

Objective Caml version 3.08.3

```

(* ----- *)
(* Objekttyp für "Zählerobjekte" *)
(* ----- *)

# type counter = <get: int; inc: unit>;

type counter = < get : int; inc : unit >

(* ----- *)
(* Einzelnes Zählerobjekt *)
(* ----- *)

# let c: counter =
object
  val mutable x = 0
  method get = x
  method inc = x <- x + 1
end;;

val c : counter = <obj>

# c#get;;

- : int = 0

# c#inc;;

- : unit = ()

# c#inc;;

- : unit = ()

# c#get;;

- : int = 2

(* ----- *)
(* Eine Funktion auf Zählerobjekten *)
(* ----- *)

# let inc2 (c: counter) = (c#inc; c#inc);;

val inc2 : counter -> unit = <fun>

# inc2 c;;

- : unit = ()

# c#get;;

- : int = 4

(* ----- *)
(* Eine Zählerklasse *)
(* ----- *)

# class counter =
object
  val mutable x = 0
  method get = x
  method inc = x <- x + 1
end;;

class counter :
  object val mutable x : int method get : int method inc : unit end

# let (c1, c2) = (new counter, new counter);;

val c1 : counter = <obj>
val c2 : counter = <obj>

# c1#get;;

- : int = 0

```

```

# c1#inc;;
- : unit = ()

# inc2 c2;;
- : unit = ()

# c1#get;;
- : int = 1

# c2#get;;
- : int = 2

(* ----- *)
(* Ein erweiterter Zählertyp mit neuer Methode "reset" *)
(* ----- *)

# type reset_counter = <get: int; inc: unit; reset: unit>;

type reset_counter = < get : int; inc : unit; reset : unit >

(* ----- *)
(* Ein reset_counter-Objekt kann von der Klasse counter erben. *)
(* ----- *)

# let rc: reset_counter =
object
  inherit counter
  method reset = x <- 0
end;;

val rc : reset_counter = <obj>

# rc#get;;
- : int = 0

(* ----- *)
(* Eine reset_counter-Klasse kann von der Klasse counter erben. *)
(* ----- *)

# class reset_counter =
object
  inherit counter
  method reset = x <- 0
end;;

class reset_counter :
object
  mutable x : int
  method get : int
  method inc : unit
  method reset : unit
end

# let rc1 = new reset_counter;;

val rc1 : reset_counter = <obj>

# let rc2 = new reset_counter;;

val rc2 : reset_counter = <obj>

(* inc2 rc2;; *)

# rc1#inc;;
- : unit = ()

(* ----- *)
(* Die Funktion inc2 ist zu speziell definiert: Wir haben festgelegt, dass ihr *)
(* Argument vom Typ counter sein muss. Lässt man den Argumenttyp offen, so er- *)
(* hält man eine polymorphe Funktion, die auf alle Objekte anwendbar ist, die *)
(* mindestens eine Methode mit Namen inc haben. *)
(* ----- *)

```

```

# let inc2 c = (c#inc; c#inc);;

val inc2 : < inc : 'a; .. > -> 'a = <fun>

# inc2 rc2;;

- : unit = ()

# rc1#get;;

- : int = 1

# rc2#get;;

- : int = 2

(* ----- *)
(* Zusätzliche Erweiterung: Die Methode "backup" speichert den aktuellen Zähler- *)
(* lerstand ab, die Methode reset wird so verändert, dass sie den Zähler auf *)
(* den abgespeicherten Zwischenstand (statt auf 0) zurücksetzt. Dazu muss eine *)
(* neue Instanzvariable b eingeführt werden. *)
(* ----- *)

# type backup_counter = <get: int; inc: unit; reset: unit; backup: unit>;;

type backup_counter = < backup : unit; get : int; inc : unit; reset : unit >

# class backup_counter =
object
  inherit reset_counter
  val mutable b = 0
  method backup = b <- x
  method reset = x <- b
end;;

class backup_counter :
  object
    val mutable b : int
    val mutable x : int
    method backup : unit
    method get : int
    method inc : unit
    method reset : unit
  end

# let bc = new backup_counter;;

val bc : backup_counter = <obj>

# inc2;;

- : < inc : 'a; .. > -> 'a = <fun>

# inc2 bc;;

- : unit = ()

# bc#backup;;

- : unit = ()

(* ----- *)
(* Die polymorphe Funktion inc2 kann auch auf backup_counter-Objekte angewandt *)
(* werden. *)
(* ----- *)

# inc2 bc;;

- : unit = ()

# bc#get;;

- : int = 4

# bc#reset;;

- : unit = ()

```

```
# bc#get;;  
- : int = 2  
#
```