Iterator-Pattern
Verhaltensmuster

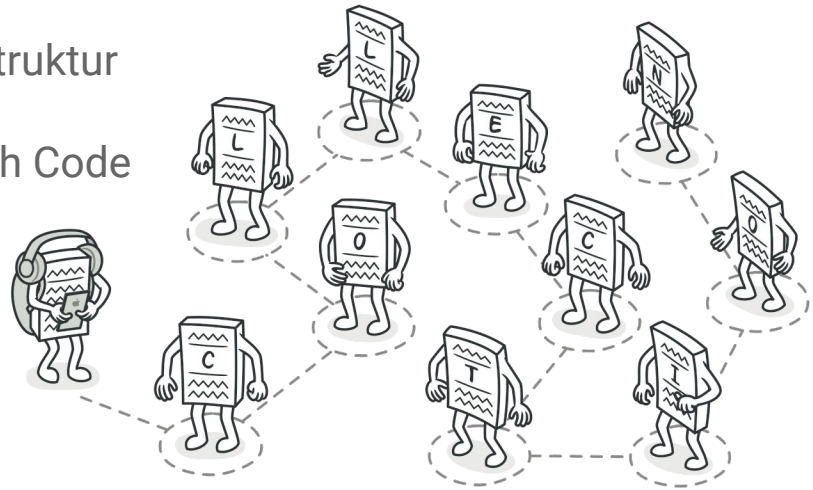
Yassine Bensaleh, Andrea Engel,
Rene Fischer, Sascha Görnert,
Niko Lockenvitz, Julian Rolle

Mannheim, den 16.01.2020

DHBW
Duale Hochschule
Baden-Württemberg

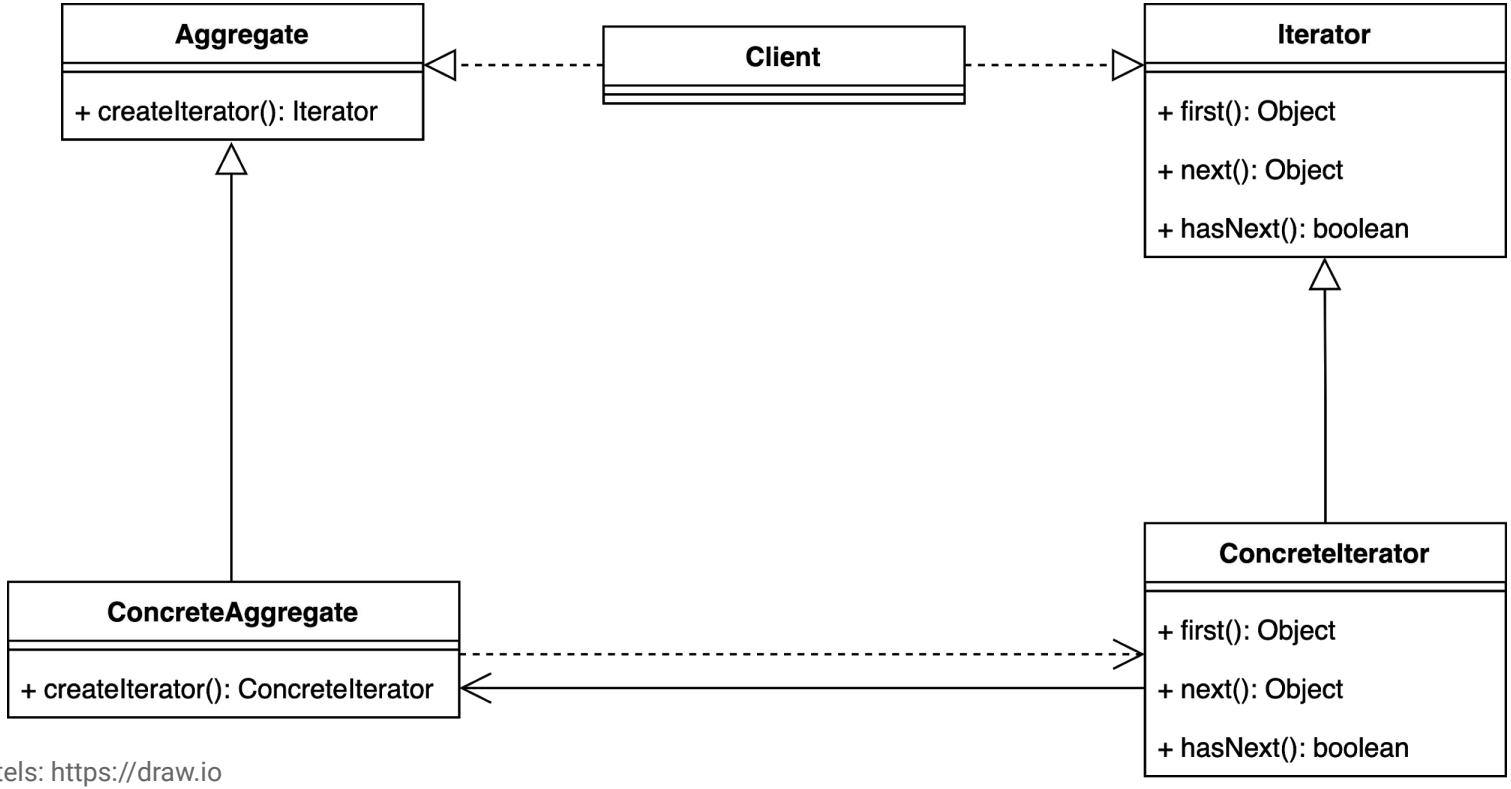
Verwendung

- Bekannt: Collection-Framework (Java)
- Sequenzieller Zugriff → ohne Enthüllung Struktur
- Verwendung von SRP → Auslagerung durch Code
- Open/Closed-Prinzip → neuer Iterator
- mehrere Iterationswege bereitstellen



Quelle:
https://refactoring.guru/images/patterns/content/iterator/iterator-2x.png

UML



Quelle:
Darstellung mittels: https://draw.io

Codebeispiel

Vorteile

- Sequenzieller Zugriff

  → ohne Enthüllung Datenstruktur
- Verwendung SOLID-Prinzipien

  → SRP, Open/Closed-Prinzip
- einheitliche Schnittstelle

  → Bsp. Collection-Framework (Java)
- Vereinfachung von Code

  → Logik bei Iterator

Nachteile

- Erzeugen & Vernichten von Aggregaten

  → Laufzeit-, Speicherkosten
- Aufwand bei kleinen Collections

  → “Overkill”

Quellen

- https://www.journaldev.com/1716/iterator-design-pattern-java, 11.01.2020
- https://www.geeksforgeeks.org/iterator-pattern/, 13.01.2020
- https://refactoring.guru/design-patterns/iterator, 15.01.2020
- https://de.wikipedia.org/wiki/Iterator_(Entwurfsmuster), 15.01.2020
- https://en.wikipedia.org/wiki/Iterator_pattern, 15.01.2020