**Abstract Factory Design Pattern**

*„Stelle ein Interface für die Erzeugung von Gruppen von ähnlichen oder abhängigen Objekten bereit, ohne dabei die konkreten Klassen zu spezifizieren“*

**Allgemeines**

* Erzeugungsmuster
* Schnittstelle zur Erzeugung von Objekten
* Konkrete Klassen nicht näher festgelegt
* „Factory of Factories“ 🡪 Konkrete Factories hinter abstrakter Factory

**Motivation**

* System unabhängig von
  + Erzeugung
  + Zusammenstellung
  + Darstellung seiner Objekte
* Gruppe von ähnlichen oder abhängigen Objekte
* Bereitstellung einer Klassenbibliothek
  + Öffentliche Interfaces
  + Private Implementierungen

**Konkrete Anwendungen**

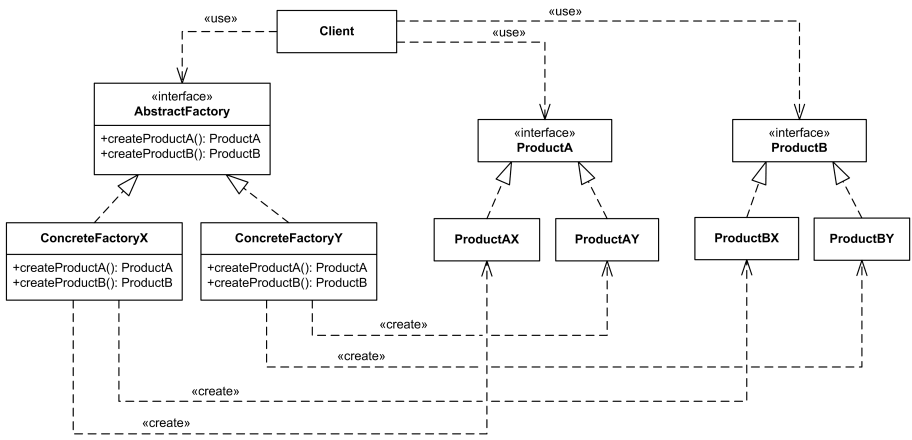
* InterViews
  + Sammlung von Application Frameworks
  + Entwickeln von benutzerdefinierten, grafischen Editoren
  + Abstrakte Factories zur Erzeugung der UI-Elemente
    - Verschiedene Themes z.B.
  + Abstrakte Factory für das Layout
    - Horizontale oder vertikale Orientierung der Elemente
* ET++
  + Von Erich Gamma
  + Application Framework
  + Entwickeln von Applikationen mit
    - interaktiver Benutzerschnittstelle
    - Textueller und grafischer Darstellung von Daten
    - Baumeditor, Spreadsheets, CASE-Tools, …
  + Abstract Factory zur Portabilität zwischen verschiedenen Fenstersystemen (z.B. X11 unter Windows und SunView in UNIX)

**Verwendete Patterns in der Abstract Factory**

|  |  |  |
| --- | --- | --- |
| **Factory Method oder Prototype** | | **Singleton** |
| Factory Method erzeugt konkrete Klassen | Konkrete Factory hat Prototypenobjekte und erzeugt neue Objekte durch Klonen des Protoypen | Normalerweise nur eine abstrakte Factory pro Objektgruppe |

**Komponenten und UML**

* Factory
  + Interface AbstractFactory
  + Konkrete Factory – implementiert Abstract Factory (VehicleFactory implements AbstractFactory) 🡪 erzeugt Objekte
* Zu erzeugende Objekte
  + Interface mit gemeinsamen Eigenschaften (Vehicle)
  + Konkrete Klassen – implementieren Interface (Car implements Vehicle)
* Optional: Factory Producer, der die abstrakten Factories erzeugt



**Vor- und Nachteile**

|  |  |
| --- | --- |
| Vorteile | Nachteile |
| Isolation konkreter Klassen  Einfaches Tauschen der Objekte  Konsistenz der Objekte | Erweiterbarkeit aufwändig  Lesbarkeit erschwert  Bei falscher Nutzung: „Anti pattern“ |

**Beispiel: Fahrzeug- und Kundenverwaltung, Durchführung von Verkäufen**

**Allgemeine Information**

Das selbst implementierte Beispiele beinhaltet Funktionalitäten zur Verwaltung von Fahrzeugen, Kunden und den Verkäufen von Fahrzeugen zwischen den Kunden. Die abstrakte Factory erzeugt dabei die konkreten Factories *VehicleFactory/SpecialVehicleFactory* und *CustomerFactory*. Alle Fahrzeuge implementieren das Interface *Vehicle* und alle Kunden *Customer*.

Ein Fahrzeug kann ein Objekt von folgenden Klassen sein: PKW, LKW, Motorrad oder Traktor.  
Ein Kunde ist entweder eine Privatperson oder ein Unternehmen.

Die Anwendung ist mittels ASP.NET Webforms realisiert und hat einen direkten Zugriff auf die LocalDB-Instanz. Durch einen Chart werden die Gesamterlöse pro Kunde angezeigt.

**Datenbank**

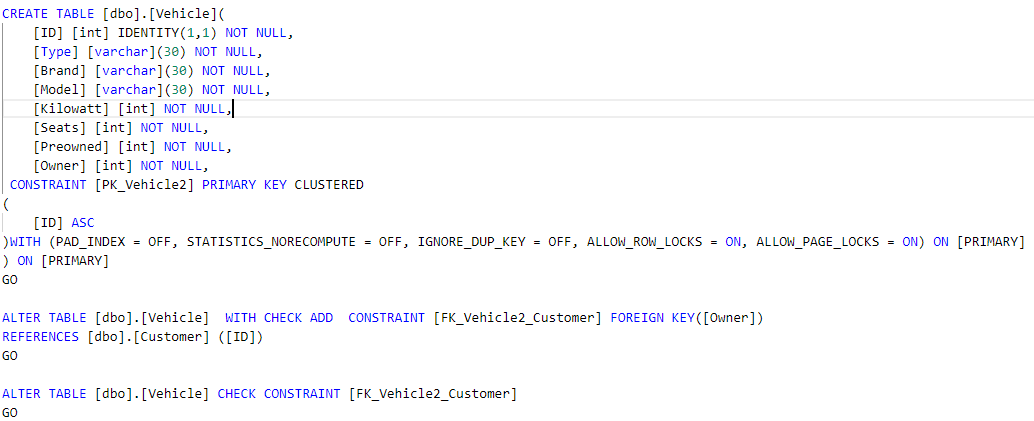


Abbildung 1 Fahrzeugtabelle

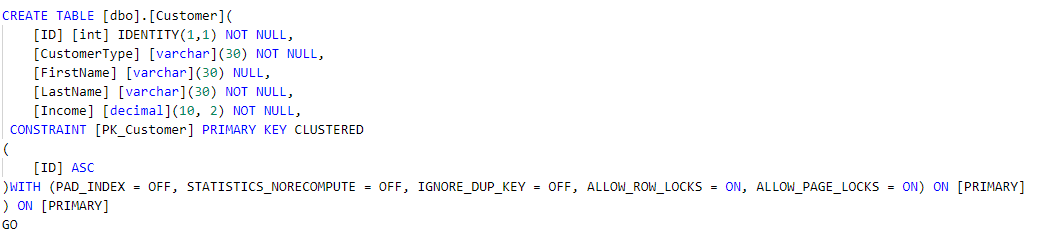


Abbildung 2 Kundentabelle

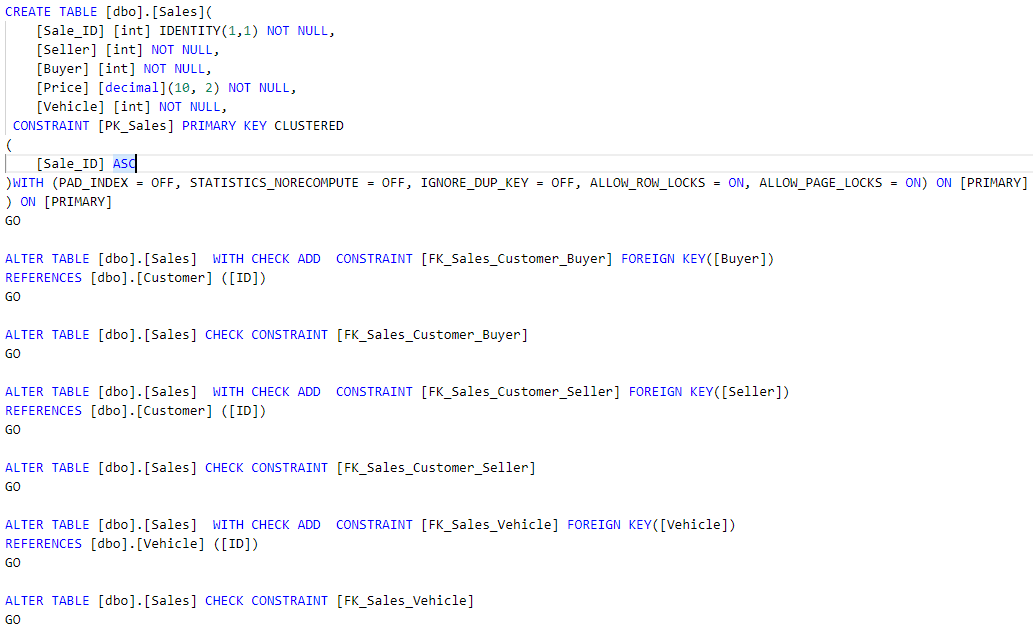


Abbildung 3 Verkaufstabelle

**Anwendung**

Die Kommunikation der ASP.NET Anwendung zur lokalen SQL-Server Instanz erfolgt mittels des SQLDataAdapters und den SqlCommands.

Es gibt drei verschiedene Seiten (Fahrzeuge, Kunden und Verkäufe), wobei in auf der Fahrzeug- und Kundenseite das Abstract Factory Pattern zur Anwendung kommt.

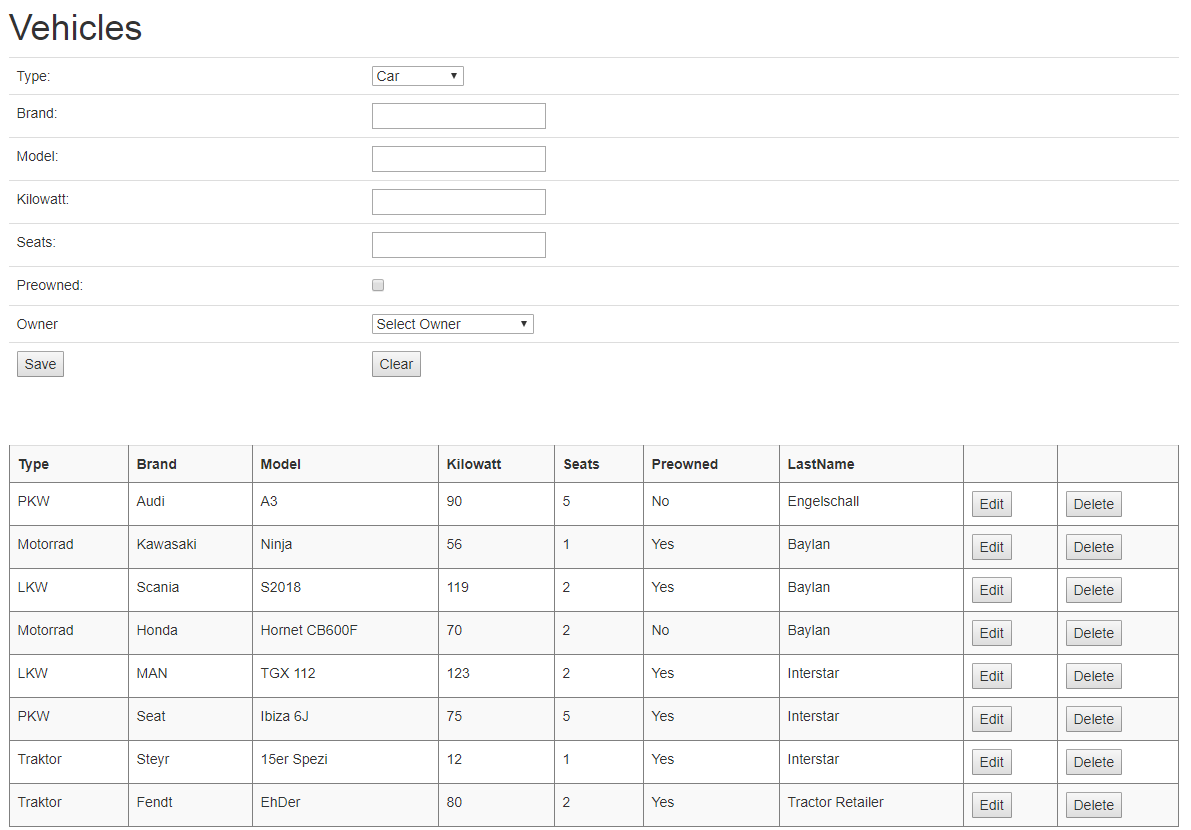


Abbildung 4 Fahrzeugseite

Da die Kundenseite von der Struktur her exakt gleich ist, ist dafür kein extra Screenshot im Handout. Auf der Kunden- und Fahrzeugseite kann der Benutzer alle CRUD-Operationen auf die jeweiligen Daten ausführen wie im obigen Screenshot zu sehen ist.

Das Abstract Factory Pattern wird bei Fahrzeugen wie folgt angewendet:

Die Fahrzeugseite beinhaltet eine AbstractFactory mit dem generischen Typ Vehicle und eine mit dem Typ Customer. Wenn der Benutzer mit der Eingabe der Daten im Formular fertig ist, wird ein Objekt vom Interfacetyp Vehicle von der abstrakten Fahrzeugfactory erzeugt. Die Programmlogik kennt also nur die Interfaces und nicht die genauen Implementierungen.





Bei einem Verkauf wählt man einen Verkäufer aus, dann werden automatisch die in seinem Besitz befindlichen Fahrzeuge in das Fahrzeugdropdown eingefüllt und der Kunde aus dem Käuferdropdown ausgewählt. Zum Abschluss der Transaktion gibt man einen Preis in € ein und klickt den Completebutton. Im unteren Bereich der Seite sieht man die Erlöse aller Fahrzeugverkäufe für jeden Kunden, welche mit dem Klick auf Complete ebenfalls aktualisiert werden.

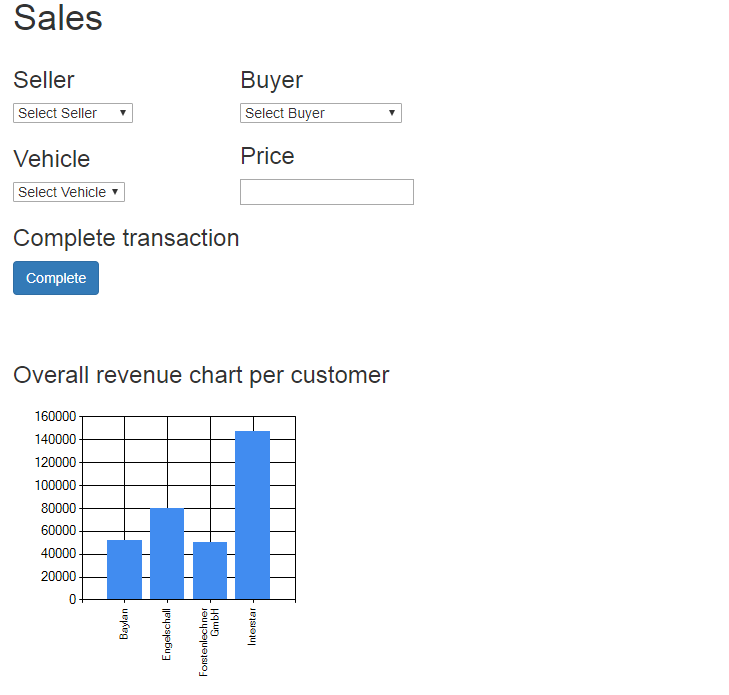


Abbildung 5 Verkaufsseite

**Github-Repo: https://github.com/SimonPrimetzhofer/AbstractFactory-Pattern**

**Quellen**

https://www.tutorialspoint.com/design\_pattern/abstract\_factory\_pattern.htm

https://www.philipphauer.de/study/se/design-pattern/abstract-factory.php

https://de.wikipedia.org/wiki/Abstrakte\_Fabrik

https://refactoring.guru/design-patterns/abstract-factory

https://www.geeksforgeeks.org/abstract-factory-pattern/

https://stackoverflow.com/questions/5739611/differences-between-abstract-factory-pattern-and-factory-method

https://medium.com/@hitherejoe/design-patterns-abstract-factory-39a22985bdbf

https://www.baeldung.com/java-abstract-factory-pattern

https://github.com/eugenp/tutorials/tree/master/patterns/design-patterns/src/main/java/com/baeldung/creational/abstractfactory