Pattern III "Erzeugungsmuster"

XP-Praktikum 2005a

Philip Ritzkopf

23. Februar 2005

Übersicht

Grundidee

Erzeugungsmuster

- bilden Abstraktion des Instanziierungsprozesses

Grundidee

Erzeugungsmuster

- bilden Abstraktion des Instanziierungsprozesses
- machen Systeme unabhängig davon, wie Objekte erzeugt, zusammengesetzt und dargestellt werden

Grundidee

Erzeugungsmuster

- bilden Abstraktion des Instanziierungsprozesses
- machen Systeme unabhängig davon, wie Objekte erzeugt, zusammengesetzt und dargestellt werden
- kapseln welche konkreten Klassen das System verwendet

Grundidee

Erzeugungsmuster

- bilden Abstraktion des Instanziierungsprozesses
- machen Systeme unabhängig davon, wie Objekte erzeugt, zusammengesetzt und dargestellt werden
- kapseln welche konkreten Klassen das System verwendet
- vermeiden hardcoded Instanziierungsanweisungen

Zweck

Factory Method

- erlaubt Erzeugung konkreter Objekte und ist dabei nur von abstrakten Schnittstellen abhängig

Zweck

Factory Method

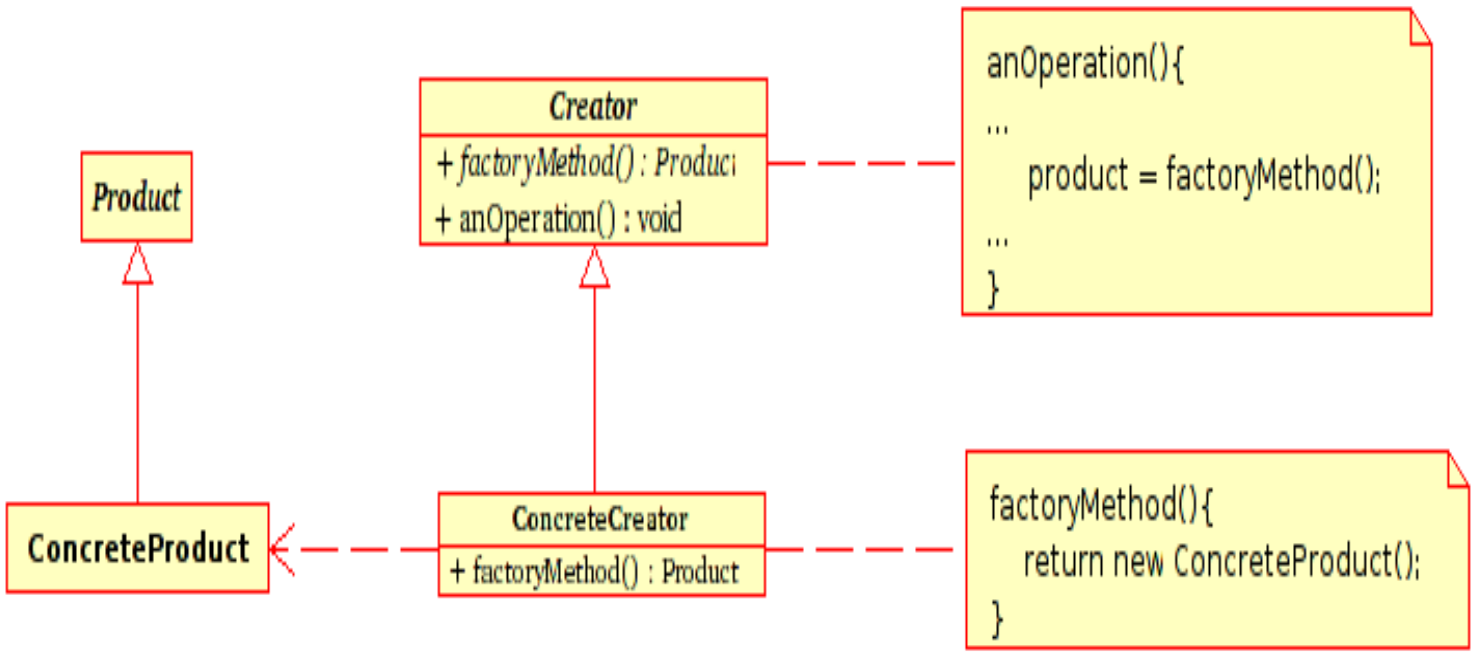
- erlaubt Erzeugung konkreter Objekte und ist dabei nur von abstrakten Schnittstellen abhängig
- lässt Unterklassen entscheiden welche konkreten Klassen instanziiert werden sollen

Zweck

Factory Method

- erlaubt Erzeugung konkreter Objekte und ist dabei nur von abstrakten Schnittstellen abhängig
- lässt Unterklassen entscheiden welche konkreten Klassen instanziiert werden sollen
- verschiebt das Problem der Abhängigkeit von konkreten Klassen, d.h. jemand muss einen konkreten Creator erzeugen

Struktur


Product
Creator
+ factoryMethod() : Product
+ anOperation() : void
anOperation(){
...
product = factoryMethod();
...
}
ConcreteProduct
ConcreteCreator
+ factoryMethod() : Product
factoryMethod(){
return new ConcreteProduct();
}

Teilnehmer I

- Product
  - definiert die Schnittstelle für die Objekte, die durch die Factory Method erzeugt werden

Teilnehmer I

- Product
  - definiert die Schnittstelle für die Objekte, die durch die Factory Method erzeugt werden
- ConcreteProduct
  - implementiert die durch Product festgelegte Schnittstelle

Teilnehmer II

- Creator
  - deklariert die Factory Method, welche ein Objekt vom Typ Product liefert
  - kann eine Standardimplementierung für Factory Method besitzen, die ein Standard- ConcreteProduct Objekt zurück gibt
  - kann Factory Method aufrufen, um ein ConcreteProduct Objekt zu liefern

Teilnehmer II

- Creator
  - deklariert die Factory Method, welche ein Objekt vom Typ Product liefert
  - kann eine Standardimplementierung für Factory Method besitzen, die ein Standard- ConcreteProduct Objekt zurück gibt
  - kann Factory Method aufrufen, um ein ConcreteProduct Objekt zu liefern
- ConcreteCreator
  - redefiniert die Factory Method, um ein ConcreteProduct Exemplar zurück zu geben

Zusammenspiel

- Creator verlässt sich darauf, dass die Unterklassen die Factory Method redefinieren, so dass eine Instanz des gewünschten ConcreteProduct geliefert wird

Anwendbarkeit

Benutze Factory Method, wenn

- eine Klasse die Klasse von Objekten, die sie erzeugen muss nicht vorhersehen kann

Anwendbarkeit

Benutze Factory Method, wenn

- eine Klasse die Klasse von Objekten, die sie erzeugen muss nicht vorhersehen kann
- Subklassen einer Klasse die zu erzeugenden Objekte festlegen soll

Anwendbarkeit

Benutze Factory Method, wenn

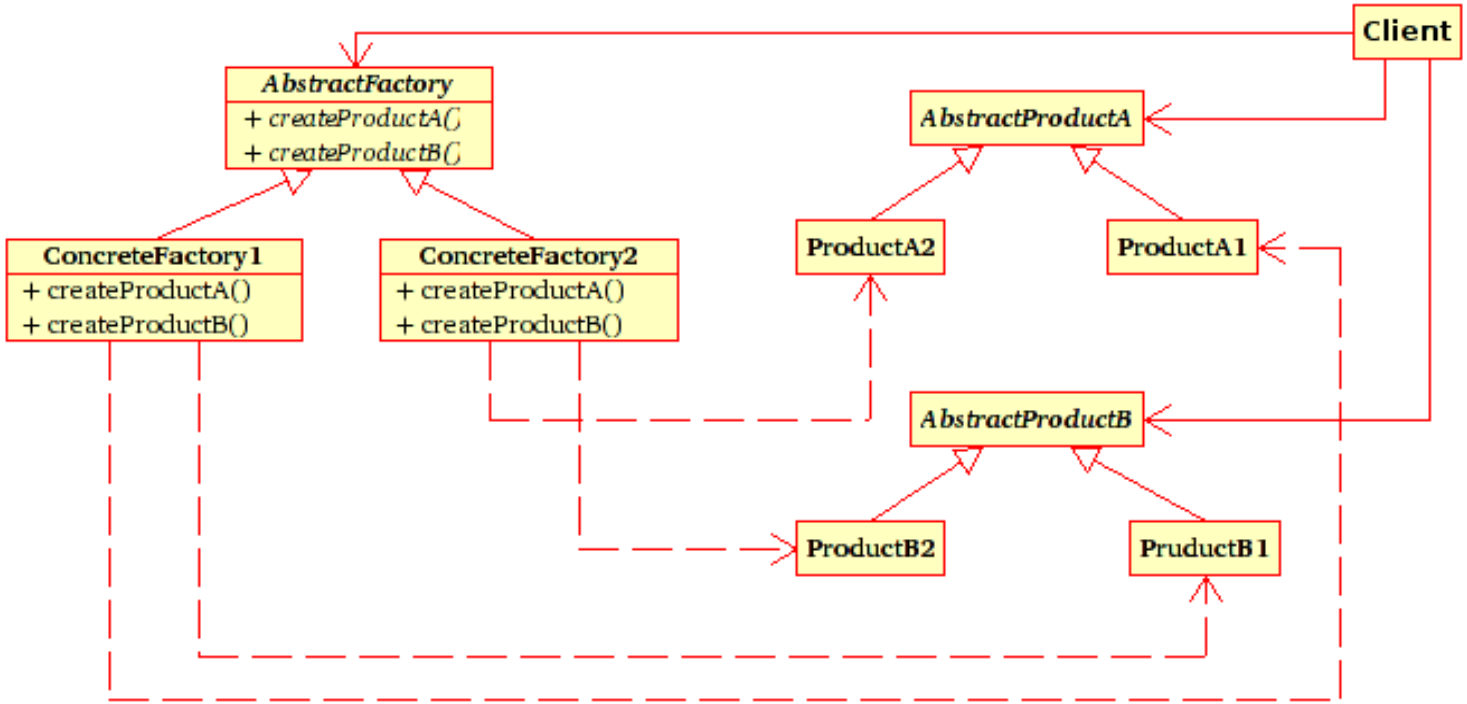
- eine Klasse die Klasse von Objekten, die sie erzeugen muss nicht vorhersehen kann
- Subklassen einer Klasse die zu erzeugenden Objekte festlegen soll
- Klassen Zuständigkeiten an eine von mehreren Hilfsunterklassen delegieren, aber das Wissen über die Weiterleitung an einem Ort lokalisiert werden soll

Zweck

Abstract Factory

- stellt eine Schnittstelle zur Erzeugung von Familien verwandter Objekte zur Verfügung ohne ihre konkreten Klassen angeben zu müssen

Struktur


Client
AbstractFactory
+ createProductA()
+ createProductB()
AbstractProductA
ConcreteFactory1
+ createProductA()
+ createProductB()
ConcreteFactory2
+ createProductA()
+ createProductB()
ProductA2
ProductA1
AbstractProductB
ProductB2
PruductB1

Teilnehmer I

- AbstractFactory
  - deklariert abstrakte Schnittstelle für Erzeugungsoperationen

Teilnehmer I

- AbstractFactory
  - deklariert abstrakte Schnittstelle für Erzeugungsoperationen
- ConcreteFactory
  - implementiert Operationen zur Erzeugung von Produkten

Teilnehmer II

- AbstractProduct
  - deklariert Schnittstelle für einen Produkttyp

Teilnehmer II

- AbstractProduct
  - deklariert Schnittstelle für einen Produkttyp
- ConcreteProduct
  - definiert Produkt, das von ConcreteFactory erzeugt wird und implementiert Produktoperationen

Teilnehmer II

- AbstractProduct
  - deklariert Schnittstelle für einen Produkttyp
- ConcreteProduct
  - definiert Produkt, das von ConcreteFactory erzeugt wird und implementiert Produktoperationen
- Client
  - verwendet nur abstrakte Schnittstellen von AbstractFactory und AbstractProduct

Zusammenspiel

- Normalerweise wird zur Laufzeit nur eine Instanz einer ConcreteFactory erzeugt
- ConcreteFactory erzeugt Produktobjekte mit einer bestimmten Implementierung
- Um unterschiedliche Produktobjekte erzeugen zu können müssen Klienten unterschiedliche ConcreteFactory Objekte verwenden
- AbstractFactory verschiebt die Verantwortung für die Erzeugung von Produktobjekten zur ConcreteFactory

Anwendbarkeit

Benutze Abstract Factory, wenn

- das System unabhängig davon sein soll wie seine Produkte erzeugt, zusammengesetzt und dargestellt werden

Anwendbarkeit

Benutze Abstract Factory, wenn

- das System unabhängig davon sein soll wie seine Produkte erzeugt, zusammengesetzt und dargestellt werden
- das System mit unterschiedlichen Produktfamilien konfigurierbar sein soll

Anwendbarkeit

Benutze Abstract Factory, wenn

- das System unabhängig davon sein soll wie seine Produkte erzeugt, zusammengesetzt und dargestellt werden
- das System mit unterschiedlichen Produktfamilien konfigurierbar sein soll
- eine Familie verwandter Produkte dafür ausgelegt ist, dass sie zusammen verwendet werden und diese Bedingung eingehalten werden soll

Anwendbarkeit

Benutze Abstract Factory, wenn

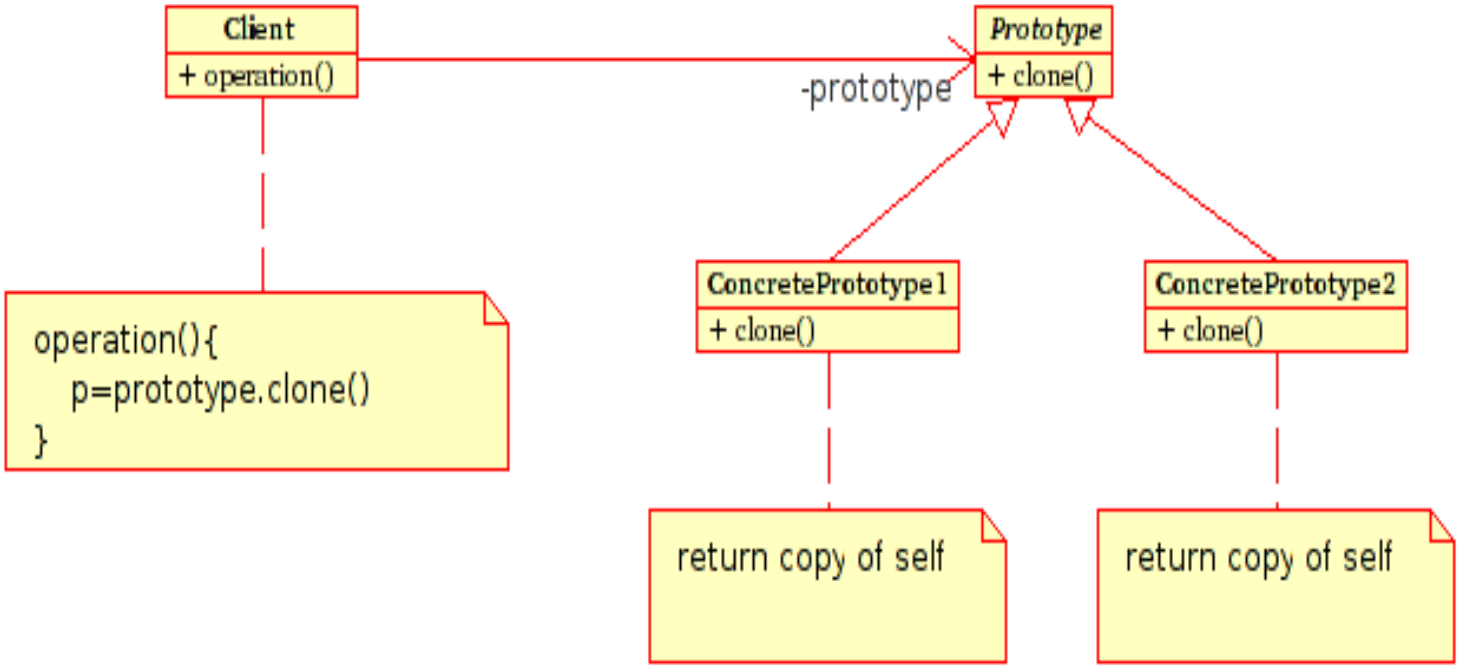
- das System unabhängig davon sein soll wie seine Produkte erzeugt, zusammengesetzt und dargestellt werden
- das System mit unterschiedlichen Produktfamilien konfigurierbar sein soll
- eine Familie verwandter Produkte dafür ausgelegt ist, dass sie zusammen verwendet werden und diese Bedingung eingehalten werden soll
- eine Klassenbibliothek erstellt werden soll, die nur ihre Schnittstellen aber nicht die Implementierung offen legt

Zweck

Prototype

- spezifiziert Arten zu erzeugender Objekte durch ein pototypisches Exemplar und erzeugt neue Objekte durch kopieren (klonen) dieses Prototypen

Struktur


Client
+ operation()
Prototype
+ clone()
-prototype
operation(){
p=prototype.clone()
}
ConcretePrototype1
+ clone()
ConcretePrototype2
+ clone()
return copy of self
return copy of self

Teilnehmer

- Prototype
  - deklariert eine Schnittstelle, um sich selbst zu klonen

Teilnehmer

- Prototype
  - deklariert eine Schnittstelle, um sich selbst zu klonen
- ConcretePrototype
  - implementiert eine Operation, um sich selbst zu klonen

Teilnehmer

- Prototype
  - deklariert eine Schnittstelle, um sich selbst zu klonen
- ConcretePrototype
  - implementiert eine Operation, um sich selbst zu klonen
- Client
  - erzeugt neue Objekte, indem es den Prototyp auffordert sich zu klonen

Zusammenspiel

- Ein Klient fordert einen Prototyp auf sich zu klonen

Anwendbarkeit

Benutze Prototype, wenn

- Klassen der zu erzeugenden Objekte zur Laufzeit spezifiziert werden müssen

Anwendbarkeit

Benutze Prototype, wenn

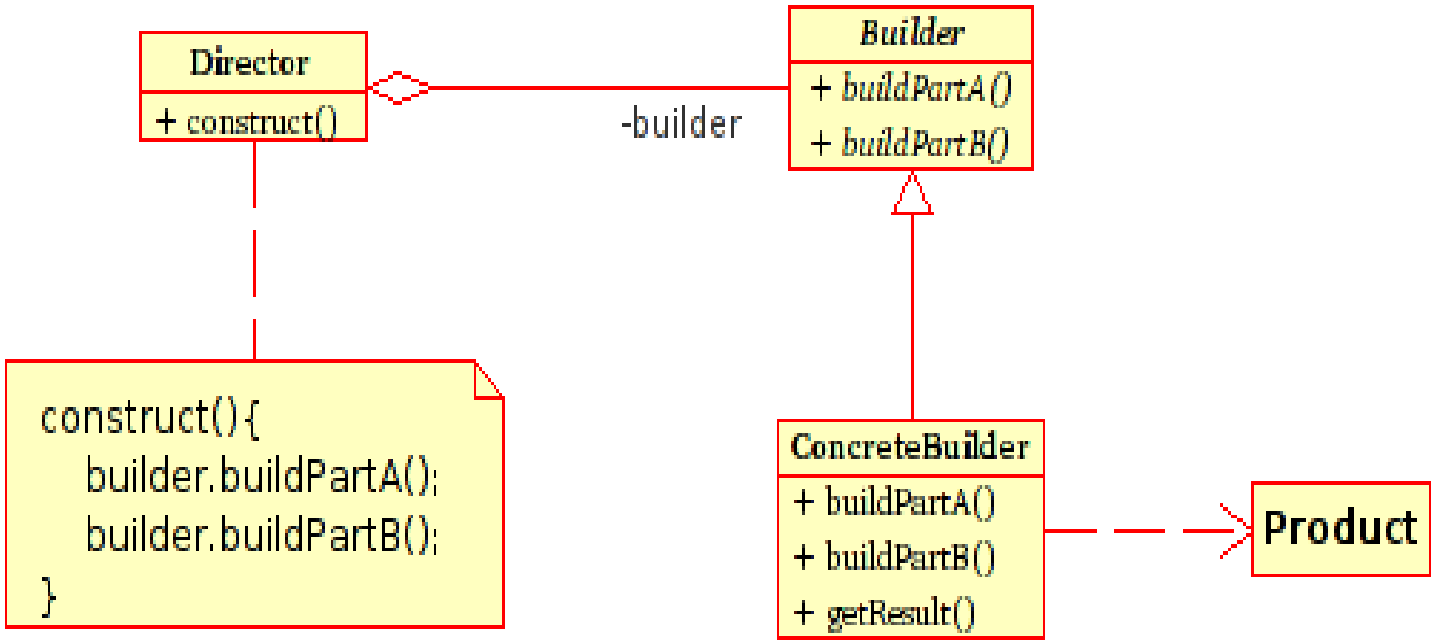
- Klassen der zu erzeugenden Objekte zur Laufzeit spezifiziert werden müssen
- parallele Klassenhierarchien zwischen Fabriken (Factory) und Produkten (Product) vermieden werden sollen

Zweck

Builder

- trennt die Konstruktion eines komplexen Objekts von seiner Darstellung, so dass derselbe Konstruktionsprozess unterschiedliche Darstellungen erzeugen kann

Struktur


Director
+ construct()
-builder
Builder
+ buildPartA()
+ buildPartB()
construct(){
builder.buildPartA();
builder.buildPartB();
}
ConcreteBuilder
+ buildPartA()
+ buildPartB()
+ getResult()
Product

Teilnehmer I

- Builder
  - deklariert eine abstrakte Schnittstelle zur Erzeugung von Teilen eines Produktobjekts

Teilnehmer I

- Builder
  - deklariert eine abstrakte Schnittstelle zur Erzeugung von Teilen eines Produktobjekts
- ConcreteBuilder
  - konstruiert und setzt Teile des Produkts zusammen, indem es die Builder Schnittstelle implementiert
  - definiert und verwaltet die Repräsentation die es erzeugt
  - bietet eine Schnittstelle an, um das Produkt abrufen zu können

Teilnehmer II

- Director
  - benutzt Builder-Schnittstelle um ein Objekt zu konstruieren

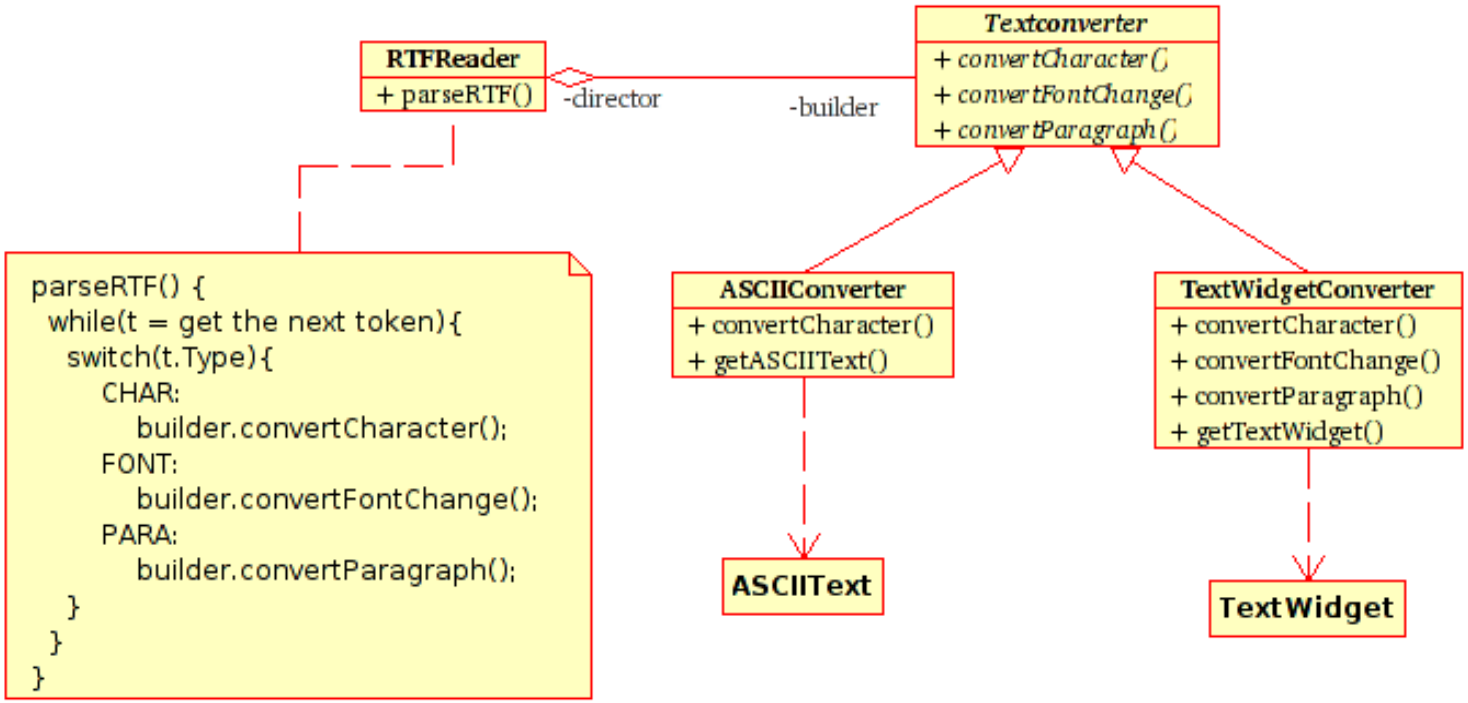
Teilnehmer II

- Director
  - benutzt Builder-Schnittstelle um ein Objekt zu konstruieren
- Product
  - repräsentiert das zu konstruierende komplexe Objekt
  - interne Darstellung wird durch ConcreteBuilder erzeugt
  - umfasst Klassen, die die einzelnen Teile definieren

Zusammenspiel

- Klient instanziiert ein Director-Objekt und konfiguriert es mit dem gewünschten Builder-Objekt
- Director sendet Nachrichten an den Builder immer wenn ein Produktteil erzeugt werden soll
- Builder bearbeitet die Anforderungen des Director und fügt dem Produkt Teile hinzu
- Klient ruft das Produkt beim Builder ab

Beispiel


RTFReader
+ parseRTF()
-director
-builder
Textconverter
+ convertCharacter()
+ convertFontChange()
+ convertParagraph()
parseRTF() {
while(t = get the next token){
switch(t.Type){
CHAR:
builder.convertCharacter();
FONT:
builder.convertFontChange();
PARA:
builder.convertParagraph();
}
}
}
ASCIIConverter
+ convertCharacter()
+ getASCIIText()
TextWidgetConverter
+ convertCharacter()
+ convertFontChange()
+ convertParagraph()
+ getTextWidget()
ASCIIText
TextWidget

Anwendbarkeit

Benutze Builder, wenn

- der Erzeugungsprozess eines komplexen Objekts unabhängig von den einzelnen Komponenten des Objekts und davon, wie diese zusammengesetzt werden, sein soll

Anwendbarkeit

Benutze Builder, wenn

- der Erzeugungsprozess eines komplexen Objekts unabhängig von den einzelnen Komponenten des Objekts und davon, wie diese zusammengesetzt werden, sein soll
- der Konstruktionsprozess unterschiedliche Darstellungen des zu erzeugenden Objekts ermöglichen muss

Literatur I

Robert C. Martin

Agile Software Development - Principles, Patterns, and Practices

Prentice Hall, Pearson Education Inc.

Bruce Eckel, Chuck Allison

Thinking in C++ Volume Two: Practical Programming

http://mindview.net/Books/TICPP/ThinkingInCPP2e.html

Bruce Eckel

Thinking in Patterns with Java

http://mindview.net/Books/TIPatterns/

Literatur II

E. Gamma, R. Helm, R. Johnson, J. Vlissides
Design Patterns - Elements of Reusable Object-Oriented Software
Addison Wesley