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.

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)
```

4. This question involves a path through a two-dimensional (2D) array of integers, where the path is based on the values of elements in the array. When an element of the 2D array is accessed, the first index is used to specify the row and the second index is used to specify the column. The following `Location` class represents a row and column position in the 2D array.

```
public class Location
{
   private int theRow;
   private int theCol;

   public Location(int r, int c)
   {
      theRow = r;
      theCol = c;
   }

   public int getRow()
   {  return theRow;  }

   public int getCol()
   {  return theCol;  }
}
```


The following `GridPath` class contains the 2D array and methods to use to determine a path through the array. You will write two methods of the `GridPath` class.

```
public class GridPath
{
   /** Initialized in the constructor with distinct values that never change */
   private int[][] grid;

   /**
    *  Returns the Location representing a neighbor of the grid element at row and col,
    *  as described in part (a)
    *  Preconditions: row is a valid row index and col is a valid column index in grid.
    *       row and col do not specify the element in the last row and last column of grid.
    */
   public Location getNextLoc(int row, int col)
   {  /* to be implemented in part (a) */  }

   /**
    *  Computes and returns the sum of all values on a path through grid, as described in
    *  part (b)
    *  Preconditions: row is a valid row index and col is a valid column index in grid.
    *       row and col do not specify the element in the last row and last column of grid.
    */
   public int sumPath(int row, int col)
   {  /* to be implemented in part (b) */  }

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

(a) Write the `getNextLoc` method, which returns a `Location` object that represents the smaller of two neighbors of the `grid` element at `row` and `col`, according to the following rules.

- The two neighbors that are considered are the element below the given element and the element to the right of the given element, if they exist.
- If both neighbors exist, the `Location` of the neighbor with the smaller value is returned. Two neighbors will always have different values.
- If only one neighbor exists, the `Location` of the existing neighbor is returned.

For example, assume that `grid` contains the following values.

	0	1	2	3	4
0	12	3	4	13	5
1	11	21	2	14	16
2	7	8	9	15	0
3	10	17	20	19	1
4	18	22	30	25	6

The following table shows some sample calls to `getNextLoc`.

Method Call	Explanation
`getNextLoc(0, 0)`	Returns the neighbor to the right (the `Location` representing the element at row 0 and column 1), since `3 < 11`
`getNextLoc(1, 3)`	Returns the neighbor below (the `Location` representing the element at row 2 and column 3), since `15 < 16`
`getNextLoc(2, 4)`	Returns the neighbor below (the `Location` representing the element at row 3 and column 4), since the given element has no neighbor to the right
`getNextLoc(4, 3)`	Returns the neighbor to the right (the `Location` representing the element at row 4 and column 4), since the given element has no neighbor below

In the example, the `getNextLoc` method will never be called with row `4` and column `4`, as those values would violate the precondition of the method.

Complete the `getNextLoc` method.

```
/**
 *  Returns the Location representing a neighbor of the grid element at row and col,
 *  as described in part (a)
 *  Preconditions: row is a valid row index and col is a valid column index in grid.
 *       row and col do not specify the element in the last row and last column of grid.
 */
public Location getNextLoc(int row, int col)
```

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 Location

private int theRow
private int theCol

public Location(int r, int c)
public int getRow()
public int getCol()

public class GridPath

private int[][] grid

public Location getNextLoc(int row, int col)
public int sumPath(int row, int col)
```

(b) Write the `sumPath` method, which returns the sum of all values on a path in `grid`. The path begins with the element at `row` and `col` and is determined by successive calls to `getNextLoc`. The path ends when the element in the last row and the last column of `grid` is reached.

For example, consider the following contents of `grid`. The shaded elements of `grid` represent the values on the path that results from the method call `sumPath(1, 1)`. The method call returns `19` because 3 + 2 + 9 + 4 + 0 + 1 = 19.

	0	1	2	3	4
0	12	30	40	25	5
1	11	3	22	15	43
2	7	2	9	4	0
3	8	33	18	6	1

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

```
/**
 *  Computes and returns the sum of all values on a path through grid, as described in
 *  part (b)
 *  Preconditions: row is a valid row index and col is a valid column index in grid.
 *       row and col do not specify the element in the last row and last column of grid.
 */
public int sumPath(int row, int col)
```

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 Location

private int theRow
private int theCol

public Location(int r, int c)
public int getRow()
public int getCol()

public class GridPath

private int[][] grid

public Location getNextLoc(int row, int col)
public int sumPath(int row, int col)
```

STOP

END OF EXAM

1. This question involves reasoning about a simulation of a frog hopping in a straight line. The frog attempts to hop to a goal within a specified number of hops. The simulation is encapsulated in the following `FrogSimulation` class. You will write two of the methods in this class.

```
public class FrogSimulation
{
   /** Distance, in inches, from the starting position to the goal. */
   private int goalDistance;

   /** Maximum number of hops allowed to reach the goal. */
   private int maxHops;


   /** Constructs a FrogSimulation where dist is the distance, in inches, from the starting
    *  position to the goal, and numHops is the maximum number of hops allowed to reach the goal.
    *  Precondition: dist > 0; numHops > 0
    */
   public FrogSimulation(int dist, int numHops)
   {
      goalDistance = dist;
      maxHops = numHops;
   }


   /** Returns an integer representing the distance, in inches, to be moved when the frog hops.
    */
   private int hopDistance()
   {  /* implementation not shown */  }


   /** Simulates a frog attempting to reach the goal as described in part (a).
    *  Returns true if the frog successfully reached or passed the goal during the simulation;
    *          false otherwise.
    */
   public boolean simulate()
   {  /* to be implemented in part (a) */  }


   /** Runs num simulations and returns the proportion of simulations in which the frog
    *  successfully reached or passed the goal.
    *  Precondition: num > 0
    */
   public double runSimulations(int num)
   {  /* to be implemented in part (b) */  }
```

(a) Write the `simulate` method, which simulates the frog attempting to hop in a straight line to a goal from the frog's starting position of 0 within a maximum number of hops. The method returns `true` if the frog successfully reached the goal within the maximum number of hops; otherwise, the method returns `false`.

The `FrogSimulation` class provides a method called `hopDistance` that returns an integer representing the distance (positive or negative) to be moved when the frog hops. A positive distance represents a move toward the goal. A negative distance represents a move away from the goal. The returned distance may vary from call to call. Each time the frog hops, its position is adjusted by the value returned by a call to the `hopDistance` method.

The frog hops until one of the following conditions becomes true:

- The frog has reached or passed the goal.
- The frog has reached a negative position.
- The frog has taken the maximum number of hops without reaching the goal.

The following example shows a declaration of a `FrogSimulation` object for which the goal distance is 24 inches and the maximum number of hops is 5. The table shows some possible outcomes of calling the `simulate` method.

```
FrogSimulation sim = new FrogSimulation(24, 5);
```

	Values returned by `hopDistance()`	Final position of frog	Return value of `sim.simulate()`
Example 1	`5, 7, -2, 8, 6`	24	`true`
Example 2	`6, 7, 6, 6`	25	`true`
Example 3	`6, -6, 31`	31	`true`
Example 4	`4, 2, -8`	-2	`false`
Example 5	`5, 4, 2, 4, 3`	18	`false`

Class information for this question

```
public class FrogSimulation

private int goalDistance
private int maxHops

private int hopDistance()
public boolean simulate()
public double runSimulations(int num)
```

Complete method `simulate` below. You must use `hopDistance` appropriately to receive full credit.

```
/** Simulates a frog attempting to reach the goal as described in part (a).
```


(b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is `400`, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return `0.25`.

Complete method `runSimulations` below. Assume that `simulate` works as specified, regardless of what you wrote in part (a). You must use `simulate` appropriately to receive full credit.

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
```