Reflexion

Es gibt einen enormen Unterschied zwischen dem Plan, den ich für die Serverimplementierung hatte, und der tatsächlichen Implementierung. Da dies das erste Mal ist, dass ich den Server implementiere, war ich mir nicht vollständig bewusst, was die Implementierung umfassen wird.

Zunächst wollte ich in meinem Klassendiagramm einen Konverter verwenden, der der Client-Implementierung ähnelt. Ich habe jedoch schnell gemerkt, dass ich effizienter arbeiten kann, wenn ich direkt die JavaDoc-Methoden und -Klassen verwende, die von den Kursleitern bereitgestellt werden. Dies sparte viel Zeit und Mühe, sorgte für einen saubereren und leichter verständlichen Code und ermöglichte es mir, mehr Zeit in die Umsetzung der in den Vorlesungen erlernten Best Practices zu investieren.

In meinem Server-Klassendiagramm habe ich ein großes Map- Package voller verschiedener Map-Klassen und Enumerationen. In Wirklichkeit habe ich nur die Klassen FullMapGenerator und Position verwendet, die für meine Arbeit ausreichend waren.

Ein weiterer Unterschied war, dass ich vorhatte, die GameIDs manuell zu generieren, aber ich habe online eine Abkürzung gefunden und in Wirklichkeit RandomStringUtils verwendet.

Die letzte Sache, die mir jetzt wirklich auffällt, ist, wie wenig Aufmerksamkeit ich den Überprüfungen zur Spielvalidierung geschenkt habe, was sich als Schwerpunkt der Serverimplementierung herausstellte. In Wirklichkeit habe ich Interfaces implementiert und spezifische Validierungen behandelt, die von den Vorlesungen und Tutorials inspiriert waren, und genau darauf habe ich die meiste Zeit bei der Implementierung verwendet.

Y-Statements

Im Kontext von der Datenumwandlung habe ich mich angesichts von der Komplexität für die direkte Verwendung der JavaDoc-Metoden und -Klassen, entschieden und eine eigene Converter-Implementierung vernachlässigt, um eine sauberere und leichter verständliche Codebasis zu erreichen, ohne wesentliche Nachteile einzugehen, weil dies viel Zeit und Arbeit sparte und es mir ermöglichte, mehr Zeit in anderen Teilen des Codes zu investieren.

Im Kontext von der Entwicklung der Softwarearchitektur habe ich mich angesichts des Ziels, die Modularität und Wiederverwendbarkeit zu erhöhen, für die Erstellung von Interfaces entschieden, um die Open-Close-Prinzip, Interface-Seggregation-Prinzip, Zentralisierung und Wartbarkeit zu gewährleisten, unter Inkaufnahme von erheblichem Zeit- und Arbeitsaufwand, sowie eine höhere Komplexität des Codes, weil dies langfristig zu einer saubereren Implementierung und effizienteren Wartung führt.

Im Kontext von Fehlerbehandlung habe ich mich angesichts der Komplexität des Umgangs mit checked Exceptions für das Werfen von unchecked Exceptions entschieden, wenn checked Exceptions gefangen werden, um eine einfachere Fehlerbehandlung und verbesserte Lesbarkeit zu erreichen sowie die Klienten von der Fehlerbehandlungspflicht zu entbinden, unter Inkaufsnahme von einer Lernkurve für mich und einem erheblichen Zeitaufwand zum Verständnis, weil die zu einer effizienteren und wartungsfreundlichen Softwarearchitektur führt.