Discussion 06

Modules

Kenneth Fang (kwf37), Newton Ni (cn279)

Feb. 13, 2019

Key Concepts

- Signatures are **interfaces**

Key Concepts

- Signatures are **interfaces**
- Abstract types enable **information hiding**

Signatures

SIGNATURES ARE **NOT** IMPLEMENTATIONS

Signatures

```
module type Zero = sig
    val zero: int
end

(* Q: What is [Zero.zero]? *)
(* A: Error: Unbound module Zero *)
```

Signatures: Undersatisfied

```
module type Transition = sig
    val start: int
    val next: int -> int
end

module Count: Transition = struct
    let start = 0
end
```

Signatures: Oversatisfied

```
module type Transition = sig
    val start: int
    val next: int -> int
end

module Count: Transition = struct
    let start = 0
    let square x = x * x
    let next x = square x
end
```

Signatures: Implicit

- Q: do we need to explicitly write the signature?

Signatures: Implicit

- Q: do we need to explicitly write the signature?
- `module Count: Transition = struct ... end`
- `module Count = struct ... end`

Signatures: Implicit

- Q: do we need to explicitly write the signature?
- `module Count: Transition = struct ... end`
- `module Count = struct ... end`
- A: Nope—but use not clear until **functors** next week!

Abstract Types

- Idiom: abstract type + module $\approx$ private data + class

Abstract Types

- Idiom: abstract type + module $\approx$ private data + class
- `Stack.t` vs. `Stack`

Abstract Types

- Idiom: abstract type + module $\approx$ private data + class
- `Stack.t` vs. `Stack`
- Why private data?

Abstract Types

- Idiom: abstract type + module $\approx$ private data + class
- `Stack.t` vs. `Stack`
- Why private data?
  - Loose coupling

Abstract Types

- Idiom: abstract type + module $\approx$ private data + class
- `Stack.t` vs. `Stack`
- Why private data?
  - Loose coupling
  - Invariant upholding

Abstract Types: Loose Coupling

```
module type Set = sig
    type 'a t
    val empty: 'a t
    val insert: 'a t -> 'a -> 'a t
    val mem: 'a t -> 'a -> bool
end

module MySet: Set = struct
    type 'a t = 'a list
    ...
end
```

Abstract Types: Invariants

```
module type Nonzero = sig
    type t
    val t_of_int: int -> t option
    val int_of_t: t -> int
end

module NonzeroSealed: Nonzero = struct
    type t = int
    let t_of_int = function
    | 0 -> None
    | n -> Some n
    let int_of_t n = n
end
```

Digression: utop

- `#use "source.ml"`

Digression: utop

- `#use "source.ml"`
- `#require "library"`

Digression: utop

- `#use "source.ml"`
- `#require "library"`
- `#mod_use "source.ml"`

Digression: utop

- `#use "source.ml"`
- `#require "library"`
- `#mod_use "source.ml"`
- `#load "compiled.cmo"`

Digression: utop

- `#use "source.ml"`
- `#require "library"`
- `#mod_use "source.ml"`
- `#load "compiled.cmo"`
- `#load_rec "compiled.cmo"`

Digression: utop

- `#use "source.ml"`
- `#require "library"`
- `#mod_use "source.ml"`
- `#load "compiled.cmo"`
- `#load_rec "compiled.cmo"`
- `#trace function`

Digression: utop

- `#use "source.ml"`
- `#require "library"`
- `#mod_use "source.ml"`
- `#load "compiled.cmo"`
- `#load_rec "compiled.cmo"`
- `#trace function`
- `.ocamlinit`

Abstract Types: Experimenting

- `let v: NonzeroFree.t = 0;;`
- `let v: NonzeroSealed.t = 0;;`