LEHR- UND FORSCHUNGSEINHEIT
PROGRAMMIERUNG &
SOFTWARETECHNIK

Softwaretechnik 2015/2016

PST Lehrstuhl

Prof. Dr. Matthias Hölzl

Joschka Rinke

AGENDA

Übung 10:
17.12.2015

- **Fragen**
- **Besprechung Blatt09**

Aufgabe 1 – Java Reflections

Reflection:
Code, der im gleichen (laufenden) System die Struktur untersuchen und sogar ändern kann.

Anwendung in Java:
Programme, die das Verhalten/ Eigenschaften von Applikationen in der JVM zur Laufzeit untersuchen müssen.

Mögliche konkrete Beispiele:

- **Class Browser: z.B. Aufzählen der Instanzen einer Klasse**
- **Visual Development Environment: z.B. Verwendung von Typinformationen**
- **Debugger: z.B. Zugriff auf „private“-Attribute**
- **…**

Aufgabe 1 – Java Reflections

Möglicherweise kritische Punkte bei Java Reflections:

- **Performance: JVM kann eventuell nicht perfekt optimiert werden**
- **Security: Möglicherweise fehlen RuntimePermissions,**
- **Exposure: Zugriff auf private Fields**

Aufgabe 1 – Java Reflections

Die Class-Klasse:
Instanzen der Klasse Class repräsentieren Klassen/ Interfaces einer laufenden Java Anwendung

- ➜ **kein öffentlicher Konstruktor verfügbar**
- ➜ **Objekte werden automatisch durch die JVM erzeugt, wenn Klassen geladen werden**
- ➜ **Class ist der Eintrittspunkt für alle Reflection APIs in Java… …und repräsentiert das Meta-Level**

Aufgabe 1 – Java Reflections

Es ist keine Intercession möglich!

- ➜ **Der vermeintlich weitreichendste Eingriff ist mit setAccessible(true) möglich, indem Zugriffschecks auf ein Klassen-Member unterdrückt werden**
- ➜ **In erster Linie kann man also Eigenschaften und Methoden einer Klasse erfragen**

Aufgabe 1 – Java Reflections

socrative by MasteryConnect

Wahr oder falsch:
Mit Java Reflections ist es möglich einen Konstruktor,
der die Sichtbarkeit private hat, außerhalb der Klasse aufzurufen.

```
Class c = Class.forName(„package.Classname“);
Constructor[] constructors = c.getDeclaredConstructors();
constructors[0].setAccessible(true);
// call with: constructors[0].newInstance();
```

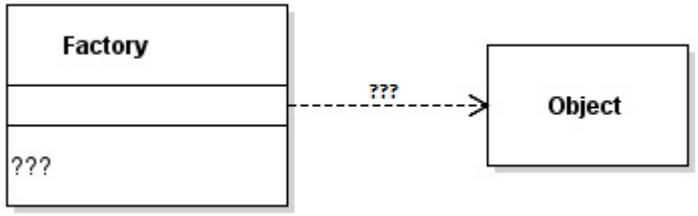
Aufgabe 2 – Creational Patterns

Creational Patterns in „Design Patterns“:

- **Abstract Factory: Interface für die Erstellung von Objekten einer Objektfamilie**
- **Builder: Abtrennung der Erzeugung komplexer Objekte von ihrer Repräsentation**
- **Factory Method: Methode für die Erzeugung von Objekten**
- **Prototype: Erzeugung neuer Objekte indem ein Prototypobjekt geklont wird**
- **Singleton: Eine Instanz und ein globaler Zugangspunkt**

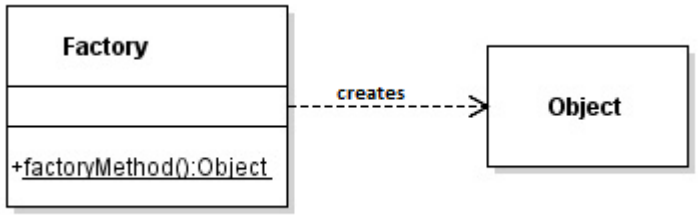
Aufgabe 2 – Creational Patterns

Factory: ???



Aufgabe 2 – Creational Patterns

Factory: Objekt zur Erzeugung anderer Objekte



Aufgabe 2 – Creational Patterns

Factory die ein Object erzeugt:

```
public class ObjectFactory {
        public static Object getObject() {
                return new Object();
        }
}
```

Aufgabe 2 – Creational Patterns

socrative by MasteryConnect

Vorteil von Factories sind, dass…

- **…sie einen Single Point of Access für die Objekterzeugung bieten.**
- **…der Code leichter zu lesen ist, da die Erzeugung in der Factory stattfindet.**
- **…es gibt keine, Factories sind ein Anti-Pattern.**
- **…sie immer die Performance erhöhen, da die Instantiierung neuer Objekte in die Fabrik ausgelagert wird.**

Bei Änderungen muss nur die Factory-Method geändert werden;

→ Single Access Point für Objekterzeugung

Aufgabe 2 – Creational Patterns

Factories werden unter anderem häufig verwendet bei der Umsetzung von:

- **Singleton → Factory-Method gibt eine einzige Instanz einer Klasse zurück**
- **Builder → eine Factory-Method kann verwendet werden um zu implementieren, welche Komponenten gebaut werden**
- **Factory-Method → meist abstrakte Factory-Methode, die in der konkreten Factory implementiert werden muss**
- **Abstract Factory → mehrere Factory-Methods**

Aufgabe 2 – Creational Patterns

socrative by MasteryConnect

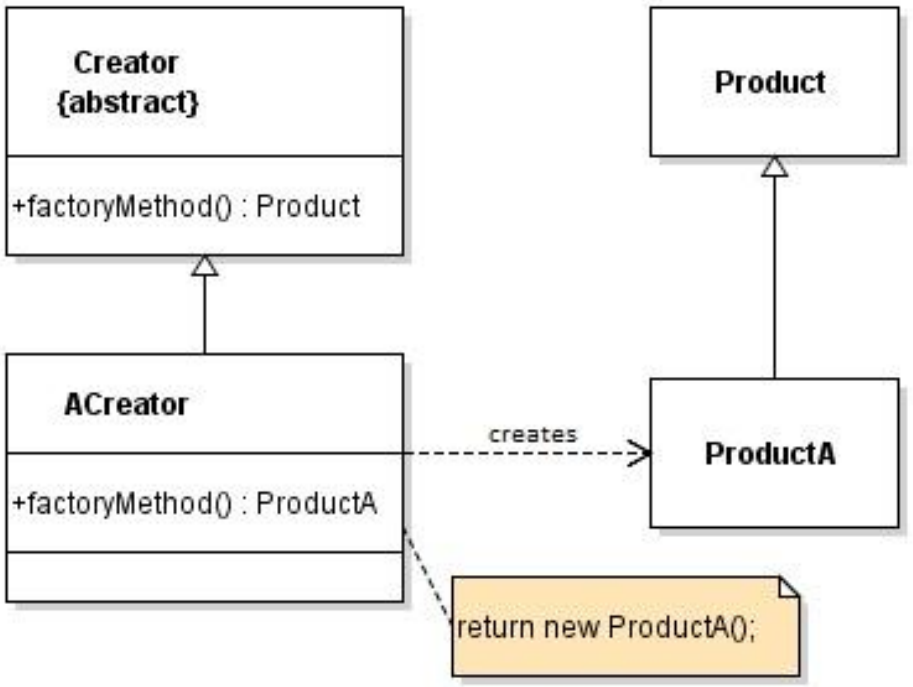
Sind Factories also immer eine Umsetzung des Factory-Method Pattern?

➜ **Das ist ein wenig Definitionssache…**

- **In Factories wird die Factory-Method meist als statisch angegeben, das ist aber nicht mehr möglich, wenn sie wie im Factory-Method Pattern als abstrakte Methode überschrieben werden muss**
- **Auf der anderen Seite haben Factories eben gerade eine Factory-Method, die andere Objekte erzeugt**

➜ **Für uns gilt: die Factory-Method in einer Factory erfüllt das Factory-Method Pattern!**

Aufgabe 2 – Creational Patterns



Behavioral Pattern: Template Method
Factory Method ist für die Erzeugung von Objekten,
was Template Method für die Implementierung von Algorithmen ist!

Aufgabe 2 – Creational Patterns

Unterschied zwischen Factory Method und Abstract Factory:

- **Abstract Method ist eine einzelne Methode, die überschrieben wird**
- **Abstract Factory ist ein Objekt für die Erzeugung anderer Objekte, das mehrere Template Methods enthält**

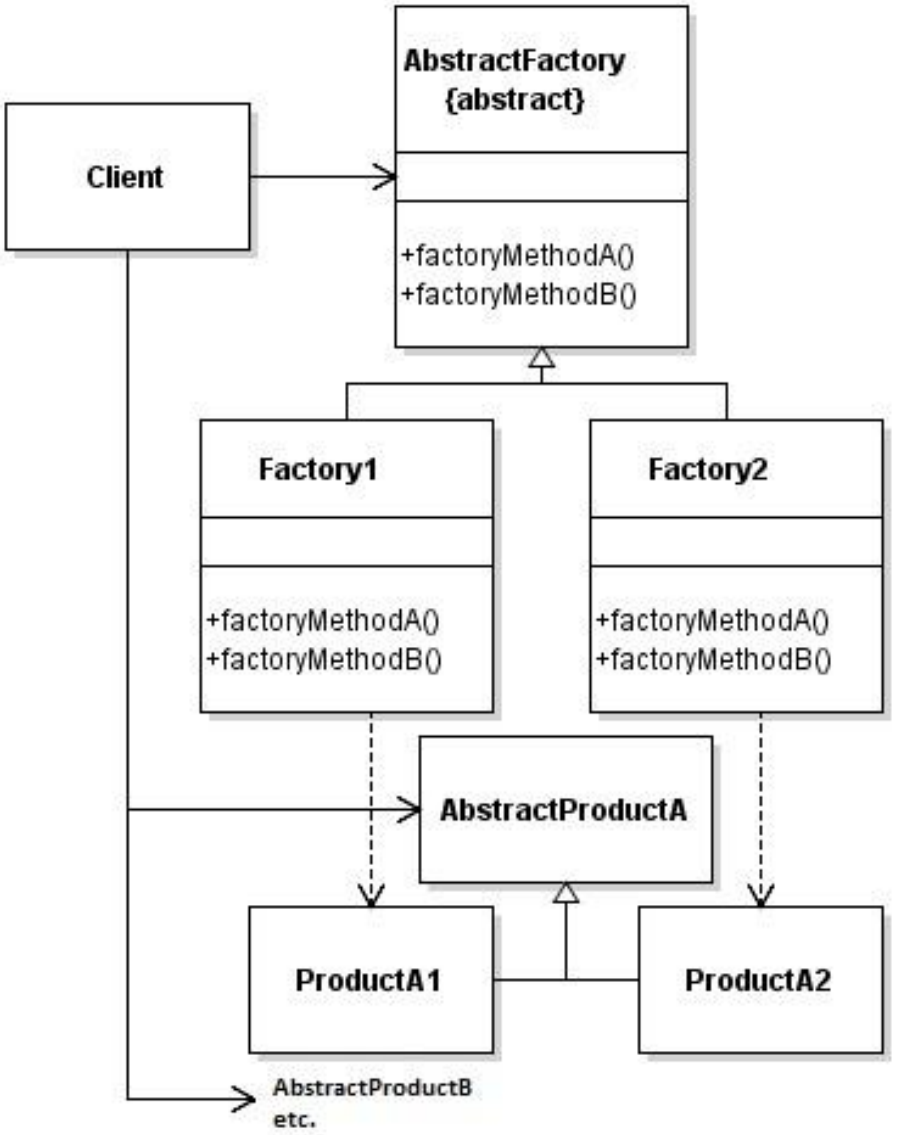
➔ **Schnittstelle zur Objektinstantiierung (ganzer Objekt Familien)**

➔ **konkrete Klasse der zu instantiierenden Objekte nicht näher festgelegt**

Aufgabe 2 – Creational Patterns

Abstract Factory:

- **Client isoliert von konkreten Klassen**
- **Austausch ist einfach**



Aufgabe 2 – Creational Patterns

socrative by MasteryConnect

Ein Nachteil der Abstract Factory ist z.B.:

- **Es gibt keine erkennbaren Nachteile**
- **Die Architektur ist sehr kompliziert und äußerst unübersichtlich**
- **Neue Produkte lassen sich nur schwer hinzufügen**
- **Es werden abstrakte Klassen verwendet**