Assignment 3

Instructor: Musard Balliu

October, 2016

In this assignment, you will implement an object-oriented application to determine the truth value of *boolean circuits*. A boolean circuit is a logical formula consisting of boolean variables $a, b, x, y, \cdots$, negation (NOT, written as !), conjunction (AND, written as $\wedge$) and disjunction (OR, written as $\vee$). For example, $(a \wedge b) \vee (c \vee !(b))$ is a boolean circuit on variables a, b and c. The *truth value* of a boolean circuit is either $true$ or $false$, and it can be determined whenever the truth values of all variables of the circuit are fixed. For example, if $a = true, b = false$ and $c = true$, the truth value of the boolean circuit above is $true$. An assignment of truth values to the boolean variables of a circuit is called a *truth assignment* for that circuit.

Two circuits are *equivalent* iff they have the same truth value for all possible truth assignments of their variables. For example, the circuit $a \wedge a$ is equivalent to the circuit a, and the circuit $!(a \vee b)$ is equivalent to the circuit $!(a) \wedge !(b)$. The circuit $a \wedge b$ is not equivalent to the circuit $a \vee b$, since, for instance, if $a = false$ and $b = true$, the truth value of the first circuit is $false$, and the truth value of the second circuit is $true$.

The *free variables* of a circuit are the variables used in that circuit (without repetitions). For example, the free variables of the circuit $(x \wedge y) \vee !(x)$ are x and y.

The task

In this assignment, you are given two classes, Assignment.java and Circuit.java, and they are already implemented for you. In particular, the method $equals()$, implemented in Circuit.java, allows to determine whether or not two circuits are equivalent.

The goal of this assignment is to complete the implementation of subclasses Variable.java, Not.java, And.java e Or.java of the superclass Circuit.java. Concretely, you have to complete the implementation of the methods `isTrueIn()`, `freeVariables()` and `toString()` for each subclass (and a few other methods, constructors and instance variables). The methods have the following specification:

- `isTrueIn(Assignment a)` determines whether a circuit is true for a given assignment `a`.
- `freeVariables()` returns an array containing the free variables of a circuit, in any order and without repetitions.
- `toString()` returns a string representation of the circuit.

You also have to implement a main class, CircuitMain.java, that creates two circuits $!(a \wedge b)$ and $!(a) \vee !(b)$ and checks whether or not they are equivalent.

Finally, you have to create n random circuits of maximal depth 5, calculate their truth value, and print to the console both the (string representation of) circuits and their truth values, for a given positive integer n provided as input by the user.

In order to get a VG, you must implement a *graphical user interface* (GUI) that handles the features in class CircuitMain.java.

The specification of each method is provided as commented java files, included in the following sections and available for download from
http://www.cse.chalmers.se/ musard/teaching/programming2016/dit948-2016/Assign3_2016.zip

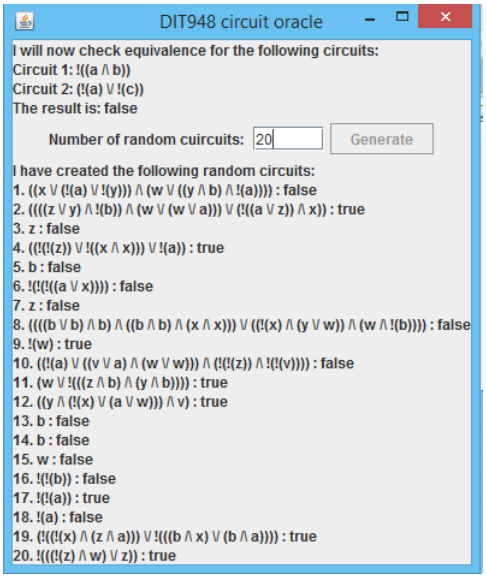
The interface, example session

The set of all variables used in a random circuit is "v","w","x","y","z","a","b", of which the variables "v","y","a" are true in the assignment (and, consequently, the remaining variables are false). The class `CircuitMain` has a `main` method which, when executed, should lead to an user session of the following kind:

```
Welcome to the DIT948 circuit oracle

I will now check equivalence for the following circuits:
Circuit 1: !((a /\ b))
Circuit 2: (!(a) \/ !(c))
```

```
The result is: false

How many random circuits should I generate? 5

I have created the following random circuits:
1. (((b \/ (y \/ z)) \/ b) /\ z) : false
2. (v /\ (((a /\ v) /\ (b \/ w)) \/ ((z \/ a) /\ !(y)))) : false
3. !((!(!(v)) /\ z)) : true
4. y : true
5. (v \/ !(!((z /\ v)))) : true
```

Similarly, the GUI implementation may result in an interactive session as below.



The classes

Class `Variable.java`

```
/**
 * This is a subclass of Circuit implementing Variable in a Circuit
 *
 */

public class Variable extends Circuit {
```

```
// Private instance variable,
// name, a String representing a variable

// code here

// Constructor with parameters

public Variable(String name) {
// code here
}

/**
 * String representation of a variable of a circuit
 */

public String toString() {
// code here
}

/**
 * Returns the name of the variable
 */

public String getName() {
// code here
}

/**
 * Returns true if the variable is true in the assignment
 */

public boolean isTrueIn(Assignment assignment) {
// code here
}

/**
 * Returns the array of free Variables in this Variable.
 */

public Variable[] freeVariables() {
```

```
// code here
}
}
```

Class `Not.java`

```
/**
 * This is a subclass of Circuit implementing the negation of a Circuit
 */

public class Not extends Circuit {

// Private instance variable,
// negated, a Circuit to be negated

// code here

// Constructor with parameters

public Not(Circuit negated) {
// code here
}

/**
 * String representation of the negation of a circuit
 */

public String toString() {
// Code here
}

/**
 * Returns true if the negation of the circuit is true in the assignment
 */

public boolean isTrueIn(Assignment assignment) {
// code here
}

/**
```

```
 * Returns the array of free Variables in the negated circuit. The order of
 * variables is not important, however, a variable should appear exactly
 * once in the array (no repetitions)
 */

public Variable[] freeVariables() {
// code here
}
}
```

Class `And.java`

```
/**
 * This is a subclass of Circuit implementing the conjunction of a left Circuit
 * and a right Circuit
 */

public class And extends Circuit {

// Private instance variables,
// left, the left Circuit
// right, the right Circuit

// code here

// Constructor with parameters

public And(Circuit left, Circuit right) {
// code here
}

/**
 * String representation of the conjunction of a left circuit and a right
 * circuit
 */

public String toString() {
// code here
}
```

```
/**
 * Returns true if the conjunction of the left circuit and the right circuit
 * is true in the assignment
 */

public boolean isTrueIn(Assignment assignment) {
// code here
}

/**
 * Returns the array of free Variables in the conjunction of a left circuit
 * and a right circuit. The order of variables is not important, however, a
 * variable should appear exactly once in the array (no repetitions)
 */

public Variable[] freeVariables() {
// code here
}
}
```

Class `Or.java`

```
/**
 * This is a subclass of Circuit implementing the disjunction of a left Circuit
 * and a right Circuit
 *
 */

public class Or extends Circuit {

// Private instance variables,
// left, the left Circuit
// right, the right Circuit

// code here

// Constructor with parameters

public Or(Circuit left, Circuit right) {
// code here
```

```
}

/**
 * String representation of the disjunction of a left circuit and a right
 * circuit
 */

public String toString() {
// code here
}

/**
 * Returns true if the disjunction of the left circuit and the right circuit
 * is true in the assignment
 */

public boolean isTrueIn(Assignment assignment) {
// code here
}

/**
 * Returns the array of free Variables in the disjunction of a left circuit
 * and a right circuit. The order of variables is not important, however, a
 * variable should appear exactly once in the array (no repetitions)
 */

public Variable[] freeVariables() {
// code here
}

}
```

Remarks and Notation

Your are NOT allowed to introduce additional (static or instance) variables in the class templates. However, you can implement additional methods, if needed.

Classes Assignment.java and Circuit.java can be used as APIs and you are not allowed to change their implementation.

The GUI provided earlier is just an example. Feel free to implement your own fancy version, so long as it contains the basic tasks as required by the assignment.

Grading

- To get a G, you do not need to implement a graphical user interface.
- To get a VG, it is necessary implement all tasks in the assignment.

  Note that these requirements are necessary, but not sufficient, to get the respective grades (see Administrative matters).

Administrative matters

Strive for readable code with appropriate comments! While the ultimate test of a program is that it does what it is supposed to do, **we should be able to read the program and understand it**.

The assignment is to be completed in groups of two students. (Groups of different size should first be agreed with the course instructor.)

Solutions must be uploaded to the GUL system **by 23:55 on October 19**.

Please note: **The submission must be made via GUL. It's no good sending it to me via email, either before or after the deadline!**

Only Java source files should be uploaded, no class files. Your Java files should be put in a zip-archive with the following name:

```
assign3_author1_author2.zip
```

where author1, author2 are the surnames (family names) of the group members. You must also supply a README-file (README.txt) containing the names of the authors and their social security numbers, together with a brief statement about each author's contribution. For example, "author1 has been responsible for user input and author2 for the program logic". A statement such as "All authors have contributed equally to all aspects of the program" is not acceptable.

Note that **each author must submit individually via the GUL** (even though it is the same program!). Only students who submit via the GUL can be graded. All comments etc. will be posted on the course web page.

Once the deadline has passed, each group **must explain solutions to the TAs during the first Thursday supervision session**. Exceptions are to be agreed with the instructor and the TAs. Group members are expected to know and explain all parts of the code despite their statement of contribution.