Enterprise Web Development (Java Enterprise Edition 6)

wissen.leben
WWU Münster

Dozent: Arne Scheffer

Enterprise Java Beans - Einführung

- Business-Logik-Beans mit zusätzlichen Funktionalitäten
  - Transaktionssicherheit
    - Container kümmert sich um die korrekte Ausführung verketteter Aktionen
      - Vermeidung von Race-Conditions, Thread-Safety
      - evtl. Performance-Einbußen bei rigiden Lock-Policies
  - Verwendungmöglichkeit per R(emote)M(ethod)I(nvocation)
    - via Remote-Interface
    - andernfalls war bisher lokales Interface nötig (J2EE), jetzt nicht mehr
  - Container kümmert sich um orthogonale Belange
    - Interceptor-Konzept → unser heutiges Thema
    - Security-Features

Enterprise Java Beans - Varianten

- Varianten:
  - Stateful: Scoped wie JSF-Klassen,
    - z.B. *SessionScoped*:
      - Variablen-Lebenszyklus ist hier eine HTTP-Session
      - Vorhaltung der Session im Browser-Cookie
        → pro Client einen Variablensatz / eine Stateful-Bean-Instanz
  - Stateless
    - „zustandslos“ → ohne Variablengedächtnis
    - bessere Skalierbarkeit
      - Container braucht i.A. nicht für jeden Client eine Bean-Instanz
  - Singleton
    - nur eine Instanz für die gesamte Anwendung
  - Message Driven
    - asynchron getriggert über Message-Queues

Ausgliederung eines EJB *LoginFacade* aus *LoginController*

- Rekapitulation: Warum lohnt sich eine Trennung
  - Kapselung von
    - Backing Bean (JSF-Logik)
    - EJB (Businesslogik)
  - saubere Trennung zwischen View-Concerns und Businesslogik
  - aber: erhöht die Komplexität
    - Klassen mit @Named könnten auch direkt die Persistenz verwenden
  - Verwendung der @Stateful-Annotation fasst die Businesslogik in ein EJB
    - wir können das Interceptor-Konzept verwenden, unser heutiges Thema
  - die Kapselung ermöglicht es uns, zu Demo- und Testzwecken
    - viewseitig das Editieren zu erlauben
    - businesslogikseitig das Editieren zu sperren
  - wir simulieren in unserem Beispiel eine faked-form-Attacke

    → damit demonstrieren wir die Sinnhaftigkeit des Interceptor-Konzepts

Einbau einer Login-Kontrolle per Interceptor

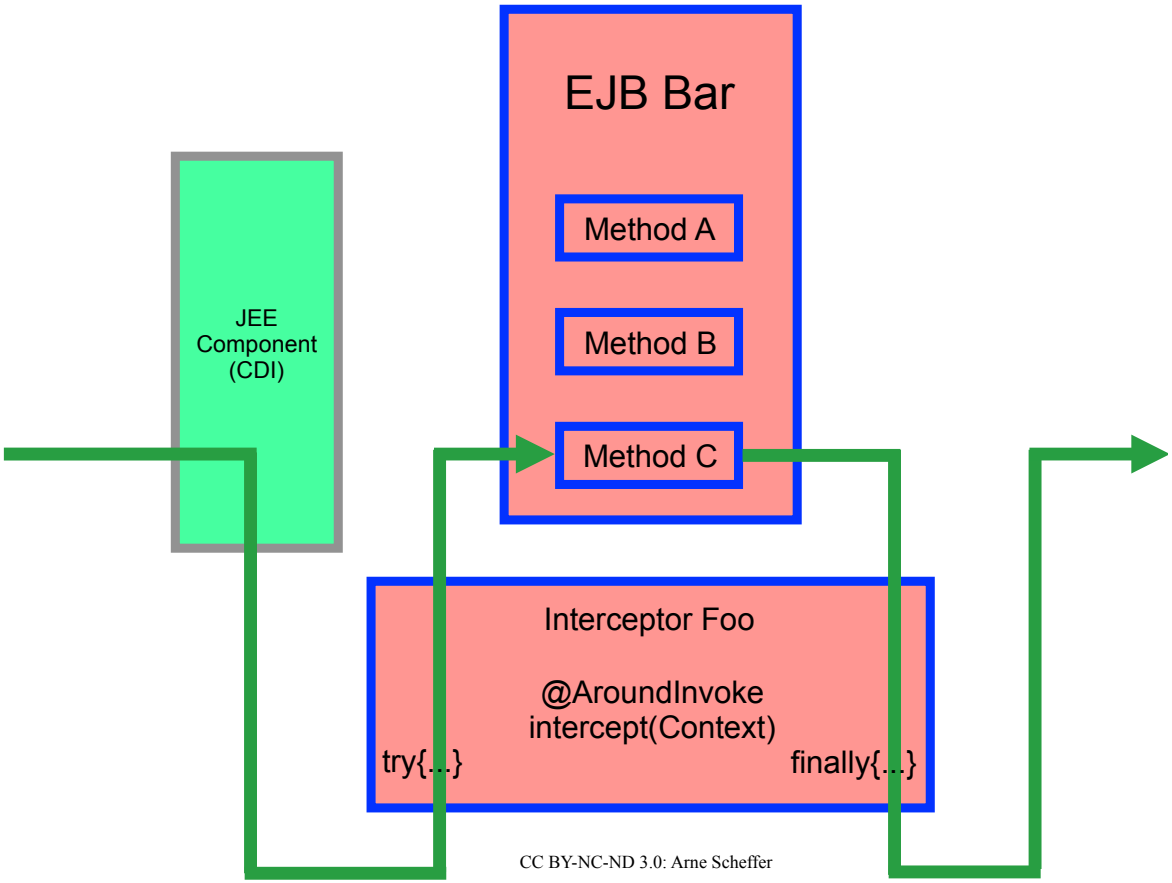
- Warum ?
  - orthogonale Belange („crosscutting concerns“)
    - Persistenzlogik ↔ Login-Kontroll-Logik
  - DRY-Verletzung
    - Persistenz-Methoden gesprenkelt mit Code der Form
      - if (login == false) react accordingly;

- besser dem Container mitteilen
  - das für alle Methoden, die persistieren
    - vorher ein Check (allgemeiner: Code) durchzuführen ist

  → wir fangen Persistenz-Methodenaufrufe mit einem Interceptor ab
    - und machen darin unseren Login-Check

- Loose Coupling über Container-Hinweise (Annotationen / XML)

Interceptor-Konzept



Vorgehen: Verwendung eines Interceptors

- mit einer selbstkreierten Annotation wird der Interceptor angebunden an
  - Methoden, Klassen, Annotationen
- Vorgehen zur Verwendung eines Interceptors
  - (Paket login.interception erzeugen)
  - InterceptorBinding (= Interceptor-Annotation) kreieren: *LoginInterception.java*

```
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.TYPE;
@Inherited
@InterceptorBinding
@Retention(RetentionPolicy.RUNTIME)
@Target({METHOD, TYPE})
public @interface LoginInterception {}
```

  - Interceptor schreiben und annotieren mit dieser Annotation und *@Interceptor*
  - Interceptor-Klasse in *beans.xml* aktivieren
  - abzufangende Methoden etc. annotieren

Unser Interceptor: *LoginInterceptor.java*

```
@LoginInterception @Interceptor
public class LoginInterceptor {
 @Inject AuthenticationFacadeIF loginfacade;

 @AroundInvoke
 public Object intercept(InvocationContext context) throws Exception
 {
    System.out.println("Invoking method: " + context.getMethod());
    if (loginfacade.isAuthenticated()) return context.proceed();
     else throw new LoginException();
 }
}

class LoginException extends Exception {
 public LoginException() {
  super("Possible faked form attack; try to reach persistence layer without login detected.");
 }
}
```

Aktivierung, abzufangende Klassen annotieren

- Interceptor-Klasse in WEB-INF in der Dateien *beans.xml* aktivieren:
  - stellt bei mehreren class-Einträgen die Durchführungsreihenfolge sicher

```
<?xml version="1.0" encoding="UTF-8"?>
<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>login.interception.LoginInterceptor</class>
 </interceptors>
</beans>
```

- abzufangende Methoden annotieren in *LinkFacade.java* :

```
@LoginInterception @Override
public void create(Link link) { super.create(link); }
@LoginInterception @Override
public void edit(Link link)   { super.edit(link); }
@LoginInterception @Override
public void remove(Link link) { super.remove(link); }
```

Demo (Simulation eines Programmierfehlers), Benefits

- Demo
  - einfach mal in *LoginController.java*
    - *isEditable()* auf *true* setzen
  - auch nicht Angemeldete dürfen nun jeden Knopf sehen und benutzen
  - die *LoginInterception* in *LinkFacade.java*
    - sollte Änderungsversuche an den Daten nun abfangen

- Benefits
  - othogonale Belange bleiben getrennt
  - steckbar: einfach in *beans.xml* aktivieren und deaktivieren
    - kein neuer Build notwendig
  - Einhakpunkte auch gut nutzbar für Statistiken und Debug-Logs

Erweiterung: Kapselung durch eigene Annotation im EJB

- InterceptorBinding *PersistInterception.java* im neuen Paket *InterceptorBindings* erzeugen

```
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.TYPE;
@Inherited
@LoginInterception // stellt die Verbindung sicher
@InterceptorBinding
@Retention(RetentionPolicy.RUNTIME)
@Target({METHOD, TYPE})
public @interface PersistInterception {}
```

- abzufangende Methoden neu annotieren in *LinkFacade.java* :

```
@PersistInterception @Override
public void create(Link link)   { super.create(link); }
@PersistInterception  @Override
public void edit(Link link)       { super.edit(link); }
@PersistInterception @Override
public void remove(Link link) { super.remove(link); }
```

Java Authentication and Authorization Service (JAAS)

- Am Anfang war das **Subject**....
  - Das Subject repräsentiert die Quelle eines Requests
  - Allgemeines Vorgehen
    1.) Authentifizierung des Subjects
      - in unserem Fall Authentifizierung eines Users

    2.) Zuweisung von Repräsentationen dieses Users - sogenannter **Principals**
      - in unserem Fall z.B.
        - Userkennung
        - Vor- und Nachname
        - Matrikelnummer
      - in manchen Fällen auch
        - Rollen (etwas anderer Fall, da man sich diese meist teilt)

    3.) Autorisierung einfordern, und den Aktionsradius des Subjects einschränken
      - Subject.**doAS**, Subject.**doAsPrivileged** (Angabe des AccessControlContextes)

MyLoginModule.java (noch ohne login-Methode)

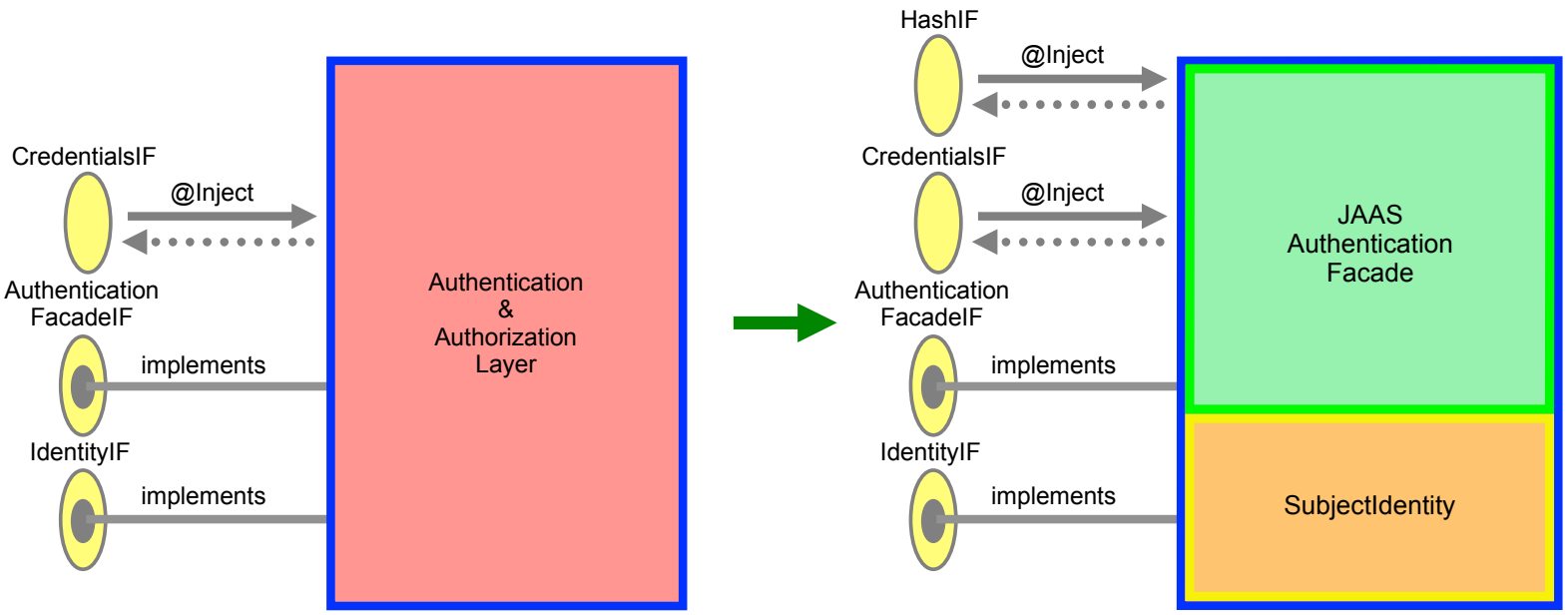
```
public class MyLoginModule implements LoginModule, Serializable {
  private Subject subject; private CallbackHandler callbackHandler;
  private Map<String, ?> sharedState; private Map<String, ?> options;
  private boolean commitSucceeded = false; private boolean loginSucceeded = false;
  private String username; private String roleName; private String fullName;

  @Override public void initialize(Subject subject, CallbackHandler callbackHandler,
                                   Map<String, ?> sharedState, Map<String, ?> options) {
    this.subject = subject;                this.callbackHandler = callbackHandler;
    this.sharedState = sharedState; this.options = options; }
  @Override public boolean login() throws LoginException { /* … */ }
  @Override public boolean abort() throws LoginException { return true; }
  @Override public boolean logout() throws LoginException { return true; }
  @Override public boolean commit() throws LoginException {
    subject.getPrincipals().add(new UsernamePrincipal(username));
    subject.getPrincipals().add(new FullNamePrincipal(fullName));
    subject.getPrincipals().add(new RoleNamePrincipal(roleName));
    commitSucceeded = true; return true; } }
```

SubjectIdentity.java (kapselt Subject konform zu IdentityIF)

```
public class SubjectIdentity implements IdentityIF{
   private Subject subject=null;
   public SubjectIdentity(Subject subject){ this.subject=subject;}
   @Override public String getFullName() { if (subject!=null){
    for (Principal p: subject.getPrincipals()){ if (p instanceof FullNamePrincipal) return p.getName();  }}
    return "not set"; }
   @Override public String getUsername() { if (subject!=null){
    for (Principal p: subject.getPrincipals()){ if (p instanceof UsernamePrincipal) return p.getName();  } }
    return "not set"; }
   @Override public boolean isAuthorized(String permission) {
    boolean result=false; final SecurityManager sm; final String perm=permission;
    if (System.getSecurityManager() == null) { sm = new SecurityManager(); }
     else                                    { sm = System.getSecurityManager();}
    try { Subject.doAsPrivileged(subject, new PrivilegedExceptionAction() {
          @Override public Object run() { sm.checkPermission(new MyPermission(perm)); return null; }}, null);
         result = true; } catch (AccessControlException ace) { /*...*/ result = false; }
                        catch (PrivilegedActionException pae)       { result = false; }
    return result;                                                  }
```

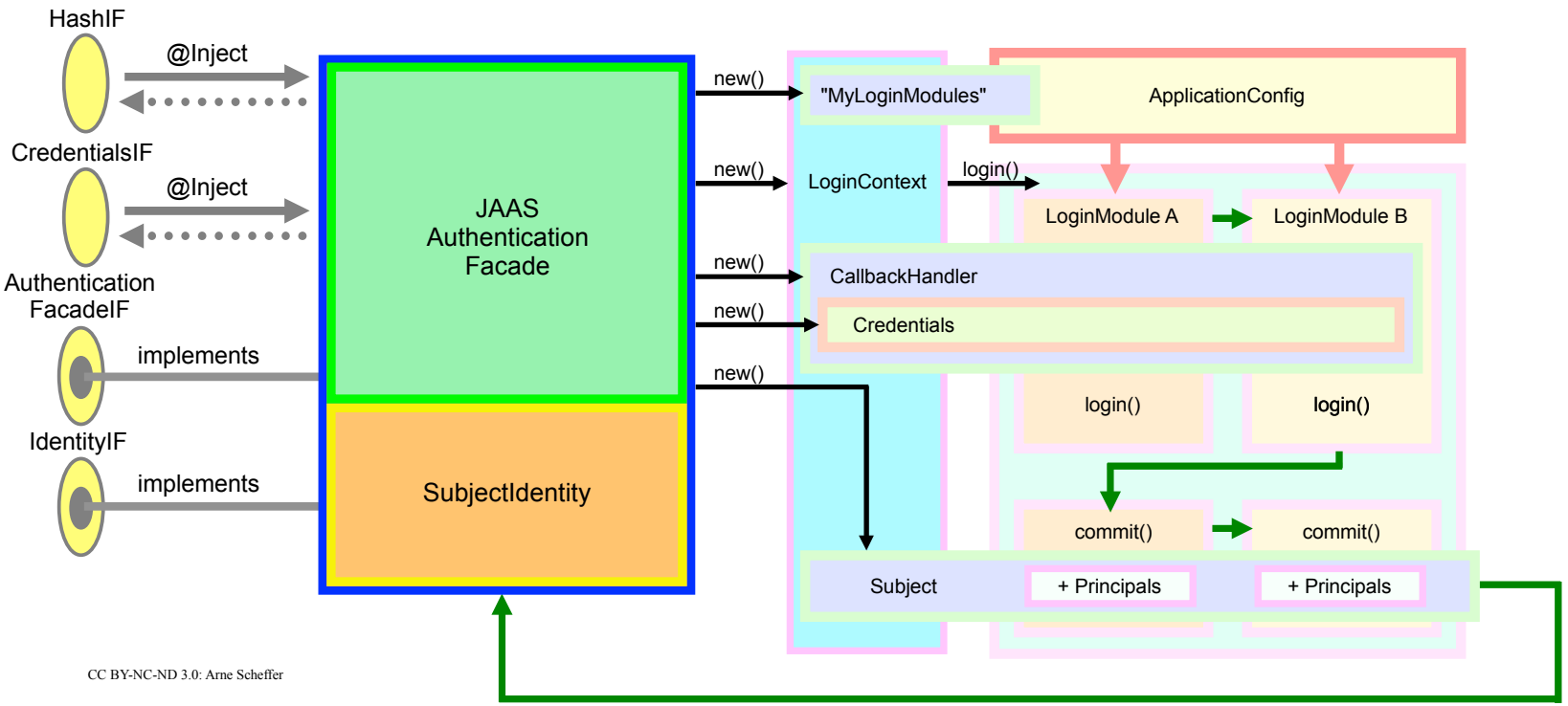
Umzusetzende Struktur: gewohntes Vorgehen



Login-Vorgang bei JAAS (vereinfacht)

- Client-Applikationscode
  - Bereitstellung eines Login-Callback-Handlers
    - gibt auf Anforderung des/der Login-Moduls/-Module Informationen raus
      - Credentials, Zertifikate, Schlüssel, etc.
      - über die Methode *handle()*
  - Durchführung des Login-Vorganges
    - Erzeugung einer *LoginContext*-Instanz
      - Mitgabe des Callback-Handlers, eines neuen Subjects, der Konfiguration
    - Aufrufen der Methode *login* der *LoginContext*-Instanz
    - Nutzung des in der Folge erweiterten Subjects
      - Verwendung der zugewiesenen Principals
      - Verwendung der *Subject.doAs*...-Methoden,
        - um Aktionen mit den Rechten von Subject durchzuführen
  - Eintrag dafür in *server.policy*: grant Principal jaas.RoleNamePrincipal "admin"
    {permission jaas.MyPermission "write";};

JAAS-Implementierung im Detail (vereinfacht)



MyLoginModule → Methode login

```
@Override public boolean login() throws LoginException {
  NameCallback nameCallback = new NameCallback("Username");
  FullNameCallback fullNameCallback = new FullNameCallback("FullName");
  RoleNameCallback roleNameCallback = new RoleNameCallback("RoleName");

  Callback[] callbacks = new Callback[]{nameCallback, fullNameCallback, roleNameCallback};
     try {
        callbackHandler.handle(callbacks);
     } catch (IOException ex) {
        Logger.getLogger(MyLoginModule.class.getName()).log(Level.SEVERE, null, ex);
     } catch (UnsupportedCallbackException ex) {
        Logger.getLogger(MyLoginModule.class.getName()).log(Level.SEVERE, null, ex);
     }
  username = nameCallback.getName(); fullName = fullNameCallback.getName();
  roleName = roleNameCallback.getName();
  loginSucceeded = true;
  return true;  }
```

LoginCallbackHandler

```
public class LoginCallbackHandler implements CallbackHandler {

 private String username = null; private String fullName = null; private String roleName = null;

 public LoginCallbackHandler(String username, String fullName, String roleName) {
  this.username = username; this.fullName = fullName; this.roleName = roleName;
 }

 public void handle(Callback[] callbacks) {
  for (int i = 0; i < callbacks.length; i++) {
   if (callbacks[i] instanceof FullNameCallback) {
    FullNameCallback fnc = (FullNameCallback) callbacks[i];    fnc.setName(fullName);
   } else if (callbacks[i] instanceof RoleNameCallback) {
    RoleNameCallback rnc = (RoleNameCallback) callbacks[i]; rnc.setName(roleName);
   } else if (callbacks[i] instanceof NameCallback) {
    NameCallback nc = (NameCallback) callbacks[i]; nc.setName(username);
   }                                                  } } }
```

JAASAuthenticationFacade → Methode authenticate

```
@Override public void authenticate() throws LoginException{
    boolean result = false;
 try  {
      user=(Users) usersFacade.findByUsername(credentials.getUsername());
      if (user.getUserPassword().substring(0,40).equals(
         hashif.hashCode(credentials.getPassword()+user.getUserPassword().substring(40,80))) )
      { try {
        Configuration configuration = new MyConfig();
        loginContext = new LoginContext("MyLoginModule",new Subject(),
              new LoginCallbackHandler(credentials.getUsername(),
              user.getUserFirstname()+" "+user.getUserSecondname(),
              rolesFacade.findByUsername(credentials.getUsername()).getRoleValue()), configuration);
        loginContext.login();
        result = true;
          } catch (LoginException e) {
              System.out.println("[authenticate] LoginException: " + e.getMessage()); result = false; }
          } catch …  // wie bei den bisherigen Login-Implementierungen
```

Weitere unterstützende Klassen

```
public class FullNameCallback extends NameCallback{
   public FullNameCallback(String fnc){super(fnc);}}        // RoleNameCallback analog
public class FullNamePrincipal extends SimplestPrincipal{
   public FullNamePrincipal(String str){super(str);}}        // UsernamePrincipal, RoleNamePrincipal analog

public class MyPermission extends BasicPermission {
  public MyPermission(String perm)                        { super(perm); }
  public MyPermission(String perm, String actions) { super(perm, actions); } }

public class MyConfig  extends Configuration {
   public AppConfigurationEntry[] getAppConfigurationEntry(String name) {
     AppConfigurationEntry[] result = null;
     if (name.equals("MyLoginModule")) {
       result = new AppConfigurationEntry[1];Hashtable options = new Hashtable(3);
       result[0] = new AppConfigurationEntry("jaas.MyLoginModule",
                        AppConfigurationEntry.LoginModuleControlFlag.REQUIRED, options);
     }
     return result; }}   // hier applikativ; Konfiguration ist aber auch als Konfigurationsdatei  möglich
```