Metaklassid

- 2-tasemelistes OO-keeltes on kõik objektid klasside isendid, aga klassid pole programmselt manipuleeritavad (näit. C++)
- Metaklass on klass, mille isenditeks on klassid
  - näit. Javas klass `Class`
- 3-tasemelistes OO-keeltes on kõik objektid klasside isendid, kõik klassid on metaklassi isendid; metaklass on klass ja seega iseenda isend

Metaklassid

- Smalltalk'is on 5 taset — näiteks klass `Integer` kuulub metaklassi `Integer class`; kõik metaklassid kuuluvad (metameta)klassi `Metaclass`, mis on ainuke isend (metametameta)klassile `Metaclass class`, mis ise kuulub klassi `Metaclass`.
- Self'is on ainult 1 tase — kõik objektid on samaaegselt vaadeldavad klassidena ning kõik klassid objektidena; võimaldab esiatada potensiaalselt lõpmatu palju metatasemeid

Metaklassid

- Eemaldame süntaksist `class` ja `simpleinstance` reeglid
- Isendite (= klasside = metaklasside = . . .) deklareerimiseks lisame uue reegli:

  ⟨*exp*⟩ := `instance` ⟨*exp*⟩, ⟨*exp*⟩, ⟨*varlist*⟩ ⟨*methdecls*⟩

- Abstraktne süntaks:

```
(define-record new-instance
        (class-exp parent-exp i-vars methdecls))
```

Metaklassidega interpretaator

- “Isendite” esitamine:

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

- “Klassikirjelduste” selektorid:

```
(define class->m-env instance->m-env)
(define class->i-vars instance->i-vars)
(define class->parent instance->parent)
```

“Isendite” deklareerimine

```
(new-instance (class-exp parent-exp i-vars methdecls)
  (let ((inst-class (eval-exp class-exp env class inst))
        (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-i-vars
            (append i-vars (class->i-vars parent-class))))
      ...
```

“Isendite” deklareerimine

```
...
(letrec ((new-inst
          (make-instance inst-class parent-class
            new-i-vars
            (extend-env (map decl->var methdecls)
              (map (lambda (open-method)
                     (open-method (lambda () new-inst)))
                open-methods)
              (class->m-env parent-class))
            (make-vals (class->i-vars inst-class)))))
  (meth-call 'initialize inst-class (list new-inst))
  new-inst))))
```

“Klassimuutujad”

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

Algkeskond

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

Näide

```
define metametacountclass =
  instance baseobject, baseobject, (classcount)
    (initialize = method () &classcount := 0)

define metacountclass =
  instance metametacountclass, baseobject, (instcount)
    (initialize = method ()
                    begin
                      &instcount := 0;
                      &&classcount := +(&&classcount, 1)
                    end;
     new = method () instance self, baseobject, () ();
     howmany = method () &&classcount)
```

Näide (järg)

```
define countclass =
  instance metacountclass, baseobject, ()
    (initialize = method ()
                    &&instcount := +(&&instcount, 1);
     howmany = method () &&instcount)

define metastackclass =
  instance classof(metacountclass), metacountclass, (pushed)
    (initialize = method ()
                  begin
                     &pushed := 0;
                     $initialize(super)
                  end)
```

Näide (järg)

```
define stackclass =
  instance metastackclass, countclass, (stk, localpushed)
    (initialize = method () begin
                    &localpushed := 0;
                    &stk := emptylist;
                    $initialize(super)
                  end;
     ...)
```

Näide (järg)

```
define boundedstackclass =
  instance classof(stackclass), stackclass, (bound)
    (initialize = method () begin
                    &bound := 10;
                    $initialize(super)
                  end;
     ...)
```

Näide (järg)

```
--> "define stackinstance = $new(stackclass)"
--> "define boundedstackinstance = $new(boundedstackclass)"
--> "define boundedstackinstance2 = $new(boundedstackclass)"
--> "$push(stackinstance, 11)"
--> "$push(stackinstance, 12)"
--> "$push(boundedstackinstance, 13)"
--> "$push(boundedstackinstance2, 14)"
--> "$push(boundedstackinstance2, 15)"
--> "$pushed(stackinstance)"
2
--> "$pushed(boundedstackinstance)"
3
```

Järgmiseks korraks

- Lugeda läbi EOPL ptk. 7.3, 7.4
- “Mängida” interpretaatoriga:
  - `loeng17.ss`
  - globaalsete definitsioonide jaoks kasutada esimese koduülesande lahendust!