HLIN603 - TD 3

```
class min ( init : int) =
  object(self)
    val x = init
    method get = x
    method min y = if self#get < y then self#get else y
  end;;

class min_max ( init : int ) =
  object ( self )
    inherit min init
    method max y = if self#min y = y then self#get else y
  end;;

class other =
  object
    method get = 1
    method min n = n-1
    method max n = n+1
  end;;

class ['a] cell ( init : 'a ) =
  object
    val mutable cont = init
    method get = cont
    method set n = cont <- n
  end;;

let m = new min 1 ;;
let mm = new min_max 2 ;;
let o = new other ;;
let c = new cell 1 ;;
```

Exercice 1

1. Types des fonctions :

let natural (o:min) = o#min(0) = 0;;
min -> bool

let negative (o : #min) = o#min (0) = o#get ;;
#min -> bool

let positive o = o#get > 0 ;;
<get : int, ..> -> bool

2. Evaluation des appels de fonctions :

Donner les réponses d'OCaml et expliquer pourquoi.

natural m;;
- : bool = true
m est de type min = < get : int; min : int -> int >
Donc pas de problème de typage.
o#min(0) rend 0, donc natural rend true.

natural mm;;
Error : mm est de type min_max
On attend des objets qui ont exactement le type de min.
Ici min_max rajoute la méthode max.

natural o;;
Error : erreur de typage, type de l'objet attendu : min, type de l'objet passé en paramètre : other
On attend des objets qui ont exactement le type de min.
Ici other rajoute la méthode max.

natural (o :> min);; ça veut dire quoi le o:>min ? c'est un cast de o en type min
- : bool = false
Typage : other est bien un sous-type de min (on retrouve toutes les méthodes de min dans other avec les mêmes types).
o#min(0) renvoie -1
Et -1 n'est pas égal à 0

natural c;;
Erreur de typage : c est de type int cell = <get : int; set : int -> unit > et ce n'est pas compatible avec min = <get : int; min : int -> int>
Il manque la méthode min à c.

Error: This expression has type int cell
but an expression was expected of type min
The second object type (min) has no method set

let negative (o : #min) = o#min (0) = o#get ;;
#min -> bool

negative m;;
- : bool = false
Pas d'erreur de typage car on attend un objet de type "au moins" min. (#min) (un type qui contient au moins les méthodes de la classe min avec les mêmes types et éventuellement d'autres méthodes en plus = type ouvert).
Evaluation : o#min(0) = 0, o#get = 1, 0 n'est pas égal à 1, donc false

negative mm;;

- : bool = false

Pas d'erreur de typage.
mm est de type min_max, et contient toutes les méthodes de min, au moins (type ouvert dans negative).
Evaluation : o#min(0) = 0, o#get = 2, 2 n'est pas égal à 0, false.

negative o;;

- : bool = false

Pas d'erreur de typage.
o est de type other et contient toutes les méthodes de min et plus (max) mais comme le type est ouvert donc aucun problème à avoir une méthode en plus.
Evaluation : o#min(0) = -1, o#get = 1, -1 n'est pas égal à 1, false

negative c;;

error : c de type cell ne respecte pas #min
c est de type int cell = <get : int; set : int -> unit>
et on n'a pas la méthode min dans ce type, donc on ne vérifie pas la condition #min.

Error: This expression has type int cell
but an expression was expected of type #min
The first object type has no method min

let positive o = (o#get > 0) ;;
<get : int, ..> -> bool

positive m;;

- : bool = true

Pas d'erreur de typage.
m est de type min = <get :int; min : int -> int>
ce qui est compatible avec le type ouvert attendu par la fonction : <get : int; .. >
o#get = 1 > 0, true

positive mm;;

- : bool = true

Pas d'erreur de typage.
mm est de type min-max = <get : int; min : int -> int; max : int -> int>
ce qui est compatible avec le type ouvert attendu par la fonction : <get : int; .. >
o#get = 2 > 0, true

positive o;;

- : bool = true

Pas d'erreur de typage.
o est de type other = <get : int; min : int -> int; max : int -> int>
ce qui est compatible avec le type ouvert attendu par la fonction : <get : int; .. >
o#get = 1 > 0, true

positive c;;

- : bool = true

Pas d'erreur de typage.
c est de type int cell = <get : int; set : int -> unit>
ce qui est compatible avec le type ouvert attendu par la fonction : <get : int; .. >
o#get = 1 > 0, true

Exercice 2

1.

```
class virtual ['a] add_magma =
  object
    method virtual add : 'a -> 'a -> 'a
  end;;

class virtual ['a] mul_magma =
object
   method virtual mul : 'a -> 'a -> 'a
end;;
```

https://fr.wikipedia.org/wiki/Magma_(alg%C3%A8bre)

2.

```
class virtual ['a] add_monoid =
object
  inherit ['a] add_magma
  method virtual add_id : 'a
end;;


class virtual ['a] mul_monoid =
object
   inherit ['a] mul_magma
   method virtual mul_id : 'a
end;;
```

3.

```
class virtual ['a] add_group =
object
   inherit ['a] add_monoid
   method virtual add_inv : 'a -> 'a
end;;
```

‘4.

```
class virtual ['a] ring =
object
   inherit ['a] add_group
   inherit ['a] mul_monoid
end ;;
```

5.

```
class int_ring = (* pas virtual ici, on veut des int *)
  object
    inherit [int] ring
    method add x y = x + y
    method add_id = 0
    method add_inv x = -x
    method mul x y = x * y
    method mul_id = 1
  end;;
```

6.

```
class ['a, 'b] polynomial (r : 'b) (p : ('a * int) list) =
object (self)
  constraint 'b = 'a ring
  val ra = r
  val rep = p
  method private monomial (c, n) x =
    if n = 0 then c
    else ra#mul x (self#monomial (c, (n - 1)) x)
  method eval x =
    List.fold_left (fun a b -> ra#add a (self#monomial b x)) ra#add_id rep
end;;
```

Rappel :
List.fold_left f a [b1; ...; bn] is f (... (f (f a b1) b2) ...) bn.

7.

```
let ir = new int_ring;;

class int_polynomial (p : (int * int) list) =
object
  inherit [int, int ring] polynomial ir p
end;;
```

8.
(* p = x^2 - 5x + 6 *) c'est quoi
let p = new int_polynomial [(1, 2); (-5, 1); (6, 0)];;

p#eval (-3);;

#printable

ra#add_id = valeur de départ pour a
rep = valeur de départ pour b
fold_left ~ foreach