Technische Hochschule Ingolstadt

Fakultät für Elektrotechnik und Informatik

Zukunft in Bewegung

Entwurfsmuster

Software Engineering

Prof. Dr. Bernd Hafenrichter

Motivation

- Der Erzeugungsprozess von Objekten ist essentiel für die Erweiterbarkeit und die Flexibilität eines Softwaresystems.
- Werden Objekte direkt über den Operator "new" erzeugt so spricht man an dieser Stelle von einer engen Kopplung da das erzeugende Programm und das erzeugte Programm bereits im Quelltext zur Übersetzungszeit festgelegt ist
- Möchte man solch ein System erweitern so müssen alle Stellen im Programm geändert werden an denen Objekte direkt erzeugt werden

```
Public void testMethod() {
   Klasse klasse = new Klasse();
}
```

Enge Kopplung

Motivation

- Frage: Wie kann die Objekterzeugung modifiziert werden so dass die hohe Abhängigkeit reduziert wird??

- Erzeugungsmuster versuchen diese Abhängigkeit zu reduzieren und dadurch den Programmaufbau flexibler zu gestalten.

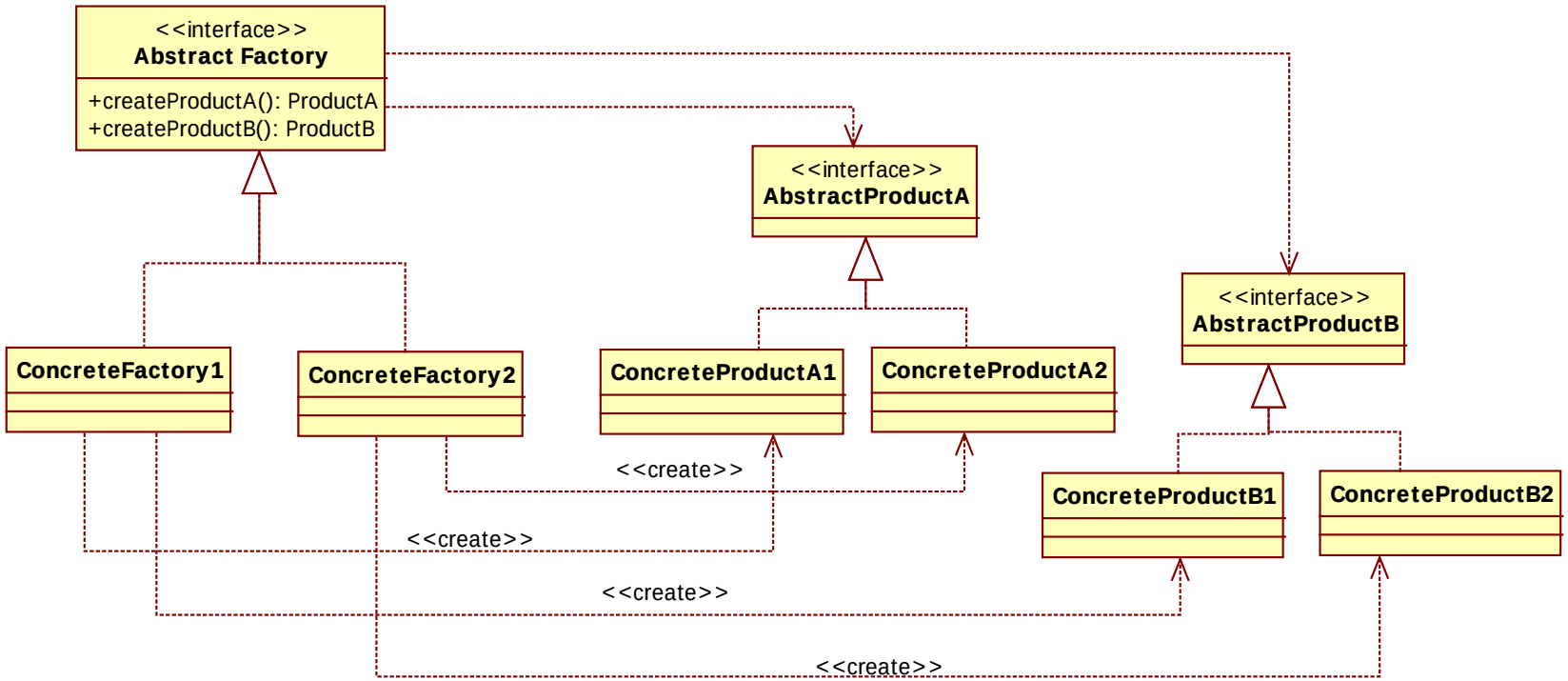
Abstrakte Fabrik

- **Zweck**:
  - Biete eine Schnittstelle zum Erzeugen von Familien verwandter oder voneinander abhängiger Objekte
  - Benenne keine konkreten Klassen

- **Beispiel:**
  - GUI Framework mit Themes
  - Je nach eingestellter Theme müssen andere Widgets (z.B. Button, Window, Scrollbar) erzeugt werden.

Abstrakte Fabrik

- **Lösung:**



Abstrakte Fabrik (Akteure)

AbstractFactory:

- deklarierte eine abstrakte Schnittstele für Operationen, die konkrete Produktobjekte erzeugen

ConcreteFactory:

- implementiert die Operationen zur Erzeugung konkreter Produktobjekte

AbstractProduct:

- definiert eine Scnittstelle für einen bestimmten Typ von Produktobjekten

ConcreteProduct :

- definiert ein von der entsprechenden KonkrtenFabrik zu erzeugendes Produktobjekte , implementiert die AbstraktesProdukt-Schnittstelle

Abstrakte Fabrik (Bewertung)

Vorteile:

- Konkrete Klassen werden isoliert.
- Der Austausch von Produktfamilien ist auf einfache Art und Weise möglich

Nachteil:

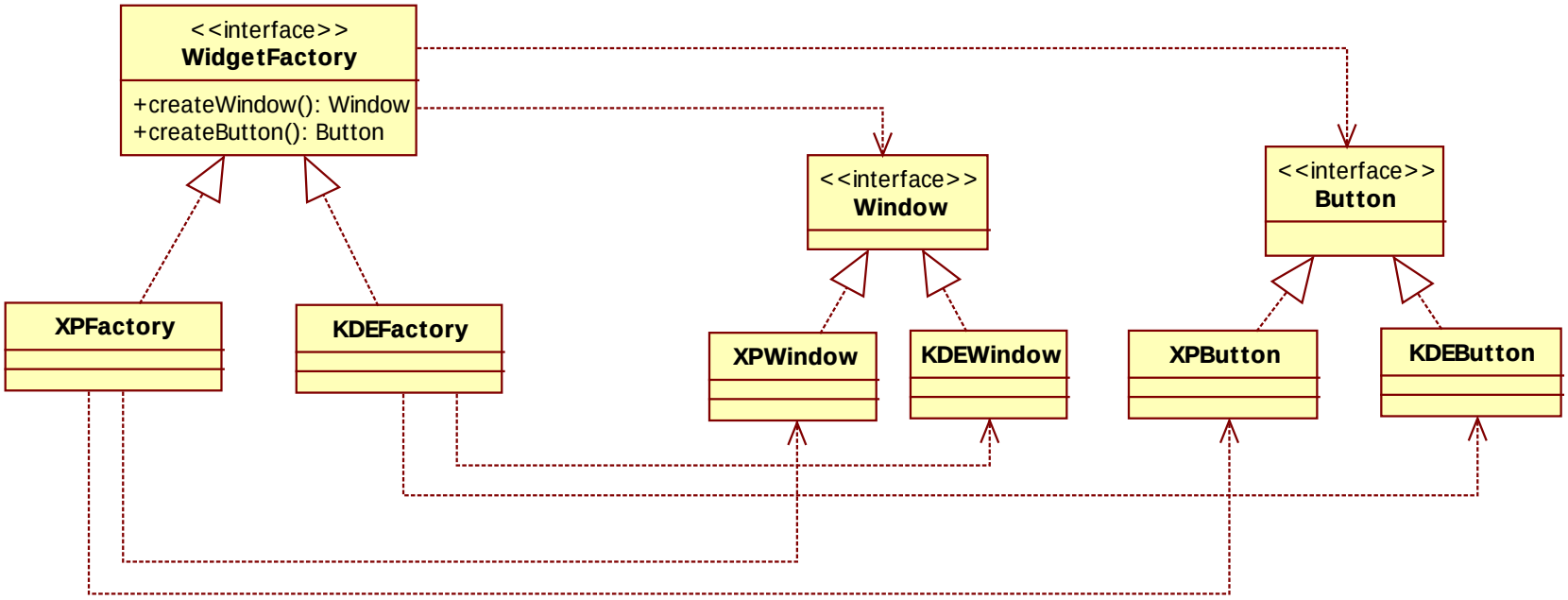
- Neue Produktarten lassen sich schwer hinzufügen, da in allen konkreten Fabriken Änderungen vorzunehmen sin

Abstrakte Fabrik

- **Verwendung (allgemein):**
  - ein System unabhängig von der Art der Erzeugung seiner Produkte arbeiten soll
  - ein System mit einer oder mehreren Produktfamilien konfiguriert werden soll
  - eine Gruppe von Produkten erzeugt und gemeinsam genutzt werden soll oder
  - wenn in einer Klassenbibliothek die Schnittstellen von Produkten ohne deren Implementierung bereitgestellt werden sollen

Abstrakte Fabrik

- **Beispiel: Graphische Oberfläche mit pluggable Look & Feel**

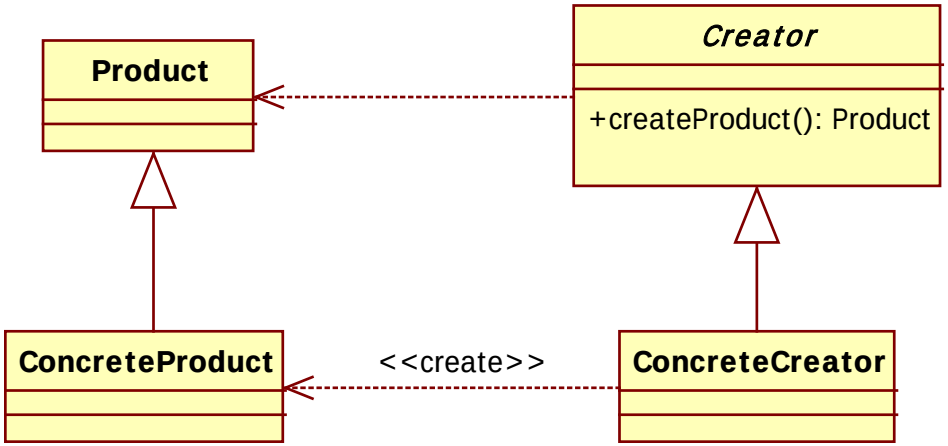


Fabrikmethode

- **Zweck**:
  - Definiere eine Klassenschnittstelle mit Operationen zum Erzeugen eines Objektes, aber lasse Unterklassen entscheiden, von welcher Klasse das zu erzeugende Objekt ist.
  - Fabrikmehtoden ermöglichen es einer Klasse, die Erzeugung von Objekten an Unterklassen zu delegieren
- **Beispiel:**
  - Verwendung eines Frameworks für eine Anwendung, die mehrere unterschiedliche Dokumente gleichzeitig anzeigen kann.

Fabrikmethode

- **Lösung:**



Fabrikmethode (Akteure)

Product:

- definiert die Klasse des von der Fabrikmethode erzeugten Objekts

ConcredeProduct:

- implementiert die Produktschnitstelle

Creator:

- deklariert die Fabrikmethode, die ein Objekt des Typs Produkt zurückgiebt. Der Erzeuger kann möglicherweise eine Defaultimplementierung der Fabrikmethode definieren, die ein vordefiniertes KonkretesProdukt-Objekt liefert

ConcretCreator:

- überschreibt die Fabrikmethode, so dass sie ein Exemplar von KonkretesProdukt zurückgibt

Fabrikmhode

Bewertung:

Vorteile:

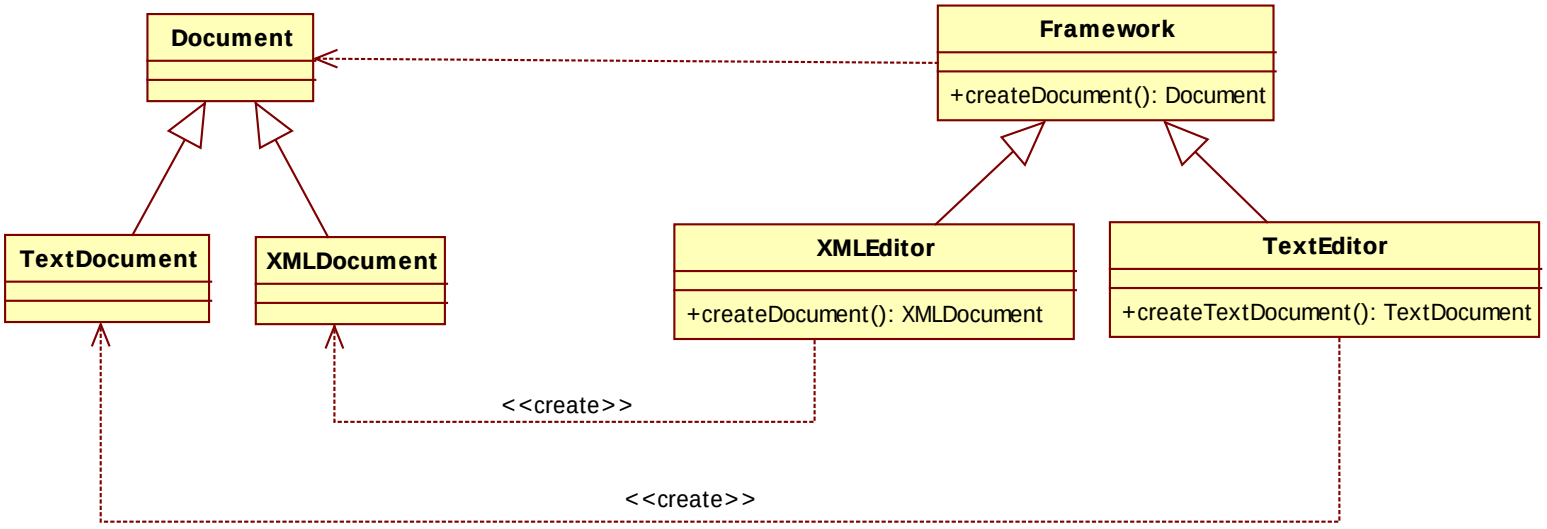
- Fabrikmethoden entkoppeln ihre Aufrufer von Implementierungen konkreter Produkt-Klassen.
- Das ist insbesondere wertvoll, wenn Frameworks sich während der Lebenszeit einer Applikation weiterentwickeln - so können zu einem späteren Zeitpunkt Instanzen anderer Klassen erzeugt werden, ohne dass sich die Applikation ändern muss.

•**Nachteil**:

- Die Verwendung dieses Erzeugungsmusters läuft auf Unterklassenbildung hinaus. Es muss eine eigene Klasse vorhanden sein, die die Klassen-Methode zur Erzeugung aufnehmen kan

Fabrikmethode

- **Beispiel: Framework zur Bearbeitung beliebiger Dokumente**



Fabrikmethode

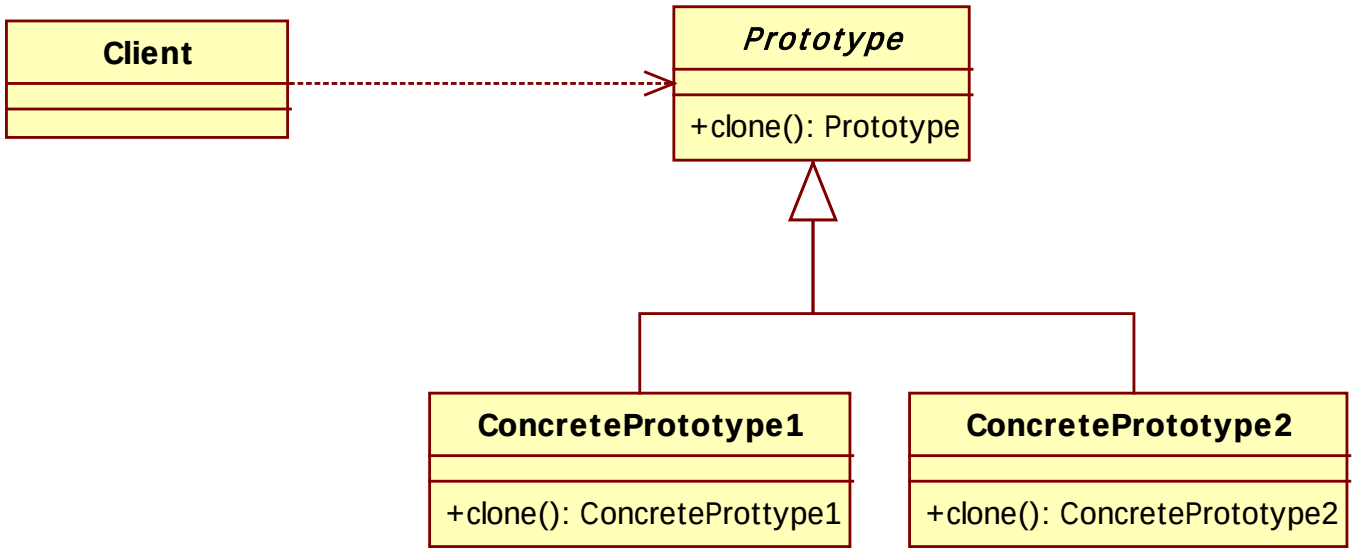
- **Verwendung (allgemein):**
  - Die Fabrikmethode (in der GoF-Bedeutung) findet Anwendung, wenn eine Klasse die von ihr zu erzeugenden Objekte nicht kennen kann bzw. soll,
  - wenn Unterklassen bestimmen sollen, welche Objekte erzeugt werden.
  - Typische Anwendungsfälle sind Frameworks und Klassenbibliotheken.

Prototype

- **Zweck**:
  - Bestimme die Arten zu erzeugender Objekte durch die Verwendung eines prototypischen Exemplars
  - Erzeuge neue Objekte durch Kopieren dieses Prototypen
- **Beispiel:**
  - Dokumentvorlagen in Textbearbeitungsprogrammen

Prototype

- **Lösung:**



Prototype

Prototype:

- deklariert eine Methode um sich selbst zu klonen

ConcreteProduct:

- implementiert eine Operation um sich selbst zu klonen

Client:

- Erzeugt ein neues Objekt indem es einem Prototypen befiehlt sich selbst zu klonen

Prototype

Bewertung:

Vorteile:

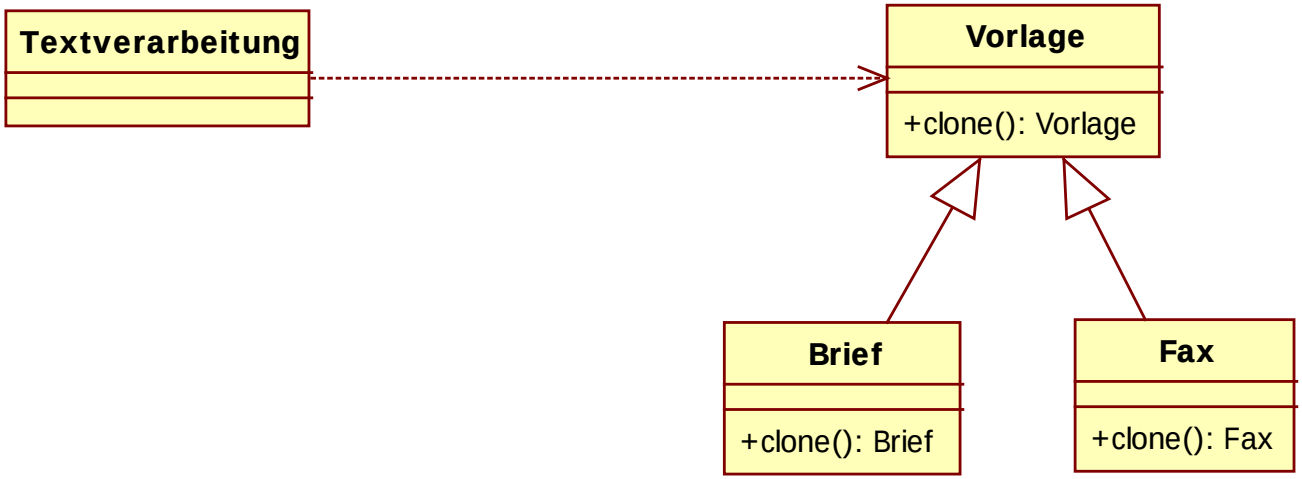
- Komplexe Objekte lassen sich schneller erzeugen.
- Neue Unterklassen können zur Laufzeit eingebunden werden.
- Neue Objekte können durch Variation der Struktur spezifiziert werden.
- Es gibt keine Erzeuger-Klassenhierarchie parallel zur Klassenhierarchie der Produkte.

Nachteil:

- Die Erstellung einer Kopie eines Objektes kann aufwendig sein.
- Jede Unterklasse muss die Kopie-Operation implementieren.
- Eventuelle Initialisierungen des kopierten Objekts müssen zusätzlich erfolgen.

Prototype

- **Beispiel: Dokumentvorlage in einem Textverarbeitungsprogramm**



Prototype

- **Verwendung (allgemein):**
  - Klassen der zu erzeugenden Objekte sollen erst zur Laufzeit spezifiziert werden
  - wenn die Erzeugung weiterer Instanzen einer Klasse teuer ist und sich die Objekte ähneln
  - Eine Hierarchie von Fabriken parallel zu einer Hierarchie von Produkten soll vermieden werden

Singleton

- **Zweck**:
  - Sichere ab, dass eine Klasse genau ein Exemplare besitzt
  - Stelle einen globalen Zugriffspunkt bereit

- **Beispiel:**
  - Ein Applikation Protokoll/Logging
  - Gemeinsamer Zugriff auf Ressourcen (z.B. Drucker, Cache)
  - Realisierung einer Komponenten-Architektur

Singleton

- **Lösung:**

Singleton

Bewertung:

Vorteile:

- Das Muster bietet eine Verbesserung gegenüber globalen Variablen
  - Zugriffskontrolle kann realisiert werden. (Lese-/Schreibzugriff)
  - Der globale Namensraum wird nicht „verschmutzt“
  - Man erklangt Kontrolle über die Initialisierungsreihenfolge
- Die Einzelinstanz muss nur erzeugt werden, wenn sie benötigt wird. (= Lazy loading)
- Alternativ: Eager Loading = (Bewusste Initialisierung)
- Sollten später mehrere Objekte benötigt werden, ist eine Änderung leichter möglich als bei globalen Variablen.

Singleton

Bewertung (continued):

Nachteil:

- Abhängigkeiten zur Singleton-Klasse werden verschleiert, d. h. ob eine Singleton-Klasse verwendet wird, erschließt sich nicht aus dem Interface einer Klasse, sondern nur anhand der Implementierung.
- Zudem wird die Kopplung erhöht, was Wiederverwendbarkeit und Übersichtlichkeit einschränkt.
- Führt einen globalen Status eins
- Der Test wird erschwert (da keine Mocken möglich ist)
- Wann kann die Ressourcen-Freigabe eines Singletons erfolgen?
- Performance-Engpass bei multithreaded-Applikationen aufgrund von Synchronisation
- Ist der Singleton auch wirklich über alle Bibliotheken/Module einzigartig?

Singleton

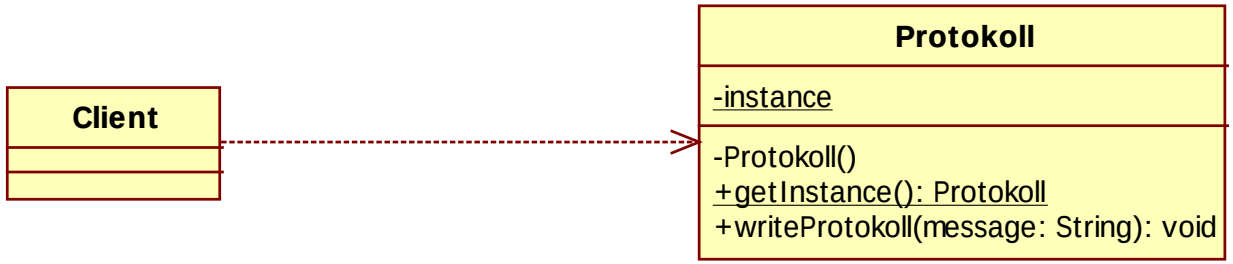
Bewertung:

Vorteile/Nachteil/Problematisch:

- Das Einzelstück kann durch Unterklassenbildung spezialisiert werden.
  - Welche Unterklasse verwendet werden soll, kann zur Laufzeit entschieden werden.
  - Dies kann zu weiteren Schwierigkeiten führen da der Konstruktor **protected** sein muss

Singleton

- **Beispiel: Ein zentrales Protokollobjekt**



Singleton

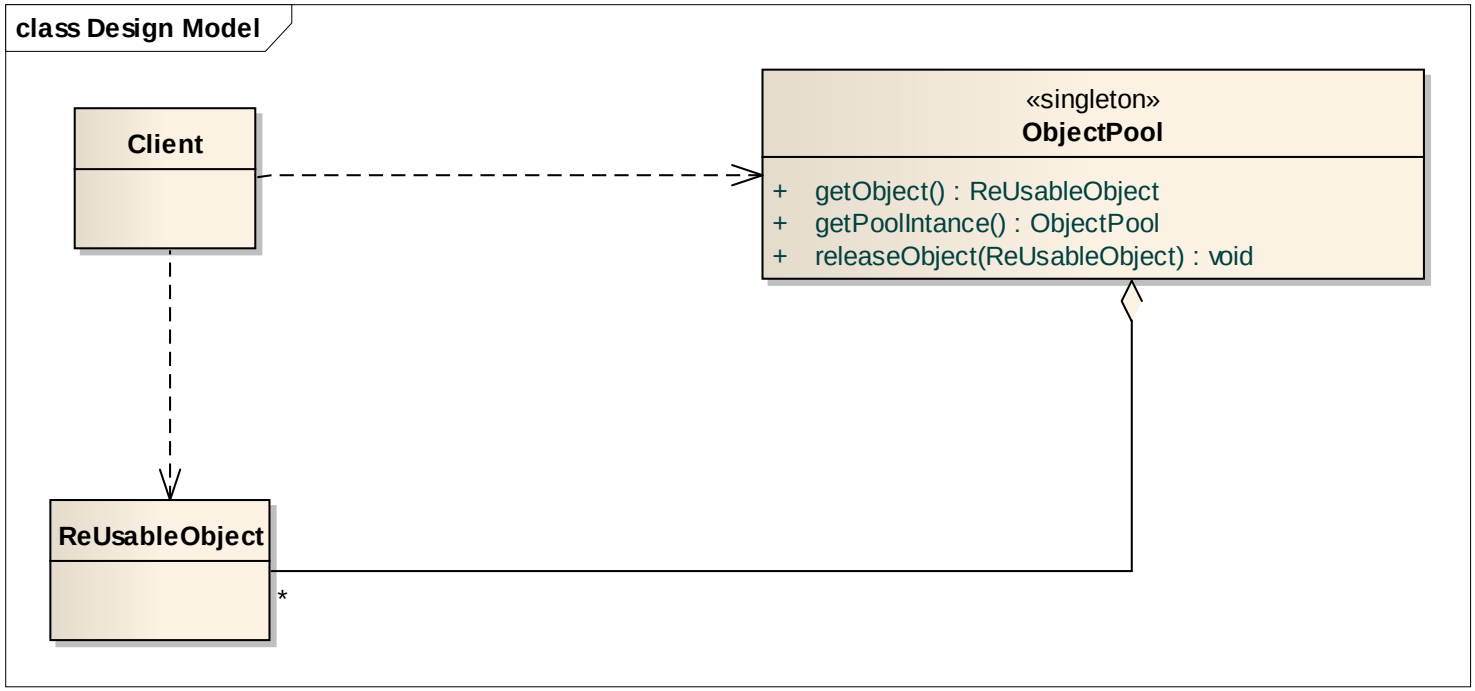
- **Verwendung (allgemein):**
  - nur ein Objekt zu einer Klasse existieren darf und ein einfacher Zugriff auf dieses Objekt benötigt wird oder
  - (das einzige Objekt durch Unterklassenbildung spezialisiert werden soll)

Objektpool

- **Problem**
  - Die Erzeugung von Objektinstanzen kann unter Umständen sehr Zeitaufwendig sein.
- **Zweck**:
  - Definiere einen Pool von Objekten
  - Stelle diese Objekte einem Client zur Verfügung
  - Mach die Objekte wiederverwendbar wenn es nicht mehr benötigt wird.
- **Beispiel:**
  - Thread-Pool
  - Datenbank-Pool

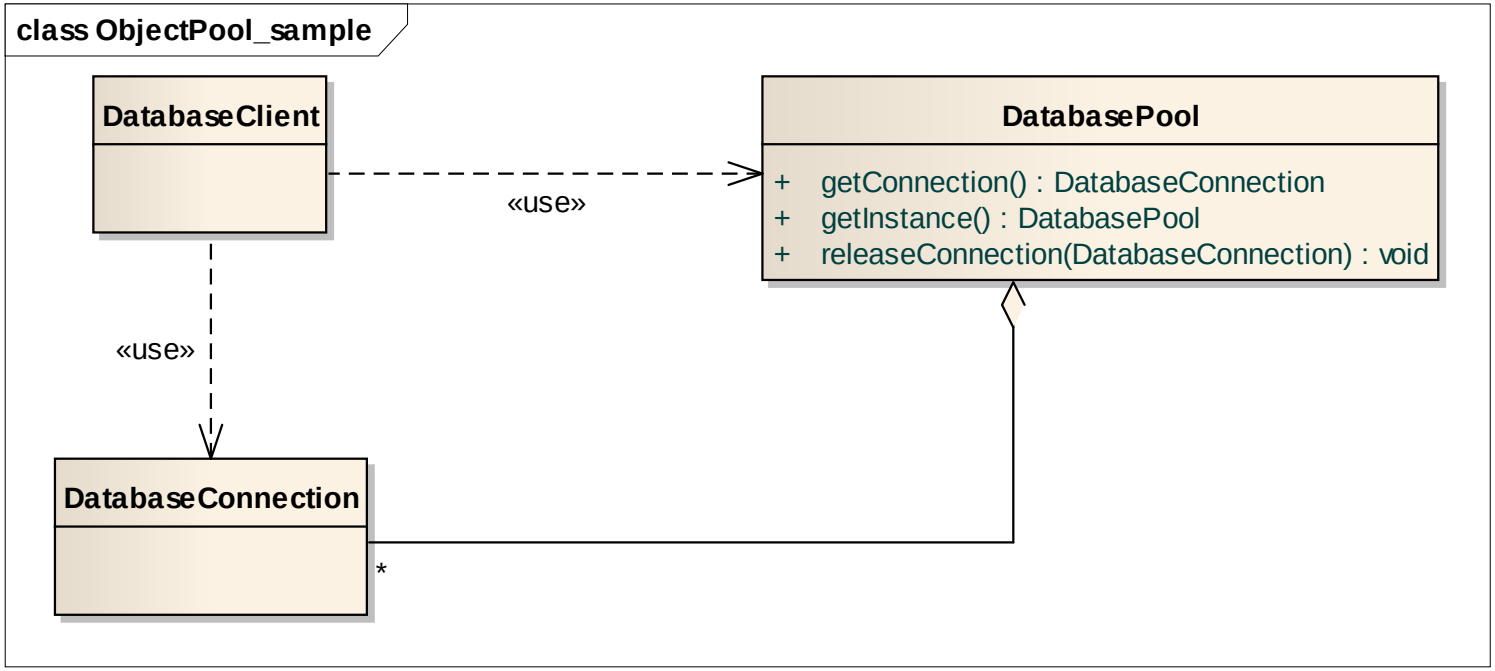
Objektpool

- **Lösung**



Objektpool

- **Anwendungsbeispiel: Connection-Pool**



Objektpool

Bewertung:

Vorteile:

- Erzeugung von aufwendigen Objekten wird optimiert
- Systemressourcen werden geschont

Nachteil:

- Rückgabe von Objekten muss durch Client erfolgen (Programmierfehler)
- Die Wiederverwendbaren Objekte müssen sich vor Ihrer Wiederverwendung wieder in einem Initialzustand befinden
- Der Pool ist eine zentrale Stelle welche unter Umständen zu einem Deadlock führen kann.