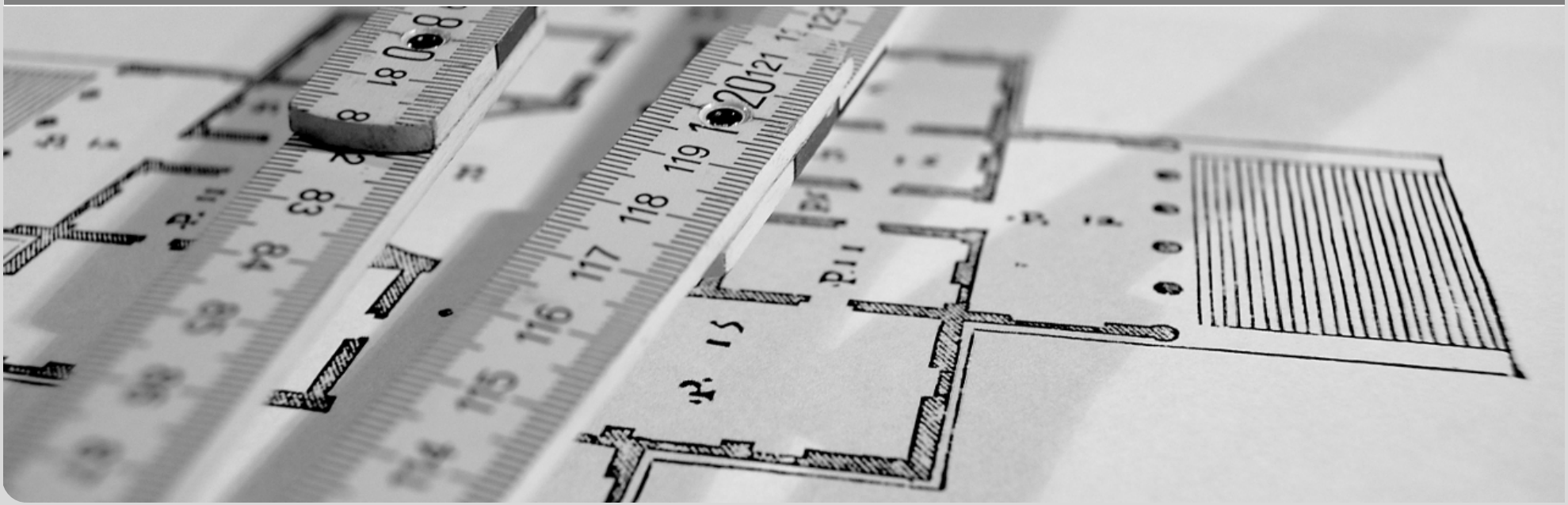
Karlsruher Institut für Technologie

SDQ Software Design and Quality

Programmieren
12. Tutorium

Robin Rüde | 9. Februar 2015



Gliederung

1

Organisatorisches

Zu Blatt6 und Abschlussaufgaben

- Trennung von Einlesen und Speichern:
  Die Datenquelle sollte austauschbar sein
- Funktionen kurz halten. 80 Zeilen sollten nie nötig sein.
- Objektattribute nicht als Cache verwenden.
- Mögliche Fälle nicht hardcoden. Codeduplikation vermeiden.
  Schleifen verwenden.
  Beispiel Blatt 6C:

```
if (lines2[i].split("=")[0].equalsIgnoreCase("title") && bsp.getTitle().equals("unknown")) {
  bsp.setTitle(lines2[i].split("=")[1]);
} else if (lines2[i].split("=")[0].equalsIgnoreCase("year") && bsp.getYear().equals("unknown")) {
  bsp.setYear(lines2[i].split("=")[1]);
} else if (lines2[i].split("=")[0].equalsIgnoreCase("creator") && bsp.getCreator().equals("unknown"
  bsp.setCreator(lines2[i].split("=")[1]);
} else {
  Terminal.printLine("Error, invalid book!");
}
```

Zu Blatt6 und Abschlussaufgaben

- Schachtelungstiefe gering halten. Mehr als 4 Ebenen sollten nie nötig sein.
- Weitere Designprinzipien aus der Vorlesung beachten
- Beim erstmaligen Schreiben nicht zu sehr auf Struktur achten. Lieber lösen und nachträglich darüber nachdenken / umstrukturieren.
- Sachen mit Refactoring umbenennen, nicht manuell. JavaDoc konsistent halten

2

Objektorientierte Designprinzipien

Objektorientierte Designprinzipien

Datenkapselung

Minimierung der Zugriffsmöglichkeiten auf Klassen und Attribute.
Zum Beispile Durchsetzung von Einschränkungen oder Unabhängigkeit von internen Implementierungen

Komposition statt Vererbung

Wiederverwendung durch Zusammensetzen besthender Objekte zu einem Objekt mit erweiterter Funktionalität.
+ Gute Datenkapselung
- mehr Objekte

Objektorientierte Designprinzipien

Vererbung

Wiederverwendung durch Erweiterung bestehender Objekte.

+ Implementierung einfach
- schlechte/keine Datenkapselung

Programmieren gegen Schnittstellen

+ Unabhängigkeit von Implementierung
+ Wiederverwendbarkeit
- leicht erhöhte Designkomplexität

Open-Closed-Principle

Open-Closed

Software-Komponenten offen für Erweiterung geschlossen für Änderung

3

Java-Api

Wildcards in generischen Typen

- `<?>` $\rightarrow$ beliebiger, aber fester Typ
- `<? super Animal>` $\rightarrow$ beliebige Verallgemeinerung von Animal
- `<? extends Animal>` $\rightarrow$ beliebige Spezialisierung von Animal

4

Weiterführendes

Mobile Entwicklung

- Crossplatform
  - HTML/JS mit PhoneGap etc. oder einfach im Browser
  - Java mit RoboVM
  - C# mit Xamarin ($)
  - ...
- Android
  - Bekannte Umgebung: Eclipse+Plugin oder Android Studio
  - Meist mit Java, Natives mit c++.
  - GUIs per Designer/XML oder Java
  - Mit third-party tools alternativ auch C#,Python,etc.
  - Publishen im Play Store relativ simpel
  - Beispiel
- iOS
  - Entwicklung zwinged auf Mac
  - deutlich weniger Nutzer, aber mit mehr Geld
  - ObjC, Swift, ... kenn ich mich nicht aus
- Andere ...

Spieleentwicklung

Engines für Win/Mac/Lin/iOS/Anrdoid/HTML5:

- Unity
  - Szenen mit GUI erstellen, Code in C#.
  - recht simpel aber auch einschränkend
  - Basisversion kostenlos
- Unreal Engine 4
  - Für Studenten kostenlos
  - Wird von sehr vielen AAA-Spielen verwendet
  - Visuelles Coden + C++
  - Quelloffen
- libgdx
  - Java
  - Keine IDE, eher lowlevel
  - sehr flexibel
  - open source

Ende

Vielen Dank für eure Aufmerksamkeit!
Fragen?