Administrative Issues

• HW3 (due at March 18th)
• Essay (due at May 1st)
  – ABET requirement
  – you must get a passing score (37.5), otherwise you will receive Incomplete (I) for this course
• MP2 (due at April 3rd)
  – released at March 20th

About method arguments

\[
\text{A>>f: x}
\]
\[
x := ...
\]

Assignment to argument is not allowed.

Why?

• If you want to change x, send a message to x
• If you just want to refer to another object, why don’t use a different time?

Object (Instance), Class, Inheritance

ProtoObject

Object

Magnitude

Number class

Integer class

SmallInteger class

Number

Integer

SmallInteger

Metaclass

Metaclass class

inheritance

instance of

Examples

“print it” and “inspect it”

• 3
• 3 class
• 3 class class
• 3 class class class
• 3 class class class class
• 3 class class class class class
• SmallInteger superclass
Be wary of the confusing names in the following several slides!!!

**Instance Variables**

- **instance variable:** the variables for instances
  - defined for the instances of a class
  - click "instance" in system browser
  - Inherited in subclass (cannot be redefined in subclass)
  - each instance has its own copy

  Example: Number subclass: #Fraction
  instanceVariableNames: 'numerator denominator'
  classVariableNames: "'
  poolDictionaries: "'
  category: 'Kernel-Numbers'

**Instance Methods**

- **instance methods:**
  - defined for the instances of a class
  - click "instance" in system browser
  - Inherited in subclass (can be overridden in subclass)

  Example: Fraction>>+

**Class Instance Variables**

- **class instance variables:** the instance variable of a class
  (remember a class is an instance of the corresponding metaclass)
  - defined for a class
  - click "class" in system browser
  - Inherited in subclass (cannot be redefined in subclass)
  - The class and each subclass have their own copy

  Example: Very few in the system itself

  Float class
  instanceVariableNames: "' (empty)

**Class Instance Variables (Example)**

Examples:?

**Class Methods**

- **class methods:** It is not called "class instance method"!
  - defined for a class
  - click "class" in system browser
  - Inherited in subclass (can be overridden in subclass)

  Example:
  Fraction class>>numerator: denominator:
  Fraction class>>one
Class Variables

- **Class Variable**: It doesn’t pair with “class method”, which pairs with “class instance variable”
  - defined for both a class and its instances
  - click “instance” in system browser
  - Inherited in a subclass (cannot be redefined in subclass)
  - There is only one copy (shared by the class itself, all the subclasses, and all the instances of the class/subclasses)

Example:
```smalltalk
Number variable

instanceVariableNames: 'Epsilon Help Infinity e+10 e+2 Mutilation MinusLogBase2
Not/Negatitivity NegativeSine2Pi PositivePiDegree Sqrt2 ThreePi TwentyPi'
```

For Accessing Private Data: Setter and Getter

Example:
```smalltalk
class Complex
t

instance variable real, imaginary

• getter
Complex>>real
  "real"

• setter
Complex>>real: aNumber
  real _ aNumber 
  (remember _ means :=)
```

Class Methods for Constructing New Instances

Example:
```smalltalk
class Complex
class methods:
  Complex class>>real: aNumber1 imaginary: aNumber2
  | newComplex |
  newComplex _ super new.
  newComplex
  real: aNumber1;
  imaginary: aNumber2.
  ^
  "newComplex
  Complex class>>new
  "self real: 0 imaginary: 0
```

Inheritance and Handling Messages

- Inheritance:
  - Smalltalk allows only single inheritance

- Method Lookup along the inheritance chain
When an object receives a message:
  - If the class of the object has the method, use it;
  - Otherwise check the superclass, and the superclass of the superclass, and so on.

- Return value of a method
  - Message receiver if no explicit return (i.e., no ^)
Inheritance and Handling Messages

- **Overriding**
  - Multiple classes on inheritance chain may define the same method
  - Only the lowest one (starting from the receiver object) is used
  - Need to say "super methodName" if want to extend the method defined in some superclass (and this is a good practice)
    - E.g., initialize, new, ...

- **self and super**
  - Both `self` and `super` refer to the message receiver itself!
  - "self methodName" will start method looking up from the class of the message receiver.
  - "super methodName" will start method looking up from the parent class of the class that defines the method which sends this message "super methodName".

Example

```plaintext
A>>m2 ...
A subclass: #B ...
B>>m2
super m2
B subclass: #C ...
C>>m1
self m2
C := C new.
C m1
```

What will happen if the lookup of `m2` starts from the superclass of message receiver?

Examples of Super and Self

```plaintext
A B C D
<table>
<thead>
<tr>
<th>test</th>
<th>result1</th>
<th>self test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```plaintext
A B C D
<table>
<thead>
<tr>
<th>test</th>
<th>result2</th>
<th>result1</th>
<th>super test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```plaintext
```

Abstract Method, Abstract Class

- **self subclassResponsibility**
- **Example:**

```plaintext
Number>>+
```

Different types of numbers (Integer, Float, Complex, ... decide the implementation of `+` for themselves)