UNIVERSITY OF DELAWARE

Cool Overview and SVN Configuration

Sameer Kulkarni

September 14, 2009

Acknowledgements

The present presentation has been based on a similar talk given by Timo Kötzing and can be found at http://www.cis.udel.edu/~cavazos/cisc672-fall08/lectures/CoolOverview.pdf .

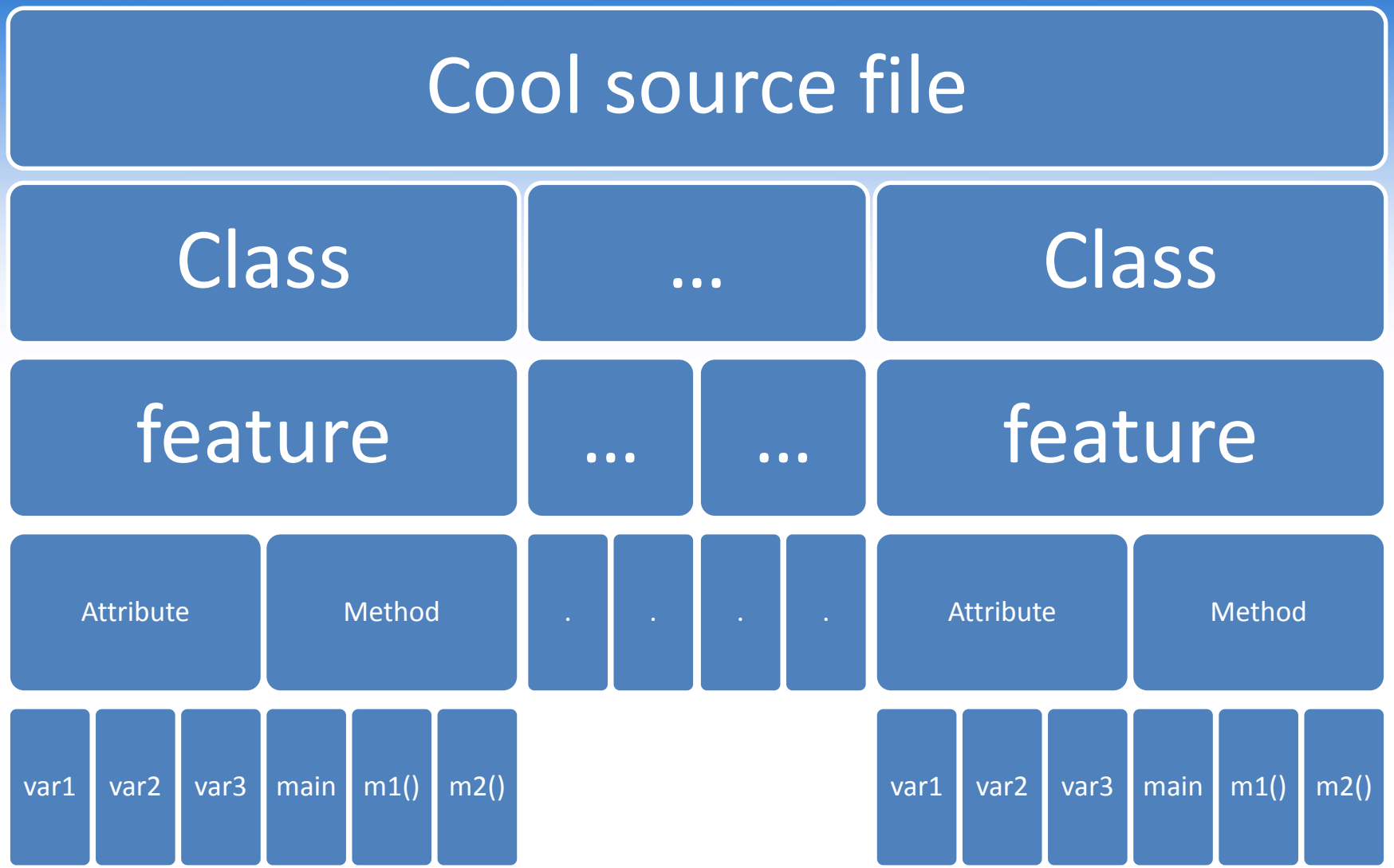
Disclaimer

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.

Isn't it COOL??

- Classroom Object Oriented Language
- Collection of classes spread over $(files)^{+}$.
- *Main* class with a *main* method.
- Closest to Java
- The more restricted the language, the easier it is to write the compiler ☺


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

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

Attributes

- Local variables
- Scope lasts till 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
  1 or “String”
  - The type of such an <expr> is the type of the constant
- Identifier (id)
  like a local variable
  - The type of such an <expr> is the type of the id
- 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.

The Cool Manual, cont'd…

- 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.

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

SVN

- The sub versioning tool is installed on stimpy and can be used from there to checkout code and other resource files

  svn co svn://<username>@svn.acad.ece.udel.edu:67209/repos/cisc672_09f

- The usernames and passwords have been given to you.
- Tortoise SVN is a GUI based svn tool that can be used if you are using windows.

  svn+ssh://<username>@svn.acad.ece.udel.edu/repos/cisc672_09f

SVN cont'd…

- *svn update:* would sync your repository to the latest version present on the server
- *svn commit*: to commit the changes you have made.
- *svn add*: adds presently non-subversioned files to the local repository. This will not update the server, till you commit your changes
- *svn delete*: removes the files from svn control, but the files will remain on the local system till you commit your changes.

http://www.linuxfromscratch.org/blfs/edguide/chapter03.html

Commiting to SVN

- Commit comments: when you commit, svn would ask for your comments, which can be used for future reference.
- SVN_EDITOR : would be used to open an editor to add your comments unless you specify some in the command line.