MAYFLOWER

Strategy Pattern

Pattern DoJo, 02. März 2021

Definition

Das Strategy Pattern definiert eine Familie austauschbarer Algorithmen und kapselt diese in Klassen, die zur Laufzeit dynamisch geladen werden können.

Jeff liebt seinen Job!

Jeff ist der leitende Entwickler einer App, welche die kürzeste Route zwischen zwei Punkte berechnet.

Sein Projekt läuft großartig und verfügt über eine gesunde Struktur. Schauen wir uns das Herzstück seiner Applikation mal an..

Alle handelnden Personen sind frei erfunden

Neue Anforderungen

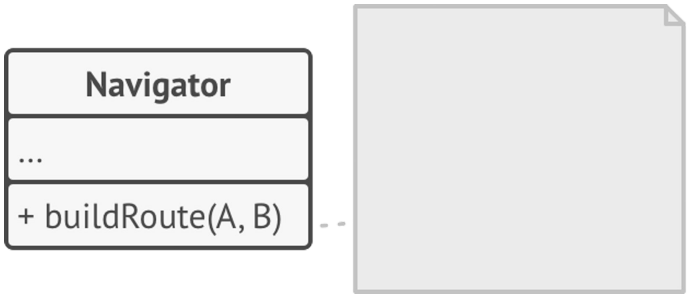
Nun soll Jeff das Programm erweitern, sodass die kürzeste Route neben PKW nun auch für Fahrradfahrer und für den ÖPNV berechnet werden kann.

Hierfür werden möglicherweise völlig andere Routen berechnet.

Problem

In der Navigator.buildRoute(A, B) wird die kürzeste Straßennavigation für PKW zwischen den beiden Punkten A und B berechnet.

Navigator1.kt



Problem

Um diese Erweiterung durchzuführen, würde man in der Navigator.buildRoute(A, B) ansetzen und je nach gewünschtem Transportmittel den Codefluss verzweigen.

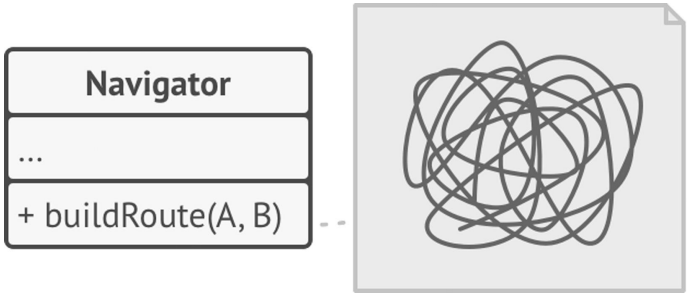
Navigator2.kt

Die hier unterschiedlichen Algorithmen würde man in eigene Funktionen überführen, um ein riesiges switch-Konstrukt zu vermeiden.

Navigator3.kt

Problem

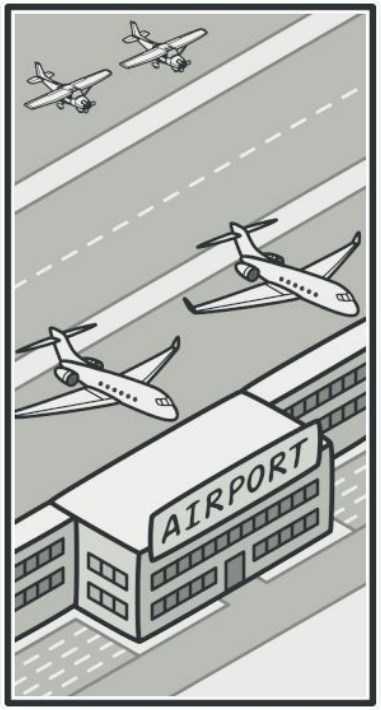
Im Zuge der Ausarbeitung für den neuen Algorithmus würde diese Erweiterung die Klasse Navigator aufblähen und viel Code innerhalb dieser Einheit definieren. Dieser ist an die Kontext-Klasse Navigator gebunden und sollte besser sauber getrennt und unabhängig definiert werden.



Technische Schulden

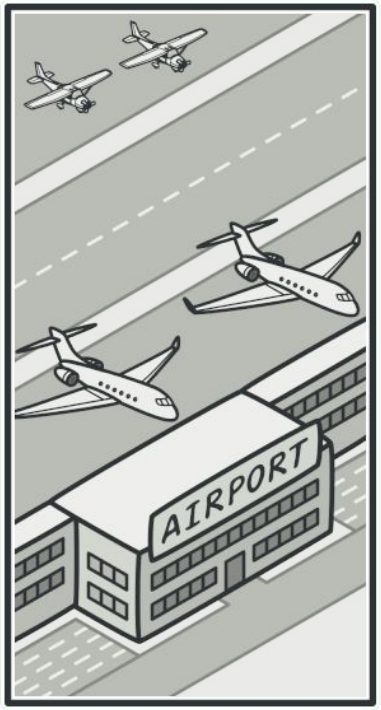
Der neue Code macht was er soll, aber Jeff hat trotzdem Technische Schulden produziert.

Seine zuvor übersichtliche Klasse Navigator implementiert nun viel unterschiedliches Verhalten. Dies macht sie schlechter lesbar und die Struktur seines Codes spröder.



Verwendung

Unterschiedliche Verhaltensweisen und deren verwendete Algorithmen sind innerhalb einer Klasse fest integriert und sollen von deren Kontext unabhängig und wiederverwendbar gemacht werden.



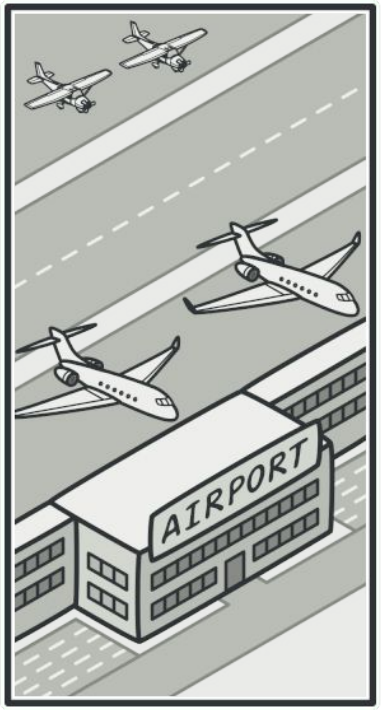
Implementierung

1. Identifizierung des Algorithmus, der unterschiedlich implementiert werden muss.

2. Definition einer Strategy Interface mit den erforderlichen abstrakten Methoden.

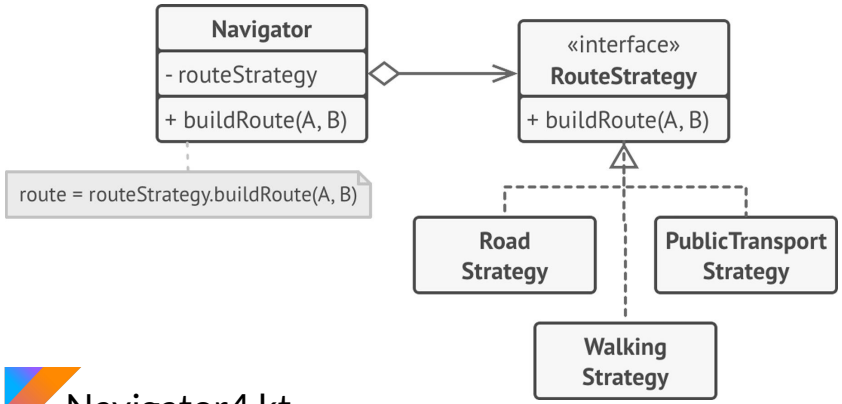
3. Auslagerung der verschiedenen Algorithmen in eigene Klassen, welche jeweils das Strategy Interface implementieren.

4. Erzeugung einer Referenz auf die Strategy Interface in der Kontext Klasse und Instanziierung einer konkreten Strategy Klasse.



Lösung

Die buildRoute(A, B) wird von einer neuen Schnittstelle RouteStrategy abstrakt deklariert. Für alle benötigten Varianten werden Klassen erstellt welche diese Schnittstelle implementieren und das konkrete Verhalten definieren.



Navigator4.kt

Jeff und das Strategy Pattern haben gerockt!

Jeff hat mit dem Einsatz des Strategy Patterns die Grundlage für eine gute Codestruktur geschaffen, mit der die Anwendung auch in Zukunft gesund wachsen kann.

Anwendbarkeit

- Kontextunabhängige Auslagerung und Entkopplung von Logik oder Verhalten.
- Beliebige Verwendung und Austausch der verschiedenen Algorithmen zur Laufzeit.
- Kommunikation des Kontext mit dem Strategy Object lediglich über dessen Strategy Interface.
- Das Strategy Pattern bieten eine flexiblere Alternative zur Unterklassenbildung der Kontexte.

Vorteile

- Zusammengehörige Algorithmen werden kontextunabhängig getrennt und in eine klare Codestruktur überführt.

- Neue Strategien können ohne großen Aufwand eingeführt werden da sie die bereits geschaffene Struktur verwenden.

- Das Pattern ermöglicht eine Auswahl aus verschiedenen Implementierungen und erhöht so die Flexibilität und Wiederverwendbarkeit unseres Quellcodes.

Nachteile

- Die Komplexität des Programms wird erhöht.
- Gegenüber der Implementierung der Algorithmen im Kontext entsteht ein zusätzlicher Kommunikations-aufwand zwischen Strategie und Kontext.
- Übermäßige Anwendung des Strategy Patterns kann zu Overengineering führen und die Programmstruktur durch die zusätzlichen Klassen und Interfaces aufblähen.

Ja zum Strategy Pattern!

Familiäres Verhalten vom Kontext entkoppeln, klar abgrenzen und damit unabhängig und wieder-verwendbar machen. Das nenne ich Refactoring!

Developer, Würzburg

Verzweigungen im Code bei denen verwandtes Verhalten angewandt wird sind prädestiniert für die Anwendung des Strategy Patterns.

Developer, Berlin

Zitate dienen nur Veranschaulichungszwecken

Nein zum Strategy Pattern!

Die Erhöhung der Komplexität unseres Quellcodes muss einen ausreichenden Mehrwert bringen.

Developer, München

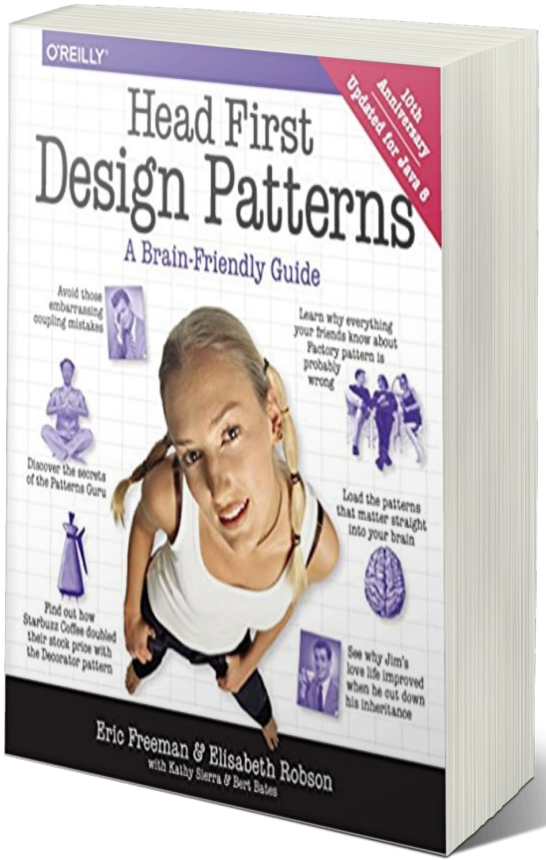
Werden die verwandten Algorithmen außerhalb der Kontextklasse gar nicht verwendet oder definieren sie nur wenig Code, sollte das Pattern nicht verwendet werden.

Developer, Würzburg

Vielen Dank für Ihre Aufmerksamkeit!

Die wichtigsten Design Patterns werden in dem modernen Klassiker Head First Design Patterns ausführlich behandelt.

Mit dem einzigartigen Head First Lernkonzept gelingt es den Autoren, die Materie witzig, leicht verständlich und dennoch gründlich darzustellen. Das ist nicht nur unterhaltsam, sondern auch effektiv.



Quellen und mehr Info

O’Reilly - Head First Design Patterns
Strategy Pattern - Refactoring Guru
Wikipedia - Strategie (Entwurfsmuster)
Google Slides Template “Zündende Idee”
Try the Kotlin Programming Language!
Code Beispiele auf GitHub


Du kannst mithelfen, das Wissen über Modernisierungs- und Design-Patterns weiter zu verbessern, indem Du Dein Wissen in der Gruppe Dev teilst.