Smalltalk
Lecture 4

Fraction class

```
Number subclass: #MyFraction.
MyFraction instanceVariableNames: 'num den'.
MyFraction extend [
    num [ ^ num ]
    den [ ^ den ]
    num: n den: d [ num := n. den := d ]
]

f1 := MyFraction new num: 10 den: 4.
f2 := MyFraction new num: 15 den: 6.
f3 := MyFraction new num: -10 den: 4.
f4 := MyFraction new num: 15 den: -6.
```

printOn:

```
MyFraction extend [
    printOn: aStream [
        num printOn: aStream.
        aStream nextPut: $/.
        den printOn: aStream.
        aStream cr ]
]
```

```
f1 printOn: Transcript.
f2 printOn: Transcript.
f3 printOn: Transcript.
f4 printOn: Transcript.
```

```
10/4
15/6
-10/4
15/-6
```

+ and -

```
MyFraction extend [
    + f [ ^ MyFraction new num: (num * f den) + (den * f num)
                           den: (den * f den) ]

    - f [ ^ MyFraction new num: (num * f den) - (den * f num)
                           den: (den * f den) ]
]

f5 := f1 + f2.
f6 := f1 - f2.
f5 printOn: Transcript.
f6 printOn: Transcript.
```

```
120/24
0/24
```

* and /

```
MyFraction extend [
    * f [ ^ MyFraction new num: (num * f num)
                              den: (den * f den) ]
    / f [ ^ MyFraction new num: (num * f den)
                              den: (den * f num) ]
]

f7 := f1 * f2.
f8 := f1 / f2.
f7 printOn: Transcript.
f8 printOn: Transcript.
```

```
150/24
60/60
```

= and ~=

```
MyFraction extend [
    = f [ ^ (num * f den) = (den * f num) ]
]
```

```
f1 = f2 printOn: Transcript.  Transcript cr.
f2 = f3 printOn: Transcript.  Transcript cr.
f3 = f4 printOn: Transcript.  Transcript cr.
```

```
true
false
true
```

```
f1 ~= f2 printOn: Transcript.  Transcript cr.
f2 ~= f3 printOn: Transcript.  Transcript cr.
f3 ~= f4 printOn: Transcript.  Transcript cr.
```

```
false
true
false
```

asFloat

```
MyFraction extend [
    asFloat [ ^ num asFloat / den asFloat ]
]
```

```
f1 asFloat printOn: Transcript.  Transcript cr.
f3 asFloat printOn: Transcript.  Transcript cr.
f5 asFloat printOn: Transcript.  Transcript cr.
f7 asFloat printOn: Transcript.  Transcript cr.
```

```
2.5
-2.5
5.0
6.25
```

reduce

```
MyFraction extend [
    reduce [ |g|
        g := num gcd: den.
        den < 0 ifTrue: [ g := 0 - g ].
        num := num / g.
        den := den / g ]
]

f1 reduce.   f2 reduce.   f3 reduce.   f4 reduce.
f5 reduce.   f6 reduce.   f7 reduce.   f8 reduce.
```

reduce

```
f1 printOn: Transcript.
f2 printOn: Transcript.
f3 printOn: Transcript.
f4 printOn: Transcript.
```

```
5/2
5/2
-5/2
-5/2
```

```
f5 printOn: Transcript.
f6 printOn: Transcript.
f7 printOn: Transcript.
f8 printOn: Transcript.
```

```
5/1
0/1
25/4
1/1
```

Always reduce

```
MyFraction extend [
   num: n den: d [ num := n. den := d. self reduce ]
]
```

```
f1 := MyFraction new num: 10 den: 4.
f2 := MyFraction new num: 15 den: 6.
f3 := MyFraction new num: -10 den: 4.
f4 := MyFraction new num: 15 den: -6.

f1 printOn: Transcript.
f2 printOn: Transcript.
f3 printOn: Transcript.
f4 printOn: Transcript.
```

```
5/2
5/2
-5/2
-5/2
```

Always reduce

```
f5 := f1 + f2.
f6 := f1 - f2.
f7 := f1 * f2.
f8 := f1 / f2.
```

```
f5 printOn: Transcript.
f6 printOn: Transcript.
f7 printOn: Transcript.
f8 printOn: Transcript.
```

```
5/1
0/1
25/4
1/1
```

Better = and ~=

```
MyFraction extend [
    = f [ ^ (num = f num) & (den = f num) ]
]
```

```
f1 = f2 printOn: Transcript.  Transcript cr.
f2 = f3 printOn: Transcript.  Transcript cr.
f3 = f4 printOn: Transcript.  Transcript cr.
```

```
true
false
true
```

```
f1 ~= f2 printOn: Transcript.  Transcript cr.
f2 ~= f3 printOn: Transcript.  Transcript cr.
f3 ~= f4 printOn: Transcript.  Transcript cr.
```

```
false
true
false
```

OOP: inheritance and late method binding

```
Object extend: #A.
A extend: #B.
B extend: #C.
C extend: #D.
D extend: #E.

x := Array new: 5.
x at: 1 put: A new;
   at: 2 put: B new;
   at: 3 put: C new;
   at: 4 put: D new;
   at: 5 put: E new.
```

Object and self

```
Object extend [
    m [^ 1]
    n [^ self m]
    output [ Transcript display: self m; space;
                        display: self n; tab ]
]

Object new output.
Transcript cr.
```

```
1 1
```

Subclass A and super

```
A extend [
    p [^ self n]
    q [^ super m]
    output [ super output.
             Transcript display: self p; space;
                        display: self q; cr ]
]

(x at: 1) output.
```

```
1 1       1 1
```

Subclasses B and C

```
B extend [
    m [^ 2]
]
```

```
(x at: 2) output.
```

```
2 2      2 1
```

```
C extend [
    p [^ super q]
    q [^ super p]
]
```

```
(x at: 3) output.
```

```
2 2      1 2
```

Subclasses D and E

```
D extend [
    m [^ 3]
]
```

```
(x at: 4) output.
```

```
3 3        1 3
```

```
E extend [
    n [^ 4]
]
```

```
(x at: 5) output.
```

```
3 4        1 4
```

Iterators to:do: and do:

```
1 to: 5 do: [ :k | y := x at: k.
                   y output ].
```

```
1 1        1 1
2 2        2 1
2 2        1 2
3 3        1 3
3 4        1 4
```

```
x do: [ :each | each output ]
```

```
1 1        1 1
2 2        2 1
2 2        1 2
3 3        1 3
3 4        1 4
```