



Advanced Java technologies: JBoss

Časť 2.

Contexts and Dependency Injection (CDI)

Enterprise JavaBeans 3.1

Jozef Hartinger

11.3.2013

Contexts and Dependency Injection for the Java EE platform (CDI)

- Java EE Špecifikácia (JSR-299)
- Niekoľko implementácií
 - **Weld** (JBoss)
 - **OpenWebBeans** (Apache)
 - **CanDI** (Caucho Resin)
- CDI 1.1 (JSR-346) tesne pred vydaním
- Definuje komponentový model
 - komponenty spravované servrom
 - komponentám sú poskytovné služby
 - správa životného cyklu
 - správa závislostí
 - ďalšie

Motivácia

- Voľné prepojenie komponent
 - Jednoduchý komponentový model
 - Silne typovaný
- Integrácia vrámci platformy
- Rozšíriteľnosť

Loose coupling (Voľné prepojenie)

- Rozšíriteľnosť a zmeny aplikácie
- Testovanie
- Inversion of control



Strong typing

- Java (rozhrania, triedy, anotácie)
- Refactoring
- Kompilátor

Spôsoby zabalenia CDI aplikácie

- WAR
 - WEB-INF/beans.xml
 - WEB-INF/classes
 - WEB-INF/lib
- JAR
 - META-INF/beans.xml
- EAR

- Bean descriptor (beans.xml)

Charakteristika CDI Beany

- Trieda
- Nie je abstraktná
- Má vhodný (public) konštruktor
 - Bezparametrický
 - @Inject (vid'. Ďalej)
- Nie je final a nemá final metódy
- Spĺňa ďalšie (menej podstatné) podmienky [\[1\]](#) [\[2\]](#)

- Triedy ktoré spĺňajú tieto kritéria sú implicitne CDI Beany (**nie je nutné ich nijak označiť**)
- Session Beany (EJB) sú zároveň CDI Beany

Vlastnosti CDI Beany

- Scope (vid'. ďalej)
- Množina typov (Bean type closure)
 - Všetky nadtypy a všetky (aj nepriamo) implementované rozhrania
- Meno (@Name)
 - voliteľné
- Množina kvalifikátorov (Qualifiers – vid'. ďalej)
- Množina stereotypov (Stereotypes)
- Množina interceptor väzieb (Interceptor bindings)

Príklad CDI Beany

```
public class Registrator {  
  
    @Inject  
    private User user;  
  
    public void register() {  
        // ...  
    }  
}
```


Príklad CDI Beany

```
@SessionScoped
@Named
@Stateful
public class LoginManager implements Serializable {

    private static final long serialVersionUID = 1L;

    @Inject
    private EntityManager em;

    @Inject
    private UserManager userManager;

    private User currentUser;
```

Životný cyklus

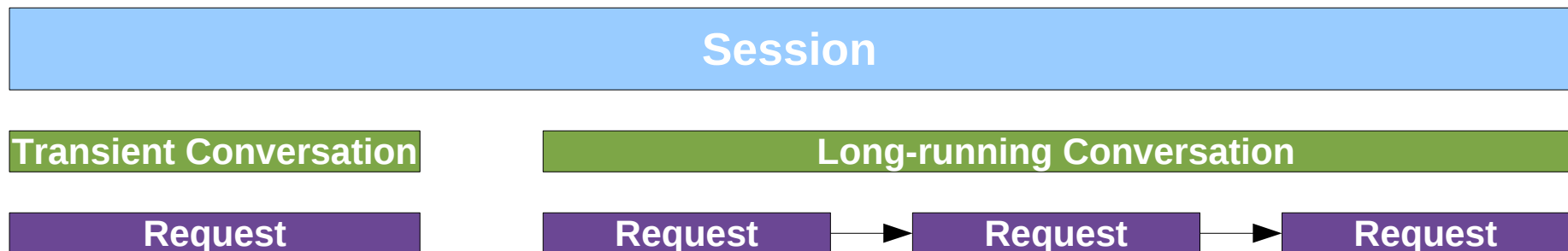
- Inversion of control – o životný cyklus sa stará kontainer
- **Vytvorenie novej inštancie** (vid'. constructor Injection)
- Field injection (naplnenie atribútov objektu)
- @PostConstruct / @Inject initializer
- **Uloženie do kontextu** (scope)
- @PreDestroy / @Disposer method (vid'. ďalej)

Kontexty - Scopes

- Servlet
 - **Request Scope** - @RequestScoped
 - **Session Scope** - @SessionScoped
 - **Application Scope** - @ApplicationScoped
- Implicitné
 - **Dependent Scope** - @Dependent
 - Vždy sa vytvára nová inštancia
 - Jej životný cyklus je zviazaný s Beanou ktorá ju požaduje
- JSF
 - **Conversation Scope** - @ConversationScoped

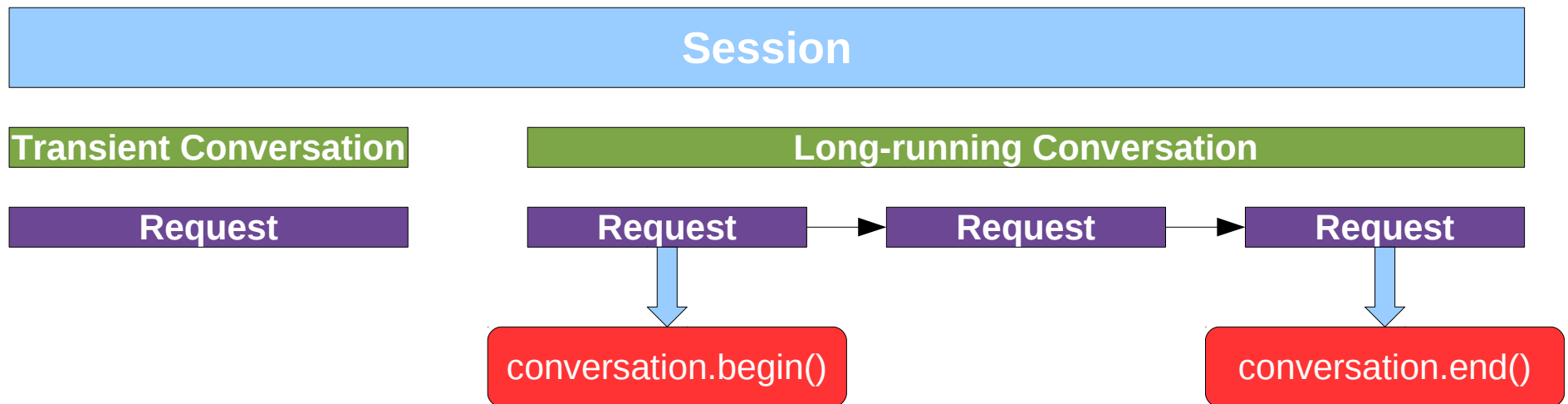
Conversation scope

- Uchováva stav sekvencie po sebe idúcich logicky súvisiacich requestov
- Samotná konverzácia má 2 stavy
 - **Transient** – jeden JSF request
 - **Long-running** – sekvencia JSF requestov



Ovládanie konverzácií

```
public class NewAuctionWizzard {  
  
    @Inject  
    private Conversation conversation;  
  
}
```



Conversation API

```
public interface Conversation
{
    public void begin();

    public void begin(String id);

    public void end();

    public String getId();

    public long getTimeout();

    public void setTimeout(long milliseconds);

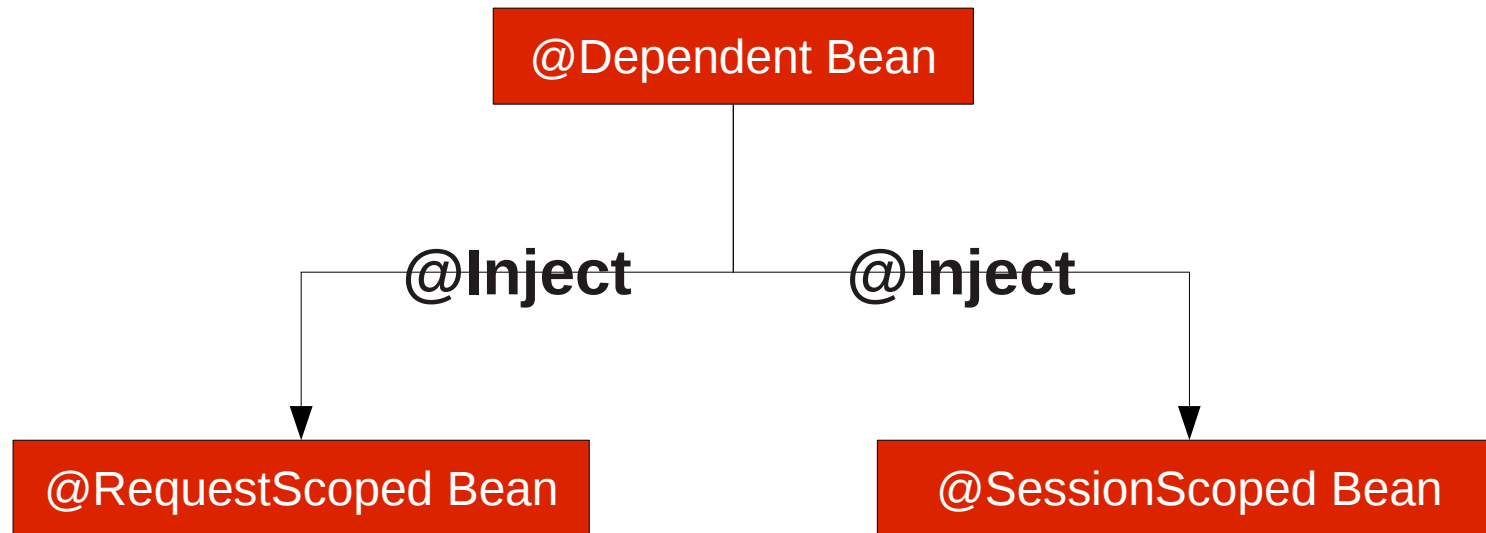
    public boolean isTransient();
}
```

Propagácia konverzácie

- String identifier
 - Set by application
 - Generated by container
- Vrámci session
- Propagácia pomocou **cid** parametru
- Automatická propagácia pri odosielaní JSF formuláru

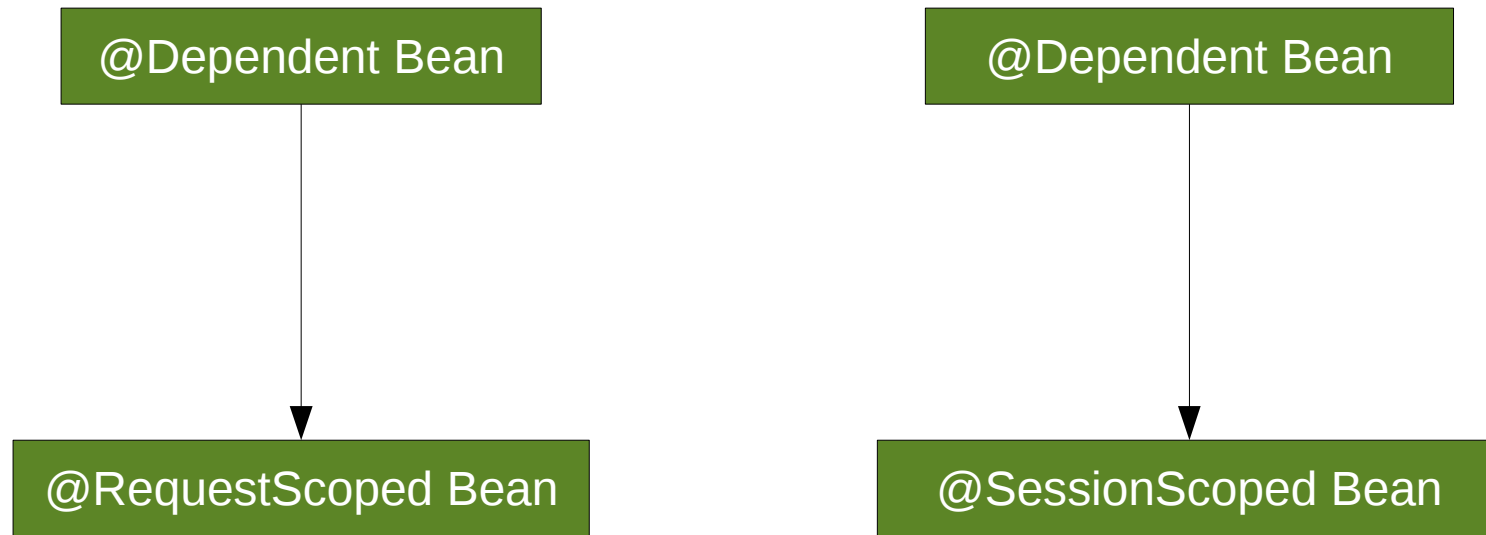
Dependent Pseudo Scope

- Na úrovni tried



Dependent Pseudo Scope

- Na úrovni objektov



Passivation

- Veľkosť pamäte potrebnej pre uloženie sessions rastie lineárne s počtom existujúcich sessions
- Session a Conversation – passivating scopes
- Odkladanie neaktívnych sessions na sekundárne úložisko
- Replikácia sessions
 - Load balancing
 - Failover
- Každá @SessionScoped alebo @ConversationScoped komponenta musí byť serializovateľná
 - `java.io.Serializable`
 - `java.io.Externalizable`

Dependency Injection

- Mechanizmus získavania závislostí
- Spúšťa sa explicitne pomocou anotácie @Inject

```
public class PaymentManager {  
  
    @Inject  
    private PaymentProcessor pp;  
}
```



Pravidlá DI - Typesafe resolution

- Výber na základe
 - Typu
 - Kvalifikátorov (Qualifiers)
 - Slúžia primárne na odlíšenie viacerých inštancií daného typu
- Na dependency injection sa použije objekt daného typu
 - Z kontextu (ak objekt existuje v kontexte)
 - Nový
 - vytvorí ju container
 - pri `@Dependent` sa vytvára vždy nový objekt

Qualifiers

```
@Qualifier
@Target({ TYPE, METHOD, PARAMETER, FIELD })
@Retention(RUNTIME)
public @interface LoggedIn {
}
```


```
@Inject
@LoggedIn
private User user;
```

Pravidlá DI - Typesafe resolution

- Injection point Beanu X vyžaduje
 - Typ (Z)
 - Množinu kvalifikátorov
- Bean Y poskytuje
 - Množinu typov
 - Množinu kvalifikátorov
- Bean Y je vhodným kandidátom ak
 - Vyžadovaný typ sa nachádza v množine poskytovaných typov
 - Všetky požadované kvalifikátory sa nachádzajú v množine poskytovaných kvalifikátorov

```
public class Vehicle
    @Wild
    public class Cat
    @Domestic
    public class Dog

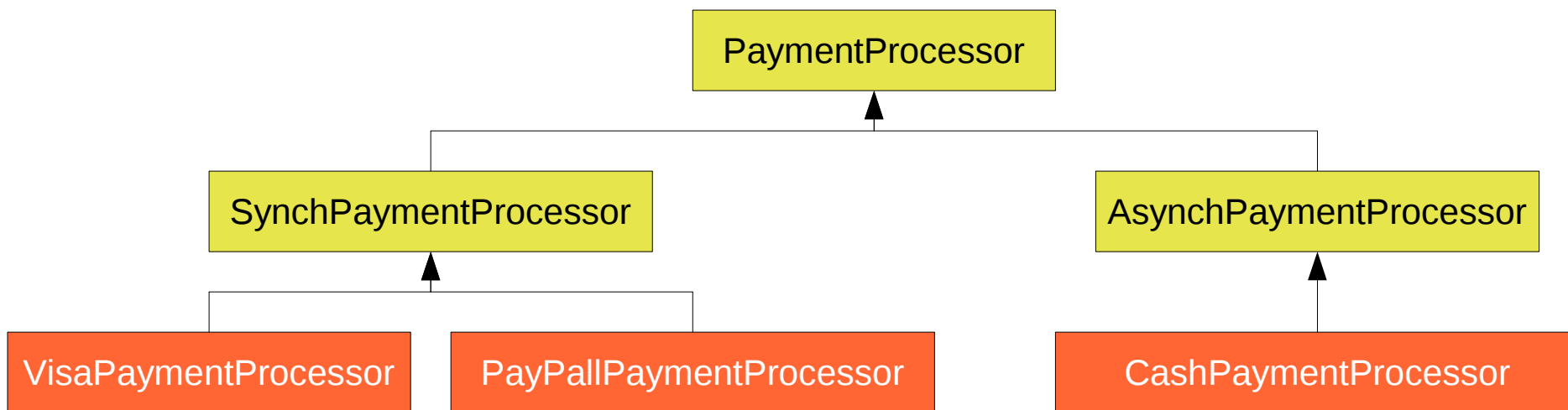
@Inject
@Wild
Animal animal;
```



Poskytované typy

- Tranzitívny uzáver rozširovaných a implementovaných typov
- `java.lang.Object` je vždy typom
- `java.io.Serializable` nie je nikdy typom

Príklad typovej rezolúcie (typy)



```
@Inject
private PaymentProcessor pp;
```

```
@Inject
private SynchPaymentProcessor spp;
```

```
@Inject
private VisaPaymentProcessor vpp;
```


Príklad typovej rezolúcie (kvalifikátory)

```
public interface Animal
```

```
public class Sheep
```

```
@Domestic  
public class Dog
```

```
@Wild  
@Hungry  
public class Lion
```

```
@Wild  
public class Elephant
```

```
@Domestic  
@Lazy  
public class Cat
```

```
@Inject  
@Any  
private Animal animal;
```

```
@Inject  
@Domestic  
private Animal domestic;
```

```
@Inject  
@Wild @Lazy  
private Animal lazyWildAnimal;
```



Špeciálne kvalifikátory

- @Default – implicitný kvalifikátor v prípade, že daná komponenta / injection point nedefinuje iný kvalifikátor
- @Any – implicitný kvalifikátor každej komponenty
- @Named

```
public class Shelter {
```

```
    @Inject  
    Animal animal1;
```

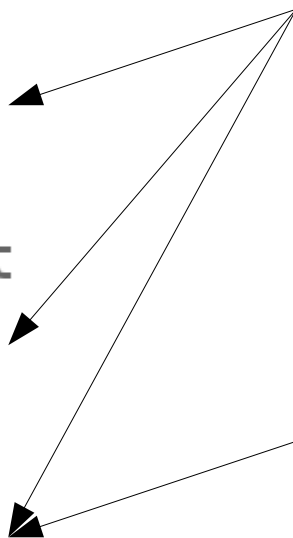
```
    @Inject @Default  
    Animal animal2;
```

```
    @Inject @Any  
    Animal animal3;
```

```
}
```

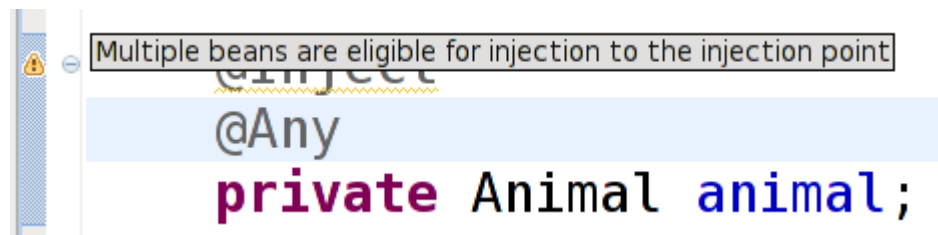
```
public class Dog  
    implements Animal {  
}
```

```
@Domestic  
public class Cat  
    implements Animal {  
}
```



Pravidlá DI - Typesafe resolution

- 1 injection point = práve jeden vhodný Bean
- Kontrolované pri deploy (alebo tooling)



Multiple beans are eligible for injection to the injection point

```
@Inject  
@Any  
private Animal animal;
```

Typy DI

- Field injection
- Constructor
 - Immutable objects
- Initializer methods
- Producer method/Disposer/Observer

```
@Inject  
private PaymentProcessor pp;
```

```
@Inject  
public PaymentManager(PaymentProcessor pp) {  
    this.pp = pp;  
}
```

```
@Inject  
public void init(PaymentProcessor pp) {  
    this.pp = pp;  
}
```

Producer method

- Pre prípady keď Java trieda nestačí na vytvorenie objektu
- Rozšírená kontrola
 - Výsledok JPA dotazu
 - Náhodné číslo
 - ...
- Konkrétna metóda (nie je abstraktná)
 - Umiestnená na CDI Bean
 - Tranzitívny uzáver typu návratovej hodnoty = typ objektu
 - Kvalifikátory, Scope a meno objektu sa definujú na metóde

Producer method

```
@ApplicationScoped
public class RandomNumberGenerator {

    private final java.util.Random random = new java.util.Random();

    @Produces
    @Random
    public int generateRandomInt() {
        return random.nextInt();
    }
}
```

Producer method

```
@Produces
@Named
@CurrentAuction
public Auction getCurrentAuction() {
    if (currentAuction != null &&
        !em.contains(currentAuction)) {
        currentAuction = em.merge(currentAuction);
    }
    return currentAuction;
}
```

Producer field

- Zjednodušenie producer metód

```
@Produces  
@Zero  
private final int zero = 0;
```


Disposer method

- Niektoré objekty vyžadujú explicitné zničenie
- @PreDestroy nefunguje na produkovaných objektoch

```
public void close(@Disposes Connection connection) {  
    try  
    {  
        connection.close();  
    }  
    catch (SQLException e)  
    {  
        e.printStackTrace();  
    }  
}
```

Alternatives

- @Alternative
- Nahradenie komponenty inou na základe prostredia
 - Konfigurácia
 - Testovanie
- Alternative musí byť explicitne aktivovaná (enabled) v component descriptore (beans.xml)
- Rozšírenie **Typesafe resolution** algoritmu
 1. Výber vhodných kandidátov na základe typu a kvalifikátorov
 2. V prípade, že existuje viac ako jeden kandidát sú eliminovaní všetci kandidáti okrem aktivovaných alternatives

Integrácia s prezentačnou vrstvou

- Prístup k CDI komponentám pomocou mena
- @Named

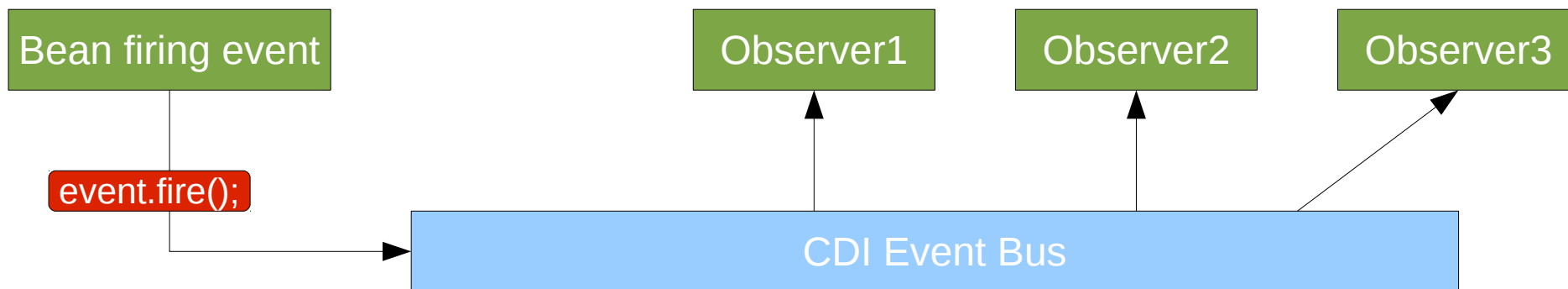
```
<h:form>
  <h:inputText value="#{student.firstName}" />
  <h:inputText value="#{student.surname}" />
  <h:commandButton id="register"
    action="#{registrator.register}" value="Register" />
</h:form>
```

Použitie DI v non-contextual komponentách

- Servlet / Filter / ServletListener
- Message-Driven bean
- JAX-RS resources / providers
- JAX-WS endpoints

Udalosti (Events)

- Založené na návrhovom vzore “Observer”
- Oddelenie producentov udalostí od ich konzumentov
- Loose coupling – ďalšie observery sa môžu pridať neskôr bez zásahu do producenta udalostí
- Synchronne
- Eventy môžu niesť “náklad”



Udalosti (Events) - Vyvolanie udalosti

- Vstavany Event bean

```
public class RegistrationManager {  
    @Inject  
    @Registered  
    private Event<User> userEvent;  
}
```

- Vyvolanie udalosti

```
public void register()  
{  
    em.persist(user);  
    userEvent.fire(user);  
}
```

Udalosti (Events) - Odchytenie udalosti

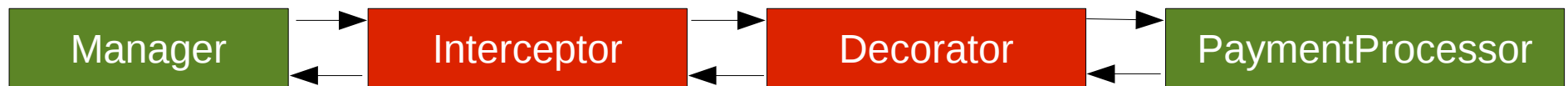
- Observer method
- Pravidlá typovej rezolúcie rovnaké ako pre DI

```
public void log(@Observes User user) {  
    System.out.println("Hello " + user.getName());  
}
```

```
public void log(@Observes @Registered User user) {  
    System.out.println("Hello " + user.getName());  
}
```

Dekorátory a Interceptory

- Odchytávanie volaní metód
- Izolovanie kódu, ktorý by sme inak opakovali v metódach
- Zmena správania v závislosti na prostredí

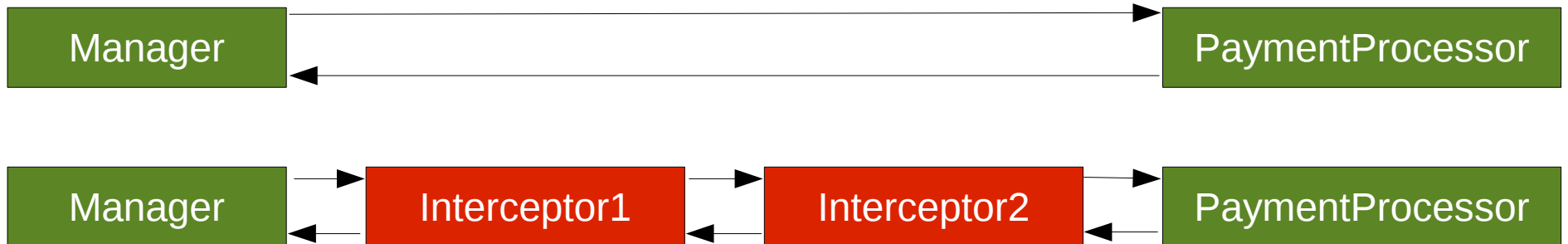


Interceptory

- Oddelenie technických detailov od logiky aplikácie
- Cross-cutting concerns (logovanie, zabezpečenie, caching)
- Môžu odchytať
 - Volania metód
 - Životný cyklus (@PostConstruct, @PreDestroy)

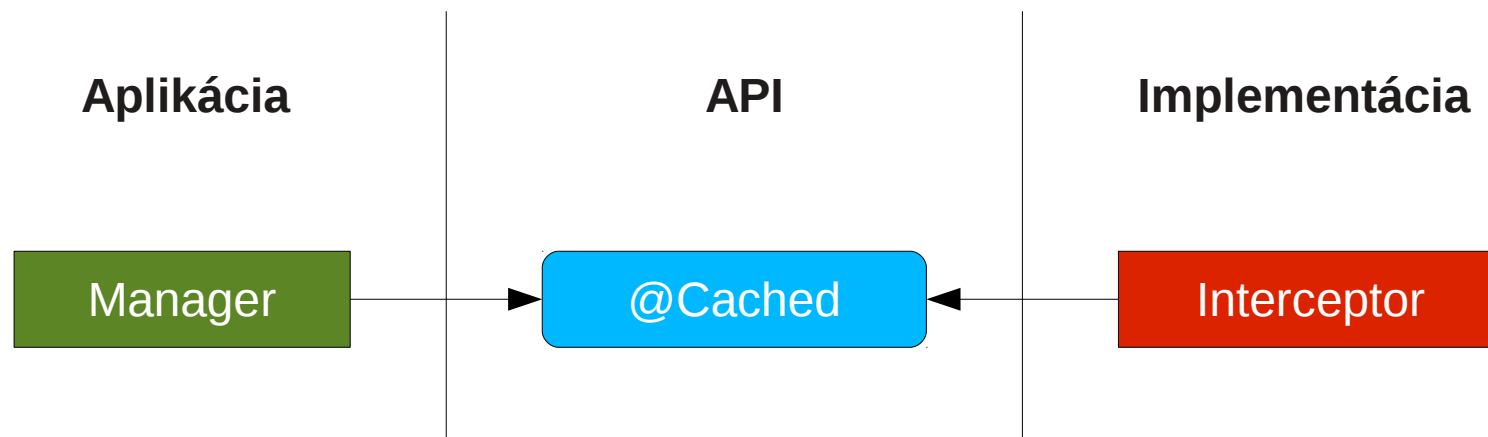
Interceptory

```
public class PaymentManager {  
  
    @Inject  
    private PaymentProcessor pp;  
  
    pp.foo();  
}
```



Interceptor binding

- Anotácia reprezentujúca funkcionálnosť poskytovanú interceptorom (loose coupling)



Decorator

- “Typovaný” interceptor – “pozná” sémantiku volania ktoré obaluje
- Nie je nutné anotovať odchyťávaný objekt
- Implementuje priamo metódy rozhrania ktoré obaluje
 - Nemusí implementovať všetky metódy rozhrania (abstraktná trieda)
- Špeciálny injection point (@Delegate)

@Decorator

```
public abstract class LargeTransactionDecorator implements Account {
```

```
    @Inject
```

```
    @Delegate
```

```
    private Account account;
```

```
    @Inject
```

```
    @LoggedIn
```

```
    private User user;
```

```
    @Inject
```

```
    @SecurityLimit
```

```
    private BigDecimal limit;
```

```
public void withdraw(BigDecimal amount) {
```

```
    // security check
```

```
    if (amount.compareTo(limit) > 0) {
```

```
        if (!user.isAuthorizedForLargeTransactions()) {
```

```
            throw new SecurityException("Amount too high.");
```

```
        }
```

```
    }
```

```
    // proceed
```

```
    account.withdraw(amount);
```

```
}
```

```
}
```

Aktivácia a poradie

```
<beans>
  <decorators>
    <class>cz.muni.fi.pv243.FooDecorator</class>
    <class>cz.muni.fi.pv243.BarDecorator</class>
  </decorators>
  <interceptors>
    <class>cz.muni.fi.pv243.FooInterceptor</class>
  </interceptors>
</beans>
```

Stereotypy

- Obrana pred “annotation hell”
- Anotácia združujúca
 - Scope
 - Interceptor bindings
 - @Name
- Vstavovaný interceptor **@Model**

```
@Named
@RequestScoped
@Documented
@Stereotype
@Target( { TYPE, METHOD, FIELD } )
@Retention(RUNTIME)
public @interface Model
{
}
```

InjectionPoint metadata

- Vstavovaná komponenta ktorú môžeme použiť pri konštrukcii @Dependent komponent
- umožňuje vytvoriť objekt “prispôsobený” na mieru hostiteľskej komponente

```
public class UserManager {  
  
    private final Logger logger =  
        Logger.getLogger(UserManager.class.getSimpleName());  
  
    public UserManager() {  
        logger.fine("Instance created");  
    }  
  
    // ...  
}
```


InjectionPoint metadata

```
public class Loggers {
    @Produces
    public Logger getLogger(InjectionPoint ip) {
        Member member = ip.getMember();
        Class<?> declaringClass = member.getDeclaringClass();
        return Logger.getLogger(declaringClass.getSimpleName());
    }
}
```

```
public class UserManager {

    @Inject
    private Logger logger;

    public UserManager() {
        logger.fine("Instance created");
    }
}
```

InjectionPoint metadata

```
public interface InjectionPoint {  
    Type getType();  
    Set<Annotation> getQualifiers();  
    Bean<?> getBean();  
    Member getMember();  
    Annotated getAnnotated();  
    boolean isDelegate();  
    boolean isTransient();  
}
```

Tvorba rozšíření s InjectionPoint

```
public class UserManager {  
  
    @Inject @SystemProperty("java.version")  
    private String version;  
  
}
```

Tvorba rozšíření s InjectionPoint

```
@Produces
@SystemProperty("")
public String readProperty(InjectionPoint ip) {
    Annotated annotated = ip.getAnnotated();
    SystemProperty annotation =
        annotated.getAnnotation(SystemProperty.class);
    String propertyName = annotation.value();
    return System.getProperty(propertyName);
}
```

Contexts and Dependency Injection

- Vlastnosti CDI komponent
- Životný cyklus (Scopes)
- Dependency Injection
- Udalosti (Events)
- Dekorátory a Interceptory (Decorators and Interceptors)
- Rozšíriteľnosť

Enterprise JavaBeans 3.1

- Zjednodušenie existujúceho komponentového modelu
 - Rozhrania (views) nie sú povinné
 - Session beans sú automaticky CDI komponentami
 - Scopes (Stateful session bean)
 - Dependency injection
 - Events
 - Interceptory / Dekorátory
 - Možnosť používať v jednoduchej webovej aplikácii (.war)
- @Singleton session bean
- Asynchrónne volanie metód

Singleton session bean

- Jediná inštancia zdieľaná vrámci celej aplikácie
 - Vrámci jednej JVM
- V porovnaní s **@ApplicationScoped** navyše
 - Inicializovaná pri štarte aplikácie (**@Startup**)
 - Podporuje paralelný prístup
 - @Lock(LockType.READ)
 - @Lock(LockType.WRITE)
 - @AccessTimeout
 - @ConcurrencyManagement
- DEMO na cvičení

Asynchrónne volania metód

- Abstrakcia od priameho použitia vlákien, exekútorov a thread pools
- Použitie
 - Asynchrónne vykonávanie dlhotrvajúcej úlohy
 - Odosielanie potvrdzujúceho e-mailu
 - Návratová hodnota **void**
 - Paralelizácia výpočtu
 - Návratová hodnota **Future<V>** - **AsyncResult<V>**
- @Asynchronous
- DEMO na cvičení



Questions?

jharting@redhat.com | www.redhat.com