Reflexion

Es gibt einen wesentlichen Unterschied zwischen meinem Plan und der Realität.

Beginnen wir mit der Validierung der Karte. Mein Plan war es, einfach eine separate Klasse für die Validierung der Karte zu implementieren, aber nach mehr als der Hälfte der Umsetzung des Projekts begann ich, über das Testen nachzudenken. Daraufhin beschloss ich, das Strategy- Pattern zu implementieren. Das hat die Lesbarkeit stark verbessert und das Testen erleichtert.

Die andere große Änderung war der Pathfinding-Algorithmus. Als ich mein Klassendiagramm entwarf, hatte ich keinen genauen Plan, wie ich den Algorithmus implementieren würde, was zu einer Menge Verwirrung und Schwierigkeiten bei der Implementierung führte. Ich stützte mich stark auf eine Implementierung, die ich im Internet gefunden hatte (die im Projekt gemäß den Regeln angegeben ist). Ich habe einen einfachen Dijkstra-Algorithmus implementiert, der in meinem Klassendiagramm nicht richtig beschrieben ist.

Was die Best Practices betrifft, so habe ich so viel implementiert, wie ich es tun konnte. Erstens, das oben beschriebene Strategy-Pattern. Außerdem habe ich die Verwendung von "null" vermieden und mich stattdessen für die Verwendung von Optional entschieden. Ich habe versucht, die Verantwortlichkeiten von Paketen, Klassen und Methoden so weit wie möglich zu trennen, um "Single Responsibility" zu gewährleisten. Ich habe dafür gesorgt, dass Kapselung und Information Hiding vorhanden sind, indem ich so viele Methoden wie möglich privat gehalten und unnötige Details versteckt habe. Ich habe auch zwischen der Verwendung von toString()-Methoden und format...()-Methoden unterschieden.

Y-Statements

Im Kontext der HalfMap-Validierung habe ich mich angesichts von Bedenken bezüglich der Komplexität und Flexibilität für das Strategy-Pattern entschieden und die Option für eine selbstständige Klasse vernachlässigt, um eine verbesserte Wartbarkeit und Erweitbarkeit zu erreichen, unter Inkaufnahme von erhöhtem Implementierungsaufwand, weil ich die Validierung logisch trennen und die Testphase erleichtern wollte.

Im Kontext der Implementierung habe ich mich angesichts von Bedenken bezüglich der Nullreferenzierung für die Verwendung von Optional entschieden und die Verwendung von null vernachlässigt, um die Robustheit und Wartbarkeit des Systems zu verbessern, unter Inkaufnahme einer erhöhten Implementierungszeit, um die Codequalität zu steigen und die Fehleranfälligkeit zu reduzieren.

Im Kontext der Implementierung habe ich mich angesichts von Bedenken bezüglich der Skalierbarkeit für die Trennung von Klassen nach ihren Verantwortlichkeiten entschieden, um Single Responsibility zu erreichen, unter Inkaufnahme eines erhöhten Zeitaufwands, um die Wartbarkeit des Systems zu erleichtern.