Cool Overview

UNIVERSITY OF DELAWARE

Disclaimer

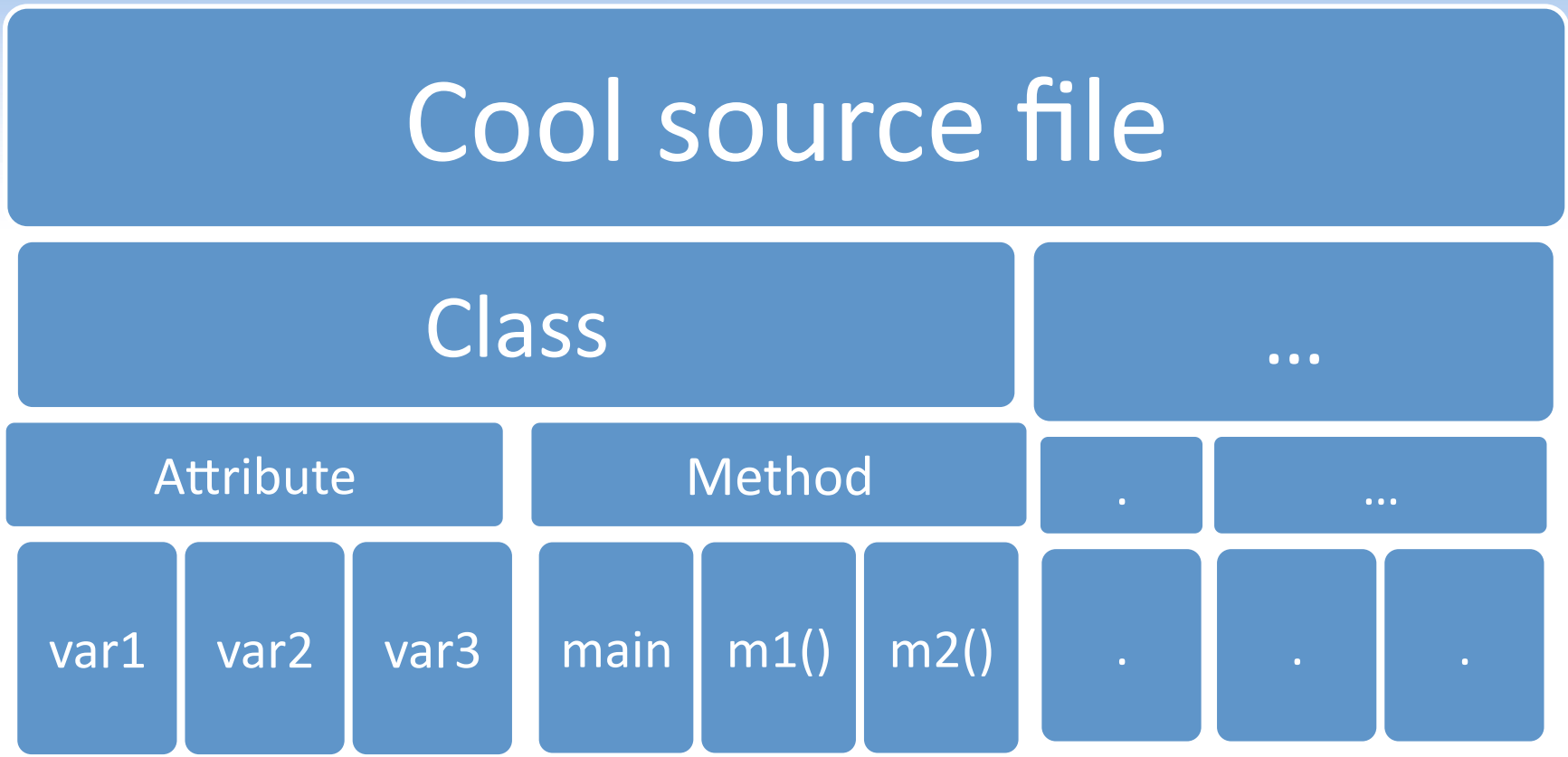
The following does not describe the Cool language in depth. It is not designed to be used as a syntax reference, but rather as an introduction into programming with Cool, and also into object oriented programming in general.

For actually writing your own Cool compiler please read the Cool manual carefully.

UNIVERSITY OF DELAWARE

What is Cool?

- Classroom Object Oriented Language
- Collection of classes spread over files
- *Main* class with a *main* method.
- Similar to Java
- The more restricted the language, the easier to write a compiler


Cool source file
Class
…
Attribute
Method
.
…
var1
var2
var3
main
m1()
m2()
.
.
.

UNIVERSITY OF DELAWARE

Class

- Object is the super class for all other classes
- IO, Int, String and Bool are basic types (in JAVA parlance primitive types), and cannot be inherited
- Multiple inheritance is not allowed
- Restricted Function overriding

UNIVERSITY OF DELAWARE

Attributes

- Local variables
- Scope lasts until the class
- Garbage collection is automatic

Method

<id> (<param_id$_1$> : <type>,...,< param_id$_n$> : <type>): <type> {
    <expr>
        …
};

e.g.,

```
sum (num1 : Int, num2 : Int) : Int {
 total <- num1 + num2
};
```

<expr>

- Constant
  Example: 1 or “String”
  The type of such an <expr> is the type of the constant

- Identifier (id)
  Example: a local variable
  The type of such an <expr> is the type of the id

<expr> cont'd

- Assignment
  <id> <- <expr>
  The type of such an <expr> is the type of <expr> and should be the same as the <id>

- Dispatch
  [<expr>[@<type>]].id(<expr>,...,<expr>)
  The type of dispatch is however more complicated, please read pg. 8 of the Cool manual

IO Example

```
class Main {
      myIO : IO <- new IO;
      myInput : Int;
      main() : Int {{
        myIO.out_string("How many? ");
        myInput <- myIO.in_int();
        while 0 <  myInput loop
              myIO.out_string(''Hello world!'')
        pool;
        0;
      }};
};
```

Inheritance

```
class Silly {
    f() : Int {5};
};
class Sally inherits Silly { };
class Main {
    x : Int <- (new Sally).f( );

    main() : Int {x};
};
// remember restriction in function overriding.
```

Inheritance cont'd…

```
class Silly {
    f() : Int {5};
};
class Sally inherits Silly {
    f() : Int {7};
};
class Main {
    x : Int <- (new Sally)@Silly.f( );
     main() : Int {x};
};
```

The COOL Manual

- The Cool manual will be your main reference when working on any of the phases of your Cool compiler.
- Sections 1 and 2 (2 pages) explain how to compile and run (using the SPIM interpreter) a Cool program.
- Sections 2-11 (13 pages) are required to build the two phases of the syntax analysis.
- Section 12 (5 pages) is sufficient for the semantic analyzer (together with earlier pages).
- Section 13 (8 pages) are necessary for the code generator. Furthermore you should read the SPIM manual (<25 pages) explaining our target language.

$$
\begin{array}{rcl}
\mathit{program} & ::= & \mathit{class};^{+} \\
\mathit{class} & ::= & \textbf{class TYPE [inherits TYPE] } \{ \ \mathit{feature};^{*} \ \} \\
\mathit{feature} & ::= & \mathrm{ID}(\mathit{formal},^{*}) : \mathrm{TYPE} \ \{ \ \mathit{expr} \ \} \\
 & | & \mathrm{ID} : \mathrm{TYPE} \ [\ \texttt{<-} \ \mathit{expr} \ ] \\
\mathit{formal} & ::= & \mathrm{ID} : \mathrm{TYPE} \\
\mathit{expr} & ::= & \mathrm{ID} \ \texttt{<-} \ \mathit{expr} \\
 & | & \mathit{expr}[@\mathrm{TYPE}].\mathrm{ID}(\mathit{expr},^{*}) \\
 & | & \mathrm{ID}(\mathit{expr},^{*}) \\
 & | & \textbf{if } \mathit{expr} \textbf{ then } \mathit{expr} \textbf{ else } \mathit{expr} \textbf{ fi} \\
 & | & \textbf{while } \mathit{expr} \textbf{ loop } \mathit{expr} \textbf{ pool} \\
 & | & \{ \ \mathit{expr};^{+} \ \} \\
 & | & \textbf{let } [\![\mathrm{ID} : \mathrm{TYPE} \ [\ \texttt{<-} \ \mathit{expr} \ ],]\!]^{+} \textbf{ in } \mathit{expr} \\
 & | & \textbf{case } \mathit{expr} \textbf{ of } [\![\mathrm{ID} : \mathrm{TYPE} \Rightarrow \mathit{expr};]\!]^{+} \textbf{esac} \\
 & | & \textbf{new TYPE} \\
 & | & \textbf{isvoid } \mathit{expr} \\
 & | & \mathit{expr} + \mathit{expr} \\
 & | & \mathit{expr} - \mathit{expr} \\
 & | & \mathit{expr} * \mathit{expr} \\
 & | & \mathit{expr} \,/\, \mathit{expr} \\
 & | & \tilde{} \,\mathit{expr} \\
 & | & \mathit{expr} < \mathit{expr} \\
 & | & \mathit{expr} <= \mathit{expr} \\
 & | & \mathit{expr} = \mathit{expr} \\
 & | & \textbf{not } \mathit{expr} \\
 & | & (\mathit{expr}) \\
 & | & \mathrm{ID} \\
 & | & \mathrm{integer} \\
 & | & \mathrm{string} \\
 & | & \textbf{true} \\
 & | & \textbf{false}
\end{array}
$$