Classroom Object Oriented Language: COOL

S. M. M. Sadrnezhaad, R. Yazdanian, S. Karimi

Design goals

- Taught in compiler course in some universities
- Designed to be implementable in a short time (one semester)
- Give students an idea of many features of OO languages, including:
  - Abstraction
  - Static Typing
  - Inheritance
  - Memory management
  - Simple reflection
- Leaves many features out which are common to other languages

The not so COOL parts

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

The reference compiler

- Built with C++
- Generates code for MIPS simulator named SPIM

Where does execution start?

- A class "Main" must be defined
  - Containing a no-args "main" method
  - Starts program execution

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 operations
  - Logical operations
  - Assignment x ← E

Basic structure and classes

- Basically, a COOL program is a set of classes
  - Similar to Java
- Almost every type is a class (next slide)
- Blocks defined using { and }.
- Each COOL class has a set of features
  - Methods
    - No explicit return
  - Attributes (variables)

Classes and types: continued

- Primitive types: Boolean, Int, String
- Built-in types: The above, + Null, Nothing, Unit, Any
- Every type except Null and Nothing is a class
- Int, Boolean, Unit cannot be superclass
- “Any” class is the root of class hierarchy
  - Similar to “Object” in Java
- Each type “conforms” to ancestors
- “Nothing” conforms to all
- “Null” conforms to all but Int, Boolean, Unit
- Null and Nothing cannot be superclass (obviously)

Information hiding

- Attributes only accessed through methods of same class ("Information hiding")
- Methods are global (public)
- Getters and Setters necessary
- Example

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

Inheritance

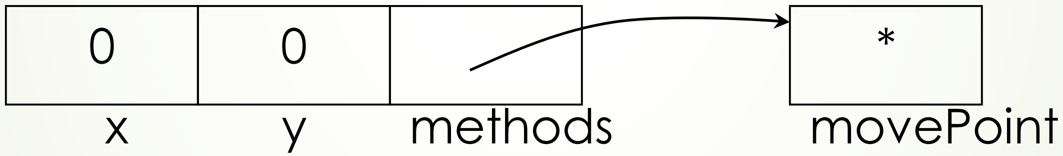
- We can extend points to colored points using subclass (class hierarchy)

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

0	0	0	*
x	y	color	movePoint

Classes: behind the scenes

- An instance of a class has pointer to method
  - All instances of a class point to single instance of method
- Example:



The “new” command

- Explicit

```
class Main {
    i : IO <- new IO;
    main():Int {
        i.out_string("Hello World!\n"); 1; };
};
```

- Anonymous

```
class Main {
    main():Object {
    (new IO).out_string((new IO).in_string().concat("\n"))};
};
```

Typing

- Static typing: Type defined at variable definition time
- Static type inferred at compile time for each expression
  - Bottom-up
- Dynamic, exec time type may differ
  - But dynamic type "conforms" to static type
- Errs on the conservative side
- Attributes of classes initialized
  - Boolean: false, Unit: (), Int: 0
  - Others: Null

Method invocation

- Dynamic type may differ from static
- Methods invoked instead of calling -> Late binding
- Invoked by dispatch
  - Find class of the instance
  - Find the method
  - Eval args
  - Bind the method and the object
  - Run method

Memory Management: Environment and Store

- Environment: Mapping of identifiers to locations
  - i.e. symbol table
- Store: Where the values are stored -> Heap
- Looking up a variable's value:
  - Look its location up in environment
  - Look up the value at location in store
- When new is invoked
  - New env. and store created, new identifier added to them
- Dynamic allocation w/ Garbage Collector

Conclusion

- Object oriented
  - Objects, virtual method calls, but no overloading
- Strongly typed
  - Primitives for int, boolean, string
  - Reference types
- Dynamic allocation and Garbage Collection
  - Heap allocation, automatic deallocation
- Many things are left out
  - short time implementation

Question time!

Thank you for (if) your attention! :)