Assignment 04 — 07.10.2020– v1.0b
Smalltalk: Reflection

Please submit this assignment by email to pascal.gadient@inf.unibe.ch before 14. October 2020, 10:15am.

Exercise 1 - Hierarchy traversal (1 pt)

Write a method that finds the class with the longest inheritance chain among all Smalltalk classes in the GT programming environment.

NB: To access all classes of Smalltalk, you can use `SystemNavigation default allClasses`.

Exercise 2 - Method overrides (2 pts)

Write a method to find all methods that override an abstract method in GT.

Exercise 3 - Query methods (2 pts)

Write a method that finds all classes with at least one query method in GT.

NB: Query methods test a property of an object. Such methods are prefixed with `is`, `was` *or* `will`.

Exercise 4 - Root methods (2 pts + 2 pts BONUS)

i) Find all root methods in GT.

   NB: A "root method" is a method whose selector has been implemented in a class, such that the superclasses of that class do not understand it.

ii) (BONUS) Find all duck-typed methods in GT.

   NB: Duck-typed methods have the same selector but are not related by inheritance. That is, after finding all root methods, find those with the same selector.

Please continue reading on the next page.

Exercise 5 - Dynamic coding (3 pts)

This exercise carries on with exercise 3 of the second assignment. As stated before, you have to download the `CallGraph` code from Github, and you must store the `Calls.txt` file in the same folder as the GT image file.

Your task is to redefine the method `doesNotUnderstand: aMessage` in the provided class `Call`. The redefined method should dynamically create an instance variable and a method that returns the number of arguments. In order to achieve that, you are supposed to follow these three steps:

Step 1: Within the method, add dynamically the instance variable `numberOfArguments` to the class `Call` if it does not already exist.

Step 2: Within the method, add dynamically the method below to the class `Call`. Since you are adding that method during run time, you must compile it from a String representation.

```
numberOfArguments
numberOfArguments := args size.
^ numberOfArguments.
```

Step 3: So far, the initial execution does nothing but enable the `numberOfArguments` method. Hence, we have to resend the initial message to `self`.

You can test your implementation by executing the following code:

```
(CallGraph fromFile: 'Calls.txt') calls
  collect: [ :each | each numberOfArguments]
```

After you successfully implemented the `doesNotUnderstand` method, the statement will print the number of arguments for every call in the call graph (without raising a `doesNotUnderstand` error).