Pärimine

- Eesmärk — koodi korduvkasutamine sarnase funktsionaalsusega klasside korral
- Delegeerimine — “koodi jagamine” objektide vahel; dünaamiline ja paindlik; millal ja kuidas delegeerida võib sõltuda süsteemi olekust
- Pärimine — “koodi jagamine” klasside vahel; staatiline ja efektiivne
  - ühene pärimine
  - mitmene pärimine

Delegeerimine

Näide — piiratud mahutavusega magasin

```
define boundedstackclass = simpleclass () (bound, stack)
      (initialize = method ()
                    begin
                      &bound := 10;
                      &stack := simpleinstance(stackclass)
                    end;
       setbound = method (x) &bound := x;
       push = method (x)
                if less($localpushed(&stack), &bound)
                then $push(&stack, x)
                else error();
       ...
```

Delegeerimine

Näide — piiratud mahutavusega magasin (järg)

```
  ...
  empty = method () $empty(&stack);
  pop = method () $pop(&stack);
  top = method () $top(&stack);
  pushed = method () $pushed(&stack);
  localpushed = method () $localpushed(&stack)
)
0
```

Pärimine

- Uus süntaks — BNF:

$$
\begin{array}{rcl}
\langle exp \rangle & := & \texttt{class}\ \langle exp \rangle, \langle varlist \rangle\ \langle varlist \rangle \\
 & & \qquad \langle methdecls \rangle\ \langle exp \rangle \\
 & | & \texttt{\$}\langle var \rangle\ \texttt{(}\langle super\text{-}rands \rangle\texttt{)} \\
\langle super\text{-}rands \rangle & ::= & \texttt{(super)} \\
 & | & \texttt{(super,}\ \langle exps \rangle\texttt{)}
\end{array}
$$

- Abstraktne süntaks:

```
(define-record new-class (parent-exp c-vars i-vars
                                    methdecls init-exp))
(define-record super-meth-app (name rands))
```

Näide: magasin

```
define stackclass =
  class baseobject, (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 boundedstackclass =
  class stackclass, () (bound)
    (initialize = method () begin
                              &bound := 10;
                              $initialize(super)
                             end;
     push = method (x)
              if less(&localpushed, &bound)
              then $push(super, x)
              else error();
     setbound = method (x) &bound := x
    )
  noop
```

Klasside deklareerimine

```
(new-class (parent-exp c-vars i-vars methdecls init-exp)
  (let ((parent-class
         (eval-exp parent-exp env class inst))
        (open-methods
         (map (lambda (decl)
                (eval-exp (decl->exp decl)
                          env class inst))
              methdecls)))
    (let ((new-c-vars (append c-vars
                              (class->c-vars parent-class)))
          (new-i-vars (append i-vars
                              (class->i-vars parent-class)))
          ...
```

Klasside deklareerimine

```
    ...
    (let ((new-c-vars ...)
          (new-i-vars ...)
          (new-c-vals (make-shared-c-vals
                                (class->c-vals parent-class)
                                c-vars)))
    ...

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

Klasside deklareerimine

```
...
(letrec ((new-class
          (make-class parent-class new-c-vars
                      new-c-vals   new-i-vars
            (extend-env (map decl->var methdecls)
              (map (lambda (open-method)
                     (open-method (lambda () new-class)))
                   open-methods)
              (class->m-env parent-class)))))
     (eval-exp init-exp env new-class
               (make-instance new-class '#()))
     new-class))))
```

Ülemklassi meetodide väljakutsumine

```
(super-meth-app (name rands)
  (let ((args (map (lambda (x)
                     (eval-exp x env class inst))
                   rands)))
    (meth-call name (class->parent class)
               (cons inst args))))
```

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

Algkeskond

```
(define init-env
  (extend-env '(baseobject parentof classof)
    (map make-cell
         (list (make-class '* '() '#() '() init-meth-env)
               (make-prim-proc 'class->parent)
               (make-prim-proc 'instance->class)))
    base-env))
```

Skoopimine

Leksiline skoopimine vs. isendi ja klassi muutujad:

```
define aclass = class baseobject, (cv) (iv) () 0

let x = 3; cv = 4 in
let bclass = class aclass, () ()
               (initialize = method () +(x, &&cv);
                getiv = method () &iv)
               0
in ...
```

Skoopimine

Staatiline vs. dünaamiline pärimine

```
define aclass = class baseobject, (cv) (iv)
              (initialize = method () &iv := 8;
               getcv = method () &&cv;
               getiv = method () &iv)
              &&cv := 6
define bclass = class aclass, (cv) (iv)
              (initialize = method ()
                   begin
                     $initialize(super);
                     &iv := 9
                   end)
              &&cv := 7
```

Skoopimine

Staatiline vs. dünaamiline pärimine (järg)

```
--> "define x = simpleinstance bclass"
--> "define c = $getcv(x)"
--> "define i = $getiv(x)"
```

Staatilise pärimise korral:

```
--> "cons (c, i)"
(6 . 8)
```

Dünaamilise pärimise korral:

```
--> "cons (c, i)"
(7 . 9)
```

Dünaamiline pärimine

Isendimuutujate dünaamiline pärimine:

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

Dünaamiline pärimine

Klassimuutujate dünaamiline pärimine:

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

Järgmiseks korraks

- Lugeda läbi EOPL ptk. 7.2
- “Mängida” interpretaatoriga:
  - `loeng16-1.ss` (pärimine staatiline)
  - `loeng16-2.ss` (pärimine dünaamiline)
  - globaalsete definitsioonide jaoks kasutada esimese koduülesande lahendust!