ABB Technikerschule

Technik. Informatik. Wirtschaft. Management →

Dominik Meyer
Software Engineering

Entwurfsmuster (Factory Pattern)

Ziel

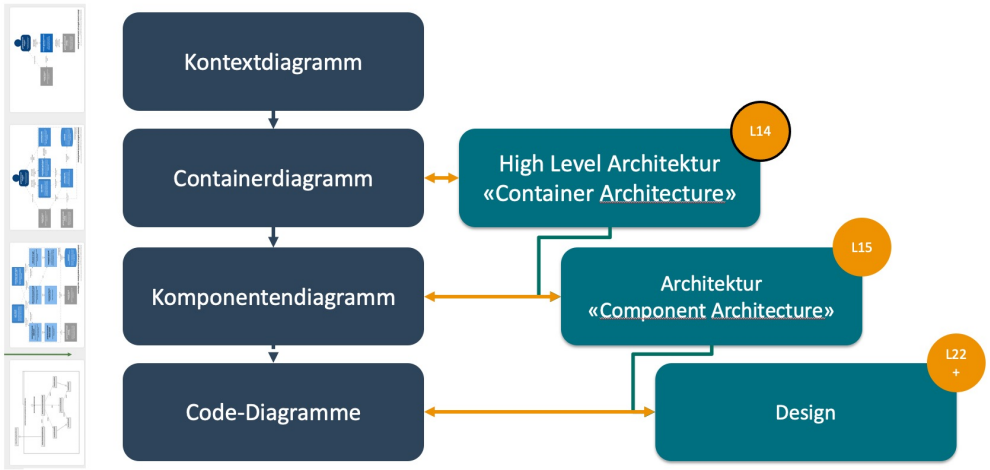
Nach der Lektion begründen die Studierenden die Anwendung des Factory Patterns und haben dessen einfache Form (ohne abstrakte Ersteller) in ihrem `spring-starter` Projekt angewandt.

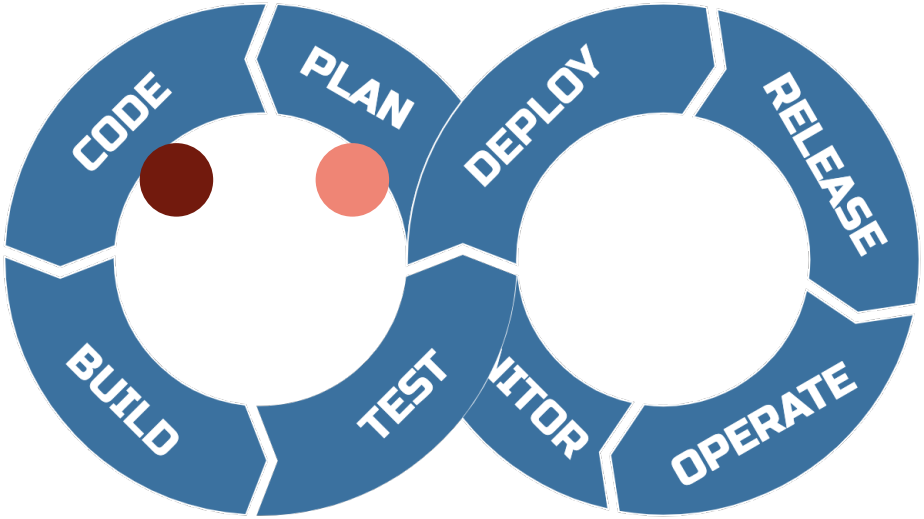
Entwurfsmuster

– Architektur
– Container für Tests und Delivery
– Unit -& Integrationtests
– Github Actions zur Automation

– Applikation gem.
«Design Considerations»
– Entwurfsmuster

Unsere Flughöhen





Agenda

– Ziel
– Transferaufgabe
– Entwurfsmuster → Factory Pattern

Transferaufgabe SWD

– Strategy Pattern

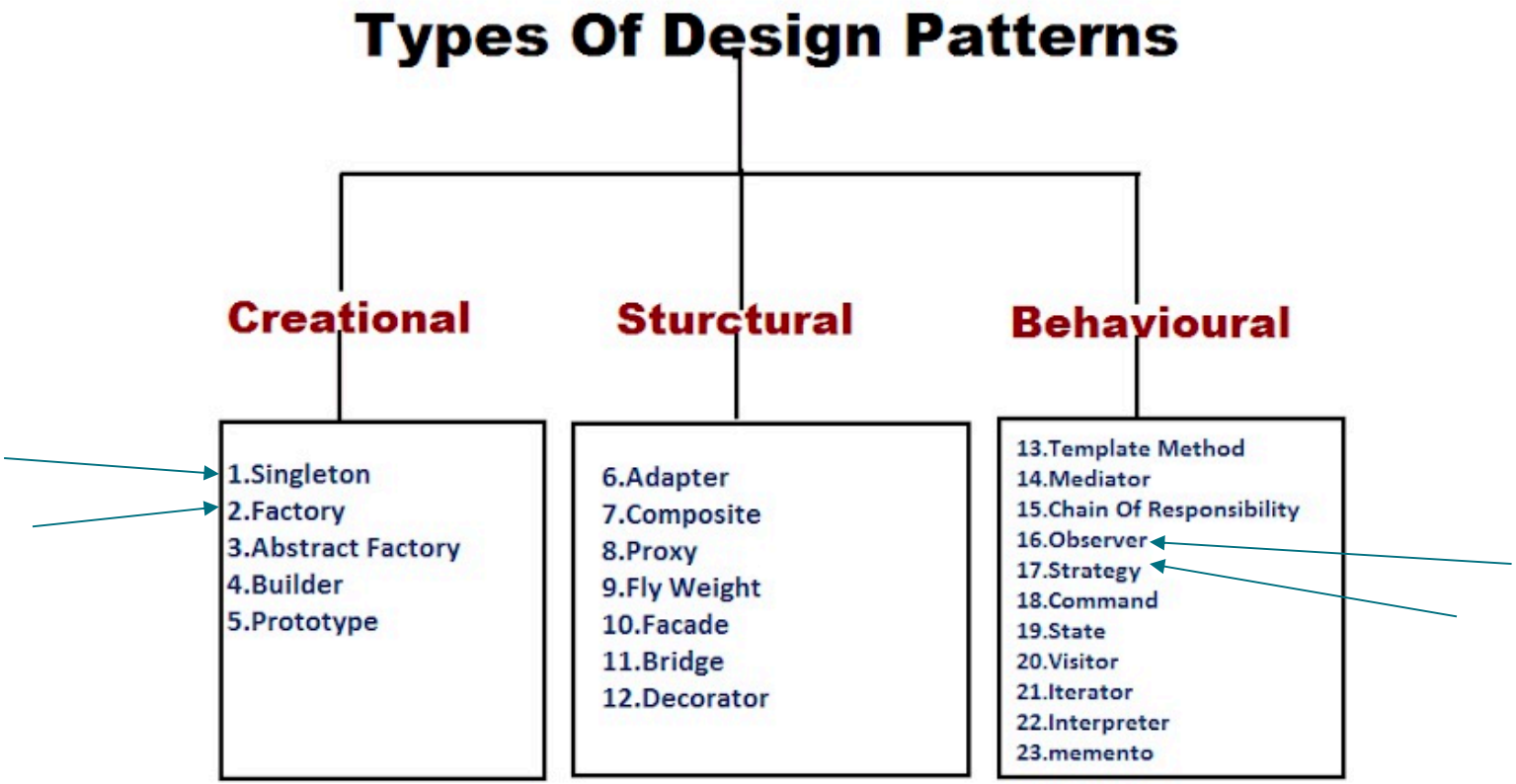
– Der Auftrag war es, zu kommentieren

Design Considerations

- Compatibility - With others or an older self
- Extensibility - Adding capabilities without major changes
- Maintainability - Documentation, DevOps, Transparency, clean
- Modularity - isolated components, exchangeable, division
- Reliability - functions under load for a specific time
- Reusability - Don’t repeat yourself DRY, reuse in other designs
- Robustness - Operate under stress / faulty data, fail graceful
- Security - Withstand hostile acts and influence, keep data safe
- Usability - Emotions, help, experience, journeys

- Design != Architektur, jedoch ist klar
  - Ungeeignete/schlechte Architektur kann kein gutes Design fördern

- Considerations sind in «Design Prinzipien» SOLID und GRASP verankert


Types Of Design Patterns
Creational
Sturctural
Behavioural
1.Singleton
2.Factory
3.Abstract Factory
4.Builder
5.Prototype
6.Adapter
7.Composite
8.Proxy
9.Fly Weight
10.Facade
11.Bridge
12.Decorator
13.Template Method
14.Mediator
15.Chain Of Responsibility
16.Observer
17.Strategy
18.Command
19.State
20.Visitor
21.Iterator
22.Interpreter
23.memento

Factory Pattern

– refactoring.guru/design-patterns/factory-method

– Wie lösen wir folgendes Problem?

Wir wissen zum Zeitpunkt zu dem wir das Programm schreiben noch nicht welche genaue Klasse wir brauchen. Ebenso wollen wir Nutzern unseres Programmes die Möglichkeit geben, selber neue Klassen hinzuzufügen ohne unser Programm anzufassen.

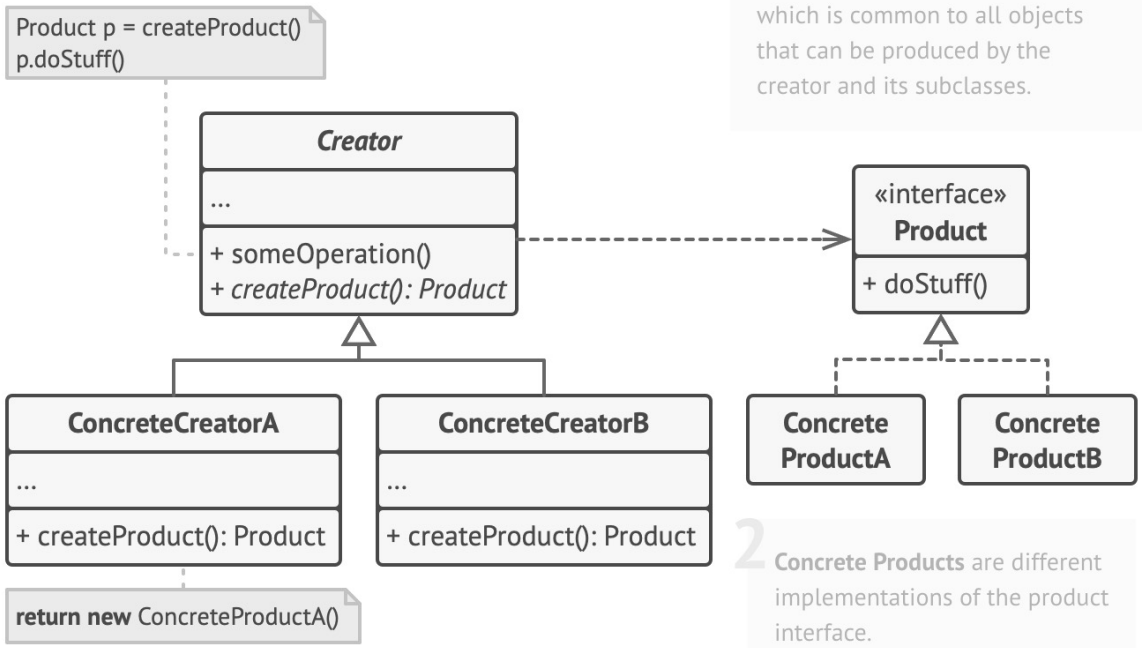
- Expert
- Creator
- Low Coupling
- High Cohesion

O Open/closed principle

3 The **Creator** class declares the factory method that returns new product objects. It's important that the return type of this method matches the product interface.

You can declare the factory method as abstract to force all subclasses to implement their own versions of the method. As an alternative, the base factory method can return some default product type.

Note, despite its name, product creation is **not** the primary responsibility of the creator. Usually, the creator class already has some core business logic related to products. The factory method helps to decouple this logic from the concrete product classes. Here is an analogy: a large software development company can have a training department for programmers. However, the primary function of the company as a whole is still writing code, not producing programmers.



1 The **Product** declares the interface, which is common to all objects that can be produced by the creator and its subclasses.

2 **Concrete Products** are different implementations of the product interface.

4 **Concrete Creators** override the base factory method so it returns a different type of product.

Note that the factory method doesn't have to **create** new instances all the time. It can also return existing objects from a cache, an object pool, or another source.

Group Programming

https://github.com/nds-swe/spring-starter/releases/tag/1.1.0

```
git clone --branch 1.1.0 https://github.com/nds-swe/spring-starter.git
```

Zielkontrolle

Heute nur eine Lernkontrolle, Ende Lektion 27.

WEITER WISSEN.

Wir begleiten Sie!