Lab session 3 – Smalltalk & Pharo

Unit	Programming languages: principles and design (6G6Z1110) Programming languages – SE frameworks (6G6Z1115)
Lecturer	Rob Frampton
Week	3
Portfolio element	This lab session provides help for the "Smalltalk/Pharo" element.

Description

This lab is a hands-on tutorial on Pharo – a recently developed Smalltalk fork. The main aim is to provide you with introductory material on using Pharo and Smalltalk. Some of the lab is based on extracts taken from the "*Learning Object-Oriented Programming, Design and TDD with Pharo*", a book which is publicly available on the website pharo.org (and on Moodle as well).

By the end of this lab session, you should:

1. have learnt how to use Pharo,
2. have practiced Smalltalk syntax, and
3. be capable to implement the "Smalltalk/Pharo" portfolio element.

Exercises

Exercise 1. Implement a simple counter on Pharo[1]

Note: there is an additional PDF on Moodle which talks you through this exercise should you need it

With this exercise, we will implement a simple counter on Pharo. We will create a class Counter to record the value and several methods for assigning an initial value, incrementing and decrementing it. Figure 1 shows the diagram for *Counter*, which includes an instance variable '*count*' to store the value; and four methods: '*count*', '*count:*', '*increment*', and '*decrement*'; to access count (accessor), set count (setter), increment count, and decrement count, respectively. Since Pharo (as Smalltalk) is pure object-oriented, any new class must inherit from a superclass, in this case the generic class '*Object*'.

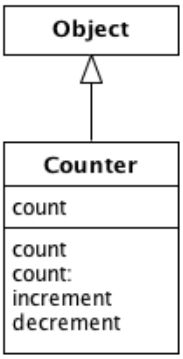

Figure 1. UML diagram for "Counter"

[1] Based on "Learning Object-Oriented Programming, Design and TDD with Pharo", Ducasse & Pollet, 2017. Chapter 4. Please find URL to full book on Moodle.

The following should be a valid code to test your program:

```
| c |                                  "declares variable c"
c := Counter new.                      "creates object Counter and assigns to c"
c increment increment increment.       "sends increment message (3 times) to object c"
c count:5.                             "sets counter to 5 (sends setter message to c)"
c decrement decrement                  "sends decrement message (2 times) to object c"
```

Exercise 2. Implement Vehicle Tax system in Pharo

In this exercise, we will implement simple vehicle taxation system. We will create three classes in the `MyVehicle` package as follows:

- A `Vehicle` class with:
  - Instance variables `age` and `wheels`
  - A getter and setter for `age`
  - A getter for `wheels`
- A `Car` class which inherits from `Vehicle`, and initialises `wheels` to 4 upon creation
- A `Bike` class which inherits from `Vehicle`, and initialises `wheels` to 2 upon creation

On the `Vehicle` class, implement a method called `computeTax`, which performs the following logic:

- If this vehicle has two wheels, return the result of the formula **100 + 2.5*a*** where a is the vehicle age
- Otherwise, return the result of the formula **150 + 6*a*** where a is the vehicle age

You should be able to test your implementation using the following code:

```
| car |
car := Car new.
car age: 10.
car computeTax
```

which should result in the value **210**, and

```
| bike |
bike := Bike new.
bike age: 5.
bike computeTax
```

which should result in the value **112.5**.

Exercise 3. Compute Pi in Pharo

In this exercise, we will write a Smalltalk function to compute Pi using the well-known Gregory-Leibniz Series. Create a class called `PiComputer` with a single method named `computePiForIterations:`, which takes an argument `numIterations`. Then, implement the function using the following pseudocode:

```
function computePi(numIterations)
        x := 1

        for i := 1 to numIterations
                if i is even
                        x := x + 1/(2i+1)
                else
                        x := x - 1/(2i+1)
                end if
        end for

        x := 4x

        return x
end function
```

Hint: To see if a `Number` is even, send it the `even` message

Note: Division in Pharo results in a fraction rather than a decimal, which makes it difficult to see when your output is correct. Therefore, you should initialize the variable `x` as a Float object using the following code:

```
x := Float new + 1.
```

You can then test your implementation using the following code:

```
PiComputer new computePiForIterations: 100
```

See how many digits of π you can compute correctly by increasing the iterations!