LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR MATHEMATIK, INFORMATIK UND STATISTIK
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR DATENBANKSYSTEME UND DATA MINING

Design Patterns

SEP 2018

Tobias Lingelmann

2018-05-08

DBS

Wissenschaftliche Betreuer:
Daniel Kaltenthaler, Johannes Lohrer
Verantwortlicher Professor:
Prof. Dr. Peer Kröger

Inhalt

- Allgemeines
- Ziele von Entwurfsmustern
- Mögliche Probleme
- Verhaltensmuster (Behavioral)
- Strukturmuster (Structural)
- Erzeugungsmuster (Creational)

Allgemeines

- Wiederverwendbare Vorlagen für häufiger vorkommende Herausforderungen.
- Ursprünglich aus der Architektur (Christopher Alexander, 1979)
- 1994: “GoF-Patterns” aufgrund des Buches “Design Patterns” von Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides (Gang of Four)

Ziele

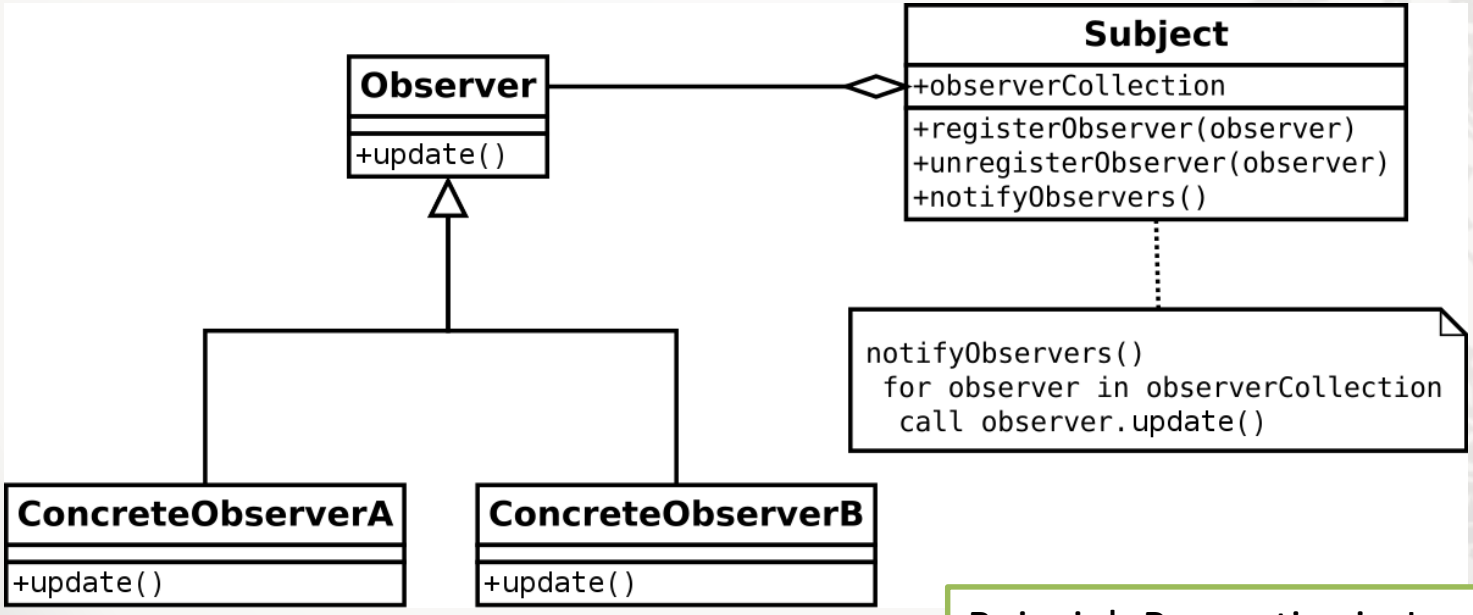
- Bieten bewährte Lösungen für Probleme
- Unterstützen das Schreiben flexiblen, wiederverwendbaren und wartbaren Codes
- Helfen bei der Umsetzung bestimmter Design-Prinzipien (SOLID, etc.)
- Generell unabhängig von der Sprache (da nur „Ideen“, nicht „Implementierungen“)

Mögliche Probleme

- Nützliches Werkzeug zur Lösung bestimmter Probleme, aber nicht „Mittel für alles“. (Ein Hammer ist für eine Schraube nur zweite Wahl. ☺)
- Entwurfsmuster könnten fälschlicherweise als Garant für gutes Design/guten Code angesehen werden.
- Manche „Design Patterns“ werden bei falscher Anwendung zu „Anti-Patterns“.

Observer-Pattern

Ein Subjekt verwaltet eine Liste an Observern, die benachrichtigt werden, sollte sich etwas beim Subjekt ändern.

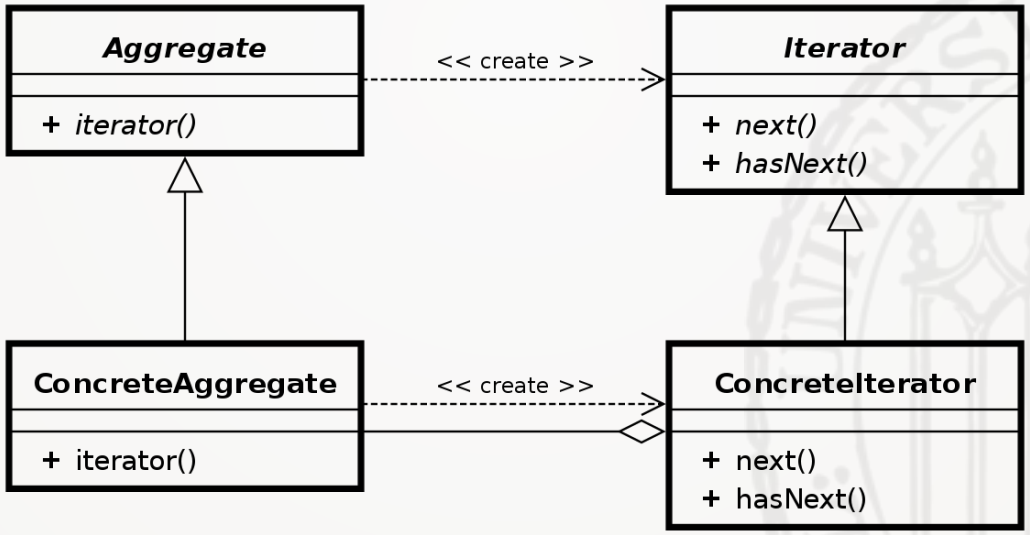


Quelle

Beispiel: Properties in JavaFX

Iterator-Pattern

Ein Iterator („Cursor“) wird genutzt um sequentiell auf Daten zuzugreifen ohne deren genaue Struktur offen zu legen.

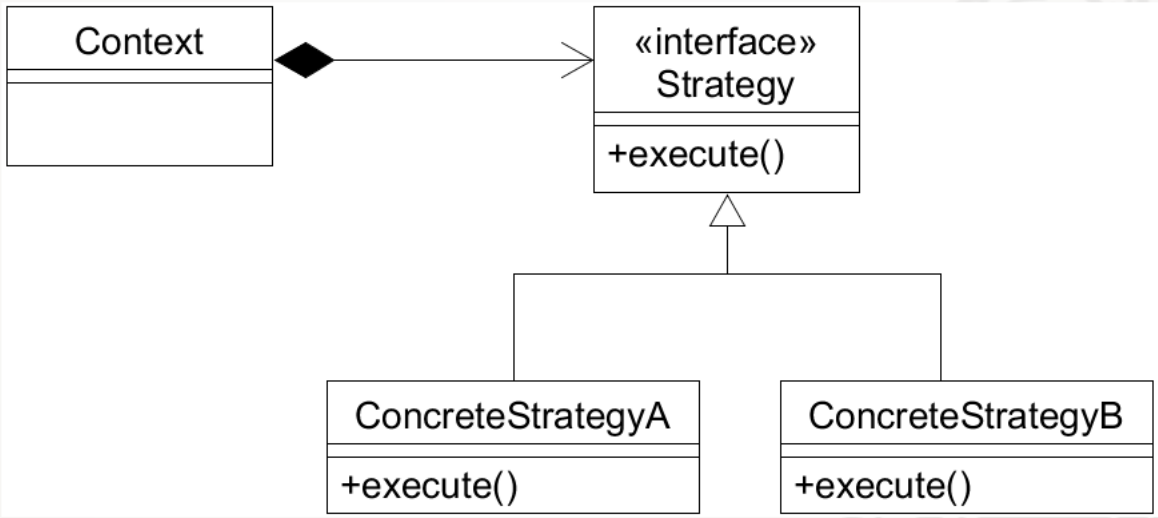


Quelle

Beispiel: `for (String a : myStringList) { …`

Strategy-Pattern

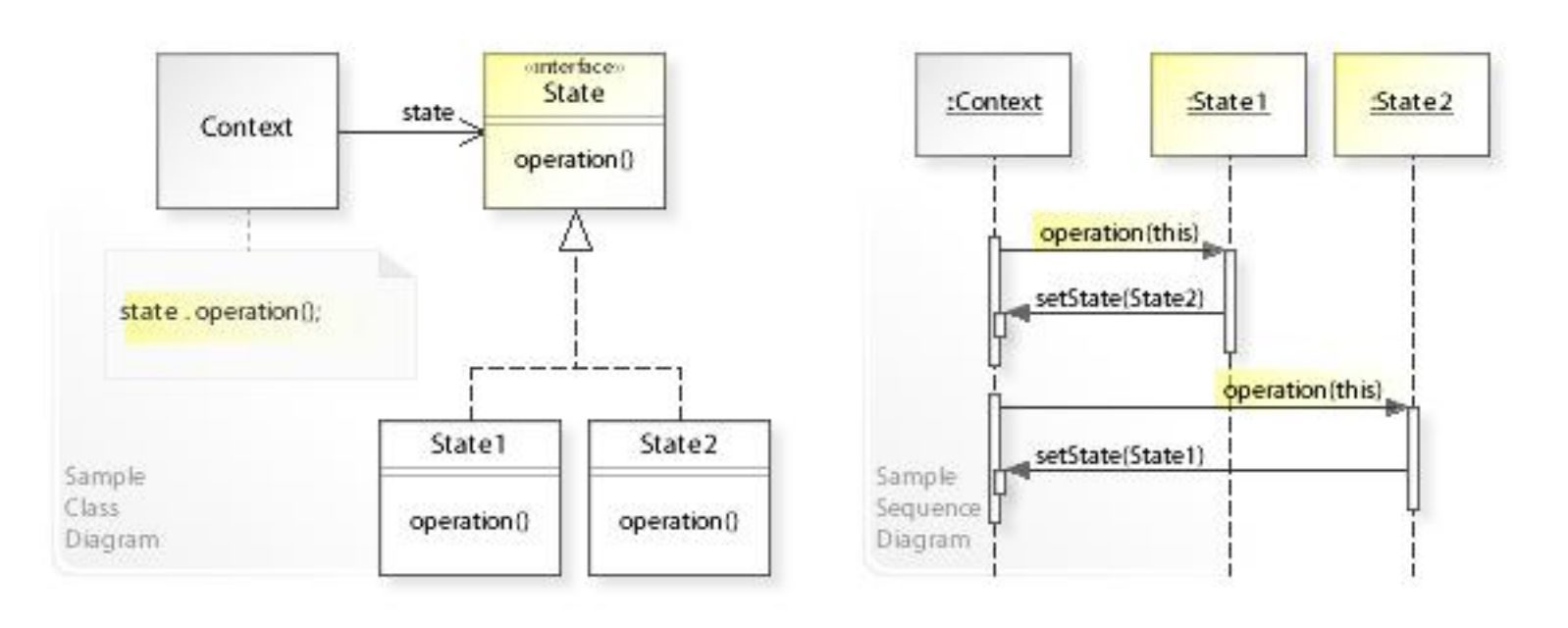
Erlaubt während der Laufzeit aus mehreren Algorithmen zu wählen. (Auch gut, wenn sich sonst, wenig bis nichts ändert, um Unterklassen zu vermeiden.)



Beispiel: Unterschiedliche Sortieralgorithmen

State-Pattern

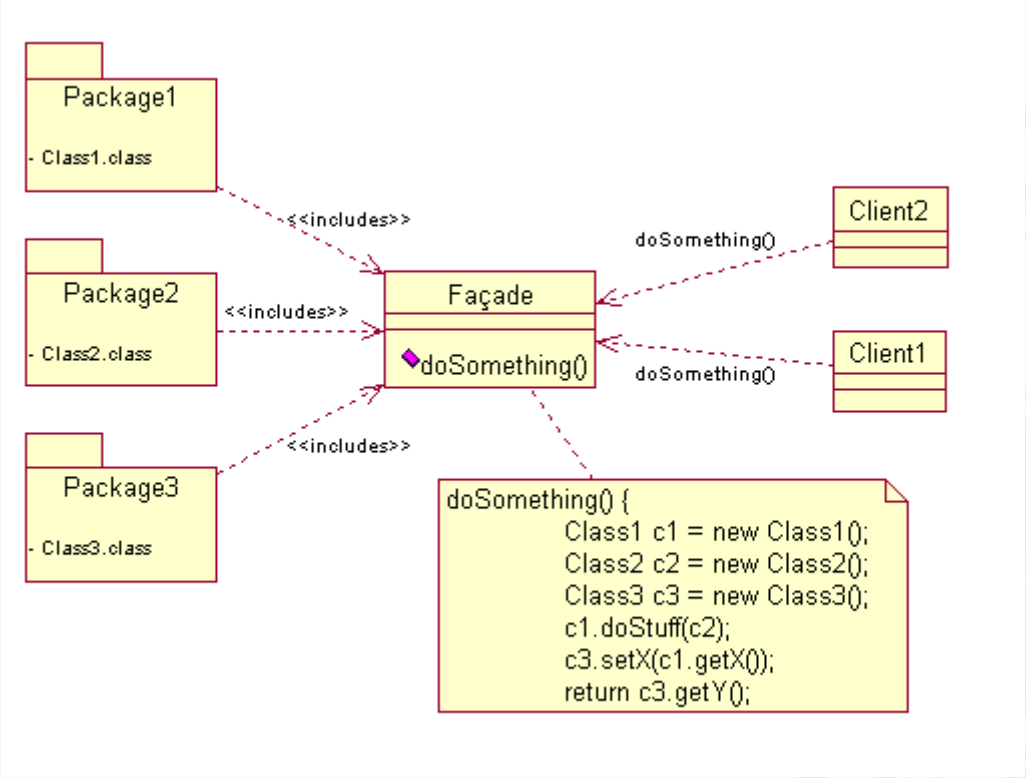
Implementiert eine (endliche) Zustandsmaschine, bei der sich das Verhalten basierend auf dem aktiven Zustand ändert.



Quelle

Facade-Pattern

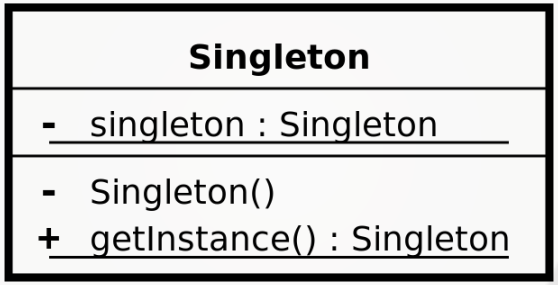
Versteckt ein komplexes System hinter einer einfach aufzurufenden Fassade/API.



Quelle

Singleton-Pattern

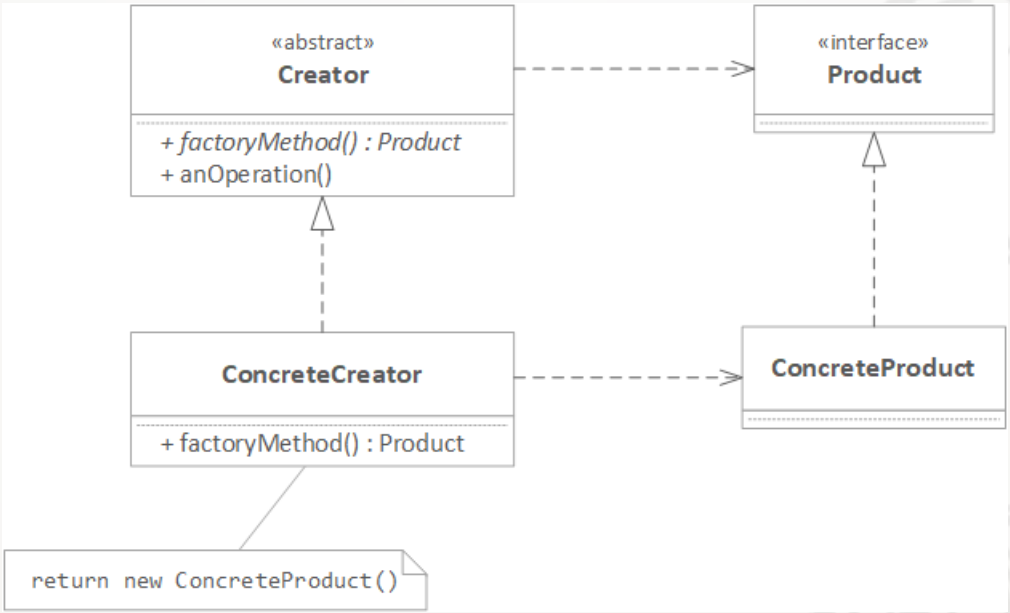
Sorgt dafür, dass nur eine Instanz dieses Objekts existiert.



Wird oft aus Bequemlichkeit genutzt, obwohl nicht nötig. → Aufpassen mit der Verwendung!

Factory-Pattern

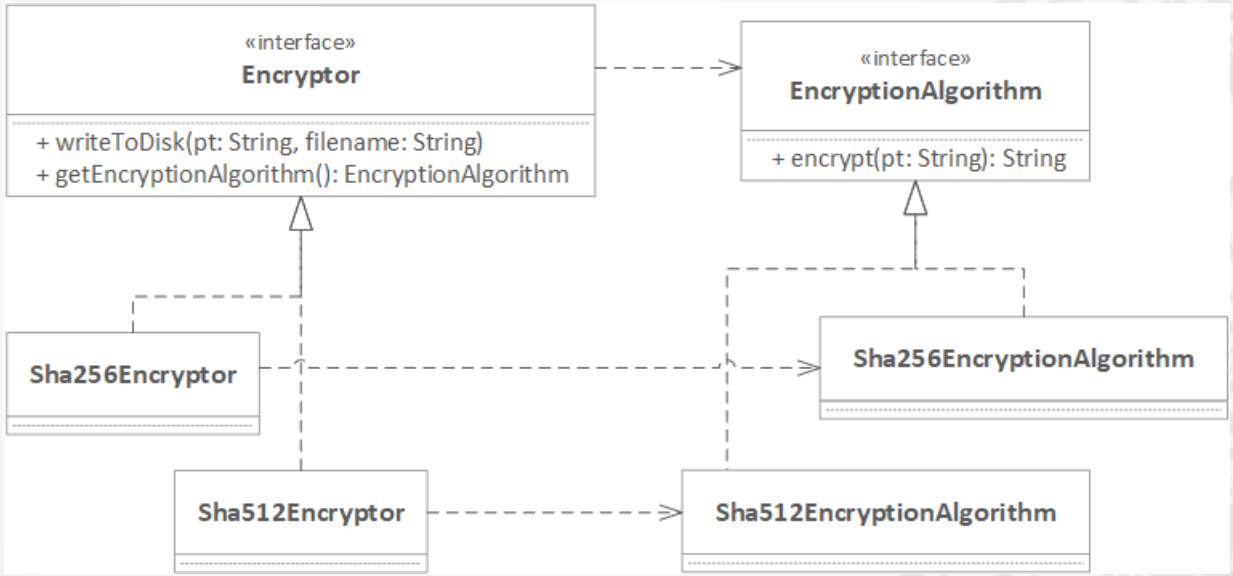
GoF: „Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.”



Quelle

Factory-Pattern

GoF: „Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.”



Quelle

Factory-Pattern

Weit geläufiger: Irgendeine Methode, deren einzige Aufgabe es ist den Erstellungsprozess eines Objekts zu „verstecken“ und das Objekt zurück zu geben.

```
public class JsonHandlerFactory {
    public static JsonHandler createHandler() {
        return new JsonPrintHandler();
    }
}
```

Linksammlung

- https://en.wikipedia.org/wiki/Software_design_pattern
- http://www.java2s.com/Tutorials/Java/Java_Design_Patterns/index.htm
- https://www.tutorialspoint.com/design_pattern/index.htm
- https://github.com/EnterpriseQualityCoding/FizzBuzzEnterpriseEdition ;-)
- https://dzone.com/articles/java-the-factory-pattern