```java
import java.util.LinkedList;
import java.util.List;

public class DNFSolver {

	private static final String v = "v";
	private static final String not = "¬";


	public DNFSolver() {
	}

	/**
	 * Evaluates a given DNF-query in the following form:
	 *		"A¬BCvBC¬BvA¬AB¬C" (satisfiable)
	 *
	 * Constraints for input:
	 *		- OR: 'v'
	 *		- AND: (implicit; e.g. A AND B = AB)
	 *		- NOT: '¬'
	 *		- all variables are single chars (upper case)
	 *		- no spaces
	 *
	 * @param str A logic formula in DNF (disjunktive normal form).
	 * @return true, if satisfiable, false, if not.
	 */
	public boolean solve(String str) {
		List<String> list = new LinkedList<String>();

		// split up input string
		String temp = "";
		String token;
		for (int i = 0; i < str.length(); i++) {
			token = str.substring(i,i+1);
			if (token.equals(v)) {
				list.add(temp);
				temp = "";
			}
			else temp += token;
		}
		list.add(temp);

		System.out.println(list.size()+" disjunct elements found.");

		// print disjunct members
		/*
		for (int i = 0; i < list.size(); i++)
			System.out.println(i+": "+list.get(i));
		*/

		// create boolean lists
		List<String> variables = new LinkedList<String>();
		List<Boolean> vars = new LinkedList<Boolean>();
		List<Boolean> varsNegated = new LinkedList<Boolean>();
		boolean isNegated = false;

		// loop through list elements
		for (int c = 0; c < list.size(); c++) {
			String s = list.get(c);
			System.out.println(" - checking on '"+s+"'");

			// clear lists
			variables.clear();
			vars.clear();
			varsNegated.clear();

			// loop through variables
			for (int j = 0; j < s.length(); j++) {
				String var = s.substring(j,j+1);

				if (var.equals(not)) {
```

```
                                        isNegated = true;
                                        continue;
                                }

                                // new variable
                                if (!variables.contains(var)) {
                                        variables.add(var);                     // add variable to list
                                        vars.add(!isNegated);                   // if negated, add
false here
                                        varsNegated.add(isNegated);             // if negated, add

true here
                                }
                                else {
                                        int pos = variables.indexOf(var);
                                        if (isNegated) varsNegated.set(pos, true);
                                        else vars.set(pos,true);
                                }

                                isNegated = false;
                        }

                        // check for valid occupancy
                        boolean validSolution = true;

                        for (int j = 0; j < variables.size(); j++) {
                                if (vars.get(j) && varsNegated.get(j)) {
                                        validSolution = false;
                                        break;
                                }
                        }
                        if (validSolution) {
                                System.out.println("First valid solution found:");
                                for (int j = 0; j < variables.size(); j++) {
                                        System.out.println(variables.get(j)+" = "+(vars.get(j)));
                                }
                                return true;
                        }
                }
                return false;
        }
}
```