ENTWICKLUNGSPORTFOLIO TEILKOMPONENT CLIENT

MODUL 326 OBJEKTORIENTIERT ENTWERFEN UND IMPLEMENTIEREN

TOBIAS LÜSCHERINF7F
VERSION 1.0
16.11.2016

ENTWURFPRINZIP 1 / SINGLE RESPONSIBILITY

PROBLEMSTELLUNG

Das es Klassen gibt welche mehr als nur eine Aufgabe haben und diese auch ausführen. somit hat man das Problem, dass es oft mehrere Gründe gibt eine Klasse anzupassen.

AUSWAHL DER PASSENDEN ENTWURFSPRINZIPS / -MUSTERS.

Wir suchten nach einer Lösung die Klassen zu kapseln aber trotzdem noch OO zu programmieren. Wir suchten in der Liste der Entwurfsprinzipe nach einem ähnlichen fall und fanden die Single Responsibility. Dieses Muster gibt vor das jede Klasse nur eine Aufgabe hat und keine Klasse doppelt vorkommt.

ANWENDUNG AUF DAS URSPRÜNGLICHE PROBLEM

Wir haben die Klassennamen geändert und deren Inhalte gekürzt so dass sie nur noch dies tun was ihnen als Klassenname vorgibt, z.B. haben wir den GameManager welcher nur die verschiedenen GameElemente managet. Dieser führt z.B. den GameCreater aus welcher das Spielfeld und die Listener dafür erstellt. So hat jede Klasse nur noch eine Aufgabe welche aber perfekt ausgeführt wird. Zudem haben wir geschaut das bei neu erstellten Klassen Single Responsibility ebenfalls eingehalten wird.

LERNPROZESS

Wir gingen wie schon gesagt sehr Systematisch vor. Wir schauten uns jede Klasse an und überdachten deren Aufgabe. So erstellten wir neue Klassen oder strukturierten sie um. Dabei gab es Klassen welche genau eine Aufgabe hatten, diese aber geordnet ausführten. Wir teilten bei diesem Prozess die Arbeit auf. Der Weg war sehr einfach und verständlich, da jeder bereits für einen Teil zuständig war, überarbeitete er auch seinen Teil. Verständlicher Weise gab es bei diesem Factoring neue Überschneidungen welche wir zusammen angeschaut und umgesetzt haben.

Ich sehe den Sinn hinter Single Responsibility sehr gut und ich denke wir werden es auch weiterhin anwenden. Ich finde es eine sehr gute Lösung da man ab sofort schon ohne dass man den ganzen Code liest die Aufgabe sieht und diese dann in einem neuen Code teil gleich benutzen kann. Ebenfalls sind lange Dokumentation so nicht mehr wirklich nötig.

ENTWURFPRINZIP 2/ MVC

PROBLEMSTELLUNG

Wir hatten alle Models Views und Controller wild vermischt und mussten somit immer viel an verschiedene Klassen anpassen.

AUSWAHL DER PASSENDEN ENTWURFSPRINZIPS / -MUSTERS.

Wir kannten MVC schon aus dem Betrieb und für uns war ohne lang zu suchen gleich klar, dass wir dieses Entwurfsprinzip benutzen werden.

Zudem passte es zu dem benutzten Fronten JavaFX welches selber schon auf MVC setzt.

ANWENDUNG AUF DAS URSPRÜNGLICHE PROBLEM

Wir haben 3 Ordner erstellt welche Model, View und Controller heissen. Wir haben danach die Klassen überarbeitet und in eines dieser 3 Typen eingefügt. So gab es aus vielen Klassen plötzlich 3 oder 4 und nicht mehr nur eine.

LERNPROZESS

Auch hier haben wir die Arbeit logischer weise aufgeteilt. Jeder hat seinen Teil im vorherein schon umgesetzt also ist er ebenfalls für das Factoring zuständig. So habe ich meine Teile überarbeitet und Lukas seine. Dabei haben wir immer möglichst schnell kommuniziert was der andere erstellt da wir ja z.B. Models mehrfach in verschiedenen Klassen hatten und wir die Arbeit nicht doppelt umsetzen wollten. Das Praktische dabei war das wir in der Schule immer nahe bei einander arbeiteten und so bei Fragen direkt fragen konnten. Ich denke das Prinzip ist sehr gut und wir werden es weiterhin nutzen.

Der Mehrwert für das Team ist extrem gross.