

Bachelor of Science (BSc) in Informatik
Modul Software-Entwicklung 1 (SWEN1)

V4 – Framework Design

SWEN1/PM3 Team:
R. Ferri (feit), D. Liebhart (lieh), K. Bleisch (bles), G. Wyder (wydg)

Ausgabe: HS24

Um was geht es?

- Ein Framework ist ein Programmiergerüst, das dem Anwendungsprogramm einen Rahmen gibt und wiederverwendbare Funktionalität zur Verfügung stellt.
- Es bietet gezielt Orte an, wo es erweitert oder angepasst werden kann.
- In Frameworks kommen gewisse Design Patterns zum Einsatz.
- Frameworks werden heutzutage sehr häufig eingesetzt.

Lernziele LE 13 – Framework Design

- Sie sind in der Lage:
 - die **Eigenschaften von Frameworks** zu nennen,
 - **Design Patterns** im Einsatz von Frameworks anzuwenden,
 - Prinzipien von **modernen Frameworks** zu verstehen,
 - die **Auswahl und den Gebrauch von Frameworks** kritisch einzuschätzen.

Agenda

1. **Einleitung und Definition**
2. Design Patterns in Frameworks
3. Fallstudie Persistenz-Framework
4. Moderne Framework Patterns
5. Wrap-up und Ausblick

Framework Charakterisierung

- Leider gibt es keine allgemein akzeptierte exakte Definition eines Frameworks und der Begriff wird für viele Programmbibliotheken eingesetzt.
- Für unsere Zwecke möchten wir den Begriff folgendermassen abgrenzen:
 - Ein Framework enthält **keinen applikationsspezifischen Code**.
 - Ein Framework gibt aber **den Rahmen («Frame») des anwendungsspezifischen Codes** vor.
 - Die **Klassen eines Frameworks arbeiten eng zusammen**, dies im Gegensatz zu einer reinen Klassenbibliothek wie z.B. die Java Collection Klassen.
 - Ein Framework muss **für den Einsatz gezielt erweitert und/oder angepasst** werden.
- Applikations-Container wie z.B. Spring Framework oder Java EE (neu Jakarta EE) schliessen wir ebenfalls ein.

Framework Entwicklung

- Die Entwicklung eines neuen Frameworks ist eine **aufwändige Angelegenheit**.
- Wiederverwendbare Software (und dazu gehören natürlich Frameworks) sollte ein **höheres Level im Bereich Zuverlässigkeit besitzen**, was ebenfalls mit zusätzlichem Aufwand verbundenen ist.
- Erweiterbare Software (und dazu gehören natürlich Frameworks) **erfordert eine tiefergehende Analyse darüber, welche Teile erweiterbar sein sollen**, was zu einem höheren Architektur- und Designaufwand führt.
- Eigentlich sprechen alle diese Punkte dagegen, **selber ein Framework zu entwickeln**. Weshalb wird dies trotzdem behandelt?

- Alle **Design Patterns**, die wir heute behandeln, können für die Entwicklung erweiterbarer Software eingesetzt werden.
- Dies muss nicht zwingend ein Framework sein, das auf GitHub publiziert wird, sondern es kann auch einfach eine **Komponente** sein, die in mehreren eigenen Anwendungen **in verschiedenen Kontexten eingesetzt** wird.
- Das Wissen um den Aufbau von Frameworks hilft auch, deren **Einsatz, aber auch deren Grenzen zu verstehen**.

Kritische Bemerkungen zu Frameworks

- Frameworks tendieren dazu, im Laufe der Zeit **immer mehr Funktionalität** zu «sammeln».
- Was auf den ersten Blick positiv scheint, kann im zweiten Blick zu **inkonsistentem Design und funktionalen Überschneidungen** führen, die den Einsatz immer mehr erschweren.
- Der **Einsatz** eines Frameworks sollte **gut überlegt werden**.
- Einerseits erfordert dies gute Kenntnisse des Frameworks, andererseits ist nach der «Verheiratung» der Anwendung mit dem Framework eine **«Scheidung» nur noch schwierig und mit hohem Aufwand möglich**.
- Allenfalls sollte das Framework **nur über eigene Schnittstellen verwendet werden** (keine direkte Abhängigkeit), was aber unter Umständen die Nützlichkeit des Einsatzes in Frage stellt.

Agenda

1. Einleitung und Definition
2. **Design Patterns in Frameworks**
3. Fallstudie Persistenz-Framework
4. Moderne Framework Patterns
5. Wrap-up und Ausblick

Recap: Aufbau Design Patterns

- Beschreibungsschema
 - Name
 - Beschreibung Problem
 - Beschreibung Lösung
 - Hinweise für Anwendung
 - Beispiele

Recap: Anwendung von Design Patterns

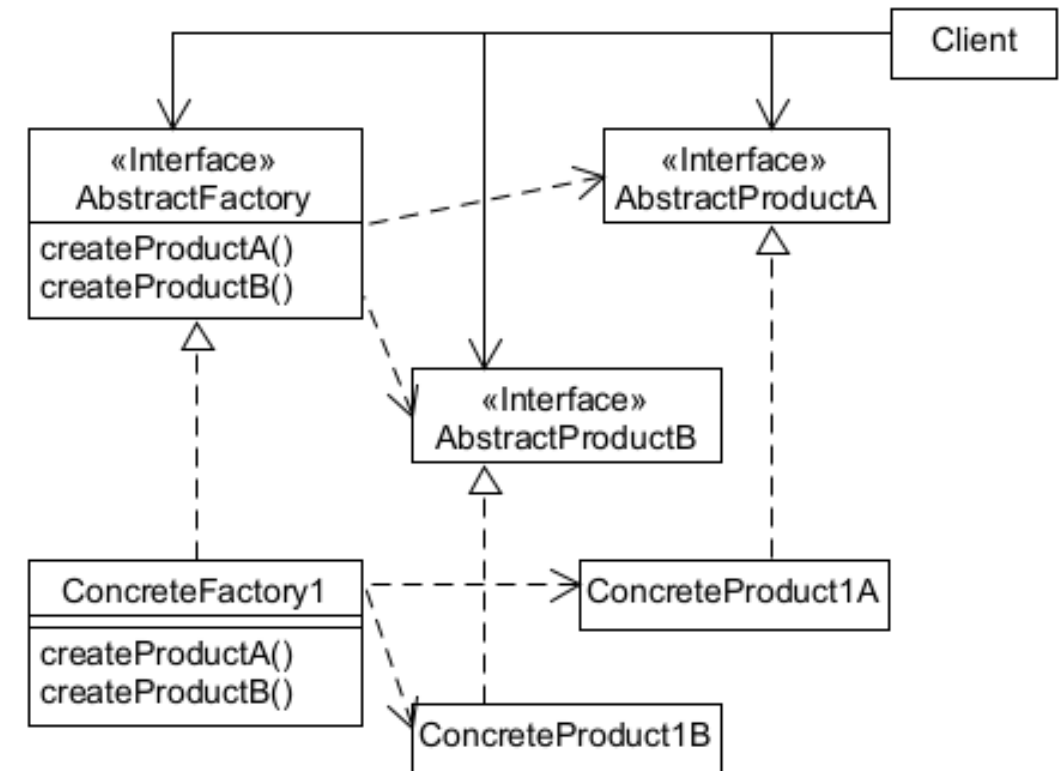
- Design Patterns sind ein wertvolles Werkzeug, um **bewährte Lösungen** für wiederkehrende Probleme schnell zu finden.
- Sie helfen, im Team effizient über Lösungsmöglichkeiten zu kommunizieren.
- Ihre Anwendung stellt aber immer einen **Trade-Off zwischen Flexibilität und Komplexität** dar.
- Es ist keineswegs so, dass ein Programm automatisch besser wird, wenn mehr Patterns angewendet werden.

Design Patterns

- Abstract Factory
- Factory Method
- Command
- Template Method

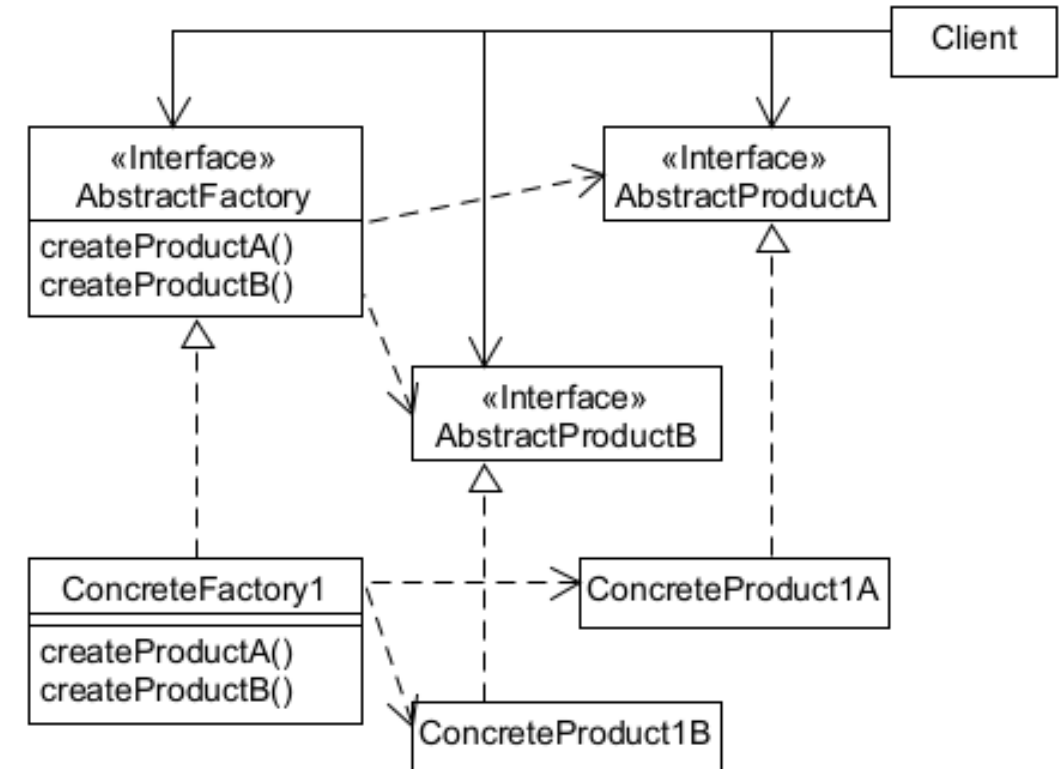
Abstract Factory: Problem und Lösung

- Problem
 - Die Erzeugung **verschiedener**, inhaltlich zusammengehörender Objekte («Product»), **ohne** aber die **konkreten** Klassen zu kennen, damit diese austauschbar sind.
- Lösung
 - Eine AbstractFactory und abstrakte «Products» definieren.
 - Die AbstractFactory hat für jedes «Product» eine eigene «create» Methode.
 - Eine konkrete Factory davon ableiten, die dann konkrete «Products» erzeugt.



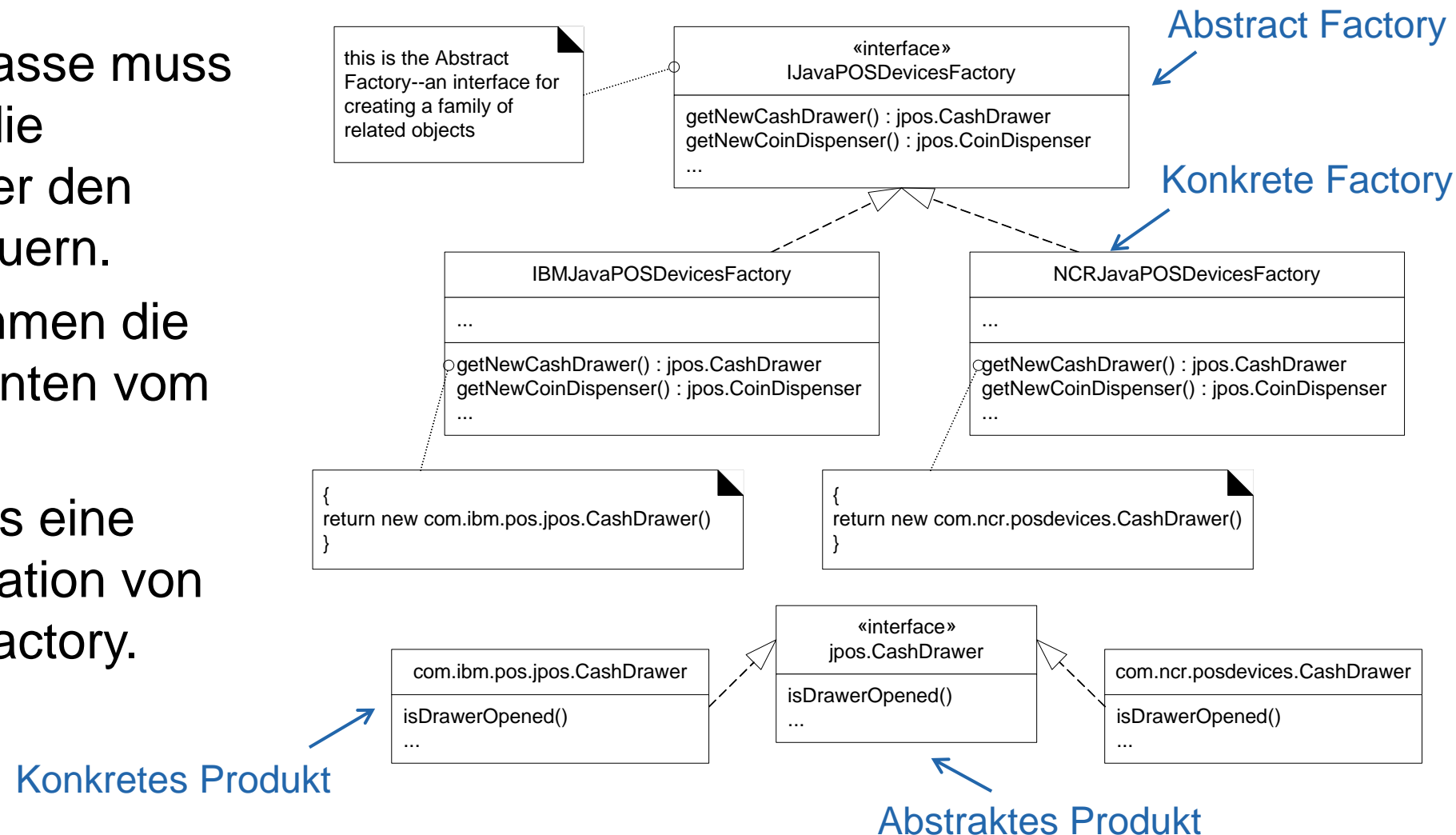
Abstract Factory: Hinweise

- Hinweise
 - Eigentlich «nur» eine Verallgemeinerung einer «SimpleFactory».
 - Die verschiedenen Produkte hängen inhaltlich miteinander zusammen, zum Beispiel verschiedene Teile der anzusteuern Hardware.



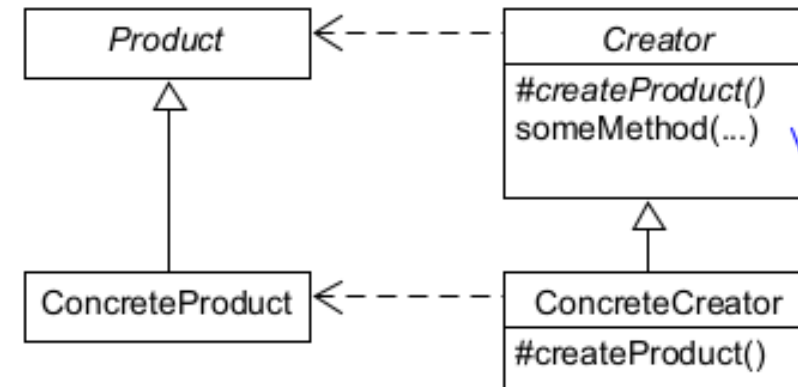
Abstract Factory: Beispiel Point Of Sale (POS) Terminal

- Die elektronische Kasse muss Hardware wie z.B. die Notenschublade oder den Münzspender ansteuern.
- Typischerweise kommen die einzelnen Komponenten vom selben Hersteller.
- Pro Hersteller gibt es eine konkrete Implementation von IJavaPOSDevicesFactory.



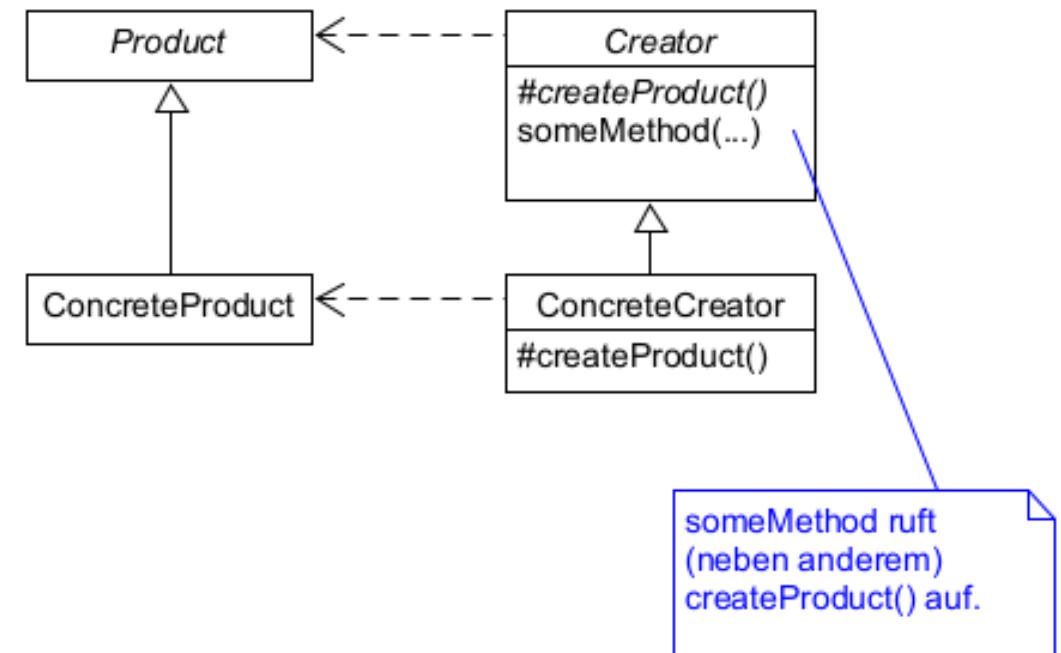
Factory Method: Problem und Lösung

- Problem
 - Eine (wiederverwendbare) Klasse *Creator* hat die Verantwortlichkeit, eine Instanz der Klasse *Product* zu **erzeugen**. Es ist aber klar, dass *Product* noch **spezialisiert** werden muss.
- Lösung
 - Eine abstrakte Methode in der Klasse *Creator* definieren, die als Resultat *Product* zurückliefert.
 - Konkrete Klassen von *Creator* können dann die richtige Subklasse von *Product* erzeugen.



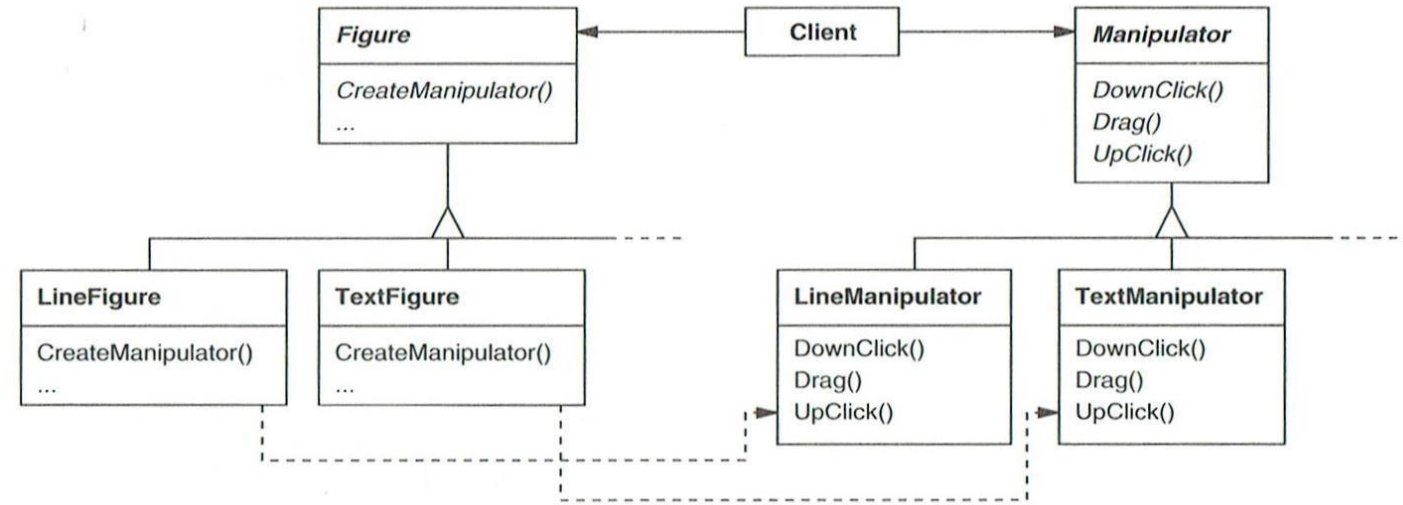
Factory Method: Hinweise

- Hinweise
 - Es ist durchaus erlaubt, dass bereits Creator und Produkt konkret sind und somit eine Basisfunktionalität zur Verfügung stellen.
 - Es gibt parallele Vererbungshierarchien mit Creator wie auch Product an der Spitze.
 - Kann auch als Variante des Design Patterns «Template Method» interpretiert werden.



Factory Method: Beispiel GoF (2/2)

- Das Zeichenprogramm («Client») besitzt eine Klassen-Hierarchie von Figuren.
- Um Figuren übers UI verändern zu können, gibt es eine abstrakte Manipulator Klasse.
- Jede konkrete Figur hat nun die Aufgabe, seine Manipulator Klasse zu instanziiieren.



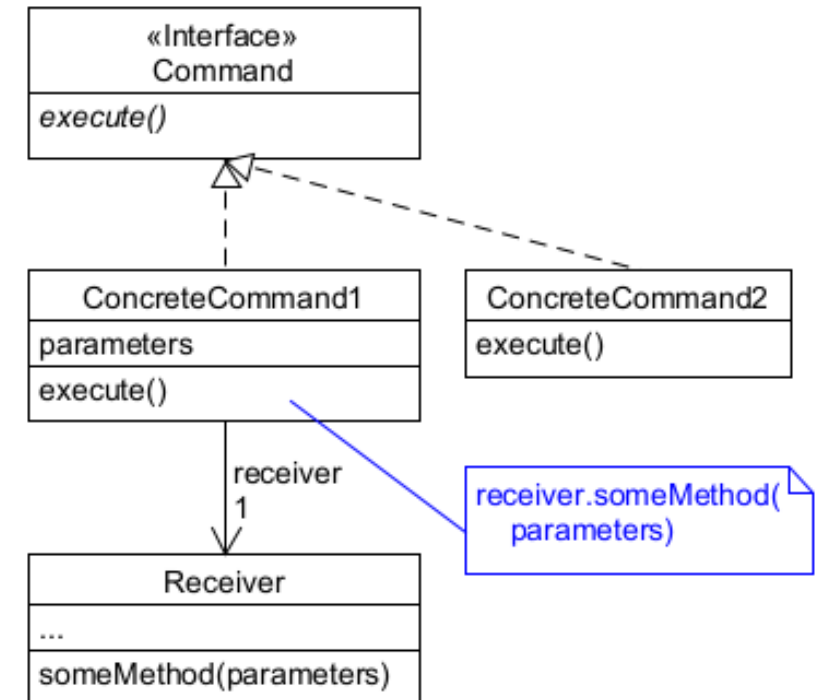
Command: Problem und Lösung

- Problem

- Aktionen müssen für einen späteren Gebrauch **gespeichert** werden und dabei können sie noch allenfalls priorisiert oder protokolliert werden und/oder Unterstützung für ein **Undo** anbieten.

- Lösung

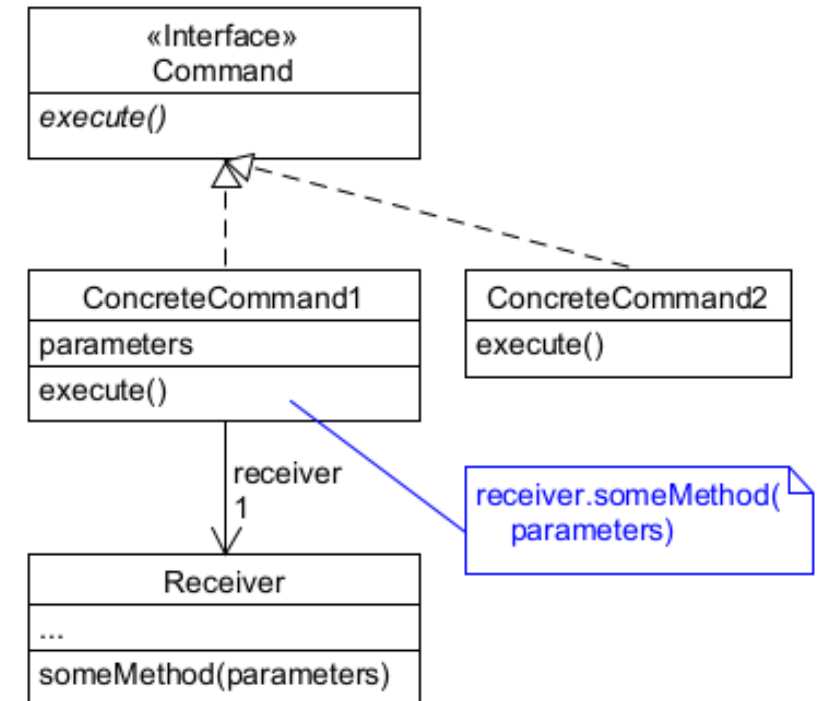
- Ein Interface wird definiert, das nur die Auslösung der Aktion erlaubt.
- Implementationen dieses Interface überschreiben die Methode zur Auslösung der Aktion.
- Meistens bedeutet die Aktion, dass eine Methode auf einem anderen Objekt aufgerufen wird.
- Dazu muss die Aktion die Parameter dieser Methode zwischenspeichern.



Command: Hinweise

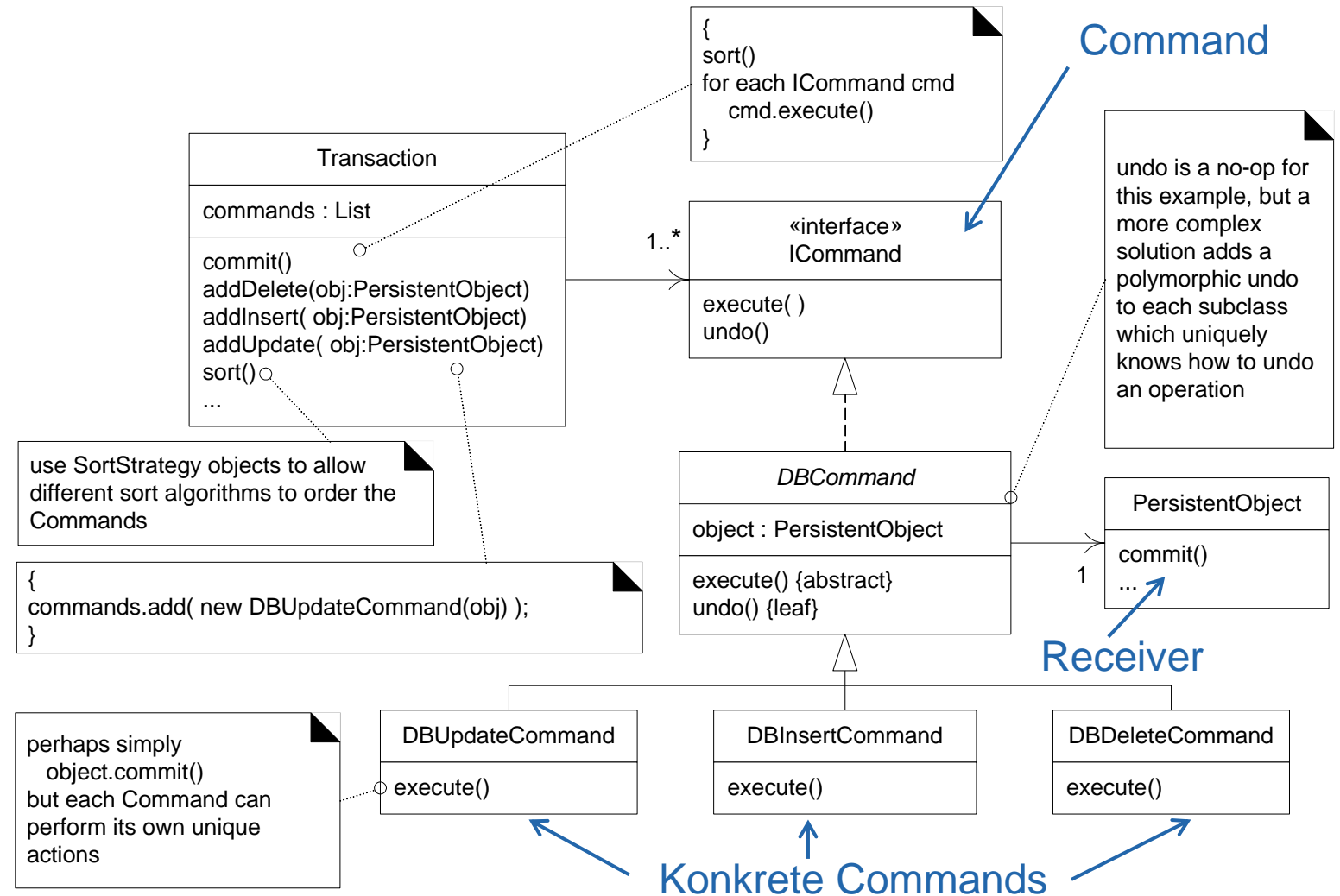
- Hinweise

- Erstellung der Aktion und das Auslösen liegen zeitlich auseinander.
- Bevor Aktionen ausgelöst werden, können sie bei Bedarf noch sortiert oder priorisiert werden. Denken Sie dabei an eine Datenbank.
- Der Receiver muss nicht zwingend über eine Assoziation sichtbar sein. Es ist auch ein Lookup über z.B. einen Namen denkbar.
- Falls eine Rückabwicklung der Aktion («Undo») notwendig ist, kann die entsprechende Methode direkt in der Aktion eingefügt werden oder es gibt eine separate Aktion dafür.



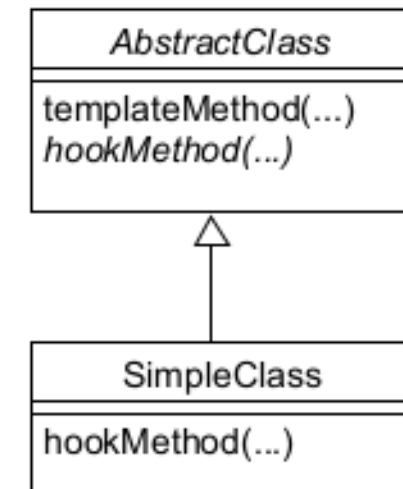
Command: Beispiel Point Of Sale Terminal

- Eine Transaktion eines Persistenz-Frameworks setzt sich aus den Aktionen für jedes veränderte Objekt zusammen.
- Aktionen sind update, insert und delete.
- Eine undo Methode ist ebenfalls vorhanden.



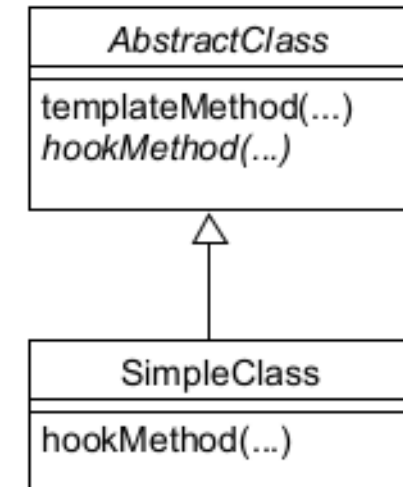
Template Method: Problem und Lösung

- Problem
 - Ein Ablauf/Algorithmus soll so entworfen werden, dass er in gewissen Punkten angepasst werden kann.
- Lösung
 - In einer abstrakten Klasse wird eine Template Method hinzugefügt, die diesen Ablauf/Algorithmus implementiert.
 - Die Template Method ist fertig geschrieben, ruft aber noch abstrakte Methoden («hookMethod») auf.
 - Diese Methoden dienen als Variations- resp. Erweiterungspunkte und mit ihrer Implementation kann der Ablauf/Algorithmus auf den aktuellen Kontext angepasst werden.



Template Method: Hinweise

- Hinweise
 - Die «hookMethod» kann entweder rein abstrakt sein oder bereits eine Standard-Implementation enthalten.
 - Eine Factory Method kann in diesem Zusammenhang ebenfalls als «hookMethod» interpretiert werden.
 - Es ist nicht einfach, im Voraus alle Orte zu identifizieren, wo Anpassungen notwendig sein müssen.
 - Verwandtschaft mit einer Strategy. Eine Strategy benutzt Delegation, um einen ganzen Algorithmus zu variieren, während Template Method Vererbung benutzt, um einen Teil des Algorithmus zu variieren.
 - **Hollywood Prinzip: «Don't call us, we call you».** Der eigene Code wird von fremdem Code aufgerufen (oder: der Code des Frameworks ruft den Code der Umsetzung auf).



Template Method: Beispiel Larman GUI Framework

- Ein GUI Framework stellt Komponenten zur Verfügung.
- Die Basisklasse GUIComponent stellt die Template Method update() zur Verfügung, die repaint() aufruft.
- Die Methode repaint() muss dann von unserer Klasse überschrieben werden.

```
// this is the template method
// its algorithm is the unvarying part

public void update()
{
    clearBackground();

    // this is the hook method
    // it is the varying part
    repaint();
}
```

Template
Method

hook method

- varying part
- overridden in subclass
- may be abstract, or have a default implementation

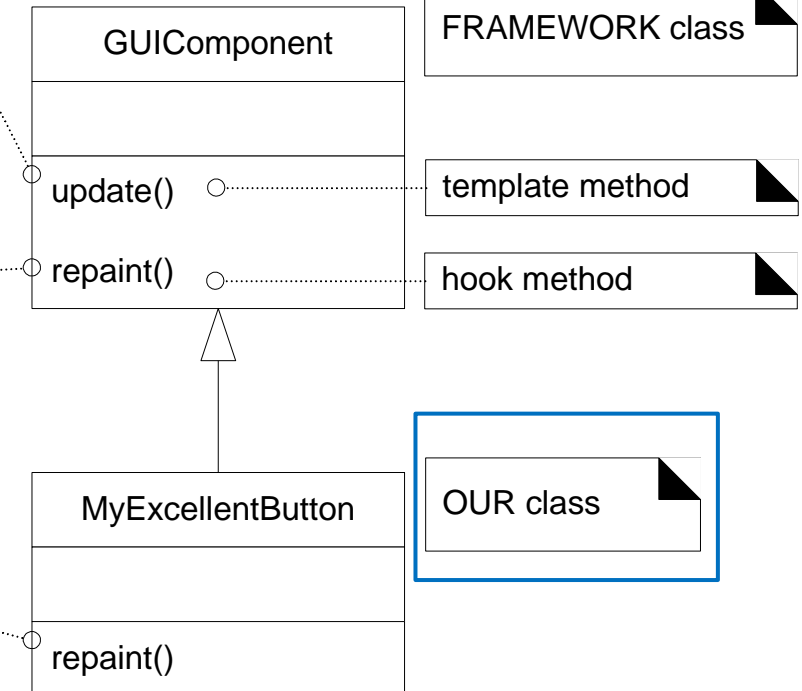
Hook
Method

hook method overridden

- fills in the varying part of the algorithm

HOLLYWOOD PRINCIPLE:
Don't call us, we'll call you

Note that the MyExcellentButton--repaint method is called from the inherited superclass update method. This is typical in plugging into a framework class.



Agenda

1. Einleitung und Definition
2. Design Patterns in Frameworks
- 3. Fallstudie Persistenz-Framework**
4. Moderne Framework Patterns
5. Wrap-up und Ausblick

Einleitung Persistenz-Framework im Buch von Larman

- Framework für Speicherung von Objekten (siehe [1] Kap. 38).
- Primäres Ziel: Prinzipien des Framework Designs zeigen.
- Sekundäres Ziel: Problemstellungen von Persistenz-Frameworks und mögliche Lösungsansätze zeigen.
- Was fehlt?
 - Eigentliche RDB-Zugriffe. Im Buch werden verschiedene Lösungen skizziert.
 - Eigentliche Behandlung von Collections und Assoziationen. Im Buch wird dafür die Verwendung vom Design Pattern «Virtual Proxy» erwähnt.
 - Abfragen (Queries) werden gar nicht behandelt. Da ja beliebige Speichertechnologien unterstützt werden sollen, ist dies aber auch nicht verwunderlich.
 - Der vollständige Programmcode.

Themen Persistenz-Framework von Larman

- Persistenz-Fassade
- Mapping auf RDB
- Mapper für jede Klasse
- Objekt-Identifikation
- Verfeinerung Mapper
- Zustandsverwaltung bezüglich Transaktionen
- Proxy für Lazy Loading von referenzierten Objekten

Agenda

1. Einleitung und Definition
2. Design Patterns in Frameworks
3. Fallstudie Persistenz-Framework
4. **Moderne Framework Patterns**
5. Wrap-up und Ausblick

Moderne Framework Patterns

- Die bewährten Design Patterns finden nach wie vor ihre Anwendung im Framework Design.
- In den letzten Jahren wurden aber noch weitere Mechanismen populär.
 - **Dependency Injection**, meistens gesteuert über Annotationen
 - **Convention over Configuration**: Nur durch das Einhalten von (Namens-)Konventionen wird das Framework aktiv und macht das Gewünschte.
 - **Implementation von Interfaces basierend auf den Methoden des Interfaces** (z.B. Spring Data Repository-Interfaces). Der Methodenname spezifiziert sozusagen seine Implementation, allenfalls noch ergänzt mit Annotationen

Annotationen

- Ist ein Standard Java-Sprachelement ab Java 5 (z.B. `@override`).
- Können selber deklariert werden.
- Werden «normalen» Sprachelementen hinzugefügt
- Vorteil: Wenn beim Laden einer annotierten Klasse die Annotations-Klasse nicht gefunden wird, gibt es keine Fehlermeldung, sondern die Annotation wird stillschweigend entfernt.
- Anders gesagt fügen Annotationen keine harte Abhängigkeit hinzu und sind somit geeignet, die Domänenlogik frei von ungewünschten (technischen) Abhängigkeiten zu halten.

Steuerung über Annotationen

- Annotationen per se haben ja keine Funktionalität. Es braucht «jemand», der die Annotationen liest und dann Aktionen ausführt.
- Auswertung von Annotationen:
 - Beim Starten der Anwendung wird das Framework ebenfalls gestartet.
 - Das Framework sucht die Anwendungsklassen auf dem Klassenpfad ab, untersucht allfällige Annotationen und führt die gewünschten Aktionen aus.
- Mögliche Aktionen des Frameworks:
 - Dependency Injection von Framework Objekten in Anwendungsobjekte (über Constructor oder Set-Methode).
 - Automatisches Implementieren von Interfaces.
 - Hinzufügen von Funktionalität zu Anwendungsklassen.
- Achtung: Dieser Vorgang kann zu unerwünschten Verzögerungen beim Start führen.

Java Mechanismen für das Hinzufügen von Funktionalität

- 2 Zeitpunkte
 - Während (respektive am Schluss) der Kompilierung über einen AnnotationProcessor.
 - Beim Starten einer Anwendung können Anwendungsklassen beim Laden (über einen Framework-Classloader) noch verändert werden.
- Was wird verändert
 - Quellcode hinzufügen.
 - Bytecode hinzufügen und bestehenden abändern.
 - Für das Implementieren von Interfaces kann `java.lang.reflect.Proxy` eingesetzt werden.
- Wer verändert
 - AnnotationProcessor kann Quellcode und Bytecode hinzufügen.
 - Beim Starten einer Anwendung kann Byte Code verändert und hinzugefügt, sowie die Proxy Klasse angewendet werden.

Agenda

1. Einleitung und Definition
2. Design Patterns in Frameworks
3. Fallstudie Persistenz-Framework
4. Moderne Framework Patterns
5. **Wrap-up und Ausblick**

Wrap-up

- Gerade Frameworks müssen **sorgfältig** mit **bewährten Design Patterns** entworfen werden.
- Traditionelle Framework Patterns sind die **Template Methode** und die **Factory Method**, die es erlauben, dass in Framework Klassen ein Algorithmus realisiert wird, der aber in anwendungsspezifischen, abgeleiteten Klassen noch an den aktuellen Kontext **angepasst** werden kann.
- Das **Command** Pattern erlaubt es, dass das Framework anwendungsspezifischen Code aufrufen kann, ohne dass das Framework angepasst werden muss.
- **AbstractFactory** dient dazu, die Erzeugung einer Familie verwandter Objekte zu ermöglichen.
- Larman hat die Grundzüge eines Persistenz-Frameworks in seinem Buch entworfen, das didaktischen Zwecken dient und den Entwurf eines Frameworks an einem umfangreicheren Beispiel demonstriert.
- Moderne Frameworks setzen auf die Steuerung durch **Annotationen**, vor allem für **Dependency Injection**.

Ausblick

- In der nächsten Lerneinheit werden wir:
 - den ganzen Stoff SWEN1 kurz repetieren und
 - eine alte Semesterendprüfung (SEP) gemeinsam lösen.

Quellenverzeichnis

- [1] Larman, C.: UML 2 und Patterns angewendet, mitp Professional, 2005
- [2] Seidel, M. et al.: UML @ Classroom: Eine Einführung in die objektorientierte Modellierung, dpunkt.verlag, 2012
- [3] Martin, R. C.: Clean Architecture: A Craftsman's Guide to Software Structure and Design, mitp Professional, 2018