1 Notes

The NetBeans project folder is ‘LogicErrorTool’, in the same directory as this document.

1.1 ‘.java’ files

The code colour key is:

Colour **Source**

- Original
- NetBeans template (Table 1.1)
- Oracle (n. d.)

Table 1.1: NetBeans templates used for different ‘.java’ file types.

Type	**Template**
‘package-info’	Java Package Info
Subclass of ‘JFrame’	JFrame Form
Subclass of ‘JDialog’	JDialog Form
Main class that is subclass of ‘Object’	Java Main Class
Subclass of ‘JPanel’	JPanel Form
Test suite	Test Suite – Junit 4.x
Unit test	Test Class – Junit 4.x
Non-main class that is subclass of ‘Object’, or subclass of class not part of Java’s API	Java Class
Enum	Java Enum
Interface	Java Interface

Non-template code in ‘main()’ (for subclasses of ‘JFrame’ and ‘JDialog’), and in ‘initComponents()’ and variable declarations (for both classes and ‘JPanel’) was generated by NetBeans, from a model specified via the Design View.

1.2 Other files

‘.class’ files were generated by NetBeans when the project was built. ‘built-jar.properties’, ‘LogicErrorTool.jar’, and ‘README.txt’ were generated by NetBeans when the ‘LogicErrorTool’ class was compiled. Other NetBeans project files were generated by NetBeans when the project was first created.

2 Code listings

2.1 Source packages

2.1.1 Default package

2.1.1.1 File: Experiment.java

```
import cs.Guard;
import cs.loop.ArithmeticLoop;
import cs.loop.Loop;
import cs.loop.cg.ArithmeticCorrectionGenerator;
import cs.loop.ic.ArithmeticIterationCounter;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import maths.Relation;
import maths.is.ArithmeticInfiniteSequence;

/**
 * The experiment, as part of the experimental research method
used.
 */
class Experiment {

    /**
     * A main method.
     *
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Random random = new Random();
        List<Loop> infiniteLoops = new ArrayList<>();
        int numInfiniteLoopSpawns = 100;

        while (infiniteLoops.size() < numInfiniteLoopSpawns) {
            Relation[] relations = Relation.values();
            Loop loop = new ArithmeticLoop(new
ArithmeticInfiniteSequence(random.nextInt(),
random.nextInt()), new
Guard(relations[random.nextInt(relations.length)],
random.nextInt()));

            if (!loop.terminates()) {
                infiniteLoops.add(loop);
            }
        }

        int numOccurrencesOfA = 0;
```

```
        for (Loop infiniteLoop : infiniteLoops) {
            List<Loop> corrections = new
ArithmeticCorrectionGenerator((ArithmeticLoop)
infiniteLoop).getCorrections();
            for (Loop correction : corrections) {
                if ((new
ArithmeticIterationCounter((ArithmeticLoop)
correction)).getCount() > 1) {
                    numOccurrencesOfA++;
                    break;
                }
            }
        }

        System.out.println(numOccurrencesOfA + " occurrences
of A, out of " + numInfiniteLoopSpawns + " infinite loop
spawns");
    }
}
```

2.1.2 Package: cs

2.1.2.1 File: ArrayPositionRange.java

```
package cs;

import cs.loop.ArithmeticLoop;
import cs.loop.Loop;
import java.util.ArrayList;
import java.util.List;
import maths.Relation;
import maths.is.ArithmeticInfiniteSequence;

/**
 * A contiguous range of 1-based positions in an array.
 */
public class ArrayPositionRange {

    private final int start, end;

    /**
     * The sole constructor.
     *
     * @param start the start position in the array
     * @param end the end position in the array
     */
    public ArrayPositionRange(int start, int end) {
        this.start = start;
        this.end = end;
    }
```

```
    /**
     * A method to generate loops to traverse the 0-based
array index range
     * corresponding to the array position range.
     *
     * @return loops whose control variable assumes values
from the index range
     */
    public List<Loop> getTraversingLoops() {
        int a = start - 1, b = end - 1, d;
        List<Loop> loops = new ArrayList<>();
        List<Guard> guards = new ArrayList<>();

        if (a > b) {
            guards.add(new Guard(Relation.GT, b - 1));
            guards.add(new Guard(Relation.GTOET, b));
            d = -1;
        } else {
            guards.add(new Guard(Relation.LT, end));
            guards.add(new Guard(Relation.LTOET, b));
            d = 1;
        }

        guards.stream().forEach((guard) -> {
            loops.add(new ArithmeticLoop(new
ArithmeticInfiniteSequence(a, d), guard));
        });

        return loops;
    }
}
```

2.1.2.2 File: Guard.java

```
package cs;

import maths.Relation;
import java.util.Objects;

/**
 * A guard that compares a control variable's value with some
other value.
 */
public class Guard implements Cloneable {

    private Relation R;
    private int b;

    /**
     * The sole constructor.
     *
```

```
     * @param R a relation defined on a control variable i and
b such that i R b
     * @param b a value with which the value of i is compared
     */
    public Guard(Relation R, int b) {
        setR(R);
        setB(b);
    }

    /**
     * A getter.
     *
     * @return a relation defined on a control variable i and
b such that i R b
     */
    public Relation getR() {
        return R;
    }

    /**
     * A setter.
     *
     * @param R a relation defined on a control variable i and
b such that i R b
     */
    public final void setR(Relation R) {
        this.R = R;
    }

    /**
     * A getter.
     *
     * @return a value with which a control variable's value
is compared
     */
    public int getB() {
        return b;
    }

    /**
     * A setter.
     *
     * @param b a value with which a control variable's value
is compared
     */
    public final void setB(int b) {
        this.b = b;
    }

    @Override
    public boolean equals(Object o) {
```

```
        if (o instanceof Guard) {
            Guard guard = (Guard) o;
            return (guard.R).equals(R) && guard.b == b;
        } else {
            return false;
        }
    }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 59 * hash + Objects.hashCode(this.R);
        hash = 59 * hash + Objects.hashCode(this.b);
        return hash;
    }

    @Override
    public Object clone() throws CloneNotSupportedException {
        Guard guard = (Guard) super.clone();
        guard.R = R;
        guard.b = b;
        return guard;
    }
}
```

2.1.2.3 File: Metaobject.java

```
package cs;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

/**
 * A model of a base object that declares at least 1 string
instance variable
 * and can be in 1 of multiple initial states. Each initial
state describes a
 * set of initialisations performed by a base object
constructor.
 */
public class Metaobject {

    private final List<String> vars;
    private List<Map<String, String>> initialStates;

    /**
     * The sole metaobject constructor.
     *
     * @param vars the base object's variables
     * @param initialStates mappings from names to values for
the variables
```

```
     */
    public Metaobject(List<String> vars, List<Map<String,
String>> initialStates) {
        this.vars = vars;
        this.initialStates = initialStates;
    }

    /**
     * A getter.
     *
     * @return mappings from names to values for the base
object's variables
     */
    public List<Map<String, String>> getInitialStates() {
        return initialStates;
    }

    /**
     * A method to find variables of the base object
implicitly initialised to
     * null.
     *
     * @return variables of the base object that are not
initialised in all of
     * the object's constructors
     */
    public List<String> getUninitialisedVars() {
        List<String> uninitialisedVars = new ArrayList<>();

        for (String var : vars) {
            for (Map<String, String> initialState :
initialStates) {
                if (!initialState.containsKey(var)) {
                    uninitialisedVars.add(var);
                    break;
                }
            }
        }

        return uninitialisedVars;
    }

    /**
     * A method to initialise the base object's variables to
an empty string
     * wherever they are not initialised in a constructor of
the object.
     */
    public void addMissingInitialisations() {
```

```
getUninitialisedVars().stream().forEach((uninitialisedVar) ->
{
            initialStates.stream().filter((initialState) ->
(!initialState.containsKey(uninitialisedVar))).forEach((initia
lState) -> {
                initialState.put(uninitialisedVar, "\"\"");
            });
        });

        List<Map<String, String>> distinctInitialStates = new
ArrayList<>();

        //remove duplicate constructors
        initialStates.stream().filter((initialState) ->
(!distinctInitialStates.contains(initialState))).forEach((init
ialState) -> {
            distinctInitialStates.add(initialState);
        });

        initialStates = distinctInitialStates;
    }
}
```

2.1.2.4 File: package-info.java

```
/**
* A package for classes related to Computer Science.
 */
package cs;
```

2.1.3 Package: cs.loop

2.1.3.1 File: ArithmeticLoop.java

```
package cs.loop;

import cs.Guard;
import maths.is.ArithmeticInfiniteSequence;

/**
 * A loop for which the values the control variable assumes
form an arithmetic
 * sequence.
 */
public class ArithmeticLoop extends Loop {

    /**
     * The sole constructor.
     *
     * @param ais: an arithmetic infinite sequence that is a
superset of the
```

```
     * values the loop's control variable assumes
     * @param guard: a guard such that for as long as it
evaluates to TRUE, the
     * loop runs
     */
    public ArithmeticLoop(ArithmeticInfiniteSequence ais,
Guard guard) {
        super(ais, guard);
    }

    @Override
    boolean terminatesEventually_LTorLTOET() {
        return ((ArithmeticInfiniteSequence) is).getD() > 0;
    }

    @Override
    boolean terminatesEventually_GTorGTOET() {
        return ((ArithmeticInfiniteSequence) is).getD() < 0;
    }
}
```

2.1.3.2 File: GeometricLoop.java

```
package cs.loop;

import cs.Guard;
import maths.Relation;
import maths.is.GeometricInfiniteSequence;

/**
 * A loop for which the values the control variable assumes
form a geometric
 * sequence.
 */
public class GeometricLoop extends Loop {

    /**
     * The sole constructor.
     *
     * @param gis a geometric infinite sequence that is a
superset of the values
     * the loop's control variable assumes
     * @param guard a guard such that for as long as it
evaluates to TRUE, the
     * loop runs
     */
    public GeometricLoop(GeometricInfiniteSequence gis, Guard
guard) {
        super(gis, guard);
    }

    @Override
```

```
    boolean terminatesEventually_neitherNETnorET() {
        return ((GeometricInfiniteSequence) is).getR() < -1 ?
is.getA() != 0 : super.terminatesEventually_neitherNETnorET();
    }

    @Override
    boolean terminatesEventually_LTorLTOET() {
        int r = ((GeometricInfiniteSequence) is).getR();
        int a = is.getA();
        int b = guard.getB();
        return r > 1 ? a > 0 :
((guard.getR()).equals(Relation.LT) ? (r == 0 ? b <= 0 : -a >=
b) : (r == 0 ? b < 0 : -a > b));
    }

    @Override
    boolean terminatesEventually_GTorGTOET() {
        int r = ((GeometricInfiniteSequence) is).getR();
        int a = is.getA();
        int b = guard.getB();
        return r > 1 ? a < 0 :
((guard.getR()).equals(Relation.GT) ? (r == 0 ? b >= 0 : -a <=
b) : (r == 0 ? b > 0 : -a < b));
    }
}
```

2.1.3.3 File: Loop.java

```
package cs.loop;

import cs.Guard;
import maths.is.InfiniteSequence;
import java.util.Objects;

/**
 * A loop for which the values the control variable assumes
form a sequence.
 */
public abstract class Loop implements Cloneable {

    InfiniteSequence is;
    Guard guard;

    Loop(InfiniteSequence is, Guard guard) {
        this.is = is;
        this.guard = guard;
    }

    /**
     * A getter.
     *
```

```
     * @return an infinite sequence that is a superset of the
values that the
     * loop's control variable assumes
     */
    public InfiniteSequence getIS() {
        return is;
    }

    /**
     * A getter.
     *
     * @return a guard such that for as long as it evaluates
to TRUE, the loop
     * runs
     */
    public Guard getGuard() {
        return guard;
    }

    /**
     * A method to analyse loop termination.
     *
     * @return a truth value for whether the loop terminates
     */
    public boolean terminates() {
        return terminatesImmediately() ||
terminatesEventually();
    }

    private boolean terminatesImmediately() {
        int a = is.getA(), b = guard.getB();

        switch (guard.getR()) {
            case LT:
                return a >= b;
            case LTOET:
                return a > b;
            case GT:
                return a <= b;
            case GTOET:
                return a < b;
            case NET:
                return is.startsWith(b);
            default: //ET
                return !is.startsWith(b);
        }
    }

    private boolean terminatesEventually() {
        if (is.hasSingleDistinctTerm()) {
            return false;
```

```
        } else {
            switch (guard.getR()) {
                case NET:
                    return
is.containsNotStartsWith(guard.getB());
                case ET:
                    return true;
                default: //LT, LTOET, GT, GTOET
                    return
terminatesEventually_neitherNETnorET();
            }
        }
    }

    //assumptions: IS has multiple distinct terms and R ∉
{'≠', '='}
    boolean terminatesEventually_neitherNETnorET() {
        switch (guard.getR()) {
            case LT:
            case LTOET:
                return terminatesEventually_LTorLTOET();
            default: //GT, GTOET
                return terminatesEventually_GTorGTOET();
        }
    }

    //assumptions: IS has multiple distinct terms and R ∈
{'<', '≤'}
    abstract boolean terminatesEventually_LTorLTOET();

    //assumptions: IS has multiple distinct terms and R ∈
{'>', '≥'}
    abstract boolean terminatesEventually_GTorGTOET();

    @Override
    public boolean equals(Object o) {
        if (o instanceof Loop) {
            Loop loop = (Loop) o;
            return (loop.is).equals(is)
                    && (loop.guard).equals(guard);
        } else {
            return false;
        }
    }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 37 * hash + Objects.hashCode(this.guard);
        return hash;
    }
```

```
    @Override
    public Object clone() throws CloneNotSupportedException {
        Loop loop = (Loop) super.clone();
        loop.is = (InfiniteSequence) is.clone();
        loop.guard = (Guard) guard.clone();
        return loop;
    }
}
```

2.1.3.4 File: package-info.java

```
/**
 * A package for classes related to loops.
 */
package cs.loop;
```

2.1.4 Package: cs.loop.cg

2.1.4.1 File: ArithmeticCorrectionGenerator.java

```
package cs.loop.cg;

import cs.loop.ArithmeticLoop;
import cs.loop.Loop;
import maths.is.ArithmeticInfiniteSequence;
import java.util.List;

/**
 * A utility to generate corrections to an erroneously
infinite arithmetic loop.
 */
public class ArithmeticCorrectionGenerator extends
CorrectionGenerator {

    /**
     * The sole constructor.
     *
     * @param al an infinite arithmetic loop
     */
    public ArithmeticCorrectionGenerator(ArithmeticLoop al) {
        super(al);
    }

    @Override
    public List<Loop> getCorrections() {
        try {
            List<Loop> corrections = super.getCorrections();
            Loop clone;
```

```
            for (int dPrime :
getChangesToIntInput(((ArithmeticInfiniteSequence)
loop.getIS()).getD())) {
                clone = (Loop) loop.clone();
                ((ArithmeticInfiniteSequence)
(clone.getIS())).setD(dPrime);

                if (clone.terminates()) {
                    corrections.add((Loop) clone.clone());
                }
            }

            return corrections;
        } catch (CloneNotSupportedException e) {
            throw new RuntimeException(e.getMessage());
//shouldn't reach here
        }
    }
}
```

2.1.4.2 File: CorrectionGenerator.java

```
package cs.loop.cg;

import cs.loop.Loop;
import maths.Relation;
import java.util.ArrayList;
import java.util.List;

/**
 * A utility to generate corrections to an erroneously
infinite loop.
 */
public abstract class CorrectionGenerator {

    Loop loop;

    CorrectionGenerator(Loop loop) {
        this.loop = loop;
    }

    Loop getLoop() {
        return loop;
    }

    /**
     * A method to generate corrections to an erroneously
infinite loop by
     * changing 1 input at a time.
     *
     * @return finite loops with similar inputs to an infinite
loop
```

```
     */
    public List<Loop> getCorrections() {
        try {
            List<Loop> corrections = new ArrayList<>();
            Loop clone;

            for (int aPrime :
getChangesToIntInput((loop.getIS()).getA())) {
                clone = (Loop) loop.clone();
                (clone.getIS()).setA(aPrime);

                if (clone.terminates()) {
                    corrections.add((Loop) clone.clone());
                }
            }

            for (int bPrime :
getChangesToIntInput((loop.getGuard()).getB())) {
                clone = (Loop) loop.clone();
                (clone.getGuard()).setB(bPrime);

                if (clone.terminates()) {
                    corrections.add((Loop) clone.clone());
                }
            }

            for (Relation RPrime : getChangesToR()) {
                clone = (Loop) loop.clone();
                (clone.getGuard()).setR(RPrime);

                if (clone.terminates()) {
                    corrections.add((Loop) clone.clone());
                }
            }

            return corrections;
        } catch (CloneNotSupportedException e) {
            throw new RuntimeException(e); //shouldn't reach
here
        }
    }

    static int[] getChangesToIntInput(int input) {
        return new int[]{input + 1, input - 1, -input};
    }

    private Relation[] getChangesToR() {
        Relation R = (loop.getGuard()).getR();

        switch (R) {
            case LT:
```

```
                return new Relation[]{Relation.GT,
Relation.LTOET};
            case LTOET:
                return new Relation[]{Relation.GTOET,
Relation.LT};
            case GT:
                return new Relation[]{Relation.LT,
Relation.GTOET};
            case GTOET:
                return new Relation[]{Relation.LTOET,
Relation.GT};
            default: //NET or ET
                return new Relation[]{R.getComplement()};
        }
    }
}
```

2.1.4.3 File: GeometricCorrectionGenerator.java

```
package cs.loop.cg;

import cs.loop.GeometricLoop;
import cs.loop.Loop;
import maths.is.GeometricInfiniteSequence;
import java.util.List;

/**
 * A utility to generate corrections to an erroneously
infinite geometric loop.
 */
public class GeometricCorrectionGenerator extends
CorrectionGenerator {

    /**
     * The sole constructor.
     *
     * @param gl an infinite geometric loop
     */
    public GeometricCorrectionGenerator(GeometricLoop gl) {
        super(gl);
    }

    @Override
    public List<Loop> getCorrections() {
        try {
            List<Loop> corrections = super.getCorrections();
            Loop clone;

            for (int rPrime :
getChangesToIntInput(((GeometricInfiniteSequence)
loop.getIS()).getR())) {
                clone = (Loop) loop.clone();
```

```
                ((GeometricInfiniteSequence)
(clone.getIS())).setR(rPrime);

                if (clone.terminates()) {
                    corrections.add((Loop) clone.clone());
                }
            }

            return corrections;
        } catch (CloneNotSupportedException e) {
            throw new RuntimeException(e.getMessage());
//shouldn't reach here
        }
    }
}
```

2.1.4.4 File: package-info.java

```
/**
 * A package for classes related to infinite loop error
correction generation.
 */
package cs.loop.cg;
```

2.1.5 Package: cs.loop.ic

2.1.5.1 File: ArithmeticIterationCounter.java

```
package cs.loop.ic;

import cs.loop.ArithmeticLoop;
import maths.is.ArithmeticInfiniteSequence;

/**
 * A utility to count iterations in a finite arithmetic loop.
 */
public class ArithmeticIterationCounter extends
IterationCounter {

    //store d as constant to make f function (called in loop)
more efficient
    private final int d;

    /**
     * The sole constructor.
     *
     * @param al a finite arithmetic loop
     */
    public ArithmeticIterationCounter(ArithmeticLoop al) {
        super(al);
        d = ((ArithmeticInfiniteSequence) al.getIS()).getD();
    }
```

```
    @Override
    int f(int i) {
        return i + d;
    }
}
```

2.1.5.2 File: GeometricIterationCounter.java

```
package cs.loop.ic;

import cs.loop.GeometricLoop;
import maths.is.GeometricInfiniteSequence;

/**
 * A utility to count iterations in a finite geometric loop.
 */
public class GeometricIterationCounter extends
IterationCounter {

    //store r as constant to make f function (called in loop)
more efficient
    private final int r;

    /**
     * The sole constructor.
     *
     * @param gl a finite geometric loop
     */
    public GeometricIterationCounter(GeometricLoop gl) {
        super(gl);
        r = ((GeometricInfiniteSequence) gl.getIS()).getR();
    }

    @Override
    int f(int i) {
        return i * r;
    }
}
```

2.1.5.3 File: IterationCounter.java

```
package cs.loop.ic;

import cs.Guard;
import cs.loop.Loop;
import maths.Relation;

/**
 * A utility to count iterations in a finite loop.
 */
public abstract class IterationCounter {
```

```
    private final Loop loop;
    /*store R and b as constants to make iRB function (called
in loop) more
    efficient*/
    private final Relation R;
    private final int b;

    IterationCounter(Loop loop) {
        this.loop = loop;
        Guard guard = loop.getGuard();
        R = guard.getR();
        b = guard.getB();
    }

    /**
     * A method to count iterations in a finite loop. Note
that this method will
     * actually run the loop, with only the final expression
in the loop's body.
     *
     * @return the number of iterations in a finite loop
     */
    public int getCount() {
        int count = 0;
        int i = (loop.getIS()).getA();

        while (iRB(i)) {
            count++;
            i = f(i);
        }

        return count;
    }

    private boolean iRB(int i) {
        switch (R) {
            case LT:
                return i < b;
            case LTOET:
                return i <= b;
            case GT:
                return i > b;
            case GTOET:
                return i >= b;
            case NET:
                return i != b;
            case ET:
                return i == b;
            default:
                throw new NullPointerException("R is null");
```

```
        }
    }

    abstract int f(int i);
}
```

2.1.5.4 File: package-info.java

```
/**
 * A package for classes related to finite loop iteration
counting.
 */
package cs.loop.ic;
```

2.1.6 Package: cs.rmc

2.1.6.1 File: ArithmeticRecursiveMethodCall.java

```
package cs.rmc;

import cs.Guard;
import cs.loop.ArithmeticLoop;
import cs.loop.Loop;
import maths.is.ArithmeticInfiniteSequence;

/**
 * A call to a recursive method, for which the values of the
actual parameter
 * form an arithmetic sequence.
 */
public class ArithmeticRecursiveMethodCall extends
RecursiveMethodCall {

    /**
     * The sole constructor.
     *
     * @param ais an arithmetic infinite sequence that is a
superset of the
     * values the recursive method's actual parameter assumes
     * @param baseCase the recursive method's base case
     */
    public
ArithmeticRecursiveMethodCall(ArithmeticInfiniteSequence ais,
Guard baseCase) {
        super(ais, baseCase);
    }

    @Override
    public Loop toLoop() {
        return new ArithmeticLoop((ArithmeticInfiniteSequence)
is, getLoopGuard());
    }
```

```
}
```

2.1.6.2 File: GeometricRecursiveMethodCall.java

```
package cs.rmc;

import cs.Guard;
import cs.loop.GeometricLoop;
import cs.loop.Loop;
import maths.is.GeometricInfiniteSequence;

/**
 * A call to a recursive method, for which the values of the
actual parameter
 * form a geometric sequence.
 */
public class GeometricRecursiveMethodCall extends
RecursiveMethodCall {

    /**
     * The sole constructor.
     *
     * @param gis a geometric infinite sequence that is a
superset of the values
     * the recursive method's actual parameter assumes
     * @param baseCase the recursive method's base case
     */
    public
GeometricRecursiveMethodCall(GeometricInfiniteSequence gis,
Guard baseCase) {
        super(gis, baseCase);
    }

    @Override
    public Loop toLoop() {
        return new GeometricLoop((GeometricInfiniteSequence)
is, getLoopGuard());
    }
}
```

2.1.6.3 File: RecursiveMethodCall.java

```
package cs.rmc;

import cs.Guard;
import cs.loop.Loop;
import maths.is.InfiniteSequence;

/**
 * A call to a recursive method, for which the values of the
actual parameter
 * form a sequence.
```

```
 */
public abstract class RecursiveMethodCall {

    InfiniteSequence is;
    private final Guard baseCase;

    RecursiveMethodCall(InfiniteSequence is, Guard baseCase) {
        this.is = is;
        this.baseCase = baseCase;
    }

    /**
     * A method to generate an iterative version of the
recursive method call.
     *
     * @return a loop equivalent to the recursive method call
     */
    public abstract Loop toLoop();

    Guard getLoopGuard() {
        return new Guard((baseCase.getR()).getComplement(),
baseCase.getB());
    }
}
```

2.1.6.4 File: package-info.java

```
/**
 * A package for classes related to recursive method calls.
 */
package cs.rmc;
```

2.1.7 Package: maths

2.1.7.1 File: Relation.java

```
package maths;

/**
 * A comparison-based relation between numbers.
 */
public enum Relation {

    /**
     * A less-than relation.
     */
    LT,
    /**
     * A less-than-or-equal-to relation.
     */
    LTOET,
    /**
```

```
     * A greater-than relation.
     */
    GT,
    /**
     * A greater-than-or-equal-to relation.
     */
    GTOET,
    /**
     * A not-equal-to relation.
     */
    NET,
    /**
     * An equal-to relation.
     */
    ET;

    /**
     * A method to compute the complement of this relation.
     *
     * @return a relation S for this relation R such that for
all numbers x and
     * y, if x S y then ¬(x R y)
     */
    public Relation getComplement() {
        switch (this) {
            case LT:
                return GTOET;
            case LTOET:
                return GT;
            case GT:
                return LTOET;
            case GTOET:
                return LT;
            case NET:
                return ET;
            default: //ET
                return NET;
        }
    }

    @Override
    public String toString() {
        switch (this) {
            case LT:
                return "<";
            case LTOET:
                return "≤";
            case GT:
                return ">";
            case GTOET:
                return "≥";
```

```
            case NET:
                return "≠";
            default: //ET
                return "=";
        }
    }
}
```

2.1.7.2 File: package-info.java

```
/**
 * A package for classes related to mathematics.
 */
package maths;
```

2.1.8 Package: maths.ae

2.1.8.1 File: Node.java

```
package maths.ae;

/**
 * A node in an arithmetic expression tree.
 */
public class Node implements Cloneable {

    private String label;
    private Node left, right;

    /**
     * The constructor for a term node.
     *
     * @param term the term with which to label the node
     */
    public Node(String term) {
        label = term;
    }

    /**
     * The constructor for an operation node.
     *
     * @param operation the operation with which to label the
node
     * @param operand1 the operand with which to label the
left sub-node
     * @param operand2 the operand with which to label the
right sub-node
     */
    public Node(Operation operation, Node operand1, Node
operand2) {
        this(operation.toString());
        left = operand1;
```

```
        right = operand2;
    }

    private boolean isLeaf() {
        return left == null; //full binary tree so if no left
subnode, then no right subnode either
    }

    @Override
    public String toString() {
        if (isLeaf()) {
            //term node
            return label;
        } else {
            //operation node
            Node[] subnodes = new Node[]{left, right};
            int numSubnodes = subnodes.length;
            String[] subnodeStrings = new String[numSubnodes];

            for (int i = 0; i < numSubnodes; i++) {
                Node subnode = subnodes[i];
                String subnodeString = subnode.toString();

                if (!subnode.isLeaf() &&
(Operation.toOperation(label.charAt(0))).precedes(Operation.to
Operation((subnode.label).charAt(0)))) {
                    subnodeString = "(" + subnodeString + ")";
//bracket lower-precedence operations
                }

                subnodeStrings[i] = subnodeString;
            }

            return subnodeStrings[0] + " " + label + " " +
subnodeStrings[1];
        }
    }

    @Override
    public Object clone() throws CloneNotSupportedException {
        Node node = (Node) super.clone();
        node.label = "" + label; //make copy of string

        if (!node.isLeaf()) {
            node.left = (Node) left.clone();
            node.right = (Node) right.clone();
        }

        return node;
    }
}
```

2.1.8.2 File: Operation.java

```
package maths.ae;

/**
 * A binary arithmetic operation.
 */
public enum Operation {

    /**
     * An exponentiation.
     */
    EXPONENTIATE,
    /**
     * A division.
     */
    DIVIDE,
    /**
     * A multiplication.
     */
    MULTIPLY,
    /**
     * An addition.
     */
    ADD,
    /**
     * A subtraction.
     */
    SUBTRACT;

    static Operation toOperation(char operator) {
        switch (operator) {
            case '^':
                return EXPONENTIATE;
            case '/':
                return DIVIDE;
            case '*':
                return MULTIPLY;
            case '+':
                return ADD;
            case '-':
                return SUBTRACT;
            default:
                throw new IllegalArgumentException("Invalid
operator");
        }
    }

    //has higher precedence than
    boolean precedes(Operation op) {
        return compareTo(op) < 0;
```

```
    }

    @Override
    public String toString() {
        switch (this) {
            case EXPONENTIATE:
                return "^";
            case DIVIDE:
                return "/";
            case MULTIPLY:
                return "*";
            case ADD:
                return "+";
            default: //SUBTRACT
                return "-";
        }
    }
}
```

2.1.8.3 File: package-info.java

```
/**
 * A package for classes related to arithmetic expressions.
 */
package maths.ae;
```

2.1.9 Package: maths.is

2.1.9.1 File: ArithmeticInfiniteSequence.java

```
package maths.is;

/**
 * An arithmetic infinite sequence of integers.
 */
public class ArithmeticInfiniteSequence extends
InfiniteSequence implements Cloneable {

    private int d;

    /**
     * The sole constructor.
     *
     * @param a the first term in the AIS
     * @param d the difference between consecutive terms in
the AIS
     */
    public ArithmeticInfiniteSequence(int a, int d) {
        super(a);
        setD(d);
    }
```

```
    /**
     * A getter.
     *
     * @return the difference between consecutive terms in the
AIS
     */
    public int getD() {
        return d;
    }

    /**
     * A setter.
     *
     * @param d the difference between consecutive terms in
the AIS
     */
    public final void setD(int d) {
        this.d = d;
    }

    @Override
    public boolean hasSingleDistinctTerm() {
        return d == 0;
    }

    @Override
    public boolean containsNotStartsWith(int b) {
        int delta = b - a;
        return delta % d == 0 && Math.signum(delta) ==
Math.signum(d);
    }

    @Override
    public boolean equals(Object o) {
        boolean equals = super.equals(o);

        if (o instanceof ArithmeticInfiniteSequence) {
            equals &= (d == ((ArithmeticInfiniteSequence)
o).d);
        }

        return equals;
    }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 11 * hash + this.d;
        return hash;
    }
```

```
    @Override
    public Object clone() throws CloneNotSupportedException {
        ArithmeticInfiniteSequence ais =
(ArithmeticInfiniteSequence) super.clone();
        ais.d = d;
        return ais;
    }
}
```

2.1.9.2 File: GeometricInfiniteSequence.java

```
package maths.is;

/**
 * A geometric infinite sequence of integers.
 */
public class GeometricInfiniteSequence extends
InfiniteSequence implements Cloneable {

    private int r;

    /**
     * The sole constructor.
     *
     * @param a the first term in the GIS
     * @param r the ratio between consecutive terms in the GIS
     */
    public GeometricInfiniteSequence(int a, int r) {
        super(a);
        setR(r);
    }

    /**
     * A getter
     *
     * @return the ratio between consecutive terms in the GIS
     */
    public int getR() {
        return r;
    }

    /**
     * A setter.
     *
     * @param r the ratio between consecutive terms in the GIS
     */
    public final void setR(int r) {
        this.r = r;
    }

    @Override
    public boolean hasSingleDistinctTerm() {
```

```
        return r == 1;
    }

    @Override
    public boolean containsNotStartsWith(int b) {
        switch (r) {
            case 0:
                return b == 0;
            case -1:
                return a == -b;
            default:
                return a == 0 || b == 0 ? false : (b % a == 0
&& b % r == 0 ? r > 1 || (Math.signum(a) == Math.signum(b)) ==
(Math.log(Math.abs(b / a)) / Math.log(Math.abs(r)) % 2 == 0) :
false);
        }
    }

    @Override
    public boolean equals(Object o) {
        boolean equals = super.equals(o);

        if (o instanceof GeometricInfiniteSequence) {
            equals &= (r == ((GeometricInfiniteSequence)
o).r);
        }

        return equals;
    }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 97 * hash + this.r;
        return hash;
    }

    @Override
    public Object clone() throws CloneNotSupportedException {
        GeometricInfiniteSequence gis
                = (GeometricInfiniteSequence) super.clone();
        gis.r = r;
        return gis;
    }
}
```

2.1.9.3 File: InfiniteSequence.java

```
package maths.is;

/**
 * An infinite sequence of integers.
```

```
 */
public abstract class InfiniteSequence implements Cloneable {

    int a;

    InfiniteSequence(int a) {
        setA(a);
    }

    /**
     * A getter.
     *
     * @return the first term in the IS
     */
    public int getA() {
        return a;
    }

    /**
     * A setter.
     *
     * @param a the first term in the IS
     */
    public final void setA(int a) {
        this.a = a;
    }

    /**
     * A method to determine whether the IS starts with some
integer.
     *
     * @param b the integer
     * @return a truth value for whether b is the first term
of the IS
     */
    public boolean startsWith(int b) {
        return a == b;
    }

    boolean contains(int b) {
        return startsWith(b) ? true : (hasSingleDistinctTerm()
? false : containsNotStartsWith(b));
    }

    /**
     * A method to determine whether the IS has exactly 1
distinct term.
     *
     * @return a truth value for whether the IS is equivalent
to a
     * singleton
```

```
     */
    public abstract boolean hasSingleDistinctTerm();

    /**
     * A method to determine whether the IS contains but does
not start
     * with some integer.
     *
     * @param b the integer
     * @return a truth value for whether b is a term of the IS
and not the
     * first
     */
    public abstract boolean containsNotStartsWith(int b);

    @Override
    public boolean equals(Object o) {
        return o instanceof InfiniteSequence ? (a ==
((InfiniteSequence) o).a) : false;
    }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 97 * hash + this.a;
        return hash;
    }

    @Override
    public Object clone() throws CloneNotSupportedException {
        InfiniteSequence is = (InfiniteSequence)
super.clone();
        is.a = a;
        return is;
    }
}
```

2.1.9.4 File: package-info.java

```
/**
 * A package for classes related to infinite sequences of
integers.
 */
package maths.is;
```

2.1.10 Package: ui

2.1.10.1 File: AbstractTableModelImpl.java

```
package ui;

import javax.swing.table.AbstractTableModel;
```

```
/**
 * A read-only implementation of AbstractTableModel.
 */
public class AbstractTableModelImpl extends AbstractTableModel
{

    private final Object[][] rowData;
    private final Object[] columnData;

    /**
     * The sole constructor.
     *
     * @param rowData the entries to display in the table
     * @param columnData the column headers to display in the
table
     */
    public AbstractTableModelImpl(Object[][] rowData, Object[]
columnData) {
        super();
        this.rowData = rowData;
        this.columnData = columnData;
    }

    @Override
    public int getColumnCount() {
        return columnData.length;
    }

    @Override
    public String getColumnName(int columnIndex) {
        return columnData[columnIndex].toString();
    }

    @Override
    public int getRowCount() {
        return rowData.length;
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        return rowData[rowIndex][columnIndex];
    }

    @Override
    public boolean isCellEditable(int rowIndex, int
columnIndex) {
        return false;
    }
}
```

2.1.10.2 File: LogicErrorTool.java

```
package ui;

class LogicErrorTool extends javax.swing.JFrame {

    /**
     * Creates new form LogicErrorTool
     */
    private LogicErrorTool() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to
initialize the form.
     * WARNING: Do NOT modify this code. The content of this
method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated
Code">
    private void initComponents() {

        jLabel1 = new javax.swing.JLabel();
        jTabbedPane1 = new javax.swing.JTabbedPane();
        infiniteLoopPanel1 = new ui.jp.InfiniteLoopPanel();
        infiniteRecursionPanel1 = new
ui.jp.InfiniteRecursionPanel();
        arithmeticExpressionPanel1 = new
ui.jp.ArithmeticExpressionPanel();
        offByOnePanel1 = new ui.jp.OffByOnePanel();
        nullPointerPanel1 = new ui.jp.NullPointerPanel();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_C
LOSE);
        setTitle("LogicErrorTool 2.0");

        jLabel1.setText("Error to check for");

        jTabbedPane1.addTab("Infinite loop",
infiniteLoopPanel1);
        jTabbedPane1.addTab("Infinite recursion",
infiniteRecursionPanel1);
        jTabbedPane1.addTab("Arithmetic expression",
arithmeticExpressionPanel1);
        jTabbedPane1.addTab("Off-by-one", offByOnePanel1);
        jTabbedPane1.addTab("Null pointer",
nullPointerPanel1);
```

```
        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                    .addComponent(jLabel1)
                    .addComponent(jTabbedPane1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel1)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                .addComponent(jTabbedPane1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );

        pack();
    }// </editor-fold>

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look
and feel setting code (optional) ">
```

```
        /* If Nimbus (introduced in Java SE 6) is not
available, stay with the default look and feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel
/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException |
InstantiationException | IllegalAccessException |
javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(LogicErrorTool.class.getNam
e()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>

        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(() -> {
            new LogicErrorTool().setVisible(true);
        });
    }

    // Variables declaration - do not modify
    private ui.jp.ArithmeticExpressionPanel
arithmeticExpressionPanel1;
    private ui.jp.InfiniteLoopPanel infiniteLoopPanel1;
    private ui.jp.InfiniteRecursionPanel
infiniteRecursionPanel1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JTabbedPane jTabbedPane1;
    private ui.jp.NullPointerPanel nullPointerPanel1;
    private ui.jp.OffByOnePanel offByOnePanel1;
    // End of variables declaration
}
```

2.1.10.3 File: package-info.java

```
/**
 * A package for classes related to LogicErrorTool's user
interface.
 */
package ui;
```

2.1.11 Package: ui.jp

2.1.11.1 File: ArithmeticExpressionPanel.java

```
package ui.jp;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.HashMap;
import java.util.Map;
import javax.swing.DefaultListModel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.ListModel;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import maths.ae.Node;
import maths.ae.Operation;

/**
 * A JPanel where the user can check for arithmetic expression
error.
 */
public class ArithmeticExpressionPanel extends
javax.swing.JPanel {

    private final Map<String, Node> node_namesToVals;

    /**
     * Creates new form ArithmeticExpressionPanel
     */
    public ArithmeticExpressionPanel() {
        initComponents();

        node_namesToVals = new HashMap<>();

        for (String name : new String[]{"RESULT", "STORAGE"})
{
            node_namesToVals.put(name, new Node("0"));
        }
    }
```

```
    /**
     * This method is called from within the constructor to
initialize the form.
     * WARNING: Do NOT modify this code. The content of this
method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated
Code">
    private void initComponents() {

        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        operationComboBox = new javax.swing.JComboBox<>();
        jLabel3 = new javax.swing.JLabel();
        aTextField = new javax.swing.JTextField();
        jLabel4 = new javax.swing.JLabel();
        bTextField = new javax.swing.JTextField();
        jLabel5 = new javax.swing.JLabel();
        addButton = new javax.swing.JButton();
        removeButton = new javax.swing.JButton();
        jLabel6 = new javax.swing.JLabel();
        expressionTextField = new javax.swing.JTextField();
        checkButton = new javax.swing.JButton();
        jScrollPane2 = new javax.swing.JScrollPane();
        instructionList = new javax.swing.JList<>();
        jLabel7 = new javax.swing.JLabel();
        jLabel8 = new javax.swing.JLabel();
        jLabel9 = new javax.swing.JLabel();

        jLabel1.setText("Instruction");

        jLabel2.setText("Operation");

        operationComboBox.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "ADD",
"DIVIDE", "EXPONENTIATE", "MULTIPLY", "STORE", "SUBTRACT" }));

        jLabel3.setText("a");

        aTextField.setName("a"); // NOI18N

        jLabel4.setText("b");

        bTextField.setName("b"); // NOI18N

        jLabel5.setText("Instructions");

        addButton.setText("Add");
```

```
        addButton.setEnabled(false);

        removeButton.setText("Remove");
        removeButton.setEnabled(false);

        jLabel6.setText("Expression");

        expressionTextField.setName(""); // NOI18N

        checkButton.setText("Check");

        jScrollPane2.setViewportView(instructionList);
        instructionList.setModel(new DefaultListModel<>());
        instructionList.addListSelectionListener(new
ListSelectionListenerImpl());

        jLabel7.setText("Special variables:");

        jLabel8.setText("• RESULT");

        jLabel9.setText("• STORAGE");

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(this);
        this.setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jLabel1)

.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)

.addGroup(layout.createSequentialGroup()
                                    .addComponent(jLabel2)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addComponent(operationComboBox,
```

```
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                                    .addComponent(jLabel4)
                                    .addComponent(jLabel3))

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)

.addGroup(layout.createSequentialGroup()
                                        .addGap(6, 6, 6)

.addComponent(bTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, 80,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addComponent(jLabel9))

.addGroup(layout.createSequentialGroup()

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addComponent(aTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, 80,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)

.addComponent(jLabel7)

.addComponent(jLabel8)))))
                            .addComponent(addButton)

.addGroup(layout.createSequentialGroup()
                                .addComponent(jLabel6)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
```

```
.addComponent(expressionTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, 120,
javax.swing.GroupLayout.PREFERRED_SIZE))
                                .addComponent(checkButton))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                                .addComponent(jLabel5)
                                .addComponent(jScrollPane2,
javax.swing.GroupLayout.PREFERRED_SIZE, 248,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addComponent(removeButton))
                            .addGap(0, 0, Short.MAX_VALUE))))
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(jLabel1)
                    .addComponent(jLabel5))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                    .addComponent(jScrollPane2,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                                .addComponent(jLabel2)
                                .addComponent(operationComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                                    .addComponent(jLabel3)
                                    .addComponent(aTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UN
RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                                    .addComponent(jLabel4)
                                    .addComponent(bTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jLabel9)))

.addGroup(layout.createSequentialGroup()
                                .addComponent(jLabel7)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                                .addComponent(jLabel8)))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                        .addComponent(addButton)))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                            .addComponent(expressionTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jLabel6))
```

```
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                            .addComponent(checkButton))
                        .addComponent(removeButton))

.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );

        operationComboBox.addActionListener(new
ComboBoxActionListener());
        (aTextField.getDocument()).addDocumentListener(new
OperandTextFieldDocumentListenerImpl());
        (bTextField.getDocument()).addDocumentListener(new
OperandTextFieldDocumentListenerImpl());
        addButton.addActionListener(new
AddButtonActionListenerImpl());
        removeButton.addActionListener(new
RemoveButtonActionListenerImpl());
        checkButton.addActionListener(new
CheckButtonActionListenerImpl(this));
    }// </editor-fold>


    // Variables declaration - do not modify
    private javax.swing.JTextField aTextField;
    private javax.swing.JButton addButton;
    private javax.swing.JTextField bTextField;
    private javax.swing.JButton checkButton;
    private javax.swing.JTextField expressionTextField;
    private javax.swing.JList<String> instructionList;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JComboBox<String> operationComboBox;
    private javax.swing.JButton removeButton;
    // End of variables declaration

    private void updateAddButton() {
        boolean shouldBeEnabled = true;

        if (!((String)
operationComboBox.getSelectedItem()).equals("STORE")) {
```

```
            for (JTextField operandTextField : new
JTextField[]{aTextField, bTextField}) {
                String operand = operandTextField.getText();

                for (String s : new String[]{"(", ",", " ",
")"}) {
                    operand = operand.replace(s, "");
                }

                if (operand.isEmpty()) {
                    shouldBeEnabled = false;
                    break;
                }
            }
        }

        addButton.setEnabled(shouldBeEnabled);
    }

    private class AddButtonActionListenerImpl implements
ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {
            String operationString
                    = (String)
operationComboBox.getSelectedItem();
            String instructionString = operationString;

            if (!operationString.equals("STORE")) {
                instructionString += "(" +
aTextField.getText() + ", "
                        + bTextField.getText() + ")";
            }

            ((DefaultListModel<String>)
instructionList.getModel()).addElement(instructionString);
        }
    }

    private class RemoveButtonActionListenerImpl implements
ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {
            ((DefaultListModel)
instructionList.getModel()).remove(instructionList.getSelected
Index());
        }
    }
```

```
    private class CheckButtonActionListenerImpl implements
ActionListener {

        ArithmeticExpressionPanel panel;


CheckButtonActionListenerImpl(ArithmeticExpressionPanel panel)
{
            this.panel = panel;
        }

        @Override
        public void actionPerformed(ActionEvent e) {
            ListModel<String> model =
instructionList.getModel();

            for (int i = 0; i < model.getSize(); i++) {
                String instruction = model.getElementAt(i);

                switch (instruction) {
                    case "STORE":
                        try {
                            //clone result
                            node_namesToVals.put("STORAGE",
(Node) (node_namesToVals.get("RESULT")).clone());
                        } catch (CloneNotSupportedException
ex) {
                            throw new RuntimeException(ex);
                        }
                        break;
                    default:
                        //instruction format:
"<OPERATION>(<operand 1>, <operand 2>)"
                        String[] instructionParts =
instruction.split("\\(");
                        instructionParts[1] =
instructionParts[1].replace(")", "");
                        String[] operandStrings =
instructionParts[1].split(",");
                        operandStrings[1] =
operandStrings[1].trim();
                        int numOperands =
operandStrings.length;
                        Node[] operandNodes = new
Node[numOperands];

                        for (int j = 0; j < numOperands; j++)
{
                            String operandString =
operandStrings[j];
```

```
                            operandNodes[j] =
operandString.equals("RESULT") ||
operandString.equals("STORAGE") ?
node_namesToVals.get(operandString) : new Node(operandString);
                        }

                        node_namesToVals.put("RESULT", new
Node(Operation.valueOf(instructionParts[0]), operandNodes[0],
operandNodes[1]));
                    }
            }

            String result =
(node_namesToVals.get("RESULT")).toString();

            if
((trimAll(expressionTextField.getText())).equals(trimAll(resul
t))) {
                JOptionPane.showMessageDialog(panel,
"Expression is correct");
            } else if (JOptionPane.showConfirmDialog(panel,
"Expression is incorrect. Change to '" + result + "'?", "Error
check", JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION)
{
                expressionTextField.setText(result);
            }
        }

        //remove all spaces from string
        String trimAll(String str) {
            return str.replace(" ", "");
        }
    }

    private class ComboBoxActionListener implements
ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {
            updateAddButton();
        }
    }

    private class OperandTextFieldDocumentListenerImpl
            implements DocumentListener {

        @Override
        public void insertUpdate(DocumentEvent e) {
            updateAddButton();
        }
```

```
        @Override
        public void removeUpdate(DocumentEvent e) {
            updateAddButton();
        }

        @Override
        public void changedUpdate(DocumentEvent e) {
            updateAddButton();
        }
    }

    private class ListSelectionListenerImpl implements
ListSelectionListener {

        @Override
        public void valueChanged(ListSelectionEvent e) {

removeButton.setEnabled(instructionList.getSelectedIndex() !=
-1);
        }
    }
}
```

2.1.11.2 File: InfiniteLoopPanel.java

```
package ui.jp;

import ui.lec.CorrectionOptionDialog;
import ui.AbstractTableModelImpl;
import cs.Guard;
import cs.loop.ArithmeticLoop;
import cs.loop.GeometricLoop;
import cs.loop.Loop;
import cs.loop.cg.ArithmeticCorrectionGenerator;
import cs.loop.cg.CorrectionGenerator;
import cs.loop.cg.GeometricCorrectionGenerator;
import cs.loop.ic.ArithmeticIterationCounter;
import cs.loop.ic.GeometricIterationCounter;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.HashMap;
import java.util.Map;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import maths.Relation;
import maths.is.ArithmeticInfiniteSequence;
import maths.is.GeometricInfiniteSequence;
import maths.is.InfiniteSequence;
import java.util.List;
import ui.lec.CorrectionRequestListener;
```

```
/**
 * A JPanel where the user can check for infinite loop error.
 */
public class InfiniteLoopPanel extends javax.swing.JPanel
implements CorrectionRequestListener {

    private final Map<String, Integer> intInput_namesToVals;

    /**
     * Creates new form InfiniteLoopPanel
     */
    public InfiniteLoopPanel() {
        initComponents();
        intInput_namesToVals = new HashMap<>();

        for (JTextField textField : new
JTextField[]{aTextField, bTextField, operandTextField}) {
            intInput_namesToVals.put(textField.getName(),
null);
        }
    }

    /**
     * This method is called from within the constructor to
initialize the form.
     * WARNING: Do NOT modify this code. The content of this
method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated
Code">
    private void initComponents() {

        jLabel1 = new javax.swing.JLabel();
        aTextField = new javax.swing.JTextField();
        jLabel2 = new javax.swing.JLabel();
        RComboBox = new javax.swing.JComboBox<>();
        bTextField = new javax.swing.JTextField();
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        operatorComboBox = new javax.swing.JComboBox<>();
        operandTextField = new javax.swing.JTextField();
        jLabel5 = new javax.swing.JLabel();
        checkButton = new javax.swing.JButton();
        jLabel6 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        jLabel8 = new javax.swing.JLabel();
        operandLabel = new javax.swing.JLabel();
```

```
        jLabel1.setText("i :=");

        aTextField.setName("a"); // NOI18N

        jLabel2.setText("WHILE i");

        RComboBox.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "<", "≤",
">", "≥", "≠", "=" }));

        bTextField.setName("b"); // NOI18N

        jLabel3.setText("//do something with i");

        jLabel4.setText("i := i");

        operatorComboBox.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "+", "*"
}));

        operandTextField.setName("operand"); // NOI18N

        jLabel5.setText("END_WHILE");

        checkButton.setText("Check");
        checkButton.setEnabled(false);

        jLabel6.setText("a");

        jLabel7.setText("R");

        jLabel8.setText("b");

        operandLabel.setText("d");

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(this);
        this.setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                    .addComponent(checkButton)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(28, 28, 28)
                        .addComponent(jLabel6))
```

```
                    .addGroup(layout.createSequentialGroup()
                        .addGap(53, 53, 53)
                        .addComponent(jLabel7)
                        .addGap(64, 64, 64)
                        .addComponent(jLabel8))
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jLabel1)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                        .addComponent(aTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, 80,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jLabel2)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                            .addComponent(jLabel3)

.addGroup(layout.createSequentialGroup()
                                .addComponent(RComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                                .addComponent(bTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, 80,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                    .addComponent(jLabel5)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(53, 53, 53)
                        .addComponent(jLabel4)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                        .addComponent(operatorComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
```

```
                                .addComponent(operandLabel)
                                .addComponent(operandTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, 80,
javax.swing.GroupLayout.PREFERRED_SIZE))))

.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel6)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(jLabel1)
                    .addComponent(aTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(jLabel7)
                    .addComponent(jLabel8))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(jLabel2)
                    .addComponent(RComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(bTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
```

```
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                .addComponent(jLabel3)
                .addGap(5, 5, 5)
                .addComponent(operandLabel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(jLabel4)
                    .addComponent(operatorComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(operandTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                .addComponent(jLabel5)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                .addComponent(checkButton)

.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );

        (aTextField.getDocument()).addDocumentListener(new
DocumentListenerImpl(aTextField));
        (bTextField.getDocument()).addDocumentListener(new
DocumentListenerImpl(bTextField));
        operatorComboBox.addActionListener(new
ComboBoxActionListenerImpl());

(operandTextField.getDocument()).addDocumentListener(new
DocumentListenerImpl(operandTextField));
        checkButton.addActionListener(new
ButtonActionListenerImpl(this));
    }// </editor-fold>

    // Variables declaration - do not modify
    private javax.swing.JComboBox<String> RComboBox;
    private javax.swing.JTextField aTextField;
    private javax.swing.JTextField bTextField;
```

```
    private javax.swing.JButton checkButton;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel operandLabel;
    private javax.swing.JTextField operandTextField;
    private javax.swing.JComboBox<String> operatorComboBox;
    // End of variables declaration

    private boolean plusOperatorIsSelected() {
        return
(operatorComboBox.getSelectedItem()).equals("+");
    }

    @Override
    public void correctionRequested(Map<String, Object>
namesToValues) {
        intInput_namesToVals.put("a", (Integer)
namesToValues.get("a"));
        intInput_namesToVals.put("b", (Integer)
namesToValues.get("b"));
        String operandName = plusOperatorIsSelected() ? "d" :
"r";
        intInput_namesToVals.put((operandName), (Integer)
namesToValues.get(operandName));

aTextField.setText(Integer.toString(intInput_namesToVals.get("
a")));
        RComboBox.setSelectedItem(namesToValues.get("R"));

bTextField.setText(Integer.toString(intInput_namesToVals.get("
b")));

operandTextField.setText(Integer.toString(intInput_namesToVals
.get(operandName)));
    }

    private class ButtonActionListenerImpl implements
ActionListener {

        InfiniteLoopPanel panel;

        ButtonActionListenerImpl(InfiniteLoopPanel panel) {
            this.panel = panel;
        }
```

```
        @Override
        public void actionPerformed(ActionEvent e) {
            Loop loop = getLoop();

            if (loop.terminates()) {
                JOptionPane.showMessageDialog(panel, "Loop is
finite (" + getIterationCount(loop) + " iteration(s))");
            } else {
                (new CorrectionOptionDialog("Loop is
infinite", new
AbstractTableModelImpl(getRowData_correctionTable(loop),
getColumnData_correctionTable()), panel)).setVisible(true);
            }
        }

        Loop getLoop() {
            Guard guard = new
Guard(Relation.values()[RComboBox.getSelectedIndex()],
intInput_namesToVals.get("b"));
            int a = intInput_namesToVals.get("a"), operand =
intInput_namesToVals.get("operand");
            return plusOperatorIsSelected() ? new
ArithmeticLoop(new ArithmeticInfiniteSequence(a, operand),
guard) : new GeometricLoop(new GeometricInfiniteSequence(a,
operand), guard);
        }

        int getIterationCount(Loop loop) {
            return (plusOperatorIsSelected() ? new
ArithmeticIterationCounter((ArithmeticLoop) loop) : new
GeometricIterationCounter((GeometricLoop) loop)).getCount();
        }

        Object[][] getRowData_correctionTable(Loop loop) {
            CorrectionGenerator generator =
plusOperatorIsSelected() ? new
ArithmeticCorrectionGenerator((ArithmeticLoop) loop) : new
GeometricCorrectionGenerator((GeometricLoop) loop);
            List<Loop> corrections =
generator.getCorrections();
            int numRows = corrections.size();
            Object[][] rowData = new
Object[numRows][getColumnData_correctionTable().length];

            for (int row = 0; row < numRows; row++) {
                Loop correction = corrections.get(row);
                InfiniteSequence is = correction.getIS();
                Guard guard = correction.getGuard();
                //using string version of R fixes issue with R
combo box not updating when user chooses to make correction
that changes R
```

```
                rowData[row] = new Object[]{is.getA(),
(guard.getR()).toString(), guard.getB(),
plusOperatorIsSelected() ? ((ArithmeticInfiniteSequence)
is).getD() : ((GeometricInfiniteSequence) is).getR(),
getIterationCount(correction)};
            }

            return rowData;
        }

        Object[] getColumnData_correctionTable() {
            return new Object[]{"a", "R", "b",
(plusOperatorIsSelected() ? "d" : "r"), "Iterations"};
        }
    }

    private class ComboBoxActionListenerImpl implements
ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {
            operandLabel.setText(plusOperatorIsSelected() ?
"d" : "r");
        }
    }

    private class DocumentListenerImpl implements
DocumentListener {

        JTextField textField;

        DocumentListenerImpl(JTextField textField) {
            this.textField = textField;
        }

        @Override
        public void insertUpdate(DocumentEvent e) {
            handleUpdate();
        }

        @Override
        public void removeUpdate(DocumentEvent e) {
            handleUpdate();
        }

        @Override
        public void changedUpdate(DocumentEvent e) {
            handleUpdate();
        }

        void handleUpdate() {
```

```
            Integer val;

            try {
                val = Integer.parseInt(textField.getText());
            } catch (NumberFormatException e) {
                val = null;
            }

            intInput_namesToVals.put(textField.getName(),
val);

checkButton.setEnabled(!intInput_namesToVals.containsValue(nul
l));
        }
    }
}
```

2.1.11.3 File: InfiniteRecursionPanel.java

```
package ui.jp;

import ui.lec.CorrectionOptionDialog;
import ui.AbstractTableModelImpl;
import cs.Guard;
import cs.loop.ArithmeticLoop;
import cs.loop.GeometricLoop;
import cs.loop.Loop;
import cs.loop.cg.ArithmeticCorrectionGenerator;
import cs.loop.cg.CorrectionGenerator;
import cs.loop.cg.GeometricCorrectionGenerator;
import cs.loop.ic.ArithmeticIterationCounter;
import cs.loop.ic.GeometricIterationCounter;
import cs.rmc.ArithmeticRecursiveMethodCall;
import cs.rmc.GeometricRecursiveMethodCall;
import cs.rmc.RecursiveMethodCall;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.HashMap;
import java.util.Map;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import maths.Relation;
import maths.is.ArithmeticInfiniteSequence;
import maths.is.GeometricInfiniteSequence;
import maths.is.InfiniteSequence;
import java.util.List;
import ui.lec.CorrectionRequestListener;

/**
```

```
 * A JPanel where the user can check for infinite recursion
error.
 */
public class InfiniteRecursionPanel extends javax.swing.JPanel
implements CorrectionRequestListener {

    private final Map<String, Integer> intInput_namesToVals;

    /**
     * Creates new form InfiniteRecursionPanel
     */
    public InfiniteRecursionPanel() {
        initComponents();
        intInput_namesToVals = new HashMap<>();

        for (JTextField textField : new
JTextField[]{aTextField, bTextField, operandTextField}) {
            intInput_namesToVals.put(textField.getName(),
null);
        }
    }

    /**
     * This method is called from within the constructor to
initialize the form.
     * WARNING: Do NOT modify this code. The content of this
method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated
Code">
    private void initComponents() {

        jLabel1 = new javax.swing.JLabel();
        aTextField = new javax.swing.JTextField();
        jLabel2 = new javax.swing.JLabel();
        RComboBox = new javax.swing.JComboBox<>();
        bTextField = new javax.swing.JTextField();
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        operatorComboBox = new javax.swing.JComboBox<>();
        operandTextField = new javax.swing.JTextField();
        jLabel5 = new javax.swing.JLabel();
        checkButton = new javax.swing.JButton();
        jLabel6 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        jLabel8 = new javax.swing.JLabel();
        jLabel9 = new javax.swing.JLabel();
        jLabel10 = new javax.swing.JLabel();
        jLabel11 = new javax.swing.JLabel();
```

```
        jLabel12 = new javax.swing.JLabel();
        jLabel13 = new javax.swing.JLabel();
        operandLabel = new javax.swing.JLabel();
        jLabel15 = new javax.swing.JLabel();

        jLabel1.setText("METHOD recursiveMethod(i)");

        aTextField.setName("a"); // NOI18N

        jLabel2.setText("IF i");

        RComboBox.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "<", "≤",
">", "≥", "≠", "=" }));

        bTextField.setName("b"); // NOI18N

        jLabel3.setText("THEN");

        jLabel4.setText("//don't recurse");

        operatorComboBox.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "+", "*"
}));

        operandTextField.setName("operand"); // NOI18N

        jLabel5.setText("ELSE");

        checkButton.setText("Check");
        checkButton.setEnabled(false);

        jLabel6.setText("//call recursiveMethod(i");

        jLabel7.setText(")");

        jLabel8.setText("END_IF");

        jLabel9.setText("END_METHOD");

        jLabel10.setText("//at some point in program, call
recursiveMethod(");

        jLabel11.setText(")");

        jLabel12.setText("b");

        jLabel13.setText("a");

        operandLabel.setText("d");
```

```
        jLabel15.setText("R");

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(this);
        this.setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                    .addComponent(jLabel1)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(31, 31, 31)
                        .addComponent(jLabel15)
                        .addGap(64, 64, 64)
                        .addComponent(jLabel12))
                    .addComponent(checkButton)
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                            .addComponent(jLabel10)
                            .addComponent(jLabel9))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                            .addComponent(jLabel13)

.addGroup(layout.createSequentialGroup()
                                .addComponent(aTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, 80,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                                .addComponent(jLabel11))))
                    .addGroup(layout.createSequentialGroup()
                        .addGap(6, 6, 6)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)

.addGroup(layout.createSequentialGroup()
                                .addComponent(jLabel2)
```

```
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                                        .addComponent(jLabel4)

.addGroup(layout.createSequentialGroup()

.addComponent(RComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addComponent(bTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, 80,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addComponent(jLabel3))))
                              .addComponent(jLabel5)
                              .addComponent(jLabel8)

.addGroup(layout.createSequentialGroup()
                                   .addGap(31, 31, 31)
                                   .addComponent(jLabel6)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addComponent(operatorComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)

.addComponent(operandLabel)

.addGroup(layout.createSequentialGroup()
```

```
.addComponent(operandTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, 80,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addComponent(jLabel7)))))))

.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel1)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(jLabel12)
                    .addComponent(jLabel15))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(jLabel2)
                    .addComponent(RComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(bTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel3))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                .addComponent(jLabel4)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
```

```
                .addComponent(jLabel5)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                .addComponent(operandLabel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(jLabel6)
                    .addComponent(operatorComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(operandTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel7))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                .addComponent(jLabel8)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                .addComponent(jLabel9)
                .addGap(4, 4, 4)
                .addComponent(jLabel13)
                .addGap(2, 2, 2)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(jLabel10)
                    .addComponent(aTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel11))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                .addComponent(checkButton)

.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );
```

```
        (aTextField.getDocument()).addDocumentListener(new
DocumentListenerImpl(aTextField));
        (bTextField.getDocument()).addDocumentListener(new
DocumentListenerImpl(bTextField));
        operatorComboBox.addActionListener(new
ComboBoxActionListenerImpl());

(operandTextField.getDocument()).addDocumentListener(new
DocumentListenerImpl(operandTextField));
        checkButton.addActionListener(new
ButtonActionListenerImpl(this));
    }// </editor-fold>

    // Variables declaration - do not modify
    private javax.swing.JComboBox<String> RComboBox;
    private javax.swing.JTextField aTextField;
    private javax.swing.JTextField bTextField;
    private javax.swing.JButton checkButton;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel15;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JLabel operandLabel;
    private javax.swing.JTextField operandTextField;
    private javax.swing.JComboBox<String> operatorComboBox;
    // End of variables declaration

    private boolean plusOperatorIsSelected() {
        return
(operatorComboBox.getSelectedItem()).equals("+");
    }

    @Override
    public void correctionRequested(Map<String, Object>
namesToValues) {
        intInput_namesToVals.put("a", (Integer)
namesToValues.get("a"));
        intInput_namesToVals.put("b", (Integer)
namesToValues.get("b"));
        String operandName = plusOperatorIsSelected() ? "d" :
"r";
```

```
        intInput_namesToVals.put((operandName), (Integer)
namesToValues.get(operandName));

aTextField.setText(Integer.toString(intInput_namesToVals.get("
a")));
        RComboBox.setSelectedItem(namesToValues.get("R"));

bTextField.setText(Integer.toString(intInput_namesToVals.get("
b")));

operandTextField.setText(Integer.toString(intInput_namesToVals
.get(operandName)));
    }

    private class ButtonActionListenerImpl implements
ActionListener {

        InfiniteRecursionPanel panel;

        ButtonActionListenerImpl(InfiniteRecursionPanel panel)
{
            this.panel = panel;
        }

        @Override
        public void actionPerformed(ActionEvent e) {
            Loop loop = getLoop();

            if (loop.terminates()) {
                JOptionPane.showMessageDialog(panel,
"Recursion is finite (" + getIterationCount(loop) + "
recursive call(s))");
            } else {
                (new CorrectionOptionDialog("Recursion is
infinite", new
AbstractTableModelImpl(getRowData_correctionTable(loop),
getColumnData_correctionTable()), panel)).setVisible(true);
            }
        }

        Loop getLoop() {
            Guard baseCase = new
Guard(Relation.values()[RComboBox.getSelectedIndex()],
intInput_namesToVals.get("b"));
            int a = intInput_namesToVals.get("a"), operand =
intInput_namesToVals.get("operand");
            RecursiveMethodCall call =
plusOperatorIsSelected() ? new
ArithmeticRecursiveMethodCall(new
ArithmeticInfiniteSequence(a, operand), baseCase) : new
```

```
GeometricRecursiveMethodCall(new GeometricInfiniteSequence(a,
operand), baseCase);
            return call.toLoop();
        }

        int getIterationCount(Loop loop) {
            return (plusOperatorIsSelected() ? new
ArithmeticIterationCounter((ArithmeticLoop) loop) : new
GeometricIterationCounter((GeometricLoop) loop)).getCount();
        }

        Object[][] getRowData_correctionTable(Loop loop) {
            CorrectionGenerator generator =
plusOperatorIsSelected() ? new
ArithmeticCorrectionGenerator((ArithmeticLoop) loop) : new
GeometricCorrectionGenerator((GeometricLoop) loop);
            List<Loop> corrections =
generator.getCorrections();
            int numRows = corrections.size();
            Object[][] rowData = new
Object[numRows][getColumnData_correctionTable().length];

            for (int row = 0; row < numRows; row++) {
                Loop correction = corrections.get(row);
                InfiniteSequence is = correction.getIS();
                Guard guard = correction.getGuard();
                rowData[row] = new Object[]{is.getA(),
((guard.getR()).getComplement()).toString(), guard.getB(),
plusOperatorIsSelected() ? ((ArithmeticInfiniteSequence)
is).getD() : ((GeometricInfiniteSequence) is).getR(),
getIterationCount(correction)};
            }

            return rowData;
        }

        Object[] getColumnData_correctionTable() {
            return new Object[]{"a", "R", "b",
(plusOperatorIsSelected() ? "d" : "r"), "Recursive calls"};
        }
    }

    private class ComboBoxActionListenerImpl implements
ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {
            operandLabel.setText(plusOperatorIsSelected() ?
"d" : "r");
        }
    }
```

```
    private class DocumentListenerImpl implements
DocumentListener {

        JTextField textField;

        DocumentListenerImpl(JTextField textField) {
            this.textField = textField;
        }

        @Override
        public void insertUpdate(DocumentEvent e) {
            handleUpdate();
        }

        @Override
        public void removeUpdate(DocumentEvent e) {
            handleUpdate();
        }

        @Override
        public void changedUpdate(DocumentEvent e) {
            handleUpdate();
        }

        void handleUpdate() {
            Integer val;

            try {
                val = Integer.parseInt(textField.getText());
            } catch (NumberFormatException e) {
                val = null;
            }

            intInput_namesToVals.put(textField.getName(),
val);

checkButton.setEnabled(!intInput_namesToVals.containsValue(nul
l));
        }
    }
}
```

2.1.11.4 File: NullPointerPanel.java

```
package ui.jp;

import ui.AbstractTableModelImpl;
import cs.Metaobject;
import java.awt.Component;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import javax.swing.DefaultListModel;
import javax.swing.JOptionPane;
import javax.swing.ListModel;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;

/**
 * A JPanel where the user can check for null pointer error.
 */
public class NullPointerPanel extends javax.swing.JPanel {

    private final VarTextFieldDocumentListenerImpl vtfdli;
    private List<Map<String, String>> initialStates;

    /**
     * Creates new form NullPointerPanel
     */
    public NullPointerPanel() {
        vtfdli = new VarTextFieldDocumentListenerImpl();
        initialStates = new ArrayList<>();
        initialStates.add(new HashMap<>());
        initComponents();
    }

    /**
     * This method is called from within the constructor to
initialize the form.
     * WARNING: Do NOT modify this code. The content of this
method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated
Code">
    private void initComponents() {

        jScrollPane2 = new javax.swing.JScrollPane();
        jTable1 = new javax.swing.JTable();
        jScrollPane1 = new javax.swing.JScrollPane();
        varList = new javax.swing.JList<>();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jScrollPane3 = new javax.swing.JScrollPane();
        initTable = new javax.swing.JTable();
        jLabel3 = new javax.swing.JLabel();
```

```
        jScrollPane4 = new javax.swing.JScrollPane();
        constructList = new javax.swing.JList<>();
        jLabel4 = new javax.swing.JLabel();
        addVarButton = new javax.swing.JButton();
        removeVarButton = new javax.swing.JButton();
        varTextField = new javax.swing.JTextField();
        jLabel5 = new javax.swing.JLabel();
        removeConstructButton = new javax.swing.JButton();
        addConstructButton = new javax.swing.JButton();
        jLabel6 = new javax.swing.JLabel();
        initTextField = new javax.swing.JTextField();
        removeInitButton = new javax.swing.JButton();
        addInitButton = new javax.swing.JButton();
        checkButton = new javax.swing.JButton();
        jLabel7 = new javax.swing.JLabel();
        jLabel9 = new javax.swing.JLabel();

        jTable1.setModel(new
javax.swing.table.DefaultTableModel(
            new Object [][] {
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null}
            },
            new String [] {
                "Title 1", "Title 2", "Title 3", "Title 4"
            }
        ));
        jScrollPane2.setViewportView(jTable1);


varList.setSelectionMode(javax.swing.ListSelectionModel.SINGLE
_SELECTION);
        jScrollPane1.setViewportView(varList);
        varList.setModel(new DefaultListModel<>());
        varList.addListSelectionListener(new
VarListSelectionListenerImpl());

        jLabel1.setText("String instance variables");

        jLabel2.setText("Selected constructor:
Initialisations");


initTable.setSelectionMode(javax.swing.ListSelectionModel.SING
LE_SELECTION);
        initTable.setModel(new ui.AbstractTableModelImpl(new
Object[][]{}, new Object[]{"Name", "Value"}));
        jScrollPane3.setViewportView(initTable);
```

```
(initTable.getSelectionModel()).addListSelectionListener(new
InitTableListSelectionListenerImpl());

        jLabel3.setText("Constructors");


constructList.setSelectionMode(javax.swing.ListSelectionModel.
SINGLE_SELECTION);
        jScrollPane4.setViewportView(constructList);
        updateConstructList();
        constructList.setSelectedIndex(0);
        constructList.addListSelectionListener(new
ConstructListSelectionListenerImpl());

        jLabel4.setText("Object-oriented class");

        addVarButton.setText("Add");
        addVarButton.setEnabled(false);

        removeVarButton.setText("Remove");
        removeVarButton.setEnabled(false);

        jLabel5.setText("Name");

        removeConstructButton.setText("Remove");
        removeConstructButton.setEnabled(false);

        addConstructButton.setText("Add");

        jLabel6.setText("Selected variable: Value: \"");

        initTextField.setEnabled(false);

        removeInitButton.setText("Remove");
        removeInitButton.setEnabled(false);

        addInitButton.setText("Add");
        addInitButton.setEnabled(false);

        checkButton.setText("Check");

        jLabel7.setText("\"");

        jLabel9.setText("//at some point in program,
instantiate class and call string instance method on each
variable");

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(this);
        this.setLayout(layout);
```

```
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(0, 0, Short.MAX_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                            .addComponent(addVarButton)

.addGroup(layout.createSequentialGroup()
                                .addComponent(jLabel5)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                                .addComponent(varTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, 80,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                                    .addComponent(jLabel1)
                                    .addComponent(jLabel4)

.addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE, 174,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addComponent(removeVarButton))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING, false)
                                    .addComponent(jLabel3)

.addGroup(layout.createSequentialGroup()

.addComponent(addConstructButton)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
```

```
.addComponent(removeConstructButton))

.addComponent(jScrollPane4,
javax.swing.GroupLayout.PREFERRED_SIZE, 174,
javax.swing.GroupLayout.PREFERRED_SIZE))))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING, false)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING, false)

.addComponent(addInitButton)

.addGroup(layout.createSequentialGroup()
                                        .addComponent(jLabel6)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addComponent(initTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, 80,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addComponent(removeInitButton)
                                    .addComponent(jLabel2,
javax.swing.GroupLayout.PREFERRED_SIZE, 246,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                                .addComponent(jLabel7))
                            .addComponent(jScrollPane3,
javax.swing.GroupLayout.PREFERRED_SIZE, 257,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                            .addComponent(jLabel9)
                            .addComponent(checkButton))
                        .addGap(0, 0, Short.MAX_VALUE)))
                .addContainerGap())
        );
        layout.setVerticalGroup(
```

```
layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
            .addGroup(layout.createSequentialGroup()

.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addComponent(jLabel4)
                .addGap(14, 14, 14)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                            .addComponent(jLabel1)
                            .addComponent(jLabel3))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING, false)
                            .addComponent(jScrollPane4,
javax.swing.GroupLayout.PREFERRED_SIZE, 0, Short.MAX_VALUE)
                            .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE, 126,
javax.swing.GroupLayout.PREFERRED_SIZE)))

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                        .addComponent(jLabel2)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                        .addComponent(jScrollPane3,
javax.swing.GroupLayout.PREFERRED_SIZE, 126,
javax.swing.GroupLayout.PREFERRED_SIZE)))

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                    .addGroup(layout.createSequentialGroup()

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                            .addComponent(removeVarButton)
```

```
.addComponent(removeConstructButton)
                                .addComponent(addConstructButton)
                                .addComponent(removeInitButton))
                            .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                                .addComponent(varTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addComponent(jLabel5))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                            .addComponent(addVarButton))
                        .addGroup(layout.createSequentialGroup()
                            .addGap(48, 48, 48)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                                .addComponent(jLabel6)
                                .addComponent(initTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addComponent(jLabel7))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                            .addComponent(addInitButton)))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                    .addComponent(jLabel9)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                    .addComponent(checkButton))
            );

            addVarButton.addActionListener(new
AddVarButtonActionListenerImpl());
            removeVarButton.addActionListener(new
RemoveVarButtonActionListenerImpl());

(varTextField.getDocument()).addDocumentListener(vtfdli);
            removeConstructButton.addActionListener(new
RemoveConstructButtonActionListenerImpl());
```

```
        addConstructButton.addActionListener(new
AddConstructButtonActionListenerImpl());
        removeInitButton.addActionListener(new
RemoveInitButtonActionListenerImpl());
        addInitButton.addActionListener(new
AddInitButtonActionListenerImpl());
        checkButton.addActionListener(new
CheckButtonActionListenerImpl(this));
    }// </editor-fold>


    // Variables declaration - do not modify
    private javax.swing.JButton addConstructButton;
    private javax.swing.JButton addInitButton;
    private javax.swing.JButton addVarButton;
    private javax.swing.JButton checkButton;
    private javax.swing.JList<String> constructList;
    private javax.swing.JTable initTable;
    private javax.swing.JTextField initTextField;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JScrollPane jScrollPane3;
    private javax.swing.JScrollPane jScrollPane4;
    private javax.swing.JTable jTable1;
    private javax.swing.JButton removeConstructButton;
    private javax.swing.JButton removeInitButton;
    private javax.swing.JButton removeVarButton;
    private javax.swing.JList<String> varList;
    private javax.swing.JTextField varTextField;
    // End of variables declaration

    private void updateAddVarButton() {
        String var = varTextField.getText();
        addVarButton.setEnabled(!(var.isEmpty() ||
((DefaultListModel) varList.getModel()).contains(var)));
    }

    private void updateRemoveVarButton() {
        if (varList.isSelectionEmpty()) {
            removeVarButton.setEnabled(false);
        } else {
            boolean noConstructorInitialisesVar = true;
```

```
            for (Map<String, String> initialState :
initialStates) {
                if
(initialState.containsKey(varList.getSelectedValue())) {
                    noConstructorInitialisesVar = false;
                    break;
                }
            }


removeVarButton.setEnabled(noConstructorInitialisesVar);
        }
    }

    private void updateAddInitButton() {
        addInitButton.setEnabled(!(varList.isSelectionEmpty()
|| constructList.isSelectionEmpty() ||
(initialStates.get(constructList.getSelectedIndex())).contains
Key(varList.getSelectedValue())));
    }

    private void updateConstructList() {
        int numInitialStates = initialStates.size();
        String[] constructListData = new
String[numInitialStates];

        for (int i = 0; i < numInitialStates; i++) {
            constructListData[i] = "c" + (i + 1);
        }

        constructList.setListData(constructListData);
    }

    private void updateInitTable() {
        int selectedIndex = constructList.getSelectedIndex();
        Object[] columnData = new Object[]{"Name", "Value"};
        Object[][] rowData;

        if (selectedIndex == -1) {
            rowData = new Object[][]{};
        } else {
            Map<String, String> initialState =
initialStates.get(selectedIndex);
            int numInitialisations = initialState.size();
            rowData = new
Object[numInitialisations][initTable.getColumnCount()];

            for (int i = 0; i < numInitialisations; i++) {
                rowData[i] = new Object[]{(String)
(initialState.keySet()).toArray()[i], (String)
(initialState.values()).toArray()[i]};
```

```
            }
        }

        initTable.setModel(new AbstractTableModelImpl(rowData,
columnData));
    }

    private class AddVarButtonActionListenerImpl implements
ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {
            ((DefaultListModel<String>)
varList.getModel()).addElement(varTextField.getText());
            updateAddVarButton();
        }
    }

    private class RemoveVarButtonActionListenerImpl implements
ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {
            ((DefaultListModel)
varList.getModel()).remove(varList.getSelectedIndex());
            updateAddVarButton();
        }
    }

    private class AddConstructButtonActionListenerImpl
implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {
            initialStates.add(new HashMap<>());
            updateConstructList();
        }
    }

    private class RemoveConstructButtonActionListenerImpl
implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {

initialStates.remove(constructList.getSelectedIndex());
            updateRemoveVarButton();
            updateConstructList();
        }
    }
```

```
    private class AddInitButtonActionListenerImpl implements
ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {

(initialStates.get(constructList.getSelectedIndex())).put(varL
ist.getSelectedValue(), "\"" + initTextField.getText() +
"\"");
            updateRemoveVarButton();
            updateAddInitButton();
            updateInitTable();
        }
    }

    private class RemoveInitButtonActionListenerImpl
implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {
            int nameColumn = 0;

(initialStates.get(constructList.getSelectedIndex())).remove((
String) initTable.getValueAt(initTable.getSelectedRow(),
nameColumn));
            updateRemoveVarButton();
            updateAddInitButton();
            updateInitTable();
        }
    }

    private class CheckButtonActionListenerImpl implements
ActionListener {

        Component panel;

        CheckButtonActionListenerImpl(Component panel) {
            this.panel = panel;
        }

        @Override
        public void actionPerformed(ActionEvent e) {
            List<String> vars = new ArrayList<>();
            ListModel model = varList.getModel();

            for (int i = 0; i < model.getSize(); i++) {
                vars.add((String) model.getElementAt(i));
            }

            Metaobject metaobj = new Metaobject(vars,
initialStates);
```

```
            List<String> uninitialisedVars =
metaobj.getUninitialisedVars();

            if (uninitialisedVars.isEmpty()) {
                JOptionPane.showMessageDialog(panel, "Any
variables that have been declared are initialised in all
constructors (no null pointer error)");
            } else {
                String message = "The following variables
aren't initialised in all constructors (null pointer error): "
+ uninitialisedVars.get(0);

                for (int i = 1; i < uninitialisedVars.size();
i++) {
                    message += ", " +
uninitialisedVars.get(i);
                }

                message += "\nInitialise to an empty string?";

                if (JOptionPane.showConfirmDialog(panel,
message, "Error check", JOptionPane.YES_NO_OPTION) ==
JOptionPane.YES_OPTION) {
                    metaobj.addMissingInitialisations();
                    initialStates =
metaobj.getInitialStates();
                    constructList.setSelectedIndex(0);
                    initTable.clearSelection();
                    updateConstructList();
                }
            }
        }
    }

    private class VarTextFieldDocumentListenerImpl implements
DocumentListener {

        @Override
        public void insertUpdate(DocumentEvent e) {
            updateAddVarButton();
        }

        @Override
        public void removeUpdate(DocumentEvent e) {
            updateAddVarButton();
        }

        @Override
        public void changedUpdate(DocumentEvent e) {
            updateAddVarButton();
        }
```

```
    }

    private class VarListSelectionListenerImpl implements
ListSelectionListener {

        @Override
        public void valueChanged(ListSelectionEvent e) {
            updateRemoveVarButton();
            updateAddInitButton();

initTextField.setEnabled(!varList.isSelectionEmpty());
        }
    }

    private class ConstructListSelectionListenerImpl
implements ListSelectionListener {

        @Override
        public void valueChanged(ListSelectionEvent e) {
            int minNumConstructs = 1;

removeConstructButton.setEnabled(!(constructList.isSelectionEm
pty() || initialStates.size() == minNumConstructs));
            updateAddInitButton();
            updateInitTable();
        }
    }

    private class InitTableListSelectionListenerImpl
implements ListSelectionListener {

        @Override
        public void valueChanged(ListSelectionEvent e) {

removeInitButton.setEnabled(!(initTable.getSelectionModel()).i
sSelectionEmpty());
        }
    }
}
```

2.1.11.5 File: OffByOnePanel.java

```
package ui.jp;

import ui.lec.CorrectionOptionDialog;
import ui.AbstractTableModelImpl;
import java.util.HashMap;
import java.util.Map;
import javax.swing.JTextField;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import cs.Guard;
```

```
import cs.loop.ArithmeticLoop;
import cs.loop.Loop;
import cs.ArrayPositionRange;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;
import javax.swing.JComboBox;
import javax.swing.JOptionPane;
import maths.Relation;
import maths.is.ArithmeticInfiniteSequence;
import maths.is.InfiniteSequence;
import ui.lec.CorrectionRequestListener;

/**
 * A JPanel where the user can check for off-by-one error.
 */
public class OffByOnePanel extends javax.swing.JPanel
implements CorrectionRequestListener {

    private final Map<String, Integer> intField_namesToVals;

    /**
     * Creates new form OffByOnePanel
     */
    public OffByOnePanel() {
        initComponents();
        intField_namesToVals = new HashMap<>();

        for (JTextField textField : new
JTextField[]{startTextField, endTextField, aTextField,
bTextField}) {
            intField_namesToVals.put(textField.getName(),
null);
        }
    }

    /**
     * This method is called from within the constructor to
initialize the form.
     * WARNING: Do NOT modify this code. The content of this
method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated
Code">
    private void initComponents() {

        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
```

```
        startTextField = new javax.swing.JTextField();
        endTextField = new javax.swing.JTextField();
        jLabel5 = new javax.swing.JLabel();
        jLabel6 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        jLabel8 = new javax.swing.JLabel();
        aTextField = new javax.swing.JTextField();
        RComboBox = new javax.swing.JComboBox<>();
        bTextField = new javax.swing.JTextField();
        dComboBox = new javax.swing.JComboBox<>();
        jLabel4 = new javax.swing.JLabel();
        jLabel9 = new javax.swing.JLabel();
        jLabel10 = new javax.swing.JLabel();
        jLabel11 = new javax.swing.JLabel();
        checkButton = new javax.swing.JButton();

        jLabel1.setText("Array position range (first position
is 1)");

        jLabel2.setText("Start position");

        jLabel3.setText("End position");

        startTextField.setName("Start position"); // NOI18N

        endTextField.setName("End position"); // NOI18N

        jLabel5.setText("Expected loop to traverse
corresponding array index range");

        jLabel6.setText("Initialisation: i =");

        jLabel7.setText("Guard: i");

        jLabel8.setText("Final expression: i = i +");

        aTextField.setName("a"); // NOI18N

        RComboBox.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "<", "≤",
">", "≥" }));
        RComboBox.setName("R"); // NOI18N

        bTextField.setName("b"); // NOI18N

        dComboBox.setModel(new
javax.swing.DefaultComboBoxModel<>(new String[] { "1", "-1"
}));
        dComboBox.setName("d"); // NOI18N

        jLabel4.setText("a");
```

```
        jLabel9.setText("R");

        jLabel10.setText("b");

        jLabel11.setText("d");

        checkButton.setText("Check");
        checkButton.setEnabled(false);

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(this);
        this.setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                            .addComponent(jLabel2)
                            .addComponent(jLabel3))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING, false)
                            .addComponent(endTextField,
javax.swing.GroupLayout.DEFAULT_SIZE, 80, Short.MAX_VALUE)
                            .addComponent(startTextField)))
                    .addComponent(jLabel1)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jLabel8)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                            .addComponent(dComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
.addGroup(layout.createSequentialGroup()
                                    .addGap(6, 6, 6)
                                    .addComponent(jLabel11))))
                        .addComponent(checkButton)
                        .addGroup(layout.createSequentialGroup()
                            .addComponent(jLabel7)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                                .addComponent(RComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addComponent(jLabel9))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                                .addComponent(jLabel10)
                                .addComponent(bTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, 80,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                        .addGroup(layout.createSequentialGroup()
                            .addComponent(jLabel6)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                                .addComponent(jLabel4)
                                .addComponent(aTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, 80,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                        .addComponent(jLabel5))

.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
```

```
                .addComponent(jLabel1)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(startTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel2))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(endTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel3))
                .addGap(18, 18, 18)
                .addComponent(jLabel5)
                .addGap(4, 4, 4)
                .addComponent(jLabel4)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(jLabel6)
                    .addComponent(aTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(jLabel9)
                    .addComponent(jLabel10))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(RComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(bTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel7))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                .addComponent(jLabel11)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.BASELINE)
                    .addComponent(jLabel8)
                    .addComponent(dComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                .addComponent(checkButton)

.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );

        (startTextField.getDocument()).addDocumentListener(new
ArrayFieldDocumentListenerImpl(startTextField));
        (endTextField.getDocument()).addDocumentListener(new
ArrayFieldDocumentListenerImpl(endTextField));
        (aTextField.getDocument()).addDocumentListener(new
LoopFieldDocumentListenerImpl(aTextField));
        (bTextField.getDocument()).addDocumentListener(new
LoopFieldDocumentListenerImpl(bTextField));
        checkButton.addActionListener(new
CheckButtonActionListenerImpl(this));
    }// </editor-fold>


    // Variables declaration - do not modify
    private javax.swing.JComboBox<String> RComboBox;
```

```
    private javax.swing.JTextField aTextField;
    private javax.swing.JTextField bTextField;
    private javax.swing.JButton checkButton;
    private javax.swing.JComboBox<String> dComboBox;
    private javax.swing.JTextField endTextField;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JTextField startTextField;
    // End of variables declaration

    @Override
    public void correctionRequested(Map<String, Object>
namesToValues) {
        for (String key : new String[]{"a", "b"}) {
            intField_namesToVals.put(key, (Integer)
namesToValues.get(key));
        }

        for (JTextField textField : new
JTextField[]{aTextField, bTextField}) {
            textField.setText("" +
intField_namesToVals.get(textField.getName()));
        }

        for (JComboBox comboBox : new JComboBox[]{RComboBox,
dComboBox}) {

comboBox.setSelectedItem(namesToValues.get(comboBox.getName())
);
        }
    }

    private class CheckButtonActionListenerImpl implements
ActionListener {

        OffByOnePanel panel;

        CheckButtonActionListenerImpl(OffByOnePanel panel) {
            this.panel = panel;
        }

        @Override
```

```
        public void actionPerformed(ActionEvent e) {
            ArrayPositionRange apr = new
ArrayPositionRange(intField_namesToVals.get("Start position"),
intField_namesToVals.get("End position"));
            List<Loop> loops = apr.getTraversingLoops();
            Loop expectedLoop = new ArithmeticLoop(new
ArithmeticInfiniteSequence(intField_namesToVals.get("a"),
Integer.parseInt((String) dComboBox.getSelectedItem())), new
Guard(Relation.values()[RComboBox.getSelectedIndex()],
intField_namesToVals.get("b")));

            for (Loop loop : loops) {
                if (loop.equals(expectedLoop)) {
                    JOptionPane.showMessageDialog(panel, "Loop
is correct (no off-by-one error)");
                    return;
                }
            }

            (new CorrectionOptionDialog("Loop is incorrect
(likely off-by-one error)", new
AbstractTableModelImpl(getRowData_correctionTable(apr),
getColumnData_correctionTable()), panel)).setVisible(true);
        }

        Object[][]
getRowData_correctionTable(ArrayPositionRange apr) {
            List<Loop> loops = apr.getTraversingLoops();
            int numRows = loops.size();
            Object[][] rowData = new
Object[numRows][getColumnData_correctionTable().length];

            for (int row = 0; row < numRows; row++) {
                Loop loop = loops.get(row);
                InfiniteSequence is = loop.getIS();
                Guard guard = loop.getGuard();
                rowData[row] = new Object[]{is.getA(),
(guard.getR()).toString(), guard.getB(),
((ArithmeticInfiniteSequence) is).getD()};
            }

            return rowData;
        }

        Object[] getColumnData_correctionTable() {
            return new Object[]{"a", "R", "b", "d"};
        }
    }

    private class ArrayFieldDocumentListenerImpl implements
DocumentListener {
```

```
        JTextField textField;

        ArrayFieldDocumentListenerImpl(JTextField textField) {
            this.textField = textField;
        }

        @Override
        public void insertUpdate(DocumentEvent e) {
            handleUpdate();
        }

        @Override
        public void removeUpdate(DocumentEvent e) {
            handleUpdate();
        }

        @Override
        public void changedUpdate(DocumentEvent e) {
            handleUpdate();
        }

        void handleUpdate() {
            Integer val;

            try {
                val = Integer.parseInt(textField.getText());

                if (val < 1) {
                    val = null;
                }
            } catch (NumberFormatException e) {
                val = null;
            }

            intField_namesToVals.put(textField.getName(),
val);

checkButton.setEnabled(!intField_namesToVals.containsValue(nul
l));
        }
    }

    private class LoopFieldDocumentListenerImpl implements
DocumentListener {

        JTextField textField;

        LoopFieldDocumentListenerImpl(JTextField textField) {
            this.textField = textField;
        }
```

```
        @Override
        public void insertUpdate(DocumentEvent e) {
            handleUpdate();
        }

        @Override
        public void removeUpdate(DocumentEvent e) {
            handleUpdate();
        }

        @Override
        public void changedUpdate(DocumentEvent e) {
            handleUpdate();
        }

        void handleUpdate() {
            Integer val;

            try {
                val = Integer.parseInt(textField.getText());
            } catch (NumberFormatException e) {
                val = null;
            }

            intField_namesToVals.put(textField.getName(),
val);

checkButton.setEnabled(!intField_namesToVals.containsValue(nul
l));
        }
    }
}
```

2.1.11.6 File: package-info.java

```
/**
 * A package for subclasses of JPanel.
 */
package ui.jp;
```

2.1.12 Package: ui.lec

2.1.12.1 File: CorrectionOptionDialog.java

```
package ui.lec;

import ui.AbstractTableModelImpl;
import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.HashMap;
```

```
import java.util.Map;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.table.TableModel;
import javax.swing.table.TableRowSorter;

/**
 * A dialog that displays logic error corrections, one of
which the user can
 * choose to make.
 */
public class CorrectionOptionDialog extends
javax.swing.JDialog {

    private CorrectionRequestListener crl;

    /**
     * The constructor to be called by a correction request
listener.
     *
     * @param errorMsg an error message to display
     * @param atmi a model for a table in which to display
corrections
     * @param crl the correction request listener that called
this constructor
     */
    public CorrectionOptionDialog(String errorMsg,
AbstractTableModelImpl atmi, CorrectionRequestListener crl) {
        this(new Frame(), true);
        errorLabel.setText(errorMsg);
        correctionTable.setModel(atmi);
        correctionTable.setRowSorter(new
TableRowSorter<>(atmi));
        this.crl = crl;
    }

    /**
     * Creates new form NewJDialog
     *
     * @param parent
     * @param modal
     */
    private CorrectionOptionDialog(java.awt.Frame parent,
boolean modal) {
        super(parent, modal);
        initComponents();
    }

    /**
     * This method is called from within the constructor to
initialize the form.
```

```
     * WARNING: Do NOT modify this code. The content of this
method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated
Code">
    private void initComponents() {

        errorLabel = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jScrollPane1 = new javax.swing.JScrollPane();
        correctionTable = new javax.swing.JTable();
        changeButton = new javax.swing.JButton();


setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_O
N_CLOSE);

        errorLabel.setText(" ");

        jLabel2.setText("Suggestions");

        correctionTable.setModel(new
javax.swing.table.DefaultTableModel(
            new Object [][] {

            },
            new String [] {

            }
        ));

correctionTable.setSelectionMode(javax.swing.ListSelectionMode
l.SINGLE_SELECTION);
        jScrollPane1.setViewportView(correctionTable);

(correctionTable.getSelectionModel()).addListSelectionListener
(new ListSelectionListenerImpl());

        changeButton.setText("Change");
        changeButton.setEnabled(false);

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
            .addGroup(layout.createSequentialGroup()
```

```
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                    .addComponent(jScrollPane1,
javax.swing.GroupLayout.DEFAULT_SIZE, 488, Short.MAX_VALUE)
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
lignment.LEADING)
                            .addComponent(errorLabel)
                            .addComponent(jLabel2)
                            .addComponent(changeButton))
                        .addGap(0, 0, Short.MAX_VALUE)))
                .addContainerGap())
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(errorLabel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                .addComponent(jLabel2)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED)
                .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE, 249,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RE
LATED, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(changeButton))
        );

        changeButton.addActionListener(new
ActionListenerImpl(this));

        pack();
    }// </editor-fold>

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
```

```
        //<editor-fold defaultstate="collapsed" desc=" Look
and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not
available, stay with the default look and feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel
/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException |
InstantiationException | IllegalAccessException |
javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(CorrectionOptionDialog.clas
s.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>

        //</editor-fold>
        //</editor-fold>

        /* Create and display the dialog */
        java.awt.EventQueue.invokeLater(() -> {
            CorrectionOptionDialog dialog = new
CorrectionOptionDialog(new javax.swing.JFrame(), true);
            dialog.addWindowListener(new
java.awt.event.WindowAdapter() {
                @Override
```

```
                public void
windowClosing(java.awt.event.WindowEvent e) {
                    System.exit(0);
                }
            });
            dialog.setVisible(true);
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton changeButton;
    private javax.swing.JTable correctionTable;
    private javax.swing.JLabel errorLabel;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JScrollPane jScrollPane1;
    // End of variables declaration

    private class ListSelectionListenerImpl implements
ListSelectionListener {

        @Override
        public void valueChanged(ListSelectionEvent e) {
            changeButton.setEnabled(true);
        }
    }

    private class ActionListenerImpl implements ActionListener
{

        CorrectionOptionDialog dialog;

        ActionListenerImpl(CorrectionOptionDialog dialog) {
            this.dialog = dialog;
        }

        @Override
        public void actionPerformed(ActionEvent e) {

crl.correctionRequested(getColumnNamesToVals_rowDatum(correcti
onTable.convertRowIndexToModel(correctionTable.getSelectedRow(
))));
            dialog.dispose();
        }

        Map<String, Object> getColumnNamesToVals_rowDatum(int
row) {
            Map<String, Object> columnNamesToVals = new
HashMap<>();
            TableModel tm = correctionTable.getModel();
```

```
            for (int column = 0; column < tm.getColumnCount();
column++) {

columnNamesToVals.put(tm.getColumnName(column),
tm.getValueAt(row, column));
            }

            return columnNamesToVals;
        }
    }
}
```

2.1.12.2 File: CorrectionRequestListener.java

```
package ui.lec;

import java.util.Map;

/**
 * An interface to be implemented by classes that generate a
correction option
 * dialog.
 */
public interface CorrectionRequestListener {

    /**
     * A callback mechanism. When the user clicks 'Change' in
a correction
     * option dialog, it calls this method with the selected
correction.
     *
     * @param namesToValues a mapping from names to correct
values for inputs to
     * the correction request listener
     */
    void correctionRequested(Map<String, Object>
namesToValues);
}
```

2.1.12.3 File: package-info.java

```
/**
 * A package for classes related to logic error correction via
LogicErrorTool's user interface.
 */
package ui.lec;
```

2.2 Test packages

2.2.1 Default package

2.2.1.1 File: TestSuite.java

```
import cs.loop.ArithmeticLoopTest;
import cs.loop.GeometricLoopTest;
import cs.loop.cg.ArithmeticCorrectionGeneratorTest;
import cs.loop.cg.GeometricCorrectionGeneratorTest;
import cs.loop.ic.ArithmeticIterationCounterTest;
import cs.loop.ic.GeometricIterationCounterTest;
import cs.npe.MetaobjectTest;
import cs.rmc.ArithmeticRecursiveMethodCallTest;
import cs.rmc.GeometricRecursiveMethodCallTest;
import maths.ae.NodeTest;
import maths.is.ArithmeticInfiniteSequenceTest;
import maths.is.GeometricInfiniteSequenceTest;
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

/**
 * A class to group all tests together.
 */
@RunWith(Suite.class)
@Suite.SuiteClasses({
    ArithmeticLoopTest.class,
    GeometricLoopTest.class,
    ArithmeticCorrectionGeneratorTest.class,
    GeometricCorrectionGeneratorTest.class,
    ArithmeticIterationCounterTest.class,
    GeometricIterationCounterTest.class,
    MetaobjectTest.class,
    ArithmeticRecursiveMethodCallTest.class,
    GeometricRecursiveMethodCallTest.class,
    NodeTest.class,
    ArithmeticInfiniteSequenceTest.class,
    GeometricInfiniteSequenceTest.class
})
public class TestSuite {
}
```

2.2.2 Package: cs

2.2.2.1 File: ArrayPositionRangeTest.java

```
package cs;

import cs.loop.ArithmeticLoop;
import cs.loop.Loop;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import maths.Relation;
import maths.is.ArithmeticInfiniteSequence;
import org.junit.Test;
```

```
import static org.junit.Assert.*;

/**
 * A class to test methods of ArrayPositionRange.
 */
public class ArrayPositionRangeTest {

    /**
     * A method to test generating loops to traverse the 0-
based array index
     * range corresponding to the array position range, in the
case that the
     * start position precedes the end position.
     */
    @Test
    public void testGetTraversingLoops_startBeforeEnd() {
        List<Loop> loops = new ArrayList<>();
        loops.addAll(Arrays.asList(new Loop[]{
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(0, 1), new Guard(Relation.LT, 10)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(0, 1), new Guard(Relation.LTOET,
9))
        }));
        assertEquals(loops, new ArrayPositionRange(1,
10).getTraversingLoops());
    }

    /**
     * A method to test generating loops to traverse the 0-
based array index
     * range corresponding to the array position range, in the
case that the
     * start position succeeds the end position.
     */
    @Test
    public void testGetTraversingLoops_startAfterEnd() {
        List<Loop> loops = new ArrayList<>();
        loops.addAll(Arrays.asList(new Loop[]{
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(9, -1), new Guard(Relation.GT, -
1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(9, -1), new Guard(Relation.GTOET,
0))
        }));
        assertEquals(loops, new ArrayPositionRange(10,
1).getTraversingLoops());
    }
}
```

2.2.2.2 File: MetaobjectTest.java

```
package cs;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 * A class to test methods of Metaobject.
 */
public class MetaobjectTest {

    /**
     * A method to test finding variables of the base object
implicitly
     * initialised to null, in the case that there are no
variables.
     */
    @Test
    public void testGetUninitialisedVars_noVars() {
        List<String> vars = new ArrayList<>();
        List<Map<String, String>> initialStates = new
ArrayList<>();
        initialStates.add(new HashMap<>());
        assertEquals(vars, (new Metaobject(vars,
initialStates)).getUninitialisedVars());
    }

    /**
     * A method to test finding variables of the base object
implicitly
     * initialised to null, in the case that there is a
variable that no
     * constructor initialises.
     */
    @Test
    public void
testGetUninitialisedVars_varInitialisedNowhere() {
        List<String> vars = new ArrayList<>();
        vars.add("var");
        List<Map<String, String>> initialStates = new
ArrayList<>();
        initialStates.add((new HashMap<>()));
        assertEquals(vars, (new Metaobject(vars,
initialStates)).getUninitialisedVars());
    }

    /**
```

```
     * A method to test finding variables of the base object
implicitly
     * initialised to null, in the case that there is a
variable that only 1 of
     * 2 constructors initialises.
     */
    @Test
    public void
testGetUninitialisedVars_varInitialisedNotEverywhere() {
        List<String> vars = new ArrayList<>();
        vars.add("var");
        List<Map<String, String>> initialStates = new
ArrayList<>();
        initialStates.add(new HashMap<>());
        Map<String, String> config = new HashMap<>();
        config.put("var", "");
        assertEquals(vars, (new Metaobject(vars,
initialStates)).getUninitialisedVars());
    }

    /**
     * A method to test finding variables of the base object
implicitly
     * initialised to null, in the case that there is a
variable that all
     * constructors initialise.
     */
    @Test
    public void
testGetVarsNotInitialisedInAllConstructors_varInitialisedEvery
where() {
        List<String> vars = new ArrayList<>();
        vars.add("var");
        List<Map<String, String>> initialStates = new
ArrayList<>();
        Map<String, String> config = new HashMap<>();
        config.put("var", "");
        assertEquals(new ArrayList<>(), (new Metaobject(vars,
initialStates)).getUninitialisedVars());
    }

    /**
     * A method to test initialising the base object's
variables to an empty
     * string wherever they are not initialised in a
constructor of the object.
     */
    @Test
    public void testAddMissingInitialisations() {
        List<String> vars = new ArrayList<>();
        vars.add("var1");
```

```
        vars.add("var2");
        Map<String, String> initialState1 = new HashMap<>();
        initialState1.put("var1", "");
        List<Map<String, String>> initialStates = new
ArrayList<>();
        initialStates.add(initialState1);
        Map<String, String> initialState2 = new HashMap<>();
        initialState2.put("var2", "");
        initialStates.add(initialState2);
        Metaobject metaobj = new Metaobject(vars,
initialStates);
        metaobj.addMissingInitialisations();
        assertEquals(new ArrayList<>(),
metaobj.getUninitialisedVars());
    }
}
```

2.2.3 Package: cs.loop

2.2.3.1 File: ArithmeticLoopTest.java

```
package cs.loop;

import cs.Guard;
import maths.Relation;
import maths.is.ArithmeticInfiniteSequence;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 * A class to test methods of ArithmeticLoop.
 */
public class ArithmeticLoopTest {

    /**
     * A method test termination analysis of finite loops.
     */
    @Test
    public void testTerminates_finiteLoops() {
        for (Loop loop : new Loop[]{
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 1), new Guard(Relation.LT, 1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 1), new Guard(Relation.LTOET,
0)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 1), new Guard(Relation.GT, 1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(0, 1), new Guard(Relation.GTOET,
1)),
```

```
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(0, 1), new Guard(Relation.NET, 1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(0, 1), new Guard(Relation.ET, 1))
        }) {
            assertTrue(loop.terminates());
        }
    }

    /**
     * A method to test termination analysis of infinite
loops.
     */
    @Test
    public void testTerminates_infiniteLoops() {
        for (Loop loop : new Loop[]{
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(0, 0), new Guard(Relation.LT, 1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.LTOET,
1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.GT, 0)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.GTOET,
1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 2), new Guard(Relation.NET, 2)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.ET, 1))
        }) {
            assertFalse(loop.terminates());
        }
    }
}
```

2.2.3.2 File: GeometricLoopTest.java

```
package cs.loop;

import cs.Guard;
import maths.Relation;
import maths.is.GeometricInfiniteSequence;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 * A class to test methods of GeometricLoop.
 */
public class GeometricLoopTest {

    /**
```

```
     * A method to test termination analysis of finite loops
where R =
     * '{@literal <}'.
     */
    @Test
    public void testTerminates_finiteLoops_LT() {
        for (Loop loop : new Loop[]{
            new GeometricLoop(new GeometricInfiniteSequence(1,
1), new Guard(Relation.LT, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
2), new Guard(Relation.LT, 2)),
            new GeometricLoop(new GeometricInfiniteSequence(-
1, 0), new Guard(Relation.LT, 0)),
            new GeometricLoop(new GeometricInfiniteSequence(-
1, -1), new Guard(Relation.LT, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
-2), new Guard(Relation.LT, 2))
        }) {
            assertTrue(loop.terminates());
        }
    }

    /**
     * A method to test termination analysis of finite loops
where R = '≤'.
     */
    @Test
    public void testTerminates_finiteLoops_LTOET() {
        for (Loop loop : new Loop[]{
            new GeometricLoop(new GeometricInfiniteSequence(1,
1), new Guard(Relation.LTOET, 0)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
2), new Guard(Relation.LTOET, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(-
1, 0), new Guard(Relation.LTOET, -1)),
            new GeometricLoop(new GeometricInfiniteSequence(-
1, -1), new Guard(Relation.LTOET, -1)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
-2), new Guard(Relation.LTOET, 1))
        }) {
            assertTrue(loop.terminates());
        }
    }

    /**
     * A method to test termination analysis of finite loops
where R =
     * '{@literal >}'.
     */
    @Test
    public void testTerminates_finiteLoops_GT() {
```

```
        for (Loop loop : new Loop[]{
            new GeometricLoop(new GeometricInfiniteSequence(1,
1), new Guard(Relation.GT, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(-
1, 2), new Guard(Relation.GT, -2)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
0), new Guard(Relation.GT, 0)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
-1), new Guard(Relation.GT, -1)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
-2), new Guard(Relation.GT, 0))
        }) {
            assertTrue(loop.terminates());
        }
    }

    /**
     * A method to test termination analysis of finite loops
where R = '≥'.
     */
    @Test
    public void testTerminates_finiteLoops_GTOET() {
        for (Loop loop : new Loop[]{
            new GeometricLoop(new GeometricInfiniteSequence(0,
1), new Guard(Relation.GTOET, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(-
1, 2), new Guard(Relation.GTOET, -1)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
0), new Guard(Relation.GTOET, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
-1), new Guard(Relation.GTOET, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
-2), new Guard(Relation.GTOET, 1))
        }) {
            assertTrue(loop.terminates());
        }
    }

    /**
     * A method to test termination analysis of finite loops
where R ∈ {'≠',
     * '='}
     */
    @Test
    public void testTerminates_finiteLoops_NETorET() {
        for (GeometricLoop loop : new GeometricLoop[]{
            new GeometricLoop(new GeometricInfiniteSequence(1,
2), new Guard(Relation.NET, 2)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
1), new Guard(Relation.ET, 1))
        }) {
```

```
            assertTrue(loop.terminates());
        }
    }

    /**
     * A method to test termination analysis of infinite loops
where R =
     * '{@literal <}'.
     */
    @Test
    public void testTerminates_infiniteLoops_LT() {
        for (GeometricLoop loop : new GeometricLoop[]{
            new GeometricLoop(new GeometricInfiniteSequence(0,
2), new Guard(Relation.LT, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
1), new Guard(Relation.LT, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
0), new Guard(Relation.LT, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(-
1, -1), new Guard(Relation.LT, 2)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
-2), new Guard(Relation.LT, 1))
        }) {
            assertFalse(loop.terminates());
        }
    }

    /**
     * A method to test termination analysis of infinite loops
where R = '≤'.
     */
    @Test
    public void testTerminates_infiniteLoops_LTOET() {
        for (GeometricLoop loop : new GeometricLoop[]{
            new GeometricLoop(new GeometricInfiniteSequence(0,
2), new Guard(Relation.LTOET, 0)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
1), new Guard(Relation.LTOET, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
0), new Guard(Relation.LTOET, 0)),
            new GeometricLoop(new GeometricInfiniteSequence(-
1, -1), new Guard(Relation.LTOET, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
-2), new Guard(Relation.LTOET, 0))
        }) {
            assertFalse(loop.terminates());
        }
    }

    /**
```

```
     * A method to test termination analysis of infinite loops
where R =
     * '{@literal >}'.
     */
    @Test
    public void testTerminates_infiniteLoops_GT() {
        for (GeometricLoop loop : new GeometricLoop[]{
            new GeometricLoop(new GeometricInfiniteSequence(0,
2), new Guard(Relation.GT, -1)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
1), new Guard(Relation.GT, 0)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
0), new Guard(Relation.GT, -1)),
            new GeometricLoop(new GeometricInfiniteSequence(-
1, -1), new Guard(Relation.GT, -2)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
-2), new Guard(Relation.GT, -1))
        }) {
            assertFalse(loop.terminates());
        }
    }

    /**
     * A method to test termination analysis of infinite loops
where R = '≥'.
     */
    @Test
    public void testTerminates_infiniteLoops_GTOET() {
        for (GeometricLoop loop : new GeometricLoop[]{
            new GeometricLoop(new GeometricInfiniteSequence(0,
2), new Guard(Relation.GTOET, 0)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
1), new Guard(Relation.GTOET, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
0), new Guard(Relation.GTOET, 0)),
            new GeometricLoop(new GeometricInfiniteSequence(-
1, -1), new Guard(Relation.GTOET, -1)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
-2), new Guard(Relation.GTOET, 0))
        }) {
            assertFalse(loop.terminates());
        }
    }

    /**
     * A method to test termination analysis of infinite loops
where R ∈ {'≠',
     * '='}.
     */
    @Test
    public void testTerminates_infiniteLoops_NETorET() {
```

```
        for (GeometricLoop loop : new GeometricLoop[]{
            new GeometricLoop(new GeometricInfiniteSequence(1,
2), new Guard(Relation.NET, 3)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
1), new Guard(Relation.ET, 1))
        }) {
            assertFalse(loop.terminates());
        }
    }
}
```

2.2.4 Package: cs.loop.cg

2.2.4.1 File: ArithmeticCorrectionGeneratorTest.java

```
package cs.loop.cg;

import cs.Guard;
import cs.loop.ArithmeticLoop;
import cs.loop.Loop;
import maths.Relation;
import maths.is.ArithmeticInfiniteSequence;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 * A class to test methods of ArithmeticCorrectionGenerator.
 */
public class ArithmeticCorrectionGeneratorTest {

    /**
     * A method to test corrections to infinite loops where R
= '{@literal <}'.
     */
    @Test
    public void testGetCorrections_LT() {
        List<Loop> corrections = new ArrayList<>();
        corrections.addAll(Arrays.asList(new Loop[]{
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.LT, 1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(0, 0), new Guard(Relation.LT, 0)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(0, 0), new Guard(Relation.LT, -1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(0, 0), new Guard(Relation.GT, 1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(0, 1), new Guard(Relation.LT, 1))
```

```
        }));
        assertEquals(corrections, (new
ArithmeticCorrectionGenerator(new ArithmeticLoop(new
ArithmeticInfiniteSequence(0, 0), new Guard(Relation.LT,
1)))).getCorrections());
    }

    /**
     * A method to test corrections to infinite loops where R
= '≤'.
     */
    @Test
    public void testGetCorrections_LTOET() {
        List<Loop> corrections = new ArrayList<>();
        corrections.addAll(Arrays.asList(new Loop[]{
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(2, 0), new Guard(Relation.LTOET,
1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.LTOET,
0)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.LTOET, -
1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.LT, 1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 1), new Guard(Relation.LTOET,
1))
        }));
        assertEquals(corrections, (new
ArithmeticCorrectionGenerator(new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.LTOET,
1)))).getCorrections());
    }

    /**
     * A method to test corrections to infinite loops where R
= '{@literal >}'.
     */
    @Test
    public void testGetCorrections_GT() {
        List<Loop> corrections = new ArrayList<>();
        corrections.addAll(Arrays.asList(new Loop[]{
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(0, 0), new Guard(Relation.GT, 0)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(-1, 0), new Guard(Relation.GT, 0)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.GT, 1)),
```

```
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.LT, 0)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, -1), new Guard(Relation.GT, 0))
        }));
        assertEquals(corrections, (new
ArithmeticCorrectionGenerator(new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.GT,
0)))).getCorrections());
    }

    /**
     * A method to test corrections to infinite loops where R
= '≥'.
     */
    @Test
    public void testGetCorrections_GTOET() {
        List<Loop> corrections = new ArrayList<>();
        corrections.addAll(Arrays.asList(new Loop[]{
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(0, 0), new Guard(Relation.GTOET,
1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(-1, 0), new Guard(Relation.GTOET,
1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.GTOET,
2)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.GT, 1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, -1), new Guard(Relation.GTOET,
1))
        }));
        assertEquals(corrections, (new
ArithmeticCorrectionGenerator(new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.GTOET,
1)))).getCorrections());
    }

    /**
     * A method to test corrections to infinite loops where R
= '≠'.
     */
    @Test
    public void testGetCorrections_NET() {
        List<Loop> corrections = new ArrayList<>();
        corrections.addAll(Arrays.asList(new Loop[]{
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(2, 2), new Guard(Relation.NET, 2)),
```

```
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(0, 2), new Guard(Relation.NET, 2)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 2), new Guard(Relation.NET, 3)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 2), new Guard(Relation.NET, 1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 2), new Guard(Relation.ET, 2)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 1), new Guard(Relation.NET, 2))
        }));
        assertEquals(corrections, (new
ArithmeticCorrectionGenerator(new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 2), new Guard(Relation.NET,
2)))).getCorrections());
    }

    /**
     * A method to test corrections to infinite loops where R
= '='.
     */
    @Test
    public void testGetCorrections_ET() {
        List<Loop> corrections = new ArrayList<>();
        corrections.addAll(Arrays.asList(new Loop[]{
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(2, 0), new Guard(Relation.ET, 1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(0, 0), new Guard(Relation.ET, 1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(-1, 0), new Guard(Relation.ET, 1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.ET, 2)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.ET, 0)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.ET, -1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.NET, 1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 1), new Guard(Relation.ET, 1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, -1), new Guard(Relation.ET, 1))
        }));
        assertEquals(corrections, (new
ArithmeticCorrectionGenerator(new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 0), new Guard(Relation.ET,
1)))).getCorrections());
    }
}
```

2.2.4.2 File: GeometricCorrectionGeneratorTest.java

```
package cs.loop.cg;

import cs.Guard;
import cs.loop.GeometricLoop;
import cs.loop.Loop;
import maths.Relation;
import maths.is.GeometricInfiniteSequence;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 * A class to test methods of GeometricCorrectionGenerator.
 */
public class GeometricCorrectionGeneratorTest {

    /**
     * A method to test corrections to loops where R =
'{@literal <}'.
     */
    @Test
    public void testGetCorrections_LT() {
        List<Loop> corrections = new ArrayList<>();
        corrections.addAll(Arrays.asList(new Loop[]{
            new GeometricLoop(new GeometricInfiniteSequence(1,
2), new Guard(Relation.LT, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
2), new Guard(Relation.LT, 0)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
2), new Guard(Relation.LT, -1)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
2), new Guard(Relation.GT, 1))
        }));
        assertEquals(corrections, (new
GeometricCorrectionGenerator(new GeometricLoop(new
GeometricInfiniteSequence(0, 2), new Guard(Relation.LT,
1)))).getCorrections());
    }

    /**
     * A method to test corrections to loops where R = '≤'.
     */
    @Test
    public void testGetCorrections_LTOET() {
        List<Loop> corrections = new ArrayList<>();
        corrections.addAll(Arrays.asList(new Loop[]{
            new GeometricLoop(new GeometricInfiniteSequence(1,
2), new Guard(Relation.LTOET, 0)),
```

```
            new GeometricLoop(new GeometricInfiniteSequence(0,
2), new Guard(Relation.LTOET, -1)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
2), new Guard(Relation.LT, 0))
        }));
        assertEquals(corrections, (new
GeometricCorrectionGenerator(new GeometricLoop(new
GeometricInfiniteSequence(0, 2), new Guard(Relation.LTOET,
0)))).getCorrections());
    }

    /**
     * A method to test corrections to loops where R =
'{@literal >}'.
     */
    @Test
    public void testGetCorrections_GT() {
        List<Loop> corrections = new ArrayList<>();
        corrections.addAll(Arrays.asList(new Loop[]{
            new GeometricLoop(new GeometricInfiniteSequence(-
1, 2), new Guard(Relation.GT, -1)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
2), new Guard(Relation.GT, 0)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
2), new Guard(Relation.GT, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
2), new Guard(Relation.LT, -1))
        }));
        assertEquals(corrections, (new
GeometricCorrectionGenerator(new GeometricLoop(new
GeometricInfiniteSequence(0, 2), new Guard(Relation.GT, -
1)))).getCorrections());
    }

    /**
     * A method to test corrections to loops where R = '≥'.
     */
    @Test
    public void testGetCorrections_GTOET() {
        List<Loop> corrections = new ArrayList<>();
        corrections.addAll(Arrays.asList(new Loop[]{
            new GeometricLoop(new GeometricInfiniteSequence(-
1, 2), new Guard(Relation.GTOET, 0)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
2), new Guard(Relation.GTOET, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
2), new Guard(Relation.GT, 0))
        }));
        assertEquals(corrections, (new
GeometricCorrectionGenerator(new GeometricLoop(new
```

```
GeometricInfiniteSequence(0, 2), new Guard(Relation.GTOET,
0)))).getCorrections());
    }

    /**
     * A method to test corrections to loops where R = '≠'.
     */
    @Test
    public void testGetCorrections_NET() {
        List<Loop> corrections = new ArrayList<>();
        corrections.addAll(Arrays.asList(new Loop[]{
            new GeometricLoop(new GeometricInfiniteSequence(1,
2), new Guard(Relation.NET, 4)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
2), new Guard(Relation.NET, 2)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
2), new Guard(Relation.ET, 3)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
3), new Guard(Relation.NET, 3))
        }));
        assertEquals(corrections, (new
GeometricCorrectionGenerator(new GeometricLoop(new
GeometricInfiniteSequence(1, 2), new Guard(Relation.NET,
3)))).getCorrections());
    }

    /**
     * A method to test corrections to loops where R = '='.
     */
    @Test
    public void testGetCorrections_ET() {
        List<Loop> corrections = new ArrayList<>();
        corrections.addAll(Arrays.asList(new Loop[]{
            new GeometricLoop(new GeometricInfiniteSequence(2,
1), new Guard(Relation.ET, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
1), new Guard(Relation.ET, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(-
1, 1), new Guard(Relation.ET, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
1), new Guard(Relation.ET, 2)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
1), new Guard(Relation.ET, 0)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
1), new Guard(Relation.ET, -1)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
1), new Guard(Relation.NET, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
2), new Guard(Relation.ET, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
0), new Guard(Relation.ET, 1)),
```

```
            new GeometricLoop(new GeometricInfiniteSequence(1,
-1), new Guard(Relation.ET, 1))
        }));
        assertEquals(corrections, (new
GeometricCorrectionGenerator(new GeometricLoop(new
GeometricInfiniteSequence(1, 1), new Guard(Relation.ET,
1)))).getCorrections());
    }
}
```

2.2.5 Package: cs.loop.ic

2.2.5.1 File: ArithmeticIterationCounterTest.java

```
package cs.loop.ic;

import cs.Guard;
import cs.loop.ArithmeticLoop;
import maths.Relation;
import maths.is.ArithmeticInfiniteSequence;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 * A class to test methods of ArithmeticIterationCounter.
 */
public class ArithmeticIterationCounterTest {

    /**
     * A method to test counting iterations in finite
arithmetic loops.
     */
    @Test
    public void testGetCount() {
        for (ArithmeticLoop al : new ArithmeticLoop[]{
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 1), new Guard(Relation.LT, 1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 1), new Guard(Relation.LTOET,
0)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(1, 1), new Guard(Relation.GT, 1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(0, 1), new Guard(Relation.GTOET,
1)),
            new ArithmeticLoop(new
ArithmeticInfiniteSequence(0, 1), new Guard(Relation.ET, 1))
        }) {
            assertEquals(0, (new
ArithmeticIterationCounter(al)).getCount());
        }
```

```
        assertEquals(1, (new ArithmeticIterationCounter(new
ArithmeticLoop(new ArithmeticInfiniteSequence(0, 1), new
Guard(Relation.NET, 1)))).getCount());
    }
}
```

2.2.5.2 File: GeometricIterationCounterTest.java

```
package cs.loop.ic;

import cs.Guard;
import cs.loop.GeometricLoop;
import maths.Relation;
import maths.is.GeometricInfiniteSequence;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 * A class to test methods of GeometricIterationCounter.
 */
public class GeometricIterationCounterTest {

    /**
     * A method to test counting iterations in finite
geometric loops.
     */
    @Test
    public void testGetCount() {
        for (GeometricLoop gl : new GeometricLoop[]{
            new GeometricLoop(new GeometricInfiniteSequence(1,
1), new Guard(Relation.LT, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
1), new Guard(Relation.LTOET, 0)),
            new GeometricLoop(new GeometricInfiniteSequence(1,
1), new Guard(Relation.GT, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
1), new Guard(Relation.GTOET, 1)),
            new GeometricLoop(new GeometricInfiniteSequence(0,
1), new Guard(Relation.ET, 1))
        }) {
            assertEquals(0, (new
GeometricIterationCounter(gl)).getCount());
        }

        assertEquals(1, new GeometricIterationCounter(new
GeometricLoop(new GeometricInfiniteSequence(1, 2), new
Guard(Relation.NET, 2))).getCount());
    }
}
```

2.2.6 Package: cs.rmc

2.2.6.1 File: ArithmeticRecursiveMethodCallTest.java

```
package cs.rmc;

import cs.Guard;
import maths.Relation;
import static org.junit.Assert.*;
import org.junit.Test;
import maths.is.ArithmeticInfiniteSequence;
import cs.loop.ArithmeticLoop;

/**
 * A class to test methods of ArithmeticRecursiveMethodCall.
 */
public class ArithmeticRecursiveMethodCallTest {

    /**
     * A method to test converting an arithmetic recursive
method call into a
     * loop.
     */
    @Test
    public void testToLoop() {
        ArithmeticInfiniteSequence ais = new
ArithmeticInfiniteSequence(3, -1);
        int b = 0;
        assertEquals(new ArithmeticLoop(ais, new
Guard(Relation.NET, b)), (new
ArithmeticRecursiveMethodCall(ais, new Guard(Relation.ET,
b))).toLoop());
    }
}
```

2.2.6.2 File: GeometricRecursiveMethodCallTest.java

```
package cs.rmc;

import cs.Guard;
import cs.loop.GeometricLoop;
import maths.Relation;
import maths.is.GeometricInfiniteSequence;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 * A class to test methods of GeometricRecursiveMethodCall.
 */
public class GeometricRecursiveMethodCallTest {

    /**
```

```
     * A method to test converting a geometric recursive
method call into a
     * loop.
     */
    @Test
    public void testToLoop() {
        GeometricInfiniteSequence gis = new
GeometricInfiniteSequence(1, 2);
        int b = 4;
        assertEquals(new GeometricLoop(gis, new
Guard(Relation.NET, b)), (new
GeometricRecursiveMethodCall(gis, new Guard(Relation.ET,
b))).toLoop());
    }
}
```

2.2.7 Package: maths.ae

2.2.7.1 File: NodeTest.java

```
package maths.ae;

import org.junit.Test;
import static org.junit.Assert.*;

/**
 * A class to test methods of Node.
 */
public class NodeTest {

    /**
     * A method to test generating an arithmetic expression
where operations are
     * performed in descending order of precedence.
     */
    @Test
    public void testToString_descendingPrecedenceOps() {
        Node exp = new Node(Operation.EXPONENTIATE, new
Node("a"), new Node("b")),
                quot = new Node(Operation.DIVIDE, exp, new
Node("c")),
                prod = new Node(Operation.MULTIPLY, quot, new
Node("d")),
                sum = new Node(Operation.ADD, prod, new
Node("e")),
                diff = new Node(Operation.SUBTRACT, sum, new
Node("f"));
        assertEquals("a ^ b / c * d + e - f",
diff.toString());
    }
```

```
    /**
     * A method to test generating an arithmetic expression
where operations are
     * performed in ascending order of precedence.
     */
    @Test
    public void testToString_ascendingPrecedenceOps() {
        Node diff = new Node(Operation.SUBTRACT, new
Node("a"), new Node("b")),
                sum = new Node(Operation.ADD, diff, new
Node("c")),
                prod = new Node(Operation.MULTIPLY, sum, new
Node("d")),
                quot = new Node(Operation.DIVIDE, prod, new
Node("e")),
                exp = new Node(Operation.EXPONENTIATE, quot,
new Node("f"));
        assertEquals("((((a - b) + c) * d) / e) ^ f",
exp.toString());
    }
}
```

2.2.8 Package: maths.is

2.2.8.1 File: ArithmeticInfiniteSequenceTest.java

```
package maths.is;

import org.junit.Test;
import static org.junit.Assert.*;

/**
 * A class to test methods of ArithmeticInfiniteSequence.
 */
public class ArithmeticInfiniteSequenceTest {

    /**
     * A method to test checking whether an arithmetic
infinite sequence
     * contains a term, with cases that do.
     */
    @Test
    public void testContains_AISsWithTerm() {
        assertFalse(new ArithmeticInfiniteSequence(0,
0).contains(1));
        assertFalse(new ArithmeticInfiniteSequence(1,
1).contains(0));
    }

    /**
```

```
     * A method to test checking whether an arithmetic
infinite sequence
     * contains a term, with cases that do not.
     */
    @Test
    public void testContains_AISsWithoutTerm() {
        for (InfiniteSequence is : new InfiniteSequence[]{
            new ArithmeticInfiniteSequence(1, 1),
            new ArithmeticInfiniteSequence(0, 1)
        }) {
            assertTrue(is.contains(1));
        }
    }
}
```

2.2.8.2 File: GeometricInfiniteSequenceTest.java

```
package maths.is;

import org.junit.Test;
import static org.junit.Assert.*;

/**
 * A class to test methods of GeometricInfiniteSequence.
 */
public class GeometricInfiniteSequenceTest {

    /**
     * A method to test checking whether a geometric infinite
sequence contains
     * a term, with cases that do.
     */
    @Test
    public void testContains_AISsWithTerm() {
        for (InfiniteSequence is : new InfiniteSequence[]{
            new GeometricInfiniteSequence(1, 1),
            new GeometricInfiniteSequence(-1, -1)
        }) {
            assertTrue(is.contains(1));
        }

        assertTrue(new GeometricInfiniteSequence(1,
0).contains(0));
        assertTrue(new GeometricInfiniteSequence(1, -
2).contains(-2));
    }

    /**
     * A method to test checking whether a geometric infinite
sequence contains
     * a term, with cases that do not.
     */
```

```
    @Test
    public void testContains_AISsWithoutTerm() {
        for (InfiniteSequence is : new InfiniteSequence[]{
            new GeometricInfiniteSequence(-1, 1),
            new GeometricInfiniteSequence(0, 1),
            new GeometricInfiniteSequence(0, 0)
        }) {
            assertFalse(is.contains(1));
        }

        assertFalse(new GeometricInfiniteSequence(1, -
2).contains(2));
        assertFalse(new GeometricInfiniteSequence(2,
2).contains(3));
    }
}
```

Reference list

Oracle n. d., '*How to Use Tables'*, viewed 30 April 2019, https://docs.oracle.com/javase/tutorial/uiswing/components/table.html#data