

# Language Design. Overview of COOL

CS164  
Lecture 2

Prof. Necula CS 164 Lecture 2

1

## Course Administration

---

- If you drop the course, please make it official
- Anyone enrolled can get a card key now
  - Go to CS reception (Soda Hall 3rd floor)
- New anonymous feedback form
- Section 104 (F 3-4) was cancelled
- Attendance to lectures
- Questions about course policies?

Prof. Necula CS 164 Lecture 2

2

## Lecture Outline

---

- Introduction to Cool
- The Course Project

Prof. Necula CS 164 Lecture 2

3

## Cool Overview

---

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

Prof. Necula CS 164 Lecture 2

4

## 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

Prof. Necula CS 164 Lecture 2

5

## 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

x	y
0	0

Prof. Necula CS 164 Lecture 2

6

## 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`

Prof. Necula CS 164 Lecture 2

7

## Information Hiding in Cool

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

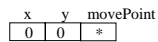
```
class Point {
  . . .
  x () : Int { x };
  setx (newx : Int) : Int { x ← newx };
};
```

Prof. Necula CS 164 Lecture 2

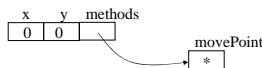
8

## 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



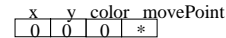
Prof. Necula CS 164 Lecture 2

9

## 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;
    }
  };
};
```



Prof. Necula CS 164 Lecture 2

10

## 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

Prof. Necula CS 164 Lecture 2

11

## 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

Prof. Necula CS 164 Lecture 2

12

## 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);
```

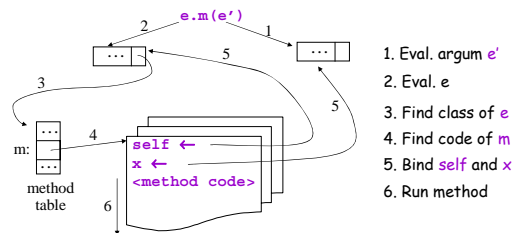
- p has static type `Point`
- p has dynamic type `ColorPoint`
- `p.movePoint` must invoke the `ColorPoint` version

Prof. Necla CS 164 Lecture 2

13

## Method Invocation

- Example: invoke one-argument method `m`



Prof. Necla CS 164 Lecture 2

14

## 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, ...

Prof. Necla CS 164 Lecture 2

15

## 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`

Prof. Necla CS 164 Lecture 2

16

## Course Project

- A complete compiler
  - Cool ==> MIPS assembly language
  - No optimizations
- Split in 5 programming assignments (PAs)
- There is adequate time to complete assignments
  - But start early and please follow directions
  - Turn in early to test the turn-in procedure
- Individual or team (max. 2 students)

Prof. Necla CS 164 Lecture 2

17

## Programming Assignment I

- Write an interpreter for a stack machine ...
- ... in Cool
- Due in 1 week
- Must be completed individually

Prof. Necla CS 164 Lecture 2

18