

# Forum:[Hack Forums](#)

Thread: [Playing With Java 8 Lambdas and Closures.](#)

**g4143** - January 1, 2017, 9:50 am

I decided to see if I could port a simple program which uses lambdas and closures to create a dictionary(it really turns out to be a linked list of anonymous functions and closures) with Java 8. It turns out I'm not a Java pro so it took me a few minutes to figure out a solution and it may not be the best solution.

Java 8 solution

Code import java.util.Optional;

```
public class Testit {

    private interface Dict<String, Integer> {

        public Optional<String> dict(Integer i);

    }

    private static Dict<String, Integer> add(Dict d, Integer i, String s) {
        return (l) -> {
            if (l.compareTo(i) == 0) {
                return Optional.of(s);
            } else {
                return d.dict(l);
            }
        };
    }

    private static Optional<String> find(Dict d, Integer i) {
        return d.dict(i);
    }

    public static void main(String[] args) {

        Dict<String, Integer> myD
            = add(add(add((k) -> Optional.empty(), 4143, "G4143"), 8888, "me"), 7777, "you");

        for (int i : new int[]{4143, 8888, 7777, 1234, 7777, 8888, 4143}) {
            find(myD, i).ifPresent((v) -> System.out.println(v));
        }

    }
}
```

Just for comparison here's the 'almost' same solution but in Chicken Scheme.

Code (use srfi-1)

(define

```

empty
(lambda (k)
  (signal (string-append "Could not locate: " (number->string k)))
)
)

(define
  add-key
  (lambda (d k v)
    (lambda (l)
      (if (= l k)
          v
          (d l)
        )
      )
    )
  )

(define
  find-key
  (lambda (d k)
    (d k)
  )
)

(define d (add-key(add-key(add-key empty 4143 "G4143") 8888 "me") 7777 "you"))

(define
  locate-key
  (lambda (d k)
    (handle-exceptions
      exn
      (begin
        (display "Caught: ")
        (print exn)
      )
      (let ((ans (find-key d k)))
        (print ans)
      )
    )
  )
)

(for-each (lambda (x) (locate-key d x)) '(4143 8888 7777 1234 7777 8888 4143))

```

Here's two solutions in OCaml. The first is without type annotations and the second has type annotations.

OCaml without type annotations. Note this solution is very concise but you still have to juggle the types but its all in your head.

Code let empty k = None

```

let add d k v =
  fun l -> if l = k then Some v else d l

```

```
let find d k = d k
```

```
let d = add(add(add empty 4143 "G4143") 8888 "me") 7777 "you"
```

```
let () =
```

```
  List.iter
```

```
    (fun x -> Option.may (fun x -> print_endline x) (find d x))
```

```
    [4143; 8888; 7777; 1234; 7777; 8888; 4143]
```

```
Code type ('a, 'b) dict = 'a -> 'b option
```

```
let empty(k:'a):'b option = None
```

```
let add(d:('a, 'b) dict) (k:'a) (v:'b):('a, 'b) dict =
```

```
fun l -> if l = k then Some v else d l
```

```
let find(d:('a, 'b) dict) (k:'a):'b option = d k
```

```
let d = add(add(add empty 4143 "G4143") 8888 "me") 7777 "you"
```

```
let () =
```

```
  List.iter
```

```
    (fun x -> Option.may (fun x -> print_endline x) (find d x))
```

```
    [4143; 8888; 7777; 1234; 7777; 8888; 4143]
```