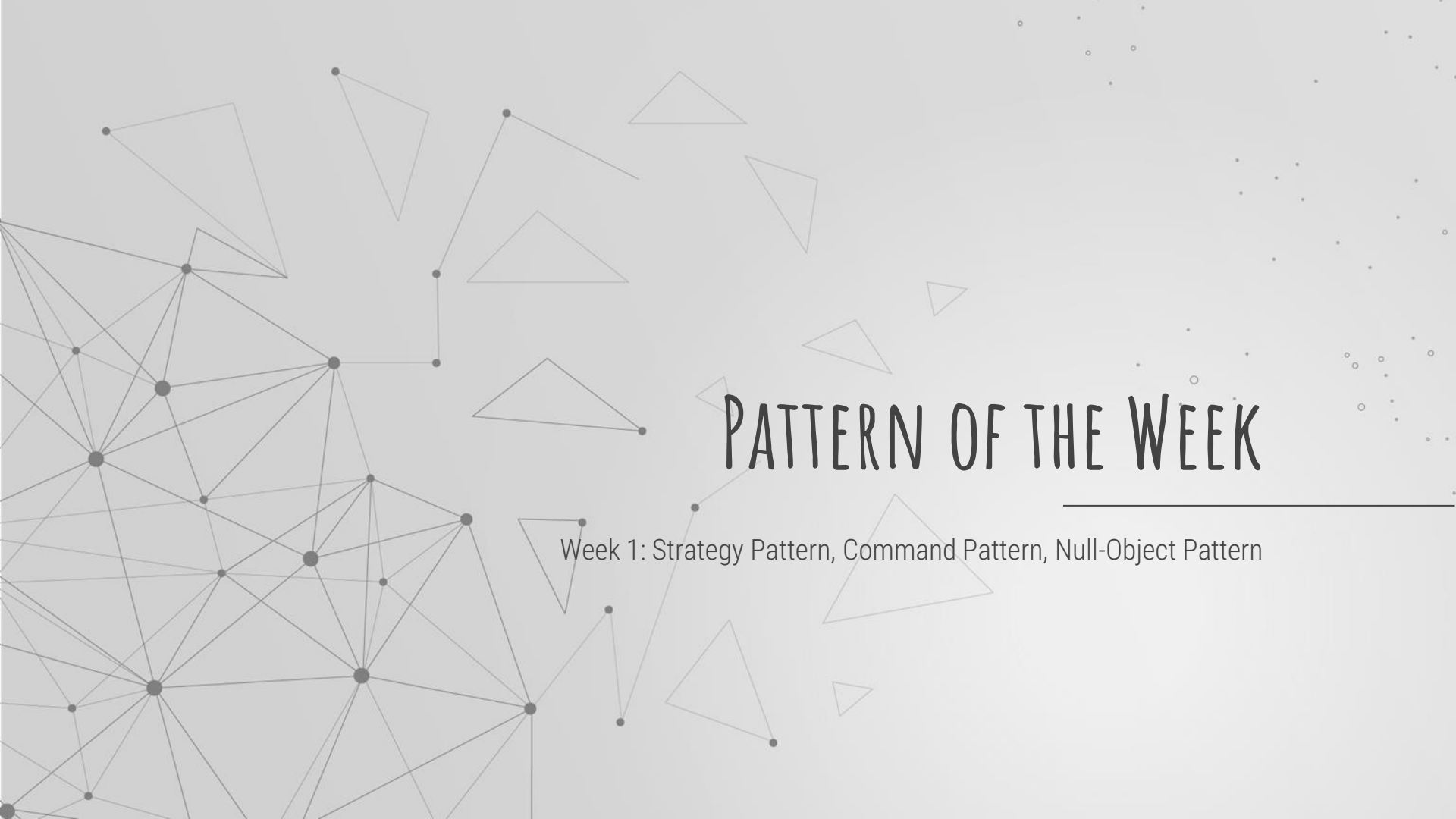


Pattern of the Week

Week 1: Strategy Pattern, Command Pattern, Null-Object Pattern

01

Strategy Pattern

Single Responsibility Principle

- Eine Funktionseinheit (Methode/Klasse/…) implementiert nur einen Aspekt
- Änderungen unabhängig von anderen Funktionseinheiten

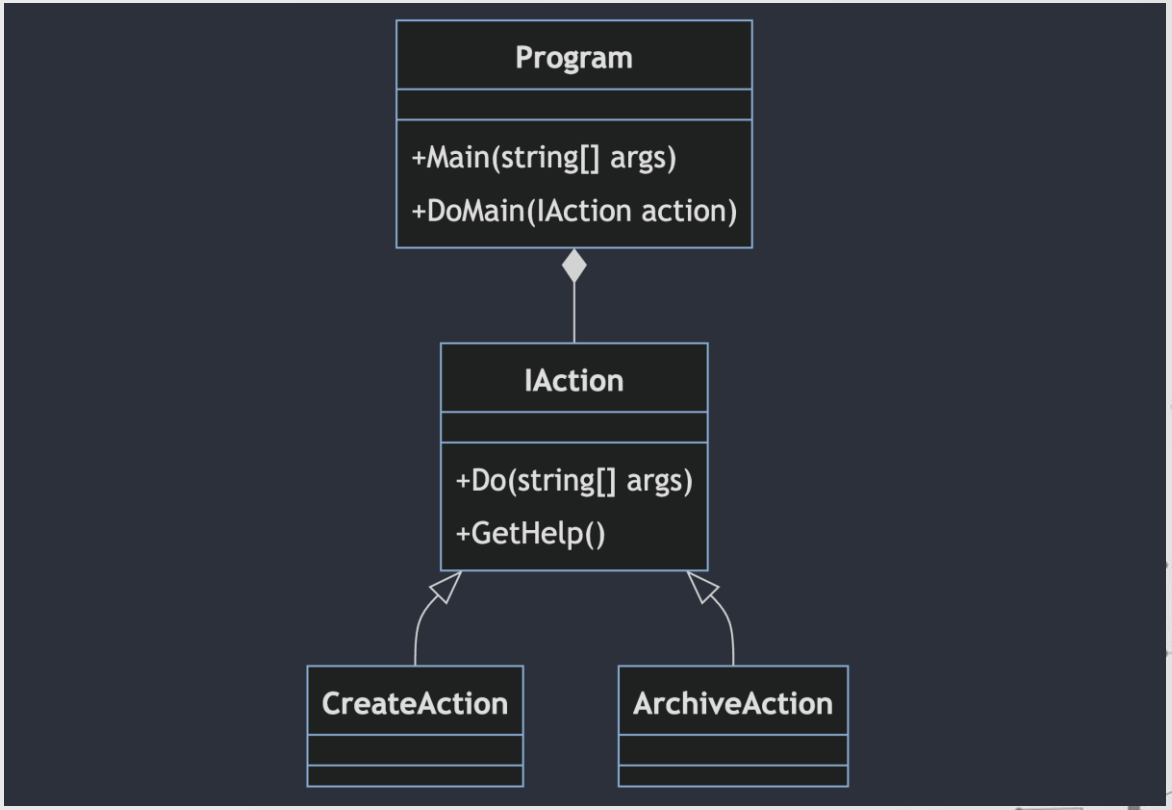
- Anti-Pattern
  - Ravioli Code (Viele sehr kleine Klassen)
  - Functor Object (Klassen mit genau einer Methode)

Strategy Pattern

- Mehrere Implementierungen einer (ähnlichen) Funktionalität
- Typischerweise Abstrahierung von (technischer) Infrastruktur
- “Extract Interface” für Testbarkeit

- Code Smell: “if, ifelse, ifelse, ifelse, else”
- Z.B. if (outlook) {} else if (notes) {} else if (thundermail) {}
- Gleicher Switch mehrfach im Code

UML Strategy Pattern



Aufgabe

Fertigstellen/Implementieren des Strategy Pattern

Projekt „PdfTools“

IAction Interface
Do() Methode
Leere GetHelp() Methode

02

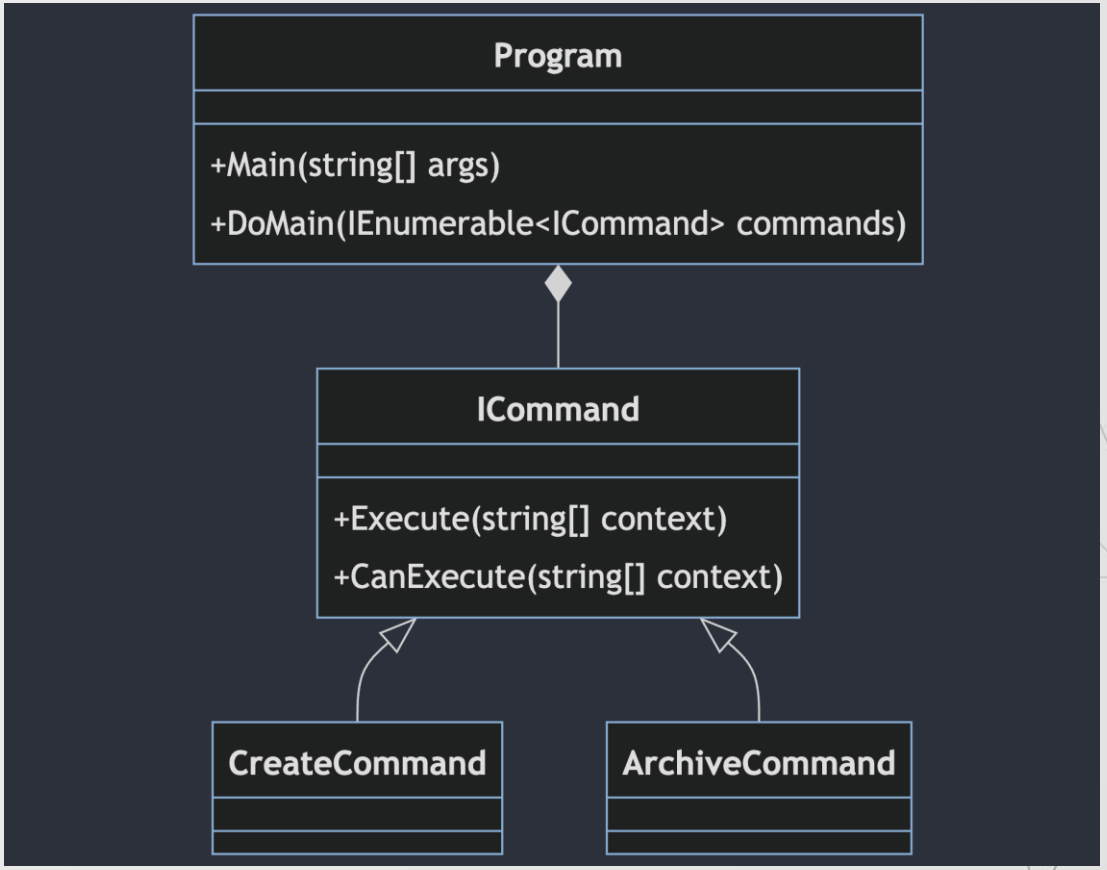
Command Pattern

Command Pattern

- Kommando Entwurfsmuster
- Verwendung z.B. Buttons einer UI

- Prüfen ob ein Kommando ausgeführt werden kann (CanExecute)
- Ausführen des Kommandos(Execute)
- Beide Methoden bekommen den gleichen Parameter “Context”

UML Command Pattern



Aufgabe

Implementieren des Command Pattern

Projekt „PdfTools“

ICommand Interface
CanExecute(ctx)
Execute(ctx)

03

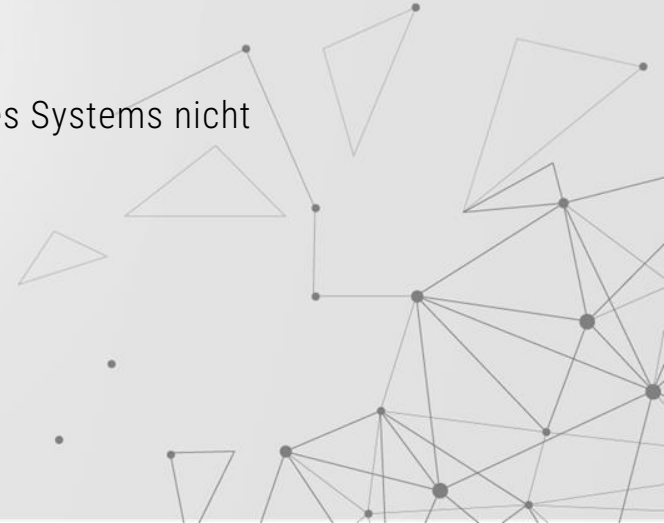
Empty-Object Pattern

aka. Null-Object Pattern

Null-Object Pattern / Empty-Object Pattern

- Implementieren des Interface mit einer „leeren“ Funktionalität
- Funktionen geben „Default()“ zurück
- Methoden machen „nichts“

- Das Empty-Object Pattern darf den Zustand/das Verhalten des Systems nicht verändern



Aufgabe

Implementiere ein Empty-Object

Projekt “PdfTools”

EmptyStrategy,
Fallback-Strategy.

Resuemee

Command vs. Strategy

- Prüfen „CanExecute()“
- Mehrere „true“ für gleichen Context
- Verwendung: z.B. UI-Buttons

- N-Methoden per Strategie
- I.d.R. nur eine Strategie
- Verwendung: z.B. MSSQL vs. MongoDB

- Leere Implementierung Empty-Object