ARS

Member of TIMETOACT GROUP SOFTWARE & CONSULTING

Contexts and Dependency Injection - Erweiterte Themen

Java EE Grundlagen

ARS

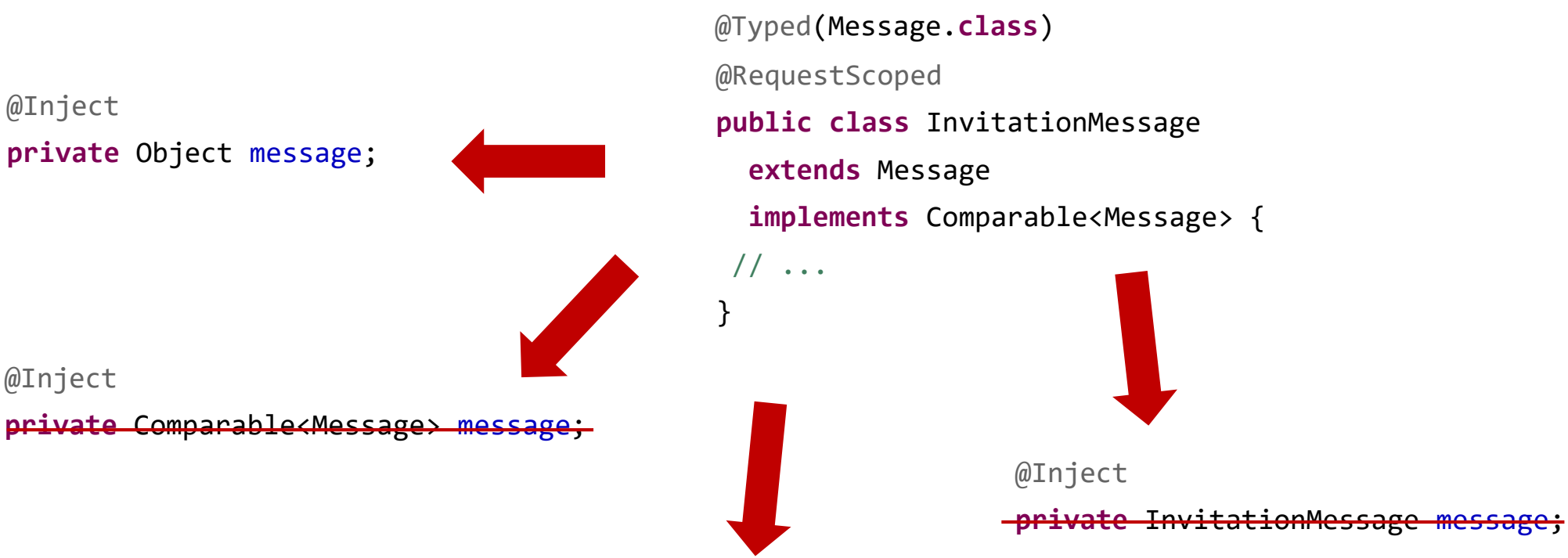
Inhalte dieses Kapitels

Contexts and Dependency Injection

CDI & Java EE

ARS

Bean Types



ARS

Interceptors [1|3]

Überblick

- Möglichkeit der Implementierung von *Cross-Cutting Concerns*
- Teil der EJB-Spezifikation (Package `javax.interceptor`)
- Verwendung in CDI möglich über Annotationen

```
@Inherited
@InterceptorBinding
@Target({ElementType.TYPE, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface Secure {}
```

```
@SessionScoped
public class Account {

  // ...

  @Secure
  public void save() {
    // ...
  }

}
```

ARS

Interceptors [2|3]

Implementierung

- Klasse mit Annotationen

```
@Secure
@Interceptor
public class SecurityInterceptor {

  @AroundInvoke
  public Object authorize(InvocationContext ic) throws Exception {
    // check security (user logged in?)
    return ic.proceed(); // invoke original business logic
  }

}
```

ARS

Interceptors [3|3]

Implementierung

- Deklaration in Datei `beans.xml`
  - Reihenfolge (wenn mehrere Interceptors)

```
<beans
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                      http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
  <interceptors>
    <class>de.ars.library.interceptors.SecurityInterceptor</class>
  </interceptors>
</beans>
```

- Alternative: Angabe einer Priority

```
@Priority(Interceptor.Priority.APPLICATION)
public class LoggingInterceptor { … }
```

ARS

Decorators

- Analogie zu *Interceptors*
  - Keine *Cross-Cutting Concerns*, sondern Erweiterung der Logik
  - „Wrapper“ für Original-Beans
  - Deklaration in `beans.xml` erforderlich

```
@Decorator
public class TimeStampedMessage extends Message {

  @Inject @Delegate
  private Message delegate;

  @Override
  public String getMessage() {
    return delegate.getMessage() + "(at " + new Date() + ")";
  }

}
```

ARS

Event Handling

- Mechanismus, Events auszulösen und zu bearbeiten
- Event = POJO

```
@Inject
private Event<Message> messageEvent;

public void doSomething() {
  messageEvent.fire(new Message());
}
```

```
public void processMessage(@Observes Message message) {
  // Event Handling
}
```

ARS

CDI & Java EE [1|4]

- Verwendung in allen Java-EE-Komponenten
  - Servlets, JSF, Webservices, EJBs, JMS
- Integration der Technologien über CDI, z.B. JSR 303

```
@WebServlet(urlPatterns = { "/login" })
public class LoginServlet extends HttpServlet {

  @Inject
  private Validator validator;

  // ...

}
```

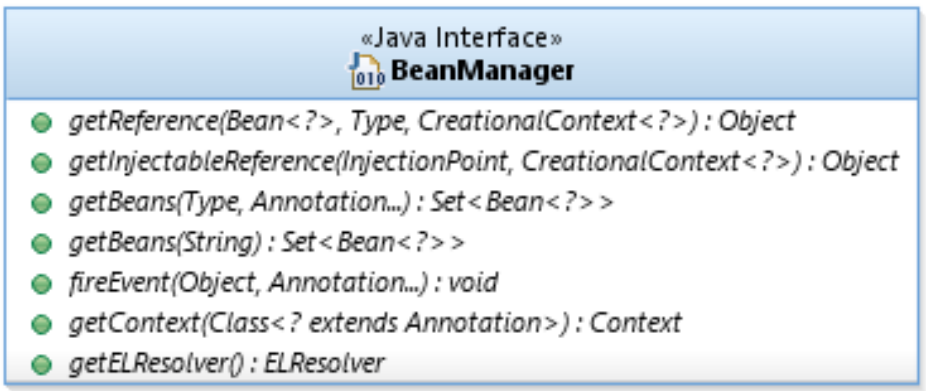
CDI & OSGi

- https://www.osgi.org/bugzilla/show_bug.cgi?id=141

CDI & Java EE [2|4]

CDI & JNDI

- CDI-Container stellt Beans in JNDI zur Verfügung
  - `@javax.annotation.ManagedBean("name")` → `java:app/<module>/<name>`
  - Bean Manager → `java:comp/BeanManager`



CDI & Expression Language

- CDI-Beans per EL verwendbar
  - `@Named("beanname")` → `${beanname}` bzw. `#{beanname}`

ARS

CDI & Java EE [3|4]

CDI vs. EJB

- Verwendung von CDI bevorzugt
  - Einfach
  - Verwendbar in allen Java-EE-Containern
  - Scoping
  - Pluggability
- Verwendung von EJBs
  - Notwendigkeit der EJB-Dienste
    - Transaktionalität, Security, Asynchronität, Thread-Pooling, Remoting, …
  - Äußere Layer (Aufrufschnittstellen der Businesslogik)

ARS

CDI & Java EE [4|4]

CDI & Servlet Pluggability

- 100% nutzbar
  - Verwendung von `@Inject` in Webmodulen
  - Bean Producers, Interceptors, Decorators in Fragmenten
- Integration von Frameworks über CDI möglich

CDI & Spring

- Nicht Out-Of-The-Box, da Spring kein Java EE-Standard
- Siehe https://www.openknowledge.de/blog/article/integration-von-spring-in-cdi-ueber-eine-cdi-extension-erster-teil.html