COOL

Cool Overview

- Classroom Object Oriented Language

- Designed to
  - Be implementable in a short time
  - Give a taste of implementation of modern
    - Abstraction
    - Static typing
    - Reuse (inheritance)
    - Memory management

- But many things are left out

A Simple Example

```
class Point {
      x : Int ← 1;
      y : Int ← 2;
};
```

- Cool programs are sets of class definitions
  - class = a collection of attributes and methods
  - instances of a class are objects

- No global variables
- No separate notion of subroutines
  - Entry point is a special class **Main** with a special method **main**

Cool Objects

```
class Point {
        x : Int ← 3;
        y : Int; (* use default value *)
};
```

- The expression “new Point” creates a new instance (i.e. object) of class Point

- An object can be thought of as a record with a slot for each attribute

x	y
3	0

Methods

- A class defines methods for manipulating the attributes

```
class Point {
     x : Int ← 0;
     y : Int ← 0;
     movePoint(newx : Int, newy : Int): Point {
        { x ← newx;
          y ← newy;
          self;
        } -- close block expression
     }; -- close method
}; -- close class
```

- Methods can refer to the current object using self

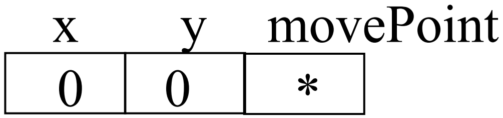
Information Hiding in Cool

- Methods are global
- Attributes are local to a class
  - They can only be accessed by the class's methods
- Example:

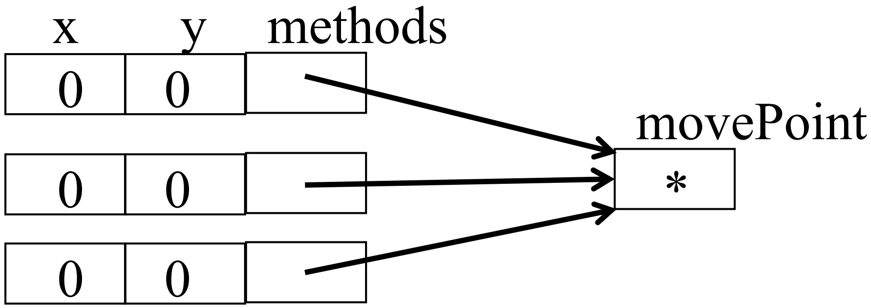
```
class Point {
      . . .
      x () : Int { x };
      setx (newx : Int) : Int { x ← newx };
};
```

Methods

- Each object knows how to access the code of a method
- As if the object contains a slot pointing to the code



- In reality implementations save space by sharing these pointers among instances of the same class



Inheritance

- We can extend functionality of points to colored points using subclassing

```
class ColorPoint inherits Point {
   color : String ← “red”;
   movePoint(newx : Int, newy : Int): Point {
     { color ← “green”;
       x ← newx; y ← newy;
       self;
     }
   };
};
```

x	y	color	movePoint
0	0	red	*

Cool Types

- Every class is a type
- Every* class inherits from exactly one other class
  - Forms a tree of classes (*class hierarchy*)
- Types of all variables must be declared
  - Compiler infers types for expressions

Base Classes

• Object	root of the class hierarchy
• Int	integers
• Bool	boolean values: true, false
• String	character strings
• IO	input/output support

Cool Type Checking

```
x : A;
x ← new B;
```

- Is *well-typed* if A is an ancestor of B in the class hierarchy
  - Anywhere an instance of A is expected an instance of B can be used
- "Well-typed" = satisfies language's type-checking rules
- Type safety:
  - A well-typed program cannot result in runtime type errors

Method Invocation and Inheritance

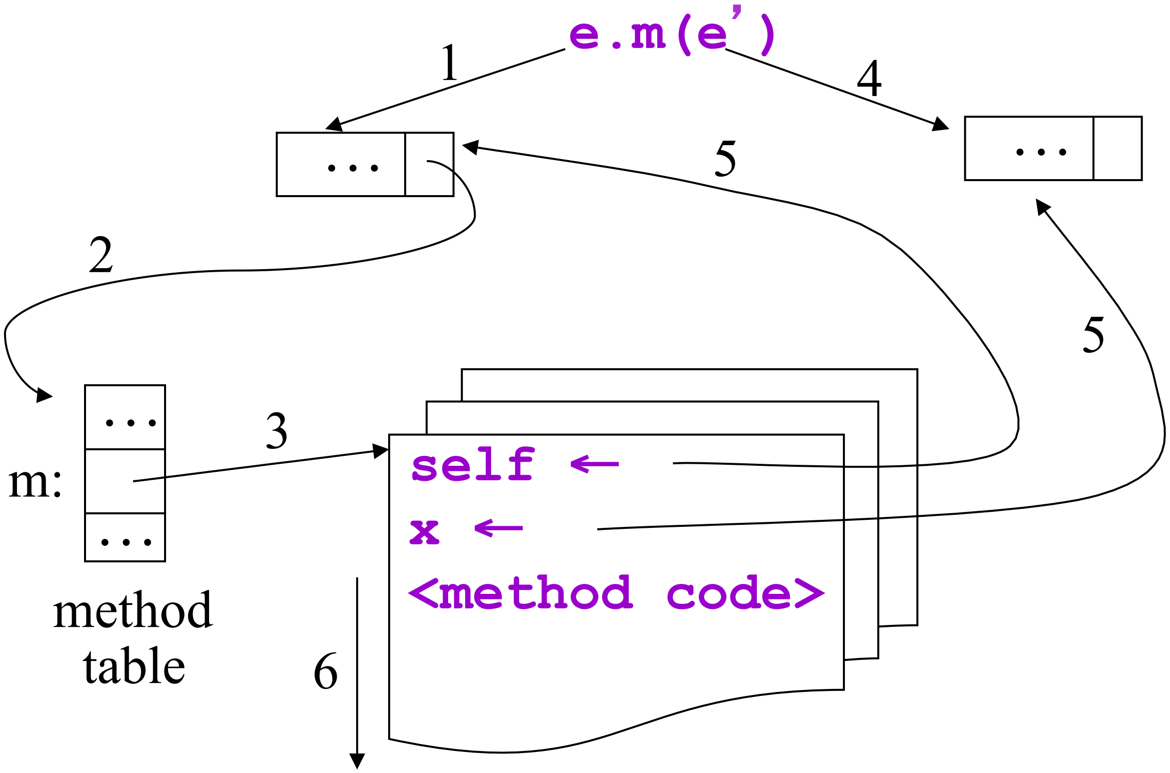
- Methods are invoked by *dispatching* them to the target object
- Understanding dispatch in the presence of inheritance is a subtle aspect of OO languages

```
p : Point;
p ← new ColorPoint;
p.movePoint(1,2);
```

  - p has static type Point
  - p has dynamic type ColorPoint
  - p.movePoint must invoke the ColorPoint version

Method Invocation

- Example: invoke one-argument method m



1. Evaluate e
2. Find class of e
3. Find code of m
4. Evaluate e'
5. Bind self and x
6. Run method

Other Expressions

- Expression language
  - every expression has a type and a value
  - Loops: while E loop E pool
  - Conditionals if E then E else E fi
  - Case statement case E of x : Type ⇒ E; … esac
  - Arithmetic, logical operations
  - Assignment x ← E
  - Primitive I/O out_string(s), in_string(), …
- Missing features:
  - arrays, floating point operations, exceptions, …

Other Expressions (Cont.)

- Blocks
  - { <expr>; ... <expr>; }
- Let
  - let <id1> : <type1> [<- <expr1>], ..., <idn> : <typen> [<- <exprn>] in <expr>

Cool Memory Management

- Memory is allocated every time new is invoked

- Memory is deallocated automatically when an object is not reachable anymore
  - Done by the garbage collector (GC)
  - Garbage collector is part of the Cool runtime