IntelliJ IDEA - Die bessere Java IDE

FS info
Fachschaft für Informatik

- kommerzielle Java IDE seit 2001
- Aufteilung in Open Source Community Edition und kommerzielle Ultimate Edition in 2009
- Verschiedene spezialisierte Ableger (PhpStorm, WebStorm, PyCharm, ...)
- AndroidStudio (Google's neue Android IDE) basiert auf IDEA

FS info
Fachschaft für Informatik

Features

- Language Support (Scala, Ruby, Python, Bash, PHP, ...)
- Framework Support (JSF, Spring, AngularJS, Play, ...)
- Database Tools (DB Editor, SQL Support)
- VCS Integration für alle populären VCS Systeme
- Sehr gute Dokumentation für die gesamte IDE
- Keymaps (Eclipse, Visual Studio, Net Beans, ...)
  für die gewohnten Hotkeys

FS info
Fachschaft für Informatik

Woher

- An den Uni Rechnern installiert (starten mit `idea`)
- Academic License über Herrn Schießl (R 125 IM) erhältlich

Wo anfangen

- IntelliJ IDEA Getting Started
  `http://www.jetbrains.com/idea/documentation/index.jsp`
- IntelliJ IDEA Q&A for Eclipse Users
  `http://www.jetbrains.com/idea/documentation/migration_faq.html`

FS info
Fachschaft für Informatik

- Plugin für IntelliJ IDEA zur Codeanalyse
- Hilft Fehler / schlechten Code zur Entwicklungszeit zu finden
- Fasst Meldungen von PMD, FindBugs, Checkstyle, Hammurapi zusammen*
- Jedes Tool hat besondere Stärken und Schwächen
- **Aber** ist kein Allheilmittel für schlechten Code

FS info
Fachschaft für Informatik

Woher & Was

- Beinhaltet folgende Komponenten
  - PMD `http://pmd.sourceforge.net/` - Findet primär ineffizienten/toten/doppelten Code
  - FindBugs `http://findbugs.sourceforge.net/` - Sucht nach bekannten Fehlermustern
  - Checkstyle `http://checkstyle.sourceforge.net/` - Prüft Programmierstil
  - Hammurapi `http://www.hammurapi.biz` - Identifiziert Probleme und übliche Fehler
- Fazit: Sehr mächtiges Tool um Fehler zu finden
- Für eclipse müssen die Tools einzeln installiert werden

- Basiert auf einem statischen Regelwerk
- Findet unter anderem folgende Probleme
  - Mögliche Bugs (z.B. leere Blöcke in `try`,`catch`, `switch`, ...)
  - Toter Code (nicht genutzte Variablen)
  - Schlechter Code (z.B. Stringkonkatenationen, statt Buffer)
  - Verkomplizierter Code (`for`-Schleifen, die auch durch `while` könnten)
  - Doppelter Code

FindBugs

- Findet Probleme durch statische Codeanalyse im Bytecode
- Fehlerklassen: scariest, scary, troubling, of concern
- Beispiele für gefundene Probleme
  - Ungültiger Syntax in regulären Ausdrücken
  - Assertion in `run()`-Methode wird von JUnit nicht erkannt
  - Mögliche Nullpointer-Dereferenzierung
  - Fehlerhafter Cast
  - Lesen von Null aus nicht geschriebenen Variablen

Checkstyle

- Bekannt aus ProgII
- Manchmal nervig, aber dennoch notwendig
- Markiert den schlechten Umgang mit:
  - JavaDoc
  - Einhaltung von Metriken
  - Codestil (Zeilenlänge, Nameingconventions)
  - Verwendung von Imports
  - Codekomplexität

Hammurapi

FS info
Fachschaft für Informatik

Hammurapi Group
Java tools and libraries

- Analyse des Quelltextes
- Idee: QA für Outsourcing-Projekte
- hebt unter anderem folgende Probleme hervor:
  - Vergleich von Objekten mit == statt `equals`
  - Exceptionketten (Exceptions werden umgelabelt)
  - innere Klassen sollten `private` sein
  - Unnötiger Verwendung eines cast oder `instanceof`
  - Loggen von Fehlern mit `System.out` oder `System.err`