Sommersemester 2020

Wolfgang Berger

Software Paradigmen

Creational Patterns

Erzeugungsmuster

Erzeugungsmuster

- Singleton
- Prototype
- Abstract Factory
- Factory
- Builder

Prototype

- Zweck
  - Objektbasiertes Erzeugungsmuster
  - Bestimme die Arten zu erzeugender Objekte durch die Verwendung eines prototypischen Exemplars und erzeuge neue Objekte durch Kopieren eines Prototypen

Prototype

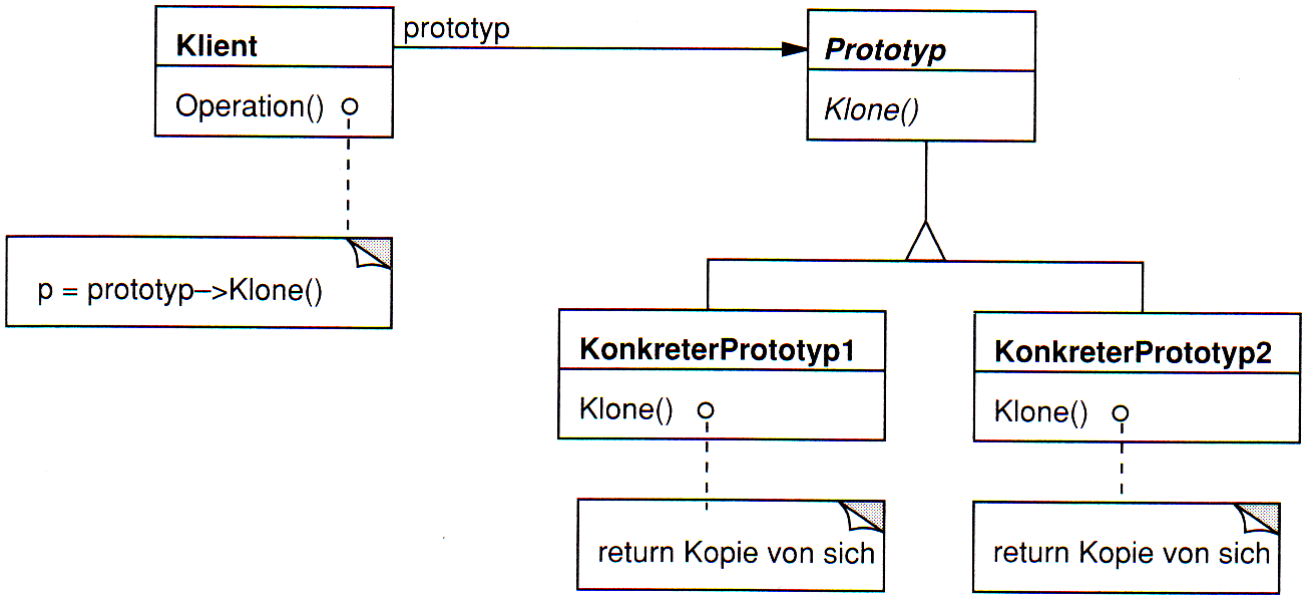
- Motivation
  - Verwende das Prototype Design Pattern wenn…
    - …die Initialisierung von Eigenschaften eines Objekts nicht statisch sondern dynamisch geschehen soll (Laden von Default-Eigenschaften aus einer Datei)
    - …ein Framework verwendet wird, das keine/eingeschränkte Vererbung anbietet.
    - …dynamische Kopplung von Daten zu Objekten der statischen vorgezogen wird (Initialisiere Objekteigenschaften nicht in der Klasse)
    - …wenn du die Anzahl der Klassen verringern möchtest

Prototype

- Anwendbarkeit
  - Verwende das Prototypmuster, wenn ein System unabhängig davon sein sollte, wie seine Produkte erzeugt, zusammengesetzt und repräsentiert werden, und…
    - … wenn die Klassen zu erzeugender Objekte erst zur Laufzeit spezifiziert werden (Anm.: damals hat es Reflection noch nicht gegeben = Erzeugen von dynamischen Klassen zur Laufzeit + natürlich Möglichkeit der Instanzierung)
    - … um zu vermeiden, eine Klassenhierarchie von Fabriken zu erstellen, die parallel zur Klassenhierarchie der Produkte verläuft (d.h. Eine Fabrik pro Produkt entwickeln zu müssen)
    - … wenn Exemplare einer Klasse nur wenige unterschiedliche Zustandskombinationen haben können.

Prototype

- Struktur



Prototype

- Teilnehmer
  - Prototyp
    - deklariert eine Schnittstelle, um sich selbst klonen zu können (z.B. Cloneable)
  - Konkreter Prototyp
    - implementiert das Prototyp Interface
  - Klient
    - erzeugt ein neues Exemplar, indem es einem Prototyp befiehlt, sich selbst zu klonen

Prototype

- Interaktionen
  - Ein Klient befiehlt einem Prototyp, sich selbst zu klonen

Prototype

- Konsequenzen
  - Versteckt konkrete Produktklassen vor dem Klienten und reduziert so die Anzahl der dem Klienten bekannten Namen (= insgesamt weniger Klassen notwendig)
  - Klient arbeitet so ohne Modifikation mit anwendungsspezifischen Objekten
  - Ermöglicht das Hinzufügen und Entfernen von Produkten zur Laufzeit (Registry für neue Prototypen verwenden)
  - Spezifikation neuer Objekte durch Variation von Werten – Entwurf neuer Objekte ohne Programmiertätigkeit (im Gegensatz zum Plugin Pattern)

Prototype

- Konsequenzen
  - Spezifikation neuer Objekte durch Variation der Struktur
  - Verringerte Unterklassenbildung
  - Dynamisches Konfigurieren einer Anwendung mit Klassen
  - Achtung: Beim Klonen der Objekt auf DEEP-COPY achten! Jedes Aggregierte oder Assoziierte Objekt muss ebenfalls Cloneable sein und auch tatsächlich geklont werden!
  - Nachteil: Je breiter die Zusammensetzungsmöglichkeiten von Produkten wird desto generischer muss die Prototypen Klasse ausfallen → erhöht die Fehlerwahrscheinlichkeit

Prototype

- Implementierung
  - Verwendung eines Prototypemanagers (dynamisches Hinzufügen und Entfernen von Prototypen)
  - Implementierung der clone() Operation
    - Achtung bei Aggregationen und Assoziationen
    - flaches versus tiefes Kopieren
    - Vor allem zirkuläre Beziehungen sind trickreich!
    - Einfach: Speichern und Laden der Objekte

Prototype

- Implementierung
  - Initialisierung der geklonten Objekte
    - Eventuell initialize() Methode entwickeln
  - Boolsche Variable cloned, default = false
    - Typgebende Variablen dürfen von geklonten Objekten nicht mehr verändert werden
    - clone() darf von geklonten Objekten nicht mehr aufgerufen werden

Prototype

- Übung 2!