Slides to Accompany *Programming Languages and Methodologies*

R. J. Schalkoff

Chapter 11, Part 2 rev. 4

ADDENDUM: OO-Software Development in `ocaml`

Why Objects? (Why OOP?)

Benefits

- Modular. This includes:
  - Enhanced productivity (faster development)
  - **Reuse**
  - Maintainability
- Encapsulation:
  - Knowledge of implementation not necessary for use
  - Can 'hide' aspects of objects
- Natural Decomposition of the Problem
- Forces Overall Design **First**

Is It All Good?

No.

- Complexity
- Learning Curve
- Execution Penalty (slower ?)
- Increased Program/Executable Size (?)
- Suitability (e.g., "Hello World")

`ocaml` Manual (4.02) Sections

- Chapter 3: Objects in OCaml
- Chapter 6.9: Classes

Classes/Objects in ocaml

The basic syntax is shown below.

```
class-type ::= [[?]label-name:]  typexpr ->  class-type
      class-body-type

class-body-type ::= object [( typexpr )]  {class-field-spec} end
     [[ typexpr  {, typexpr} ]]  classtype-path

class-field-spec ::= inherit class-body-type
     val [mutable] [virtual] inst-var-name :  typexpr
     val virtual mutable inst-var-name :  typexpr
     method [private] [virtual] method-name :  poly-typexpr
     method virtual private method-name :  poly-typexpr
     constraint typexpr =  typexpr
```

Example: Objects in CAML

Consider first the declaration of two CAML objects in the source file of Figure 1.

```
(* vehicle.caml *)

class vehicle =
object
val mutable name = "batmobile"
method print_name = name
end;;

class boat =
object
val mutable name = "leaky"
val mutable capacity = 4
method how_big = capacity
end;;
```

Figure 1: A Pair of Caml Objects

Results:

```
# #use "vehicle.caml";;
class vehicle :
  object val mutable name : string method print_name : string end
class boat :
  object
    val mutable capacity : int
    val mutable name : string
    method how_big : int
  end
# let my_vehicle = new vehicle;;
val my_vehicle : vehicle = <obj>
# my_vehicle#print_name;;
- : string = "batmobile"
# let my_boat = new boat;;
val my_boat : boat = <obj>
# my_boat#how_big;;
- : int = 4
# my_vehicle#how_big;;
This expression has type vehicle
It has no method how_big
# my_boat#print_name;;
This expression has type boat
It has no method print_name
#
```

Instead, consider the modification of the class definitions to create a hierarchy and employ inheritance. This is shown in Figure 2.

```
(* vehicle2.caml
   employs inheritence *)

class vehicle =
object
val mutable name = "batmobile"
method print_name = name
end;;

class boat =
object
inherit vehicle
val mutable capacity = 4
method how_big = capacity
end;;
```

Figure 2: Extension of the Class Structure of Figure 1 To Allow Inheritance

```
# #use "vehicle2.caml";;
class vehicle :
  object val mutable name : string method print_name : string end
class boat :
  object
    val mutable capacity : int
    val mutable name : string
    method how_big : int
    method print_name : string
  end
# let my_vehicle = new vehicle;;
val my_vehicle : vehicle = <obj>
# let my_boat = new boat;;
val my_boat : boat = <obj>
# my_vehicle#print_name;;
- : string = "batmobile"
# my_boat#print_name;;
- : string = "batmobile"
#
```

Figure 3: Behavior of the OO System of Figure 2