public void addMembers(String[] names, int gradYear)
public ArrayList<MemberInfo> removeMembers(int year)
```

**GO ON TO THE NEXT PAGE.**

4. This question involves manipulating a two-dimensional array of integers. You will write two `static` methods of the `ArrayResizer` class, which is shown below.

```
public class ArrayResizer
{
    /** Returns true if and only if every value in row r of array2D is non-zero.
     *  Precondition: r is a valid row index in array2D.
     *  Postcondition: array2D is unchanged.
     */
    public static boolean isNonZeroRow(int[][] array2D, int r)
    {  /* to be implemented in part (a) */  }

    /** Returns the number of rows in array2D that contain all non-zero values.
     *  Postcondition: array2D is unchanged.
     */
    public static int numNonZeroRows(int[][] array2D)
    {  /* implementation not shown */  }

    /** Returns a new, possibly smaller, two-dimensional array that contains only rows
     *  from array2D with no zeros, as described in part (b).
     *  Precondition: array2D contains at least one column and at least one row with no zeros.
     *  Postcondition: array2D is unchanged.
     */
    public static int[][] resize(int[][] array2D)
    {  /* to be implemented in part (b) */  }
}
```

**GO ON TO THE NEXT PAGE.**

## Question 3: Array / ArrayList — 9 points

**Canonical solution**

**(a)** — **3 points**

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

**(b)** — **6 points**

```
public ArrayList<MemberInfo> removeMembers(int year)
{
   ArrayList<MemberInfo> removed = new ArrayList<MemberInfo>();

   for (int i = memberList.size() - 1; i >= 0; i--)
   {
      if (memberList.get(i).getGradYear() <= year)
      {
         if (memberList.get(i).inGoodStanding())
         {
            removed.add(memberList.get(i));
         }
         memberList.remove(i);
      }
   }
   return removed;
}
```

**(b)** `removeMembers`

| | Scoring Criteria | Decision Rules | |
|---|---|---|---|
| **4** | Declares and initializes an `ArrayList` of `MemberInfo` objects | Responses **will not** earn the point if they initialize the variable with a reference to the instance variable | **1 point** |
| **5** | Accesses all elements of `memberList` for potential removal (*no bounds errors*) | Responses **will not** earn the point if they<br>• fail to use `get(i)`<br>• fail to attempt to remove an element<br>• skip an element<br>• throw an exception due to removing | **1 point** |
| **6** | Calls `getGradYear` or `inGoodStanding` | Responses **can** still earn the point even if they call only one of the methods<br><br>Responses **will not** earn the point if they<br>• ever include parameters in either method call<br>• ever call either method on an object other than `MemberInfo` | **1 point** |
| **7** | Distinguishes any three cases, based on graduation status and standing | Responses **will not** earn the point if they fail to behave differently in all three cases | **1 point** |
| **8** | Identifies graduating members | Responses **can** still earn the point even if they<br>• fail to distinguish three cases<br>• fail to access standing at all<br>• access the graduating year incorrectly<br><br>Responses **will not** earn the point if they confuse `<` and `<=` in the comparison | **1 point** |
| **9** | Removes appropriate members from `memberList` and adds appropriate members to the `ArrayList` to be returned | Responses **can** still earn the point even if they<br>• call `getGradYear` or `inGoodStanding` incorrectly<br>• access elements of `memberList` incorrectly<br>• initialize the `ArrayList` incorrectly<br>• fail to return the list that was built (*return is not assessed*)<br><br>Responses **will not** earn the point if they<br>• fail to declare an `ArrayList` to return<br>• fail to distinguish the correct three cases, with the exception of confusing the `<` and `<=` in the comparison | **1 point** |
| | | **Total for part (b)** | **6 points** |

| Question-specific penalties |
| --- |
| None |

**Total for question 3 9 points**

## Applying the Scoring Criteria

Apply the question scoring criteria first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

**1-Point Penalty**

v) Array/collection access confusion (`[] get`)
w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)
x) Local variables used but none declared
y) Destruction of persistent data (e.g., changing value referenced by parameter)
z) Void method or constructor that returns a value

**No Penalty**

- Extraneous code with no side-effect (e.g., valid precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators (× • ÷ ≤ ≥ <> ≠)
- `[]` vs. `()` vs. `<>`
- `=` instead of `==` and vice versa
- `length/size` confusion for array, `String`, `List`, or `ArrayList`; with or without `( )`
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, e.g., `int[size] nums = new int[size];`
- Missing `;` where structure clearly conveys intent
- Missing `{ }` where indentation clearly conveys intent
- Missing `( )` on parameter-less method or constructor invocations
- Missing `( )` around `if` or `while` conditions

**Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be **unambiguously** inferred from context, for example, "ArayList" instead of "ArrayList". As a counterexample, note that if the code declares "`int G=99, g=0;`", then uses "`while (G < 10)`" instead of "`while (g < 10)`", the context does **not** allow for the reader to assume the use of the lower case variable.*