Clean Code für Java Entwickler

Seminardauer: 3 Tage, 09:00 Uhr bis 17:00 Uhr
Schulungsunterlagen: nach Absprache

Der Begriff ?Clean Code? hat seinen Ursprung aus dem gleichnamigen Buch von Robert C. Martin. Der Autor beschreibt dort Prinzipien und Praktiken für professionelle Softwareentwickung mit Java.
Nach dieser 3-tägigen Schulung kennen Sie die Methoden und Praktiken, um hochwertigen Code zu schreiben. Sie lernen, wie Sie qualitativ gute Software von schlechter unterscheiden und was hochwertige Software kennzeichnet. Sie lernen Tipps und Tricks für die schnellere, individuelle Entwicklung und für eine bessere Lesbarkeit, Wartbarkeit und Testbarkeit Ihrer Software.
Zusätzlich führt die Schulung in weiterführende Themen wie ?Design Patterns?, ?JUnit-Tests? und ?Statische Codeanalyse-Tools? ein.

Zielgruppe

Das Seminar richtet sich an Softwareentwickler und Qualitätsmanager aus dem Java Umfeld.

Kurs Voraussetzungen

Gute Programmierkenntnisse in Java.

Agenda

Der Softwareentwicklungsprozess

Was ist Software-Qualität

Guter und schlechter Code

- Welche Auswirkung hat schlechter Code
- Hinweise auf schlechten Code (Code Smells)
- Technische Schulden, was ist den das
- Code-Qualität messen, damit es kein Blindflug wird

Warum Clean Code?

- Was ist Clean Code
- Prinzipien und Praktiken

Prinzipien

- Code Prinzipien (Guter Code)
  - Aussagekräftige Namen
  - Klassen und Funktionen
  - Lesbarer Code vs. Kommentare
  - Code-Formatierung
- Design Prinzipien (Gutes Design)
  - Kapselung
  - Kohäsion
  - Kopplung
  - You aint gonna need it (YAGNI)
  - Open / Closed Prinzip (OCP)
  - Tell dont ask
  - Low of Demeter
  - Interface Seggregation Prinzip (ISP)
  - Dependency Inversion Prinzip (DIP)
  - Liskovsches Substitutionsprinzip (LSP)
  - Principle of Least Astonishment
  - Information Hiding Principle (IHP)

- Single Level of Abstraction (SLA)
- Single Responsibility Prinzip (SRP)
- Seperation of Concerns (SoC)
- Dont Repeat Yourself (DRY)
- Keep It Simple, Stupid (KISS)
- Favour Composition over Inheritance (FCol)

Praktiken

- Continuouse Integration (CI)
- Iterative Entwicklung
- Komponentenorientierung
- Test First
- Statische Codeanalyse
- Inversion of Control Container
- Erfahrungen weitergeben
- Messen von Fehlern
- Automatisierte Unittests
- Mockups
- Code Coverage Analyse
- Teilnahme an Fachveranstaltungen
- Komplexe Refaktorisierungen
- Einfache Refaktorisierungen
- Issue Tracking
- Automatisierte Integrationstests