Appendix A: Java implementation of List Comprehensions

```
import org.apache.commons.lang3.tuple.Pair;

import java.util.*;
import java.util.function.*;
import java.util.stream.Collectors;

@SuppressWarnings("unchecked")
public class ListComprehension<T> {

    private Function<T, T> outputExpression = x -> x;
    private BiFunction<T, T,?> pairOutputExpression = (x,y) -> Pair.of(x,y);

    ListComprehension() {
    }

    public void setOutputExpression(Function<T, T> outputExpression) {
        this.outputExpression = outputExpression;
    }

    public void setPairOutputExpression(BiFunction<T, T,?> pairOutputExpression) {
        this.pairOutputExpression = pairOutputExpression;
    }

    public List<T> suchThat(Consumer<Var> predicates){
        Var x = new Var<T>();
        predicates.accept(x);
        return (List<T>) x.value().stream().map(outputExpression).collect(
                Collectors.toList());
    }

    public List<Pair<T, T>> suchThat(BiConsumer<Var, Var> predicates){
        Var x = new Var<T>();
        Var y = new Var<T>();
        predicates.accept(x,y);
        return (List<Pair<T, T>>) x.value(y).stream()
                .map(pair -> pairOutputExpression.apply((T) ((Pair) pair)
                .getLeft(), (T) ((Pair) pair).getRight()))
                .collect(Collectors.toList());
    }

    public ListComprehension<T> giveMeAll(Function<T, T> resultTransformer) {
        this.setOutputExpression(resultTransformer);
        return this;
    }

    public ListComprehension<T> giveMeAll(BiFunction<T, T, ?> resultTransformer) {
        this.setPairOutputExpression(resultTransformer);
        return this;
    }

    public class Var<T> {
        private Set<T> belongsTo = new HashSet<>();
        private Set<Predicate<T>> predicates = new HashSet<>();
        private Set<BiPredicate<T, T>> biPredicates = new HashSet<>();

        Var() {
            this.predicates.add(x -> true);
        this.biPredicates.add((x, y) -> true);
    }

    public Var belongsTo(List<T> c) {
        this.belongsTo.addAll(c);
        return this;
    }

    public Var is(Predicate<T> p) {
        this.predicates.add(p);
        return this;
    }

    public Var holds(Predicate<T> p) {
        return is(p);
    }

    public Var is(BiPredicate<T, T> p) {
        this.biPredicates.add(p);
        return this;
    }

    public Var holds(BiPredicate<T, T> p) {
        return is(p);
    }

    public List<T> value() {
        return intersect(predicates.stream()
                .map(condition -> belongsTo.stream()
                        .filter(condition)
                        .collect(Collectors.toList()))
                .collect(Collectors.toList()));
    }

    private List<T> intersect(List<List<T>> lists) {
        return belongsTo.stream()
                .filter(x -> lists.stream()
                        .filter(list -> list.contains(x)).count() == lists.size())
                .collect(Collectors.toList());
    }

    public List<Pair<T, T>> value(Var yVar) {
        List<BiPredicate<T, T>> allBiPredicates = new LinkedList<>();
        allBiPredicates.addAll(this.biPredicates);
        allBiPredicates.addAll((Collection<? extends BiPredicate<T, T>>)
                                yVar.biPredicates.stream()
                .map(new Function<BiPredicate<T, T>,BiPredicate<T, T>>() {
                    @Override
                    public BiPredicate apply(BiPredicate p) {
                        return new BiPredicate<T, T>() {
                            @Override
                            public boolean test(T x, T y) {
                                return p.test(y,x);
                            }
                        };
                    }
                }).collect(Collectors.toList()));
```

```
            List<Pair<T, T>> pairs = new LinkedList<>();

            this.value().stream().map(new Function<T, Boolean>() {
                @Override
                public Boolean apply(T x) {
                    yVar.value().stream().map(new Function<T, Boolean>() {
                        @Override
                        public Boolean apply(T y) {
                            return pairs.add(Pair.of(x, y));
                        }
                    }).collect(Collectors.toList());
                    return null;
                }
            }).collect(Collectors.toList());

            return pairs.stream().filter(pair -> holdsAll(allBiPredicates, pair))
                   .collect(Collectors.toList());
        }

        public boolean holdsAll(List<BiPredicate<T,T>> predicates, Pair<T,T> pair) {
            return predicates.stream().filter(p -> p.test(pair
                          .getLeft(),pair.getRight())).count() == predicates.size();
        }

        public <G> List<G> concat(List<List<G>> lists) {
            List<G> list = new LinkedList<>();
            lists.stream().map(l -> {
                list.addAll(l);
                return l;
            }).collect(Collectors.toList());
            return list;
        }
    }
}
```