# Assignment 8

Due: Friday, 7.07.2017, 15:59 via Git

For help, contact alp-staff@lists.iai.uni-bonn.de (staff only) or alp-course@lists.iai.uni-bonn.de (staff and participants).

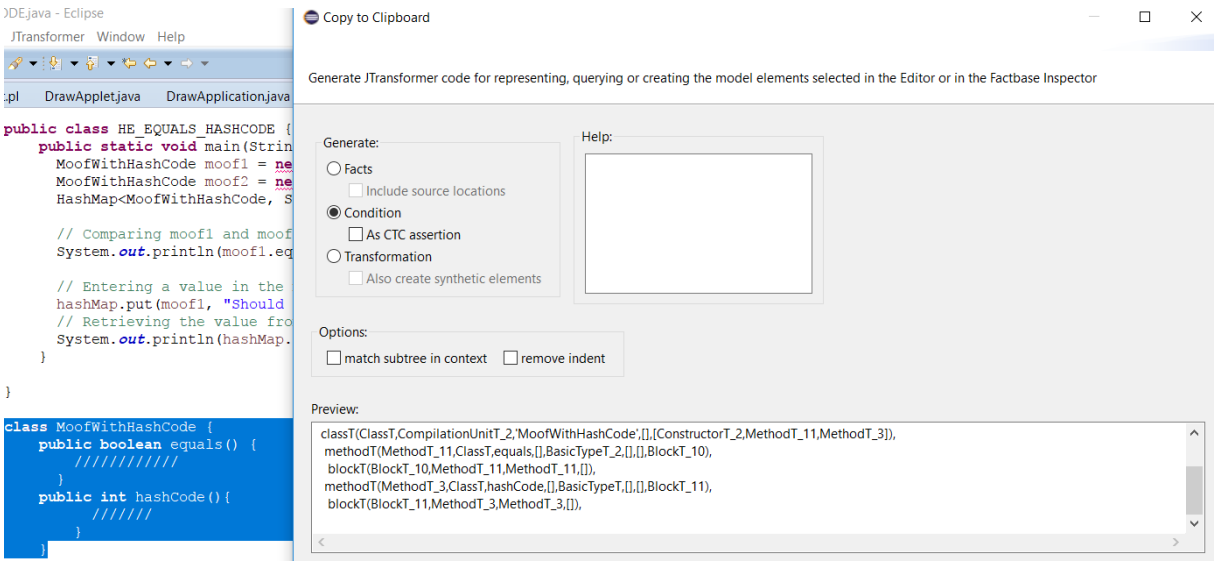
Start working on the exercises early enough so that you can contact your tutor in time if you have problems. Don't expect your tutor to be available at midnight or during weekends!

Submit your implemented predicates as a file named "assignment08/solutions.pl" in the Git repository of your group. Add to each task not just the code that you implemented but also the console output of a session in which you test that **each** solution works for the provided input data and some queries that represent sensible test cases. If no input data is provided in the text of the task, create some sensible input data. If input data is represented as facts, include them into the "solutions.pl" file and add suitable comments.

## **Task 1.** *Detect broken contracts* (10 Points)

Implement a predicate class_implements_equals_but_not_hashcode(Class, EqualsMethod) that detects the problem described under https://sewiki.iai.uni-bonn.de/teaching/labs/mdse/2013/bug_descriptions/jt-bug-he_equals_no_hashcode.

Use the sample code provided on that page to test your predicate.




```
missing_hashcode(Class, EqualsMethodId):-
        classT(Class,_,_,_,_),
        methodT(EqualsMethodId,Class,equals,_,_,_,_,_),
        not( methodT(_,Class,hashCode,_,_,_,_,_)).
```

**Tip**: Write Java code that you want to detect (or to detect that it is *not* there), then use the "Copy to clipboard" feature of JTransformer, to get Prolog code snippets that match that code. Remove parts that are too specific, rename variables so that they make sense in your context and use the code snippet in your problem detector predicate.

## **Task 2.** *Integrate your predicate into JTransformer* (10 Points)

Integrate your predicate into the graphical user interface of JTransformer as described in https://sewiki.iai.uni-bonn.de/research/jtransformer/tutorial/analysis_cc. Submit the additionally implemented predicates and a screenshot that shows that your analysis is offered in the JT Control Center and has produced results.

```
% The the 'multifile' declaration says that the predicate
% analysis_definition/5 logically belongs to the analysis_api
% module but clauses can be defined in many files:
:- multifile(analysis_api:analysis_definition/5).

% The clause defining the detector:
analysis_api:analysis_definition(
     'Missing hashCode()',              % Predicate Name
     onSave,                            % Trigger
     error,                             % Severity
     'Correctness',                     % Category
     'Class overrides equals() but not hashCode()'
                                        % Description
).

:- multifile(analysis_api:analysis_result/3). % (Name, Group, Result)

analysis_api:analysis_result('Missing hashCode()', _, Result) :-
    % Call the Missing Hash code detector that you implemented:
    missing_hashcode(Class, EqualsMethodId),
    classT(Class,_,ClassName,_,_),

    % Create a description
    with_output_to(Description,
       format('Class ~a overrides equals() but not hashCode()',
       [ClassName])
    ),

    % Alternative way to create the description:
    % atom_concat('Class ', ClassName, Temp),
    % atom_concat(Temp, ' overrides equals() but not hashCode()',
Description),

    % Yet another way to do the same, more elegant than a cascade of
    % calls to atom_concat/3:
    % atomic_list_concat(   % <-- SWI-Prolog built-in, see online manual
    %     ['Class ', ClassName, ' overrides equals() but not hashCode()'],
    %     Description
    % ),
```

```
    % ← You get points for any of the above solutions and any other that
    % works properly.

    % Wrap everything into a result term for the GUI:
    make_result_term(missing_hashcode(EqualsMethodId), Description,
Result).
```

**Task 3.** *Statement order in a block* (2 Points)

In Java, statements are typically contained in blocks limited by a pair of opening and closing curly braces. Such a block is represented in JTransformer by a blockT/4 fact (see http://sewiki.iai.uni-bonn.de/research/jtransformer/api/java/pefs/4.1/blockt).

Write a predicate before_in_block(+*StatementId1*, +*StatementId2*, +*BlockId*) that succeeds, if *StatementId1* comes before *StatementId2* in the statement list of the block with ID BlockID.

Hand in your program, the console output of a successful run, and the Java source code that you used for your test.

**Tip**: Use the predefined predicate nth1(?Index, ?List, ?Elem) – see the SWI-Prolog manual.

```
%% before_in_block(?StatementId1, ?StatementId2, ?BlockId)
%
% Succeeds if StatementId1 comes before StatementId2 in the
% statement list of the block with ID BlockID.

before_in_block(StatementId1, StatementId2, BlockId) :-
    blockT(BlockId, _, _, Elements),
    nth1(Index1, Elements, StatementId1),
    nth1(Index2, Elements, StatementId2),
    Index1 < Index2.
```