Cool Overview
Lecture 2
Previously

- Kevin’s Office Hours: Thursdays, 2-4pm, BBB 2717
- Lawrence’s Office Hours: Tuesdays/Thursdays, 11am-12:30pm, BBB Learning Center

- Compilers are language processing tools
  1. Lexing
  2. Parsing
  3. Type-checking
  4. Code generation
  5. Optimization
COOL

▶ Classroom Objected Oriented Language

▶ Easy to lex, parse, and type check

▶ Also COOL Assembly
  ▶ We’ll cover it in time for PA5
  ▶ RISC-based fake-o ISA a la MIPS
Why COOL?

I am a compiler/interpreter enthusiast. That sparked my interest, so I've been experimenting with writing a Cool compiler/interpreter in Cool itself, and to bootstrap, I've been using the Cool interpreter from the UVA CS 4610/4501 webpage.
Why COOL?

I am a compiler/interpreter enthusiast. That sparked my interest, so I've been experimenting with writing a Cool compiler/interpreter in Cool itself, and to bootstrap, I've been using the Cool interpreter from the UVA CS 4610/4501 webpage,

```
cool-cool

An interpreter for Classroom Object-Oriented Language (Cool) written in Cool.

This interpreter can be run using the UVA Cool Interpreter. A wrapper script is provided that allows multiple input files to be specified to the Cool Interpreter, and another script is provided to pass all requisite files for the interpreter itself to the Cool Interpreter:

```
$ export COOL_INTERPRETER=path/to/interpreter/cool
$ bin/cool test/interpret/hello-world.cl
Hello, World!
```

The interpreter can also run itself:

```
$ bin/cool --bootstrap test/interpret/hello-world.cl
Hello, World!
```
COOL Topics

- Program structure
- Classes
- Attributes/Methods
- Types/Inheritance
- Expressions
- Object creation
- Built in methods
- Dispatch
COOL Program Structure

- Single file
- List of classes

```java
1 class Main {
2     main ( ) : Object {
3         0
4     }
5   };
6 }
```
COOL Program Structure

- Single file
- List of classes

```java
class Main {
    main () : Object {
        0
    }
}
```

```java
class Main {
    public static void main (String args[]) {
        // ...
    }
}
```
COOL Program Structure

- No globals, everything is in a class

```plaintext
int my_global = 5;

int main () {
    return 0;
}

x = 5

y = my_func(2)

def my_func(y):
    global x
    return y + x

def main():
    return 0
```
COOL Classes

- Everything is inside classes
- classes defined in any order
- classes are top level (no nesting!)

```java
class Main {
    main ( ) : Object {
        0
    }
}

class Amazing {
    y : Int;
}

class Fantastic {
    z : Int;
}

class Main {
    public static void main (String args[]) {
        // ...
    }
}

class Amazing {
    private int y;
}

class Fantastic {
    private int z;
}
```
COOL Classes

No nested classes!

```java
1 class Main {
2     class insideMain {
3         int x;
4         int getX() {
5             return x;
6         }
7         public insideMain() {
8             x = 0;
9         }
10     }
11     } // getInsideMainX() {
12         return (new insideMain()).getX();
13     }
14     /* (mostly) valid Java, allows nesting */
15 }
```
COOL Classes

- Each Class is a list of *features*
  - Features are attributes or methods
- All attributes are private
- All methods are public
- No static methods/attributes
COOL Topics

- Program structure
- Classes
- Attributes/Methods
- Types/Inheritance
- Expressions
- Object creation
- Built in methods
- Dispatch
COOL Attributes and Methods

```scala
class MyClass {
  (* all data fields are private *)
  x : Int;  (* We provide no initializer (0 for Int) *)
  y : Bool <- false;  (* or we can provide a default *)
  z : String;
  w : MyClass;  (* Objects default to void *)

  (* all methods are public *)
  getX() : Int { x; }
  getY() : Bool { y; }
  getZ() : String { z; }

  setX( nx : Int ) : Int { x <- nx; }
  setY( ny : Bool ) : Bool { y <- ny; }
  setZ( nz : String ) : String { z <- nz; }

  (* No real constructor, but we can make a method to call later to initialize objects *)
  init( nx : Int, ny : Bool, nz : String ) : MyClass {
    setX(nx);
    setY(ny);
    setZ(nz);
    self;
  }
}
```
COOL Attributes and Methods

class Main {
  z : MyClass;
  main(): Object {
    z <- new MyClass
  }
};

class MyClass {
  myMain : Main <- (new Main);
  (* note no methods are called at
    the creation of an object! *)
  init(): Object {
    myMain
  }
};
COOL Topics

- Program structure
- Classes
- Attributes/Methods
- Types/Inheritance
- Expressions
- Object creation
- Built in methods
- Dispatch
COOL Types and Inheritance

- Everything is a class (no virtual, no interfaces)

- “Primitive” boxed types
  - Int (default 0)
  - Bool (default false)
  - String (default empty string, “”)

- Other Types
  - Main: Boilerplate for code entry
  - IO: inherit this for I/O functions!
  - Object: Everything inherits Object
COOL Type Hierarchy

Object

Main

IO

Int

Bool

String

Class A

Class B

Subclass C

can’t inherit!

“AND TO MY NO GOOD NEPHEW MILO WHO THOUGHT HE WAS GOING TO GET ALL MY CASH—LOTS OF LUCK!”
COOL Type Hierarchy

Inherit from one at a time...

can’t inherit!
COOL Type Hierarchy

- Main
- IO
- Object
- Class A
- Class B
- Subclass C
- Int
- Bool
- String

can’t inherit!

...or the other
COOL Type Hierarchy

Object

Main

IO

Int

Bool

String

can’t inherit!

Class A

Class B

Subclass C

but no multiple inheritance!
COOL Inheritance

```java
1 class A {
2     x : Int <- 5;
3     method () : Object {
4         x
5     }
6 }
7
8 class B {
9     y : Int <- 0;
10    method () : Object {
11        y
12    }
13 }
14
15 (* _not_ supported by Cool... *)
16 class C inherits A, B {
17    broken () : Object {
18        (* ...which version of
19            "method"
20            gets called? *)
21            method()
22    }
23 }
```
No Inheritance Cycles!

```java
5 (* which features would
6 each class have?? *)
7 class A inherits C {
8     x : Int;
9 }
10
11 class B inherits A {
12     y : Int;
13 }
14
15 class C inherits A {
16     z : Int;
17 }
```

**LIFE-CYCLE OF A SOFTWARE DEVELOPER**

**WHO WROTE THIS PIECE OF CRAP?!**

M. VIRKUS '17
Looking Ahead...

Object

Main

IO

Class A

Subclass B

Subsubclass C

How might we detect this?

Compiler Construction
No attribute redefinition

```java
10 class Animal {
11     noise : String <- "noise";
12     getNoise () : String {
13         noise
14     };
15 }
16 (* Subclasses are not allowed to redefine attributes *)
17 class Cat inherits Animal {
18     noise : String <- "meow";
19 }
20 }
21 class Dog inherits Animal {
22     noise : String <- "bark";
23 }
24 ```
No method signature changes

class Animal {
    noise : String;
    getNoise () : String {
        noise
    }
    init () : Animal {{
        noise <- "noise";
        self;
    }};
};

(* Subclass cannot change type signature of methods *)
class Cat inherits Animal {
    init () : Cat {
        noise <- "meow";
        self;
    }};
}
No method signature changes

```java
10 class Animal {
11     noise : String;
12     getNoise () : String {
13         noise
14     };
15     init () : Animal {{
16         noise <- "noise";
17         self;
18     }};
19 }
20
21 (* Subclass cannot change type
22     signature of methods *)
23 class Cat inherits Animal {
24     (* still won't work *)
25     init ( newnoise : String ) : Animal {{
26         noise <- newnoise;
27         self;
28     }};
29 }
30 ```
Method Override

class Animal {
    noise : String;
    getNoise () : String {
        noise
    };
    init () : Animal {{
        noise <- "noise";
        self;
    }};
}

(* Subclass cannot change type
    signature of methods *)
class Cat inherits Animal {
    init () : Animal {{
        noise <- "meow";
        self;
    }};
};
COOL Topics

- Program structure
- Classes
- Attributes/Methods
- Types/Inheritance
- Expressions
- Object creation
- Built in methods
- Dispatch
COOL Expressions

- No statements, only expressions
  - Expressions have associated types
- Every method contains exactly one expression!

```java
class Main {
    main () : Object {
        /* main returns a new object
        (an Int) containing the value
        11 * 5 + 6
    }
};
```
All COOL Expressions

- Constants: 1, true, "Hello"
- Arithmetic: 1 + 2, 1 * 3
- Identifiers: x
- Assignment: x <- expr
- Comparison: x < 2, 3 <= x
- new: new ClassName
- isvoid: isvoid x
All COOL Expressions (2)

- **Conditional:**
  
  ```
  if cond_expr then then_expr else else_expr fi
  ```

- **Loop:**
  
  ```
  while cond_expr loop body_expr pool
  ```

- **Blocks:**
  
  ```
  { expr_1 ; expr_2 ; ... ; expr_n ; }
  ```

- **Let binding:**
  
  ```
  let x : ClassName <- expr_init in expr_body
  ```

- **Case:**
  
  ```
  case expr_0 of x1 : Class1 => expr_1 ; ... ; xn : Classn => expr_n ;
  ```
Constant Expression

```java
1 class Main {
2     main () : Object    { };
3     getFive() : Int     { 5 };
4     getFalse() : Bool   { false };
5     getHello() : String { "Hello" };
6 }
```
Arithmetic

```java
class Main {
    main(): Object {};
    getArithmetic () { 5 + 6 };
};
```
class Main {
  x : Int <- 5;
  main () : Object { x + 7};
};
class Main {
    x : Int;
    main () : Object {
        x <- 5
        }
}

Comparison

```java
1 class Main {
2     main () : Object {
3         5 < 3
4         (* returns a new Bool, false *)
5     };
6 }
```
new and isvoid

```java
1 class Main {
2     main : Object { new A };
3     (* A is a subclass of Object *)
4
5     myObj : A;
6     check : Object { isvoid ( myObj ) };
7     (* uninitialized A *)
8 };
9
10 class A {};
```
Conditional

```java
class Main {
    main : Object {
        if 5 < 3 then "Hello" else "Goodbye" fi
    }
};
```
class Main {
    main : Object {
        if 5 < 3 then
            if 5 < 4 then
                "Hello"
            else
                "Aloha"
            fi
        else
            "Goodbye"
        fi
    }
};
While Loops

```java
1 class Main {
2     x : Int;
3     main () : Object {
4         while ( x < 100 ) loop
5             x <-
6             if x < 50 then
7                 (* x gets x + 2 if x < 50 ... *)
8                     x + 2
9                 else
10                 (* .. otherwise x gets x + 1 *)
11                     x + 1
12                 fi
13             pool
14         };
15     };
```
Blocks!

class Main {
    x : Int;
    y : Int <= 5;
    main (): Object {
        (* Block allows sequential expressions *)
        x <= 4;
        if x < y then
            x <= 5
            else
                x <= 6
                fi;
        (new Other).getHello();
        (* Block expr value is that of last expr in sequence *)
        } (* thus main yields a string, "Hello" *)
    }
};

class Other {
    getHello (): String { "Hello" }
};
Let

class Main {
    main () : Object {
        (* let helps create new bindings *)
        let x : Int <- 5 in
        x + 4
        (* main returns 9 *)
    }
};
class Main {
    x : A <- new B; (* or whatever... *)
    main () : Object {
        case x of
            a : A => 0;
            b : B => 1;
            c : C => 2;
            d : D => 3;
        esac
    }
}

class A {};
class B inherits A {};
class C inherits A {};
class D inherits A {};}
COOL Topics

- Program structure
- Classes
- Attributes/Methods
- Types/Inheritance
- Expressions
- Object creation
- Built in methods
- Dispatch
There is a default constructor

Basicallly, run all initializers for attributes

```java
class Main {
    main () : Object { (new ColorPoint3D).getColor() }
};

class Point2D {
    x : Int <- 0;
    y : Int <- 0;
    pointName : String ; (* default init to "" *)
};

class Point3D inherits Point2D {
    z : Int <- 0;
};
class ColorPoint3D inherits Point3D {
    color : String <- "red";
    getColor () : String {color};
};
```
COOL Topics

- Program structure
- Classes
- Attributes/Methods
- Types/Inheritance
- Expressions
- Object creation
- Built in methods
- Dispatch
Built in Methods

- **IO**: Input/Output
  - Inherit for access to
  - `out_string` and `in_string`
  - `out_int` and `in_int`

- Everything comes from Object
  - `copy` (Shallow copy)
  - `abort` (give up execution)
  - `type_name` (String with dynamic type)

```java
1 class Main inherits IO {
2     main () : Object {
3         out_string( (new B).type_name() )
4     }
5 }
6 class A {
7     class B inherits A {
8     }
9 }
```
COOL Topics

- Program structure
- Classes
- Attributes/Methods
- Types/Inheritance
- Expressions
- Object creation
- Built in methods
- Dispatch
Dispatch

- A fancy word for method calls
- Self, Dynamic, and Static dispatch

```java
class Main inherits IO{
  x : B; (* static type is B *)
  main () : Object {
    x <- new C; (* dynamic type is C *)
    out_string ( get() ); (* C *)
    out_string ( x.get() ); (* C *)
    out_string ( x@A.get() ); (* A *)
  }

  get () : String { x.get() ;
}
}

class A {
  get() : String { "I am a A\n" ;
}
}

class B inherits A {
  get() : String { "I am a B\n" ;
}
}

class C inherits B {
  get() : String { "I am a C\n" ;
}
}
```
Cool List Implementation

How might we implement a List structure in Cool?
Coming up

- PA1c due
- PA1 due 1/17
- Lexical Analysis Next
  - Brush up on your DFA/NFA/RegExp knowledge