Prof. Dr. Oscar Nierstrasz
Pascal Gadient, Pooja Rani

Solution
Assignment 02 — 23.09.2020 – v1.0
Smalltalk: A Reflective Language

Please submit this exercise by mail to pascal.gadient@inf.unibe.ch before 30 September 2020, 10:15am.

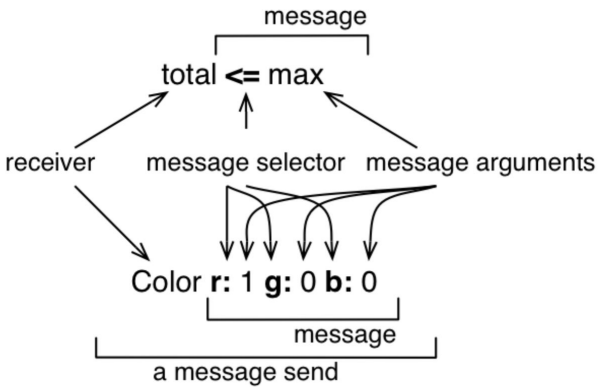
For this assignment you have to download and install Glamorous Toolkit (GT) from here.

Exercise 1 - Nature of Smalltalk and GT (5 pts)

a) What do you have to consider when you use a live system for software development? In other words, name one major threat that cannot arise in traditional software development, *e.g.* with Java, C++, etc.? **Answer:**

Everything can be changed including system objects and language features. As a result, testing of random code might alter the execution environment and introduce future problems. For instance, your GT system will immediately crash if you randomly modify the code of kernel messages and classes, e.g. *you delete the* `Object` *class.*

b) What is a message in Smalltalk? **Answer:**



All communication or interaction among objects is done by sending messages. A message is composed of the message selector and the optional message arguments. A message is sent to a receiver. The combination of a message and its receiver is called a message send.

c) What is a block in Smalltalk? **Answer:**

A block can be thought of as a lambda-expression defining an anonymous function, or as a function object. Square brackets `[ ]` *define a block, also known as a block closure or a lexical closure, which is a first-class object representing a function.*
Blocks may take one or more arguments (`[:i :j :k | ... ]`*) and can have local variables.*

d) How do Smalltalk, Pharo, and GT relate to each other? **Answer:**

Smalltalk is the programming language used in Pharo and GT. Although the language stands on its own, it has many ties to its implementations in Pharo and GT that define the available instruction set (similar to libraries in other languages). Pharo is (mostly) implemented in Smalltalk. GT is a sophisticated framework written in Smalltalk on top of Pharo that uses a headless VM. GT tries to improve productivity and leverages many new features, e.g., *native window support and interactive notebooks.*

e) What are counterparts of GT tools (Playground, Coder, Git, Monitor, ExamplesExplorer, Transcript, Morphic World, Spotter) in your favorite development environment? **Answer:**

- *Playground: a sophisticated (text-based) shell that has various inspection capabilities.*
- *Coder: an integrated development environment (IDE) to edit existing packages, classes, and messages.*
- *Git: a Git client.*
- *Monitor: similar to a resource viewer (Linux, macOS) or a task-manager (Windows).*
- *ExamplesExplorer: (offline) help resources,* e.g. *documentation and demos.*
- *Transcript: debug or console output window.*
- *Morphic World: the base Pharo windowing system and IDE, enables low-level access to Smalltalk classes and instances.*
- *Spotter: a search that inspects names and file content.*

Exercise 2 - Object inspection (4 pts)

a) Explain the difference between a `String` and a `Symbol` object in Smalltalk. Why is this differentiation important?
Hint: The execution of the code below will reveal differences.

```
('HeySmalltalker') == 'HeySmalltalker'.
('HeySmalltalker') asSymbol == #HeySmalltalker.
('Hey','Smalltalker') == 'HeySmalltalker'.
('Hey','Smalltalker') asSymbol == #HeySmalltalker.
```

Answer:

- *Symbols are immutable and unique. Strings are mutable and not unique. Consequently, multiple String objects with the value "Hello" can coexist, but only one Symbol object #Hello can be initialized during run time. Since Symbols are immutable, they should never be used when their value has to change over time. The benefit of using Symbols is the performance gain due to less expensive value comparisons.*

- `('HeySmalltalker') == 'HeySmalltalker'.`
  Comparison of two Strings, expected to return true (note that the outcome depends on the VM implementation).

  `('HeySmalltalker') asSymbol == #HeySmalltalker.`
  Comparison of two Symbols, expected to return true.

  `('Hey','Smalltalker') == 'HeySmalltalker'.`
  Comparison of two (concatenated) Strings, expected to return false (note that the outcome depends on the VM implementation).

  `('Hey','Smalltalker') asSymbol == #HeySmalltalker.`
  Comparison of two (concatenated) Symbols, expected to return true.

b) Write the equivalent of the following piece of code in Smalltalk as a block, and execute it with the values <38, 44>, <65, 48>, and <48, 48>.

Hint: Use the transcript tool (one line example below) as replacement for the `print` *method.*
`Transcript show: 'your output here'`

```
int scoreOfPlayerA, scoreOfPlayerB;
if(scoreOfPlayerA > scoreOfPlayerB)
    print "Player A Won"
else if(scoreOfPlayerA < scoreOfPlayerB)
    print "Player B Won";
else
    print "Match is declared as draw";
```

Answer:

Please download the solution here.

c) How many classes in GT implement the message `includes: anObject`?
How many messages in GT use that particular message?
Note: For this task, you are supposed to use a fresh and unaltered copy of GT with no changes of yours. **Answer:**

*In order to find the relevant classes and messages, you can use the Spotter tool, use CTRL+M/CMD+M (implementors) and CTRL+N/CMD+N (references), or you can perform this task programmatically (with code). There might exist minor differences depending on the version of the GT image you use, and the included elements in your search. We found 60 classes that implement the message, and more than four thousand messages that use it (*e.g., *4 531 in the current Apple macOS GT build).*

d) Which message in GT can be sent to a class to find all its subclasses? **Answer:**

```
subclasses
```

Exercise 3 - CallGraph (1 pt)

Find the top ten most frequently invoked methods in the provided CallGraph representation.

In order to perform this exercise, you must (i) execute the statement below to retrieve the "CallGraph Demo" in GT, and (ii) you have to download the `Calls.txt` file from here and store it in the same folder as the GT image file.

```
Metacello new baseline:  'SMAForGt';
repository:  'github://onierstrasz/sma-examples/src'; load.
```

Hint 1: You can find some examples for accessing the CallGraph in the "CallGraph Demo".

Hint 2: Don't forget to submit your code snippet and your results.

Answer:

```
cg := CallGraph fromFile:  'Calls.txt'.
result := (cg methods sorted:  [ :a :b | a calls size >= b calls size]).
```

Result:
org.clapper.util.misc.LRUMap$LRULinkedList.addToHead
org.clapper.util.misc.FileHashMap.checkValidity
org.clapper.util.misc.MultiIterator.checkIterator
org.clapper.util.misc.LRUMap.doPut
org.clapper.util.misc.LRUMap.clearTo
org.clapper.util.misc.LRUMap$LRULinkedListEntry.setKeyValue
org.clapper.util.text.AbstractVariableSubstituter.legalVariableCharacter
org.clapper.util.misc.MultiValueMap.keySet
org.clapper.util.misc.MultiValueMap.put
org.clapper.util.misc.FileHashMap$ValuesFile.getFile