Advanced SWE Softwareprojekt - Dokumentation

Finanzmanager von Cora Hartmann & Laura Obermeyer

Machine generated alternative text:
Unit Tests 
. 10 Unit Tests 
• ATRIP-Regeln 
• Code Coverage 
• Einsatz von Mocks 
Programming 
Principles 
• Analyse und 
Begründung für 
• SOLID 
• GRASP (Kopplung 
und Kohäsion) 
• DRY 
Praxisprojekt 
Code und schriftliche Dokumentation 
Objektorientierte Mainstream- 
Programmiersprache (bspw. Java, C#) 
• > 2k Zeilen Code 
. > 20 Klassen 
• Klar definierter (sinnvoller) Nutzen 
• Desktop-, Web- oder Mobilanwendung 
Entwurfsmuster 
. >= 1 Entwurfs- 
muster einsetzen 
und begründen 
• UML-Diagramm 
vorher und 
nachher 
Clean 
Architecture 
• Schichtarchitek 
tur planen und 
begründen 
. 2 Schichten 
umsetzen 
Refactoring 
• Code Smells 
identifizieren 
. 2 Refactoring 
anwenden und 
begründen 
Domain Driven Design 
• Analyse der 
Ubiquitous Language 
• Analyse und 
Begründung 
• Repositories 
• Aggregates 
• Entities 
• Value Objects 

**Wir hangeln uns am MVC Prinzip lang.**

**Startklasse als Plugin:**

* Bisher Teile der notwendigen Logik im Controller. Auslagern?
* Im Kern nochmal Main-Methode,

Klassendiagramm vor der Clean Architecture Vorlesung

Machine generated alternative text:
EMail 
-vordererTeiI 
-domain 
-laenderkennzeic.hen 
Person 
-beschreibung 
Ko nto 
-kontostand 
-bezeichnung 
Eintra 
-bezeichnung 
-beschrei bung 
-d atum 
-prcH3uktIiste 
-bezeichnung 
-beschreibung 
Name 
-Vorname 
-nach name 
UebersichtsGLlI 
aenutzerAnI 
nGUl 
Au 
abenAnzei 
Wecker 
-limit 
-enddatum 
-bezeichnung 
-beschreibung 
einnahme 
ausgabe 
stemaenderun 
-zeitstempel 
EinnahmenAnzei 
Au 
abenEin 
nGUl 
nGUl 
nGUl 
benGUl 
EinnahmenEin 
orieAnI 
StatistikenGlJI 
nGUl 
csVReader 
GUIEvent 
IJIControI 
«nterface—— 
IGLlIEventListener 
« Interface» 
IGIJIEventSender 
likation 

* Diese erste Version des Klassendiagrams entstand bevor wie die Clean Architecture in der Vorlesung behandelt haben. Im Folgenden werden die notwendigen Anpassungen beschrieben.
* Die optionalen Funktionalitäten wurden mit im Klassendiagramm modelliert. Ob sie später implementiert werden, ist noch offen.
* Wir haben gemerkt, dass die Aufteilung nicht klein/genau genug ist, insbesondere, dass unsere **GUI** Klassen mehr als eine Aufgabe haben (Single Responsibility) und zu viel Logik haben, was sie laut Clean Architecture nicht haben sollten. Sie sollten nur zum Anzeigen der Benutzeroberfläche genutzt werden. Unsere GUI Klassen werden daher aufgeteilt, so dass möglichst keine Logik in den Plugins vorhanden ist. Die Logik wird in die Schicht Adapter/ApplicationCode ausgelagert
* Unser **Model Paket** konnte so beibehalten werden. Die Klassen entsprechen Entities auf Domain Code Ebene der Clean Architecture.
* Die Klassen CSVReader und CSVWriter des **Util Pakets** sehen wir als Plugins an, da eine hohe Abhängigkeit zum Aufbau unserer csv Dateien, die als Datenbank dienen, besteht. Eine Änderung würde eine komplette Anpassung des Codes mit sich ziehen. Die Klassen sind also direkt von "der Außenwelt" abhängig.
* Prüfen, ob wirklich keine Anwendungsspezifische Logik vorhanden ist; Single Responsibility erfüllt?; alle Methoden der Klassen benötigt?
* **Paket Event:** Vermutlich Adapter, da die Klassen zur Kommunikation dienen und somit nicht zum anwendungsspezifischen Code des Application Codes gehören. Wenn wir Adapter und Application Code nicht unterscheiden, ist es doch auf einer Ebene.
* Bisher wurden die GUIs zu Beginn in unserem GUI Controller instanziiert, der ebenfalls für die Kommunikation zwischen den GUI Klassen zuständig war. In Berücksichtigung der Clean Architecture wird ein StartController eingeführt. Die StartApplikation ganz außen enthält die Main Methode, welche die Applikation startet und mit dem StartController für die Instanziierungen der Plugins zuständig ist. Der GUI Controller soll nur für die Kommunikation zuständig sein.

**Wenn wir die Adapter Schicht noch einführen sollten, sind die Controller ein Adapter und nicht im Application Code**

Klassendiagramm mit Berücksichtigung der Clean Architecture

Machine generated alternative text:
application Code 
EMail 
-vordererTeiI 
-domain 
-laenderkennzeic.hen 
Person 
-beschreibung 
Ko nto 
-kontostand 
-bezeichnung 
Eintra 
-bezeichnung 
-beschreibung 
-d atum 
produktliste 
Ka orie 
-bezeichnung 
-beschreibung 
Name 
-Vorname 
-nachname 
Uebersichts 
BenutzerAnI 
ik 
UebersichtsGLlI 
BenutzerAnI 
nGUl 
Au 
abenAnzei 
Wecker 
-limit 
-enddatum 
-bezeichnung 
-beschreibung 
einnahme 
ausgabe 
stemaenderun 
-zeitstempel 
ik 
benLo 
Au 
abenAnzei 
EinnahmenAnzei 
ik 
ik 
ik 
EinnahmenAnzei 
Au 
abenEin 
Au 
abenEin 
nGUl 
nGUl 
nGUl 
EinnahmenEin 
EinnahmenEin 
orieAnI 
StatistikenGlJI 
benGUl 
nGUl 
orieAnI 
Statistiken L 
csVReader 
controller 
GUIEvent 
GLIIControIIer 
« Interface» 
StartControIIer 
IGLlIEventListener 
« Interface» 
IGLlIEventSender 
likation 
ik 

**Paket Application Code:**

* Diese Klassen repräsentieren die überwiegenden UseCases, die wir in unserer Anwendung umsetzen wollen, wie z.B., Ausgaben hinzufügen, aktuelle Kontoübersicht anzeigen, etc.