1

* Entwurfsmuster aus dem Bereich der Softwareentwicklung
* Kategorie der Erzeugungsmuster
  + Objektmuster
    - Abstrakte Fabrik (*abstract factory*)
    - Einzelstück *(singleton)*
    - Erbauer *(builder)*
    - Prototyp *(prototype)*
* eine mehrfache Anwendung der Fabrikmethode
* Eine abstrakte Fabrik (Abstract Factory) wird im Allgemeinen mittels Fabrikmethoden realisiert.

2

Gemeinsamkeiten:

Beide Pattern entkoppelt den Client von konkreten Typen. Der Client kennt nur die Schnittstelle zur Factory und den Produkten. Welche Produkte konkret erstellt werden, entscheidet die Unterklasse/Implementierung der Schnittstelle. Der Client stützt sich allein auf Abstraktion.

Abstract Factory

* Ganze Produktfamilie wird erstellt
* Breite Schnittstelle
* nur ein Interface und enthält keinen Implementierungscode.
* Benutzt Factory Methods, aber ohne generischen Code.

Factory Method

* Ein Produkt wird erstellt
* Schmale Schnittstelle
* Schnittstelle ist oft abstrakt und enthält generischen Code (Herstellungscode, den alle Produkte durchlaufen müssen)

3

* Das Abstract Factory Design Pattern dient der Definition einer zusammenhängenden Familie aus Produkten (engl. products). Die Familien können ausgetauscht werden.
* Diese Factoryschnittstelle, die namensgebende Abstract Factory, definiert für jedes Produkt der Produktfamilie (Produktsatz) eine Operation, mit der der Client eine Instanz des jeweiligen Produkts erhalten kann.
* Der Client ist damit von einer bestimmten Factoryimplementierung entkoppelt. Es stützt sich allein auf Abstraktion - sowohl bei den Produkten als auch bei der Factory.

4

Vorteile

* + Clientcode allgemeingültig kein Code für spezielle Fälle notwendig.
  + Konsistenz. Es wird sichergestellt, dass nur jene Objekte zum Client gelangen, die auch zusammenpassen. Es ist weiterhin gewährleistet, dass immer nur ein konkretes Familienmitglied eines Typs zur gleichen Zeit im Einsatz ist.
  + Flexibilität. Ganze Objektfamilien können ausgetaucht werden, ohne dass der Clientcode bricht, da sich der Client nur auf Abstraktionen (Abstract Factory, Productschnittstellen) stützt.
  + Einfache Erweiterung mit neuen Produktfamilien. Neue Productsets können sehr einfach ins System integriert werden. Dazu ist lediglich das erneute Implementieren der Factoryschnittstelle nötig. Anschließend muss nur noch an einer zentralen Stelle im Client die neue Factory instanziiert werden.
  + Wiederverwendbarkeit. Konkrete Products können Mitglieder verschiedener Produktfamilien sein.
  + Verschlankung des Clientcodes

Nachteile

* + Unflexibilität hinsichtlich neuer Familienmitglieder. Soll der Produktfamilie ein neues Produkt hinzugefügt werden, so ist eine Änderung der Schnittstelle der Abstract Factory notwendig. Dies führt aber zum Brechen von Code aller konkreten Factorys. Der Änderungsaufwand ist groß. Daher sollte zu Beginn sehr genau überlgleichegt werden, welche Produkttypen erstellt werden sollen. Je weiter das System zum Zeitpunkt der nötigen Erweiterung fortgeschritten ist, umso mehr Code muss modifiziert werden.

5

* Eine Objektfamilie bereitgestellt werden soll, aber noch keine Aussagen zu den konkreten Implementierungen gemacht werden kann oder soll. Stattdessen werden Interfaces bereitgestellt.
* Wenn ein System losgelöst davon sein soll, *wie* bestimmte Objekte erstellt werden.
* Wenn verschiedene Objekte zu einem Kontext erstellt werden und daher immer zusammenhängend erstellt werden müssen. Also immer dann, wenn Konsistenz im Objektpool sichergestellt werden muss.

Beispiel

| **Klasse** | **Abstract Factory Teilnehmer** |
| --- | --- |
| AbstractGenerator | AbstractFactory |
| Konkrete Generatoren | Concrete Factorys |
| Spielweltobjekte (Tiere, Pflanzen, Untergrund) | Products |
| konkrete Spielweltobjekte (Elefant, Sand, Schnee, Kaktus etc.) | Concrete Products |