Homework 3

# Semantic Analysis

Michael Faes, with slides by Remi Meier

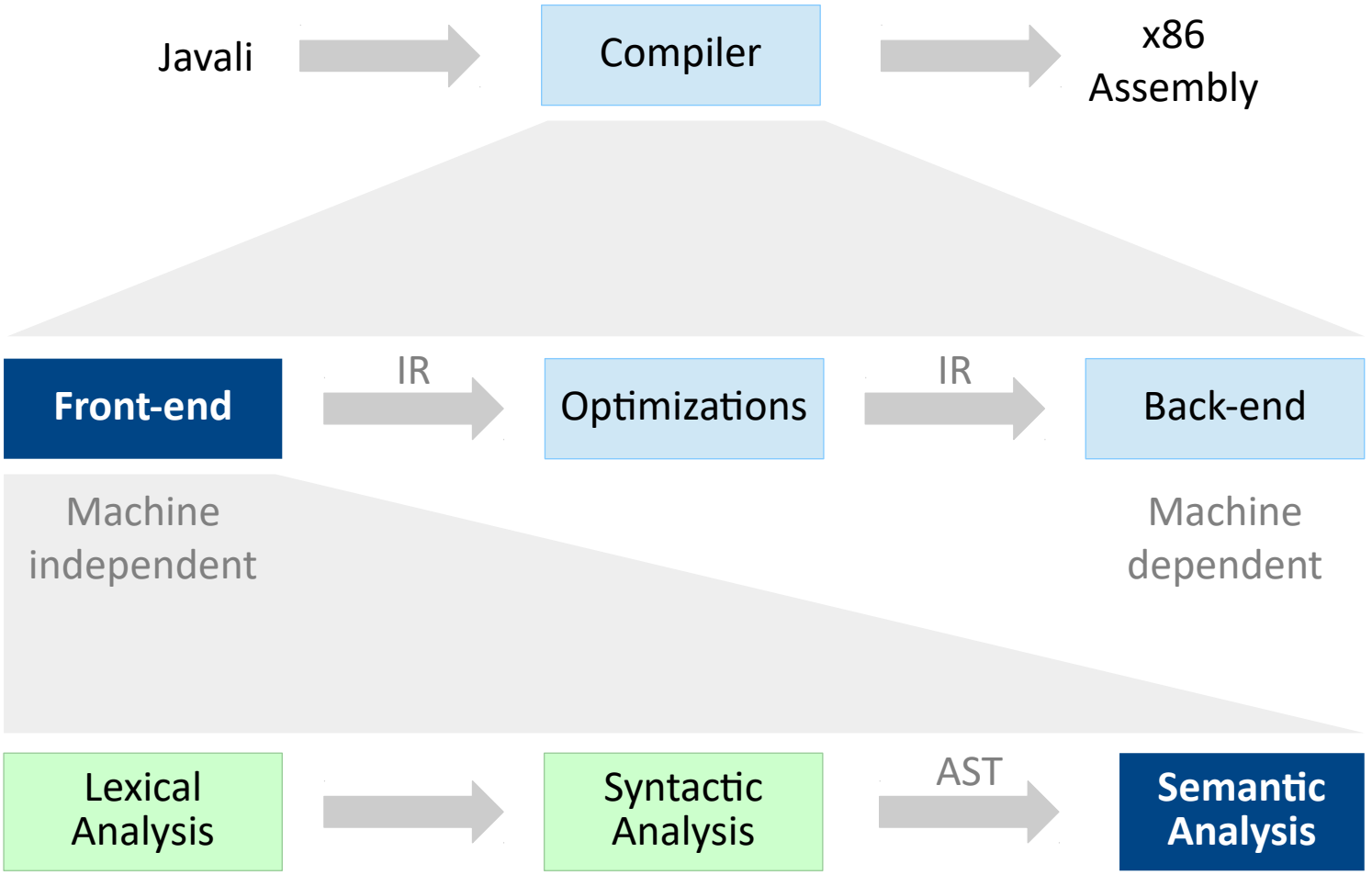
Compiler Design – 29.03.2018

# First Things First: HW1 Grading

- Overall Grade (25 pts)
  - 20 pts for the implementation
  - 5 pts for good test cases
- Implementation is graded automatically
  - Percentage of passed tests
- Exceptions
  - Points removed if traversal order is wrong, even if all tests pass (due to stack allocation)

- Questions: **aristeidis.mastoras@inf.ethz.ch**

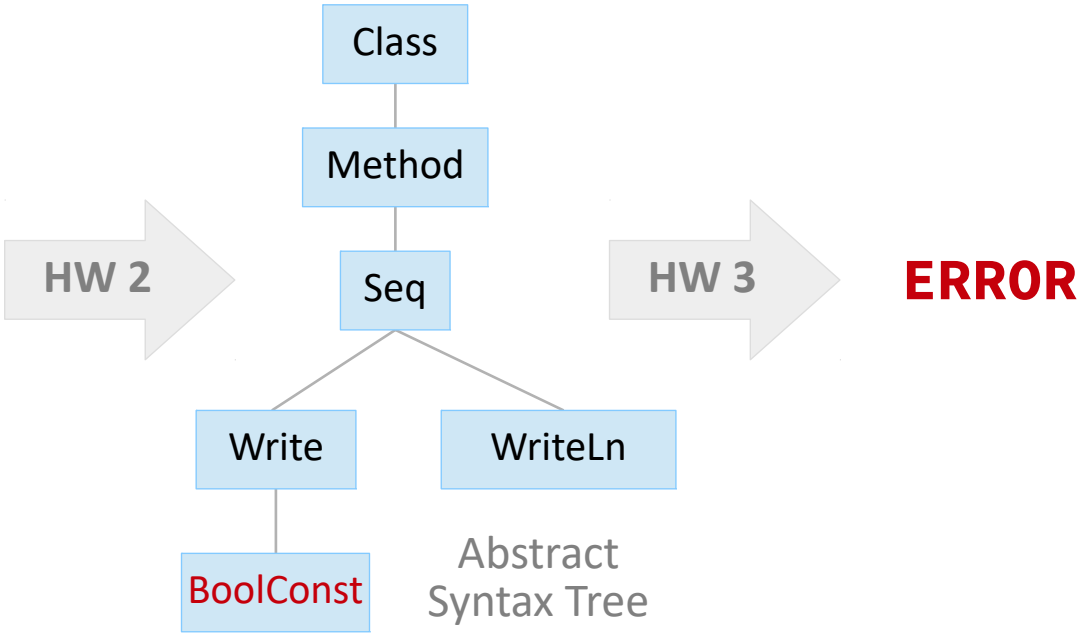
# Compiler phases



# Homework 3

```
class Main {
    void main() {
        write(true);
        writeln();
    }
}
```

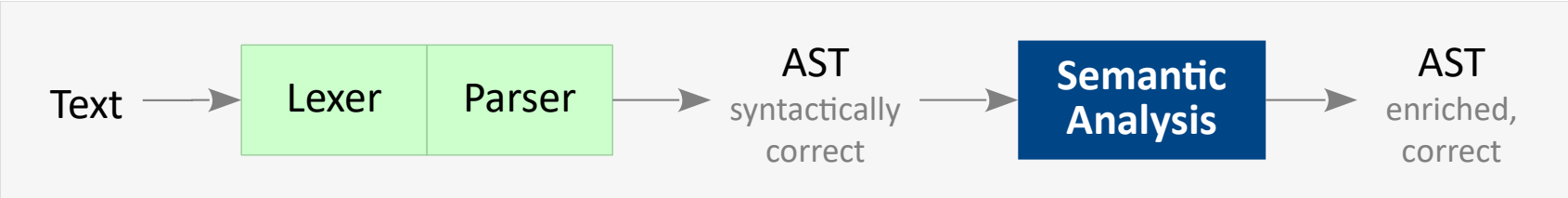
Text




**How do we…**

- check a program for mistakes that are difficult / impossible to catch with grammar rules?

# Semantic Analysis




**Input**

- Syntactically well-formed AST

**Output**

- **Enriched AST** with types and other semantic information
- Warnings: warn programmer about possible mistakes
- Error messages: **ensure correctness** for later compiler stages

**Two parts:**

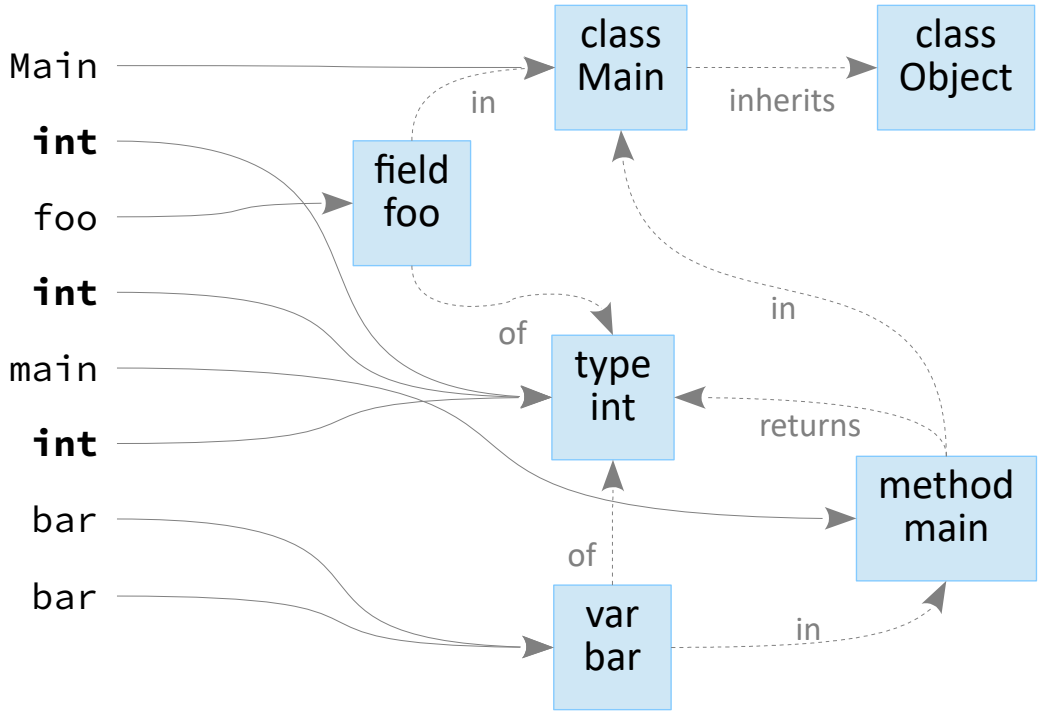
1) Collect information (enrich AST)
2) Check for correctness of the AST

# What information?

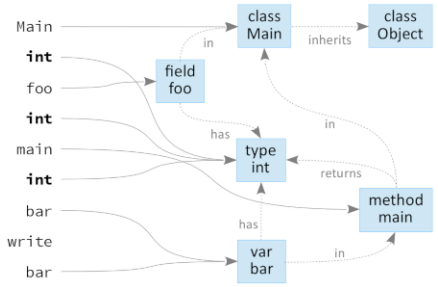
We use the following terms:

- **Names** are words in the program text
- **Symbols** are semantic entities storing information about names

```
class Main {
  int foo;
  int main() {
    int bar;
    write(bar);
  }
}
```



# Symbol table




Keep information about symbols in **symbol table**:

- mapping: name → symbol
- central repository of information
- hierarchical structure through **lexical scoping**

Lexical scoping:

```
class A {

  void foo(int bar) {


  }
}

class B extends A {
}
```

Javali scopes:

- **Global:** all class symbols, built-in types
- **Class:** members and super-class members
- **Method:** parameters and local variables

# Scoping rules

```
class A {

  void foo(int bar) {


  }
}

class B extends A {
}
```

Javali scopes:

global, class, method

Rules:

- within a scope: symbols have **unique names**
- inner scopes **hide** symbols of outer scopes

But:

- separate **namespace** for types
- separate namespace for methods

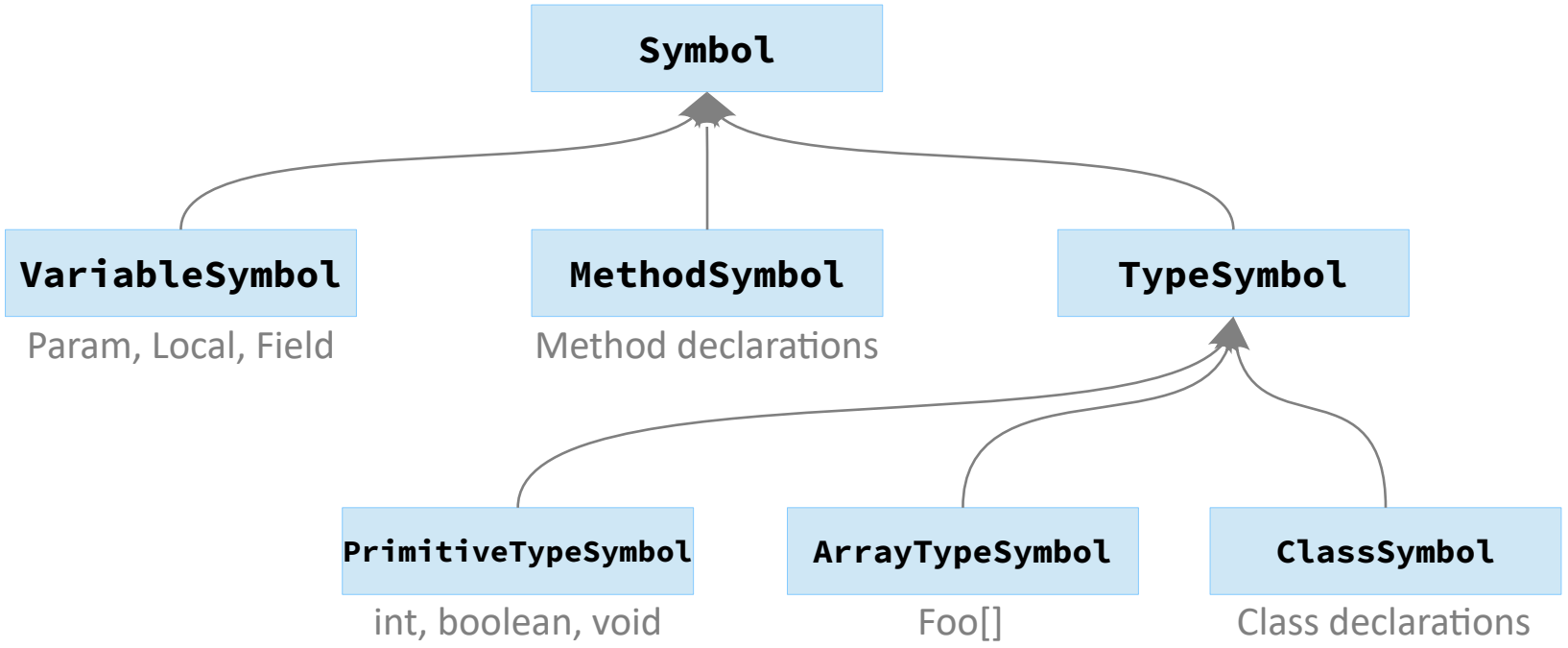
```
class foo {
    int foo;
    void foo() {}
}
```

# Scoping rules: example

```
class foo {
  foo foo;
  void foo(int foo) {
    write(foo);
  }
  void bar() {
  }
}
```

```
class bar extends foo {
  foo foo;
  void bar() {
    foo();
    bar(null);
    foo = new bar();
    foo.foo = null;
  }
}
```

# Implementation




**Hints:**

- Create symbols for every class, method, parameter, local var, array type, field, and primitive type (mind special cases Object & null)
- Enrich AST with symbols

# Implementation

No need for a single symbol table

- link tables to reflect scopes:
  e.g., a symbol table per method symbol
- collect as much information as you need in whatever way you want

Use visitors

- to collect type information
- to check for correctness → **second part**

# Check for correctness

Look at list of checks in homework description

- throw **SemanticFailure** on detected error

```
class Test {
  void m() {
    write(1);
  }
}
```

**INVALID_START_POINT**: no Main or main()

```
class A extends B { }
class B extends A { }
…
```

**CIRCULAR_INHERITANCE**

```
class Main {
  void main() {
    int a, a;
  }
}
```

**DOUBLE_DECLARATION**

# Check for correctness

```
…
int m() {
  if (true) {
    return 0;
  }
}
…
```

MISSING_RETURN

```
…
void m() {
  int a;
  boolean b;
  a = a + b;
}
…
```

TYPE_ERROR

# Notes

- Some checks can be performed while collecting information
- **Visitors** are your friends

# Questions?