Factory-Method / Virtual-Constructor

Problemstellung (Anwendungsgebiete)

- Objekterzeugung zur Laufzeit wenn zu erzeugender Typ davor zu Compile-Zeit nicht bekannt - weil zB dynamisch gebunden.
- Wenn Unterklassen Objekte bestimmen, die Oberklasse erzeugen soll.
- Bei kovarianten Problemen hilfreich.

Lösung (Implementierung, beliebig abstrakt)

Anwendungsbeispiel mit Implementierung : System zur Verwaltung von Dokumenten

Wie in NewDocManager ist der genaue Typ des zu erzeugenden Objekts zu Compile Zeit oft nicht bekannt. Ein einfaches new reicht nicht aus.

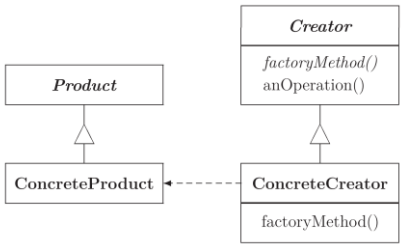
```
public abstract class Document { ... }          //Zu erstellende Objekte
public class Text extends Document { ... }
... // classes Picture, Video, ...


public abstract class DocCreator {
        protected abstract Document create();
}
public class TextCreator extends DocCreator {   //Zuständig für Text Dokumenten-Erstellung
        protected Document create() {
                return new Text();
        }
}
... // classes PictureCreator, VideoCreator, ...


public class NewDocManager {                    //Wählt einen DocCreator
        private DocCreator c = ...;
        public void set(DocCreator c) {         //Wählt Untertyp von DocCreator mit set()
                this.c = c;
        }
        public Document newDoc() {
                return c.create();              //Ruft create() im Untertyp von DocCreator
        }
}
```

Entwurfsstruktur-Diagramm

Die „factoryMethod" (create) retourniert ein ConcreteProduct. Alle Creator müssen das tun.
Die Methode „anOperation" kann von der abstrakten Klasse selbst als factoryMethod verwendet werden oder für sonstiges.
Es wird bei der Notation angenommen, dass jede solche Vererbungsbeziehung gleichzeitig auch eine Untertypbeziehung ist.



Alternative Implementierung

Die factoryMethod() kann abstrakt sein oder eine default Implementierung haben.

Wenn die zu erzeugenden Objekte noch *Konstruktorparameter* haben:

- Argumente an factoryMethod weiterleiten
- default Argumente an zentraler Stelle ablegen
- Parameter mitzugeben, die bestimmen, welche Art von ConcreteProdukt mit vordefinierten Konstruktorparametern erzeugt werden soll.

lazy initialization: Neues Objekt wird nur einmal in einer Unterklasse mit createProduct erzeugt und gespeichert. getProduct gibt bei jedem Aufruf *dasselbe* Objekt zurück. Sobald einmal dieses Objekt erstellt wurde wird kein neues mehr erstellt.

```
public abstract class Creator {
        private Product product = null;
        protected abstract Product createProduct();

        public Product getProduct() {
                if (product == null)
                        product = createProduct();
                return product;
        }
}
```

Konsequenzen (Vor- und Nachteile)

+ Erhöhte Flexibilität, Objekterzeugung zur Laufzeit
+ *Anknüpfungspunkte (Hooks nach Template-Method)* für Unterklassen die überschrieben werden - beidseitige Abhängigkeit.
+ Bei kovarianten Problemen hilfreich: Beispielsweise erzeugt eine Methode generiereFutter in der Klasse Tier, basierend auf das aufrufende Tier das richtige Futter in Form eines Objektes.
  Klassenhierarchien Tier und Futter. Untertyp von Futter wird in Untertyp von Tier erzeugt.

- *parallele Klassenhierarchien*: die Creator-Hierarchie mit der Product-Hierarchie.
  Viele Unterklassen von „Creator“ zu erzeugen, die nur new mit einem bestimmten „ConcreteProduct“ aufrufen.
  In Java gibt es (abgesehen von aufwändiger Reflexion) keine Möglichkeit die vielen Klassen zu vermeiden.
  Es geht aber mit Templates in C++.