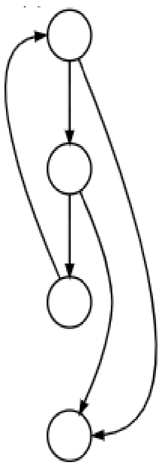
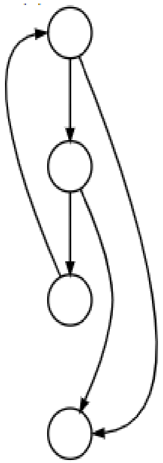


Program Analysis

Venkatesh Vinayakarao

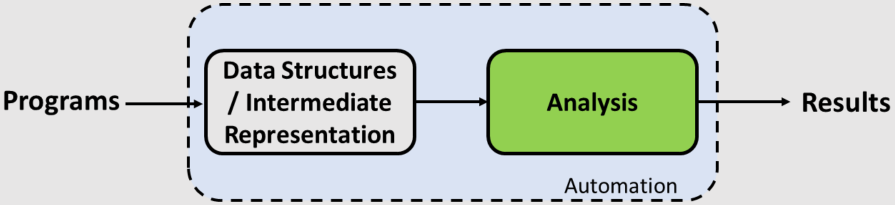
venkateshv@cmi.ac.in
Mar – Apr, 2018
Chennai Mathematical Institute



“The supply of grand challenges … shows little sign of drying up.”

– **Harman and O’Hearn in “Opportunities and Open Problems for Static and Dynamic Program Analysis”, Madrid, Spain, 2018**.

Quick Review



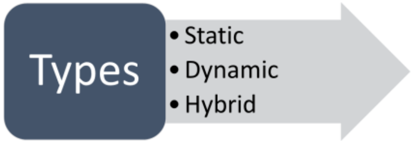
Programs

Data Structures / Intermediate Representation

Analysis

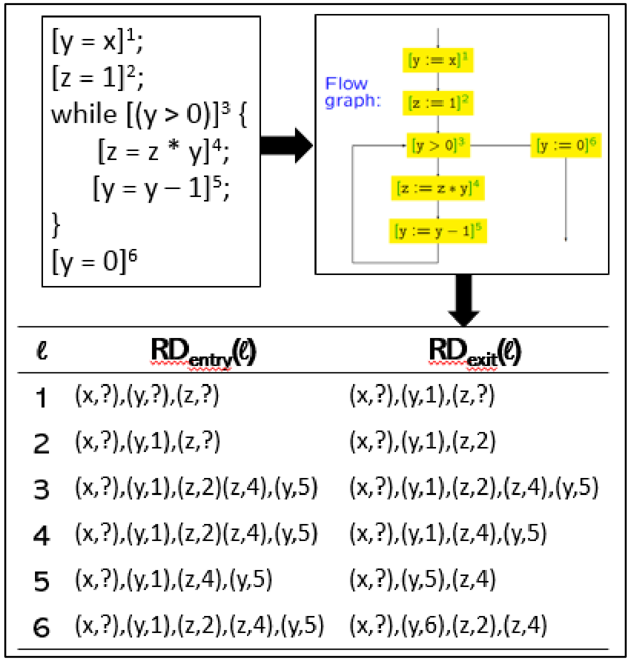
Results

Automation



Types

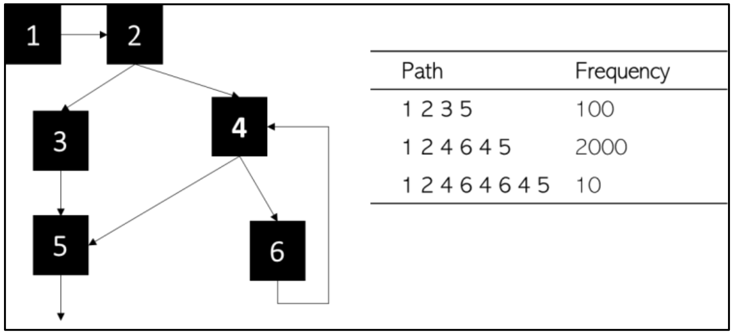
- Static
- Dynamic
- Hybrid



```
$[y = x]^1$;
$[z = 1]^2$;
while $[(y > 0)]^3$ {
      $[z = z * y]^4$;
     $[y = y - 1]^5$;
}
$[y = 0]^6$
```

ℓ	$RD_{entry}(\ell)$	$RD_{exit}(\ell)$
1	(x,?),(y,?),(z,?)	(x,?),(y,1),(z,?)
2	(x,?),(y,1),(z,?)	(x,?),(y,1),(z,2)
3	(x,?),(y,1),(z,2)(z,4),(y,5)	(x,?),(y,1),(z,2),(z,4),(y,5)
4	(x,?),(y,1),(z,2)(z,4),(y,5)	(x,?),(y,1),(z,4),(y,5)
5	(x,?),(y,1),(z,4),(y,5)	(x,?),(y,5),(z,4)
6	(x,?),(y,1),(z,2),(z,4),(y,5)	(x,?),(y,6),(z,2),(z,4)

Static Analysis



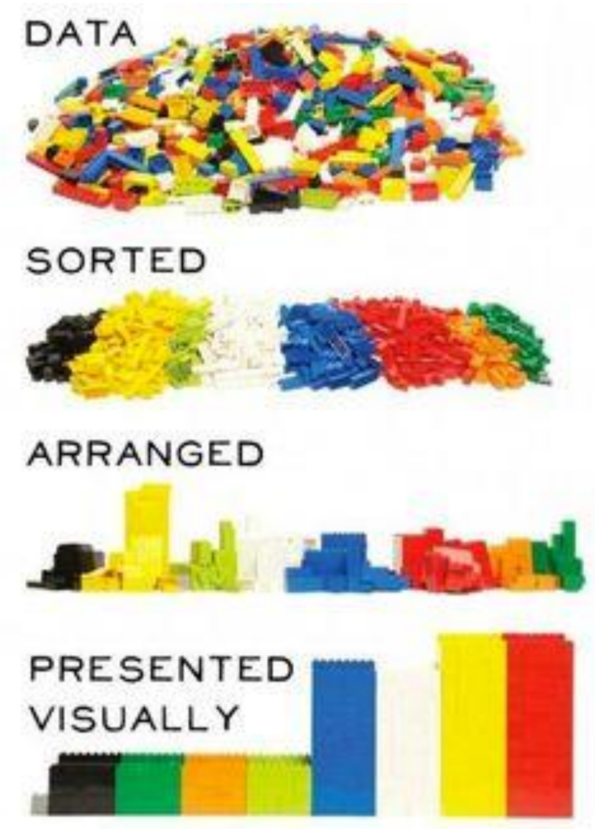
Path	Frequency
1 2 3 5	100
1 2 4 6 4 5	2000
1 2 4 6 4 6 4 5	10

Dynamic Analysis

Program Representations

Partially covered in *Sections 1.1, 1.2, 2.1, 6.2.1 and 6.2.4 of Dragon Book.*

Finding Order in Chaos



Source: https://visual.ly/community/infographic/how/what-infographic by Hot Butter Studio.

Agenda

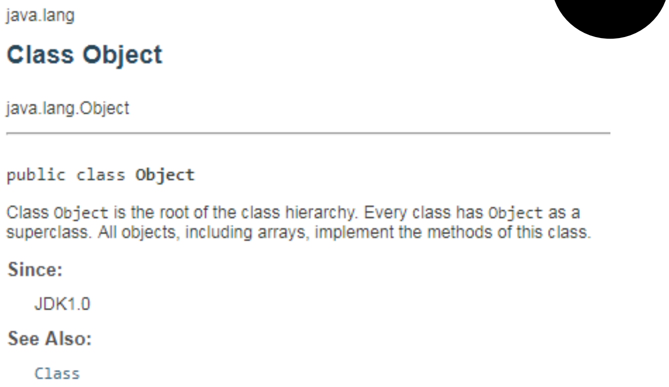
- Anatomy of a Program
  - Lexemes
  - Tokens
  - Abstract Syntax Trees (AST)
  - Intermediate Representations
- AST and CFG
  - Hands-On Session
- Popular Program Representations

Identify a Java Program

1

```
public class Fibonacci {

    public static void main(String[] args) {

        int n = 10, t1 = 0, t2 = 1;
        System.out.print("First " + n + " terms: ");

        for (int i = 1; i <= n; ++i)
        {
            System.out.print(t1 + " + ");

            int sum = t1 + t2;
            t1 = t2;
            t2 = sum;
        }
    }
}
```

2



java.lang

Class Object

java.lang.Object

`public class Object`

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

Since:

JDK1.0

See Also:

`Class`

3

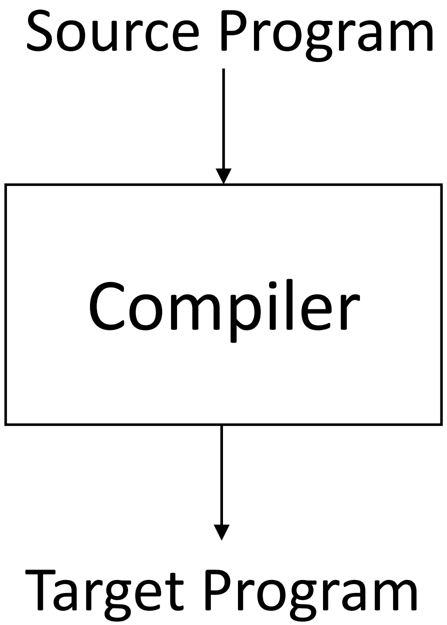
A Simple Grammar

Our programs are usually defined by a simple grammar…

a ::= x | n | a_1 op_a a_2
b ::= true | false | not b | b_1 op_b b_2 | a_1 op_r a_2
S ::= x := a | skip | S_1; S_2 |
        if (b) then S_1 else S_2 | while (b) do S

S refers to Statements, **a** is an Arithmetic Expression, and **b** is a Boolean Expression

Compiler

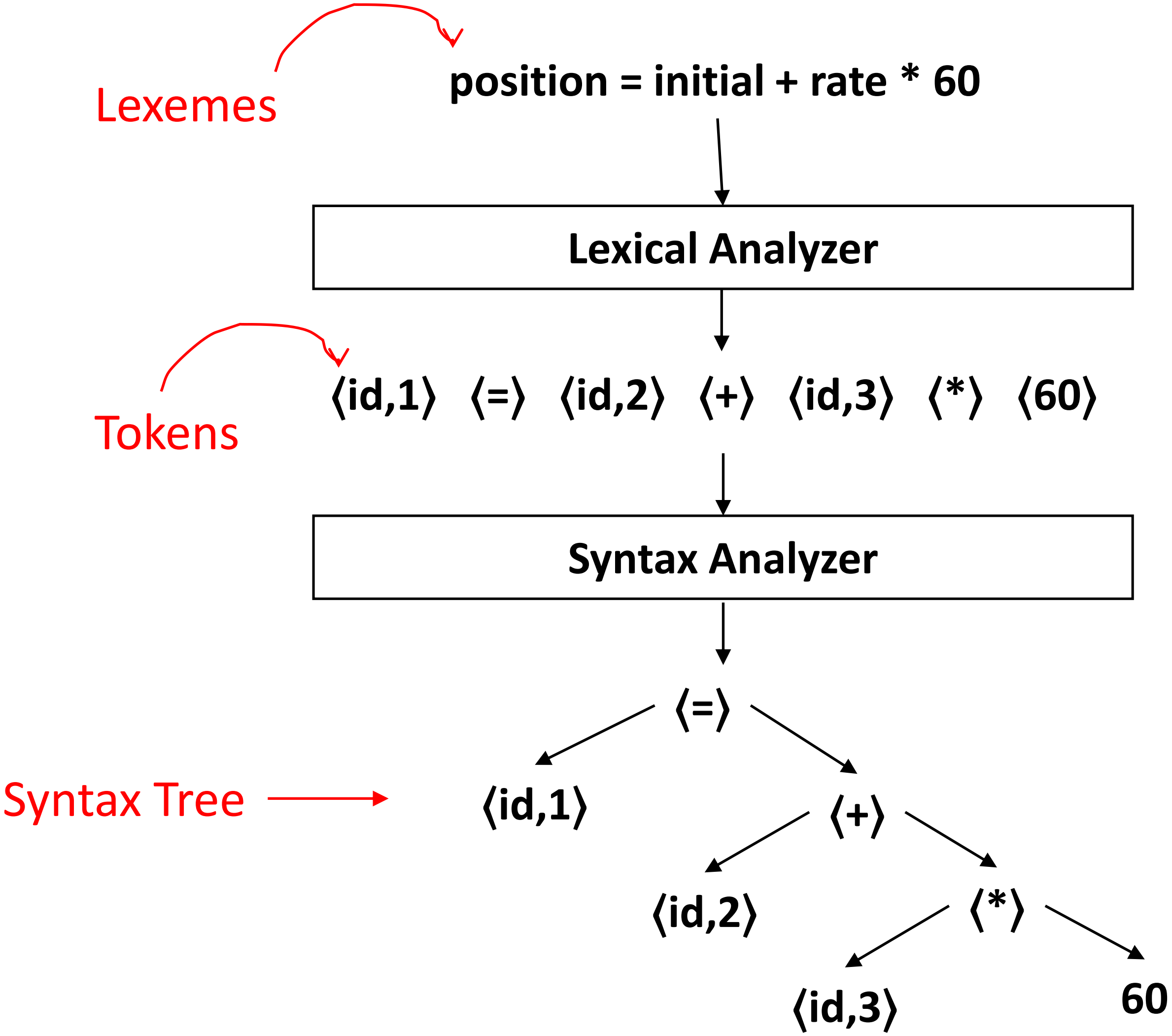


Source Program

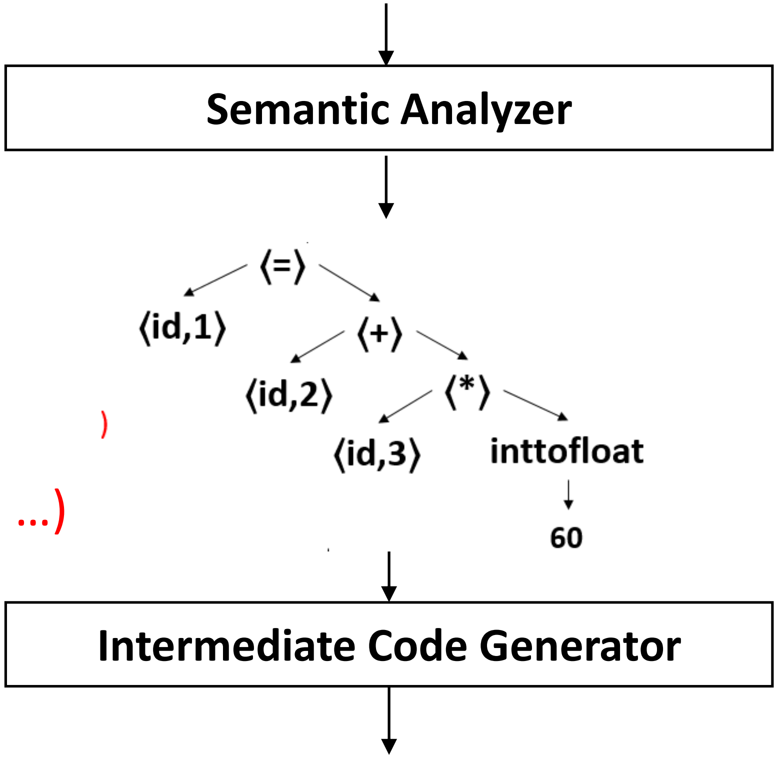
Compiler

Target Program

Structure of a Compiler



Structure of a Compiler…


Semantic Analyzer

Intermediate Code Generator

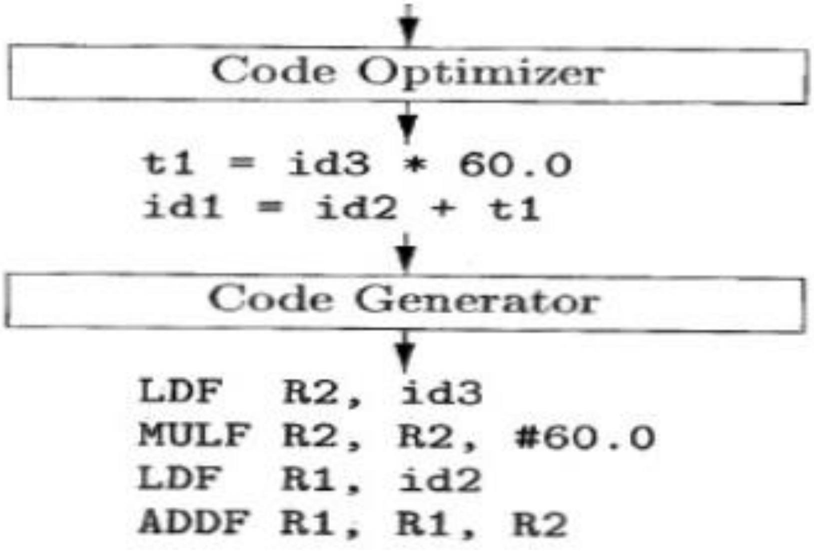
Type Checking →
(Undeclared variables,
Reserved Identifier Usage, …)

TAC and SSA →

```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

Structure of a Compiler

Program Analysis

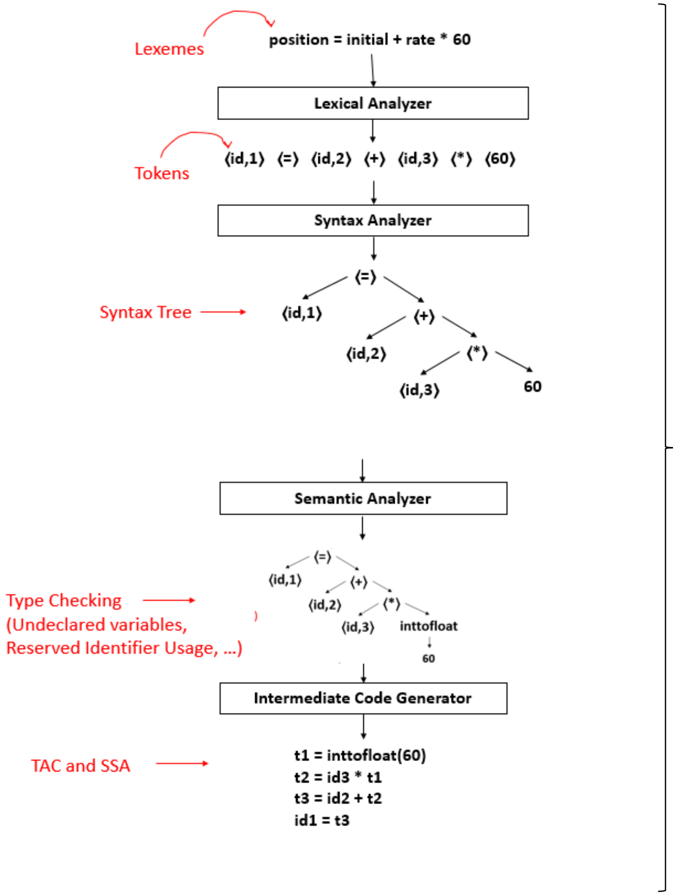


```
t1 = id3 * 60.0
id1 = id2 + t1
```

```
LDF  R2, id3
MULF R2, R2, #60.0
LDF  R1, id2
ADDF R1, R1, R2
```

“Structure of a Compiler”, taken from the dragon book.

Phases



```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

Present day compilers take multiple passes

They enrich the IR in each pass.

Our Interests

1. **Can we leverage the compiler infrastructure to analyze programs?**
2. **Can we improve compiler optimizations?**

What is the Output?

```
public class Test {
  public static void main(String[] args)
  {
      String str = "CMI";
      StringBuilder sb = new StringBuilder();
      for(int i = str.length() - 1; i >= 0; i--)
      {
        if (i == 0) sb.append('S');
        sb.append(str.charAt(i));
      }
      System.out.println(sb.toString());
  }
}
```

Graded Quiz: Parsing

- Is this grammar ambiguous? Why or Why not?

$$S \rightarrow AS \mid \epsilon$$
$$A \rightarrow A1 \mid 0A1 \mid 01$$

!GRADED!

Our First Analysis

using Eclipse JDT

Source Code as a Tree

- Abstract Syntax Tree

Code Snippet

```
while b ≠ 0
if a > b
a := a − b
else
b := b − a
return a
```

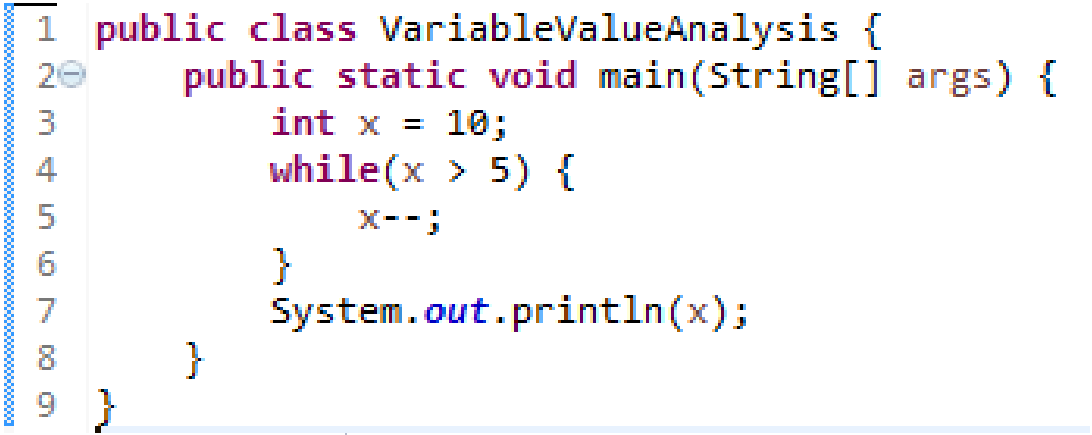
AST

- **while**
  - Op:**≠**
    - Var: **b**
    - Constant: **0**
  - Body
    - Branch
      - …
- Return
  - Var:**a**

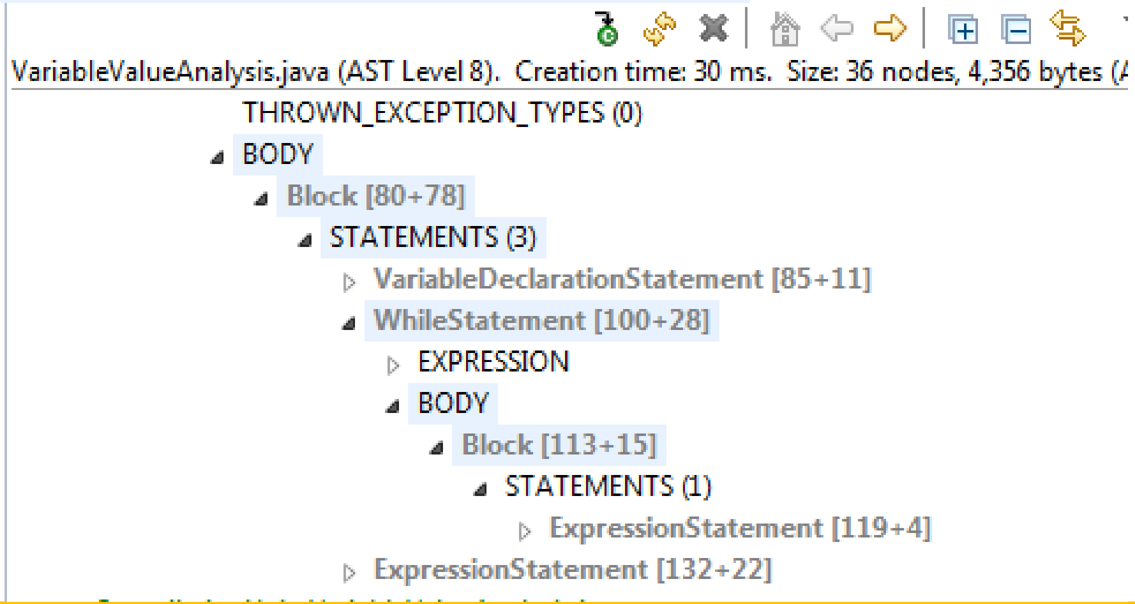
AST for Java Code

- Install Eclipse
- Install the Eclipse plugin AST View (from https://www.eclipse.org/jdt/ui/astview/index.php)
- Write any Java Code
- Follow the instruction in the AST View web page to view the AST.

An Example



```java
public class VariableValueAnalysis {
    public static void main(String[] args) {
        int x = 10;
        while(x > 5) {
            x--;
        }
        System.out.println(x);
    }
}
```



VariableValueAnalysis.java (AST Level 8). Creation time: 30 ms. Size: 36 nodes, 4,356 bytes (/

```
     THROWN_EXCEPTION_TYPES (0)
   ◢ BODY
     ◢ Block [80+78]
       ◢ STATEMENTS (3)
         ▷ VariableDeclarationStatement [85+11]
         ◢ WhileStatement [100+28]
           ▷ EXPRESSION
           ◢ BODY
             ◢ Block [113+15]
               ◢ STATEMENTS (1)
                 ▷ ExpressionStatement [119+4]
         ▷ ExpressionStatement [132+22]
```