Prof. O. Nierstrasz
David Röthlisberger, Timur Altun

12. Virtual Machines and Repetition

This exercise has two parts: The first covers the Virtual Machine lecture of Adrian Lienhard while the second part is a repetition for the entire course. These repetition questions are similar to those we will ask to you in the exam. For the repetition questions, we suggest you to not use a Pharo image while answering them to simulate exam conditions.

1 VM

Exercise 12.1: VM – Mark Sweep Collector

Implement a mark sweep collector in pseudo code. To simplify the task, let's assume we have only direct pointer fields. Furthermore, `numberOfFieldsOf:` is given. Also assume the existence of methods for accessing fields (like `fetchPointer:ofObject:` on p17 of the lecture slides), and setting and reading of object header bits.

Exercise 12.2: VM – Load Average

Implement a load average statistics like the one that `uptime` or `top` shows in UNIX-like systems. This can be implemented entirely in Smalltalk in the usual Pharo image without modifying the VM.

2 Repetition

Exercise 12.3: Knowledge questions

1. What is the difference between an class variable and class instance variable? Explain with an example.

2. What are the differences between `self` and `super`? Give an example highlighting these differences.

3. What is `thisContext`?

4. What is a metaclass? Why does `Metaclass` ultimately inherit from `Object`?

5. Explain double dispatch with a concrete example.

6. Explain what a Feature Envy is and why it is considered as bad practice. What are typical strategies to correct a Feature Envy?

Exercise 12.4: Code reading

What are the results of the following expressions?

```
1. 'hello' at: 1 put: $H; yourself
```

```
2. [:x :y | [:t | y value +  t * (x value: 2) ] value: 3]
       value: [:i | i * 3]  value: [1 + 3].
```

```
3. ((4@6) y @ 3) x
```

Exercise 12.5: Collections

1. Answer the following questions:

- What is the result of evaluating

```
#(1 2 3.4 5) select: [:each | each > 3]
```

- How do you implement the same behavior using `reject:`?

2. In Pharo's collection class we have a method `#detect:ifNone:`. The comment describes how it works:

```
detect: aBlock ifNone: exceptionBlock
"Evaluate aBlock with each of the receiver's elements as the argument.
Answer the first element for which aBlock evaluates to true. If none
evaluate to true, then evaluate the argument, exceptionBlock."
.....
```

Please implement this method using `#do:`.

3. There is a method defined in class `Array`:

```
Array>>mysteryMethod: aBlock
  | retval |
  retval := self species new: self size - 1.
    1 to: self size - 1 do:  [:i |
       retval at: i put: (aBlock value: (self at: i) value: (self at: i+1))].
    ^retval
```

What is the result of `#(1 2 3 4 5) mysteryMethod:[:a :b | b+a]`?

Exercise 12.6: Classes, metaclasses and inheritance

1. In Pharo, it is possible to check if a given object is a class or not by sending the message isBehavior to it. How would you implement such a feature?

2. We evaluate "Object new". In which class is `#new` implemented?

3. What are the results of the following expressions:

- Integer class class
- Metaclass superclass
- Class isKindOf: Object

Exercise 12.7: Seaside

Study the following code. It shows the definition of the class `MyCounter`. Assume that this component is registered as the root of your application. We show all instance-side methods:

```
WAComponent subclass: #MyCounter
  instanceVariableNames: 'count'
  classVariableNames: ''
  poolDictionaries: ''
```

```
  category: 'Exam'

initialize
  super initialize.
  self count: 0

increase
  count := count + 1

decrease
  count := count - 1

count
  ^ count

count: anInteger
  count := anInteger

renderContentOn: html
  html heading: count.
  html anchor
    callback: [ self increase ];
    with: '++'.
  html space.
  html anchor
    callback: [ self decrease ];
    with: '--'
```

Answer the following questions:

1. What happens if you press '++' ?
2. What happens if you now press the back button?
3. What happens if your press '++' again?
4. What do you need to do to fix the back-button behavior?

Exercise 12.8: Bytecode

Have a look at the following code:

```
testExercise
 | irMethod aCompiledMethod res |
 irMethod:= IRBuilder new
  numRargs:  2;
  addTemps: #(self a);      "receiver and args"
  pushTemp: #a;
  pushLiteral: 4;
  pushLiteral: 3;
  send: #@;
  send: #extent:;
  returnTop;
  ir.
```

Write down the Smalltalk code of a method doing the same as the generated bytecode.

Please save the code and send it by mail to st-staff@iam.unibe.ch. Attach your written solutions that are not part of the source-code to the mail or hand them in as hardcopy at the beginning of the next exercise session. Your mail and solutions should be clearly marked with names and matrikel numbers of the solution authors.