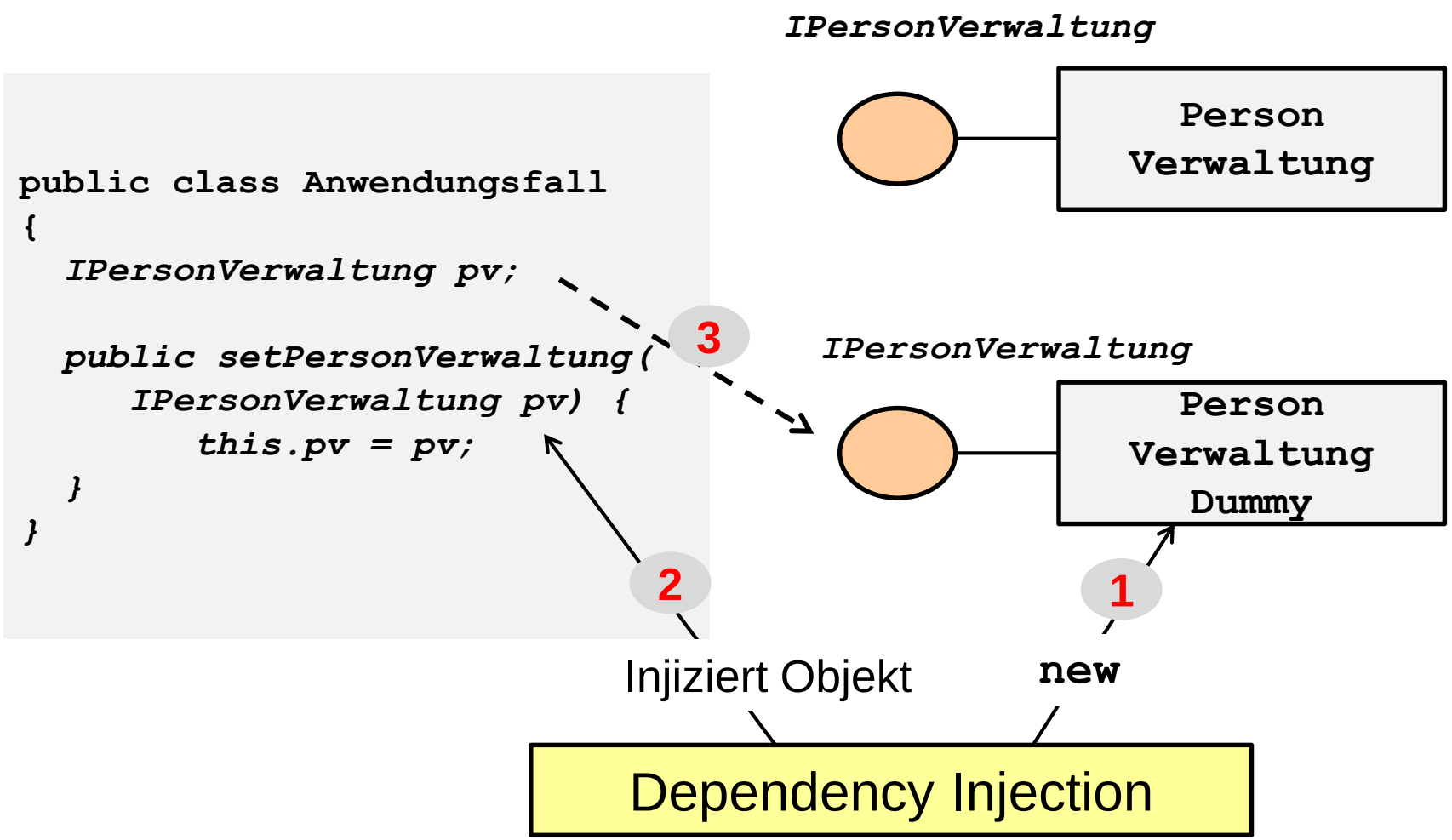
Hochschule **Rosenheim**
University of Applied Sciences

# Verteilte Verarbeitung

## Dependency Injection und das Interceptor Pattern

# Dependency Injection



# Dependency Injection Voraussetzungen

1. Anwendungsfall ist gegen ein Interface (IPersonVerwaltung) programmiert
2. Objekt, das IPersonVerwaltung impl. kann von außen gesetzt werden!

```
public class Anwendungsfall {
  private IPersonVerwaltung
        pv = null;

  public Person findPerson(Long id) {
        return pv.findById(id);
  }

  public void setPersonVerwaltung(IPersonVerwaltung pv) {
     this.pv = pv;
  }
}
```

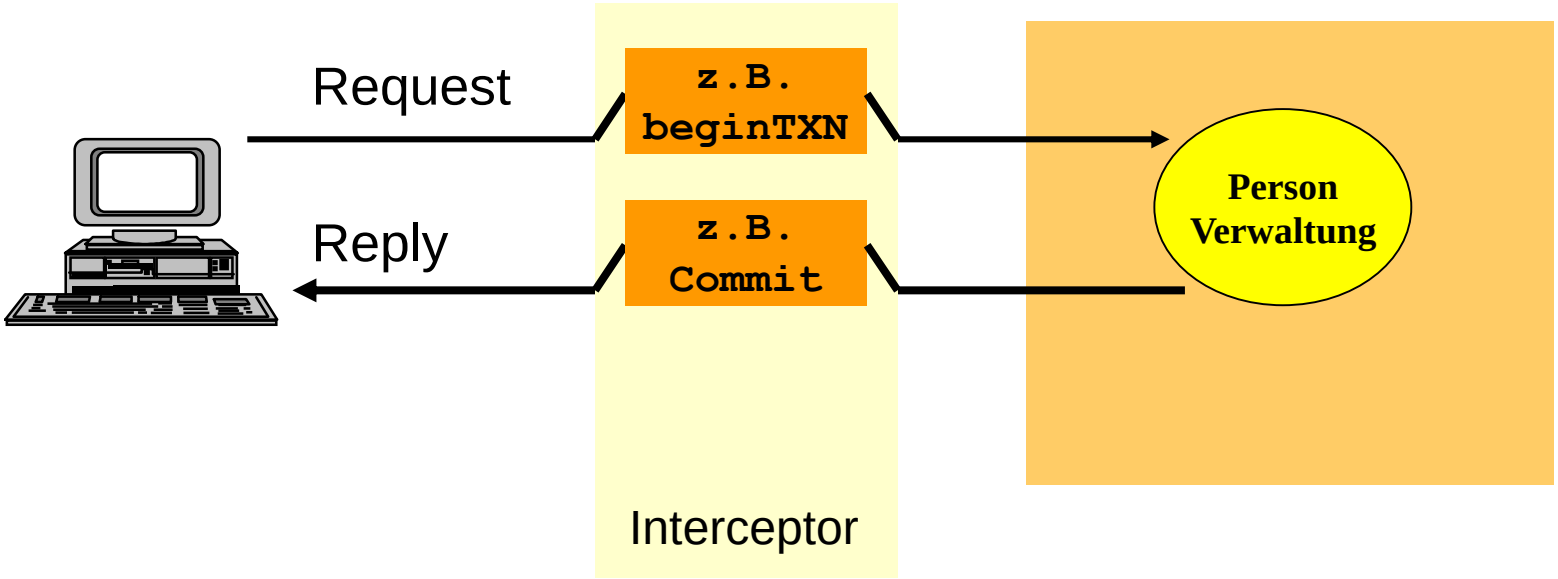
Wichtig: Klasse sollte sich an JavaBeans Konventionen halten

- Default Konstruktor
- Properties (Attribut und gleichnamige get/set Methonden)

# Hinweis für Java EE 6 und 7

- CDI = JSR 330 und JSR 299 ab Java EE 6 enthalten (Context and Dependency Injection)
- Neue Annotationen
  - **`@Inject`** = Abhängigkeit wird vom Container aufgelöst
  - **`@RequestScoped, @SessionScoped, @ApplicationScoped`** = Container erstellt / zerstört Beans innerhalb eines Requests / der Session / der Applikation …
- Enthalten in neueren Containern
  - Referenzimplementierung JSR 330: GUICE https://code.google.com/p/google-guice/
  - Referenzimplementierung JSR 299: WELD 2.0 http://docs.jboss.org/weld/reference/1.0.0/en-US/html/
- Spring
  - **`@Autowired`** = Abhängigkeit wird vom Container aufgelöst
  - **`@Bean, @Component`**, = Vom Container verwaltete Instanzen (**`@Service, @Repository, @Controller`**)

# Das Interceptor Pattern



# Komponenten im Container
# Wie kommunizieren beide?

- Interceptor Pattern
  - Aufruf vom Client an Server wird „unterbrochen“
  - = Innerhalb des Aufruf: Zusatzfunktionen
    - Logging, Transaktionssteuerung, Security, …
  - Vorteile:
    - Transaktionslogik, Security, … in der Implementierung der Komponente nicht sichtbar (A/T-Trennung)
    - Verhalten Interceptor Konfigurierbar
- Möglichkeiten der Umsetzung
  - Code Generator + Generierungsvorschrift
  - Aspektorientierte Programmierung - AOP
  - Interceptoren + Dynamische Proxies in Java

# Triviale Umsetzung des Interceptors als Decorator

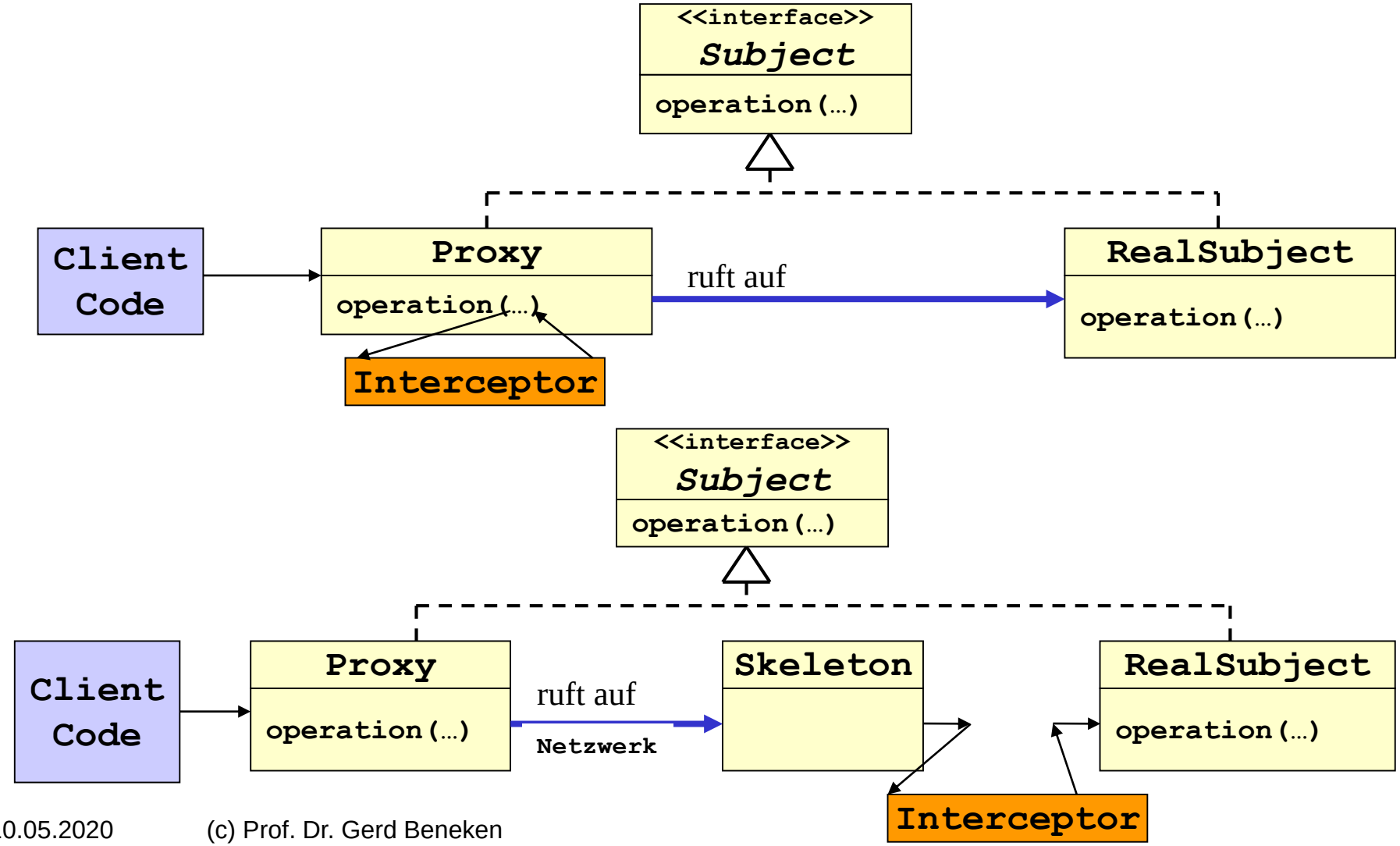
```
public class AnwendungsfallTXNDecorator
                      implements IPersonenVerwaltung {
  IPersonenVerwaltung personenVerwaltung;

  public AnwendungsfallTXNDecorator(
                       IPersonenVerwaltung personenVerwaltung) {
    this.personenVerwaltung = personenVerwaltung;
  }

  @Override
  public Person createPerson(String vorname, String nachname) {
    System.out.println("Begin Transaction");
    Person result =
          personenVerwaltung.createPerson(vorname,nachname);
    System.out.println("Commit Transaction");

    return result;
  }
```

# Einfacher Interceptor (als Proxy interpretierbar) und Relistische Umsetzung



# Umsetzung Interceptor Pattern: Interface `InvocationHandler`

```
interface
  java.lang.reflect.InvocationHandler {
    Object invoke(
                Object proxy,
                Method method,
                Object[] args)
            throws Throwable;
}
```

# Umsetzung Interceptor Pattern: Beispiel **`Interceptor`** für Tracing

```
public class Interceptor implements InvocationHandler {
    private Object delegate;

    public Interceptor(Object delegate) {
       this.delegate = delegate;
    }

    public Object invoke(Object proxy, Method method,
                         Object[] args) throws Throwable {
        try {
            System.out.println("Enter: " + method.getName());
            return method.invoke(delegate, args);
        } catch (Exception e) {
            System.out.println("Exception: " + e);
            throw e;
        } finally {
            System.out.println("Leave: " + method.getName());
        }
}}
```

# Dynamische Proxies
# java.lang.reflect.Proxy

```
public class Proxy implements java.io.Serializable {
   public static Class<?> getProxyClass(
               ClassLoader loader,
               Class<?>... interfaces)
       throws IllegalArgumentException
   public static Object newProxyInstance(
               ClassLoader loader,
               Class<?>[] interfaces,
               InvocationHandler h)
       throws IllegalArgumentException
   public static boolean isProxyClass(Class<?> cl);
   public static InvocationHandler
               getInvocationHandler(Object proxy)
       throws IllegalArgumentException;
```

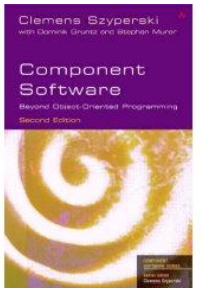
# Installation des Interceptors Am Beispiel eines **SumService**

```
public class InterceptedList {
    public static void main(String[] args) {
        List<String> strings = new ArrayList<String>();
        List<String> intercepted =
                  (List<String>) getInterceptor(strings, List.class);
        intercepted.add("Rosenheim ist Super");
        System.out.println(intercepted.get(0));
}

    public static Object getInterceptor(Object obj, Class clazz) {
        Object service = Proxy.newProxyInstance(
        clazz.getClassLoader(),
                  new Class[] { clazz },
                  new Interceptor(obj));
         return service;
     }
}
```

# Weiterführende Literatur

- Szyperski, Gruntz, Maurer: Component Software: Beyond Object-Oriented Programming, Addison Wesley, 2002




- Hanmer: Patterns for Fault Tolerant Software, Wiley, 2007


- Völter, Schmid, Wolff: Server Component Patterns, Wiley, 2002