AP2025 FR

A mathematical sequence is an ordered list of numbers. This question involves a sequence called a *hailstone sequence*. If n is the value of a term in the sequence, then the following rules are used to find the next term, if one exists.

- If n is 1, the sequence terminates.
- If n is even, then the next term is $\frac{n}{2}$.
- If n is odd, then the next term is $3n + 1$.

For this question, assume that when the rules are applied, the sequence will eventually terminate with the term $n = 1$.

The following are examples of hailstone sequences.

Example 1: 5, 16, 8, 4, 2, 1

- The first term is 5, so the second term is $5 * 3 + 1 = 16$.
- The second term is 16, so the third term is $\frac{16}{2} = 8$.
- The third term is 8, so the fourth term is $\frac{8}{2} = 4$.
- The fourth term is 4, so the fifth term is $\frac{4}{2} = 2$.
- The fifth term is 2, so the sixth term is $\frac{2}{2} = 1$.
- Since the sixth term is 1, the sequence terminates.

Example 2: 8, 4, 2, 1

- The first term is 8, so the second term is $\frac{8}{2} = 4$.
- The second term is 4, so the third term is $\frac{4}{2} = 2$.
- The third term is 2, so the fourth term is $\frac{2}{2} = 1$.
- Since the fourth term is 1, the sequence terminates.

The `Hailstone` class, shown below, is used to represent a hailstone sequence. You will write three methods in the `Hailstone` class.

```
public class Hailstone
{
     /** Returns the length of a hailstone sequence that starts with n,
       * as described in part (a).
       * Precondition: n > 0
       */
     public static int hailstoneLength(int n)
     { /* to be implemented in part (a) */ }
  /** Returns true if the hailstone sequence that starts with n is considered
long
       * and false otherwise, as described in part (b).
       * Precondition: n > 0
       */
     public static boolean isLongSeq(int n)
     { /* to be implemented in part (b) */ }
  /** Returns the proportion of the first n hailstone sequences that are
considered long,
       * as described in part (c).
       * Precondition: n > 0
       */
     public static double propLong(int n)
     { /* to be implemented in part (c) */ }
  // There may be instance variables, constructors, and methods not shown.
}
```

AP2025 FR

1. (a) The length of a hailstone sequence is the number of terms it contains. For example, the hailstone sequence in example 1 (5, 16, 8, 4, 2, 1) has a length of 6 and the hailstone sequence in example 2 (8, 4, 2, 1) has a length of 4.

Write the method `hailstoneLength(int n)`, which returns the length of the hailstone sequence that starts with `n`.

```
/** Returns the length of a hailstone sequence that starts with n,
 * as described in part (a).
 * Precondition: n > 0
 */
public static int hailstoneLength(int n)
```

Class information for this question

```
public class Hailstone
public static int hailstoneLength(int n)
public static boolean isLongSeq(int n)
public static double propLong(int n)
```

(b) A hailstone sequence is considered long if its length is greater than its starting value. For example, the hailstone sequence in example 1 (5, 16, 8, 4, 2, 1) is considered long because its length (6) is greater than its starting value (5). The hailstone sequence in example 2 (8, 4, 2, 1) is not considered long because its length (4) is less than or equal to its starting value (8).

Write the method `isLongSeq(int n)`, which returns `true` if the hailstone sequence starting with `n` is considered long and returns `false` otherwise. Assume that `hailstoneLength` works as intended, regardless of what you wrote in part (a). You must use `hailstoneLength` appropriately to receive full credit.

```
/** Returns true if the hailstone sequence that starts with n is
considered long
 * and false otherwise, as described in part (b).
 * Precondition: n > 0
 */
public static boolean isLongSeq(int n)
```

(c) The method `propLong(int n)` returns the proportion of long hailstone sequences with starting values between 1 and `n`, inclusive.

Consider the following table, which provides data about the hailstone sequences with starting values between 1 and 10, inclusive.

AP2025 FR

Starting Value	Terms in the Sequence	Length of the Sequence	Long?
1	1	1	No
2	2, 1	2	No
3	3, 10, 5, 16, 8, 4, 2, 1	8	Yes
4	4, 2, 1	3	No
5	5, 16, 8, 4, 2, 1	6	Yes
6	6, 3, 10, 5, 16, 8, 4, 2, 1	9	Yes
7	7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1	17	Yes
8	8, 4, 2, 1	4	No
9	9, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1	20	Yes
10	10, 5, 16, 8, 4, 2, 1	7	No

The method call `Hailstone.propLong(10)` returns `0.5`, since 5 of the 10 hailstone sequences shown in the table are considered long.

Write the `propLong` method. Assume that `hailstoneLength` and `isLongSeq` work as intended, regardless of what you wrote in parts (a) and (b). You must use `isLongSeq` appropriately to receive full credit.

```
/** Returns the proportion of the first n hailstone sequences that are
considered long,
 * as described in part (c).
 * Precondition: n > 0
 */
public static double propLong(int n)
```

Class information for this question

```
public class Hailstone
public static int hailstoneLength(int n)
public static boolean isLongSeq(int n)
public static double propLong(int n)
```

Part A – hailstoneLength method

Select a point value to view scoring criteria, solutions, and/or examples and to score the response. +1 indicates a point earned and -1 indicates a general or question-specific penalty.

AP2025 FR

+1 [Skill 3.C] Loops from given starting value `n` until the sequence terminates, using updated values for the current term

Responses still earn the point even if they...

· update `n` incorrectly.

+1 [Skill 3.C] Computes the next value

Responses still earn the point even if they...

· use a correct formula in an incorrect case.

+1 [Skill 3.C] Uses correct formula for next value depending on even/odd

-1 (w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)

-1 (x) Local variables used but none declared

Canonical Solution:

```java
public static int hailstoneLength(int n)
{
    int count = 1;
    while (n > 1)
    {
        if (n % 2 == 0)
        {
            n = n / 2;
        }
        else
        {
            n = 3 * n + 1;
        }
        count++;
    }
    return count;
}
```

1) Line 5: { Line 6: if (n % 2 == 0) Line 7: { Line 8: n = n / 2; Line 9: } Line 10: else Line 11: { Line 12: n = 3 * n + 1; Line 13: } Line 14: count ++; Line 15: } Line 16: return count; Line 17: }" title="">

			✓
0	1	2	3

Total number of points earned (minus penalties) is equal to 3.

☐ **+1** Loops from given starting value `n` until the sequence terminates, using updated values for the current term **(Points earned)**

☐ **+1** Computes the next value **(Points earned)**

☐ **+1** Uses correct formula for next value depending on even/odd **(Points earned)**

AP2025 FR

- ☐ **-1** [penalty] (w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check) **(General penalties)**
- ☐ **-1** [penalty] (x) Local variables used but none declared **(General penalties)**

Canonical Solution:

```
public static int hailstoneLength(int n)
{
   int count = 1;
   while (n > 1)
   {
      if (n % 2 == 0)
      {
         n = n / 2;
      }
      else
      {
         n = 3 * n + 1;
      }
      count++;
   }
   return count;
}
```

1) Line 5: { Line 6: if (n % 2 == 0) Line 7: { Line 8: n = n / 2; Line 9: } Line 10: else Line 11: { Line 12: n = 3 * n + 1; Line 13: } Line 14: count ++; Line 15: } Line 16: return count; Line 17: }">

Part B – isLongSeq method

Select a point value to view scoring criteria, solutions, and/or examples and to score the response. +1 indicates a point earned and -1 indicates a general or question-specific penalty.

+1 [Skill 3.A] Calls hailstoneLength

+1 [Skill 3.C] Correctly compares length and starting value to determine return value

Responses still earn the point even if they...

· call hailstoneLength incorrectly.

-1 (w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)

-1 (x) Local variables used but none declared

Canonical Solution:

```
public static boolean isLongSeq(int n)
{
   return hailstoneLength(n) > n;
}
```

n; Line 4: }">

AP2025 FR

		✓
0	1	2

Total number of points earned (minus penalties) is equal to 2.

- ☐ **+1** Calls hailstoneLength **(Points earned)**
- ☐ **+1** Correctly compares length and starting value to determine return value **(Points earned)**
- ☐ **-1** [penalty] (w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check) **(General penalties)**
- ☐ **-1** [penalty] (x) Local variables used but none declared **(General penalties)**

Canonical Solution:

```
public static boolean isLongSeq(int n)
{
    return hailstoneLength(n) > n;
}
```

n; Line 4: }">

Part C – propLong method

Select a point value to view scoring criteria, solutions, and/or examples and to score the response. +1 indicates a point earned and -1 indicates a general or question-specific penalty.

+1 [Skill 3.A] Calls isLongSeq in the context of a loop

+1 [Skill 3.C] Loops 1 to n (*no bounds errors*)

+1 [Skill 3.C] Calculates double proportion

Responses still earn the point even if they...

· use incorrect values for the count of long sequences or n.

+1 [Skill 3.B] Returns correctly calculated value

-1 (w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)

-1 (x) Local variables used but none declared

Canonical Solution:

AP2025 FR

```
public static double propLong(int n)
{
    int count = 0;
    for (int i = 1; i <= n; i++)
    {
        if (isLongSeq(i))
        {
            count++;
        }
    }
    return (double) count / n;
}
```

0	1	2	3	✓ 4

Total number of points earned (minus penalties) is equal to 4.

- ☐ **+1** Calls isLongSeq in the context of a loop **(Points earned)**
- ☐ **+1** Loops 1 to n (*no bounds errors*) **(Points earned)**
- ☐ **+1** Calculates double proportion **(Points earned)**
- ☐ **+1** Returns correctly calculated value **Points earned)**
- ☐ **-1** [penalty] (w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check) **(General penalties)**
- ☐ **-1** [penalty] (x) Local variables used but none declared **(General penalties)**

Canonical Solution:

```
public static double propLong(int n)
{
    int count = 0;
    for (int i = 1; i <= n; i++)
    {
        if (isLongSeq(i))
        {
            count++;
        }
    }
    return (double) count / n;
}
```

AP2025 FR

This question involves the creation and use of a spinner to generate random numbers in a game.
A `GameSpinner` object represents a spinner with a given number of sectors, all equal in size.
The `GameSpinner` class supports the following behaviors.

- Creating a new spinner with a specified number of sectors
- Spinning a spinner and reporting the result
- Reporting the length of the *current run*, the number of consecutive spins that are the same as the most recent spin

The following table contains a sample code execution sequence and the corresponding results.

AP2025 FR

Statements	Value Returned (blank if no value returned)	Comment
`GameSpinner g = new GameSpinner(4);`		Creates a new spinner with four sectors
`g.currentRun();`	`0`	Returns the length of the current run. The length of the current run is initially `0` because no spins have occurred.
`g.spin();`	`3`	Returns a random integer between `1` and `4`, inclusive. In this case, `3` is returned.
`g.currentRun();`	`1`	The length of the current run is `1` because there has been one spin of `3` so far.
`g.spin();`	`3`	Returns a random integer between `1` and `4`, inclusive. In this case, `3` is returned.
`g.currentRun();`	`2`	The length of the current run is `2` because there have been two `3`s in a row.
`g.spin();`	`4`	Returns a random integer between `1` and `4`, inclusive. In this case, `4` is returned.
`g.currentRun();`	`1`	The length of the current run is `1` because the spin of `4` is different from the value of the spin in the previous run of two `3`s.
`g.spin();`	`3`	Returns a random integer between `1` and `4`, inclusive. In this case, `3` is returned.
`g.currentRun();`	`1`	The length of the current run is `1` because the spin of `3` is different from the value of the spin in the previous run of one `4`.
`g.spin();`	`1`	Returns a random integer between `1` and `4`, inclusive. In this case, `1` is returned.
`g.spin();`	`1`	Returns a random integer between `1` and `4`, inclusive. In this case, `1` is returned.
`g.spin();`	`1`	Returns a random integer between `1` and `4`, inclusive. In this case, `1` is returned.
`g.currentRun();`	`3`	The length of the current run is `3` because there have been three consecutive `1`s since the previous run of one `3`.

2. Write the complete `GameSpinner` class. Your implementation must meet all specifications and conform to the example.

AP2025 FR

GameSpinner class

Select a point value to view scoring criteria, solutions, and/or examples and to score the response. +1 indicates a point earned and -1 indicates a general or question-specific penalty.

+1 [Skill 3.B] Declares all appropriate `private` instance variables

+1 [Skill 3.B] Declares method headers: `public int spin()` and `public int currentRun()`

+1 [Skill 3.B] Declares header: `GameSpinner(int __)` (*must not be* `private`)

+1 [Skill 3.B] Constructor initializes instance variable for number of sectors using parameter. Instance variables for previous spin and length of current run initialized correctly when declared or in constructor with default values.

Responses still earn the point even if they...

· declare instance variables incorrectly.

+1 [Skill 3.A] Computes random integer [1, number of sectors]

+1 [Skill 3.C] Compares new spin and last spin to determine required updates to state

Responses still earn the point even if they...

· use an incorrectly computed random integer for new spin; or

· incorrectly declare the instance variable intended to store last spin.

+1 [Skill 3.C] Updates instance variable that represents length of current run appropriately if new spin and previous spin are the same

Responses still earn the point even if they...

· incorrectly compare new spin and last spin.

+1 [Skill 3.C] Updates previous spin and length of current run appropriately when new spin differs from the previous spin

Responses still earn the point even if they...

· incorrectly compare new spin and last spin.

+1 [Skill 3.B] `currentRun` returns updated instance variable value

Responses still earn the point even if they...

· incorrectly update instance variables in the `spin` method.

-1 (w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)

-1 (x) Local variables used but none declared

-1 (z) Void method or constructor that returns a value

AP2025 FR

Canonical Solution:

```
public class GameSpinner
{
   private int sectors;
   private int previousSpin = 0;
   private int currentLength = 0;

   public GameSpinner(int s)
   {
      sectors = s;
   }

   public int spin()
   {
      int newSpin = (int)(Math.random() * sectors)
                    + 1;

      if (newSpin == previousSpin)
      {
         currentLength++;
      }
      else
      {
         previousSpin = newSpin;
         currentLength = 1;
      }
      return newSpin;
   }

   public int currentRun()
   {
      return currentLength;
   }
}
```

0	1	2	3	4	5	6	7	8	[✓ 9]

Total number of points earned (minus penalties) is equal to 9.

- ☐ **+1** Declares all appropriate private instance variables **(points earned)**
- ☐ **+1** Declares method headers: public int spin() and public int currentRun() **(points earned)**
- ☐ **+1** Declares header: GameSpinner(int __) (*must not be* private) **(points earned)**
- ☐ **+1** Constructor initializes instance variable for number of sectors using parameter. Instance variables for previous spin and length of current run initialized correctly when declared or in constructor with default values. **(points earned)**
- ☐ **+1** Computes random integer [1, number of sectors] **(points earned)**
- ☐ **+1** Compares new spin and last spin to determine required updates to state **(points earned)**

AP2025 FR

- ☐ **+1** Updates instance variable that represents length of current run appropriately if new spin and previous spin are the same **(points earned)**
- ☐ **+1** Updates previous spin and length of current run appropriately when new spin differs from the previous spin **(points earned)**
- ☐ **+1** currentRun returns updated instance variable value **(points earned)**
- ☐ **-1** [penalty] (w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check) **(General penalties)**
- ☐ **-1** [penalty] (x) Local variables used but none declared **(General penalties)**
- ☐ **-1** [penalty] (z) Void method or constructor that returns a value **(General penalties)**

Canonical Solution:

```
public class GameSpinner
{
   private int sectors;
   private int previousSpin = 0;
   private int currentLength = 0;

   public GameSpinner(int s)
   {
      sectors = s;
   }

   public int spin()
   {
      int newSpin = (int)(Math.random() * sectors)
                    + 1;

      if (newSpin == previousSpin)
      {
         currentLength++;
      }
      else
      {
         previousSpin = newSpin;
         currentLength = 1;
      }
      return newSpin;
   }

   public int currentRun()
   {
      return currentLength;
   }
}
```

AP2025 FR

3. SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

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).
```

AP2025 FR

```
          * 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.
    }
```

(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)
```

(b) Write the `ClubMembers` method `removeMembers`, which takes the following actions.

Returns a list of all students who have graduated and are in good standing. A member has graduated if the member's graduation year is less than or equal to the method's `year` parameter. If no members meet these criteria, an empty list is returned.
Removes from `memberList` all members who have graduated, regardless of whether or not they are in good standing.

The following example illustrates the results of a call to `removeMembers`.

The `ArrayList memberList` before the method call `removeMembers(2018)`:

"SMITH, JANE"	"FOX, STEVE"	"XIN, MICHAEL"	"GARCIA, MARIA"
2019	2018	2017	2020
false	true	false	true

The `ArrayList memberList` after the method call `removeMembers(2018)`:

"SMITH, JANE"	"GARCIA, MARIA"
2019	2020
false	true

The `ArrayList` returned by the method call `removeMembers(2018)`:

"FOX, STEVE"
2018
true

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)
```

Part A – addMembers

Select a point value to view scoring criteria, solutions, and/or examples and to score the response. +1 indicates a point earned and −1 indicates a general or question-specific penalty.

+1 **[Skill 3.D]** 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 **[Skill 3.A]** Instantiates a MemberInfo object with name from array, provided year, and good standing

+1 **[Skill 3.D]** Adds MemberInfo objects to memberList (*in the context of a loop*)

Responses can still earn the point even if they instantiate MemberInfo objects incorrectly

−1 (v) Array/collection access confusion ([ ] get)

−1 (w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)

−1 (x) Local variables used but none declared

−1 (y) Destruction of persistent data (e.g., changing value referenced by

parameter)

Canonical Solution:

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

0	1	2	3 ✓

Total number of points earned (minus penalties) is equal to 3.

☐ +1 Accesses all elements of names (*no bounds errors*) **(Points Earned)**

- ☐ +1 Instantiates a MemberInfo object with name from array, provided year, and good standing **(Points Earned)**
- ☐ +1 Adds MemberInfo objects to memberList (*in the context of a loop*) **(Points Earned)**
- ☐ −1 [penalty] (v) Array/collection access confusion ([ ] get) **(General Penalties)**
- ☐ −1 [penalty] (w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check) **(General Penalties)**
- ☐ −1 [penalty] (x) Local variables used but none declared **(General Penalties)**
- ☐ −1 [penalty] (y) Destruction of persistent data (e.g., changing value referenced by parameter) **(General Penalties)**

Canonical Solution:

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

Part B – removeMembers method

Select a point value to view scoring criteria, solutions, and/or examples and to score the response. +1 indicates a point earned and -1 indicates a general or question-specific penalty.

+1 **[Skill 3.D]** 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 **[Skill 3.D]** 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 **[Skill 3.A]** 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

AP2025 FR

· ever call either method on an object other than $\mathrm{MemberInfo}$

$+1$ **[Skill 3.C]** 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

$+1$ **[Skill 3.C]** 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

$+1$ **[Skill 3.D]** Removes appropriate members from $\mathrm{memberList}$ and adds appropriate members to the $\mathrm{ArrayList}$ to be returned

Responses can still earn the point even if they

· call $\mathrm{getGradYear}$ or $\mathrm{inGoodStanding}$ incorrectly

· access elements of $\mathrm{memberList}$ incorrectly

· initialize the $\mathrm{ArrayList}$ 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 $\mathrm{ArrayList}$ to return

· fail to distinguish the correct three cases, with the exception of confusing the $<$ and $<=$ in the comparison

-1 (v) Array/collection access confusion ($[\,]$ get)

-1 (w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)

-1 (x) Local variables used but none declared

-1 (y) Destruction of persistent data (e.g., changing value referenced by parameter)

Canonical Solution:

AP2025 FR

```
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;
}
```

removeMembers(int year) Line 2: { Line 3: ArrayList removed = new ArrayList(); Line 4 is blank. Line 5: for (int i = memberList.size() – 1; i >= 0; i--) Line 6: { Line 7: if (memberList.get(i).getGradYear() <= year) Line 8: { Line 9: if (memberList.get(i).inGoodStanding()) Line 10: { Line 11: removed.add(memberList.get(i)); Line 12: } Line 13: memberList.remove(i); Line 14: } Line 15: } Line 16: return removed; Line 17: } end code">

0	1	2	3	4	5	[✓ 6]

Total number of points earned (minus penalties) is equal to 6.

- ☐ $+1$ Declares and initializes an $\mathrm{ArrayList}$ of $\mathrm{MemberInfo}$ objects **(Points Earned)**
- ☐ $+1$ Accesses all elements of $\mathrm{memberList}$ for potential removal (*no bounds errors*) **(Points Earned)**
- ☐ $+1$ Calls $\mathrm{getGradYear}$ or $\mathrm{inGoodStanding}$ **(Points Earned)**
- ☐ $+1$ Distinguishes any three cases, based on graduation status and standing **(Points Earned)**
- ☐ $+1$ Identifies graduating members **(Points Earned)**
- ☐ $+1$ Removes appropriate members from $\mathrm{memberList}$ and adds appropriate members to the $\mathrm{ArrayList}$ to be returned **(Points Earned)**
- ☐ -1 [penalty] (v) Array/collection access confusion ([ ] get) **(General Penalties)**
- ☐ -1 [penalty] (w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check) **(General Penalties)**
- ☐ -1 [penalty] (x) Local variables used but none declared **(General Penalties)**
- ☐ -1 [penalty] (y) Destruction of persistent data (e.g., changing value referenced by parameter) **(General Penalties)**

Canonical Solution:

AP2025 FR

```
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;
}
```

removeMembers(int year) Line 2: { Line 3: ArrayList<MemberInfo> removed = new ArrayList<MemberInfo>(); Line 4 is blank. Line 5: for (int i = memberList.size() – 1; i >= 0; i--) Line 6: { Line 7: if (memberList.get(i).getGradYear() <= year) Line 8: { Line 9: if (memberList.get(i).inGoodStanding()) Line 10: { Line 11: removed.add(memberList.get(i)); Line 12: } Line 13: memberList.remove(i); Line 14: } Line 15: } Line 16: return removed; Line 17: } end code">