```java
import java.util.ArrayList;

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
        Scanner scan = new Scanner(System.in);
        System.out.print("Enter a number of cards: ");
        int numCards = scan.nextInt();
        bulgarianSolitaire(numCards);
        scan.close();



    }


    /**
     * 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 = i -1;
            }
        }

        return listOfStrings;   // This return statement should be last
    }

    /**
```

```
40      * Moves the minimum value in the list to the front,
   otherwise preserving the order of the elements
41      * @param listOfIntegers the list of Integers (list cannot
   be empty)
42      * @return the given list with the minimum value in the
   front (zeroth element)
43      */
44     public static ArrayList<Integer>
   minimumToFront(ArrayList<Integer> listOfInts)  {
45
46         int min = listOfInts.get(0);
47         int index = 0;
48         for(int i = 0; i < listOfInts.size(); i++) {
49             if (listOfInts.get(i) < min) {
50                 min = listOfInts.get(i);
51                 index = i;
52             }
53         }
54         listOfInts.remove(index);
55         listOfInts.add(0, min);
56
57
58
59         return listOfInts;   // This return statement should be
   last
60     }
61
62     /**
63      * Removes all elements from the given list whose values
   are in the range min through max (inclusive).
64      * If no elements in range min-max are found in the list,
   the list's contents are unchanged.
65      * If an empty list is passed, the list remains empty.
   Assume min < max.
66      * @param listOfInts the list of Integers (list can be
   empty)
67      * @param min the minimum value in the range
68      * @param max the maximum value in the range
69      * @return the given list with the range min-max removed
70      */
71     public static ArrayList<Integer>
   filterRange(ArrayList<Integer> listOfInts, int min, int max)  {
```

```
72
73
74          ArrayList<Integer> minMaxList = new
    ArrayList<Integer>();
75          for (int i = min;i <= max; i++) {
76              minMaxList.add(i);
77          }
78          listOfInts.removeAll(minMaxList);
79          return listOfInts;   // This return statement should be
    last
80      }
81
82
83      /**
84       * Models/simulates the game of Bulgarian Solitaire.
85       * @param numCards the number of cards to start with; n
    must be a triangular number (a triangular
86       * number is a number that can be written as the sum of the
    first n positive integers).
87       */
88      public static void bulgarianSolitaire(int numCards)  {
89
90          // Check if given number of cards is triangular
91          int n = (int) Math.sqrt(2*numCards);
92          if (n*(n+1)/2 != numCards)  {
93              System.out.println(numCards + " is not
    triangular");
94              return;
95          }
96          ArrayList<Integer> piles = new ArrayList<Integer>();
97          ArrayList<Integer> finalConfig = new
    ArrayList<Integer>();
98          Random random = new Random();
99
100         // Randomize piles for initial configuration
101         int cardCount = numCards;
102         while (cardCount != 0) {
103             int cardsForPile = random.nextInt(cardCount) + 1;
104             piles.add(cardsForPile);
105             cardCount -= cardsForPile;
106         }
107
```

```
108         // Print start setup
109         System.out.println(piles);
110 
111         // Create final configuration
112         cardCount = numCards;
113         for (int i = 1; cardCount > 0; i++) {
114             finalConfig.add(i);
115             cardCount -= i;
116         }
117 
118         // Create an array to remove piles with zero for later
119         ArrayList<Integer> zeroRemove = new
    ArrayList<Integer>();
120         zeroRemove.add(0);
121 
122         // Doing this long check in the while loop to check if
    piles match
123         // without ever using sorting. Sorting can make the
    runtime longer.
124         while (!(piles.containsAll(finalConfig) && piles.size()
    == finalConfig.size())) {
125             int count = 0;
126             for (int i = 0; i < piles.size(); i++) {
127 
128                 // Skipping over piles with zero for now, they
    will be removed later
129                 if (piles.get(i) == 0)
130                     continue;
131 
132                 if (piles.get(i) >= 1)
133                     piles.set(i, piles.get(i) - 1);
134 
135                 count++;
136             }
137             piles.add(count);
138 
139             // Remove zeroes after each step
140             piles.removeAll(zeroRemove);
141             System.out.println(piles);
142         }
143     }
144 }
```