Final Exercises Sheet

Programming Scalable Systems

Universidad Politécnica de Madrid

2022-2023

- These selected exercises will be used to grade you.
- Review class material and class book [Laurent and Eisenberg, 2014].
- We assume you have done exercice sheets 1, 2, 3, and 4, exercises here are a selection of exercises in those sheets.

Mandatory Submission

The deadline for completing this sheet is **May 20th**.

Submission will be available in `http://deliverit.fi.upm.es`. You will have to submit all your code in one module *`Final`* in a file `final.ex`:

```
defmodule Final do
  # Your code goes here
  ...
end
```

Exercises

String **Exercise 1. (10%)** Define a function `date_parts` which takes a string in ISO date format `"yyyy-mm-dd"` and returns a list of integers in the form `[yyyy, mm, dd]`. This function does not need to do any error checking.

```
iex(1)> c("final.ex")
[Dates]
iex(2)> Final.date_parts("2013-06-15")
[2013,6,15]
```

Use the *`String`*`.split/3` function to accomplish this task.

Fibs **Exercise 2. (10%)** Define a function `fibs` such that `fibs(n)` returns a list containing the elements of the Fibonacci sequence upto the `n`-th element:

```
@spec fibs(pos_integer()) :: [pos_integer()]
```

Note: We expect an implementation with complexity $O(n)$. Use private functions if you need.

HO **Exercise 3. (20%)** Write *the classical* higher order functions on lists:

```
@spec map([a], (a -> b)) :: [b] when a: any(), b: any()
@spec reduce([a], b, ((a, b) -> b)) :: b when a: any(), b: any()
@spec filter([a], (a -> boolean())) :: [a] when a: any()
```

Bank **Exercise 4. (40%)** Implement in Elixir a simple bank application, and test it. Use a submodule *Bank* in module *Final*, fill in the following skeleton functions:

```
defmodule Final do
  ...
  defmodule Bank do
    #  Creates a bank and returns a "bank" pid
    def create_bank() do
      # fill in
    end

    # Creates a new account with account balance 0. Returns true if the
    # account could be created (it was new) and false otherwise.
    def new_account(bank,account) do
      # fill in
    end

    # Withdraws money from the account (if quantity =< account balance).
    # Returns the amount of money withdrawn.
    def withdraw(bank,account,quantity) do
      # fill in
    end

    #  Increases balance of account by quantity, returning the new balance.
    def deposit(bank,account,quantity) do
      # fill in
    end

    #  Transfers quantity from one account (if the balance is sufficient)
    #  to another account. Returns the amount of money transferred.
    def transfer(bank,from_account, to_account, quantity) do
      # fill in
    end

    #  Returns the current balance for the account
    def balance(bank,account) do
      # fill in
    end
  end
end
```

Implement the bank internally as a process, which listens to sent messages and responds to them, similar to the skeleton below.

OTPBank **Exercise 5. (20%)** In exercise 4 you have been asked to implement a *bank server*. Implement the server using OTP *GenServer*[1]. Use *Final.OTPBank* as module name to implement the server.

References

[Laurent and Eisenberg, 2014] Laurent, S. S. and Eisenberg, J. D. (2014). *Introducing Elixir*. O'Reilly. **class book**.

[1] https://hexdocs.pm/elixir/GenServer.html