Statikus kódellenőrzés Java-ban

JUM

2009.09.16

Mai téma:

- AbstractProcessor
- Checkstyle
- PMD

- Saját szabályok
- Konzolból (+maven)
- Eclipse alatt

Bevezetés

- kód konvenciók betartása
- kód ellenőrzés futtatás nélkül
- vannak kedvenc hibalistánk
- egyszer létrehozzuk a szabályt és onnantól él
- más fejlesztési tapasztalat, más kód
- mindenki hibázik
- alap warningok figyelése (?)

Bevezetés

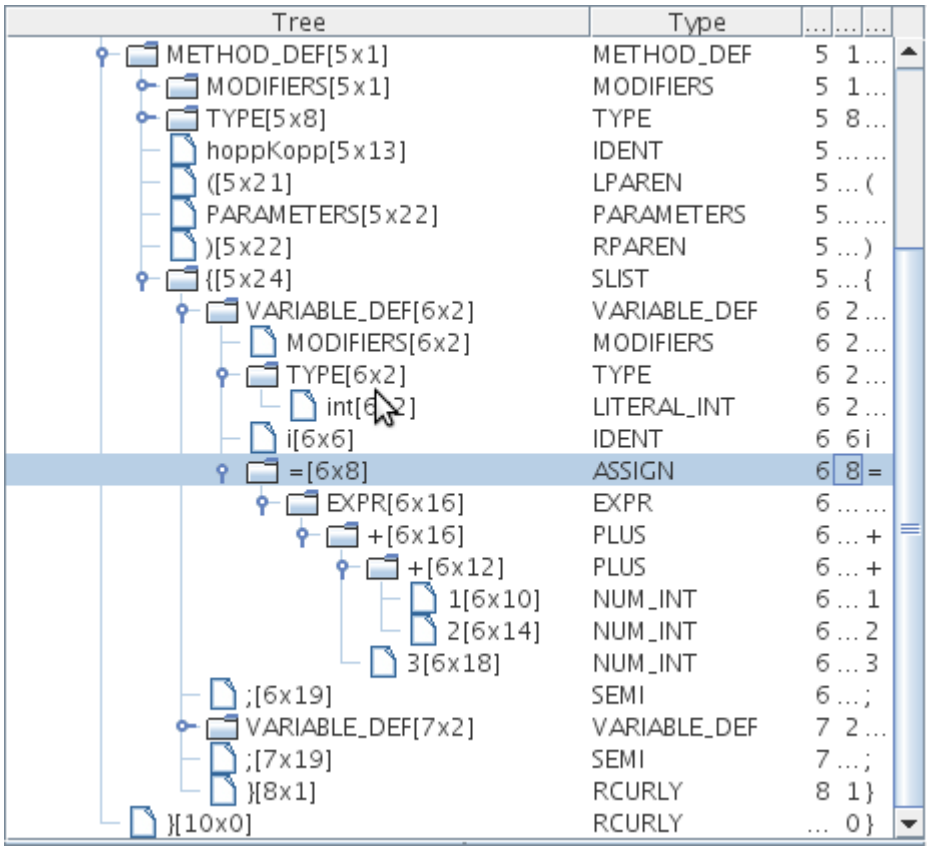
- Forráskód vizsgálata, hogyan?
  - Reguláris kifejezéssel
  - Nincs struktúrális fa
- IDE eszközök tudják
  - Nem szabható testre
  - Nem bővíthető
  - Mintha fordítanának
  - AST: abstract syntax tree

Bevezetés

- Írási módok
  - Prefix
    - +1*23: 1+(2*3)
    - *1+23: 1*(2+3)
    - +*123: (1*2)+3
    - *+123: (1+2)*3
  - Postfix
    - 32*1+

AST

```
public void hoppKopp()
{
        int i = 1 + 2 + 3;
        int j = 1 - 2 + 3;
}
```



Kályha

- Java 6.0 Compiler API
  - új ‘Java Compiler API’ 6.0-ban
  - korábban non-standard: com.sun.javac.tools.javac
  - java fájlok fordítása java fájlból
  - ${JDK_DIR}/lib/tools.jar

javax.tools package (JDK6)

- JavaCompiler
- CompilationTask
- DiagnosticListener (errors, warnings, line number, …)
- AbstractProcessor
- TreePathScanner

CodeCompile DEMO

Checkstyle

- http://checkstyle.sourceforge.net
- ANTLR
- Konfigurációval szabályozható
- Modulok
- TreeWalker AST-t készít

Checkstyle DEMO

PMD

- http://pmd.sourceforge.net/
- JavaCC
- EBNF
- kezdetben csak kód mint szöveg ellenőrzés
- AST-vel már több:
  - Osztály szintű tervezés
  - Duplikált kód
  - Bug minták

Ennyi :-)