



OCaml

Object Oriented Programming - 1

Summary: The main theme of this module is to introduce the object oriented programming style with OCaml.

Version: 1.00

Contents

I	Foreword : The ultimate mega gangsta chocolate butter cake	2
I.1	Ingredients :	2
I.2	Recipe :	2
II	General rules	3
III	Ocaml piscine, general rules	4
IV	Day-specific rules	6
V	Exercise 00: Do What I do. Hold tight and pretend it's a plan!	7
VI	Exercise 01: The Name Of The Doctor!	8
VII	Exercise 02: You are a good Daaaaaalek!	10
VIII	Exercise 03: The Day of The Doctor!	12
IX	Exercise 04: The Time War!	13
X	Submission and peer-evaluation	15

Chapter I

Foreword : The ultimate mega gangsta chocolate butter cake

I.1 Ingredients :

- 250g of chocolate
- 250g of butter (Yeah... I know)
- 150g of sugar
- 4 eggs
- 1 big spoon of flour

I.2 Recipe :

- Melt the chocolate with the butter.
- Add sugar.
- Add eggs one by one by mixing between each one.
- Add flour progressively while mixing the whole thing
- Pour into a cake plate (or something similar)
- Put the cake plate into a larger plate with a little water in it (bain marie style)
- Put the two plates like this in the oven for 45 minutes at 180 degrees
- Put it 12 hours (Not a joke) into your refrigerator.

Done!

Chapter II

General rules

- Your project must be realized in a virtual machine.
- Your virtual machine must have all the necessary software to complete your project. These softwares must be configured and installed.
- You can choose the operating system to use for your virtual machine.
- You must be able to use your virtual machine from a cluster computer.
- You must use a shared folder between your virtual machine and your host machine.
- During your evaluations you will use this folder to share with your repository.
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and will receive a 0 during the evaluation.
- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be done after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.

Chapter III

Ocaml piscine, general rules

- Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.
- The imposed filenames must be followed to the letter, as well as class names, function names and method names, etc.
- Unless otherwise explicitly stated, the keywords `open`, `for` and `while` are forbidden. Their use will be flagged as cheating, no questions asked.
- Turn-in directories are `ex00/`, `ex01/`, ..., `exn/`.
- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description.
- Since you are allowed to use the OCaml syntaxes you learned about since the beginning of the piscine, you are not allowed to use any additional syntaxes, modules and libraries unless explicitly stated otherwise.
- The exercises must be done in order. The graduation will stop at the first failed exercise. Yes, the old school way.
- Read each exercise FULLY before starting it! Really, do it.
- The compiler to use is `ocamlopt`. When you are required to turn in a function, you must also include anything necessary to compile a full executable. That executable should display some tests that prove that you've done the exercise correctly.
- Remember that the special token `";;"` is only used to end an expression in the interpreter. Thus, it must never appear in any file you turn in. Regardless, the interpreter is a powerful ally, learn to use it at its best as soon as possible!
- The subject can be modified up to 4 hours before the final turn-in time.
- In case you're wondering, no coding style is enforced during the OCaml piscine. You can use any style you like, no restrictions. But remember that a code your peer-evaluator can't read is a code he or she can't grade. As usual, big functions are a weak style.
- You will NOT be graded by a program, unless explicitly stated in the subject. Therefore, you are given a certain amount of freedom in how you choose to do the

exercises. However, some piscine day might explicitly cancel this rule, and you will have to respect directions and outputs perfectly.

- Only the requested files must be turned in and thus present on the repository during the peer-evaluation.
- Even if the subject of an exercise is short, it's worth spending some time on it to be absolutely sure you understand what's expected of you, and that you did it in the best possible way.
- By Odin, by Thor! Use your brain!!!


Chapter IV

Day-specific rules

- This day is pure fun. Bonus points will be allowed if you provide clever references to the Doctor Who universe.
- You are in a functional programming piscine, so your coding style **MUST** be functional (Except for the side effects for the input/output). I insist, your code **MUST** be functional, otherwise you'll have a tedious defence session.
- For each exercise of the day, you must provide sufficient material for testing during the defence session. **Every functionality that can't be tested won't be graded!**
- Don't be lazy. If something is not tested in an exercise, the defence will stop immediately.

Chapter V


Exercise 00: Do What I do. Hold tight and pretend it's a plan!

	Exercise 00
Exercise 00: Do What I do. Hold tight and pretend it's a plan!	
Turn-in directory : <i>ex00/</i>	
Files to turn in : people.ml, main.ml, Makefile	
Allowed functions : Stdlib modules	

- Write a class *people* that has the following attributes :
 - A name attribute of type string.
 - An hp attribute of type int initialized to 100.
 - A to_string method that returns the name of the object with attributes values.
 - A talk method that print the following string on the standard output :
I'm [NAME]! Do you know the Doctor?
 - A method die which prints the following sentence on the standard output :
Aaaarghh!
 - An initializer which indicate that the object has been created (feel free to use something explicit and wise to describe it!)
- You have to simulate all the methods in the main to provide sufficient testing for the defence.

Chapter VI

Exercise 01: The Name Of The Doctor!

	Exercise 01
Exercise 01: The Name Of The Doctor!	
Turn-in directory : <i>ex01/</i>	
Files to turn in : <code>doctor.ml</code> , <code>people.ml</code> , <code>main.ml</code> , <code>Makefile</code>	
Allowed functions : <code>Stdlib</code> modules	


- Write a class *doctor* that has the following attributes :
 - A name attribute of type string.
 - An age attribute of type int.
 - A sidekick attribute of type people
 - An hp attribute of type int initialized to 100.
 - A `to_string` method that returns the name of the object with attributes values.
 - A `talk` method that print the following string on the standard output :
Hi! I'm the Doctor!
 - An initializer which indicate that the object has been created (feel free to use something explicit and wise to describe it!)
 - A method `travel_in_time` which takes two arguments of type int : *start* and *arrival* and changes the age of the doctor logically (Think before coding some weird arithmetics... Please...). This method also draw a TARDIS on the standard output. (If you don't know what a TARDIS is, google it!)
 - A method `use_sonic_screwdriver` which prints the following sentence on the standard output : *Whiiiiwhiiiiwhiii Whiiiiwhiiiiwhiii Whiiiiwhiiiiwhiii*
 - A private method `regenerate` that sets the hp of the doctor to 100 (the maximum)

- You have to simulate all the methods in the main to provide sufficient testing for the defence.



Chapter VII

Exercise 02: You are a good Daaaaaalek!


	Exercise 02
Exercise 02: You are a good Daaaaaalek!	
Turn-in directory : <i>ex02/</i>	
Files to turn in : <i>dalek.ml</i> , <i>doctor.ml</i> , <i>people.ml</i> , <i>main.ml</i> , <i>Makefile</i>	
Allowed functions : <i>Pervasives</i> , <i>String</i> and <i>Random</i> modules	

- Write a class *dalek* that has the following attributes :
 - A name attribute of type string randomly generated with the format : *DalekXXX* with XXX is a random set of chars (*DalekSec* for example).
 - An hp attribute of type int initialized to 100.
 - A shield attribute of type bool mutable, initialized to true and change it's value each time the exterminate method is used.
 - A to_string method that returns the name of the object with attributes values.
 - A talk method that randomly prints one of the following strings on the standard output :
 - * *Explain! Explain!*
 - * *Exterminate! Exterminate!*
 - * *I obey!*
 - * *You are the Doctor! You are the enemy of the Daleks!*
 - A method exterminate which takes an argument of type people object and kill it instantly.
 - A method die which prints the following sentence on the standard output : *Emergency Temporal Shift!*

- You have to simulate a battle between the doctor, a dalek and a human in the main to provide sufficient testing for the defence. Also feel free to add any setter that you want.

Chapter VIII

Exercise 03: The Day of The Doctor!

	Exercise 03
Exercise 03: The Day of The Doctor!	
Turn-in directory : <i>ex03/</i>	
Files to turn in : <code>army.ml</code> , <code>dalek.ml</code> , <code>doctor.ml</code> , <code>people.ml</code> , <code>main.ml</code> , <code>Makefile</code>	
Allowed functions : <code>Stdlib</code> and <code>List</code> modules	


- Write a parameterized class *army* that has the following attributes :
 - A member attribute of type 'a list which contains a list of instance of one of the 3 previous classes.
 - An add method that adds an instance ot the list (front or back).
 - A delete method that removes the head of the list member. (front or back)
- You have to simulate a construction and a destruction of an army of each type in the main to provide sufficient testing for the defence.

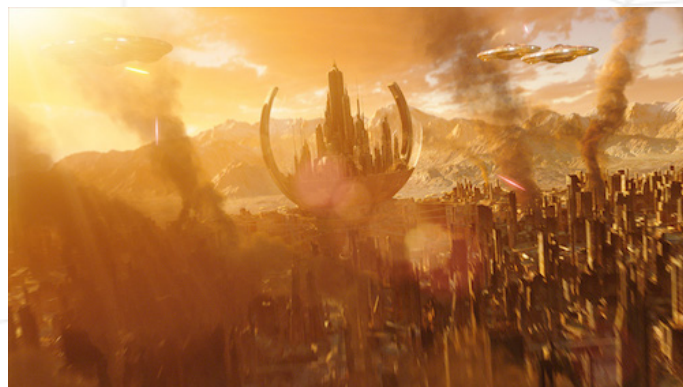


This exercise is not mandatory.

Chapter IX

Exercise 04: The Time War!

	Exercise 04
Exercise 04: The Time War!	
Turn-in directory : <i>ex04/</i>	
Files to turn in : <i>galifrey.ml</i> , <i>army.ml</i> , <i>dalek.ml</i> , <i>doctor.ml</i> , <i>people.ml</i> , <i>main.ml</i> , <i>Makefile</i>	
Allowed functions : Everything functional (so no while, for, Array etc...)	



- Write a class *galifrey* that has the following attributes :
 - A member attribute of type dalek list which contains a list of instance of dalek type.
 - A member attribute of type doctor list which contains a list of instance of doctor type.
 - A member attribute of type people list which contains a list of instance of people type.
 - A *do_time_war* method that launch the greatest battle in time and space. You will need to add more methods to handle correct behaviour. For example one to select one of the attack of an instance or another to check if there is

any object of a list that is still alive. Feel free to add any method that seems necessary.

- You have to simulate a time war in the main to provide sufficient testing for the defence.



This exercise is not mandatory.

Chapter X

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.



The evaluation process will happen on the computer of the evaluated group.