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

public class ArrayListExercises {

	public static void main(String[] args) {

		bulgarianSolitaire(45);
		// 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(listOfStrings.get(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);
		for (int i=1;i<listOfInts.size();i++) {
			if (listOfInts.get(i)<min) {
				min= listOfInts.get(i);
			}
		}
```

```
                listOfInts.remove((Integer) min);
            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= listOfInts.size()-1;i>=0; i--) {
                        if (listOfInts.get(i)>=min &&
listOfInts.get(i)<=max) {
                                listOfInts.remove(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;
        }
                // starting config
```

```
        ArrayList<Integer> piles = new ArrayList<>();
        Random rand = new Random();
        int remainCards = numCards;

        while (remainCards > 0) {
            int pileSize = rand.nextInt(remainCards) + 1;
            piles.add(pileSize);
            remainCards -= pileSize;
        }
        System.out.println("starting config: " + piles);

        // target confiig
        ArrayList<Integer> targetConfig = new ArrayList<>();
        for (int i = 1; i <= n; i++) {
            targetConfig.add(i);
        }

        // solitaire steps
        while (true) {
            ArrayList<Integer> sortedPiles = new ArrayList<>(piles);
            sortedPiles.sort(null);
            if (sortedPiles.containsAll(targetConfig)) {
                break;
            }

            ArrayList<Integer> newPiles = new ArrayList<>();
            int newPileSize = 0;

            for (int pile : piles) {
                if (pile > 1) {
                    newPiles.add(pile - 1);
                }
                newPileSize++;
            }

            // addding the new pile from the removed cards
            newPiles.add(newPileSize);

            piles = newPiles;
            System.out.println(piles);
        }

        System.out.println("Final config: " + piles);

        }
}
```