Numbers

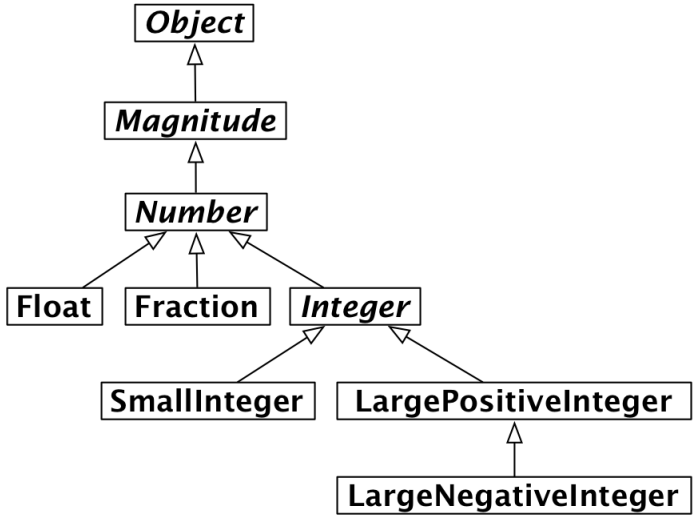
Stéphane Ducasse and Damien Cassou

http://stephane.ducasse.free.fr/ stephane.ducasse@inria.fr damien.cassou@inria.fr

Numbers

- SmallInteger, Integer,
  - 4, 2r100 (4 in base 2), 3r11 (4 in base 3)
- Automatic coercion
  - `1 + 2.3` -> 3.3
  - `1 class` -> SmallInteger
  - `1 class maxVal class` -> SmallInteger
  - `(1 class maxVal + 1) class` -> LargePositiveInteger
- Fraction, Float
  - 3/4, 2.4e7
  - `(1/3) + (2/3)` -> 1
  - `1000 factorial / 999 factorial` -> 1000
  - `2/3 + 1` -> (5/3)

Numbers


Object
Magnitude
Number
Float
Fraction
Integer
SmallInteger
LargePositiveInteger
LargeNegativeInteger

Small ints are real objects

- Small ints are real objects: instance of class `SmallInteger`
- No need for boxing or unboxing

```
1 class
> SmallInteger
```

- Operations on small ints are plain messages (no exception to the rule)
- But small ints are heavily optimized
  - Small integers encoded on 30 bits
  - The pointer itself is the small integer

```
2 raisedTo: 30
> 1073741823

SmallInteger maxVal
> 1073741823
```

Automatic Coercion

What is the largest small integer?

```
1 class maxVal
> 1073741823
```

```
1 class maxVal +1
> 1073741824

(1 class maxVal + 1) class
> LargePositiveInteger
```

Fun With Numbers

1000 factorial

402387260077093773543702433923003985719374864210714632543799904299385123986290205920442084869694048004799886101971960586316668729948085589013238296699445909974245040870737599188236277271887325197795059509952761208749754624970436014182780946464962910563938874378864873371191810458257836478499770124766328898359557354325131853239584630755574091142624174743493475534286465766116677973966688202912073791438537195882498081268678383745597317461360853795345242215865932019280908782973084313928444032812315586110369768013573042161687476096758713483120254785893207671691324484262361314125087802080002616831510273418279777047846358681701643650241536913982812648102130927612448963599287051149649754199093422215668325720808213331861168115536158365469840467089756029009505376164758477284218896796462449451607653534081989013854424879849599533191017233555566021394503997362807501378376153071277619268490343526252000158885351473316117021039681759215109077880193931781141945452572238655414610628921879602238389714760885062768629671466746975629112340824392081601537808898939645182632436716167621791689097799119037540312746222899880051954444142820121873617459926429565817466283029555702990243241531816172104658320367869061172601587835207515162842255402651704833042261439742869330616908979684825901254583271682264580665267699586526822728070757813918581788896522081643483448259932660433676601769996128318607883861502794659551311565520360939881806121385586003014356945272242063446317974605946825731037900840244 32...

Fraction

- denominator , numerator , ...
- Can handle large numbers

```
1000 factorial / 999 factorial
> 1000
```

Fun With Large Numbers

```
1000 factorial numberOfDigits
> 2568
```

```
100000 factorial numberOfDigits
> 456574
```

- Takes some seconds :)

Messages sent to objects

```
(1 / 3) + (2 / 3)
>1
```

```
2 / 3 + 1
> 5 / 3
```

- Numbers are objects
- Mathematical operations are messages sent to objects

Number limits

- There is no notion of precedence

```
2 * 3 + 5
> 11
```

```
2 + 3 * 5
> 25
```

- Use parenthesis to enforce precedence

```
2 + (3 * 5)
> 11
```

Points

- Points are created using the message `@`

```
10 @ 100
(10 @ 100) x
> 10
(10 @ 100) y
>100
```

- Written in a functional way: Most `Point` methods are returning new points

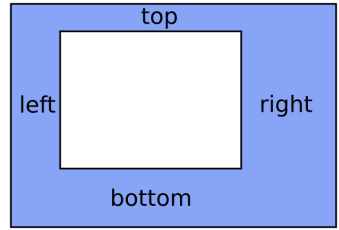
Rectangle

Rectangle left: 0 right: 2 top: 1 bottom: 1

Rectangle origin: 20@20 corner: 100@200

Margin

- Delta to apply to a rectangle
- Useful for GUI
- Encodes 4 numbers, expressed in 3 different ways
  - a number (same space all around)
  - two numbers (same left/right and top/bottom)
  - four numbers
- Used to compute the extended (or inner) rectangle



Summary

- Automatic coercion
- Numbers are real objects
- Number can be infinitely large