Compiler ULTIMATE

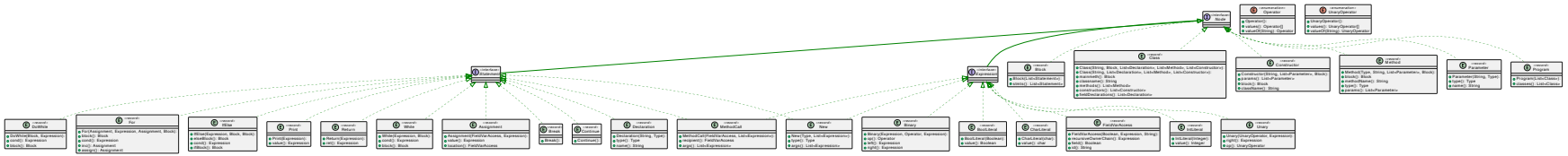
7/5/2024

Parser

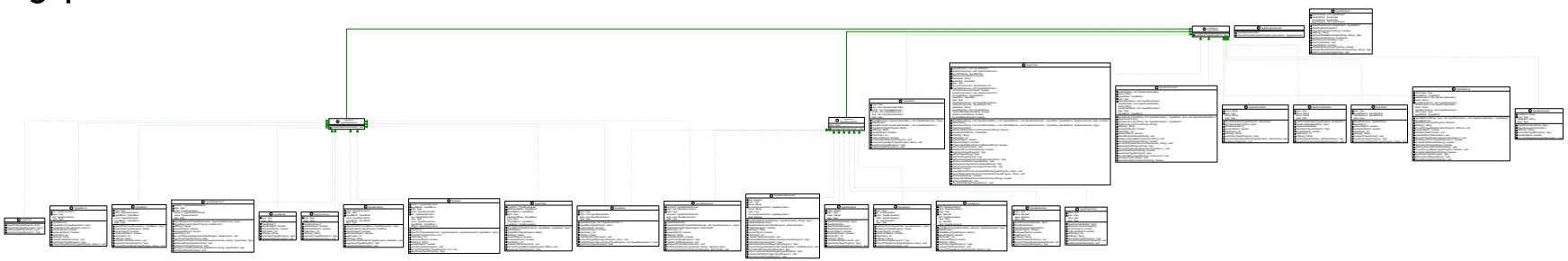
Besonderheiten und Änderungen zu Java:

- Pro Datei nur eine Klasse, mehrer Dateien einlesen
- Keine Accessmodifier
- print statt System.out.println
- i++ eingeschränkt
- Kein ++i
- Zugriff auf Felder nur mit this
- Logische Statements Klammern, sonst von links nach rechts
- Fehlermeldungen werden gesammelt

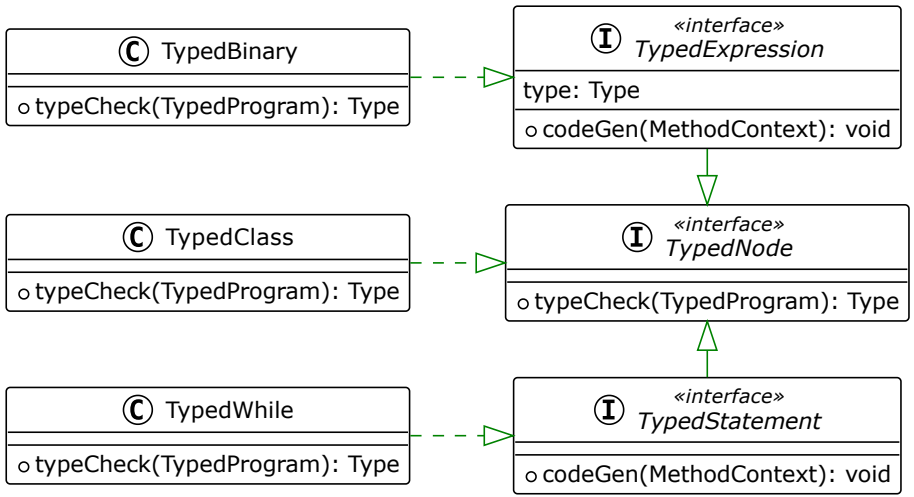
AST



TypedAST



Grobe TypedAST Struktur



Phasen der Typisierung

- Initisialisierungsphase
  - Programm -> Klassen (Name, Feld-Variablen, Methoden, Konstruktoren)
- Konvertierungsphase
  - Überführung von untypisierten AST zu typisierten AST
- TypeCheck-Phase
  - Semantische Überprüfung

Konvertierung von untypisierten AST zu typisierten AST

```
public class TypedDeclaration implements TypedNode {
    private String name;
    private Type type;

    ...

    private void convertToTypedDeclaration(Declaration declaration) {
        name = declaration.name();
        type = declaration.type();
    }
```

TypeCheck von While-Statement

```
public class TypedWhile implements TypedStatement {
    private TypedExpression cond;
    private TypedBlock typedBlock;
    private Type type;

    ...

    @Override
    public Type typeCheck(TypedProgram typedProgram) {
        if (cond.typeCheck(typedProgram) != Type.BOOL) {
            throw new InvalidWhileConditionException();
        }
        type = typedBlock.typeCheck(typedProgram);
        return type;
    }
}
```

Speichern von lokalen Variablen

- Methoden und Konstruktoren haben jeweils eine Liste von lokalen Variablen

```
public int calculateResult(boolean isMultiple, int x, int y) {
    int resultat = 0;// add resultat to list

    if(isMultiple) {
      int a = x * y;// add a to list

      resultat = a;
    }// remove a from list
    else {
      int b = x + y;// add b to list

      resultat = b;
    } // remove b from list

    return resultat;
} // clear list
```

Code Generierung

- Nutzung von ASM
- Dynamische Bindung/Polymorphie

```
public class TypedWhile implements TypedStatement {

    ...

    @Override
    public void codeGen(MethodContext ctx) {
        Label loopStart = new Label();
        Label loopEnd = new Label();
        ctx.getBreakLabels().push(loopEnd);
        ctx.getContinueLabels().push(loopStart);

        ctx.getMv().visitLabel(loopStart);
        cond.codeGen(ctx);

        ctx.getMv().visitJumpInsn(Opcodes.IFEQ, loopEnd);
        ctx.popStack();
        typedBlock.codeGen(ctx);
        ctx.getMv().visitJumpInsn(Opcodes.GOTO, loopStart);
        ctx.getMv().visitLabel(loopEnd);
        ctx.getContinueLabels().pop();
        ctx.getBreakLabels().pop();
```

```
    }
}
```

Code Generierung

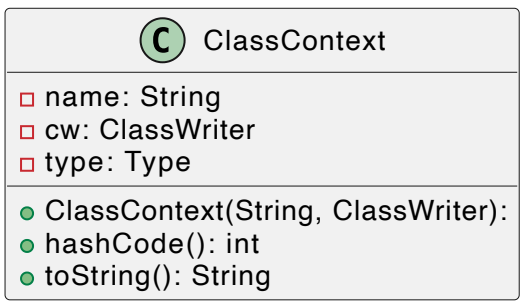
- Nutzung von ASM
- Dynamische Bindung/Polymorphie

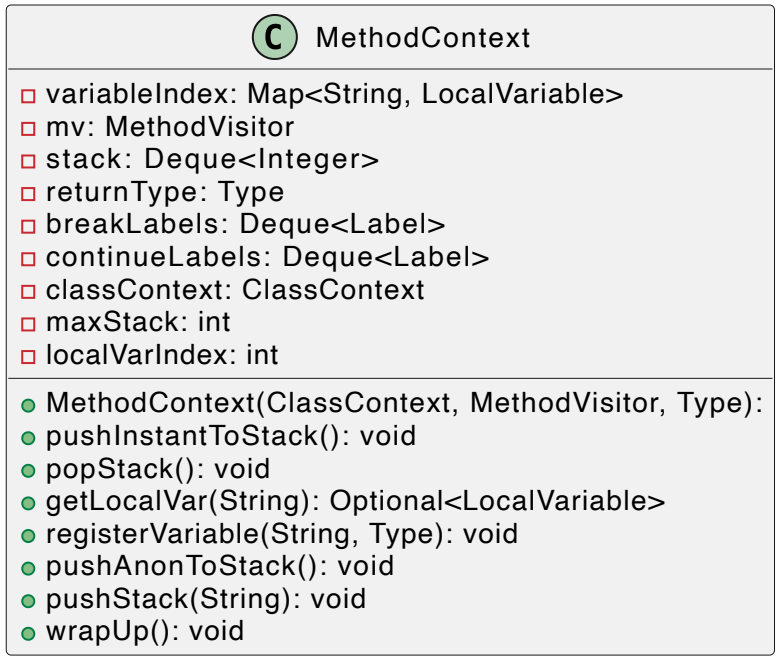
```
public class TypedIntLiteral implements TypedExpression {

    ...

    @Override
    public void codeGen(MethodContext ctx) {
        if (value >= Byte.MIN_VALUE && value <= Byte.MAX_VALUE) {
            ctx.getMv().visitIntInsn(Opcodes.BIPUSH, value);
        } else if (value >= Short.MIN_VALUE && value <= Short.MAX_VALUE) {
            ctx.getMv().visitIntInsn(Opcodes.SIPUSH, value);
        } else {
            ctx.getMv().visitLdcInsn(value);
        }
        ctx.pushInstantToStack();
    }
}
```

Context Klassen für Class und Method





ASMifier

Tool, um aus Java Bytecode ASM Code zu generieren.

Klick