

Objective Caml version 3.06

```
(* Vergleich zwischen Modulen und Objekten *)

(* Effizienz:
   Late binding kostet Zeit, da die richtige Methode zur Laufzeit
   bestimmt wird
*)

# class virtual test () =
object
  method virtual sum: unit -> int
  method virtual sum2: unit -> int
end;;

class virtual test :
  unit ->
  object
    method virtual sum : unit -> int
    method virtual sum2 : unit -> int
  end

# class a x =
object (self)
  inherit test ()
  val a = x
  method a = a
  method sum () = a
  method sum2 () = self#a
end;;

class a :
  int ->
  object
    val a : int
    method a : int
    method sum : unit -> int
    method sum2 : unit -> int
  end

# class b x y =
object (self)
  inherit a x as super
  val b = y
  method b = b
  method sum () = b + a
  method sum2 () = self#b + super#sum2 ()
end;;

class b :
  int ->
  int ->
  object
    val a : int
    val b : int
    method a : int
    method b : int
    method sum : unit -> int
    method sum2 : unit -> int
  end

# let x = (new b 1 2);;
val x : b = <obj>

# x#sum();;
- : int = 3

# x#sum2();;
- : int = 3

(* Zum Vergleich: Statische Bindung *)

# let f a b = a + b;;
val f : int -> int -> int = <fun>

# ignore;;
- : 'a -> unit = <fun>
```

```
# let iter g a n =
  for i = 1 to n do ignore (g a) done; g a;;
val iter : ('a -> 'b) -> 'a -> int -> 'b = <fun>
```

```
# let go i j =
  match i with
  | 1 -> iter (fun x -> x#sum()) (new b 1 2) j
  | 2 -> iter (fun x -> x#sum2()) (new b 1 2) j
  | 3 -> iter (fun x -> f 1 x) 2 j;;
```

*Characters 16-165:*

*Warning: this pattern-matching is not exhaustive.  
Here is an example of a value that is not matched:*

```
0
match i with
  | 1 -> iter (fun x -> x#sum()) (new b 1 2) j
  | 2 -> iter (fun x -> x#sum2()) (new b 1 2) j
  | 3 -> iter (fun x -> f 1 x) 2 j..
val go : int -> int -> int = <fun>
```

```
# go 1 10000000;;
- : int = 3
```

```
# go 2 10000000;;
- : int = 3
```

```
# go 3 10000000;;
- : int = 3
```

(\* Uebersetzung: Module -> Klassen \*)

(\* Eine Signatur für Listen \*)

```
# module type LIST =
sig
  type 'a list = C0 | C1 of 'a * 'a list
  val add: 'a list -> 'a -> 'a list
  val empty: 'a list -> bool
  val length: 'a list -> int
  val hd: 'a list -> 'a
  val tl: 'a list -> 'a list
  val append: 'a list -> 'a list -> 'a list
  val fold_left: ('a -> 'b -> 'a) -> 'a -> 'b list -> 'b
end;;
```

```
module type LIST =
sig
  type 'a list = C0 | C1 of 'a * 'a list
  val add : 'a list -> 'a -> 'a list
  val empty : 'a list -> bool
  val length : 'a list -> int
  val hd : 'a list -> 'a
  val tl : 'a list -> 'a list
  val append : 'a list -> 'a list -> 'a list
  val fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'b
end
```

(\* Eine entsprechende abstrakte Klasse \*)

```
# class virtual ['a] list () =
object (self: 'b)
  method virtual add: 'a -> 'a list
  method virtual empty: unit -> bool
  method virtual hd: 'a
  method virtual tl: 'a list
  method virtual length: unit -> int
  method virtual append: 'a list -> 'a list
end;;
```

```
class virtual ['a] list :
  unit ->
  object
    method virtual add : 'a -> 'a list
    method virtual append : 'a list -> 'a list
    method virtual empty : unit -> bool
    method virtual hd : 'a
    method virtual length : unit -> int
```

```

    method virtual tl : 'a list
  end

(* und zwei Klassen, die den Konstruktoren C0 und C1 entsprechen,
   zunaechst ohne fold_left *)

# class ['a] c1_list (t,q) =
object (self)
  inherit ['a] list () as super
  val t = t
  val q = q
  method add x = new c1_list (x, (self: 'a #list :> 'a list))
  method empty () = false
  method length () = 1 + q#length ()
  method hd = t
  method tl = q
  method append l = new c1_list (t, q#append l)
end;;

class ['a] c1_list :
'a * 'a list ->
object
  val q : 'a list
  val t : 'a
  method add : 'a -> 'a list
  method append : 'a list -> 'a list
  method empty : unit -> bool
  method hd : 'a
  method length : unit -> int
  method tl : 'a list
end

# class ['a] c0_list () =
object (self)
  inherit ['a] list () as super
  method add x = new c1_list (x, (self: 'a #list :> 'a list))
  method empty () = true
  method length () = 0
  method hd = failwith "c0_list hd"
  method tl = failwith "c0_list tl"
  method append l = l
end;;

class ['a] c0_list :
unit ->
object
  method add : 'a -> 'a list
  method append : 'a list -> 'a list
  method empty : unit -> bool
  method hd : 'a
  method length : unit -> int
  method tl : 'a list
end

# let l = new c1_list (4, new c1_list (7, new c0_list ()));;
val l : int list = <obj>

# l#hd;;
- : int = 4

# l#tl#hd;;
- : int = 7

# let ll = l#append l;;
val ll : int list = <obj>

# ll#length ();;
- : int = 4

(* Jetzt wird fold_left hinzugefügt *)

# class virtual ['a,'b] fold_left () =
object (self)
  method virtual f: 'a -> 'b -> 'a
  method iter r (l: 'b list) =
    if l#empty () then r else self#iter (self#f r (l#hd)) (l#tl)
end;;

```

```

class virtual ['a, 'b] fold_left :
  unit ->
  object
    method virtual f : 'a -> 'b -> 'a
    method iter : 'a -> 'b list -> 'a
  end

# class ['a,'b] gen_fl f =
object
  inherit ['a,'b] fold_left ()
  method f = f
end;;

class ['a, 'b] gen_fl :
  ('a -> 'b -> 'a) ->
  object method f : 'a -> 'b -> 'a method iter : 'a -> 'b list -> 'a end

# let afl = new gen_fl (+);;
val afl : (int, int) gen_fl = <obj>

# afl#iter 0 l;;
- : int = 11

# afl#iter 0 ll;;
- : int = 22

(* Simulation von Vererbung in Modulen *)

# module type POINT =
sig
  type point
  val new_point: int * int -> point
  val get_x: point -> int
  val get_y: point -> int
  val moveto: point -> (int * int) -> unit
  val rmoveto: point -> (int * int) -> unit
  val display: point -> unit
  val distance: point -> float
end;;

module type POINT =
  sig
    type point
    val new_point : int * int -> point
    val get_x : point -> int
    val get_y : point -> int
    val moveto : point -> int * int -> unit
    val rmoveto : point -> int * int -> unit
    val display : point -> unit
    val distance : point -> float
  end

# module ColoredPoint =
  functor (P: POINT) ->
  struct
    type colored_point = {p: P.point; c: string}
    let new_colored_point p c = {p = P.new_point p; c = c}
    let get_c self = self.c
    let get_x self = let super = self.p in P.get_x super
    let moveto self = let super = self.p in P.moveto super
  end;;

module ColoredPoint :
  functor (P : POINT) ->
  sig
    type colored_point = { p : P.point; c : string; }
    val new_colored_point : int * int -> string -> colored_point
    val get_c : colored_point -> string
    val get_x : colored_point -> int
    val moveto : colored_point -> int * int -> unit
  end

```