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).
 *  Returns true if the frog successfully reached or passed the goal during the simulation;
 *          false otherwise.
 */
public boolean simulate()
```

(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.
 *  Precondition: num > 0
 */
public double runSimulations(int num)
```

2. This question involves reasoning about pairs of words that are represented by the following `WordPair` class.

```
public class WordPair
{

    /** Constructs a WordPair object. */
    public WordPair(String first, String second)
    {  /* implementation not shown */  }


    /** Returns the first string of this WordPair object. */
    public String getFirst()
    {  /* implementation not shown */  }


    /** Returns the second string of this WordPair object. */
    public String getSecond()
    {  /* implementation not shown */  }

}
```

You will implement the constructor and another method for the following `WordPairList` class.

```
public class WordPairList
{

    /** The list of word pairs, initialized by the constructor. */
    private ArrayList<WordPair> allPairs;


    /** Constructs a WordPairList object as described in part (a).
     *  Precondition: words.length >= 2
     */
    public WordPairList(String[] words)
    { /* to be implemented in part (a) */ }


    /** Returns the number of matches as described in part (b).
     */
    public int numMatches()
    { /* to be implemented in part (b) */ }

}
```

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES

Question 1: Frog Simulation

Part (a)	`simulate`	**5 points**

Intent: *Simulate the distance traveled by a hopping frog*

+1 Calls `hopDistance` and uses returned distance to adjust (or represent) the frog's position

+1 Initializes and accumulates the frog's position at most `maxHops` times (*must be in context of a loop*)

+1 Determines if a distance representing multiple hops is at least `goalDistance`

+1 Determines if a distance representing multiple hops is less than starting position

+1 Returns `true` if goal ever reached, `false` if goal never reached or position ever less than starting position

Part (b)	`runSimulations`	**4 points**

Intent: *Determine the proportion of successful frog hopping simulations*

+1 Calls `simulate` the specified number of times (*no bounds errors*)

+1 Initializes and accumulates a count of `true` results

+1 Calculates proportion of successful simulations using `double` arithmetic

+1 Returns calculated value

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES

Question 1: Scoring Notes

Part (a) `simulate`			**5 points**
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Calls `hopDistance` and uses returned distance to adjust (or represent) the frog's position	• use `hopDistance()` as a position, like `hopDistance() < 0`	• only use `hopDistance()` as a count, like `hopDistance() < maxHops`
+1	Initializes and accumulates the frog's position at most `maxHops` times (*must be in context of a loop*)		• do not use a loop
+1	Determines if a distance representing multiple hops is at least `goalDistance`	• use some number of hops `*` `hopDistance()` as the frog's final position	
+1	Determines if a distance representing multiple hops is less than starting position		
+1	Returns `true` if goal ever reached, `false` if goal never reached or position ever less than starting position	• have checks for all three conditions and correct return logic based on those checks, even if a check did not earn a point	• do not check all three conditions • only check for `goalDistance` after the loop • only check for starting position after the loop
Part (b) `runSimulations`			**4 points**
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Calls `simulate` the specified number of times (*no bounds errors*)	• do not use the result of calling `simulate`	• do not use a loop
+1	Initializes and accumulates a count of `true` results		• initialize the count inside a loop • do not use a loop
+1	Calculates proportion of successful simulations using `double` arithmetic	• perform the correct calculation on an accumulated value, even if there was an error in the accumulation	• fail to divide by the parameter
+1	Returns calculated value		• calculate values using nonnumeric types • return a count of simulations

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES

Question 1: Frog Simulation

Part (a)

```
public boolean simulate()
{
   int position = 0;

   for (int count = 0; count < maxHops; count++)
   {
      position += hopDistance();
      if (position >= goalDistance)
      {
         return true;
      }
      else if (position < 0)
      {
         return false;
      }
   }
   return false;
}
```

Part (b)

```
public double runSimulations(int num)
{
   int countSuccess = 0;

   for (int count = 0; count < num; count++)
   {
      if(simulate())
      {
         countSuccess++;
      }
   }
   return (double)countSuccess / num;
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.