

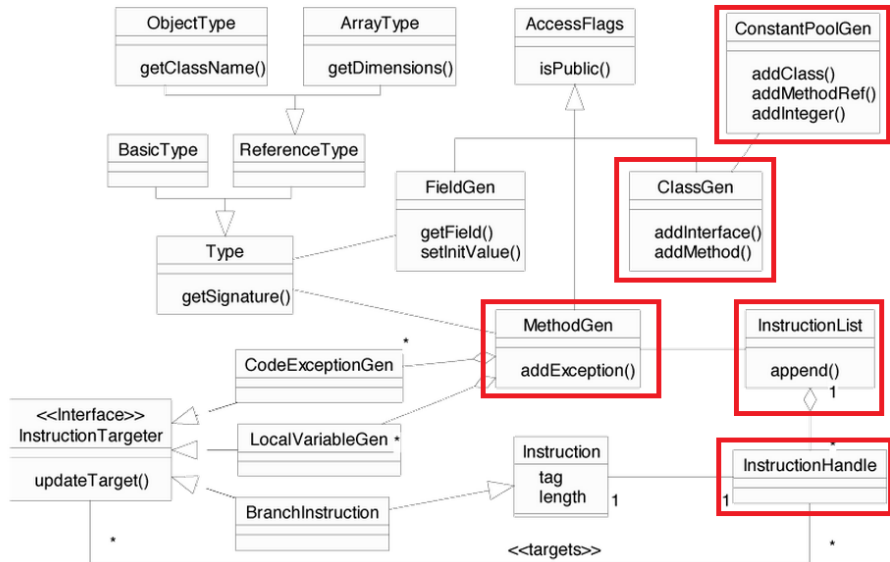
Bytecode-Generierung eines neuartigen Java Compilers

Evelyn Fikus, Franziska Fütterling, Julia Schubert,
Andreas Stadelmeier, Martin Plümicke

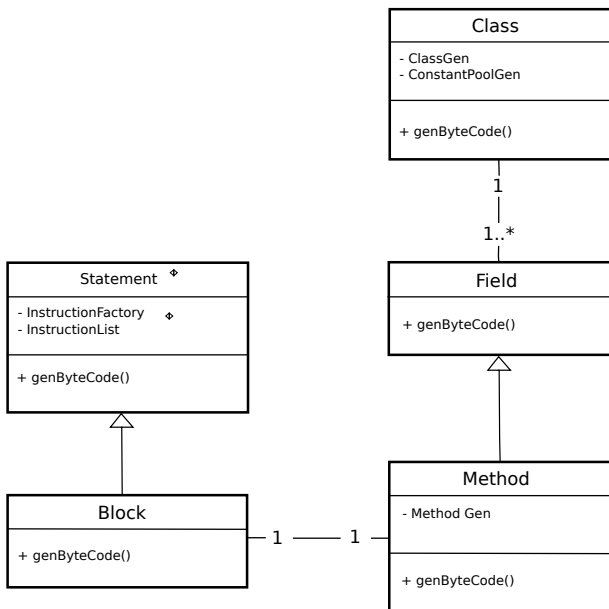
Baden-Wuerttemberg Cooperative State University
Stuttgart/Horb

5. Oktober 2015

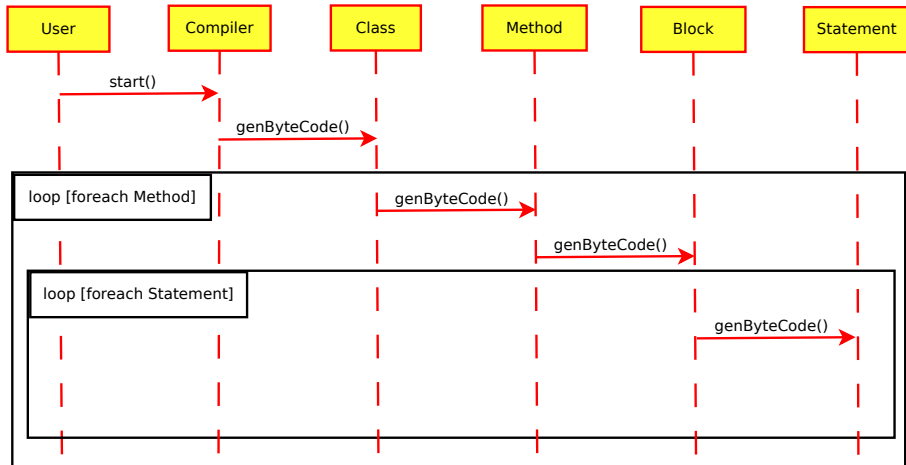
Byte Code Engineering Library (BCEL)



Architektur (Klassendiagramm)



Architektur (Sequenzdiagramm)



genByteCode in Class.java

```
1  /* genByteCode in Class.java
2  */
3  private ConstantPoolGen    _cp;
4  private ClassGen          _cg;
5
6  public ByteCodeResult genByteCode() throws IOException {
7
8      _cg = new ClassGen(name, superClass.get_Name(), name + ".
9          java", Constants.ACC_PUBLIC , new String[] { });
10     _cp = _cg.getConstantPool();
11
12     for(Field field : this.fielddecl){
13         field.genByteCode(_cg);
14     }
15
16     ByteCodeResult code = new ByteCodeResult(_cg);
17     return code;
18 }
```

genByteCode in Method.java

```
1 public void genByteCode(ClassGen cg) {
2     ConstantPoolGen _cp = cg.getConstantPool();
3     InstructionFactory _factory = new InstructionFactory(cg, _cp);
4     InstructionList il = new InstructionList();
5     Class parentClass = this.getParentClass();
6
7     MethodGen method = new MethodGen(Constants.ACC_PUBLIC, this.
8         getType().getBytecodeType(), org.apache.bcel.generic.Type.
9         NO_ARGS, new String[] { }, this.getMethod_Name(),
10        parentClass.name, il, _cp);
11
12
13    Block block = this.getBlock();
14    InstructionList blockInstructions = block.genByteCode(cg);
15    il.append(blockInstructions);
16
17    if (block.get_Statement().size() == 0) { il.append(_factory.
18        createReturn(org.apache.bcel.generic.Type.VOID)); }
19    else {
20        if (!(block.get_Statement().lastElement() instanceof
21            Return)) { il.append(_factory.createReturn(org.apache.
22                bcel.generic.Type.VOID)); }
23    }
24
25    method.setMaxStack(); //nach dem alle Instructions angehängt wurden
26
27    cg.addMethod(method.getMethod());}
```

Integration ins Eclipse-PlugIn

The screenshot shows the Eclipse IDE interface with the following components and annotations:

- Package Explorer (Left):** Shows a project structure with a package named 'lambda' containing several Java files. 'Overloading.java' is selected and highlighted.
- Code Editor (Center):** Displays the source code of 'Overloading.java'. The code includes two classes: 'Overloading2' and 'Overloading'. Annotations are present: a blue arrow points to the 'return this;' line in 'Overloading2', a green arrow points to the 'var' declaration in 'Overloading', and a red arrow points to the 'overload()' method call in 'Overloading'. A tooltip is visible over the 'overload()' call, displaying 'Remarks: Overloading2.overload()'.
- Outline (Right):** Shows a class hierarchy for 'Overloading2' and 'Overloading'. The 'Overloading' class is selected, and its members ('var', 'test', 'overload', 'Overloading') are listed.
- Annotations and Callouts:**
 - Save Button:** An arrow points to the 'Save' icon in the top toolbar with the text: "Press Save-Button and trigger the type inference process".
 - Markers of missing types:** Three arrows point to the blue, green, and red annotations with the text: "Markers of missing types".
 - Tool-Tips:** An arrow points to the tooltip over the 'overload()' call with the text: "Tool-Tips: Show remarks, as errors and different possible types".
 - Select a type:** An arrow points to the 'Overloading' class in the Outline with the text: "Select a type by pressing the right mouse button".
- Problems View (Bottom):** Currently empty, indicating no errors or warnings.

Vorgehen mit Eclipse

- ▶ Wenn nur eine Möglichkeit der Typisierung gegeben ist, wird Bytecode erzeugt.
- ▶ Wenn mehrere Typisierungen möglich sind, muss der User auswählen. Erst dann wird Bytecode erzeugt.

Vorgehen mit Eclipse

- ▶ Wenn nur eine Möglichkeit der Typisierung gegeben ist, wird Bytecode erzeugt.
- ▶ Wenn mehrere Typisierungen möglich sind, muss der User auswählen. Erst dann wird Bytecode erzeugt.

Ziel: Für Intersection–Types wird Bytecode erzeugt.

- ▶ Generics in Bytecode
- ▶ Fallunterscheidung in überladenen Methoden

Zusammenfassung und Ausblick

Zusammenfassung:

- ▶ Bytecodeerzeugung mit BCEL
- ▶ Integration in das Eclipse-PlugIn

Ausblick:

- ▶ Vollständige Implementierung
- ▶ Intelligenterer Lösung für die Überladungsauflösung