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)
```

```java
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) 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)
```

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

2. This question involves a scoreboard for a game. The game is played between two teams who alternate turns so that at any given time, one team is active and the other team is inactive. During a turn, a team makes one or more plays. Each play can score one or more points and the team's turn continues, or the play can fail, in which case no points are scored and the team's turn ends. The `Scoreboard` class, which you will write, is used to keep track of the score in a game.

   The `Scoreboard` class contains a constructor and two methods.

   - The constructor has two parameters. The first parameter is a `String` containing the name of team 1, and the second parameter is a `String` containing the name of team 2. The game always begins with team 1 as the active team.
   - The `recordPlay` method has a single nonnegative integer parameter that is equal to the number of points scored on a play or `0` if the play failed. If the play results in one or more points scored, the active team's score is updated and that team remains active. If the value of the parameter is `0`, the active team's turn ends and the inactive team becomes the active team. The `recordPlay` method does not return a value.
   - The `getScore` method has no parameters. The method returns a `String` containing information about the current state of the game. The returned string begins with the score of team 1, followed by a hyphen (`"-"`), followed by the score of team 2, followed by a hyphen, followed by the name of the team that is currently active.

```
public class ScoreBoard {

    private String team1name;
    private String team2name;
    private int whoseTurn;
    private int score1;
    private int score2;

    public ScoreBoard(String team1name, String team2name) {
        this.team1name = team1name;
        this.team2name = team2name;
        this.whoseTurn = 0; // 0 for team1, 1 for team2
        this.score1 = 0;
        this.score2 = 0;
    }

    public void recordPlay (int points) {
        if (points == 0) {
            if (whoseTurn == 1) {
                whoseTurn = 2;
            } else {
                whoseTurn = 1;
            }
        } else {
            if (whoseTurn == 1) {
                score1 += points;
            } else {
                score2 += points;}
            }
    }

    public String getScore() {
        String result = score1 + " - " + score2 + "-";
        if (whoseTurn == 1) {
            result += team1name;
        } else {
            result += team2name;
        }
        return result;
    }
}
```

GO ON TO THE NEXT PAGE.

The following table contains a sample code execution sequence and the corresponding results. The code execution sequence appears in a class other than `Scoreboard`.

Statement	Value Returned (blank if none)	Explanation
`String info;`		
`Scoreboard game = new Scoreboard("Red", "Blue");`		`game` is a new `Scoreboard` for a game played between team 1, whose name is `"Red"`, and team 2, whose name is `"Blue"`. The active team is set to team 1.
`info = game.getScore();`	`"0-0-Red"`	
`game.recordPlay(1);`		Team 1 earns 1 point because the game always begins with team 1 as the active team.
`info = game.getScore();`	`"1-0-Red"`	
`game.recordPlay(0);`		Team 1's play failed, so team 2 is now active.
`info = game.getScore();`	`"1-0-Blue"`	
`info = game.getScore();`	`"1-0-Blue"`	The score and state of the game are unchanged since the last call to `getScore`.
`game.recordPlay(3);`		Team 2 earns 3 points.
`info = game.getScore();`	`"1-3-Blue"`	
`game.recordPlay(1);`		Team 2 earns 1 point.
`game.recordPlay(0);`		Team 2's play failed, so team 1 is now active.
`info = game.getScore();`	`"1-4-Red"`	
`game.recordPlay(0);`		Team 1's play failed, so team 2 is now active.
`game.recordPlay(4);`		Team 2 earns 4 points.
`game.recordPlay(0);`		Team 2's play failed, so team 1 is now active.
`info = game.getScore();`	`"1-8-Red"`	
`Scoreboard match = new Scoreboard("Lions", "Tigers");`		`match` is a new and independent `Scoreboard` object.
`info = match.getScore();`	`"0-0-Lions"`	
`info = game.getScore();`	`"1-8-Red"`	

Write the complete `Scoreboard` class. Your implementation must meet all specifications and conform to the examples shown in the preceding table.

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.

GO ON TO THE NEXT PAGE.

3. This question involves the manipulation and analysis of a list of words. The following `WordChecker` class contains an `ArrayList<String>` to be analyzed and methods that are used to perform the analysis. You will write two methods of the `WordChecker` class.

```
public class WordChecker
{
   /** Initialized in the constructor and contains no null elements */
   private ArrayList<String> wordList;

   /**
    *  Returns true if each element of wordList (except the first) contains the previous
    *  element as a substring and returns false otherwise, as described in part (a)
    *  Precondition: wordList contains at least two elements.
    *  Postcondition: wordList is unchanged.
    */
   public boolean isWordChain()
   {  /* to be implemented in part (a) */  }

   /**
    *  Returns an ArrayList<String> based on strings from wordList that start
    *  with target, as described in part (b). Each element of the returned ArrayList has had
    *  the initial occurrence of target removed.
    *  Postconditions: wordList is unchanged.
    *      Items appear in the returned list in the same order as they appear in wordList.
    */
   public ArrayList<String> createList(String target)
   {  /* to be implemented in part (b) */  }

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

(a) Write the `isWordChain` method, which determines whether each element of `wordList` (except the first) contains the previous element as a substring. The following table shows two sample `isWordChain` method calls.

`wordList`	`isWordChain` **Return Value**	**Explanation**
`["an", "band", "band", "abandon"]`	`true`	Each element contains the previous element as a substring.
`["to", "too", "stool", "tools"]`	`false`	`"tools"` does not contain the substring `"stool"`.

Complete the `isWordChain` method.

```
/**
 *  Returns true if each element of wordList (except the first) contains the previous
 *  element as a substring and returns false otherwise, as described in part (a)
 *  Precondition: wordList contains at least two elements.
 *  Postcondition: wordList is unchanged.
 */
public boolean isWordChain()
```

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.

```
public boolean isWordChain() {
    for (int i = 1; i < wordList.size(); i++) {
        String current = wordList.get(i);
        String previous = wordList.get(i - 1);
    }
    if (current.indexOf(previous) == -1) {
        return false;
    }
    return true;
}
```

(b) Write the `createList` method, which creates and returns an `ArrayList<String>`. The method identifies strings in `wordList` that start with `target` and returns a new `ArrayList` containing each identified string without the starting occurrence of `target`. Elements must appear in the returned list in the same order as they appear in `wordList`.

Consider an example where `wordList` contains the following strings.

```
["catch", "bobcat", "catchacat", "cat", "at"]
```

The following table shows the `ArrayList` returned by some calls to `createList`. In all cases, `wordList` is unchanged.

Method Call	`ArrayList` **Returned by** `createList`	**Explanation**
`createList("cat")`	`["ch", "chacat", ""]`	Only `"catch"`, `"catchacat"`, and `"cat"` begin with `"cat"`.
`createList("catch")`	`["", "acat"]`	Only `"catch"` and `"catchacat"` begin with `"catch"`.
`createList("dog")`	`[]`	None of the words in `wordList` begin with `"dog"`.

Complete the `createList` method.

```
/**
 *  Returns an ArrayList<String> based on strings from wordList that start
 *  with target, as described in part (b). Each element of the returned ArrayList has had
 *  the initial occurrence of target removed.
 *  Postconditions: wordList is unchanged.
 *      Items appear in the returned list in the same order as they appear in wordList.
 */
public ArrayList<String> createList(String target)
```

```
public ArrayList<String> createList(String target) {
    ArrayList<String> result = new ArrayList<String>();

    for (String current : wordList) {
        if (current.indexOf(target) == 0) {
            String newStr = current.substring(target.length());
            result.add(newStr);
        }
    }
    return result;
}
```

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.

GO ON TO THE NEXT PAGE.

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)
```

```
public Location getNextLoc(int row, int col) {
    if (row == grid.length - 1) {
        return new Location(row, col + 1);
    } else if (cold == grid[0].length - 1) {
        return new Location(row + 1, 0);
    } else if (grid[row + 1][col] < grid[row][col + 1]) {
        return new Location(row + 1, col);
    } else {
        return new Location(row, col + 1);
    }
}
```

GO ON TO THE NEXT PAGE.

(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

```
public int sumPath(int row, int col) {
    int sum = 0;
    while (row < grid.length - 1 || col < grid[0].length - 1) {
        sum += grid[row][col];
        Location loc = getNextLoc(row, col);
        row = loc.getRow();
        col = loc.getCol();
    }
    return sum + grid[row][col];
}
```

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)
```