UNIVERSITÄT SALZBURG

PS Software Engineering

WS 2019/20

Andreas Naderlinger

andreas.naderlinger@sbg.ac.at

Fixplatz vs. Warteliste

Allgemeines

- Wöchentlich
  - Mittwoch 12:00 - 14:00

Start: 12:15

- Termine: PLUSonline

- Homepage zum PS:

www.softwareresearch.net → Teaching

“Programmieren im Großen”

- Die Entwicklung von kleinen Programmen unterscheidet sich fundamental von der Entwicklung großer Programme.

- Große Programme zu entwickeln heißt nicht „nur“ mehr Code schreiben zu müssen.

- Somit nicht bloß durch mehr Manpower zu beherrschen!

Unterschiedliche Aufgaben

- Kleine *Programme*
  - “Coden” und “Ausprobieren”
  - “… habe ich schnell mal programmiert”

- Mittlere bis (sehr) große *Softwaresysteme*
  - Analyse
  - Spezifikation
  - Entwurf
  - Implementierung
  - Validierung
  - Wartung
  - Dokumentation
  - Qualitätssicherung
  - Projektmanagement

Hier im PS:
Kleine aber sehr, sehr ‘saubere’ Programme!
Idee:
Unsere kleinen Komponenten (ohne Änderung) in große Softwaresysteme integrieren.

Allgemeines (2)

- Anwesenheitspflicht !
- 2 **Tests**
- **Assignments**: Wöchentliche Aufgaben
  - Selbständiges Erarbeiten von Hintergrundwissen
  - Praktische (Programmier-)Übungen: **Java,**
    (2. Teil: Python), …

    …knapp 2 Wochen Zeit

    …in 2er-Teams (meistens)
  - Voraussichtlich insgesamt ca. 9 (Prog.-)Aufgaben A1,A2,…

Aufgaben

- Abgabe jeweils vor dem PS
  - **Deadline Di 12:00 (Mittag)**
- Abgabe
  - A0: keine Abgabe
  - A1: per Mail
  - A2,…: Github classroom (Info folgt)

Beurteilung

- 30% Tests
- 70% Assignments + MA

- Für eine positive PS-Note müssen BEIDE Teile positiv sein
  - D.h. SOWOHL mind. 50% der maximalen Testpunkte ALS AUCH mind. 50% der max. Assignment-Punkte.

PS Ablauf

- Befragung zu Hintergrundwissen
- Präsentation der Lösung
  - Theoretischer Überblick (z.B. zum konkreten Pattern)
  - Klares Herausarbeiten der wesentlichen Aspekte, Design-Überlegungen
  - Vor-/Nachteile im Vergl. zu möglichen Alternativen
  - Wichtige Code Passagen (keine Scroll-Down-Demo)
  - Eventuell Vorführung der Anwendung
- Bewunderung ernten bzw. Kritik gefallen lassen
- Code-Review (in Partnerarbeit)

- What I expect you to know
  - **Java / OO**
    - **Inheritance**
    - **Polymorphism**
    - **Static/dynamic type**
    - **Encapsulation, Information Hiding**
    - **Exception handling**
- What I believe you already know too
  - Generics
  - JavaDoc

Topics covered in past years

- OO Design (Inheritance, Interfaces, Polymorphism, …)
- Design Patterns (Composite, Iterator, Decorator, Visitor, Observer, MVC)
- UML
- Best practices (Exception handling, …)
- Versioning (SVN/Git)
- TestDrivenDevelopment: JUnit
- ~~Threads / Synchronization (System- vs. Application Programmer)~~
- Java Collection Framework
- Java Streams
- Generics
- GUI: Java Swing, JavaFx
- XML
- Reflection
- Machine Learning

Lernziele

- Herangehensweise: **„ingenieurmäßige“ Entwicklung** statt Quick-Hack
  - Weg von *„schau ma moi“*-, *„aber es funktioniert“*-, *„eigentlich würde man hier besser ….“*-Mentalität

- Kenntnisse (Begriffe/Methoden/Werkzeuge)
  - DP, UML, Java Libs, UnitTests
- Verständnis
  - Auswirkungen von DP, Welches DP wann sinnvoll?
- Anwendung
  - Code <--> UML, Debugging, Versionierung
- Analyse
  - Probleme in Code erkennen, Anwendung in Komponenten zerlegen
- Synthese
  - Ein Projekt „ingenieurmäßig“ umsetzen (erweiterbar, modular, robust, korrekt, testbar, flexibel, lesbar, wartbar)
- Beurteilung
  - Codequalität einschätzen, Annahmen für fehlende Spez. treffen;

Teams of 2

Assignments A0, A1

- Note: no teamwork for assignment A0 and A1

- A0

- A1
  - Inspiration: http://minesweeperonline.com/

Some Remarks

- Main method
  - Put almost nothing in there
- JavaDoc
  - Use it for your public API
- Visibility modifier
  - **private** stuff leads to less required docu ;-)
- Naming conventions
  - MyClass, myField, myMethod, MY_CONSTANT, …
- Exception handling

Thanks.