Programming Assignment 3: Clojure

Beatrice Åkerblom
beatrice@dsv.su.se

November 21, 2017

1 Introduction

This assignment is the third out of four assignments. The assignment is due on 2017-12-15 (23.55) and submission should be done through the hand-in facilities on the course's iLearn2 page. The assignment is conducted in groups of two (2) students (exceptions are *only* provided by Beatrice).

If in doubt, please use the iLearn2 discussion forum.

2 The Assignments

Your assignment is to implement two Clojure macros and discuss the use of macros.

2.1 Safe Macro

The first part of this assignment is to write a macro called `safe` that is intended to function similarly to `try` in Java 7, and `using` in C#. It should be possible to have a binding form (i.e. `[variable value]`) that binds to an instance of a closable.

Example, Java 7 `try`

```
try (Socket s = new Socket()) {
  s.accept();
} catch(Exception e) {}
```

In the code above the socket 's' is automatically closed, just as if you had explicitly written a finally clause

```
finally {
 if (s != null) s.close();
}
```

You should define a macro 'safe' that takes two arguments. The first argument should be a vector with two elements, one variable and one value (corresponding to 's' and 'new Socket()' in the example above). The second argument should be a form (expression) that should be evaluated.

If an exception is thrown inside the form the exception should be returned. Otherwise the evaluated value of the form should be returned. If a binding is provided it should be possible to use the bound variable inside the form. After execution of the form any variables that are bound shall be closed (e.g. `(let [s (Socket.)] (. s close))`) Note that closeable classes in Java implements the `Closeable`–interface (hint: use type hints). The return value of the macro is either the return value of the executed form or an exception.

The Clojure macro shall function in the following way:

```
user>  (import java.io.FileReader java.io.File)
java.io.File
...
user> (def v (safe (/ 1 0)))
user> v
#<ArithmeticException java.lang.ArithmeticException: Divide by zero>
user> (def v (safe (/ 10 2)))
user> v
5
user> (def v (safe [s (FileReader. (File. "file.txt"))] (.read s)))
user> v
105 ; first byte of file file.txt
user> (def v (safe [s (FileReader. (File. "missing-file"))] (. s read)))
user> v
#<FileNotFoundException java.io.FileNotFoundException:
missing-file (No such file or directory)>
```

2.2 SQL-like Macro

The second macro should implement an SQL-like syntax for searching in lists of maps, as illustrated by the following example:

```
=> (def persons '({:id 1 :name "olle"} {:id 2 :name "anna"} {:id 3 :name
"isak"} {:id 4 :name "beatrice"}))
...
=> (select [:id :name] from persons where [:id > 2] orderby :name)
({:id 4 :name "beatrice"} {:id 3 :name "isak"})
```

The query format should be:

```
(select [columns]
    from #{table}
    where [:column op value]
    orderby :column)
```

where the condition operator `'op'` is one of: =, <, >, <>

2.3 Reflections

Discuss how these two macros can be implemented as functions, Why?, Why not? How? (approximately 500 words).

3 Grading

In order to receive the grade E you have to implement the safe macro and write the reflection on macros. To receive a grade better than C, you also have to implement the SQL macro.

Grading programming is more an art than a science. In the general case, it is extremely difficult to impossible to say that one program is better than the other. For the obvious reasons, it is impossible to cover the grading criteria completely. In any case, the points below are not totally black/white. This is why you should also should reflect on you implementation.

For the programming assignment for block 4 the following table will be used when grading the assignments.

Grade	Safe macro	SQL macro	Reflections
A	Good	Good	Good
B	Good	Good	OK
C	Good	OK	OK
D	OK	Missing/OK-	OK-
E	OK-	Missing/OK-	OK-
Fx	Missing/OK-	Missing/OK-	Missing/OK-
F	Missing/Incorrect	Missing	Missing/Incorrect

Table 1: Grading scheme

To get a Good, your program/answer must:

1. Completely meet the specification,e.g., produce correct output from example inputs (not too little or too much), or the like;
2. Be readable without overly commenting, be correctly indented, and use reasonable names (for variables, method, classes etc.);
3. Be well designed for the problem at hand

For every point above that is not satisfied, the program/answer takes one step down the grade ladder from Good down to Missing/Fail.