Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering
InIT Institut für angewandte
Informationstechnologie

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,
  - Prinzipen 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?

Framework Einsatz und Entwicklung erweiterbarer Software

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- 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

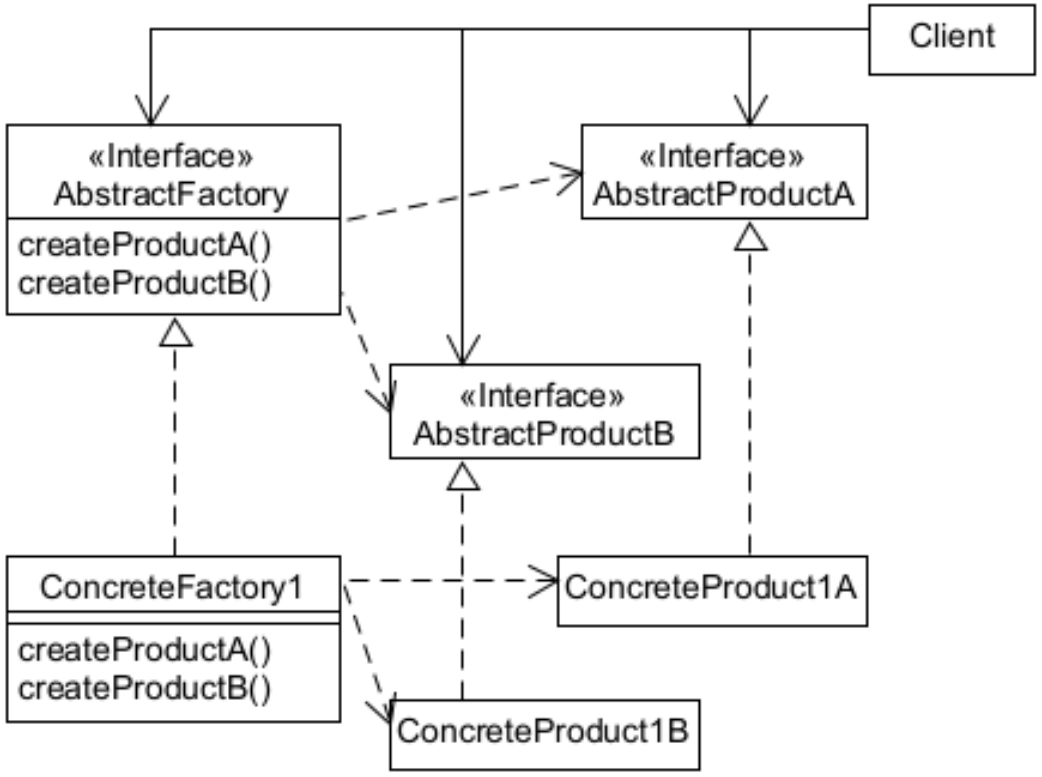
- Design Patterns sind ein wertvolles Werkzeug, um bewährte Lösungen für wiederkehrende Probleme schnell zu finden.
- Sie helfen, im Team effizient über Lösungsmöglichkeiten zu kommunizieren.
- Ihre Anwendung stellt aber immer einen Trade-Off zwischen Flexibilität und Komplexität dar.
- Es ist keineswegs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

Design Patterns

- Abstract Factory
- Factory Method
- Command
- Template Method

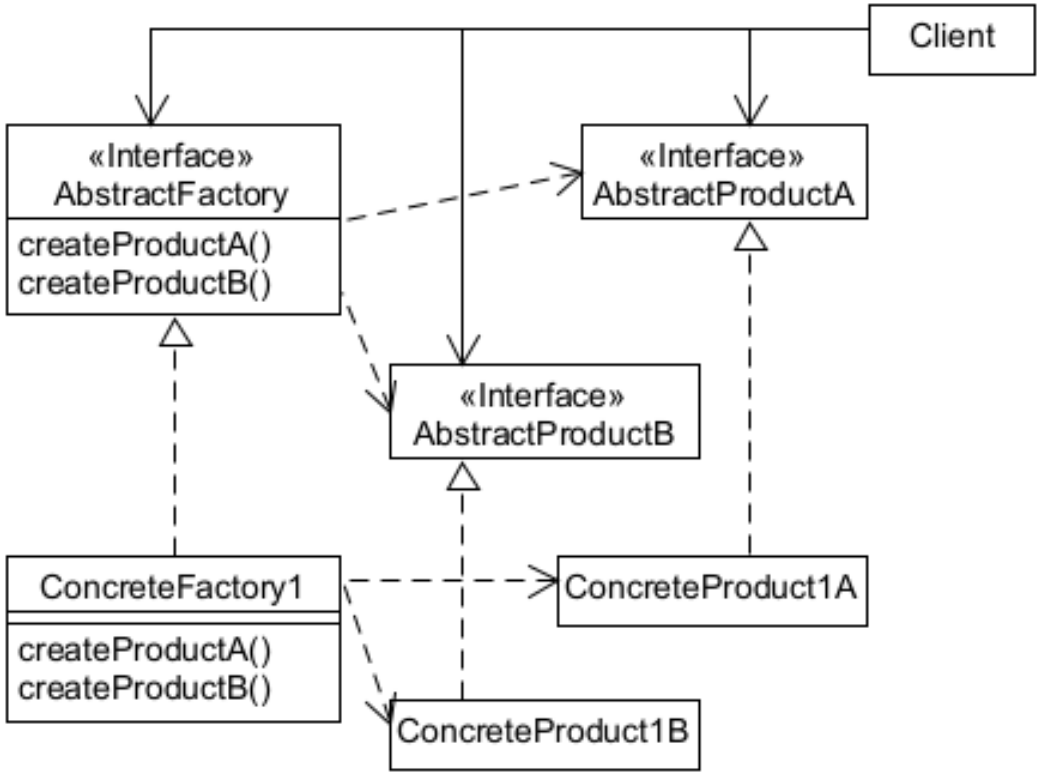
Abstract Factory: Problem und Lösung

- 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 anzusteuernden Hardware.

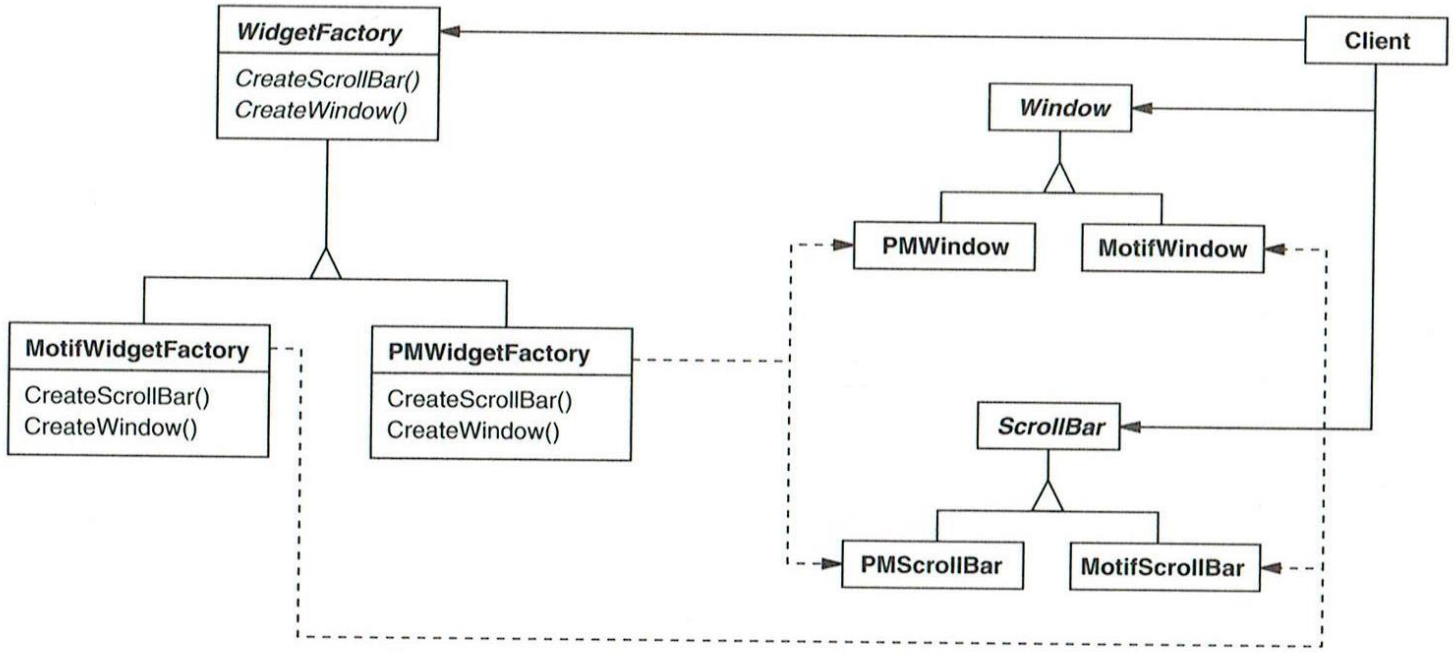


Abstract Factory: Beispiele

- JDK
  - java.awt.Toolkit ist eine AbstractFactory, die die plattformspezifischen AWT Komponenten (genauer gesagt deren «Peer») erzeugt. So gibt es für Windows, Linux und weitere GUI Plattformen konkrete Implementationen von java.awt.Toolkit
- GoF Beispiel
  - Abstract Factory für GUI Widgets. Eigentlich ganz ähnlich wie der JDK Toolkit.
- Larman, Point Of Sale Terminal (siehe nachfolgende Folie)
  - AbstractFactory für die Hardware Komponenten der elektronischen Kasse, die angesteuert werden müssen. Dazu gehört die Noten-Schublade oder der automatische Münzspender.

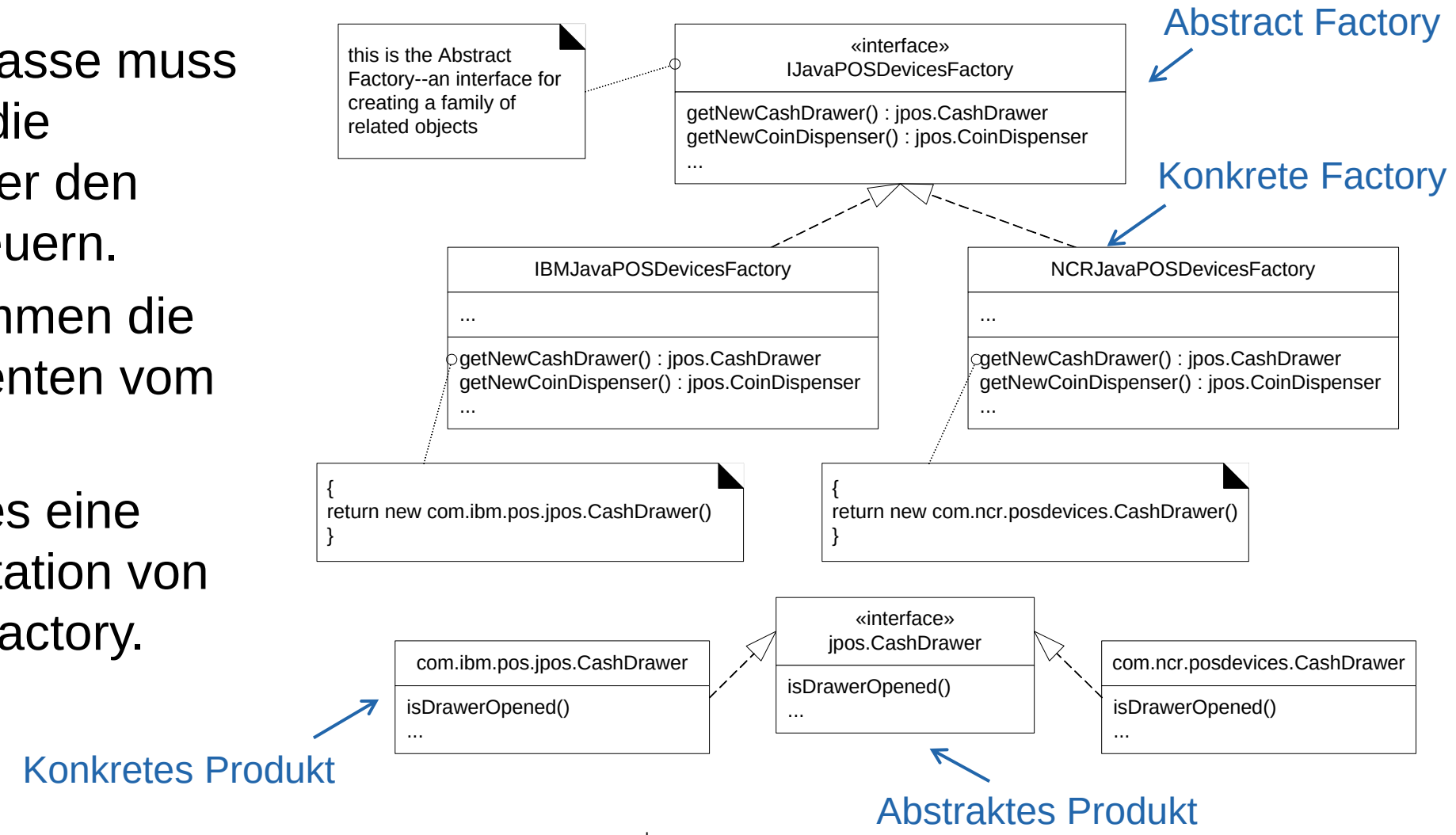
Abstract Factory: Beispiel GoF

- Pro Widget gibt es in der WidgetFactory (Abstract Factory) eine create Methode.
- Die Widgets selber sind ebenfalls abstrakt.
- Pro GUI Plattform gibt es eine konkrete Implementation von WidgetFactory, die als Resultat konkrete Implementationen der Widgets zurückliefert.



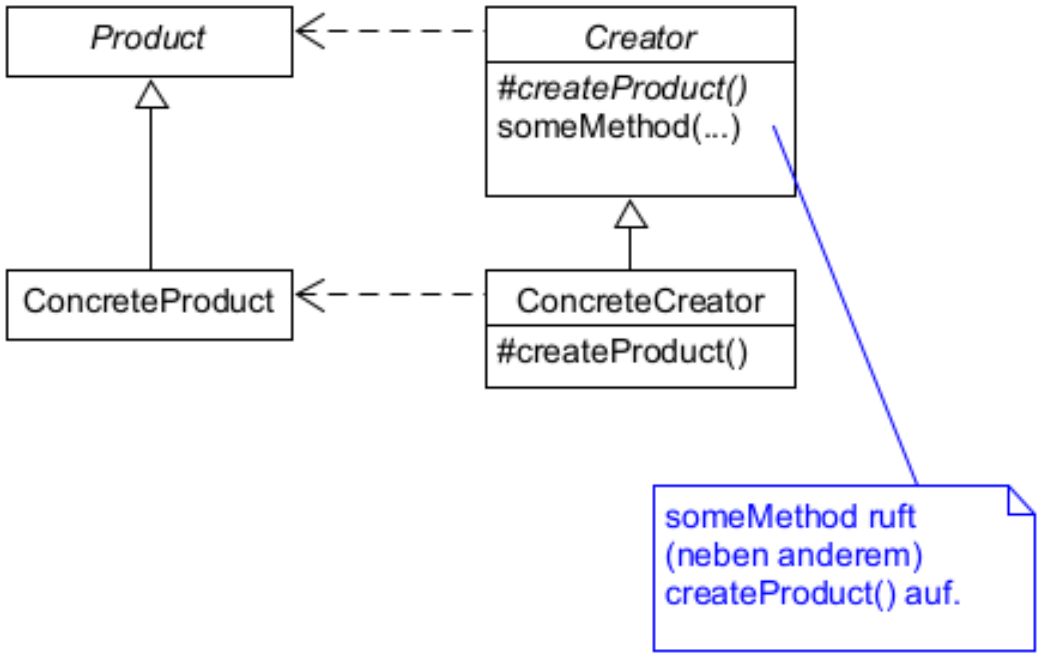
Abstract Factory: Beispiel Point Of Sale (POS) Terminal

- 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.



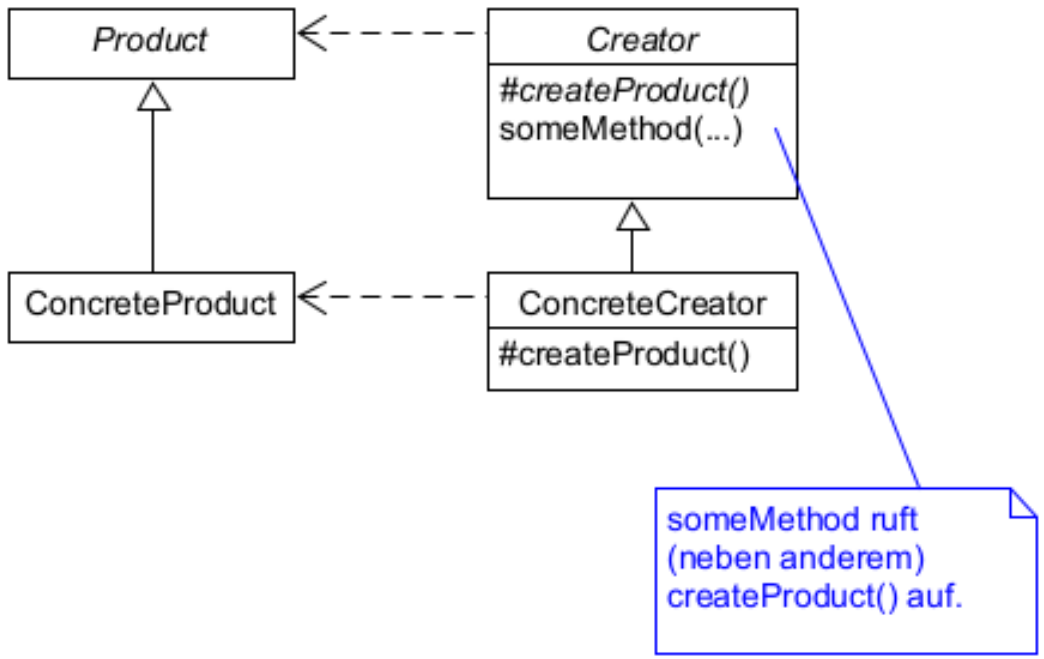
Factory Method: Problem und Lösung

- 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.

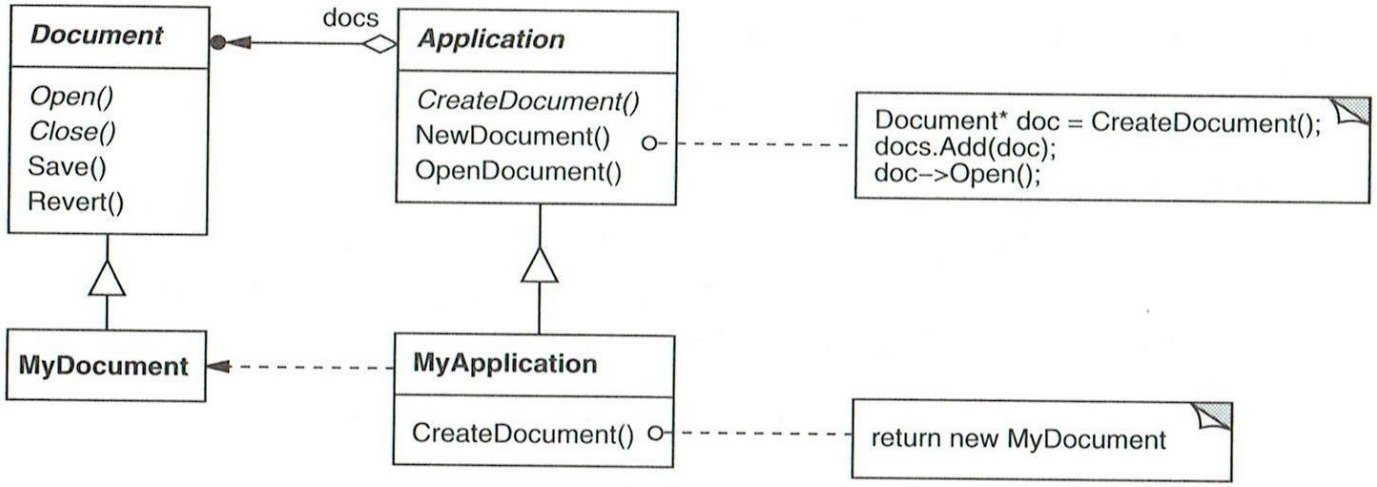


Factory Method: Beispiele

- JDK
  - java.net.URLStreamHandler ist eine abstrakte Superklasse für alle URL Protokoll Handler und besitzen eine abstrakte Methode openConnection(URL), die eine URL Connection zurückliefert. Konkrete Stream Handler implementieren diese Methode und liefern protokollspezifische, abgeleitete Klassen von URLConnection zurück.
- GoF Beispiel
  - Eine Application Klasse erzeugt ein Dokument.
  - Zeichenprogramm, mit dem Figuren (Linie, Text) verändert werden können.

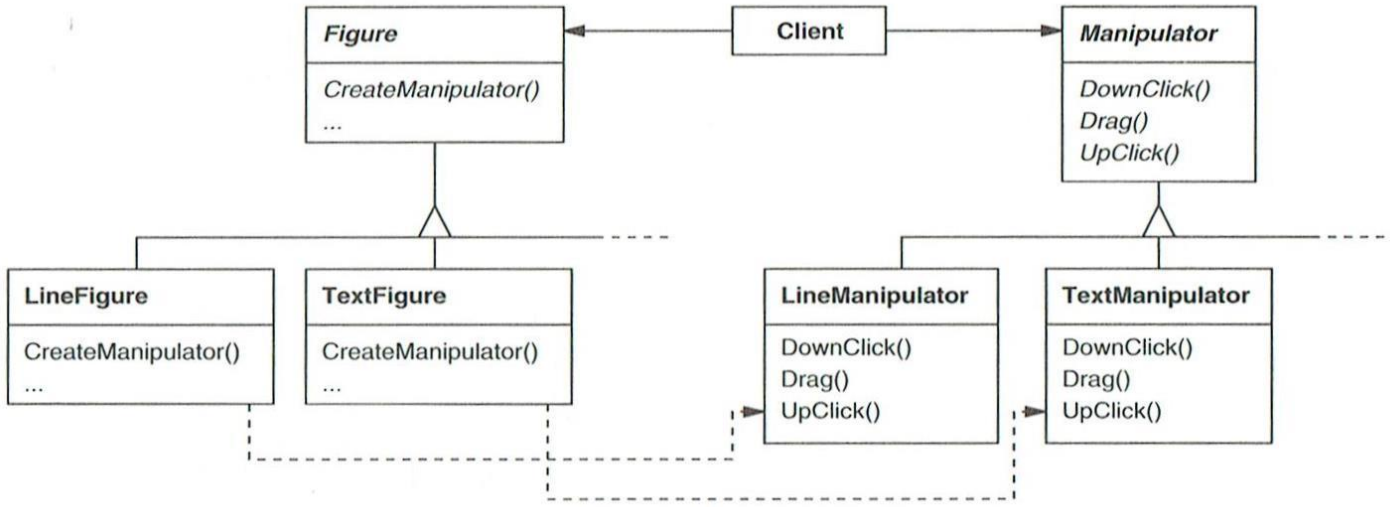
Factory Method: Beispiel GoF (1/2)

- Document und Application sind abstrakte Klassen und Teil eines Frameworks.
- Gewisse Methoden sind konkret, andere abstrakt.
- Über CreateDocument kann nun MyApplication das richtige MyDocument erzeugen.



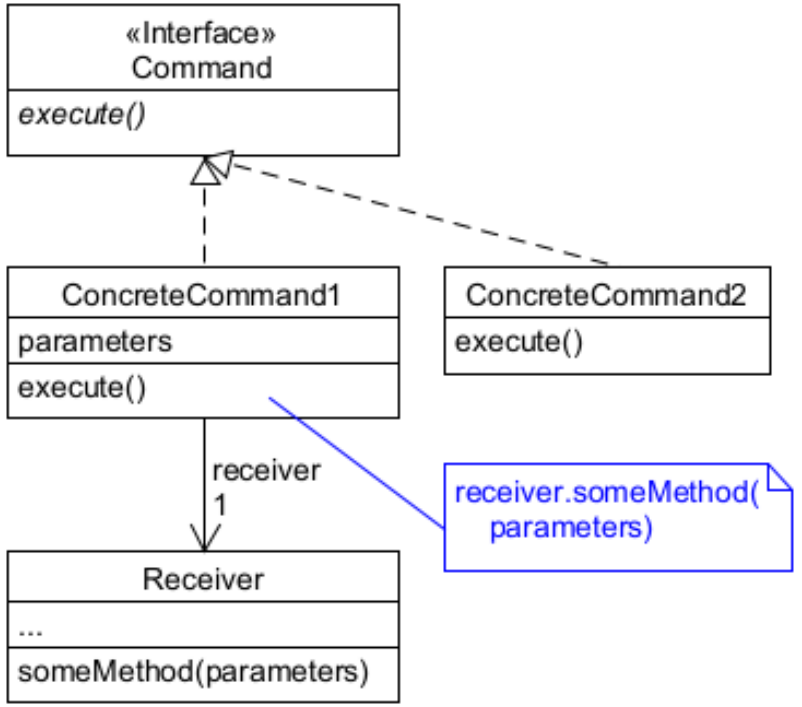
Factory Method: Beispiel GoF (2/2)

- 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 instanziieren.



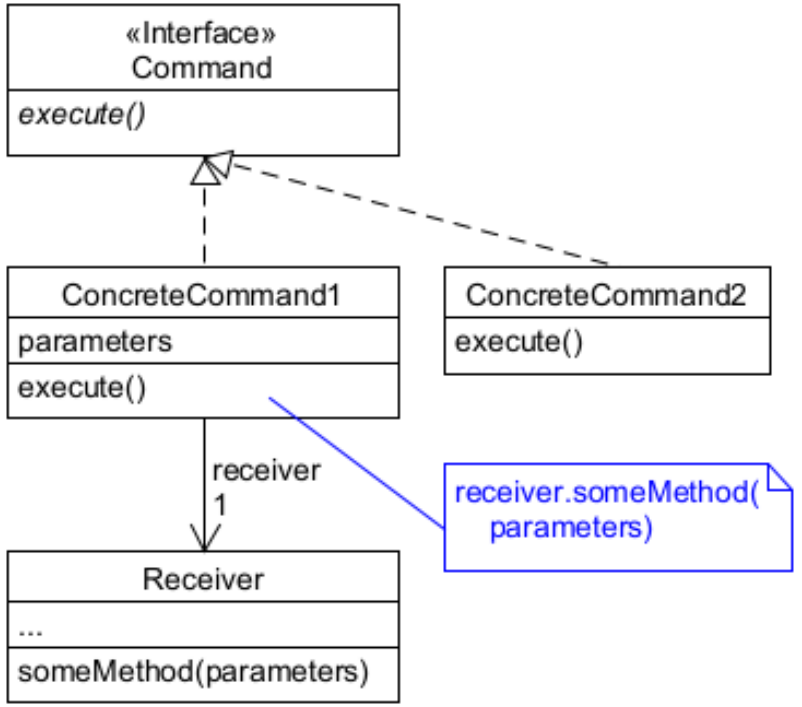
Command: Problem und Lösung

- 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.

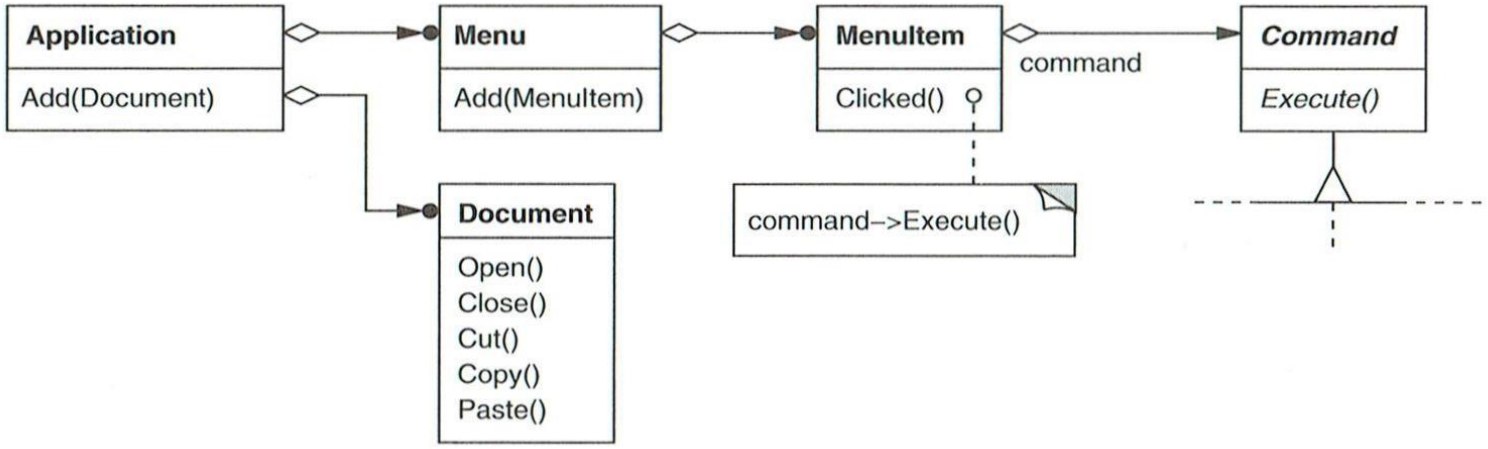


Command: Beispiele

- JDK
  - AWT: java.awt.event.ActionListener Implementationen sind alle Commands. Hervorzuheben ist dabei das Interface java.swing.Action, das noch weitere Attribute für den Command bereitstellt.
  - JavaFX: javafx.event.EventHandler.
- GoF Beispiel
  - Commands, die vom Menü eines GUI Frameworks aufgerufen werden.
- Larman
  - Die Aktionen, die beim Commit einer Datenbank-Transaktion ausgelöst werden, sind mit den Command Pattern realisiert. Dabei wird eine Undo Methode direkt im Command Objekt eingefügt.

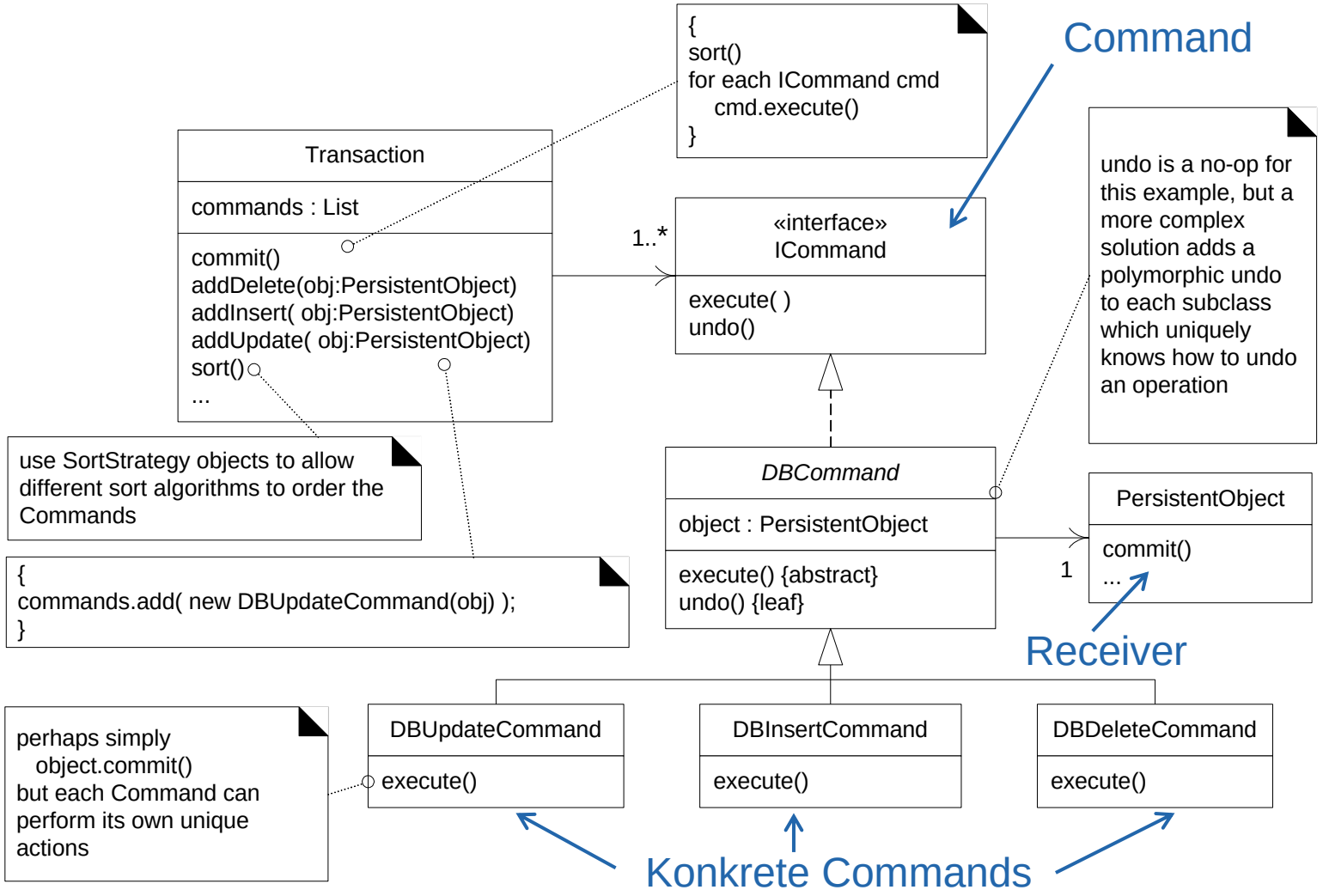
Command: Beispiel GoF

- Commands führen die Aktionen aus, die von einem MenuItem ausgelöst werden.



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.



Denkpause

Aufgabe 13.1 (5’)

Diskutieren Sie in Murmelgruppen folgende Fragen:

- Warum müssen die Commands vor dem Commit der Transaktion noch sortiert werden?
- Wie läuft der Commit ab?
- Was passiert im Fehlerfall?

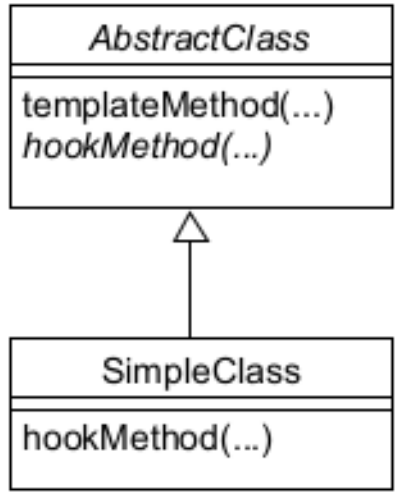
Denkpause

Aufgabe 13.1 – Musterlösung

1. Allfällige Referenzielle Integritäten berücksichtigen.
2. Zuerst sortieren, so dass Integritäten berücksichtigt werden, dann ausführen
3. Alle bereits ausgeführten Aktionen mit dem Aufruf von undo() zurücksetzen. (Und was macht man, wenn im undo() ein Fehler auftritt? DB-Transaktion abrechen und dann den Benutzer informieren, dass ein interner Fehler passiert ist und die Anwendung geschlossenen werden muss).

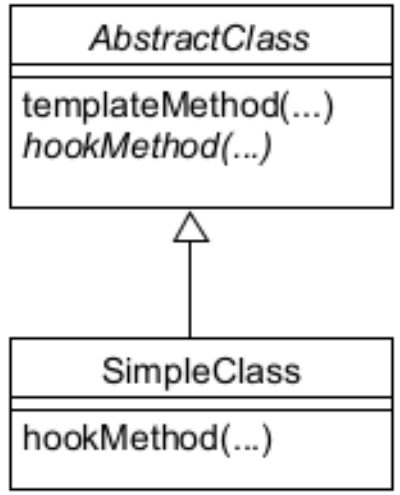
Template Method: Problem und Lösung

- 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).

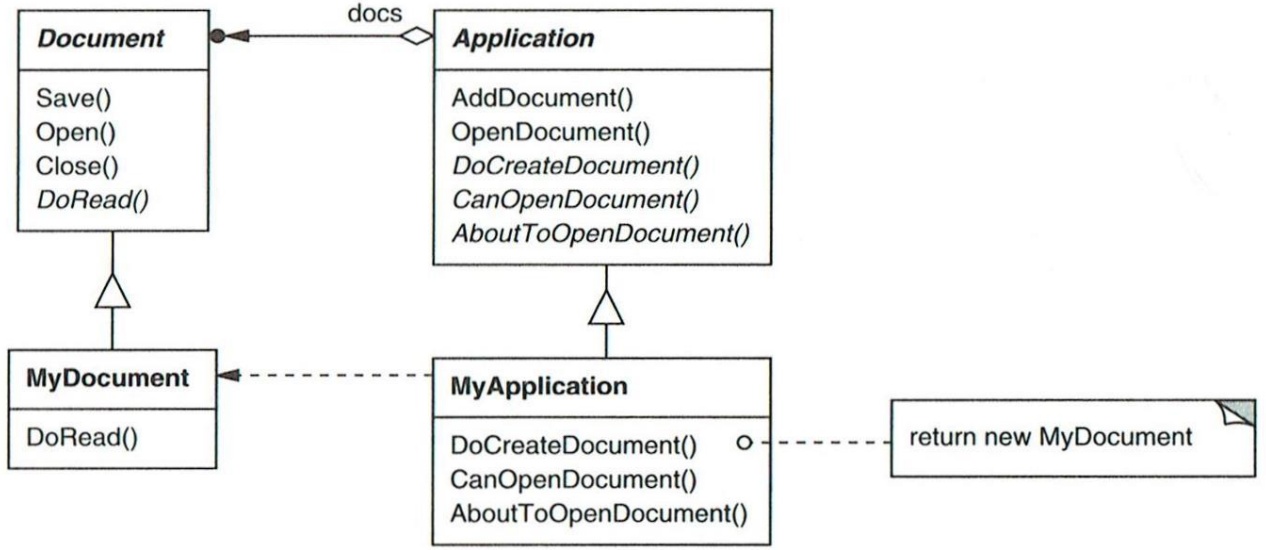


Template Method: Beispiele

- JDK
  - Collection-Klassen: Es gibt viele abstrakte Klassen in java.util, wie zum Beispiel AbstractList, die konkrete Methoden («Template Method») besitzen, die basierend auf abstrakten Methoden einen Grossteil der Funktionalität implementieren.
- GoF Beispiel
  - Applikations-Framework, das eine Application- und Document Klasse zur Verfügung stellt. Das Öffnen eines Document ist die «Template Method», die gewisse Aufgaben an abstrakte Methoden weiterleitet.
- Larman
  - Ausschnitt eines GUI Framework
  - Mapper Klassen eines Persistenz-Framework, die aus einer Datenquelle Objekte erzeugen.

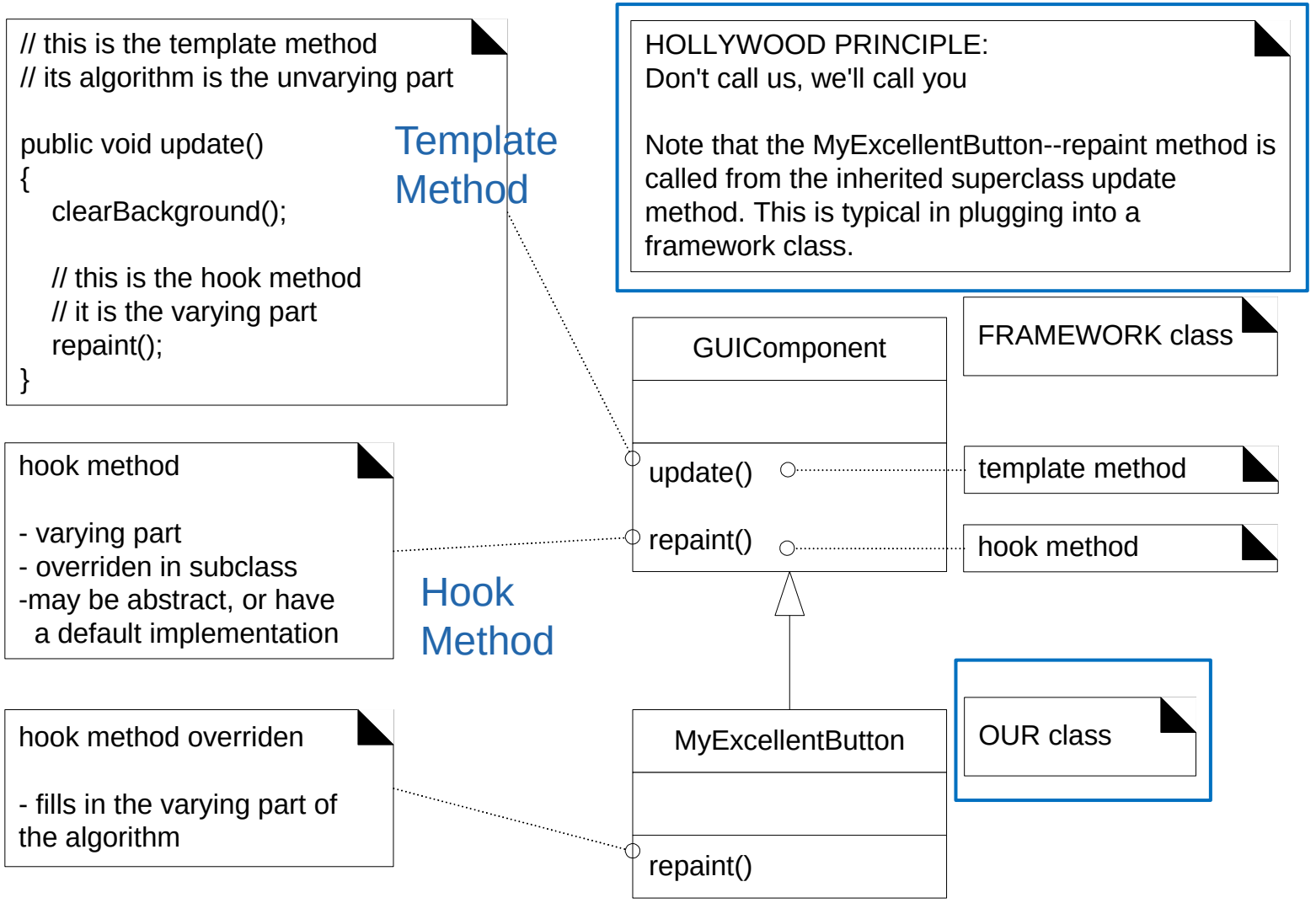
Template Method: Beispiel GoF

- Document und Application sind Framework-Klassen.
- OpenDocument(name) ist die Template Method. Sie fragt mittels CanOpenDocument(…) zuerst ab, ob das Dokument geöffnet werden kann, erzeugt dann die Document Instanz und ruft dann AboutToOpenDocument(doc) auf.
- DoCreateDocument() ist eine Factory Method.



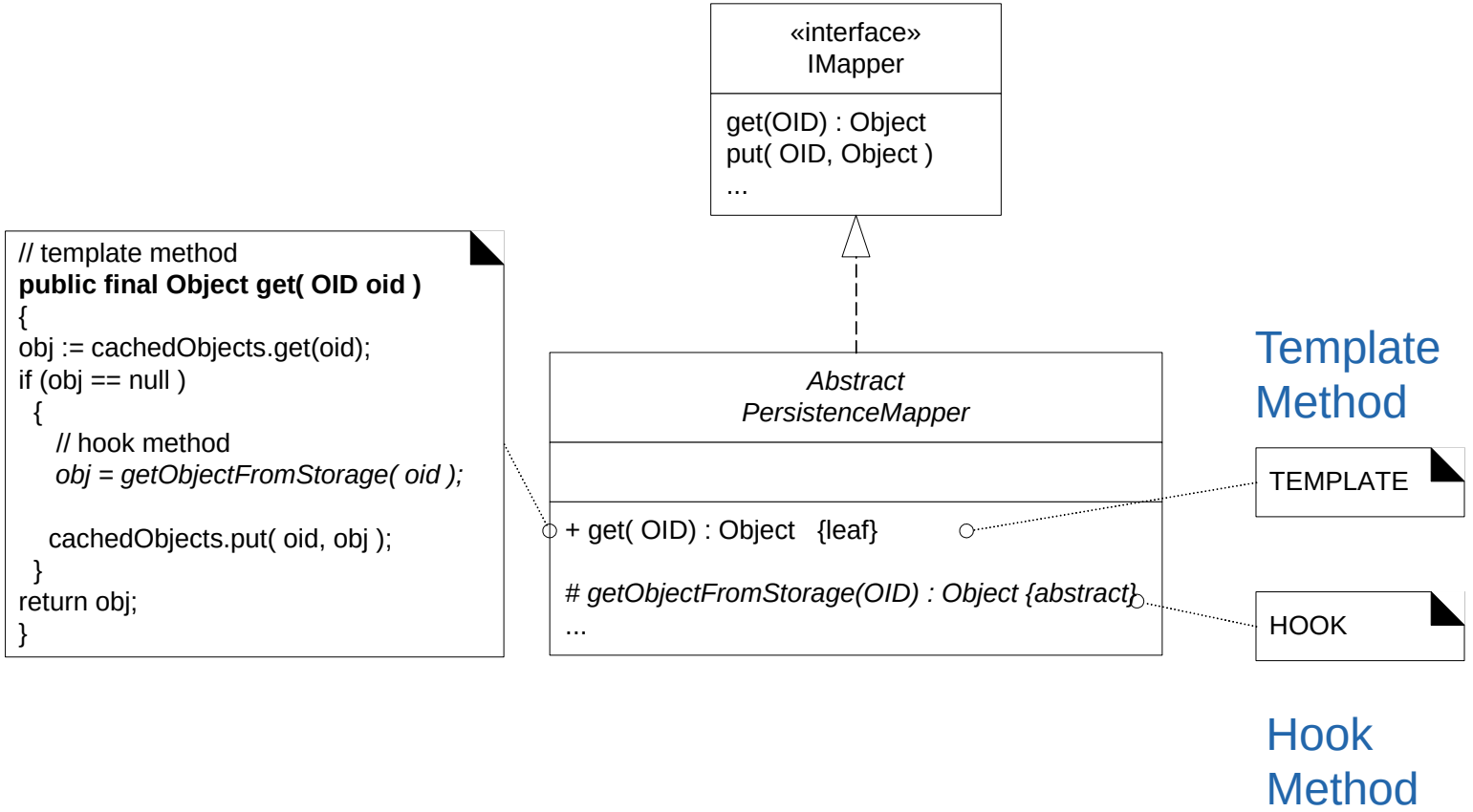
Template Method: Beispiel Larman GUI Framework

- 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.



Template Method: Beispiel Larman Persistenz (1/2)

- Eine IMapper Klasse ist für das Erzeugen und Aktualisieren von einer Anwendungsklasse verantwortlich.
- Der Basisalgorithmus ist in AbstractPersistenceMapper.
- Die Hook Methode getObjectFromStorage(OID) muss dann in abgeleiteten Klassen überschrieben werden.

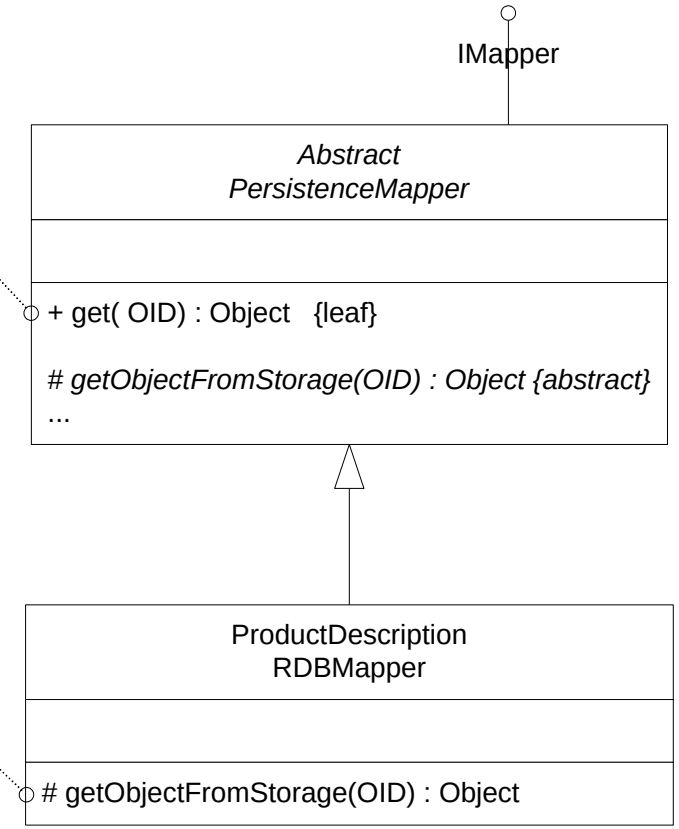


Template Method: Beispiel Larman Persistenz (2/2)

- In ProductDescription-RDBMapper wird getObjectFromStorage(OID) überschrieben, indem ein SQL Statement ausgeführt wird und mit dem Resultat ein Objekt von ProductDescription erzeugt wird.

```
// template method
public final Object get( OID oid )
{
obj := cachedObjects.get(oid);
if (obj == null )
  {
     // hook method
     obj = getObjectFromStorage( oid );

    cachedObjects.put( oid, obj )
  }
return obj
}
```

```
// hook method override
protected Object getObjectFromStorage( OID oid )
{
String key = oid.toString();
dbRec = SQL execution result of:
        "Select * from PROD_DESC where key =" + key

ProductDescription pd = new ProductDescription();
pd.setOID( oid );
pd.setPrice(     dbRec.getColumn("PRICE")     );
pd.setItemID(   dbRec.getColumn("ITEM_ID") );
pd.setDescrip( dbRec.getColumn("DESC")      );

return pd;
}
```



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

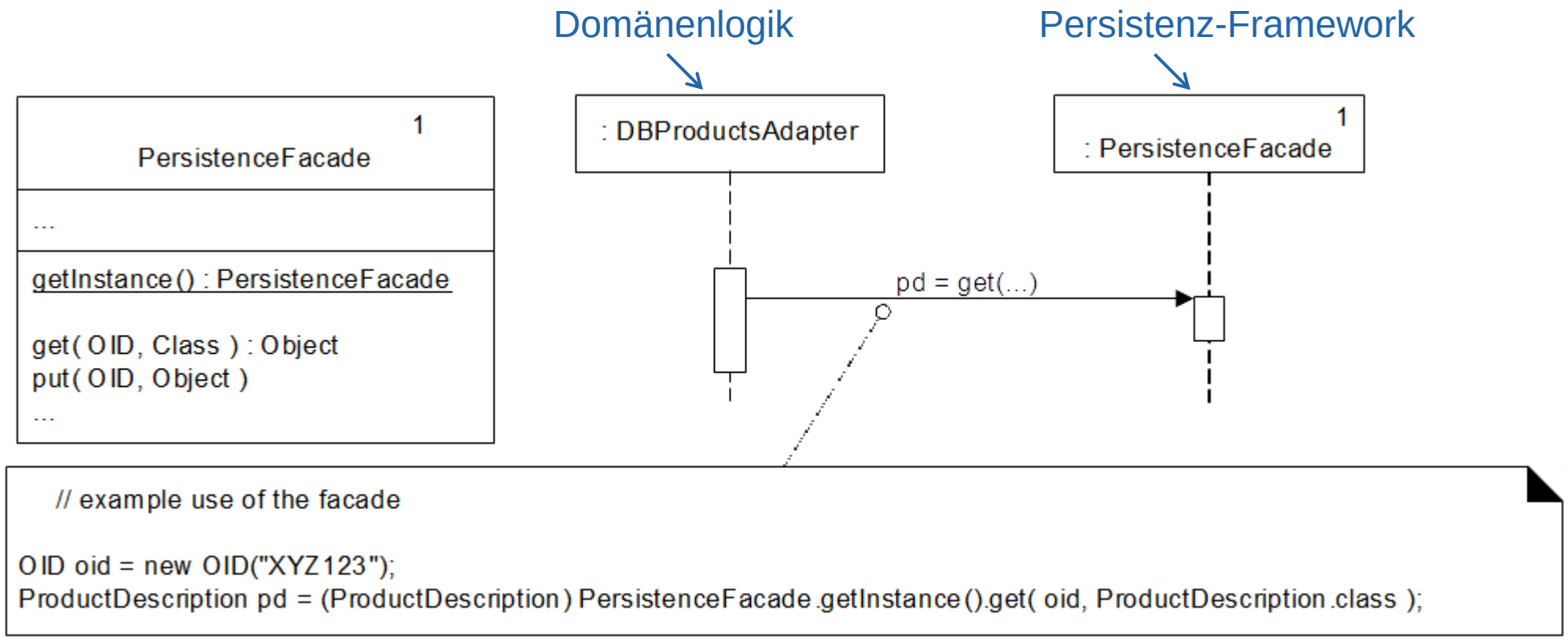
- Framework für Speicherung von Objekten (siehe [1] Kap. 38).
- Primäres Ziel: Prinzipien des Framework Designs zeigen.
- Sekundäres Ziel: Problemstellungen von Persistenz-Frameworks und mögliche Lösungsansätze zeigen.
- Was fehlt?
  - Eigentliche RDB-Zugriffe. Im Buch werden verschiedene Lösungen skizziert.
  - 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

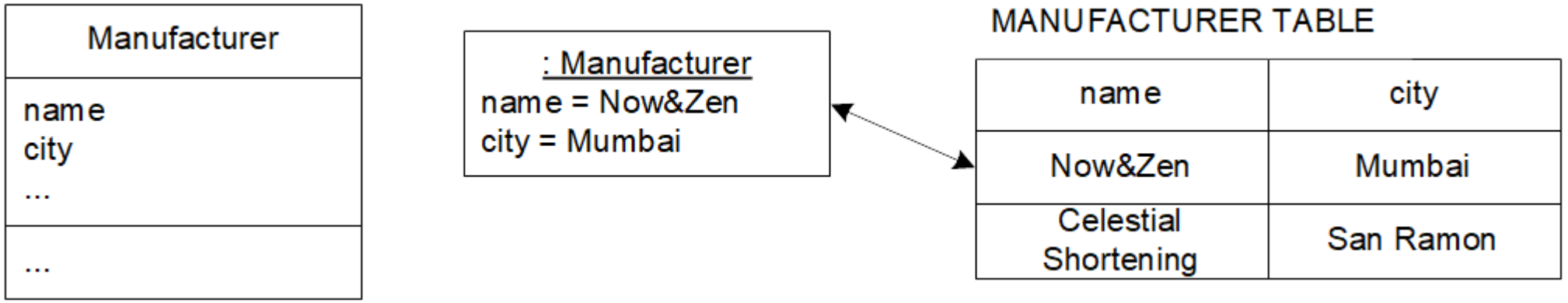
Facade für das Peristenz-Framework

- Facade ist ein Singleton und wird aus der Domänenlogik direkt angesprochen.



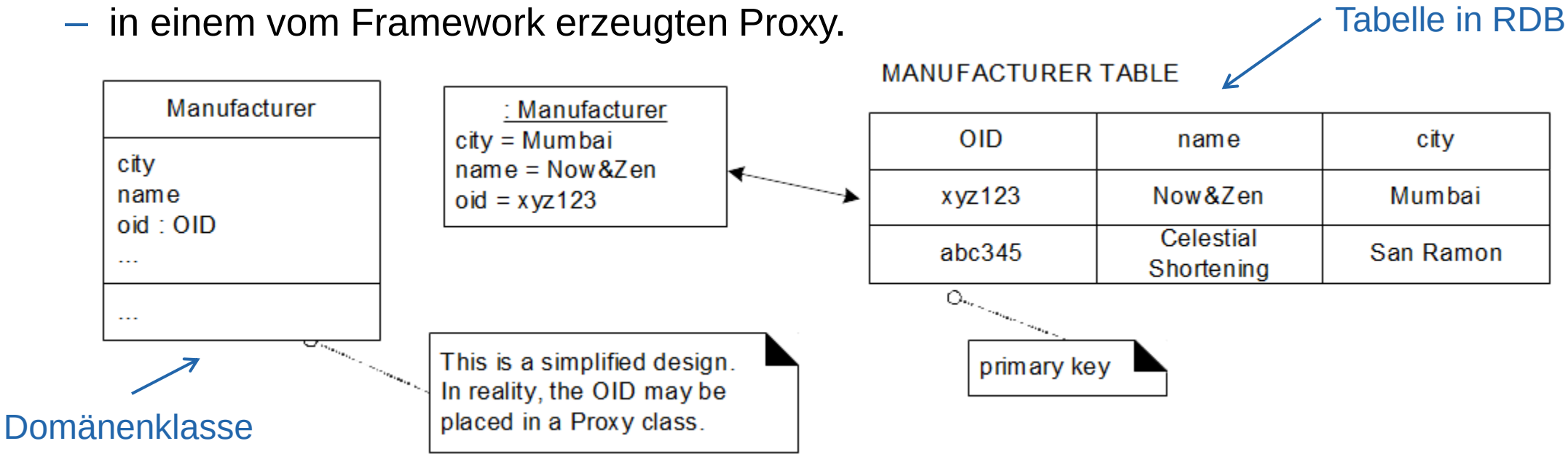
Mapping auf RDB

- Pro Klasse gibt es eine Tabelle.

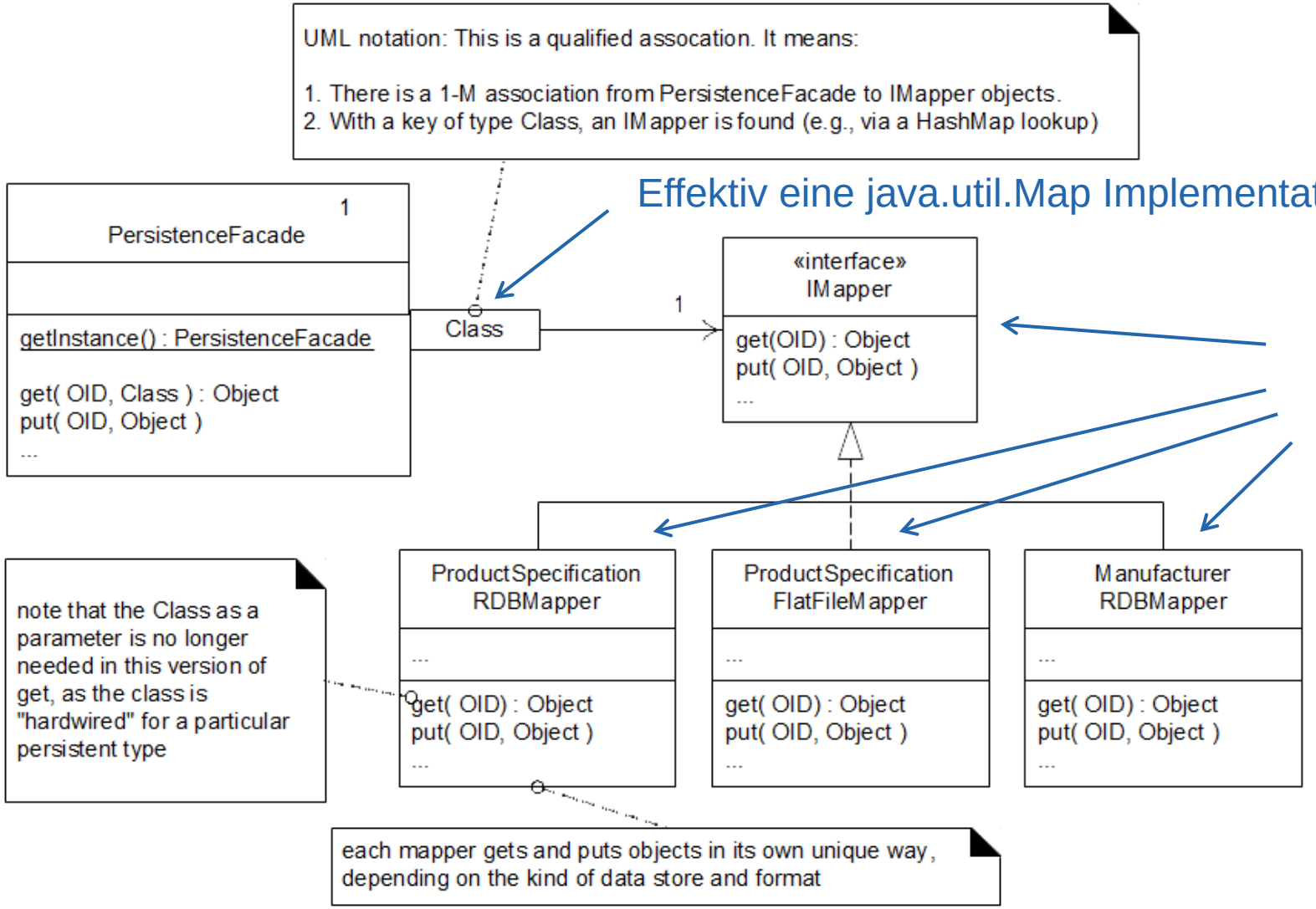


Objekt-Identifikation (OID)

- Um eine eindeutige Beziehung zwischen Objekt und Tabelleneintrag zu haben, muss jedes Objekt eine OID erhalten (entspricht Primärschlüssel im RDB).
- Diese OID kann
  - direkt im Domänenobjekt sein (wie z.B. auch in JPA) .
  - in einem vom Framework erzeugten Proxy.

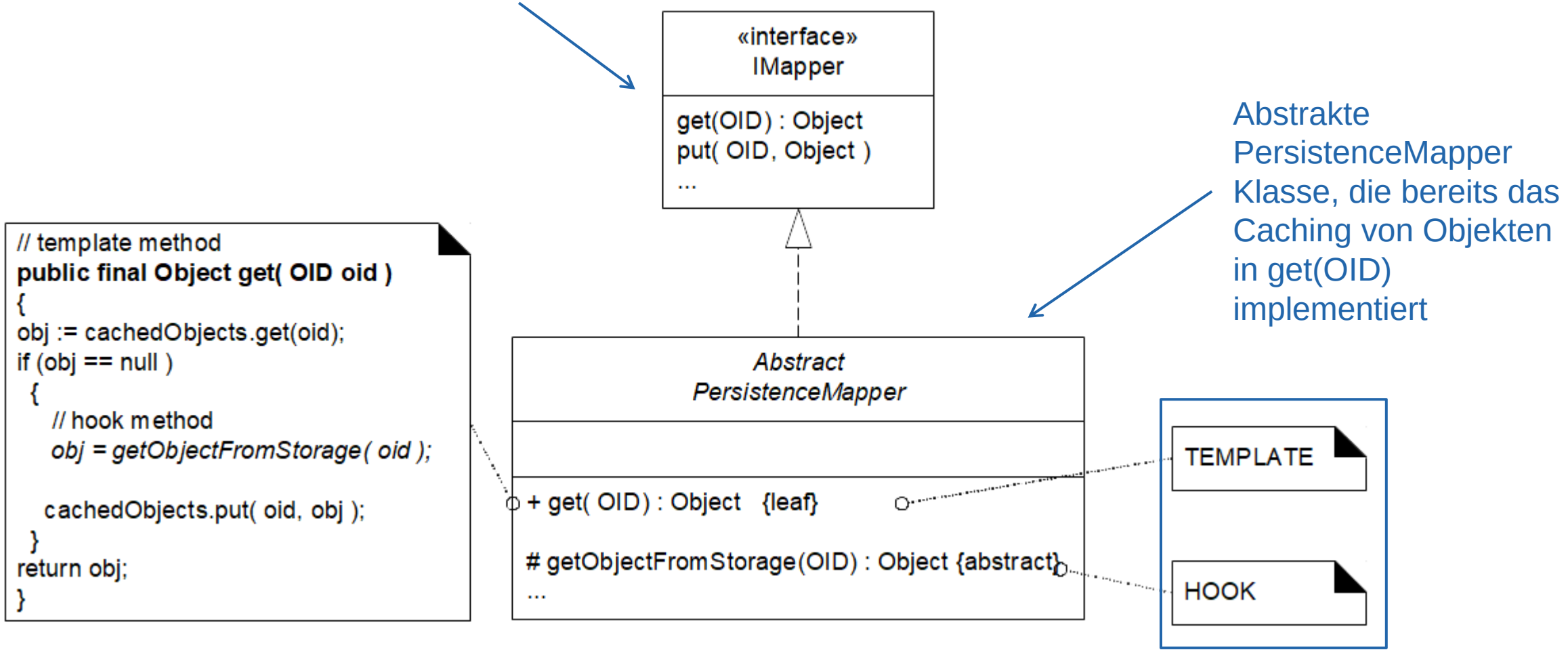


Mapper für jede Domänenklasse



Mapper werden mit Template Method weiterentwickelt

Mapper Interface



Abstrakte PersistenceMapper Klasse, die bereits das Caching von Objekten in get(OID) implementiert

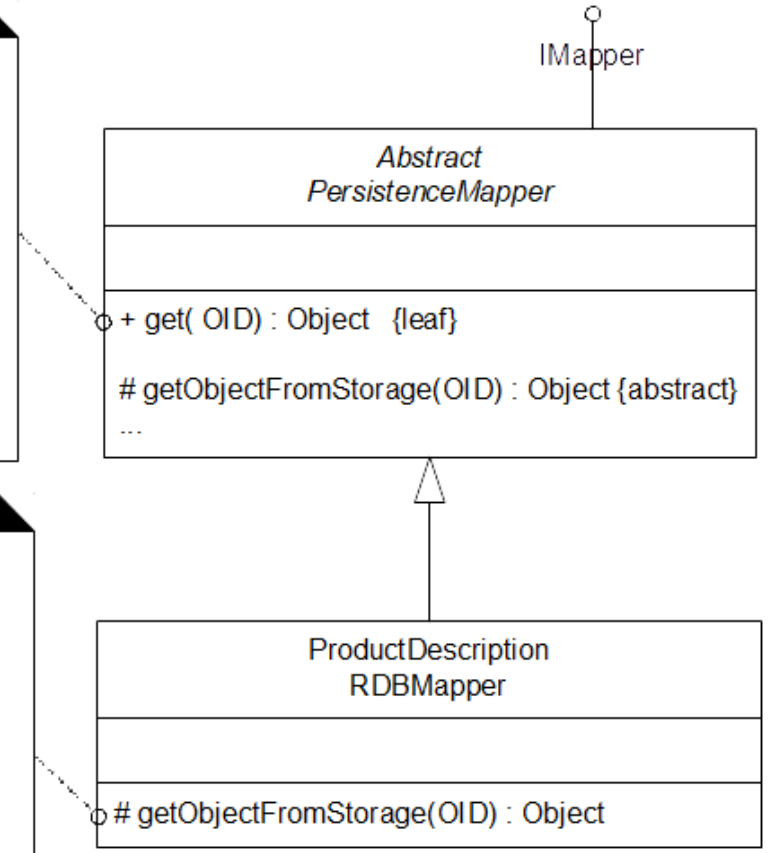
Mapper für Zugriff auf relationale Datenbank

Zugriff auf relationale Datenbank in ProductDescription-RDBMapper

Sehen Sie bereits jetzt Potential für weitere Verallgemeinerungen?

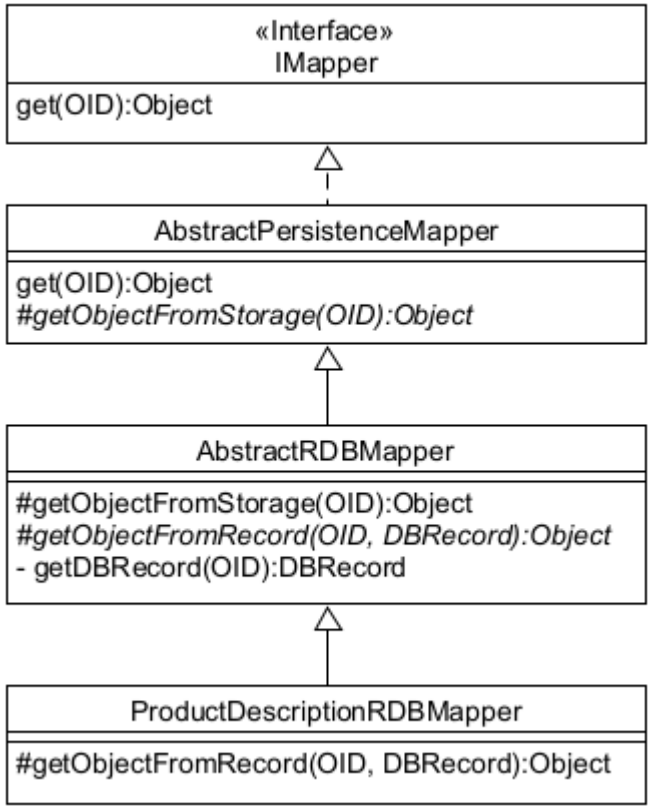
```
// template method
public final Object get( OID oid )
{
obj := cachedObjects.get(oid);
if (obj == null )
  {
    // hook method
    obj = getObjectFromStorage( oid );

   cachedObjects.put( oid, obj )
  }
return obj
}
```

```
// hook method override
protected Object getObjectFrom Storage( OID oid )
{
String key = oid.toString();
dbRec = SQL execution result of:
        "Select * from PROD_DESC where key =" + key

ProductDescription pd = new ProductDescription();
pd.setOID( oid );
pd.setPrice(     dbRec.getColumn("PRICE")     );
pd.setItemID(   dbRec.getColumn("ITEM_ID") );
pd.setDescrip( dbRec.getColumn("DESC")      );

return pd;
}
```



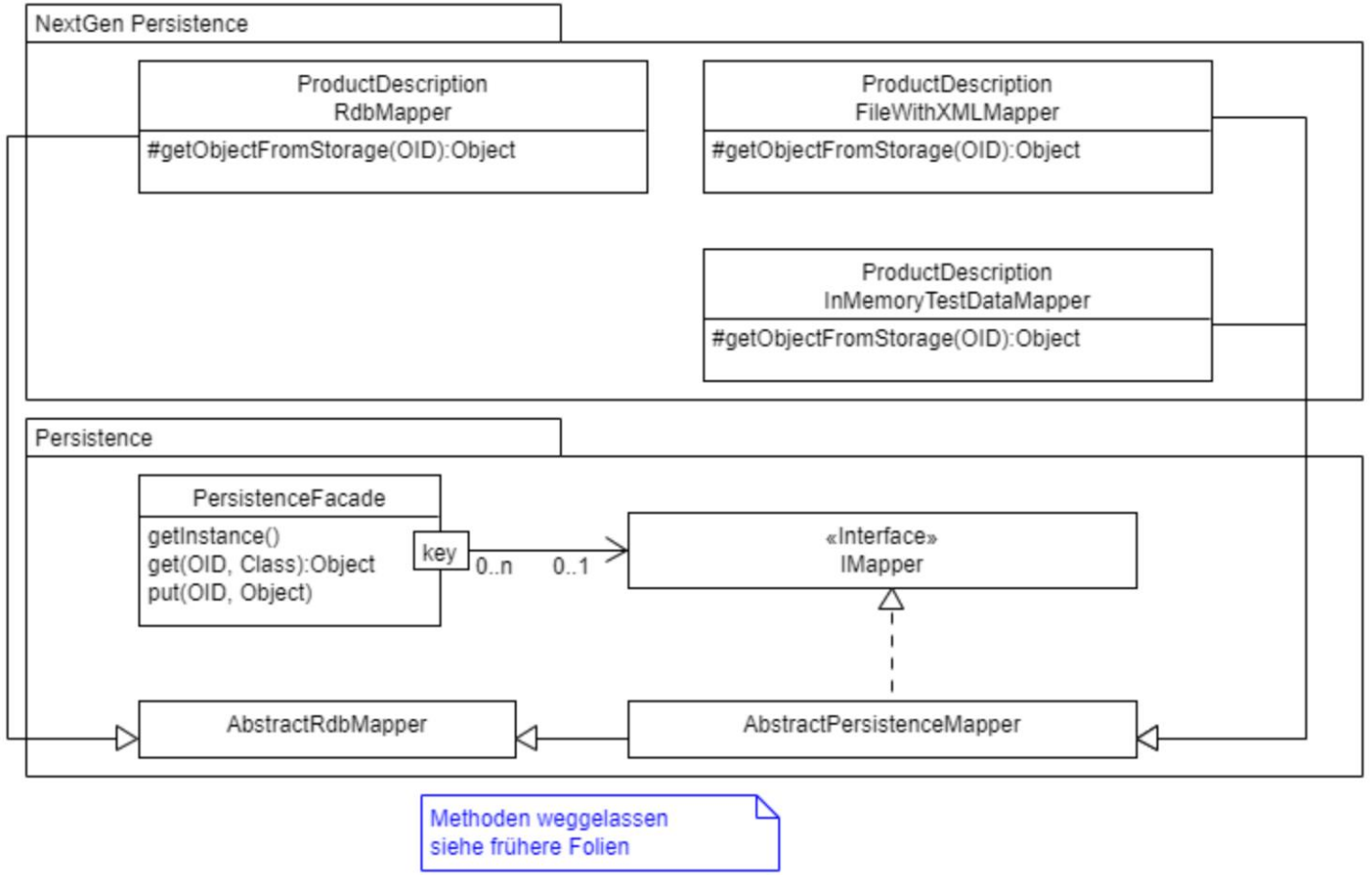
Abstrakte Mapper Klasse für Datenbankzugriff

- Wir ziehen gemeinsamen RDB-Code in eine eigene abstrakte Super-Klasse mit Namen AbstractRDBMapper.
- Die Framework Klassen bilden eine Vererbungshierarchie.
  - Jede Stufe fügt zusätzliche Funktionalität hinzu.
  - Die Hook Methode der Superklasse wird zur Templateklasse in der Subklasse.
  - Am Schluss wird vom Anwendungsentwickler noch der Anwendungsspezifische Code hinzugefügt (ProductDescriptionRDBMapper).



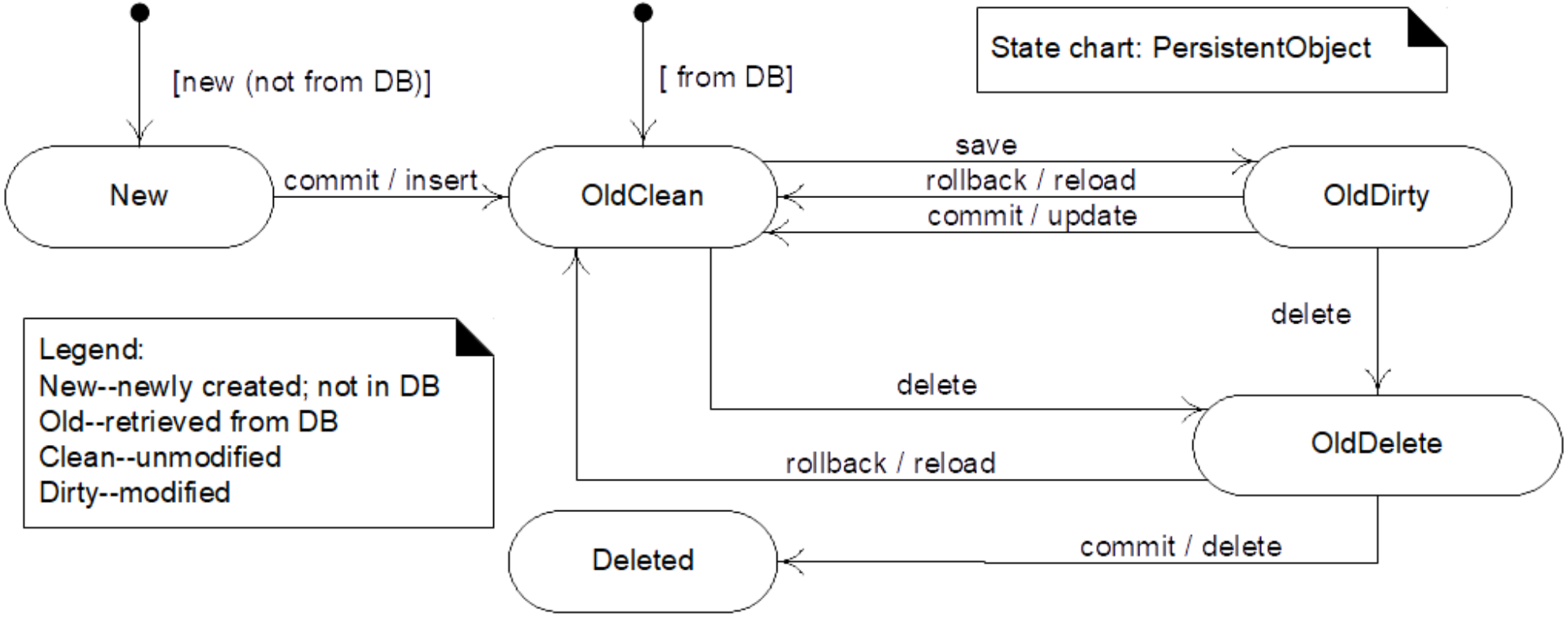
Mapper Klassen Ankoppelung an Domänenlogik

- Wir führen nun ein eigenes Package für die anwendungsspezifischen Mapper Klassen ein.
- Diese erben von den Framework-Klassen.
  - RDB: AbstractRdbMapper.
  - Einfache Dateien: AbstractPersistenceMapper.



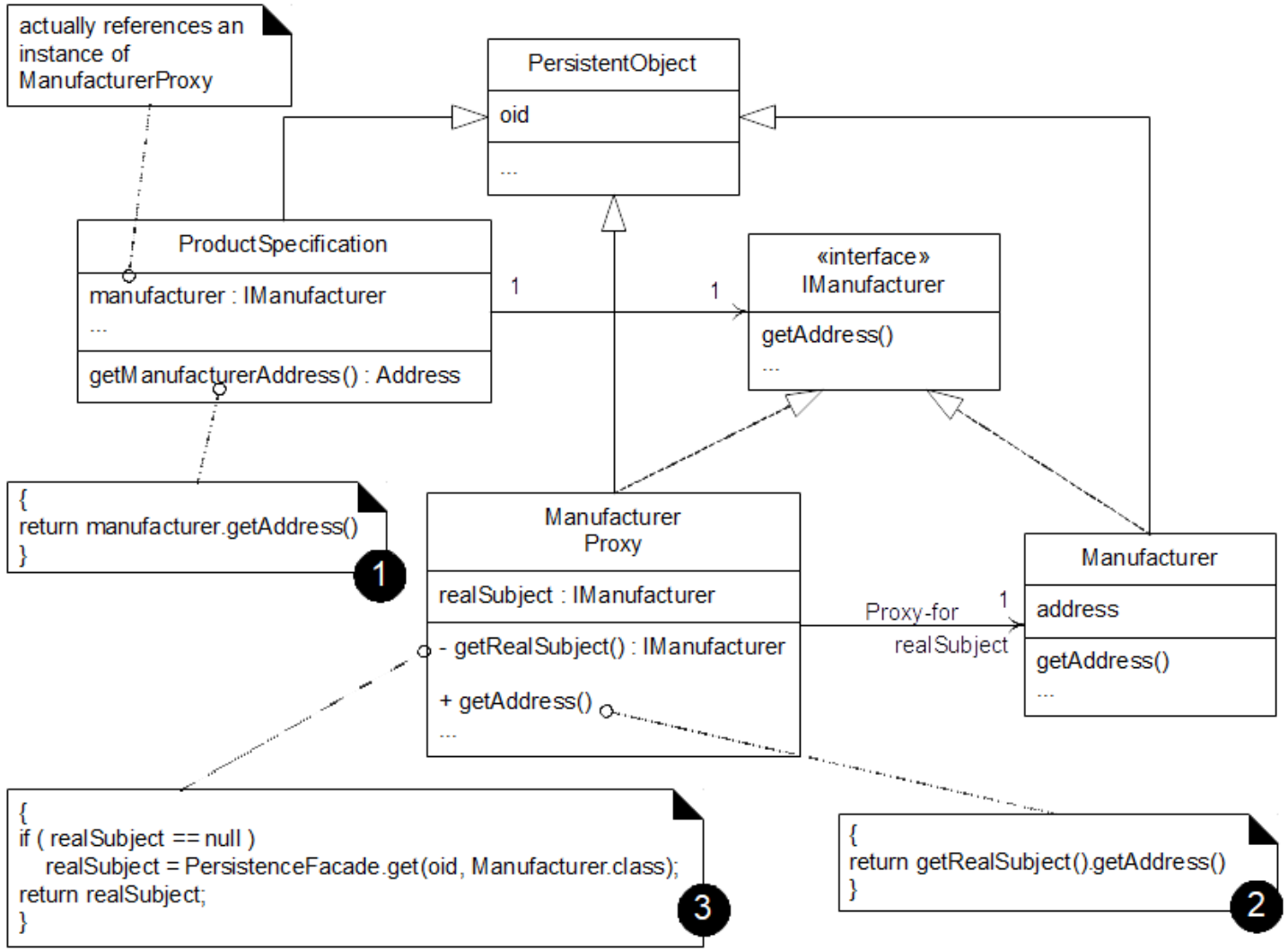
Zustandsverwaltung bezüglich Transaktionen

- Jedes Objekt befindet sich gegenüber der aktuellen Transaktion in einem bestimmten Zustand. Umsetzung erfolgt dann mit dem State Pattern.



Proxy für Lazy Loading von referenzierten Objekten

- Anwendung des Virtual Proxy Pattern.
- Framework erzeugt nicht die richtige Klasse, sondern nur ein Proxy.



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