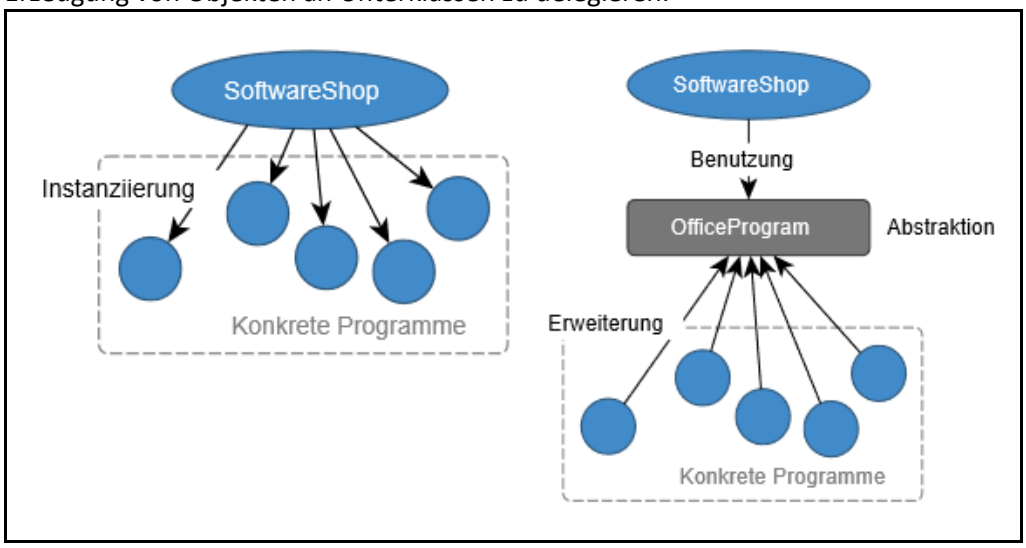
Factory Method

1 Factory Method

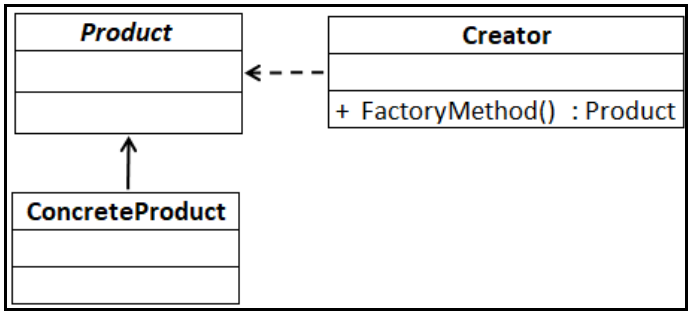
Dieses Designpattern heißt oft nur Factory und zählt zu den Erzeugungsmustern.
Es besteht im Großen und Ganzen aus einer Klasse, die ein **Objekt eines bestimmten Typs erzeugt**. Welche konkrete Klasse tatsächlich erzeugt wird, entscheidet die Fabrik. Man verzögert die Entscheidung über die konkrete Klasse auf die **Laufzeit** (und nicht die Kompilierzeit).
Die erzeugten Objekte müssen eine gemeinsame Basisklasse bzw. ein gemeinsames Interface haben.
Definition Gang Of Four:
„Definiere eine Klassenschnittstelle mit Operationen zum Erzeugen eines Objekts, aber lasse Unterklassen entscheiden, von welcher Klasse das zu erzeugende Objekt ist. Fabrikmethoden ermöglichen es einer Klasse, die Erzeugung von Objekten an Unterklassen zu delegieren."



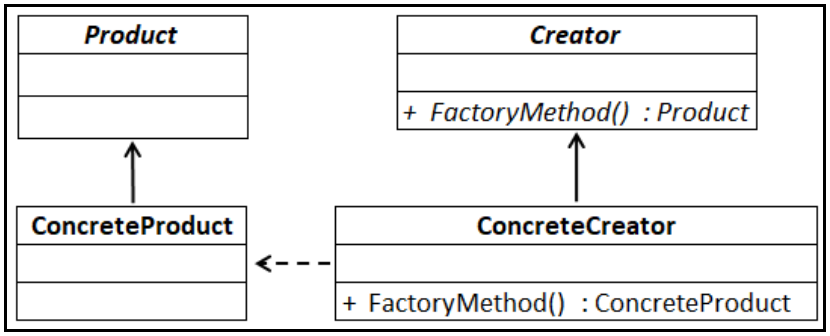
(Bild-Quelle https://www.philipphauer.de/study/se/design-pattern/factory-method.php)

1.1 UML

Einfach:



Mit Interface für Fabrik:



Hinweise:

- Die `Product`-Basisklasse ist meist abstrakt, das muss sie aber nicht unbedingt sein
- Oftmals hat die `FactoryMethod` auch Parameter, die angeben, welches Objekt erzeugt werden soll.
- Üblicherweise sind die Factory und die Produktklassen in einer eigenen **Library** zusammengefasst.

- Die verschiedenen **ConcreteProduct**-Klassen sind nicht public – der Konsument arbeitet ohnehin ausschließlich mit der Abstraktion.
- Manchmal wird die **Creator**-Klasse weggelassen und durch eine **statische Methode** in der **Product**-Klasse ersetzt

1.2 Erkennungsmerkmale

An folgenden Merkmalen kann man den Einsatz einer Factory erkennen:

- Man braucht ein bestimmtes Objekt von einer Produktfamilie, weiß aber zum Kompilierungszeitpunkt noch nicht genau welches. Die Entscheidung, welche konkrete Klasse benötigt wird, entscheidet erst der Client (also zur Laufzeit).
  Wenn man also unterschiedliche Untertypen mit **if**-Verzweigungen erzeugt, deutet das auf den erforderlichen Einsatz einer Factory hin:

```
string name = txtShape.Text;
BaseShape shape;
if (name == "Circle") shape = new Circle();
else if (name == "Rectangle") shape = new Rectangle();
else if (name == "Trapez") shape = new Trapez();
else
{
  MessageBox.Show($"Unknown type <${name}>");
  return;
}
//...
```

- Details der einzelnen Produkte sind nicht wesentlich, man kommt also im gesamten Projekt mit den Methoden der Produkt-Basisklasse aus.
- Es ist wahrscheinlich, dass im Projektverlauf noch oft zusätzlich konkrete Produktklassen hinzukommen werden.

1.3 Beispiele

In folgenden Beispielen würde sich eine Factory anbieten:

- Viele Frameworks basieren auf Factory Method
- abstrakte Basisklasse Database mit konkreten Klassen für SqliteDatabsae, SqlServerDatabase, …
- abstrakte Klasse Stream mit konkreten Klassen MemoryStream, FileStream, …

1.4 Vor-/Nachteile

1.4.1 Vorteile

- Code wird wiederverwendbarer
- Man wird gezwungen, mit Abstraktionen zu arbeiten, also mit konkreten Klassen.
- Kommen bei einer Applikation neue konkrete Produktklassen dazu, muss im Anwendercode nichts mehr geändert werden.
- Oft kann man alle möglichen Produktklassen von der Factory erfragen und damit Buttons oder Comboboxen in der App initialisieren.
- Der Anwender ist komplett von der Implementierung der konkreten Produkt-Klassen entkoppelt. Ändert sich die Product-Library, muss der Benutzer nicht einmal neu kompilieren.

1.4.2 Nachteile

- Code wird abstrakter
- Man braucht relativ viele Unterklassen.