```
Enter a triangular number! ===>>  21
[3, 7, 1, 6, 4]
[2, 6, 5, 3, 5]
[1, 5, 4, 2, 4, 5]
[4, 3, 1, 3, 4, 6]
[3, 2, 2, 3, 5, 6]
[2, 1, 1, 2, 4, 5, 6]
[1, 1, 3, 4, 5, 7]
[2, 3, 4, 6, 6]
[1, 2, 3, 5, 5, 5]
[1, 2, 4, 4, 4, 6]
[1, 3, 3, 3, 5, 6]
[2, 2, 2, 4, 5, 6]
[1, 1, 1, 3, 4, 5, 6]
[2, 3, 4, 5, 7]
[1, 2, 3, 4, 6, 5]
```

```
import java.util.ArrayList;
import java.util.Random;
import java.util.Scanner;

public class ArrayListExercises {

	public static void main(String[] args) {

		Scanner input = new Scanner(System.in);
		System.out.print("Enter a triangular number! ===>>  ");
		int upperBound = input.nextInt();
		input.close();

		bulgarianSolitaire(upperBound);
		// You do not need to handle the User Interface (UI).
		// Instead you can run the JUnit test cases found in
		// ArrayListExercisesTests.java

	}

	/**
	 * Removes all of the strings of even length from the given list
	 *
	 * @param listOfStrings the list of Strings (list can be empty)
	 * @return the given list with all even length strings removed
	 */
	public static ArrayList<String> removeEvenLength(ArrayList<String>
listOfStrings) {
		for (int i = listOfStrings.size() - 1; i >= 0; i--) {
			if (listOfStrings.get(i).length() % 2 == 0) {
				listOfStrings.remove(i);
			}
		}
		return listOfStrings; // This return statement should be last
	}

	/**
	 * Moves the minimum value in the list to the front, otherwise
preserving the
	 * order of the elements
	 *
	 * @param listOfIntegers the list of Integers (list cannot be
empty)
	 * @return the given list with the minimum value in the front
(zeroth element)
	 */
	public static ArrayList<Integer> minimumToFront(ArrayList<Integer>
listOfInts) {
		int min = listOfInts.get(0);
		int minIndex = 0;
		for (int i = 1; i < listOfInts.size(); i++) {
			if (min > listOfInts.get(i)) {
				min = listOfInts.get(i);
				minIndex = i;
```

```
                }
            }
            listOfInts.remove(minIndex);
            listOfInts.add(0, min);
            return listOfInts; // This return statement should be last
      }

      /**
       * Removes all elements from the given list whose values are in the
range min
       * through max (inclusive). If no elements in range min-max are
found in the
       * list, the list's contents are unchanged. If an empty list is
passed, the list
       * remains empty. Assume min < max.
       *
       * @param listOfInts the list of Integers (list can be empty)
       * @param min        the minimum value in the range
       * @param max        the maximum value in the range
       * @return the given list with the range min-max removed
       */
      public static ArrayList<Integer> filterRange(ArrayList<Integer>
listOfInts, int min, int max) {
            for (int i = 0; i < listOfInts.size(); i++) {
                  if (listOfInts.get(i) >= min && listOfInts.get(i) <=
max) {
                        listOfInts.remove(i);
                        i--;
                  }
            }
            return listOfInts; // This return statement should be last
      }

      /**
       * Models/simulates the game of Bulgarian Solitaire.
       *
       * @param numCards the number of cards to start with; n must be a
triangular
       *                 number (a triangular number is a number that can
be written
       *                 as the sum of the first n positive integers).
       */
      public static void bulgarianSolitaire(int numCards) {

            // Check if given number of cards is triangular
            int n = (int) Math.sqrt(2 * numCards);
            if (n * (n + 1) / 2 != numCards) {
                  System.out.println(numCards + " is not triangular");
                  return;
            }
            ArrayList<Integer> Completion = new ArrayList<>();
            ArrayList<Integer> piles = new ArrayList<>();
            for (int i = 1, tempCard = numCards; tempCard != 0; i++) {
                  tempCard = tempCard - i;
```

```
            Completion.add(i);
        }
        Random randy = new Random();

        while (numCards != 0) {
            int pile = randy.nextInt(numCards) + 1;
            numCards = numCards - pile;
            piles.add(pile);
        }

        while (!piles.containsAll(Completion)) {
            System.out.println(piles);
            int count = piles.size();
            for (int i = 0; i < piles.size(); i++) {
                piles.set(i, piles.get(i) - 1);
            }
            for (int i = 0; i < piles.size(); i++) {
                if (piles.get(i) == 0) {
                    piles.remove(i);
                    i--;
                }
            }
            piles.add(count);
        }
        System.out.println(piles);
    }
}
```