Objekt-orienteeritud keeled

- Objektipõhised (ingl. *object based*) keeled — objekt on operatsioonide (nn. meetodide) hulk mis jagavad sama olekut; olek on lokaalne objektile ning väljaspoole nähtav ainult läbi meetodide
- Klassipõhised (ingl. *class based*) keeled — klass on olekumuutujate (väljade) ja meetodide kirjeldus; ühte klassi kuuluvad objektid (klassi isendid) omavad samu meetode, kuid erinevat olekut
- Objekt-orienteeritud keeled — klassipõhised keeled, kus on lisaks päritavus (vaatleme järgmine loeng)

Uus süntaks

⟨*exp*⟩::=⟨*instance-var*⟩	
| ⟨*class-var*⟩	
| ⟨*instance-var*⟩ `:=` ⟨*exp*⟩	`i-varassign (var exp)`
| ⟨*class-var*⟩ `:=` ⟨*exp*⟩	`c-varassign (var exp)`
| `method` ⟨*varlist*⟩ ⟨*exp*⟩	`method (formals body)`
| `$`⟨*var*⟩ `(`⟨*exps*⟩`)`	`method-app (name rands)`
| `simpleinstance` ⟨*exp*⟩	`new-simpleinst (class-exp)`
| `simpleclass` ⟨*varlist*⟩	`new-simpleclass (c-vars`
⟨*varlist*⟩ ⟨*methdecls*⟩ ⟨*exp*⟩	`i-vars methdecls init-exp)`

Uus süntaks (järg)

⟨*instance-var*⟩	::=	`&`⟨*var*⟩	`i-varref (var)`
⟨*class-var*⟩	::=	`&&`⟨*var*⟩	`c-varref (var)`
⟨*methdecls*⟩	::=	`()`	
	|	`(`⟨*decls*⟩`)`	
⟨*decls*⟩	::=	⟨*decl*⟩ {`;` ⟨*decl*⟩}*	
⟨*decl*⟩	::=	⟨*var*⟩ `=` ⟨*exp*⟩	`decl (var exp)`

Näide: magasin

```
define stackclass =
  simpleclass (pushed) (stk, localpushed)
   (initialize = method () begin
                            &localpushed := 0;
                            &stk := emptylist
                           end;
    empty = method () null(&stk);
    push = method (x) begin
                        &&pushed := +(&&pushed, 1);
                        &localpushed := +(&localpushed, 1);
                        &stk := cons(x, &stk)
                       end;
    ...
```

Näide: magasin (järg)

```
...
pop = method ()
        if $empty(self) then error()
        else begin
                &&pushed := -(&&pushed, 1);
                &localpushed := -(&localpushed, 1);
                &stk := cdr(&stk)
             end;
top = method ()
        if $empty(self) then error() else car(&stk);
pushed = method () &&pushed;
localpushed = method () &localpushed)
&&pushed := 0
```

Näide: magasin (järg)

```
--> "define stack1 = simpleinstance(stackclass)"
--> "define stack2 = simpleinstance(stackclass)"
--> "begin $push(stack1, 7); $push(stack2, 6);
          $push(stack2, 8) end"
--> "$top(stack1)"
7
--> "$top(stack2)"
8
--> "cons ($localpushed(stack1), $localpushed(stack2))"
(1 . 2)
--> "cons ($pushed(stack1), $pushed(stack2))"
(3 . 3)
```

Interpretaator

- Meil on kolme liiki muutujaid — tavalised muutujad (`x`) on seotud keskonnas; isendimuutujad (`&x`) ja klassimuutujad (`&&x`) on seotud vastavas isendit või klassi esitavas kirjes
- Klass koosneb klassimuutujatest ning nende väärtustest, isendimuutujateat ning keskonnast, mis seob meetodide nimed (mis on tavalised muutujad) meetodiväärtsutega:

```
(define-record class (c-vars c-vals i-vars m-env))
```

- Isend koosneb klassist ning isendimuutujatest:

```
(define-record instance (class i-vals))
```

Interpretaator

- Avaldiste väärtustamisel koosneb kontekst lisaks keskonnale ka “hetkel kehtivast” klassist ja isendist

```
(define eval-exp
  (lambda (exp env class inst)
    (variant-case exp
       ...
    )))
```

- Klassi- ja isendimuutujate väärtustamine:

```
(i-varref (var) (lookup var (class->i-vars class)
                            (instance->i-vals inst)))
(c-varref (var) (lookup var (class->c-vars class)
                            (class->c-vals class)))
```

Interpretaator

```
(define lookup
  (lambda (var vars vals)
    (cell-ref (cell-lookup var vars vals))))

(define cell-lookup
  (lambda (var vars vals)
    (letrec ((loop (lambda (vars c)
                     (cond
                      ((null? vars)
                            (error "Unassigned variable:" var))
                      ((eq? (car vars) var) (vector-ref vals c))
                      (else (loop (cdr vars) (- c 1)))))))
      (loop vars (- (length vars) 1)))))
```

Interpretaator

```
(define assign
  (lambda (var value vars vals)
    (cell-set! (cell-lookup var vars vals) value)))
```

Isendi/klassimuutujatele omistamise väärtustamine:

```
  (i-varassign (var exp)
    (let ((value (eval-exp exp env class inst)))
      (assign var value (class->i-vars class)
                        (instance->i-vals inst))))
  (c-varassign (var exp)
    (let ((value (eval-exp exp env class inst)))
      (assign var value (class->c-vars class)
                        (class->c-vals class))))
```

Meetodid

- Meetodid on sarnased protseduuridega, aga kuuluvad mingisse kindlasse klassi
- Meetodi defineerimise hetkel pole seda klassi, kuhu ta kuulub, veel loodud!
  - meetod võib olla defineeritud väljaspool klassidefinitsiooni;
  - ka klassidefinitsiooni sees defineeritud meetodi korral, on klassi konstrueerimine alles pooleli
- Olukord on analoogne rekursiivsete protseduuridega

Meetodid

Meetodidefinitsioonide väärtustamine:

```
(method (formals body)
  (let ((new-formals (cons 'self formals)))
    (lambda (class-thunk)
      (lambda (args)
        (eval-exp body
          (extend-env new-formals
                      (map make-cell args) env)
          (class-thunk)
          (car args))))))
```

Meetodid

Meetodiväljakutsete väärtustamine:

```
(meth-app (name rands)
  (let ((args (map (lambda (rand)
                     (eval-exp rand env class inst))
                   rands)))
    (meth-call name
               (instance->class (car args))
               args)))
```

Meetodid

Meetodiväljakutsete väärtustamine (järg):

```
(define meth-call
  (lambda (name class args)
    (let ((method (meth-lookup name class)))
      (method args))))

(define meth-lookup
  (lambda (name class)
    (apply-env (class->m-env class) name)))
```

Klassi instantsieerimine

Uue isendi loomiseks tuleb leida vastav klassimall,
reserveerida mälu ning initsialiseerida isendimuutujad:

```
(new-simpleinst (class-exp)
  (let ((inst-class (eval-exp class-exp env class inst)))
    (let ((new-inst
            (make-instance inst-class
              (make-vals (class->i-vars inst-class)))))
      (meth-call 'initialize inst-class (list new-inst))
      new-inst)))

(define make-vals
  (lambda (vars)
    (list->vector (map (lambda (x) (make-cell '*)) vars))))
```

Klasside deklareerimine

- Klassi deklareerimisel:
  - luukse avatud meetodid (so. väärtustatakse meetodide definitsioonid)
  - luuakse klass ning meetodid suletakse (so. fikseeritakse klass, kuhu meetod kuulub)
  - initsialiseeritakse klassimuutujad

```
(new-simpleclass (c-vars i-vars methdecls init-exp)
  (let ((open-methods ...))
    (letrec ((new-class ...))
      (eval-exp ...)
      new-class)))
```

Klasside deklareerimine

Avatud meetodide loomine:

```
(let ((open-methods
        (map (lambda (decl)
               (eval-exp (decl->exp decl)
                         env class inst))
             methdecls)))
```

Klasside deklareerimine

Klassi loomine ja meetodide sulgemine:

```
(letrec ((new-class
          (make-class c-vars (make-vals c-vars) i-vars
           (extend-env (map decl->var methdecls)
              (map (lambda (open-meth)
                     (open-meth (lambda () new-class)))
                   open-methods)
              init-meth-env))))
```

Klasside deklareerimine

Klassimuutujate initsialiseerimine:

```
(eval-exp init-exp env new-class
          (make-instance new-class '#()))
```

Meetodide algkeskond:

```
(define init-meth-env
  (extend-env
    '(initialize)
    (list (lambda (args) "Not initialized"))
    the-empty-env))
```

Järgmiseks korraks

- Lugeda läbi EOPL ptk. 7.1
- “Mängida” interpretaatoriga:
  - fail `loeng15.ss`
  - globaalsete definitsioonide jaoks kasutada esimese koduülesande lahendust!