Overview of COOL

Kirill Yukhin

19.09.2018

Lecture Outline

- Cool
- The Course Project
- Programming Assignment 1

Cool Overview

- Classroom Object Oriented Language
- Designed to
  - Be implementable in one semester
  - Give a taste of implementation of modern
    - Abstraction
    - Static typing
    - Reuse (inheritance)
    - Memory management
    - And more …
- But many things are left out

A Simple Example

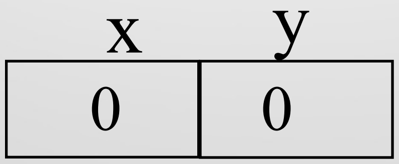
```
class Point {
        x : Int ← 0;
        y : Int ← 0;
};
```

- Cool programs are sets of class definitions
  - A special class **Main** with a special method **main**
  - No separate notion of subroutine
- class = a collection of attributes and methods
- Instances of a class are objects

Cool Objects

```
class Point {
      x : Int ← 0;
      y : Int; (* use default value *)
};
```

- The expression "new Point" creates a new object of class Point
- An object can be thought of as a record with a slot for each attribute



Methods

- A class can also define 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

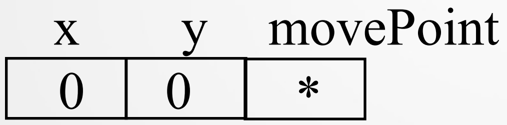
Information Hiding in Cool

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

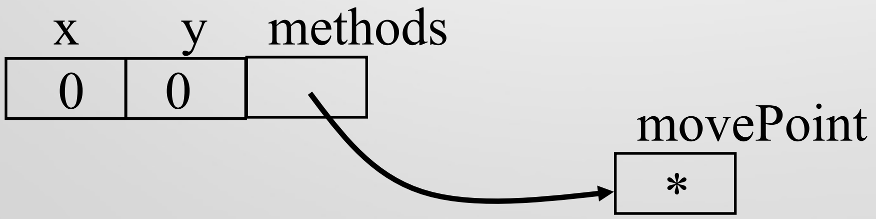
```
class Point {
      . . .
      x () : Int { x };
      setx (newx : Int) : Int { x ← newx };
};
```

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 points to colored points using subclassing => class hierarchy

```
class ColorPoint inherits Point {
    color : Int ← 0;
    movePoint(newx : Int, newy : Int): Point {
      { color ← 0;
        x ← newx; y ← newy;
        self;
      }
    };
};
```

x	y	color	movePoint
0	0	0	*

Cool Types

- Every class is a type
- Base classes:

Int	for integers
Bool	for boolean values: true, false
String	for strings
Object	root of the class hierarchy

- All variables must be declared
  - compiler infers types for expressions

Cool Type Checking

```
x : P;
x ← new C;
```

- Is well typed if P is an ancestor of C in the class hierarchy
  - Anywhere an P is expected a C can be used
- Type safety:
  - A well-typed program cannot result in runtime type errors

Method Invocation and Inheritance

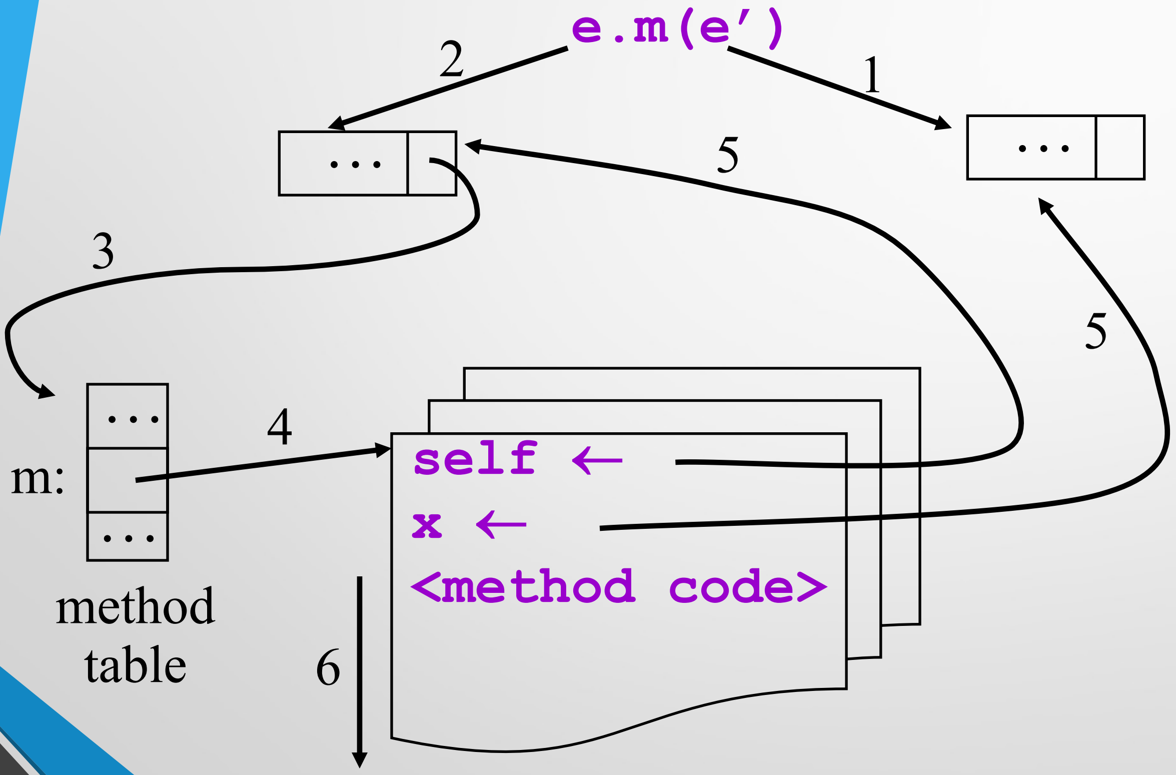
- Methods are invoked by dispatch
- Understanding dispatch in the presence of inheritance is a subtle aspect of OO languages

```
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. Eval. argum e′
2. Eval. e
3. Find class of e
4. Find code of m
5. Bind self and x
6. Run method

Other Expressions

- Expression language (every expression has a type and a value)
  - Conditionals if E then E else E fi
  - Loops: while E loop E pool
  - 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, Interfaces, Exceptions,…

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)
  - There is a Cool GC

Course Project

- A complete compiler
  - Cool ==> MIPS assembly language
  - No optimizations
- Split in few programming assignments (PAs)
- There is adequate time to complete assignments
  - But <u>start early</u> and please follow directions
  - Turn in early to test the turn-in procedure