PMD – STATIKUS KÓDELEMZŐ

NAGY BENDEGÚZ

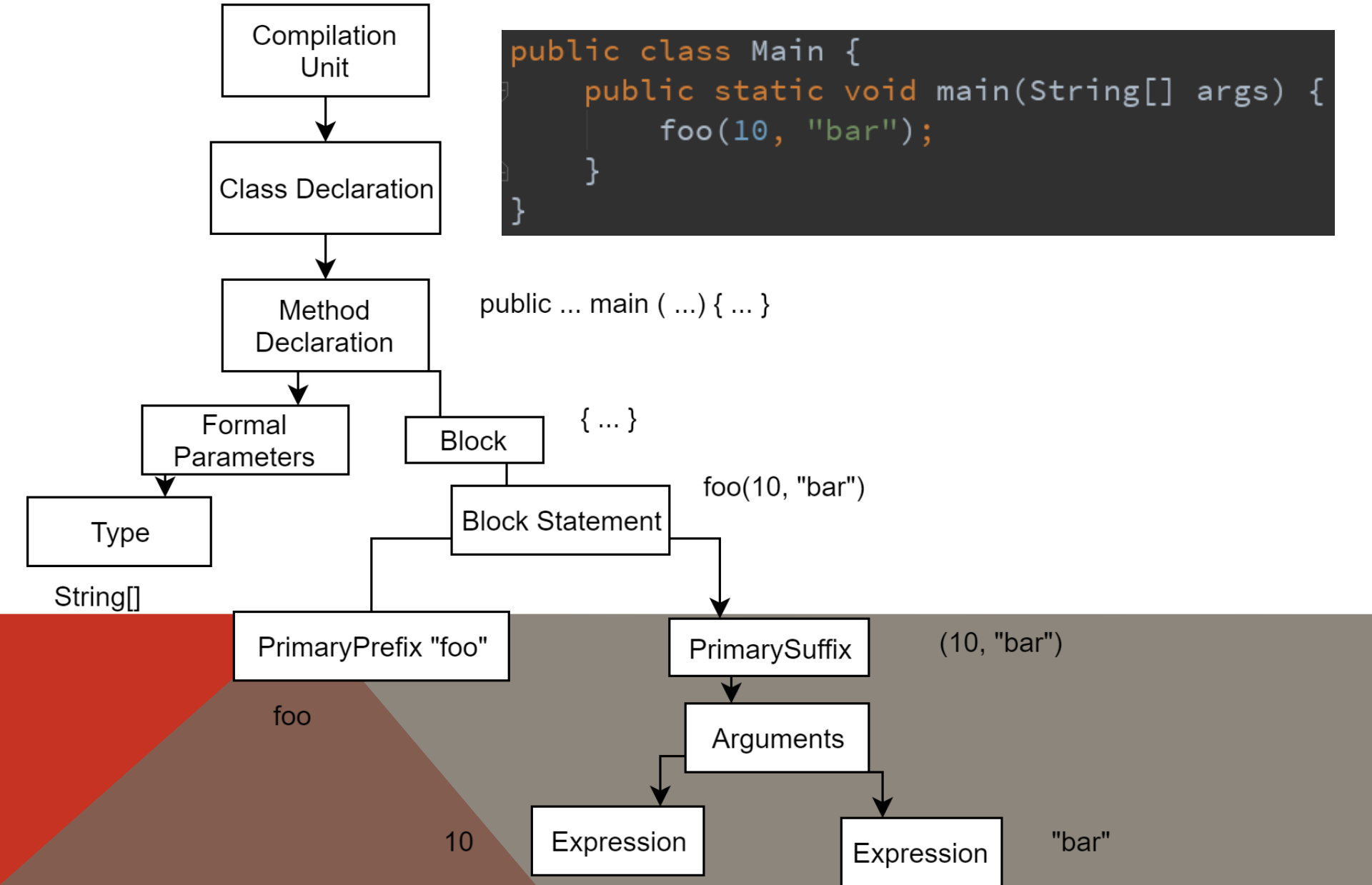
MI IS AZ A PMD?

- Hibakereső keretrendszer
- **Bővíthető** saját szabályokkal
- Több nyelvet támogat (Javáról lesz csak szó)
- Cross-platform
- Konzolból és gradle vagy maven pluginként is futtatható

HOGYAN MŰKÖDIK?

- Egyszerre csak egy forrásfájlt vizsgál
- Lebontja a forrásfájlt absztrakt szintaxis fára
- Magas szintű információt csatol a csomópontokhoz
- Lefuttatja a kapott fára a szabályokat

ABSZTRAKT SZINTAXIS FA




```java
public class Main {
    public static void main(String[] args) {
        foo(10, "bar");
    }
}
```

AMIT ÉN CSINÁLTAM

Ha megvan a szintaxis fa, jó lenne tudni például, hogy egy mezőelérésnek mi a típusa…

Részben implementált, nagyon hiányos volt

- Mező elérés, generikus típusú mező elérés
- Metódus hívás
- Metódus referencia
- this, super, qualified this kifejezések
- Generikus metódus hívások (még nincsen készen)

JAVA LANGUAGE SPECIFICATION

15.12.2.2. Phase 1: Identify Matching Arity Methods Applicable by Strict Invocation

An argument expression is considered *pertinent to applicability* for a potentially applicable method `m` unless it has one of the following forms:

- An implicitly typed lambda expression (§15.27.1).
- An inexact method reference expression (§15.13.1).
- If `m` is a generic method and the method invocation does not provide explicit type arguments, an explicitly typed lambda expression or an exact method reference expression for which the corresponding target type (as derived from the signature of `m`) is a type parameter of `m`.
- An explicitly typed lambda expression whose body is an expression that is not pertinent to applicability.
- An explicitly typed lambda expression whose body is a block, where at least one result expression is not pertinent to applicability.
- A parenthesized expression (§15.8.5) whose contained expression is not pertinent to applicability.
- A conditional expression (§15.25) whose second or third operand is not pertinent to applicability.

Let `m` be a potentially applicable method (§15.12.2.1) with arity *n* and formal parameter types F_1 ... F_n, and let e_1, ..., e_n be the actual argument expressions of the method invocation. Then:

- If `m` is a generic method and the method invocation does not provide explicit type arguments, then the applicability of the method is inferred as specified in §18.5.1.
- If `m` is a generic method and the method invocation provides explicit type arguments, then let R_1 ... R_p ($p \geq 1$) be the type parameters of `m`, let B_l be the declared bound of R_l ($1 \leq l \leq p$), and let U_1, ..., U_p be the explicit type arguments given in the method invocation. Then `m` is *applicable by strict invocation* if both of the following are true:
  - For $1 \leq i \leq n$, if e_i is pertinent to applicability then e_i is compatible in a strict invocation context with $F_i[R_1:=U_1, ..., R_p:=U_p]$.
  - For $1 \leq l \leq p$, $U_l <: B_l[R_1:=U_1, ..., R_p:=U_p]$.
- If `m` is not a generic method, then `m` is *applicable by strict invocation* if, for $1 \leq i \leq n$, either e_i is compatible in a strict invocation context with F_i or e_i is not pertinent to applicability.

If no method applicable by strict invocation is found, the search for applicable methods continues with phase 2 (§15.12.2.3).

Otherwise, the most specific method (§15.12.2.5) is chosen among the methods that are applicable by strict invocation.

The meaning of an implicitly typed lambda expression or an inexact method reference expression is sufficiently vague prior to resolving a target type that arguments containing these expressions are not considered pertinent to applicability; they are simply ignored (except for their expected arity) until overload resolution is finished.

PÉLDA: METÓDUSHÍVÁS TÍPUSA

```
public class MethodFirstPhase {
    void test() {
        int a = foo( a: 10,  b: 'a',  c: "");
    }

    Exception foo(long a, int b, String c,
                  Number... d) {...}

    Exception foo(short a, int b, String c) {...}

    int foo(long a, int b, String c) {...}

    Exception foo(long a, byte b, String c) {...}
}
```

Google

Summer of Code