Cool: Language Overview

Greta Yorsh

Classroom **O**bject **O**riented **L**anguage (Cool)

- Designed by Alex Aiken from Stanford for teaching compiler construction
- Supports modern language features
  - abstraction, reuse (inheritance), static typing, memory management
- Many features are left out
  - feasible to implement in a semester

Good News

- No arrays
- No floating point operations
- No “static” modifier
- No interfaces
- No method overloading
  (but still allow overriding)
- No exceptions
- No packages

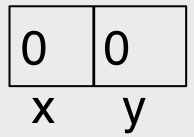
Better News

- Cool language is still rich enough for doing interesting things

Classes and Objects

- Cool programs are sets of class definitions
- A special class **Main** with a special method **main**
- A **class** is a collection of **attributes** and **methods**
- Instance of a class are **objects**
- The expression "**new** Point" creates a new object of class Point

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

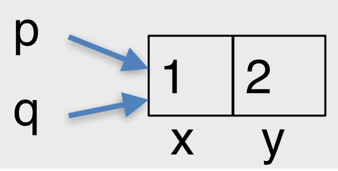


Methods

- Manipulating attributes
- Refer to the current object using **self**

```
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
```

```
class A {
  p : Point;
  q : Point;
  foo() : Int {{
    p <- new Point;
    q <-p.movePoint(1,2);
    1;
  }};
};
```



Information Hiding in Cool

- Methods are global (public)
- Attributes are local to a class (private)
- Attributes can only be accessed by the class's methods

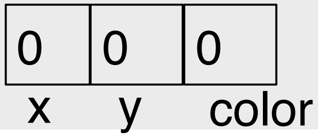
```
class Point {
  x : Int <-0;
  y : Int <- 0;
  getx () : Int { x };
  setx (newx : Int) : Int { x <- newx };
};
```

```
class A {
  p : Point;
  foo() : Int {{
    p.setx(1);     -- ok
    p.x <-1;       -- illegal
  }};
};
```

Inheritance

- We can extend points to colored points using subclass (class hierarchy)

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



Cool Types

- Every class is a type
- Base classes
  - **Int**
  - **Bool** (**true**, **false**)
  - **String**
  - **Object** (root of the class hierarchy)
- All variables must be declared
- Compiler infers types for expressions

Cool Type Checking

- Type safety: a well-typed program cannot result in type errors at runtime

```
x : A;
x <- new B;

(* well typed if A is an ancestor of B in the class hierarchy *)
```

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 has static type Point
p <- new ColorPoint;                --p has dynamic type ColorPoint
p.movePoint(1,2);                   -- invoke  ColorPoint version of movePoint

(* static dispatch: invoke the version of movePoint defined in Point *)
p@Point.movePoint(1,2);
```

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)
  - part of Cool runtime system

Other Expressions

- Every expression has a type and a value
- Expression language
  - Assignment x <- E
  - Loops: while E loop E pool
  - Conditionals: if E then E else E fi
  - Case: case E of x: Type E; esac
  - Let let x : Type <- E in E
  - Arithmetic and logical operations
  - Primitive I/O out_string(s), in_string(), ...