Objektorientiertes Programmieren

(formerly known as *Programmieren 2*)

Bachelor Wirtschaftsinformatik

Marcel Tilly

Fakultät Informatik, Cloud Computing

Organisatorisches

Moodle: **Objektorientierte Programmierung - WIF SS 2019**

- Selbsteinschreibung 'wif-oop-ss19'

Mattermost: **https://inf-mattermost.fh-rosenheim.de/wif-oop-ss19**

Gitlab: **https://inf-git.fh-rosenheim.de/wif-oop-ss19**

Übungen:

- Dienstags, 2./3./4. Stunde
- Raum: S1.31
- Tutor: Daniel Herzinger

Daniel Herzinger

This is Daniel!

Organisatorisches

Leistungsnachweis:

- **Benotete** schriftliche Prüfung (90 Minuten)
- zusätzlich: **Coding Contest**

Wichtige Termine:

-?. April: Prüfungsanmeldung im OSC
-13. Mai: Einführung in das Contestsystem (persönliche Anwesenheit erforderlich!)

Organizatorisches

Ablauf

- 2 SWS Vorlesung (Montags 11:45) in A3.13
- 2 SWS Übung (Dienstags, 3 Gruppen, mit Tutor) in S1.31

Literatur

- Offizielle Java Dokumentation
- Ullenboom, C: Java ist auch eine Insel, 2017. (Online verfügbar!)
- Bäckmann, M: Objektorientierte Programmierung für Dummies, 2002. *Das Buch ist für C++, die Methodik aber identisch zu Java.*
- Gamma, E et al.: Design Patterns, 1994. (Das Buch ist in englischer und deutscher Fassung in der Bibliothek vorhanden).

Lernziele

- Vertiefung der objektorientierten Programmierung
  - Vererbung
  - Abstrakte Basisklassen
  - Entwurfsmuster (Design Pattern)
- Abstrakte Datentypen
- Algorithmik:
  - Sortieren
  - Rekursion
  - parallele Verarbeitung
- Grundlagen professioneller Softwareentwicklung
  - Modellierung (Entwurf)
  - Versionierung
  - Testen

Tips zu den Übungen

Klären, *was* eigentlich zu tun ist

Die Angaben sind auf Gitlab, lesen Sie die Readme sorgfältig durch.

Festlegen, *wie* die Aufgabe zu lösen ist

- Entwerfen Sie eine Lösungsskizze -- Papier und Stift sind Ihre Freunde!
- Beschreiben Sie Algorithmen in kleinen, ausführbaren Schritten
- Identifizieren Sie Spezialfälle

Tips zu den Übungen

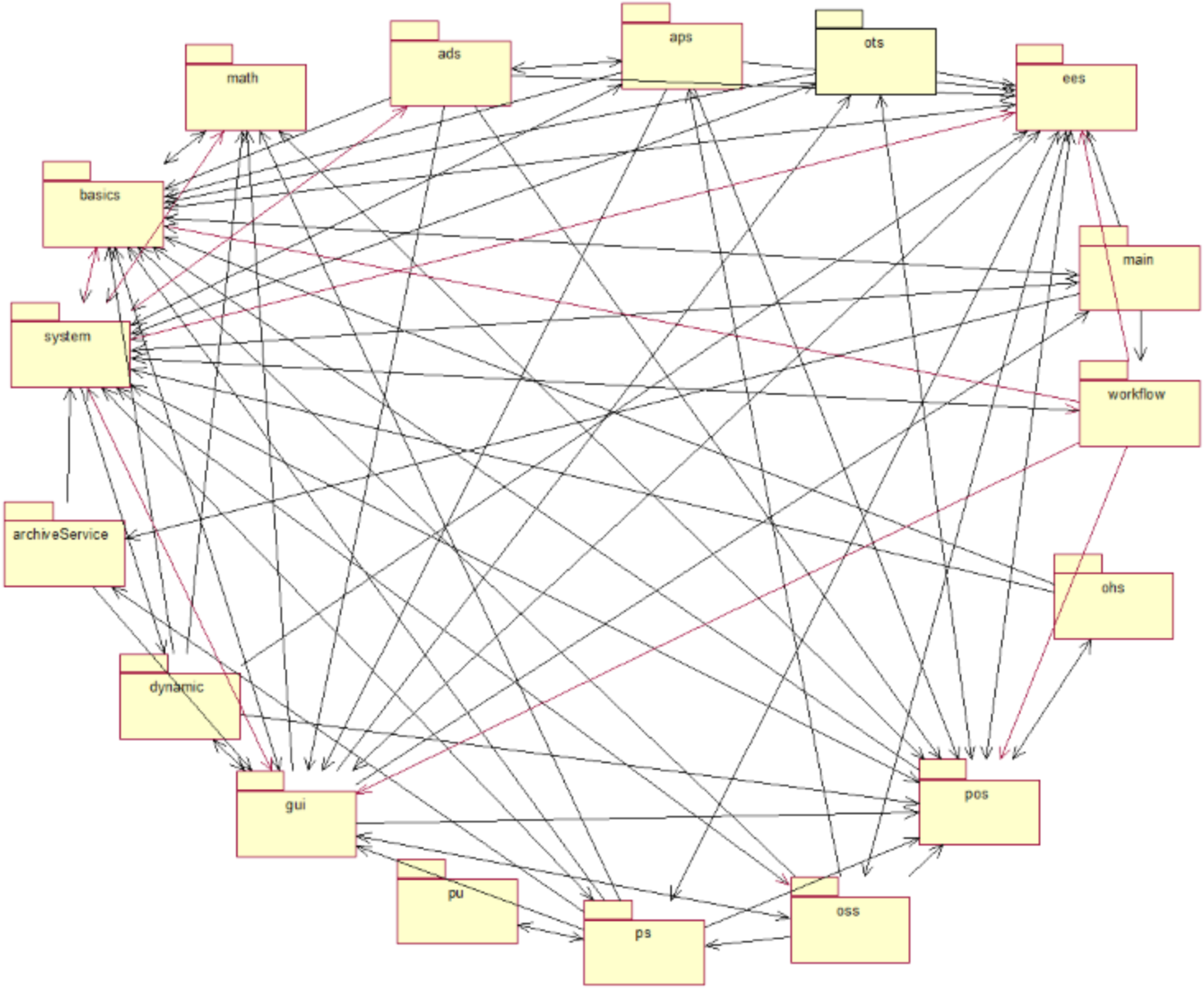
Umsetzen des textuellen Algorithmus in Java

- Arbeiten Sie die Beschreibung Schritt für Schritt ab
- Fügen Sie Kommentare ein, wo der Code nicht selbstverständlich ist

Testen

- Verwenden Sie JUnit um Ihr Programm mit vorgegebenen Eingaben zu testen.
- Erweitern Sie die Tests um weitere Ein- und Ausgaben.

Motivation: "Bad design smells!"



Bad Design: Wie kann das passieren?

Problem

- Zyklische Abhängigkeiten
- Keine klare Struktur

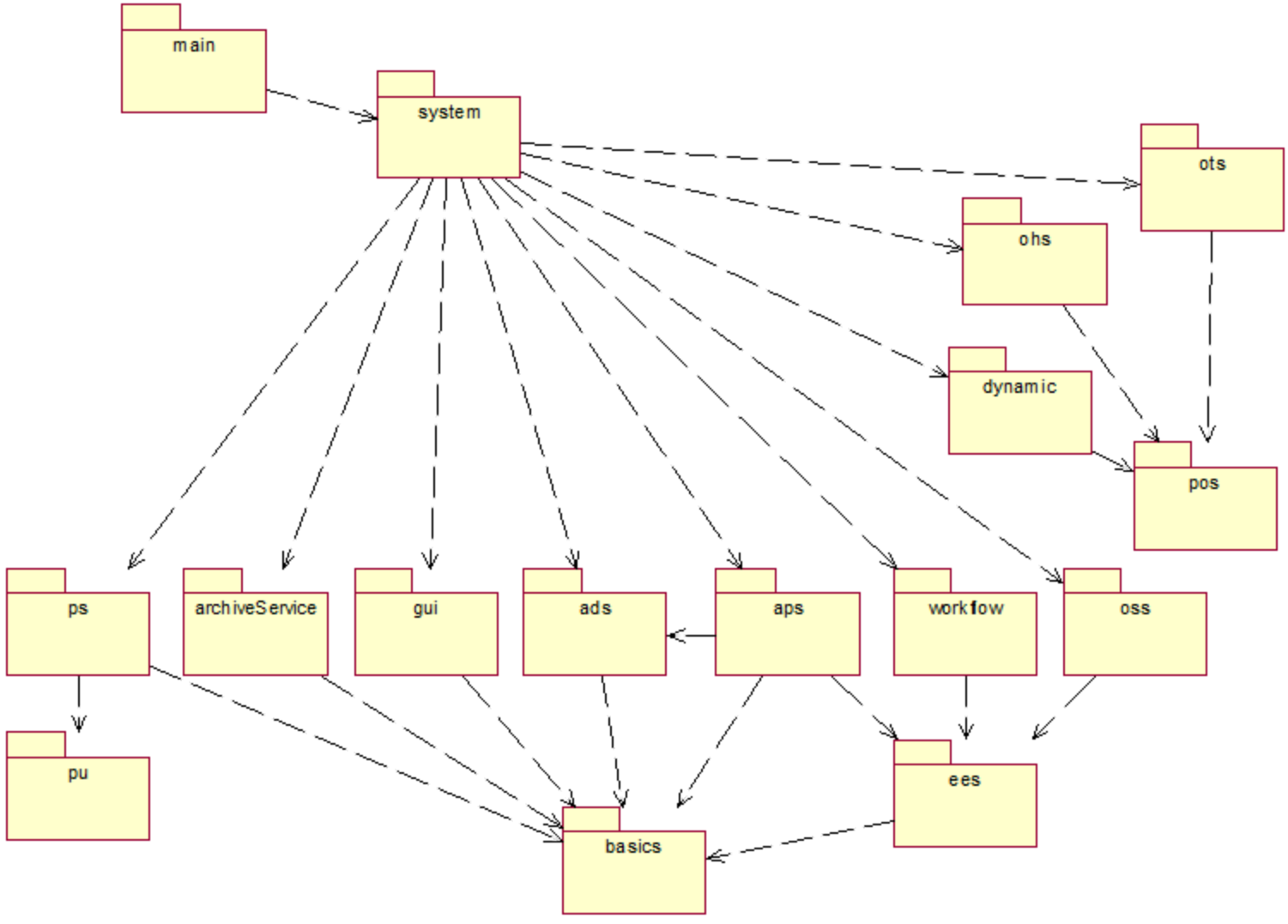
Ursachen

- Historisch gewachsen
- Viele Änderungen/ ohne Design
- adhoc
- Keine klaren Verantwortlichkeiten
- Unklarer Prozess

Effekt

- Monolithisch
- Nicht wartbar
- Nicht wiederverwendbar
- Ineffizient
- Schwer zu testen
- Nicht verlässlich

Good Design


main
system
ots
ohs
dynamic
pos
ps
archiveService
gui
ads
aps
workflow
oss
pu
ees
basics

Why does it matter?

Klare Struktur – klare Sprache

- Eindeutige Abhängigkeiten
- modular

Effekt

- Definierte Verantwortlichkeiten
- Einfachere Wartung
- Einfachere Änderungen
- Effizienter
- Modular: Besser zu testen

Anforderungen an Software

- **Korrektheit** (Correctness): Die Software erfüllt die Anforderungen
- **Einfache Handhabung** (Usability): Nutzer können das System problemlos nutzen
- **Robustheit** (Robustness): Software reagiert angemessen bei abweichenden Bedingungen
- **Erweiterbarkeit** (Extendable): beschreibt, wie leicht Software erweitert werden kann
- **Wiederverwendbarkeit** (Reuseable): Software (Elemente) kann für anderen Anwendungen wiederverwendet werden
- **Vereinbarkeit** (Composability): Wie leicht Software (Elemente) miteinander kombiniert werden können

- **Effizienz** (Efficiency): Möglichst wenig Anforderungen an die

Fragen?