**Question 1**

**25 minutes to complete, 5 minutes to upload answer.**

1. This question involves the implementation and extension of a RandomStringChooser class.

(a) A RandomStringChooser object is constructed from an array of non-null String values. When the object is first constructed, all of the strings are considered available. The RandomStringChooser class has a getRandomSequence method, which has the following behavior. A call to getRandomSequence returns a String[] array containing a random sequence of the available strings in the object.

The following code segment shows an example of the behavior of RandomStringChooser.

String[] wordArray = {"wheels", "on", "the", "bus"};

String[] sChooser = new RandomStringChooser(wordArray);

for (String str : sChooser.getRandomSequence())

{

System.out.print(str + " ");

}

One possible output is shown below.

bus the wheels on

**WRITE YOUR SOLUTION ON THE NEXT PAGE.**

Write the entire RandomStringChooser class. Your implementation must include an appropriate constructor and any necessary methods. Any instance variables must be private. The code segment in the example above should have the indicated behavior (that is, it must compile and produce a result like the possible output shown). Neither the constructor nor any of the methods should alter the parameter passed to the constructor, but your implementation may copy the contents of the array.

(b) The following partially completed RandomLetterChooser class is a subclass of the RandomStringChooser class. You will write the constructor for the RandomLetterChooser class.

public class RandomLetterChooser extends RandomStringChooser

{

/\*\* Constructs a random letter chooser using the given string str.

\* **Precondition**:

\* str contains only letters.  
 \*/

public RandomLetterChooser(String str)

{

super( getSingleLetters(str) );

}

/\*\* Returns an array of single-letter strings.

\* Each of these strings consists of a single letter from str.

\* Element k of the returned array contains the single letter at

\* position k of str.

\* For example, getSingleLetters("cat") returns the

\* array { "c", "a", "t" }.

\*/

public static String[] getSingleLetters(String str)

{ /\* to be implemented in part (b) \*/ }

}

The following code segment shows an example of using RandomLetterChooser.

RandomLetterChooser letterChooser = new RandomLetterChooser("cat");

String[] strArray = letterChooser.getRandomSequence();

for (int k = 0; k < strArray.length; k++)

{

System.out.print(strArray[k]);

}

The code segment will print the three letters in "cat" in one of the possible orders. One possible output is shown below.

act

**WRITE YOUR SOLUTION ON THE NEXT PAGE.**

Complete the getSingleLetters method below.

/\*\* Returns an array of single-letter strings.

\* Each of these strings consists of a single letter from str. Element k

\* of the returned array contains the single letter at position k of str.

\* For example, getSingleLetters("cat") returns the

\* array { "c", "a", "t" }.

\*/

public static String[] getSingleLetters(String str)

**Question 2**

**15 minutes to complete, 5 minutes to upload answer.**

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 the simulate method in this class.

public class FrogSimulation

{

/\*\* Distance in cm, 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 cm, 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 cm, to be moved

\* when the frog hops.  
 \* Returns a HopRecord object containing hop and success data.  
 \*/

public int hopDistance()

{ /\* implementation not shown \*/ }

/\*\* Simulates a frog attempting to reach the goal.  
 \* Returns a HopRecord object containing hop distances and success

\* indication.  
 \*/

public HopsRecord simulate()

{ /\* **to be implemented** \*/ }

}

public class HopsRecord

{

// instance variables and some methods not shown

/\*\* Constructs a HopRecord object.\*/

public HopsRecord()

{ /\* implementation not shown \*/ }

/\*\* Adds a hop distance to the record.\*/

public void addHop(int hop)

{ /\* implementation not shown \*/ }

/\*\* Sets the variable indicating whether the frog successfully

\* reached the goal.

\*/

public void setSuccess(boolean success)

{ /\* implementation not shown \*/ }

/\*\* Returns the total sum of all hops.\*/

public int getSum ()

{ /\* implementation not shown \*/ }

/\*\* Returns the number of hops that have occured.\*/

public int getCount ()

{ /\* implementation not shown \*/ }

}

(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 a HopsRecord object containing a list of each hop and a variable indicating whether the frog successfully reached the goal.

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.

If one of the above conditions becomes true, then the simulate method passes true or false (representing success or failure) to the HopsRecord object, and returns the object.

**WRITE YOUR SOLUTION ON THE NEXT PAGE.**

Complete the simulate method below.

/\*\* Simulates a frog attempting to reach the goal.  
 \* Returns a HopRecord object containing hop distances and success indication.  
 \*/

public HopsRecord simulate()