

Objective Caml version 3.06

```
# class point (x_init, y_init) =
object
  val mutable x = x_init
  val mutable y = y_init
  method get_x = x
  method get_y = y
  method moveto (a, b) = x <- a; y <- b
  method rmoveto (dx, dy) = x <- x + dx; y <- y + dy
  method distance () = sqrt (float (x*x + y*y))
  method to_string () =
    "(" ^ string_of_int x ^ ", " ^ string_of_int y ^ ")"
end;;

class point :
int * int ->
object
  val mutable x : int
  val mutable y : int
  method distance : unit -> float
  method get_x : int
  method get_y : int
  method moveto : int * int -> unit
  method rmoveto : int * int -> unit
  method to_string : unit -> string
end

# let p1 = new point (0,0);;
val p1 : point = <obj>

# type point =
<distance : unit -> float;
get_x : int;
get_y : int;
moveto : int * int -> unit;
rmoveto : int * int -> unit;
to_string: unit -> string;;
type point =
< distance : unit -> float; get_x : int; get_y : int;
moveto : int * int -> unit; rmoveto : int * int -> unit;
to_string : unit -> string >

# let p2: point = new point (3,4);;
val p2 : point = <obj>

# type simple_point =
<get_x : int;
get_y : int>;;
type simple_point = < get_x : int; get_y : int >

# let p3: simple_point = new point (3,0);;
Characters 23-38:
let p3: simple_point = new point (3,0);;
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

This expression has type
point =
  < distance : unit -> float; get_x : int; get_y : int;
  moveto : int * int -> unit; rmoveto : int * int -> unit;
  to_string : unit -> string >
but is here used with type simple_point = < get_x : int; get_y : int >
Only the first object type has a method distance

# let p3: point = new point (3,0);;
val p3 : point = <obj>

# p1#get_x;;
- : int = 0

# p2#get_y;;
- : int = 4

# p1#to_string;;
- : unit -> string = <fun>

# p1#to_string ();;
- : string = "(0, 0)"

# p2#to_string ();;
```

```

- : string = "(3, 4)"
# p1#move_to (5,6);;
Characters 0-2:
  p1#move_to (5,6);;
  ^^
This expression has type point
It has no method move_to

# p1#moveto (5,6);;
- : unit = ()

# p1#to_string ();;
- : string = "(5, 6)"

# p2#to_string ();;
- : string = "(3, 4)"

# let make_point x = new point (x, x);;
val make_point : int -> point = <fun>

# let p4 = make_point 5;;
val p4 : point = <obj>

# class picture n =
object
  val mutable ind = 0
  val tab = Array.create n (new point (0,0))    (* immer der gleiche Punkt! *)
  method add p =
    tab.(ind) <- p; ind <- ind + 1
  method remove () =
    if ind > 0 then ind <- ind - 1
  method to_string () =
    let s = ref "["
    in for i = 0 to ind - 1 do s := !s ^ " " ^ tab.(i)#to_string () done;
    (!s) ^ "]"
end;;

                                class picture :
int ->
object
  val mutable ind : int
  val tab : point array
  method add : point -> unit
  method remove : unit -> unit
  method to_string : unit -> string
end

# let pic = new picture 8;;
val pic : picture = <obj>

# pic#to_string ();;
- : string = "[]"

# pic#add p1; pic#add p2; pic#add p3;;
- : unit = ()

# pic#to_string ();;
- : string = "[ (5, 6) (3, 4) (3, 0)]"

# p1#moveto (0,1);;
- : unit = ()

# pic#to_string ();;
- : string = "[ (0, 1) (3, 4) (3, 0)]"

# class colored_point (x, y) c =
object
  inherit point (x, y)
  val mutable c = c
  method get_color = c
  method set_color nc = c <- nc
  method to_string () =
    "(" ^ string_of_int x ^ ", " ^ string_of_int y ^ ")" ^
    " [" ^ c ^ "]"
end;;

                                class colored_point :
int * int ->
string ->

```

```

object
  val mutable c : string
  val mutable x : int
  val mutable y : int
  method distance : unit -> float
  method get_color : string
  method get_x : int
  method get_y : int
  method moveto : int * int -> unit
  method rmoveto : int * int -> unit
  method set_color : string -> unit
  method to_string : unit -> string
end

# let pc = new colored_point (2,3) "white";;
val pc : colored_point = <obj>

# pc#get_color;;
- : string = "white"

# pc#distance ();;
- : float = 3.60555127546

# pc#to_string ();;
- : string = "(2, 3) [white]"

# p1 = pc;;
Characters 5-7:
  p1 = pc;;
    ^^
This expression has type
  colored_point =
    < distance : unit -> float; get_color : string; get_x : int; get_y :
      int; moveto : int * int -> unit; rmoveto : int * int -> unit;
      set_color : string -> unit; to_string : unit -> string >
but is here used with type
  point =
    < distance : unit -> float; get_x : int; get_y : int;
      moveto : int * int -> unit; rmoveto : int * int -> unit;
      to_string : unit -> string >
Only the first object type has a method get_color

# class colored_point (x, y) c =
object (self)
  inherit point (x, y) as super
  val mutable c = c
  method get_color = c
  method set_color nc = c <- nc
  method to_string () =
    super#to_string () ^ " [" ^ self#get_color ^ "]"
end;;

      class colored_point :
int * int ->
string ->
object
  val mutable c : string
  val mutable x : int
  val mutable y : int
  method distance : unit -> float
  method get_color : string
  method get_x : int
  method get_y : int
  method moveto : int * int -> unit
  method rmoveto : int * int -> unit
  method set_color : string -> unit
  method to_string : unit -> string
end

# class colored_point_1 coord c =
object
  inherit colored_point coord c
  val true_colors = ["white"; "black"]
  method get_color = if List.mem c true_colors then c else "UNKNOWN"
end;;

      class colored_point_1 :
int * int ->
string ->
object

```

```

    val mutable c : string
    val true_colors : string list
    val mutable x : int
    val mutable y : int
    method distance : unit -> float
    method get_color : string
    method get_x : int
    method get_y : int
    method moveto : int * int -> unit
    method rmoveto : int * int -> unit
    method set_color : string -> unit
    method to_string : unit -> string
end

# let p1 = new colored_point (1,1) "blue";;
val p1 : colored_point = <obj>

# let p2 = new colored_point_1 (1,1) "blue";;
val p2 : colored_point_1 = <obj>

# p1#to_string ();;
- : string = "(1, 1) [blue]"

# p2#to_string ();;
- : string = "(1, 1) [UNKNOWN]"

# class point_m1 (x0, y0) =
object (self)
  inherit point (x0, y0) as super
  val mutable old_x = x0
  val mutable old_y = y0
  method private mem_pos () = old_x <- x; old_y <- y
  method undo () = x <- old_x; y <- old_y
  method moveto (x1, y1) = self#mem_pos (); super#moveto (x1, y1)
  method rmoveto (dx, dy) = self#mem_pos (); super#rmoveto (dx, dy)
end;;

      class point_m1 :
int * int ->
object
  val mutable old_x : int
  val mutable old_y : int
  val mutable x : int
  val mutable y : int
  method distance : unit -> float
  method get_x : int
  method get_y : int
  method private mem_pos : unit -> unit
  method moveto : int * int -> unit
  method rmoveto : int * int -> unit
  method to_string : unit -> string
  method undo : unit -> unit
end

# let p = new point_m1 (0,0);;
val p : point_m1 = <obj>

# p#mem_pos ();;
Characters 0-1:
  p#mem_pos ();;
  ^
This expression has type point_m1
It has no method mem_pos

# p#moveto (1,1);;
- : unit = ()

# p#to_string ();;
- : string = "(1, 1)"

# p#undo ();;
- : unit = ()

# p#to_string ();;
- : string = "(0, 0)"

# class point_m2 (x, y) =
object
  inherit point_m1 (x,y) as super

```

```

    method f () = super#mem_pos ()
end;;
class point_m2 :
int * int ->
object
  val mutable old_x : int
  val mutable old_y : int
  val mutable x : int
  val mutable y : int
  method distance : unit -> float
  method f : unit -> unit
  method get_x : int
  method get_y : int
  method private mem_pos : unit -> unit
  method moveto : int * int -> unit
  method rmoveto : int * int -> unit
  method to_string : unit -> string
  method undo : unit -> unit
end

# class virtual printable () =
object (self)
  method virtual to_string: unit -> string
  method print () = print_string (self#to_string ())
end;;
class virtual printable :
unit ->
object
  method print : unit -> unit
  method virtual to_string : unit -> string
end

# class point (x_init, y_init) =
object
  inherit printable ()
  val mutable x = x_init
  val mutable y = y_init
  method get_x = x
  method get_y = y
  method moveto (a, b) = x <- a; y <- b
  method rmoveto (dx, dy) = x <- x + dx; y <- y + dy
  method distance () = sqrt (float (x*x + y*y))
  method to_string () =
    "(" ^ string_of_int x ^ ", " ^ string_of_int y ^ ")"
end;;
class point :
int * int ->
object
  val mutable x : int
  val mutable y : int
  method distance : unit -> float
  method get_x : int
  method get_y : int
  method moveto : int * int -> unit
  method print : unit -> unit
  method rmoveto : int * int -> unit
  method to_string : unit -> string
end

# let p = new point (1,1);;
val p : point = <obj>

# p#print ();;
(1, 1)- : unit = ()

# class pair x0 y0 =
object
  val x = x0
  val y = y0
  method fst = x
  method snd = y
end;;
      Characters 5-90:
..... pair x0 y0 =
object
  val x = x0
  val y = y0
  method fst = x

```

```

    method snd = y
  end..
Some type variables are unbound in this type:
class pair :
  'a ->
  'b -> object val x : 'a val y : 'b method fst : 'a method snd : 'b end
The method fst has type 'a where 'a is unbound

# class ['a, 'b] pair (x0: 'a) (y0: 'b) =
object
  val x = x0
  val y = y0
  method fst = x
  method snd = y
end;;

      class ['a, 'b] pair :
  'a ->
  'b -> object val x : 'a val y : 'b method fst : 'a method snd : 'b end

# let p = new pair 1 0;;
val p : (int, int) pair = <obj>

# let pl = new pair 1 true;;
val pl : (int, bool) pair = <obj>

# p#fst;;
- : int = 1

# pl#fst;;
- : int = 1

# new pair;;
- : 'a -> 'b -> ('a, 'b) pair = <fun>

# class ['a, 'b] acc_pair (x0: 'a) (y0: 'b) =
object
  inherit ['b, 'a] pair y0 x0
  method get1 z = if x = z then y else raise Not_found
  method get2 z = if y = z then x else raise Not_found
end;;

      class ['a, 'b] acc_pair :
  'a ->
  'b ->
  object
    val x : 'b
    val y : 'a
    method fst : 'b
    method get1 : 'b -> 'a
    method get2 : 'a -> 'b
    method snd : 'a
  end

# new acc_pair;;
- : 'a -> 'b -> ('a, 'b) acc_pair = <fun>

#

```