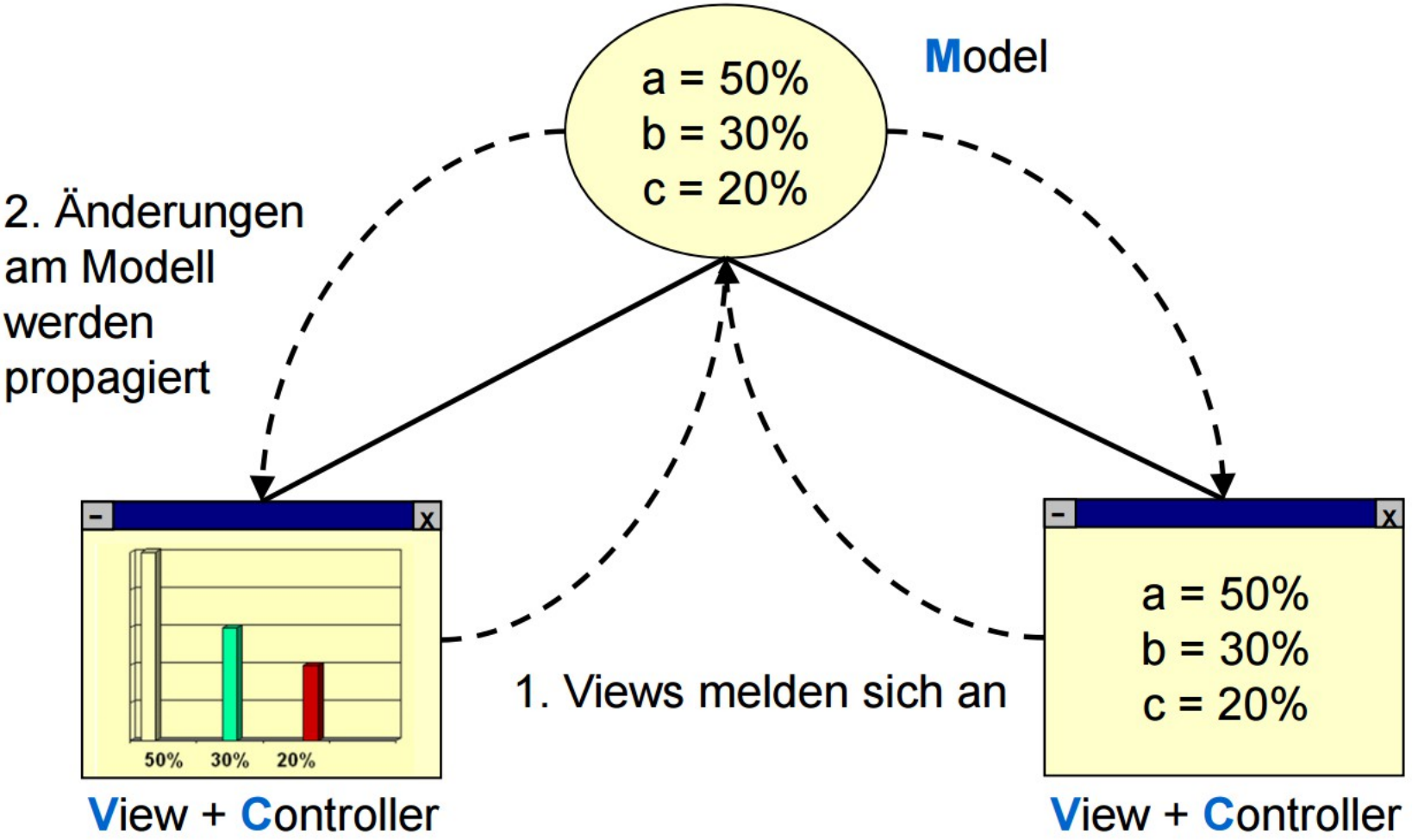
Entwurfsmuster I

Ziele

- Dokumentation von Lösungen wiederkehrender Probleme, um Programmierer bei der Softwareentwicklung zu unterstützen.
- Schaffung einer gemeinsamen Sprache, um über Probleme und ihre Lösungen zu sprechen.
- Bereitstellung eines standardisierten Katalogisierungsschemas um erfolgreiche Lösungen aufzuzeichnen

Das MVC


a = 50%
b = 30%
c = 20%
Model
2. Änderungen am Modell werden propagiert
50%
30%
20%
1. Views melden sich an
a = 50%
b = 30%
c = 20%
View + Controller
View + Controller

Das MVC

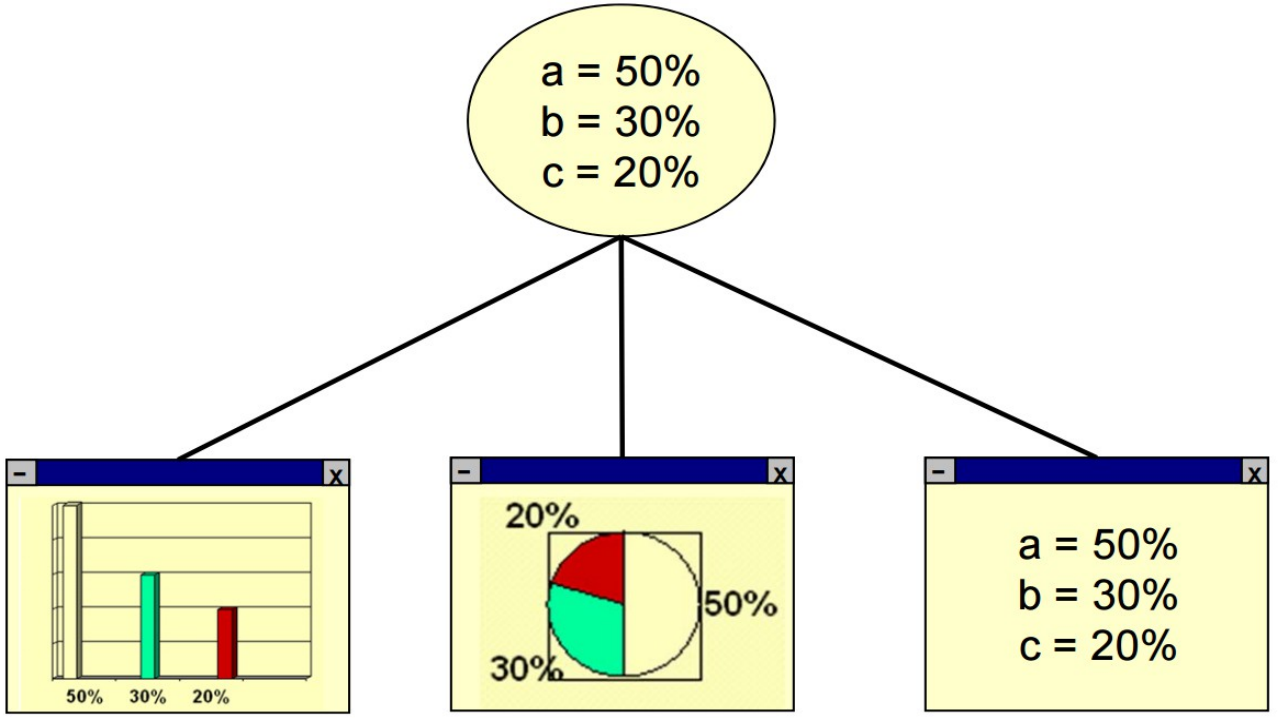
- Propagierung von Änderungen: Observer Pattern
  - Kommt z.B. auch bei Client/Server-Programmierung zur Benachrichtigung der Clients zum Einsatz
- Geschachtelte Views: Composite Pattern
  - View enthält weitere Views, wird aber wie ein einziger View behandelt.
- Reaktion auf Events im Controller: Strategy Pattern
  - Eingabedaten können validiert werden
  - Controller können zur Laufzeit gewechselt werden

Observer

- Stellt eine 1-zu-n Beziehung zwischen Objekten her
- Wenn das eine Objekt seinen Zustand ändert, werden die davon abhängigen Objekte benachrichtigt und entsprechend aktualisiert
- Verschiedene Objekte sollen zueinander konsistent gehalten werden
- Andererseits sollen sie dennoch nicht eng miteinander gekoppelt sein

Observer

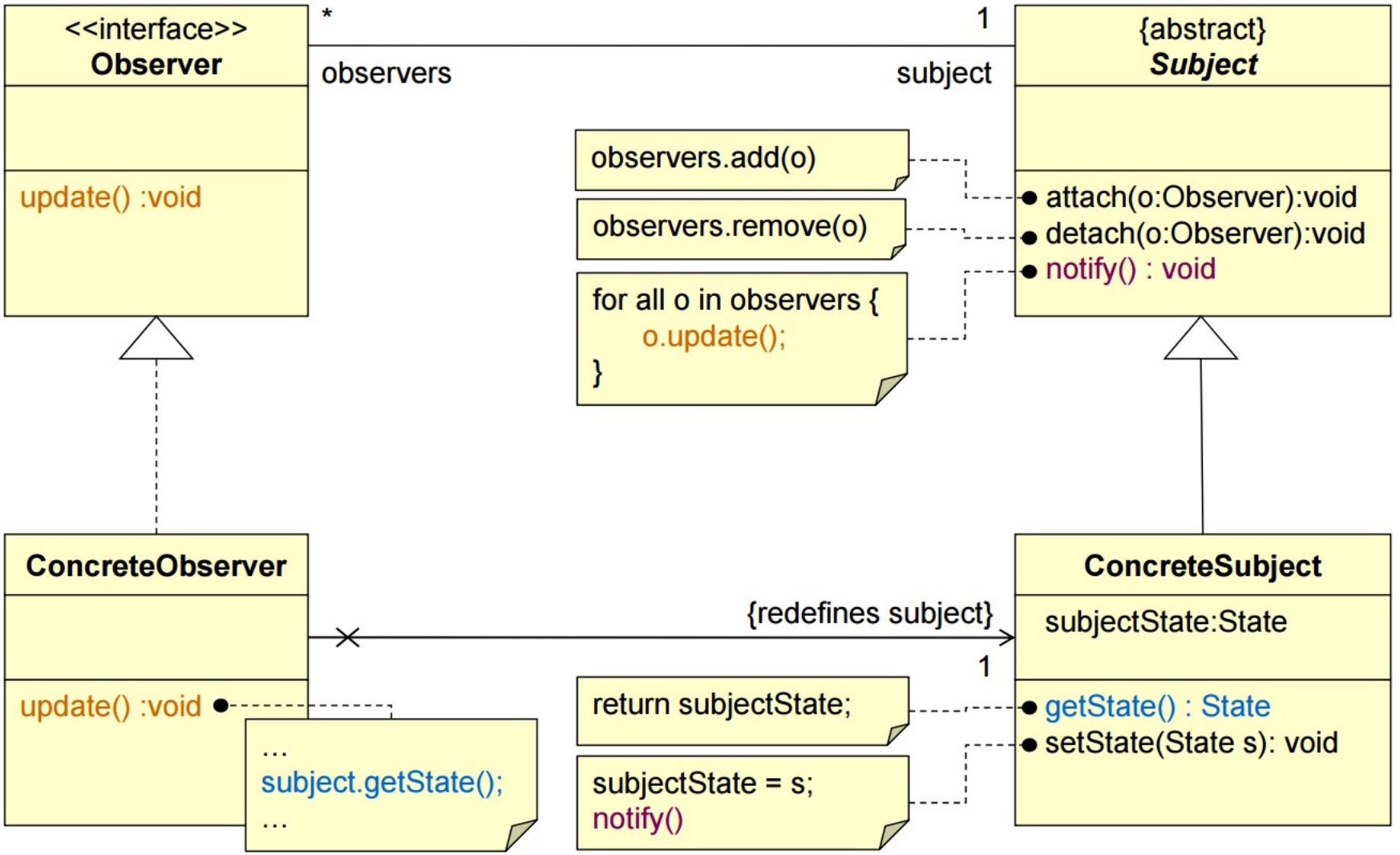
- Wenn in einer Sicht Änderungen vorgenommen werden, werden alle anderen Sichten aktualisiert – Sichten sind aber unabhängig voneinander



Observer Kontext

- Abhängigkeiten
  - Ein Aspekt einer Abstraktion ist abhängig von einem anderen Aspekt
- Folgeänderungen
  - Änderungen an einem Objekt erfordert Änderungen an anderen Objekten
  - Es ist nicht bekannt, wie viele Objekte geändert werden müssen
- Lose Kopplung
  - Objekte sollen andere Objekte benachrichtigen können, ohne Annahmen über die Beschaffenheit dieser Objekte machen zu müssen

Observer - Pull



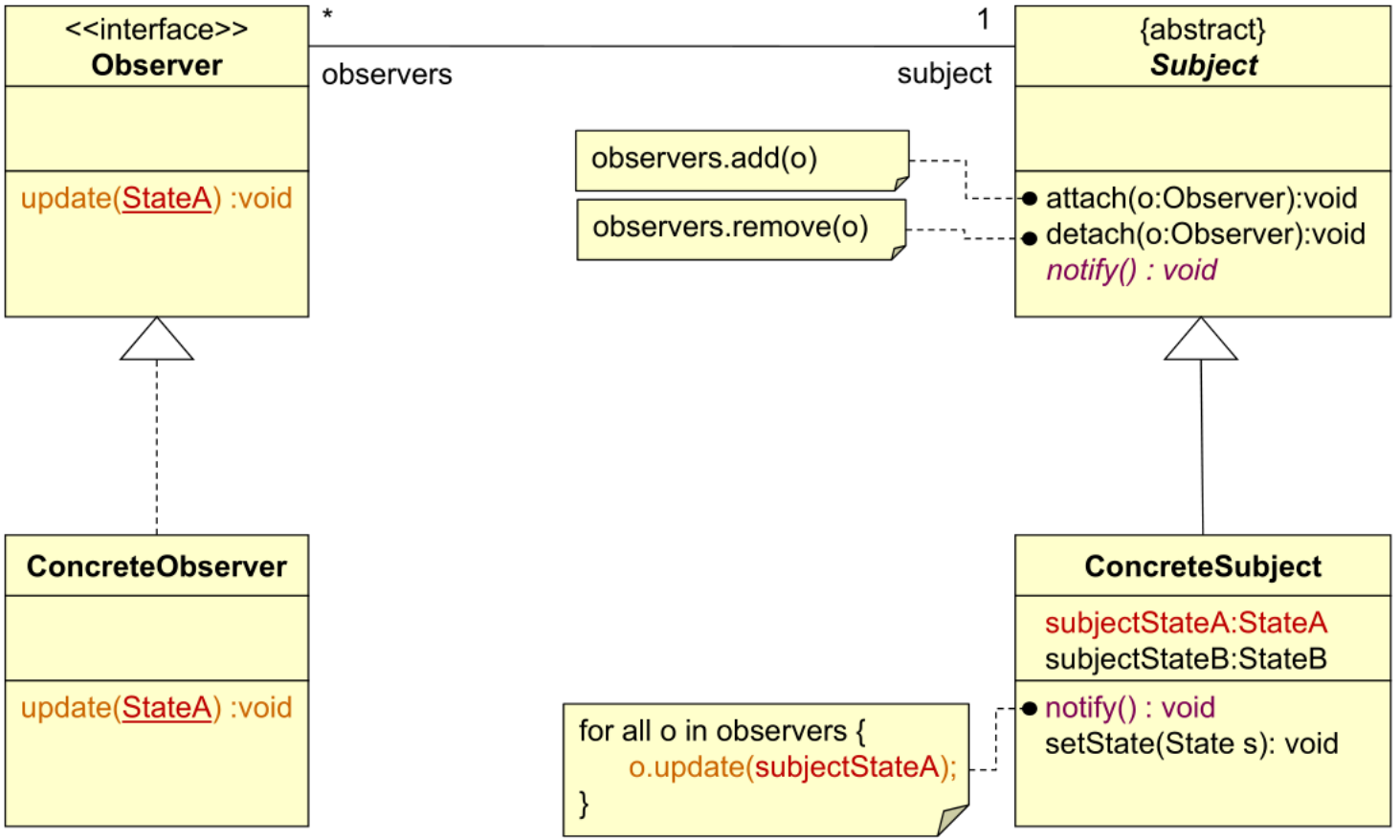
Modell

- Observer
  - update() -- auch: handleEvent
- Subject
  - attach(Observer o)
  - detach(Observer o)
  - notify()
  - setState(…)
  - getState()

Implementierung

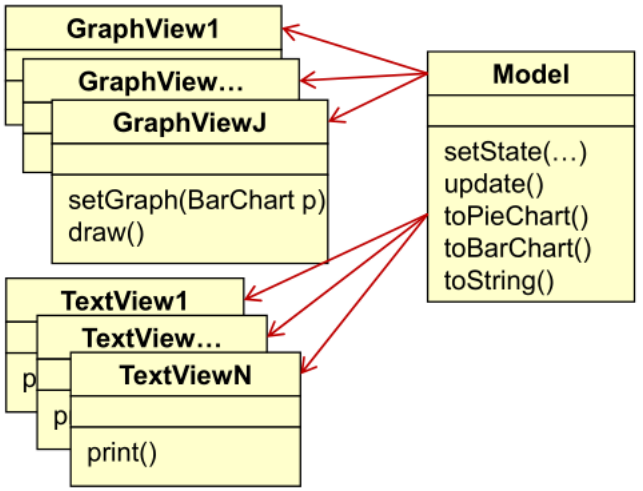
- „push“ versus „pull“
- Pull: Subjekt übergibt in „update()“ keinerlei Informationen, aber die Beobachter müssen sich die Informationen vom Subjekt holen
  - Berechnungen werden häufiger durchgeführt
- Push: Subjekt übergibt in Parametern von „update()“ detaillierte Informationen über Änderungen
  - Beobachter sind weniger wiederverwendbar (Abhängig von den Parametertypen)

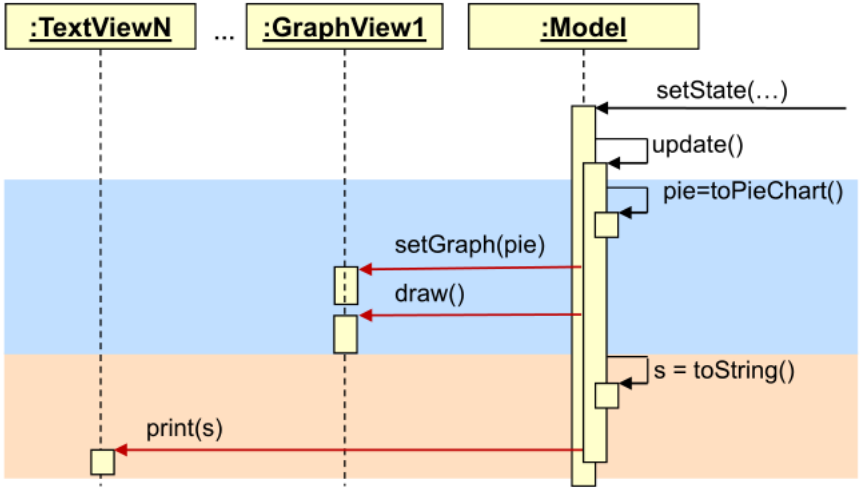
Oberver - Push



Abhängigkeiten ohne Observer

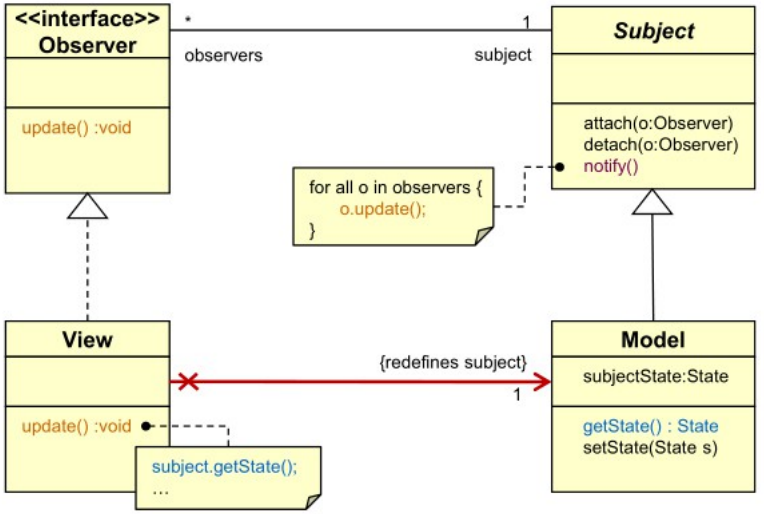
- myGraphView1.setGraph(this.toPieChart());
  myGraphView1.draw();
- myGraphView2.setGraph(this.toBarChart());
  myGraphView2.draw();
- ...

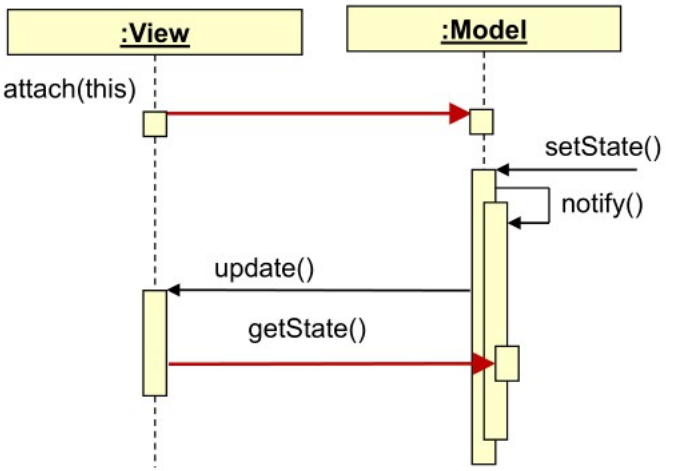




Abhängigkeiten mit Observer

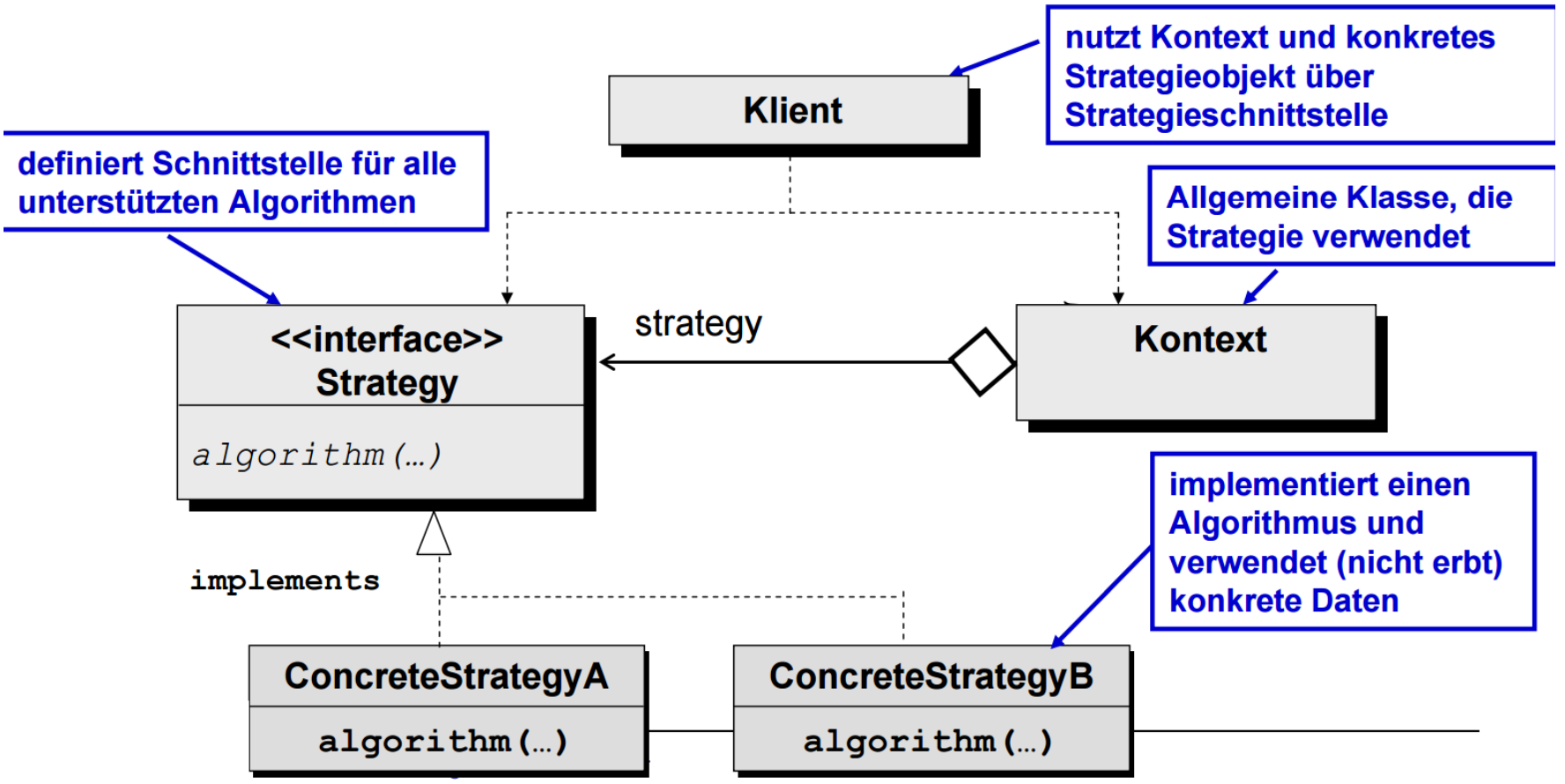
- model.getState1();
- model.getState2();





Strategy

- Das Strategiemuster entkoppelt Objekte von ihrem Verhalten und unterstützt den Austausch von Algorithmen



Vorteile

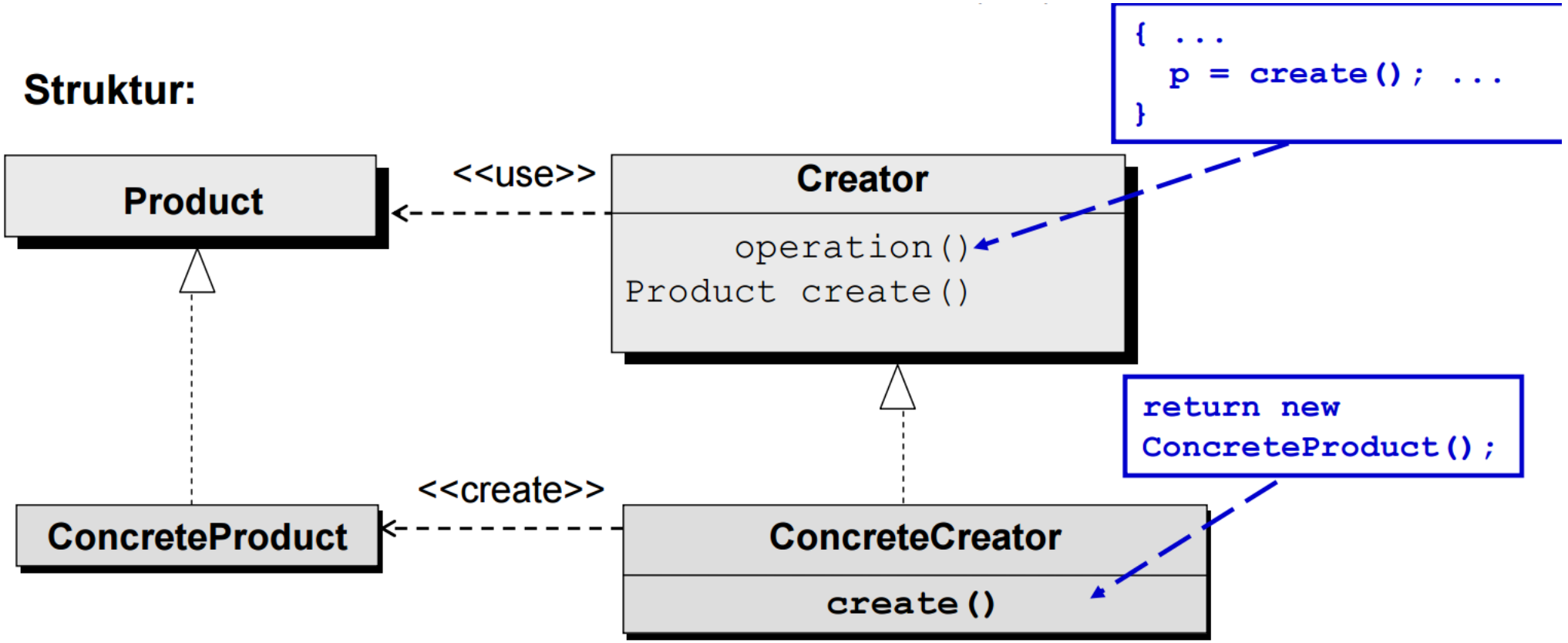
- Es wird eine Familie von Algorithmen definiert
- Strategien bieten eine Alternative zur Unterklassenbildung, helfen Mehrfachverzweigungen zu vermeiden und verbessern dadurch die Wiederverwendung
- Strategien ermöglichen die Auswahl aus verschiedenen AlgorithmenImplementationen (“Algorithmen-Polymorphie”) und erhöhen dadurch die Flexibilität

Nachteile

- Klienten müssen die unterschiedlichen Strategien kennen, um zwischen ihnen auswählen zu können
- Gegenüber der direkten Implementation der Algorithmen im Klienten erzeugen Strategien zusätzlichen Kommunikationsaufwand zwischen Strategie und Klient
- Die Anzahl der Objekte wird erhöht

Factory Method

Struktur:



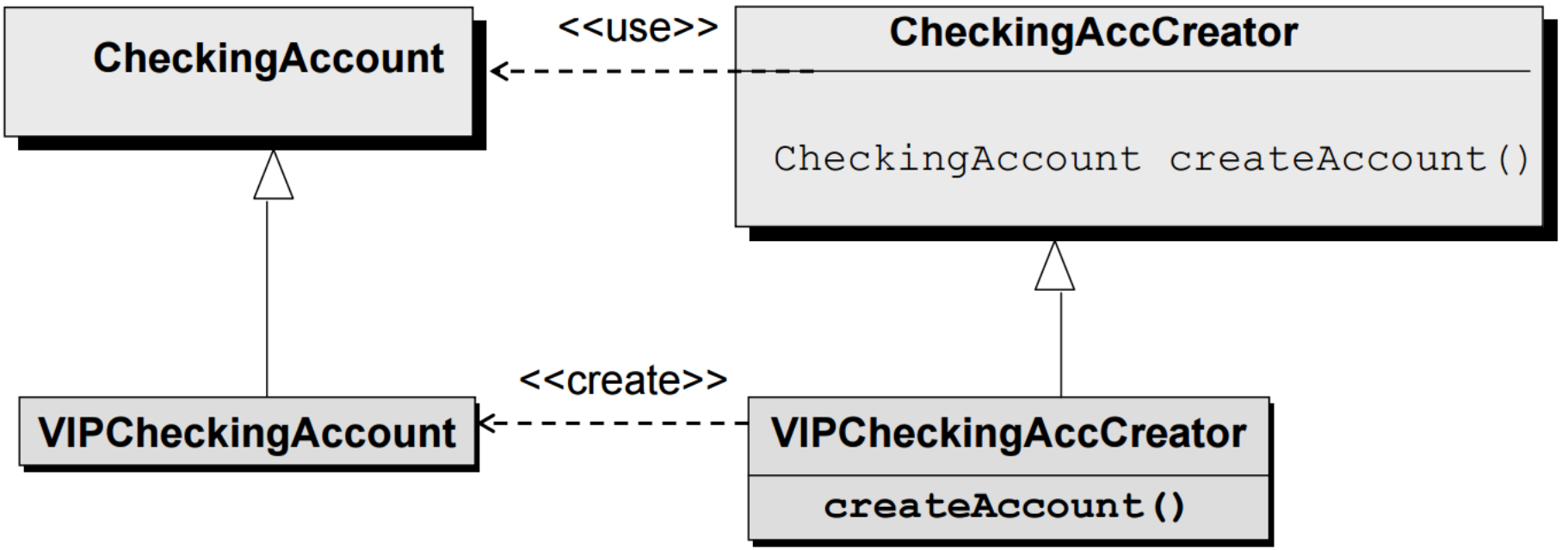
Factory Method

- Fabrikmethodenmuster definiert eine Schnittstelle zur Erzeugung eines Objektes, wobei es den Unterklassen überlassen bleibt, von welcher Klasse das zu erzeugende Objekt ist
- Beteiligte Klassen
  - Product
  - Creator
  - ConcreteProduct
  - ConcreteCreator

Factory Method

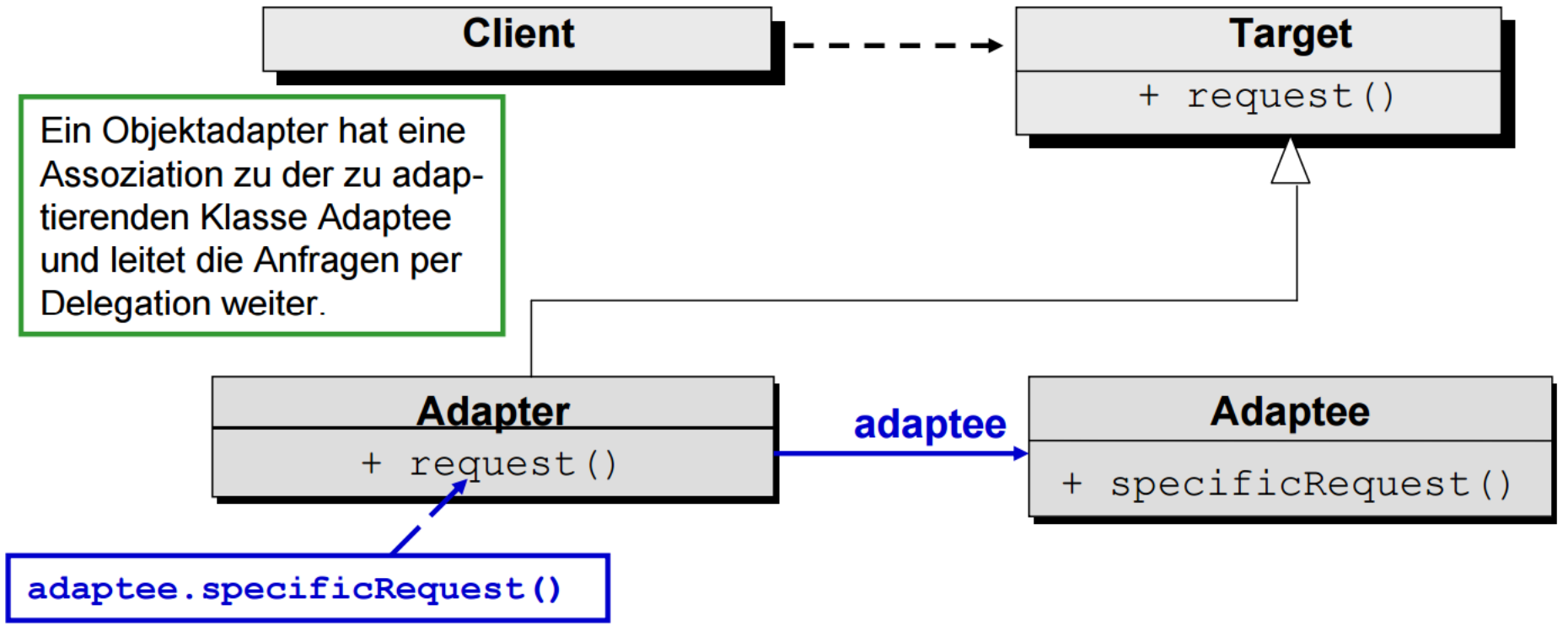
- Die abstrakte Methode create() sehr spezifisch: Sie erzeugt ein Objekt
- Das „Wie“ der Objekterzeugung ist hinter create() versteckt
- Fabrikmethoden entkoppeln ihre Aufrufer von Implementierungen konkreter Produktklassen
- Die Verwendung dieses Erzeugungsmusters läuft auf Unterklassenbildung hinaus

Fabrikmethode für Konten



Adapter

- übersetzt eine Schnittstelle in eine andere



Adapter

```
interface ITarget
{
  List<string> GetProducts();
}


public class VendorAdaptee
{
   public List<string> GetListOfProducts()
   {
      List<string> products = new List<string>();
      products.Add("Gaming Consoles");
      products.Add("Television");
      products.Add("Books");
      products.Add("Musical Instruments");
      return products;
   }
}


class VendorAdapter:ITarget
{
   public List<string> GetProducts()
   {
      VendorAdaptee adaptee = new VendorAdaptee();
      return adaptee.GetListOfProducts();
   }
}


class ShoppingPortalClient
{
   static void Main(string[] args)
   {
      ITarget adapter = new  VendorAdapter();
      foreach (string product in adapter.GetProducts())
      {
        Console.WriteLine(product);
      }
      Console.ReadLine();
   }
}
```

Vorteile

- Ein Klassenadapter passt genau eine Dienstklasse (Adaptee) an
- Ein Klassenadapter kann dadurch das Verhalten des Adaptees überschreiben
- Ein Klassenadapter wird eingesetzt, wenn ein Teil einer ganz konkreten Schnittstelle variiert werden soll
- Ein Objektadapter kann mehrere Dienstklassen (Adaptees) anpassen und alle Adaptees auf einmal um zusätzliche Operationen erweitern

Nachteile

- Für Objektadapter ist es schwieriger (als für Klassenadapter), das Verhalten des Adaptees zu überschreiben
- Klassenadapter lassen sich nur realisieren, wenn die Programmiersprache Schnittstellen oder Mehrfachvererbung unterstützt

Zusammenfassung

- Entwurfsmuster sind Regeln oder Richtlinien, um häufig auftretende Probleme bei der Erstellung eines Programms zu lösen
- Das Fabrikmethodenmuster (Factory Method) definiert eine Schnittstelle zur Erzeugung eines Objektes, wobei es den Unterklassen überlassen bleibt, von welcher Klasse das zu erzeugende Objekt ist

Zusammenfassung

- Das Adaptermuster (Adapter, Wrapper) übersetzt eine Schnittstelle in eine andere. Dadurch können Klassen miteinander kommunizieren, die zueinander inkompatible Schnittstellen zur Verfügung stellen
- Das Strategiemuster (Strategy) entkoppelt Objekte von ihrem Verhalten und unterstützt den Austausch von Algorithmen
- Das Beobachtermuster (Observer) ermöglicht die Weitergabe von Änderungen eines Objekts an abhängige Objekte