Code (note: the following code is part of a larger file, ArrayListExercises.java, but those lines of code are hidden as they do not code for Bulgarian Solitaire):

```
import java.util.ArrayList;
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(45); //driver code - change to any positive
triangular number
    }
    /**
     * Helper function for bulgarianSolitaire
     * Checks if an ArrayList has repeated elements
     * @param values the given ArrayList
     * @return whether or not the given ArrayList has repeats
     */
    public static boolean hasRepeats(ArrayList<Integer> values)
    {
       for (int value: values)
       {
             int count = 0;
             for (int reference: values)
             {
                   if (value == reference)
                   {
                         count++;
                   }
             }
             if (count > 1)
             {
                   return true;
             }
       }
       return false;
    }

    /**
```

```java
     * Helper function for bulgarianSolitaire
     * Checks if a given element is in an ArrayList
     * @param value the value that we want to check is in the ArrayList
     * @param values the ArrayList we want the value to be in
     * @return whether or not value is in values
     */
    public static boolean existsIn(ArrayList<Integer> values, int value)
    {
       for (int i = 0; i < values.size(); i++)
       {
             if (value == values.get(i))
                   return true;
       }
       return false;
    }

    /**
     * Helper function for bulgarianSolitaire
     * Removes all instances of 0 from an ArrayList of integers.
     * @param values an ArrayList of integers
     * @return the given ArrayList with only nonzero elements.
     */
    public static ArrayList<Integer> removeZeroes(ArrayList<Integer>
values)
    {
       for (int i = values.size() - 1; i >= 0; i--)
       {
             if (values.get(i) == 0)
             {
                   values.remove(i);
             }
       }
       return values;
    }

    /**
     * Helper function for bulgarianSolitaire
     * Prints all the values in an ArrayList, separated by a space.
     * For example, "[4, 5, 3, 13, 4]" would be printed as "4 5 3 13 4"
     */
    public static void fancyPrint(ArrayList<Integer> values)
    {
       for (int value: values)
```

```
      {
            System.out.print(value);
            System.out.print(" ");
      }
      System.out.println();
   }

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

      Random rand = new Random();
      int piles = rand.nextInt(numCards - 1) + 1; // # of piles.
Initialize to random # from 1 to 5.
      ArrayList<Integer> cards = new ArrayList<Integer>();
      numCards -= 1;
      for (int i = 0; i < piles; i++)
      {
            if (i == piles - 1)
            {
                  cards.add(numCards + 1);
            }
            else
            {
```

```
                  if (numCards == 0)
                  {
                        cards.add(0);
                  }
                  else
                  {
                        int card = rand.nextInt(numCards) + 1;
                        cards.add(card);
                        numCards -= card;
                  }

            }
      }
      cards = removeZeroes(cards);
      System.out.print("BULGARIAN SOLITAIRE SIMULATION\n\nInitial
configuration: ");
      fancyPrint(cards);
      int counter = 0;
      boolean satisfied = false;
      while (!satisfied)
      {
            counter++;
            for (int j = 0; j < cards.size(); j++)
            {
                  cards.set(j, cards.get(j) - 1);
            }
            cards.add(cards.size());
            cards = removeZeroes(cards);
            System.out.print("[" + counter + "] Current configuration: ");

            fancyPrint(cards);

            if ((!hasRepeats(cards)) && (cards.size() == ideal.size()))
            {
                  int successes = 0;
                  for (int ca: cards)
                  {
                        if (existsIn(ideal, ca))
                        {
                              successes++;
                        }
                  }
                  if (successes == ideal.size())
```

```
                    {
                        satisfied = true;
                        System.out.println("Done!");
                    }
                }
            }
        }
}
```

[SEE NEXT PAGE FOR OUTPUT]

Output (note: this is one possible output. As the decks are initialized randomly, the output could vary. However, the end goal is the same.):

```
BULGARIAN SOLITAIRE SIMULATION

Initial configuration: 30 10 2 2 1
[1] Current configuration: 29 9 1 1 5
[2] Current configuration: 28 8 4 5
[3] Current configuration: 27 7 3 4 4
[4] Current configuration: 26 6 2 3 3 5
[5] Current configuration: 25 5 1 2 2 4 6
[6] Current configuration: 24 4 1 1 3 5 7
[7] Current configuration: 23 3 2 4 6 7
[8] Current configuration: 22 2 1 3 5 6 6
[9] Current configuration: 21 1 2 4 5 5 7
[10] Current configuration: 20 1 3 4 4 6 7
[11] Current configuration: 19 2 3 3 5 6 7
[12] Current configuration: 18 1 2 2 4 5 6 7
[13] Current configuration: 17 1 1 3 4 5 6 8
[14] Current configuration: 16 2 3 4 5 7 8
[15] Current configuration: 15 1 2 3 4 6 7 7
[16] Current configuration: 14 1 2 3 5 6 6 8
[17] Current configuration: 13 1 2 4 5 5 7 8
[18] Current configuration: 12 1 3 4 4 6 7 8
[19] Current configuration: 11 2 3 3 5 6 7 8
[20] Current configuration: 10 1 2 2 4 5 6 7 8
[21] Current configuration: 9 1 1 3 4 5 6 7 9
[22] Current configuration: 8 2 3 4 5 6 8 9
[23] Current configuration: 7 1 2 3 4 5 7 8 8
[24] Current configuration: 6 1 2 3 4 6 7 7 9
[25] Current configuration: 5 1 2 3 5 6 6 8 9
[26] Current configuration: 4 1 2 4 5 5 7 8 9
[27] Current configuration: 3 1 3 4 4 6 7 8 9
[28] Current configuration: 2 2 3 3 5 6 7 8 9
[29] Current configuration: 1 1 2 2 4 5 6 7 8 9
[30] Current configuration: 1 1 3 4 5 6 7 8 10
[31] Current configuration: 2 3 4 5 6 7 9 9
[32] Current configuration: 1 2 3 4 5 6 8 8 8
[33] Current configuration: 1 2 3 4 5 7 7 7 9
[34] Current configuration: 1 2 3 4 6 6 6 8 9
[35] Current configuration: 1 2 3 5 5 5 7 8 9
[36] Current configuration: 1 2 4 4 4 6 7 8 9
[37] Current configuration: 1 3 3 3 5 6 7 8 9
[38] Current configuration: 2 2 2 4 5 6 7 8 9
[39] Current configuration: 1 1 1 3 4 5 6 7 8 9
[40] Current configuration: 2 3 4 5 6 7 8 10
[41] Current configuration: 1 2 3 4 5 6 7 9 8
Done!
```