1. This question simulates birds or possibly a bear eating at a bird feeder. The following `Feeder` class contains information about how much food is in the bird feeder and simulates how much food is eaten. You will write two methods of the `Feeder` class.

```
public class Feeder
{
   /**
    *  The amount of food, in grams, currently in the bird feeder; initialized in the constructor and
    *  always greater than or equal to zero
    */
   private int currentFood;

   /**
    *  Simulates one day with numBirds birds or possibly a bear at the bird feeder,
    *  as described in part (a)
    *  Precondition: numBirds > 0
    */
   public void simulateOneDay(int numBirds)
   {  /* to be implemented in part (a) */  }

   /**
    *  Returns the number of days birds or a bear found food to eat at the feeder in this simulation,
    *  as described in part (b)
    *  Preconditions: numBirds > 0, numDays > 0
    */
   public int simulateManyDays(int numBirds, int numDays)
   {  /* to be implemented in part (b) */  }

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

(a) Write the `simulateOneDay` method, which simulates `numBirds` birds or possibly a bear at the feeder for one day. The method determines the amount of food taken from the feeder on this day and updates the `currentFood` instance variable. The simulation accounts for normal conditions, which occur 95% of the time, and abnormal conditions, which occur 5% of the time.

Under normal conditions, the simulation assumes that on any given day, only birds visit the feeder and that each bird at the feeder consumes the same amount of food. This standard amount consumed is between 10 and 50 grams of food, inclusive, in 1-gram increments. That is, on any given day, each bird might eat 10, 11, . . . , 49, or 50 grams of food. The amount of food eaten by each bird on a given day is randomly generated and each integer from 10 to 50, inclusive, has an equal chance of being chosen.

For example, a run of the simulation might predict that for a certain day under normal conditions, each bird coming to the feeder will eat 11 grams of food. If 10 birds come to the feeder on that day, then a total of 110 grams of food will be consumed.

If the simulated food consumed is greater than the amount of food in the feeder, the birds empty the feeder and the amount of food in the feeder at the end of the day is zero.

Under abnormal conditions, a bear empties the feeder and the amount of food in the feeder at the end of the day is zero.

The following examples show possible results of three calls to `simulateOneDay`.

- Example 1: If the feeder initially contains 500 grams of food, the call `simulateOneDay(12)` could result in 12 birds eating 20 grams of food each, leaving 260 grams of food in the feeder.
- Example 2: If the feeder initially contains 1,000 grams of food, the call `simulateOneDay(22)` could result in a bear eating all the food, leaving 0 grams of food in the feeder.
- Example 3: If the feeder initially contains 100 grams of food, the call `simulateOneDay(5)` could result in 5 birds attempting to eat 30 grams of food each. Since the feeder initially contains less than 150 grams of food, the feeder is emptied, leaving 0 grams of food in the feeder.

GO ON TO THE NEXT PAGE.

Complete the `simulateOneDay` method.

```
/**
 *  Simulates one day with numBirds birds or possibly a bear at the bird feeder,
 *  as described in part (a)
 *  Precondition: numBirds > 0
 */
public void simulateOneDay(int numBirds)
```

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 Feeder

private int currentFood

public void simulateOneDay(int numBirds)
public int simulateManyDays(int numBirds, int numDays)
```

(b) Write the `simulateManyDays` method. The method uses `simulateOneDay` to simulate `numBirds` birds or a bear coming to the feeder on at most `numDays` consecutive days. The simulation returns the number of days that birds or a bear found food at the feeder.

Consider the following examples.

Value of `currentFood` and Method Call	**Possible Outcomes and Resulting Return Value**

`currentFood: 2400`
 	
`simulateManyDays(10, 4)`
	Day 1: `simulateOneDay` leaves 2100 grams of food in the feeder.
	Day 2: `simulateOneDay` leaves 1650 grams of food in the feeder.
	Day 3: `simulateOneDay` leaves 1500 grams of food in the feeder.
	Day 4: `simulateOneDay` leaves 1260 grams of food in the feeder.
	
	The simulation returns `4` because, on all four days of the simulation, birds or a bear found food at the feeder. The instance variable `currentFood` has the value `1260`.

`currentFood: 250`
 	
`simulateManyDays(10, 5)`
	Day 1: `simulateOneDay` leaves 150 grams of food in the feeder.
	Day 2: `simulateOneDay` leaves 0 grams of food in the feeder.
	
	The simulation returns `2` because, on two of the five simulated days, birds or a bear found food at the feeder. The instance variable `currentFood` has the value `0`.

`currentFood: 0`
 	
`simulateManyDays(5, 10)`
	The simulation returns `0` because no food was found at the feeder on any day. The instance variable `currentFood` has the value `0`.

Complete the `simulateManyDays` method. Assume that `simulateOneDay` works as intended, regardless of what you wrote in part (a). You must use `simulateOneDay` appropriately in order to receive full credit.

```
/**
 *  Returns the number of days birds or a bear found food to eat at the feeder in this simulation,
 *  as described in part (b)
 *  Preconditions: numBirds > 0, numDays > 0
 */
public int simulateManyDays(int numBirds, int numDays)
```

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 Feeder

private int currentFood

public void simulateOneDay(int numBirds)
public int simulateManyDays(int numBirds, int numDays)
```

Question 1: Methods and Control Structures — 9 points

Canonical solution

(a) — **4 points**

```
public void simulateOneDay(int numBirds)
{
   double condition = Math.random();
   if (condition < 0.05)
   {
      currentFood = 0;
   }
   else
   {
      int eachBirdEats = (int) (Math.random() * 41) + 10;
      int totalEaten = numBirds * eachBirdEats;
      if (totalEaten > currentFood)
      {
         currentFood = 0;
      }
      else
      {
         currentFood -= totalEaten;
      }
   }
}
```

(b) — **5 points**

```
public int simulateManyDays(int numBirds, int numDays)
{
   for (int daysSoFar = 0; daysSoFar < numDays; daysSoFar++)
   {
      if (currentFood == 0)
      {
         return daysSoFar;
      }
      simulateOneDay(numBirds);
   }
   return numDays;
}
```

(a) `simulateOneDay`

	Scoring Criteria	Decision Rules	
1	Generates a random value	Responses **can** still earn the point even if they • fail to save or use the generated random value Responses **will not** earn the point if they • fail to call `Math.random`, or possibly the equivalent, at least one time • make any incorrect call to `Math.random`	**1 point**
2	Identifies two cases based on a comparison of a randomly generated value and some constant that implements a 5% probability	Responses **can** still earn the point even if they • reverse the 5/95 probability • compare `double` values using `<=` or `>=` instead of `<` or `>` • incorrectly cast the 5/95 random value, as long as a suitable range is generated and the comparison divides that range appropriately • call `Math.random` incorrectly	**1 point**
3	Generates a random integer that is uniform in the range [10, 50]	Responses **can** still earn the point even if they • call `Math.random` incorrectly	**1 point**
4	Scales for `numBirds` and subtracts appropriate amount from `currentFood` in all cases (*algorithm*)	Responses **can** still earn the point even if they • compare `double` values using `<=` or `>=` instead of `<` or `>` Responses **will not** earn the point if they • reverse the 5/95 probability • incorrectly process the 5/95 range/comparison (e.g., by incorrect casting) • exit the method while `currentFood` has a negative value	**1 point**
		Total for part (a)	**4 points**

(b) `simulateManyDays`

	Scoring Criteria	Decision Rules	
5	Calls `simulateOneDay` with value of `numBirds`	Responses **will not** earn the point if they • fail to make at least one correct call to `simulateOneDay` • make any call to `simulateOneDay` on the class or on an object other than `this` (use of `this` is optional)	**1 point**
6	Loops over the simulation method call and guards that it runs at most `numDays` times	Responses **can** still earn the point even if they • call the method that simulates one day incorrectly • count the simulated days incorrectly, as long as the loop guard would work if the count were corrected • fail to guard against calls to the simulation method when `currentFood <= 0` Responses **will not** earn the point if they • fail to count the simulated days at all	**1 point**
7	Counts the number of times that the method that simulates one day is called with food available in the feeder (*algorithm*)	Responses **can** still earn the point even if they • simulate extra days when `currentFood` is `0`, as long as the count is still correct Responses **will not** earn the point if they • fail to initialize the counter appropriately • compute the correct count but return something else • fail to test if food is available	**1 point**
8	Compares `currentFood` and `0`	Responses **can** still earn the point even if they • fail to make the comparison in the context of a loop • use the result of the comparison incorrectly	**1 point**
9	Returns any `int` value, in all cases	Responses **can** still earn the point even if they • return a constant	**1 point**
		Total for part (b)	**5 points**
		Total for question 1	**9 points**

Alternate canonical for part (b):

```
public int simulateManyDays(int numBirds, int numDays)
{
   int daysSoFar = 0;
   while (currentFood > 0 && daysSoFar < numDays)
   {
      simulateOneDay(numBirds);
      daysSoFar++;
   }

   return daysSoFar;
}
```

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.*

(a) `isWordChain`

	Scoring Criteria	Decision Rules	
1	Accesses all adjacent pairs of `wordList` elements (*no bounds errors*)	Responses **can** still earn the point even if they • also access non-adjacent pairs of elements • return early, as long as bounds and indices would otherwise support accessing all necessary pairs Responses **will not** earn the point if they • fail to access elements of `wordList` correctly	**1 point**
2	Determines whether an element of the list contains a previous element of the list	Responses **can** still earn the point even if they • make just one comparison • fail to make the comparison in the context of a loop • compare every element to the first element of the list • access pairs of `wordList` elements incorrectly Responses **will not** earn the point if they • make an incorrect call to `indexOf` or use the `indexOf` return value incorrectly	**1 point**
3	Returns appropriate `boolean` in both cases (*algorithm*)	Responses **can** still earn the point even if they • incorrectly identify whether an element of `wordList` contains the previous element Responses **will not** earn the point if they • return an incorrect value due to an early return • fail to return `true` or `false`	**1 point**
		Total for part (a)	**3 points**