Free-Response Section

Scoring Guidelines

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

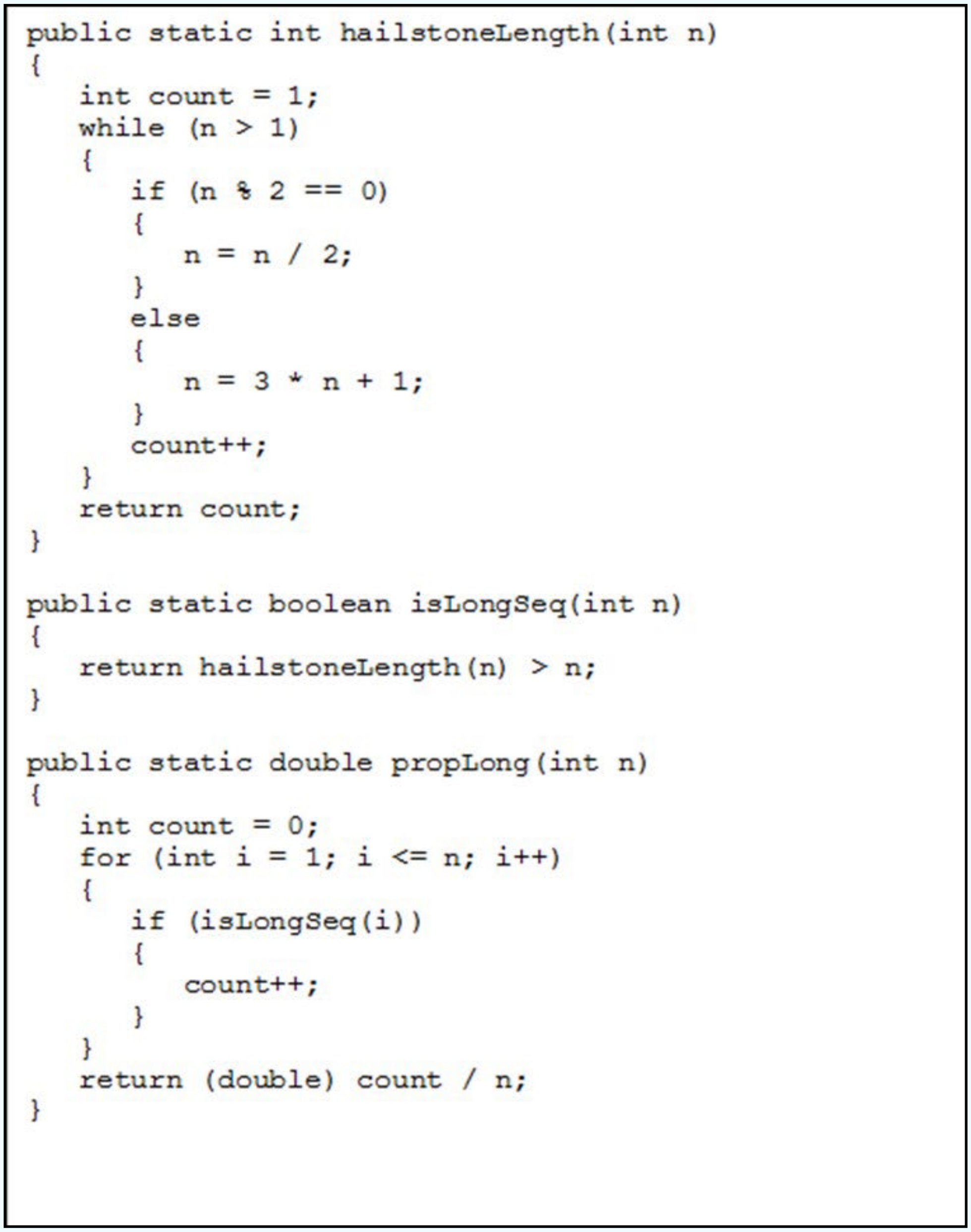
- Extraneous code with no side-effect (e.g., valid precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier 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.*

Question 1: Methods and Control Structures **9 points**

Learning Objectives: CON-1.A CON-1.C CON-1.E CON-2.A CON-2.C CON-2.E MOD-1.G MOD-2.F

Canonical solution



(a) **3 points**

```
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;
}
```

(b) **2 points**

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

(c) **4 points**

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

(a) `hailstoneLength`

	Scoring Criteria	Decision Rules	
1	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 point** **3.C** **CON-2.C**
2	Computes the next value	**Responses still earn the point even if they...** • use a correct formula in an incorrect case.	**1 point** **3.C** **CON-1.A**
3	Uses correct formula for next value depending on even/odd		**1 point** **3.C** **CON-2.A**
		Total for part (a)	**3 points**

(b) `isLongSeq`

	Scoring Criteria	Decision Rules	
4	Calls `hailstoneLength`		**1 point** **3.A** **MOD-1.G**
5	Correctly compares length and starting value to determine return value	**Responses still earn the point even if they...** • call `hailstoneLength` incorrectly.	**1 point** **3.C** **CON-1.E**
		Total for part (b)	**2 points**

(c) `propLong`

	Scoring Criteria	Decision Rules	
6	Calls `isLongSeq` in the context of a loop		**1 point** **3.A** **MOD-1.G**
7	Loops `1` to `n` (*no bounds errors*)		**1 point** **3.C** **CON-2.E**
8	Calculates `double` proportion	**Responses still earn the point even if they...** • use incorrect values for the count of long sequences or `n`.	**1 point** **3.C** **CON-1.C**
9	Returns correctly calculated value		**1 point** **3.B** **MOD-2.F**
		Total for part (c)	**4 points**

Question-specific penalties

None

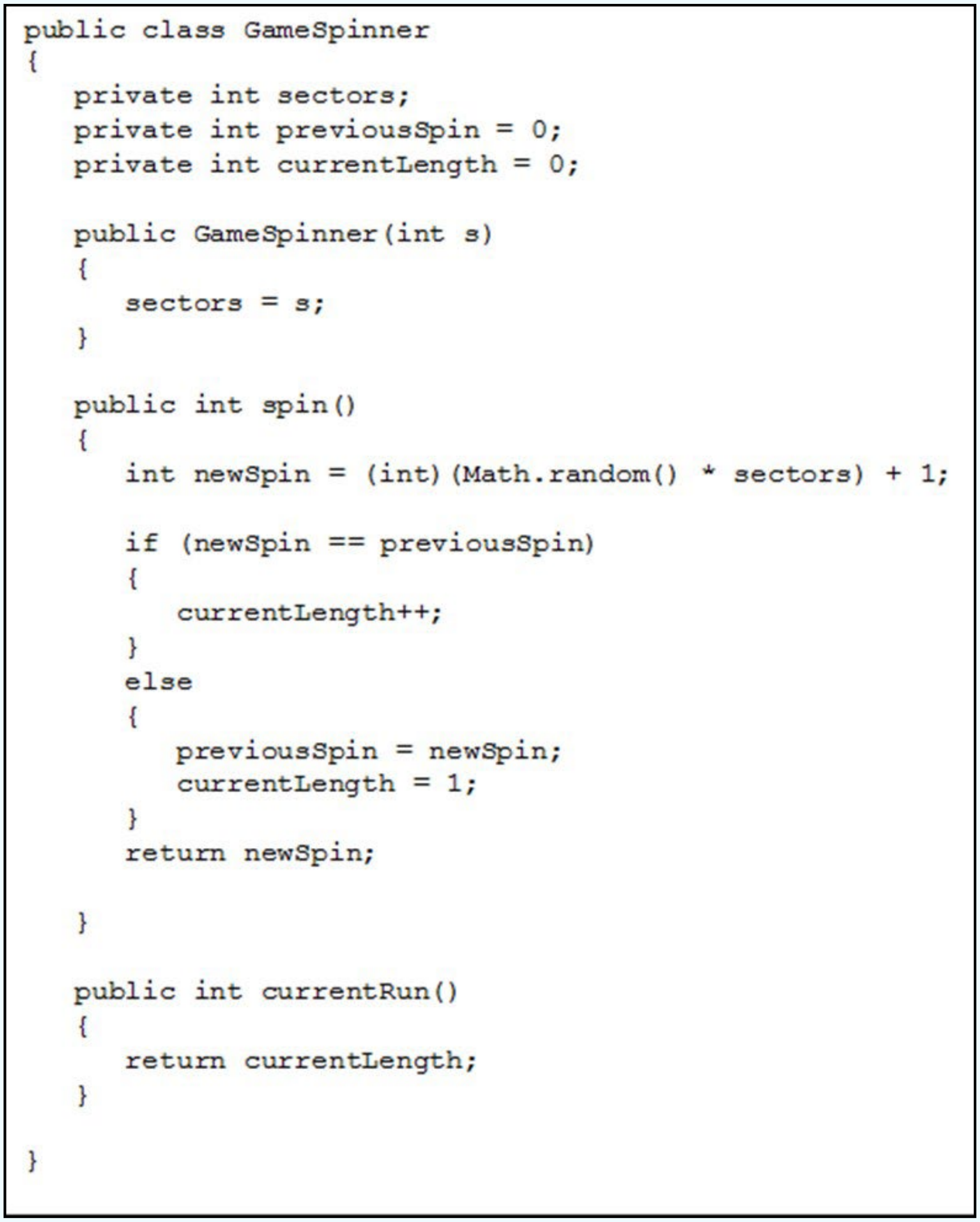
Total for question 1 **9 points**

Question 2: Class Design 9 points

Learning Objectives: MOD-2.B MOD-2.D CON-1.B CON-1.D CON-2.A

Canonical solution

9 points



```
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;
   }

}
```

`GameSpinner`

	Scoring Criteria	Decision Rules	
1	Declares all appropriate `private` instance variables		1 point 3.B MOD-2.B
2	Declares method headers: `public int spin()` and `public int currentRun()`		1 point 3.B MOD-2.D
3	Declares header: `GameSpinner(int __)` (*must not be* `private`)		1 point 3.B MOD-2.B
4	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 point 3.B MOD-2.B
5	Computes random integer [1, number of sectors]		1 point 3.A CON-1.D
6	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 point 3.C CON-2.A
7	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 point 3.B MOD-2.D
8	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 point 3.C CON-1.B
9	`currentRun` returns updated instance variable value	**Responses still earn the point even if they...** • incorrectly update instance variables in the `spin` method.	1 point 3.B MOD-2.D

Question-specific penalties

None

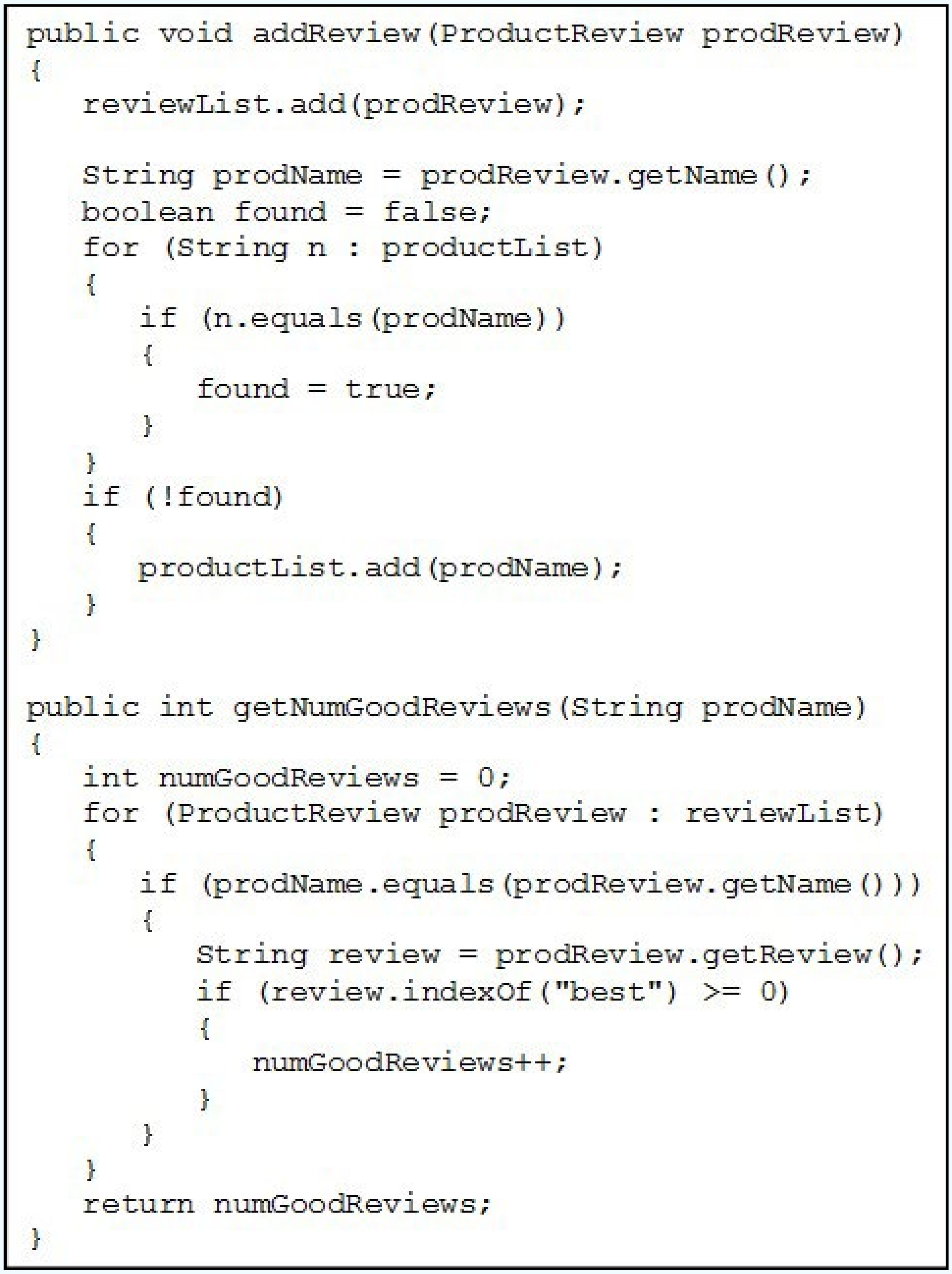
Total for question 2 **9 points**

Question 3: Array/ArrayList 9 points

Learning Objectives: VAR-1.E.b VAR-2.D VAR-2.E.a MOD-1.G CON-2.F.a CON-2.J.a CON-2.K

Canonical solution

(a) **6 points**



```
public void addReview(ProductReview prodReview)
{
   reviewList.add(prodReview);

   String prodName = prodReview.getName();
   boolean found = false;
   for (String n : productList)
   {
      if (n.equals(prodName))
      {
         found = true;
      }
   }
   if (!found)
   {
      productList.add(prodName);
   }
}
```

(b) **3 points**

```
public int getNumGoodReviews(String prodName)
{
   int numGoodReviews = 0;
   for (ProductReview prodReview : reviewList)
   {
      if (prodName.equals(prodReview.getName()))
      {
         String review = prodReview.getReview();
         if (review.indexOf("best") >= 0)
         {
            numGoodReviews++;
         }
      }
   }
   return numGoodReviews;
}
```

(a) `addReview`

	Scoring Criteria	Decision Rules	
1	Adds a `ProductReview` object to `reviewList`	**Responses still earn the point even if they...** • add a `ProductReview` object other than the one referenced by the parameter `prodReview`.	**1 point** 3.D VAR-2.D
2	Gets product name of review to be added		**1 point** 3.A MOD-1.G
3	Traverses `productList` (*no bounds errors*)	**Responses still earn the point even if they...** • use a `for`, an enhanced `for`, or a `while` loop.	**1 point** 3.D VAR-2.E.a
4	Compares name in `productList` with name from review to be added	**Responses still earn the point even if they...** • use an incorrectly accessed value for either name.	**1 point** 3.C VAR-1.E.b
5	Adds new product name to `productList`	**Responses still earn the point even if they...** • add the new product name under the wrong conditions; or • add an incorrectly accessed value for the new product name	**1 point** 3.D VAR-2.D
6	Correctly adds product name to `productList` if and only if the product name is not already in `productList`		**1 point** 3.D CON-2.K
		Total for part (a)	**6 points**

(b) `getNumGoodReviews`

	Scoring Criteria	Decision Rules	
7	Traverses `reviewList` (*no bounds errors*)	**Responses still earn the point even if they...** • use a `for`, an enhanced `for`, or a `while` loop.	**1 point** 3.D VAR-2.E.a
8	Selects all and only reviews with matching product names that contain `"best"`		**1 point** 3.C CON-2.F.a
9	Returns correct count of good reviews		**1 point** 3.D CON-2.J.a
		Total for part (b)	**3 points**
	Question-specific penalties		
	None		
		Total for question 3	**9 points**

Question 4: 2D Array 9 points

Learning Objectives: MOD-1.D.b MOD-1.G CON-1.H CON-2.A CON-2.N.c VAR-2.F VAR-2.G.a

Canonical solution

(a) **5 points**

```
public Theater(int seatsPerRow, int tier1Rows,
               int tier2Rows)
{
   theaterSeats =
      new Seat[tier1Rows + tier2Rows][seatsPerRow];
   for (int r = 0; r < tier1Rows + tier2Rows; r++)
   {
      for (int c = 0; c < seatsPerRow; c++)
      {
         if (r < tier1Rows)
         {
            theaterSeats[r][c] = new Seat(true, 1);
         }
         else
         {
            theaterSeats[r][c] = new Seat(true, 2);
         }
      }
   }
}
```

(b) **4 points**

```
public boolean reassignSeat(int fromRow, int fromCol,
                            int toRow, int toCol)
{
   Seat toS = theaterSeats[toRow][toCol];
   if (!toS.isAvailable())
   {
      return false;
   }

   Seat fromS = theaterSeats[fromRow][fromCol];
   if (toS.getTier() < fromS.getTier())
   {
      return false;
   }

   toS.setAvailability(false);
   fromS.setAvailability(true);
   return true;
}
```

(a) `Theater`

	Scoring Criteria	Decision Rules	
1	Instantiates a new `Seat[][]` with the correct number of rows and columns, based on parameters		**1 point** 3.E VAR-2.F
2	Traverses the `theaterSeats` array (*no bounds errors*)		**1 point** 3.E VAR-2.G.a
3	Instantiates a new `Seat` object with a tier and availability status	**Responses still earn the point even if they...** • incorrectly assign the new object to a `theaterSeats` element.	**1 point** 3.A MOD-1.D.b
4	Accesses a `theaterSeats` element and assigns it a new `Seat` object	**Responses still earn the point even if they...** • incorrectly instantiate the new `Seat` object; or • assign the new `Seat` object to an incorrect `theaterSeats` element.	**1 point** 3.E VAR-2.F
5	Correct tiers assigned to all array elements		**1 point** 3.C CON-2.A
		Total for part (a)	**5 points**

(b) `reassignSeat`

	Scoring Criteria	Decision Rules	
6	Accesses *from* and *to* `Seat` objects		**1 point** 3.E VAR-2.F
7	Calls `isAvailable` and `getTier` on `Seat` objects	**Responses still earn the point even if they...** • correctly call methods on `theaterSeats` elements other than the *to* and *from* seats.	**1 point** 3.A MOD-1.G
8	Checks if move can be made based on both tiers and the availability status of *to* `Seat` object		**1 point** 3.C CON-1.H
9	Correctly updates availability of both seats and returns `true` if the move can be made; otherwise, returns `false`		**1 point** 3.E CON-2.N.c
		Total for part (b)	**4 points**
	Question-specific penalties		
	None		
		Total for question 4	**9 points**