



Abbildung 9.1: Klassendiagramm für Redner, Dozenten und Sänger

9.5 Abstraktion über Klassen

Abbildung 9.1 zeigt ein sogenanntes *Klassendiagramm* für die Klassen `speaker`, `lecturer`, `arrogant-lecturer`, `singer` sowie die Objekte `madonna` und `norbert`, die beide sowohl von `singer` als auch von `lecturer` erben. Die Klasse `arrogant-lecturer` steht dabei etwas einsam in der unteren linken Ecke; das Diagramm suggeriert, daß nur Dozenten auch arrogant sein können.

Es ist jedoch genau denkbar, daß Sänger arrogant sind: Die Arroganz bezieht sich schließlich auf eine Veränderung im Verhalten der `say`-Methode von `lecturer`. Sänger haben aber ebenfalls eine `say`-Methode, die sich auf die gleiche Art und Weise verändern ließe. Es wäre schade, wenn es notwendig wäre, die Arroganz für die Anwendung auf die `singer`-Klasse noch einmal von vorn zu programmieren. Die bisherige Version von Arroganz ist leider noch fest auf `lecturer` abonniert:

```

(define make-arrogant-lecturer
  (lambda ()
    (let ((lecturer (make-lecturer)))
      (lambda (message)
        (cond ((eq? message 'say)
              (lambda (self stuff)
                (send lecturer 'say
                          (append '(it is obvious that)
                                  stuff))))
              (else (get-method lecturer message)))))))
  
```

Die Festlegung auf `lecturer` liegt im Aufruf von `make-lecturer`. Zum Glück ist es ein leichtes, darüber zu abstrahieren. Außerdem ist es dann sinnvoll, die `lecturer`-Variable umzubenennen, da sie sich nicht mehr notwendigerweise auf ein `lecturer`-Objekt bezieht:

```

(define make-make-arrogant-someone
  (lambda (make-someone)
    (lambda ()
      (let ((someone (make-someone)))
        (lambda (message)
          (cond ((eq? message 'say)
                (lambda (self stuff)
                  (send someone 'say
                                (append '(it is obvious that)
                                        stuff))))
                (else (get-method someone message)))))))
  
```

```

                stuff)))
    (else (get-method someone message))))))

```

Dabei nimmt `make-make-arrogant-someone` an, daß sein Argument ein Konstruktor für Objekte mit einer `say`-Methode ist. Damit läßt sich die Definition von `make-arrogant-lecturer` durch einen einfachen Aufruf von `make-make-arrogant-someone` ersetzen:

```

(define make-arrogant-lecturer
  (make-make-arrogant-someone make-lecturer))

```

Arrogante Dozenten funktionieren dabei wie bisher:

```

(define dick (make-arrogant-lecturer))
> (send dick 'say '(the sky is blue))
(it is obvious that the sky is blue)
> (send dick 'lecture '(the sky is blue))
(it is obvious that the sky is blue)
(it is obvious that abstraction abstraction abstraction)

```

Zusätzlich zu arroganten Dozenten läßt sich nun auch eine Klasse für arrogante Sänger definieren:

```

(define make-arrogant-singer
  (make-make-arrogant-someone make-singer))

```

Arrogante Sänger verhalten sich jetzt analog zu arroganten Dozenten:

```

(define roy (make-arrogant-singer))
> (send roy 'say '(the sky is blue))
(tra-la-la it is obvious that the sky is blue)

```

Die `make-make-arrogant-someone`-Prozedur verkörpert damit gewissermaßen die Arroganz als eine unabhängige Eigenschaft, die auf alle Klassen mit `say`-Methode anwendbar ist. Eine solche Prozedur, die eine Klasse (bzw. den Konstruktor einer Klasse) als Parameter hat und wieder eine Klasse zurückliefert, die um bestimmte Eigenschaften erweitert ist, heißt *Mixin*.