INF5510 week5 exercises

Questions can be sendt to **magnushc@ifi.uio.no**.

How to start

To get started, you must first install the Emerald compiler and follow the instruction on the link given. Afterwards, try to run the hello world example below. Copy the code in a file, for example *hello.m*

```
const hello <- object hello
    initially
        stdout.putstring["Hello World!\n" ]
    end initially
end hello
```

To compile and run, copy/save *hello.m* and do the following shown below in a terminal.

```
$ emc
Command: hello.m
Compiling hello.m
Command: q
$ emx hello.x
Hello World !
```

Alternative, you could use **ec** instead of **emc** to just compile the file. Whit **ec** you give the filename as argument.

```
$ ec hello.m
Compiling hello.m
$ emx hello.x
Hello World !
```

Exercises

Exercise 1 - Types

Exercise 1 - Types Types in Emerald contains a collection of operationsignatures, together with operation names, parameters, parametertypes and returnvalues. Implement an object that conforms to the following typeobject:

```
const SimpleCollection <- typeobject SimpleCollection
    operation add [ navn : String ] -> [ res : Boolean ]
```

```
    function contains [ navn : String ] ->[ res : Boolean ]
    operation remove [ navn : String ] ->[ res : Boolean ]
end SimpleCollection
```

Exercise 2 - Classes

In Emerald, we create objects with the help of an object constructor. Sometimes it can be an advantage to create a model of an object to use it more than once. It do not exists a class notation in Emerald, but there is a syntactic construct called class, that provides the functionality normally expected of classes. Emerald supports inheritance, that means the given object can only inherit from a superclass. It does not exist a constructor that we find in for example java, but classes can also take parameters: For example:

```
const parent <- class parent [ < Parametere > ]
    //Variables and operations
end parent
const child <- class child ( parent ) [ < Parametere > ]
    //Variables and operations
end child
```

Create a simple Person - Teacher hierarchy where the person object should have at last one attribute name and the teacher should have at least one attribute for position.

Exercise 3 - Sieve of Eratosthenes

The exercise is about creating a primary number generator with the help of Sieve of Eratosthenes. For more information, look at: `http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes`. The numbers from 2-100 should be supported.

Exercise 4 - The Moving Man

Create a program where an object (called moving man) moves and visits each node and collects their Node identification. When the man is done visiting all the nodes, print out the id's of the active nodes. Optionally, use unavailable or failure statement if the node crashes.

Exercise 5 - Watchdog

Watchdog should check the Emerald nodes if they have gone down or is up again. Create a program where there are more machines that are up, and crash some of them to see if you get notified which node crashed (printout node id and hostname).

Exercise 6 - Datastructure

Implement a datastructure that conforms to SimpleCollection that was shown from earlier assignment. (You can choose what type of datastructure you want to create, a simple one would be creating a simple List). Afterwards, create a

couple of tests where you add, find and remove the string you have stored in the datastructure.