Data Structures in Scheme

- Building data structures: Java

```
class Circle {
   private double center_x, center_y, radius;

    Circle(double x, double y, double r) {
       this.center_x = x;
       this.center_y = y;
       this.radius = r;
   }
};
```

Data Structures in Scheme

- Building data structures: Scheme - use **`list`** in a constructor function:

```
(define make-circle
  (lambda (center-x center-y radius)
    (list center-x center-y radius)))
```

Data Structures in Scheme

- Accessing fields: use **`list-ref`** in a selector function:

```
double getRadius () {     // Java
      return radius;
}

(define get-radius          ; Scheme
  (lambda (circle)
    (list-ref circle 2)))
```

Local declaration/scope via `let`

```
double area() {
  double pi = 3.14159, r = radius;
  return (pi * r * r);
}
```

```
(define area
  (lambda (circle)
    (let ((pi 3.14159)
          (r (get-radius circle)))
      (* pi r r))))
```

Local declaration/scope via `let`

```
(let ( (var1 val1)
       (var2 val2)
       ..
       (varN valN) )

  result )
```

Shortcut with `letrec`

```
> (letrec ((a 3)
           (b (* 2 a))
           (c (+ b 1)))
    c)

7

 > (let ((a 3)) ; long version
     (let ((b (* 2 a)))
       (let ((c (+ b 1)))
         c)))
7
```

Sequential execution via **`begin`**

```
(define prompt-for-command-char
  (lambda (prompt)
    (begin
        (write prompt)
        (read)))
```