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;
}
```

(a) `addMembers`

	Scoring Criteria	Decision Rules	
1	Accesses all elements of `names` (*no bounds errors*)	Responses **will not** earn the point if they fail to access elements of the array, even if loop bounds are correct	**1 point**
2	Instantiates a `MemberInfo` object with name from array, provided year, and good standing		**1 point**
3	Adds `MemberInfo` objects to `memberList` (*in the context of a loop*)	Responses **can** earn the point even if they instantiate `MemberInfo` objects incorrectly	**1 point**
		Total for part (a)	**3 points**

(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 • fail to use `get(i)` • fail to attempt to remove an element • skip an element • 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 Responses **will not** earn the point if they • ever include parameters in either method call • 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 • fail to distinguish three cases • fail to access standing at all • access the graduating year incorrectly 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 • call `getGradYear` or `inGoodStanding` incorrectly • access elements of `memberList` incorrectly • initialize the `ArrayList` incorrectly • fail to return the list that was built (*return is not assessed*) Responses **will not** earn the point if they • fail to declare an `ArrayList` to return • 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.*