CS 51 Code Review 7

Object-Oriented Programming

Sam Green and Gabbi Merz

Harvard University

Class Types and Signatures

Module Type Example

```
(* a signature for a stack module *)
module type STACK =
  sig
    exception Empty
    type element
    type stack
    val pop : stack -> (element option * stack)
    val push : element -> stack -> stack
    val top : stack -> element option
  end ;;
```

Class Type Example

```
(* class type for a stack *)
class type ['a] stack_i =
    object
        val mutable internal : 'a list
        method push : 'a -> unit
        method top : unit -> 'a option
        method pop : unit -> 'a option
    end ;;
```

Classes and Modules

Stack: Module / Functor Example

```
module MakeStack (Element: SERIALIZE)
    : (STACK with type element = Element.t) =
  struct
    exception Empty
    type element = Element.t
    type stack = element list
    let empty () : stack = []
    let push (el: element) (s: stack) : stack = el :: s
    let top (s: stack) : element = ...
    let pop (s: stack) : stack = ...
  end ;;
```

Stack: Class Example

```
class ['a] stack init =
    object(this)
        val mutable internal : 'a list = [init]
        method push e =
            internal <- e :: internal;
            ()
        method top () =
            match internal with
            | [] -> None
            | h :: _t -> Some h
        method pop () =
            match internal with
            | [] -> None
            | h :: t ->
                internal <- t;
                Some h
    end ;;
```

Stack Creation

Functional version:

```
let s = MakeStack(IntSerialize).empty ;;
```

Object-oriented version:

```
let s = new stack 5 ;;
```

Inheritance

Inheritance

```
# class student name huid = object (this)
  val mutable name: string = name
  val mutable huid: int = huid

  method print_info () =
    Printf.printf "\"%s\" HUID: %d " name huid
end

class upperclassman name huid house = object
  inherit student name huid as super
  val mutable house: string = house

  method !print_info () =
    super#print_info ();
    Printf.printf "House: %s " house
end ;;
```

Inheritance Instantiation Example

Before subclassing:

```
# let sam = new student "sam" 1 ;;
val sam : student = <obj>
# samprint_info () ;;
"sam" HUID: 1 - : unit = ()
```

After subclassing:

```
# let gabbi = new upperclassman "Gabbi" 12345678 "Kirkland" ;;
val gabbi : upperclassman = <obj>
# gabbiprint_info () ;;
"Gabbi" HUID: 12345678 House: Kirkland - : unit = ()
```

Subtyping

Supertype example

```
# class type shape =
object
  method area : float
  method bounding_box : point * point
  method center : point
  method translate : point -> unit
  method scale : float -> unit
end ;;
```

Subtype example

```
# class type quad =
object
  (* get all of the functionality from shape *)
  inherit shape
  (* add a new method *)
  method sides : float * float * float * float
end ;;

#
let sq : quad = new square_quad (3., 4.) 5. ;;

#
(* coerce our square to be a shape *)
(* Also known as an ``upcast'' *)
let a = area (sq :> shape) ;;
```