Example: Expression Hierarchy

- Define general concept `Expression`
- Implement two forms: `Number`, `Sum`
- Methods on implemented types of exprs: `evaluate`, `toString`, `draw`, ...

- Ex:
  ```java
  e = new Sum(new Number(23), new Number(2));
  print e.toString() + " = " + e.evaluate();
  ```

- Anticipate additions to library

abstract class Expr {
  public abstract String toString();
  public abstract int eval();
}

class Number extends Expr {
  private int n;
  public Number(int n) { this.n = n; }
  public String toString() { return "" + n; }
  public int eval() { return n; }
}

class Sum extends Expr {
  private Expr left, right;
  public Sum(...) { ... }
  public String toString() {
    return left.toString() + "+" + right.toString();
  }
  public int eval() { return left.eval() + right.eval(); }
}

Steve Jobs on Touring Xerox PARC

And they showed me really three things. But I was so blinded by the first one I didn’t even really see the other two. One of the things they showed me was object orienting programming - they showed me that but I didn’t even see that. The other one they showed me was a networked computer system...they had over a hundred Alto computers all networked using email etc., etc., I didn’t even see that. I was so blinded by the first thing they showed me which was the graphical user interface... within you know ten minutes it was obvious to me that all computers would work like this some day.
Dynabook
*A Personal Computer for Children of All Ages*, Alan Kay, 1972

Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I
*John McCarthy, 1960

A programming system called LISP (for LI St Processor) has been developed for the IBM 704 computer by the Artificial Intelligence group at M.I.T. ... In this article, we first describe a formalism for defining functions recursively.

Smalltalk: Try It!
*http://squeak.org/

Example: Point Class
*Class definition written in tabular form

<table>
<thead>
<tr>
<th>class name</th>
<th>Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>super class</td>
<td>Object</td>
</tr>
<tr>
<td>class vars</td>
<td>pi</td>
</tr>
<tr>
<td>instance vars</td>
<td>x y</td>
</tr>
</tbody>
</table>
| class messages and methods | ...
| instance messages and methods | ...
| constructors |

Smalltalk: Try It!

```plaintext
Example: Point Class

Class definition written in tabular form

<table>
<thead>
<tr>
<th>class name</th>
<th>Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>super class</td>
<td>Object</td>
</tr>
<tr>
<td>class vars</td>
<td>pi</td>
</tr>
<tr>
<td>instance vars</td>
<td>x y</td>
</tr>
</tbody>
</table>
| class messages and methods | ...
| instance messages and methods | ...
| constructors |

Instance Messages and Methods

<table>
<thead>
<tr>
<th>Instance methods</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>moveDx: dx Dy: dy</td>
<td>pt.moveDx: 1 Dy: 1</td>
</tr>
<tr>
<td>x &lt;- dx + x</td>
<td></td>
</tr>
<tr>
<td>y &lt;- dy + y</td>
<td></td>
</tr>
</tbody>
</table>

In Java:
```java
void moveDxDy(int dx, int dy) {
    x = x + dx;
    y = y + dy;
}
```

Instance Messages and Methods

```plaintext
Instance Methods

<table>
<thead>
<tr>
<th>Instance methods</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>moveDx: dx Dy: dy</td>
<td>pt.moveDx: 1 Dy: 1</td>
</tr>
<tr>
<td>x &lt;- dx + x</td>
<td></td>
</tr>
<tr>
<td>y &lt;- dy + y</td>
<td></td>
</tr>
</tbody>
</table>

x: xcoord y: ycoord | pt x:3 y:2 |
| x <- xcoord        |           |
| y <- ycoord        |           |

void xy(int xcoord, int ycoord) {
    x = xcoord;
    y = ycoord;
}
```
**Instance Messages and Methods**

**Instance methods**
- `moveDx: dx Dy: dy |
  x <- dx + x
  y <- dy + y
- `x: xcoord |
  y <- ycoord
- `x | | `x
  y | | `y
- draw |
  {...draw point...}

**Examples**
- `pt moveDx: 1 Dy: 1`
- `pt x: 3 y: 2`
- `x <- pt x + pt y`

**Class Messages and Methods**

**Class methods**
- `newX: xval Y: yval |
  ^ self new x: xval y: yval
- `newOrigin | |
  ^ self new x: 0 y: 0

**Examples**
- `p <- Point newX:3 Y:2`
- `p <- Point newOrigin`
- `p = Point.newOrigin()`

**Class Meta Data**

**Run-time Representation**

- Three primary operations
  - object creation
  - method lookup
  - field lookup

**Inheritance**

- Define colored points from points
  - `class name ColorPoint`
  - `super class Point`
  - `class var color`
  - `class messages and methods
    newKx yvy cov...
    ... code...
  - `instance messages and methods
    color | | ^color
    draw...
    ... code...`
**ColorPoint Methods**

**Instance Methods**

\[
\begin{align*}
x & : x\text{coord} \quad y : y\text{coord} \quad c : \text{col} \\
x & <= x\text{coord} \\
y & <= y\text{coord} \\
color & <= c
\end{align*}
\]

color | | ^color
draw | | ...

**Class Methods**

\[
\begin{align*}
\text{newX: x} & \quad \text{y: y} \quad \text{c: cv} \\
& = \text{self new x:x y:y c:cv}
\end{align*}
\]

\[
\begin{align*}
\text{newOrigin} & \\
& = \text{self newX:0 Y:0 C:red}
\end{align*}
\]

**Run-time Representation**

**Collection Hierarchy**

**Ingalls Test for OO Languages**

- Define a new kind of integer,
- Put your new integers into a rectangle,
- Ask the system to fill in the rectangle, and
- Have it work.