

Lecture 33

- › Simulating paradigms in languages that don't directly support them
- › VERY INTERESTING TOPIC

1

Simulating OOP in FP I

- › A shape is an object with 3 methods
 - › toString, getPos, move

```
datatype shape = Shape of {toString: unit -> string,  
                           getPos  : unit -> int * int  
                           move    : int * int -> shape}
```

Object is a record of functions

2

Simulating OOP in FP II

```
val i2s = Int.toString  
local  
  fun cirHelper r (x,y) () =  
    "Circle("^i2s r ^") at (" ^i2s x^ " ^i2s y^")"  
in  
  fun newCircle radius (pos as (x, y)) =  
    Shape{toString = cirHelper radius pos,  
          getPos   = fn() => pos;  
          move     = fn(a,b) => new Circle radius(x+a, y+b)}  
end  
  
val newCircle = fn: int -> int * int -> shape
```

3

Simulating OOP in FP III

```
local  
  fun recHelder (w,h) (x,y) () =  
    "Rect("^i2s w^", "^i2s h^  
          ") at (" ^i2s x^ ", " ^i2s y^")"  
in  
  fun newRect sides (pos as (x,y)) =  
    Shape{toString = recHelder sides pos,  
          getPos   = fn() => pos,  
          move     = fn(a,b) => new Rect sides (x+a, y+b)}  
end  
  
val newRect = fn: int * int -> int * int -> shape
```

4

Simulating OOP in FP IV

A small program that uses shapes:

```
val shapes = [newCircle 5 (0,0), newRect (3,42) (1,1), newCircle 1 (~1,0)]
val newShapes = map (fn (Shape s) => #move s (1,0)) shapes
```

```
val toStrings = map (fn (Shape s) => #toString s())
val s1 = toStrings shapes
val s2 = toString newShapes
```

5

Simulating OOP in C

```
typedef struct ComplexStruct* Complex;
```

```
struct ComplexStruct
{
    double re, im;
    double (*realpart)(Complex this);
    double (*imaginarypart) (Complex this);
    Complex (*add) (Complex this, Complex c);
    Complex (*multiply) (Complex this, Complex c);
};
```

What are the limits of this approach compared to an OOP language ?

6

Simulating FP in Java

- > First class function is an object with an apply method

```
interface FirstClassIntToInt {           int -> int
    public Int apply(Int x);
}
```

```
interface FirstClassObjToObj {           'a -> 'b
    public Object apply(Object x);
}
```

7

map in Java

```
static LinkedList map(FirstClassObjtoObj f, List ls) {
    if (ls.isEmpty())
        return new LinkedList();
    else {
        Object x = ls.get(0);
        List xs = ls.subList(1, ls.size());
        LinkedList res = map(f,xs);
        Object r = f.apply(x);
        res.addFirst(r);
        return res;
    }
}

fun map f [] = []
  | map f (x::xs) = f(x)::map f xs;
```

8

Using map in Java

Java:

```
FirstClassObjtoObj incr = new FirstClassObjtoObj() {  
    public Object apply(Object x) {  
        int n = ((Integer)x).intValue();  
        return new Integer(n+1);  
    }  
};
```

Linked res = map(incr, xs);

ML:

```
val res = map (fn x=>x+1) xs;
```

9

Some comments

- > Programming languages affect the way you think
- > Try to think “natively” when you program
- > Everyone did mugEngine – very few programs were good example of OOP
- > Keep rewriting your code trying to make it better even if it “works”

10