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

public class ArrayListExercises {

	public static void main(String[] args) {

		// You do not need to handle the User Interface (UI).
		// Instead you can run the JUnit test cases found in
		// ArrayListExercisesTests.java
		bulgarianSolitaire(10);
	}

	/**
	 * 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 = 0; i < listOfStrings.size(); i++) {
			if (listOfStrings.get(i).length() % 2 == 0) {
				listOfStrings.remove(i);
				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 num : listOfInts) {
			if (num < min) {
				min = num;
			}
```

```
            }
            listOfInts.remove(new 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 = 0; i < listOfInts.size(); i++) {
                if (listOfInts.get(i) <= max && listOfInts.get(i) >= min) {
                    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> piles = new ArrayList<Integer>();
```

```
		int range = numCards;
		Random randy = new Random();
		int num = 0;
		while (range > 0) {
			num = randy.nextInt(range) + 1;
			range -= num;
			piles.add(num);
		} // adds cards

		boolean didConverge = false;
		if (numCards == 1) {
			didConverge = true;
		}
		while (!didConverge) {
			for (int i = 0; i < piles.size(); i++) {
				if (piles.get(i) == 0) {
					piles.remove(i);
					i--;
				} else {
					piles.set(i, piles.get(i) - 1);
				}
			}
			piles.add(piles.size());
			piles.sort(null);
			System.out.println(piles);

			for (int j = 1; j < piles.size(); j++) {
				if (piles.size() == n && piles.get(j) - 1 == piles.get(j - 1)) {
					didConverge = true;
				} else {
					didConverge = false;
					break;
				}
			}
		}
		System.out.print("TADA");

	}
}
```