Chapter 2: Data Abstraction

2.1 Specifying Data via Interfaces

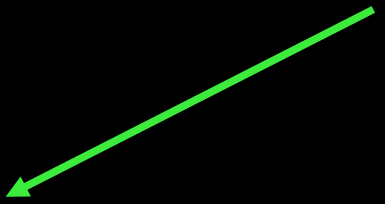
- *Interface* vs. *Implementation*
- *Interface*: How the data “talks to” the outside world, a.k.a. *client*
- *Implementation*: How the data is represented internally
- Data types like this are said to be *abstract*.
- Like Java...

```
public interface Message {

   public void report(String msg);

}
public class StdoutMessage implements Message {

   public void report(String msg) {

      System.out.println(msg);

   }

}
```

An Example from Scheme: Nonnegative Integers

Semantics:

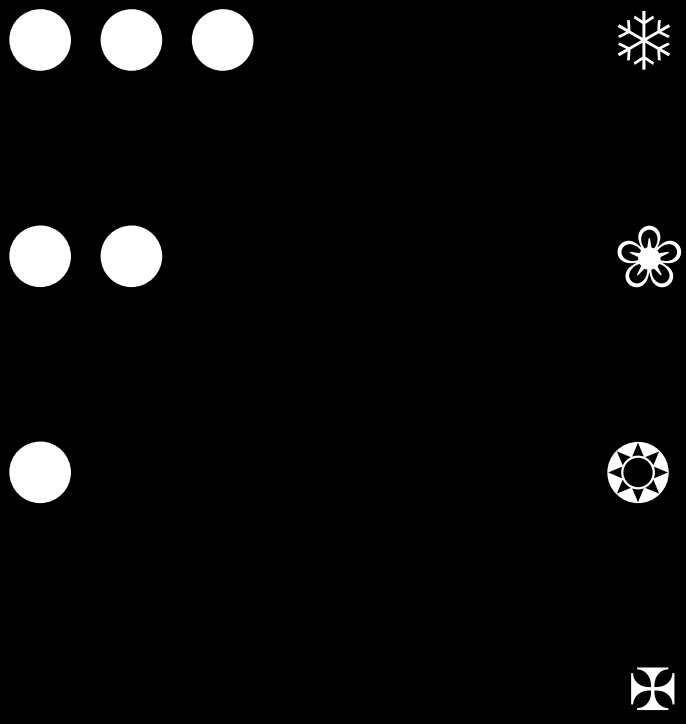
"The representation of n"



```
(iszero? [n]) = #t   n = 0
                #f   n ≠ 0

(succ [n])    = [n+1] (n ≥ 0)
(pred [n+1])  = [n]   (n ≥ 0)
```

E.g., *“Silly” Representation*

Items Representation



```
(iszero? ❄) = #f

(iszero? ✠) = #t

(succ ❁) = ❀
```

An Example from Scheme: Nonnegative Integers

Client: implementation-independent

```
(define plus
    (lambda (x y)
        (if (iszero? x)
            y
            (succ (plus (pred x) y)))))
```

Nonnegative Integers: Unary implementation

```
[0] = ()

[n+1] = (cons #t [n])

(define zero '())

(define iszero? null?)

(define succ (lambda (n) (cons #t n)))

(define pred cdr)
```

Nonnegative Integers: Ordinary implementation

```
(define zero 0 )

(define iszero? zero?)

(define succ (lambda (n) (+ n 1)))

(define pred (lambda (n) (- n 1)))
```