Smalltalk

CSCI 334

Stephen Freund

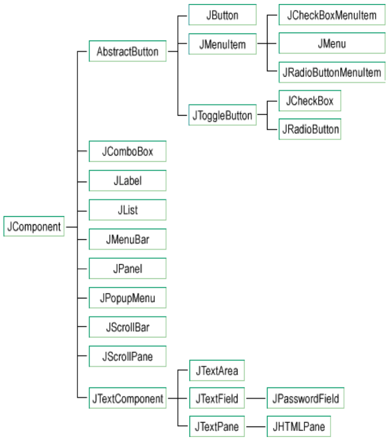
Example: Expression Hierarchy

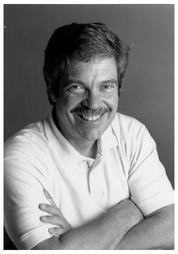
- Define general concept Expression
- Implement two forms: Number, Sum
- Methods on implemented types of exprs
  evaluate, toString, draw, ...
- Ex:

```
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(); }
}
```



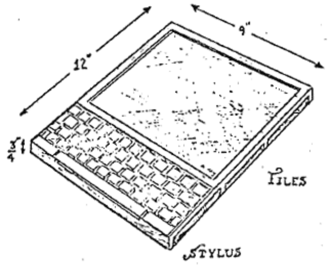


Steve Jobs on Touring Xerox PARC

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 LISt 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

class name	Point
super class	Object
class vars	pi
instance vars	x y
class messages and methods	
〈...names and code for methods...〉	
instance messages and methods	
〈...names and code for methods...〉	

constructors

Instance Messages and Methods

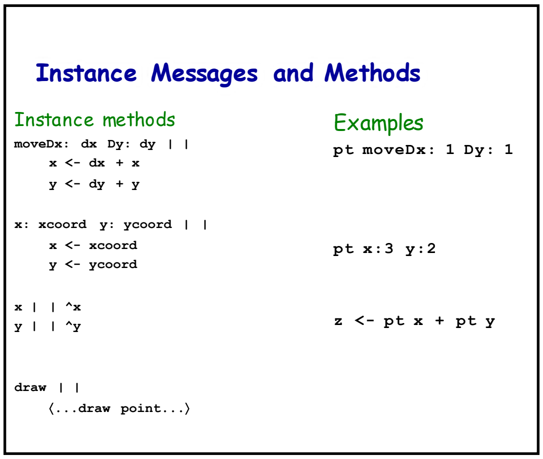
```
Instance methods                  Usage
moveDx: dx Dy: dy | |             pt moveDx: 1 Dy: 1
     x <- dx + x
     y <- dy + y
```

```
In Java:
void moveDxDy(int dx,             pt.moveDxDy(1,1);
              int dy) {
      x = x + dx;
      y = y + dy;
}
```

Instance Messages and Methods

```
Instance methods                  Usage
moveDx: dx Dy: dy | |             pt moveDx: 1 Dy: 1
     x <- dx + x
     y <- dy + y

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

```
void xy(int xcoord,               pt.xy(3,2);
        int ycoord) {
  x = xcoord;
  y = ycoord;
}
```



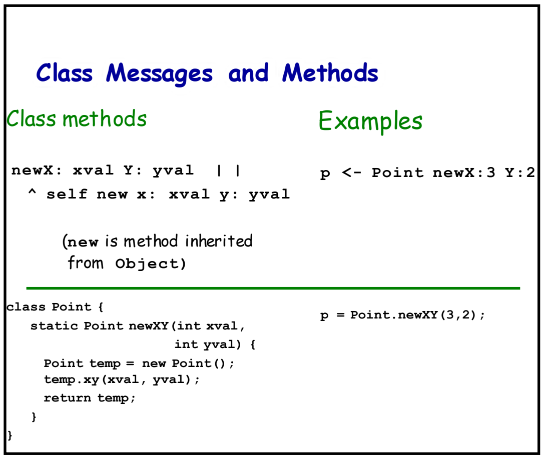
Instance Messages and Methods

Instance methods

```
moveDx:  dx  Dy:  dy  | |
     x <- dx  + x
     y <- dy  + y

x:  xcoord   y:  ycoord  |  |
     x <- xcoord
     y <- ycoord

x  |  | ^x
y  |  | ^y

draw  | |
     〈...draw  point...〉
```

Examples

```
pt moveDx: 1 Dy: 1

pt x:3  y:2

z  <-  pt x  +  pt y
```



Class Messages and Methods

Class methods

```
newX: xval Y: yval   | |
  ^ self new x:  xval y: yval
```

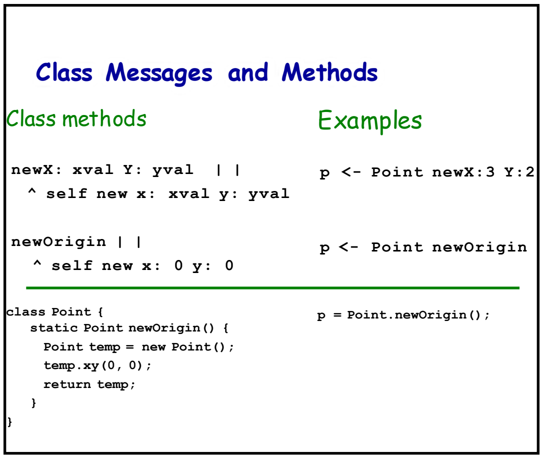
(`new` is method inherited from `Object`)

```
class Point {
   static Point newXY(int xval,
                      int yval) {
     Point temp = new Point();
     temp.xy(xval, yval);
     return temp;
   }
}
```

Examples

```
p  <- Point newX:3 Y:2
```

```
p  = Point.newXY(3,2);
```



Class Messages and Methods

Class methods

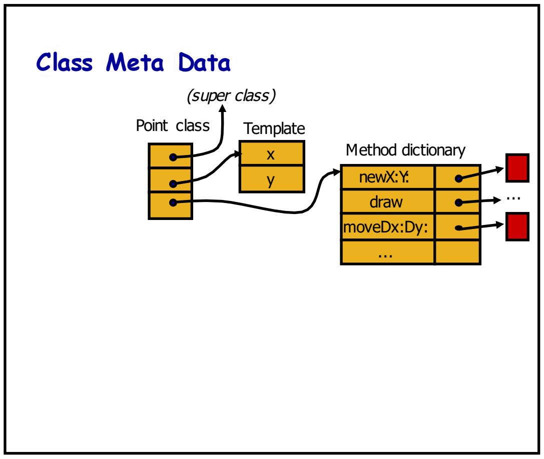
```
newX: xval Y: yval   | |
  ^ self new x:  xval y: yval

newOrigin | |
  ^ self new x:  0  y:  0
```

```
class Point {
   static Point newOrigin() {
     Point temp = new Point();
     temp.xy(0, 0);
     return temp;
   }
}
```

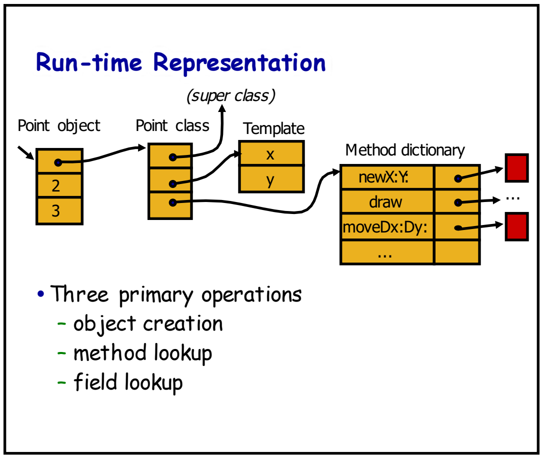
Examples

```
p  <-  Point newX:3 Y:2

p  <-  Point newOrigin
```

```
p  = Point.newOrigin();
```

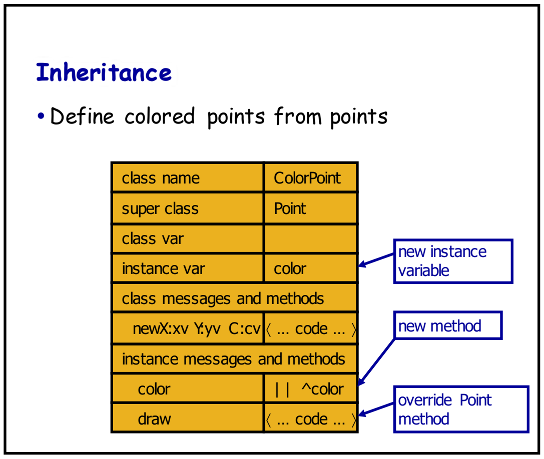


Class Meta Data



Run-time Representation

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



Inheritance

- Define colored points from points

<table>
<tr><td>class name</td><td>ColorPoint</td></tr>
<tr><td>super class</td><td>Point</td></tr>
<tr><td>class var</td><td></td></tr>
<tr><td>instance var</td><td>color</td></tr>
<tr><td colspan="2">class messages and methods</td></tr>
<tr><td>newX:xv Y:yv C:cv</td><td>〈 ... code ... 〉</td></tr>
<tr><td colspan="2">instance messages and methods</td></tr>
<tr><td>color</td><td>| | ^color</td></tr>
<tr><td>draw</td><td>〈 ... code ... 〉</td></tr>
</table>

new instance variable

new method

override Point method

ColorPoint Methods

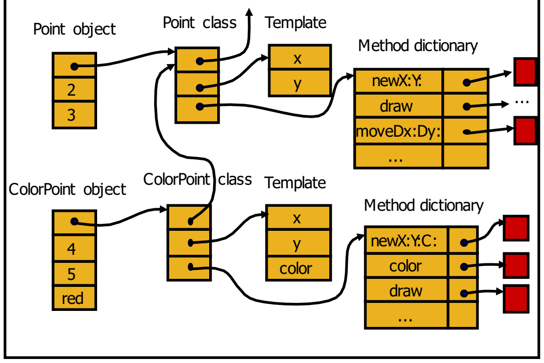
Instance Methods

```
x: xcoord y: ycoord c:col | |
    x <- xcoord
    y <- ycoord
    color <- c

color | | ^color
draw | | ...
```

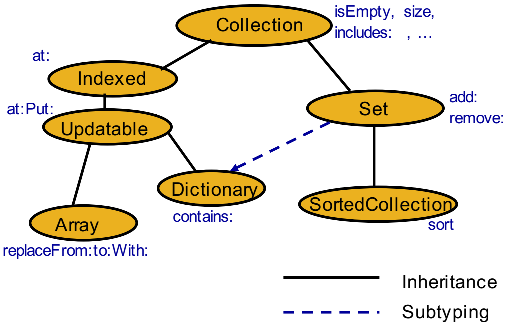
Class Methods

```
newX: xv Y: yv C:cv ||
   ^self new x:xv y:yv c:cv

newOrigin ||
   ^self newX:0 Y:0 C:red
```

Run-time Representation



Collection Hierarchy



Ingalls Test for OO Languages

- In an OO language, you should be able to:
  - 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.