objects2.fs

Ein modernisiertes objektorientiertes Paket

M. Anton Ertl
TU Wien

OOP? Wozu?

```
circle heap-new constant circle1
square heap-new constant square1

create a circle1 , square1 ,

: foo ( addr u -- )
   0 do dup i cells + @ draw loop drop ;

a 2 foo
```

Polymorphismus!

Eigenschaften von objects.fs

- Normale Forth-Zellen repräsentieren Objekte
- Objekte auf dem Stack, Return Stack, Variablen ...
- Einfaches und effizientes late binding
- Kein Parsing oder STATE-smartness
- Versucht nicht, alles zu können
- Implementation in Standard-Forth

Neu: Selektor für alle Klassen

```
object class
  selector foo
end-class X
```

```
X class
  m: ... ;m
  overrides foo
end-class Y
```

```
object class
  m: ... ;m
  method2 foo
end-class Y
```

```
X class
  m: ... ;m
  overrides foo
end-class Z
```

```
object class
  m: ... ;m
  method2 foo
end-class Z
```

Neu: Fehlermeldung bei falscher Klasse

```
object class
    selector old
    m: ;m method2 new
end-class foo

object class
end-class bar

bar heap-new constant bar1

bar1 old \ Eingabe
:1: Invalid memory address
bar1 >>>old<<<

bar1 new \ Eingabe
:2: no method defined for this object/selector combination
bar1 >>>new<<<
```

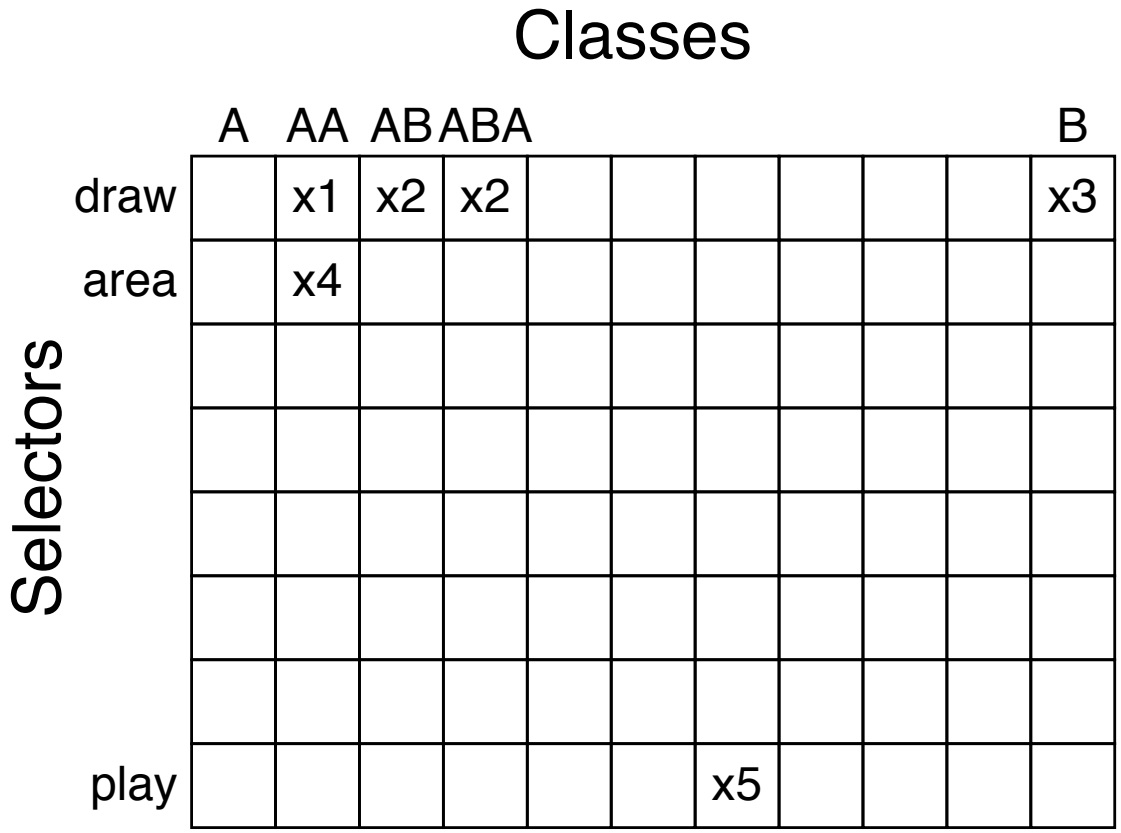
Neu: Umleitung unbekannter Aufrufe

```
object class
m: ( object -- )
    cr ." m1-aba "
    this print ;m
method2 m1
end-class ABA
ABA heap-new constant aba1

object class
m: ( ... sel-xt object -- )
    cr ." redirecting ..."
    aba1 swap execute ;m
overrides message-not-understood
end-class B
B heap-new constant b1

b1 m1
\ Ausgabe:
redirecting ...
m1-ab object:28709920 class:140313329184968
```

Implementation



Implementation

```
: do-method2 ( -- )
  does> ( ... object -- ... )
    ( object selector-addr ) @ over object-map @ map-id @ +
    ( object id ) tuck hash-multiplier um* +
    ( id object hash ) table-mask and 2* cells method2-table +
    ( id object table-entry ) rot begin ( object table-entry id )
over @ over = if \ right class and selector?
    drop cell+ perform exit then
over @ 0= if
    nip message-not-understood1 exit then
cell+ cell+
    again ;

: do-class-method ( -- )
does> ( ... object -- ... )
    ( object selector-body )
    selector-offset @ over object-map @ + ( object xtp ) perform ;
```

Native code auf VFX

```
( 080BA9E3) LEA  EBP, [EBP+-04]
( 080BA9E6) MOV  [EBP], EBX
( 080BA9E9) POP  EBX
( 080BA9EA) MOV  EDX, [EBP]
( 080BA9ED) MOV  EDX, 0 [EDX]
( 080BA9EF) MOV  ECX, [EDX+04]
( 080BA9F2) ADD  ECX, 0 [EBX]
( 080BA9F4) MOV  EAX, 9E3779B1
( 080BA9F9) MUL  ECX
( 080BA9FB) ADD  EDX, EAX
( 080BA9FD) AND  EDX, 0003FFFF
( 080BAA03) SHL  EDX, 1
( 080BAA05) SHL  EDX, 02
( 080BAA08) ADD  EDX, 080BA960
( 080BAA0E) LEA  EBP, [EBP+-04]
( 080BAA11) MOV  EBX, ECX
( 080BAA13) MOV  [EBP], EDX
( 080BAA16) NOP
( 080BAA17) NOP
```

```
( 080BAA18) MOV  EDX, [EBP]
( 080BAA1B) MOV  EDX, 0 [EDX]
( 080BAA1D) CMP  EDX, EBX
( 080BAA1F) JNZ  080BAA34
( 080BAA25) MOV  EBX, [EBP]
( 080BAA28) MOV  EDX, [EBX+04]
( 080BAA2B) MOV  EBX, [EBP+04]
( 080BAA2E) LEA  EBP, [EBP+08]
( 080BAA31) CALL EDX
( 080BAA33) NEXT,
( 080BAA34) MOV  EDX, [EBP]
( 080BAA37) MOV  EDX, 0 [EDX]
( 080BAA39) TEST EDX, EDX
( 080BAA3B) JNZ  080BAA4A
( 080BAA41) LEA  EBP, [EBP+04]
( 080BAA44) CALL MESSAGE-NOT-U
( 080BAA49) NEXT,
( 080BAA4A) ADD  EBX, 08
( 080BAA4D) JMP  080BAA18
( 080BAA4F) NEXT,
```

Mögliche Erweiterungen

- Syntax
- Test auf Klassenmitgliedschaft
- Reflection:
  Alle Klassen
  Alle Kinder einer Klasse
  Alle Selektoren

Zusammenfassung

- Essenz von OOP: Polymorphismus
- Neue Features:
  Selektor für alle Klassen
  message-not-understood
- Einigermassen effiziente Implementation