



Advanced Java technologies: JBoss

Časť 2.

Contexts and Dependency Injection (CDI)

Enterprise JavaBeans

Jozef Hartinger

March 2015

@Inject

@SessionScoped

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

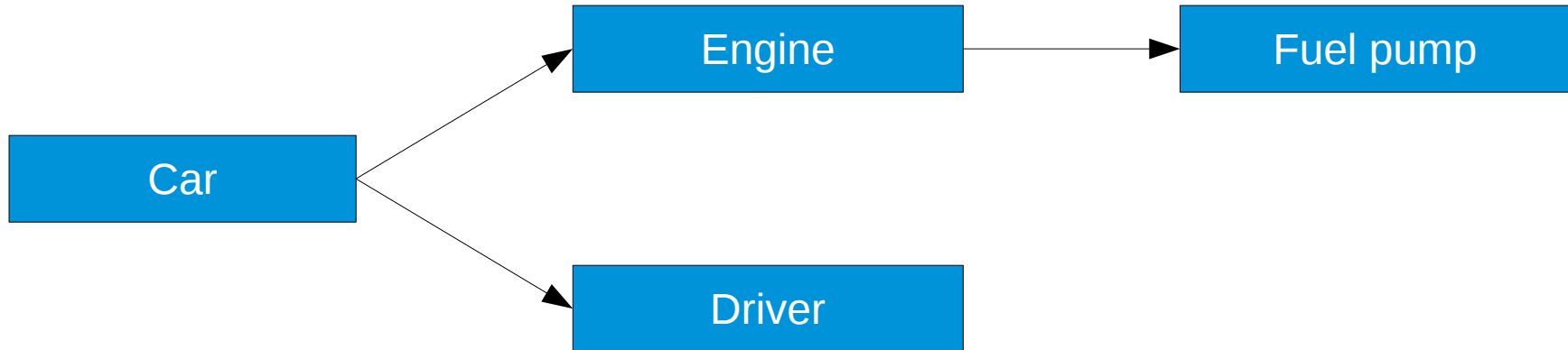
- Java EE Špecifikácia
 - Od Java EE 6 (2009)
- Niekoľko implementácií
 - **Weld** (JBoss)
 - **OpenWebBeans** (Apache)
 - **CanDI** (Resin)
- Aktuálna verzia 1.2

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

- Defínuje komponentový model
 - komponenty spravované servrom
 - komponentám sú poskytované služby
 - správa životného cyklu
 - správa závislostí
 - etc.
- Integrácia vrámci Java EE

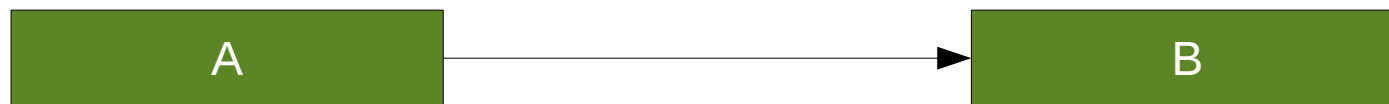
Dependency Injection (@Inject)

- Inversion of control



Loose coupling (Voľné prepojenie)

- Rozšíriteľnosť a zmeny aplikácie
- Testovanie



Strong typing

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

Charakteristika CDI Beany

- Trieda
- Nie je abstraktná
- Má vhodný konštruktor
 - Bezparametrický
 - @Inject (vid'. Ďalej)
- Obsahuje “**bean defining annotation**”
 - Scope, @Interceptor, @Decorator
- V prípade, že využíva proxy
 - Nie je final
 - Nemá final metódy
- 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
- Množina kvalifikátorov (Qualifiers - vid'. Ďalej)
- Voliteľne
 - Meno (@Named)
 - Množina stereotypov (Stereotypes)
 - Množina interceptor väzieb (Interceptor bindings)

Príklad CDI Beany

```
@RequestScoped
public class Car {

    @Inject
    private Driver driver;
    @Inject
    private Engine engine;

    public void driveTo(Location location) {
        engine.start();
        // TODO
    }
}
```

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

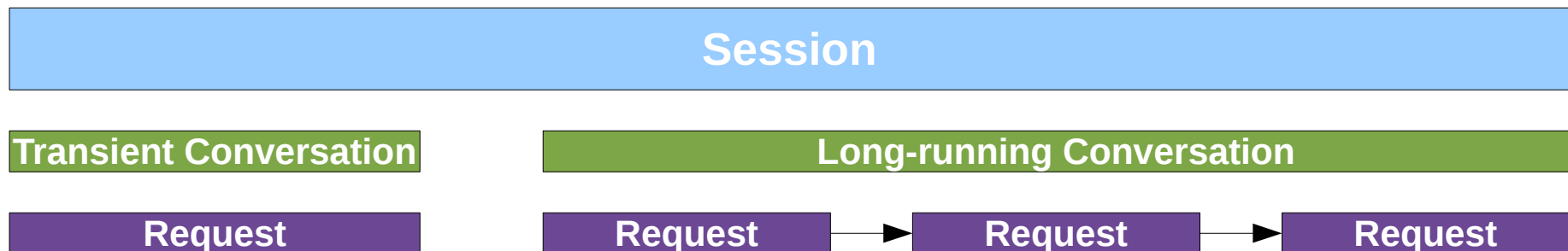
- Inversion of control – o životný cyklus sa stará kontainer
- **Vytvorenie novej inštancie** (vid'. constructor injection)
- Field injection (naplnenie atribútov objektu)
- @PostConstruct / @Inject initializer
- **Uloženie do kontextu** (scope)
- @PreDestroy / @Disposer method (vid'. ďalej)

Kontexty - Scopes

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

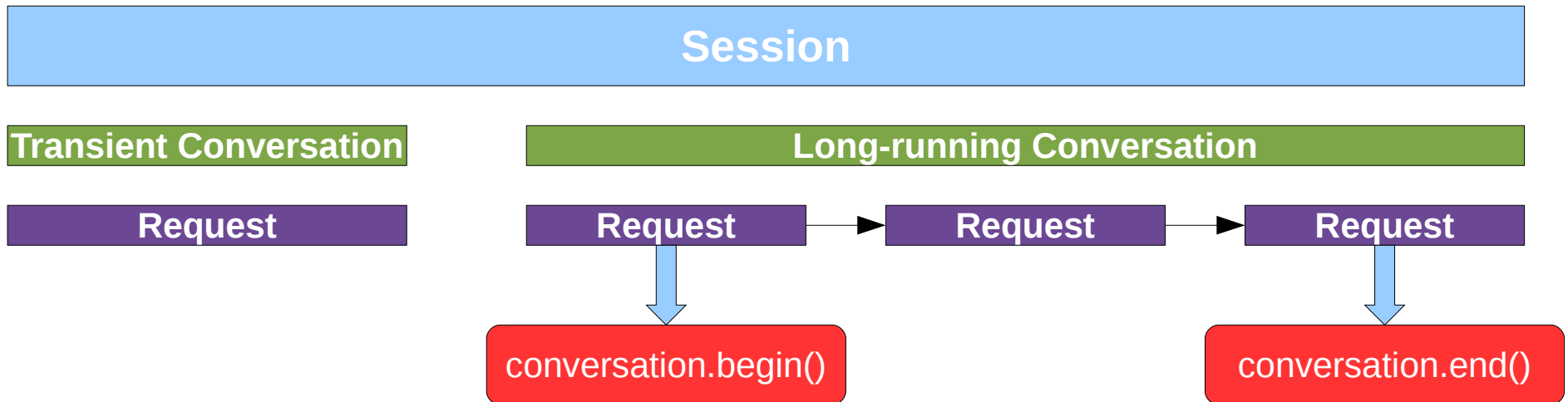
Conversation scope

- Uchováva stav sekvencie po sebe idúcich logicky súvisiacich requestov
- Samotná konverzácia má 2 stavy
 - **Transient** – jeden Servlet request
 - **Long-running** – sekvencia Servlet 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();
}
```

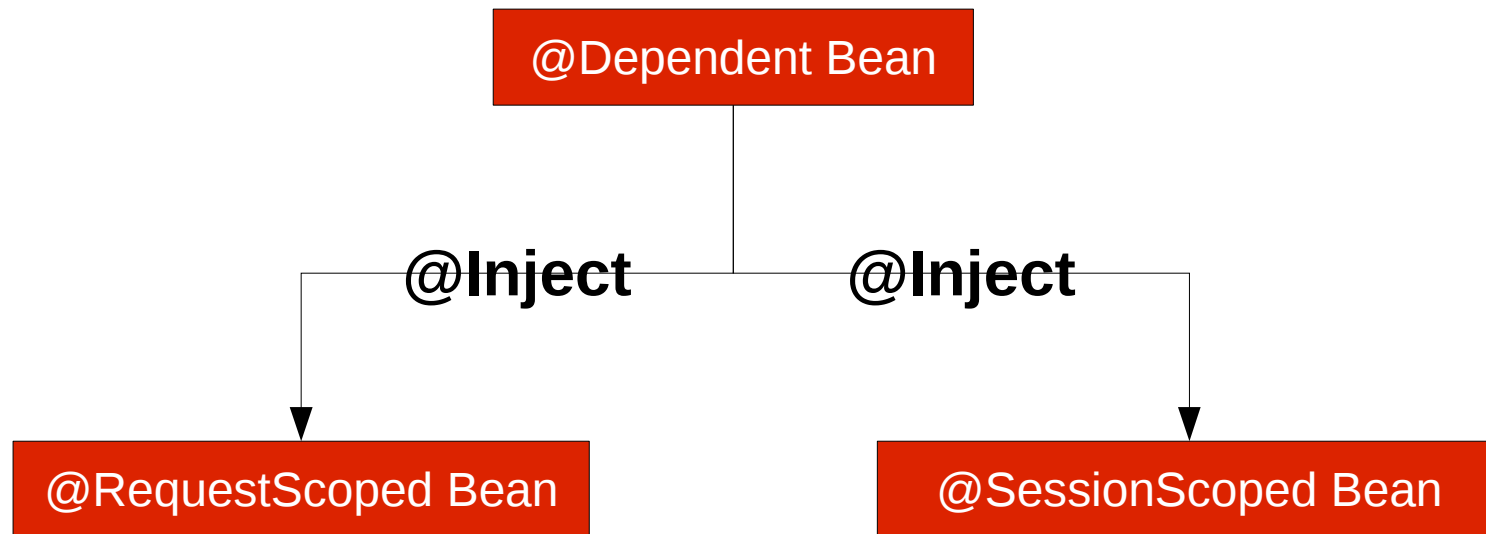
Propagácia konverzácie

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

/cdi-seminar/registration1.jsf?cid=1

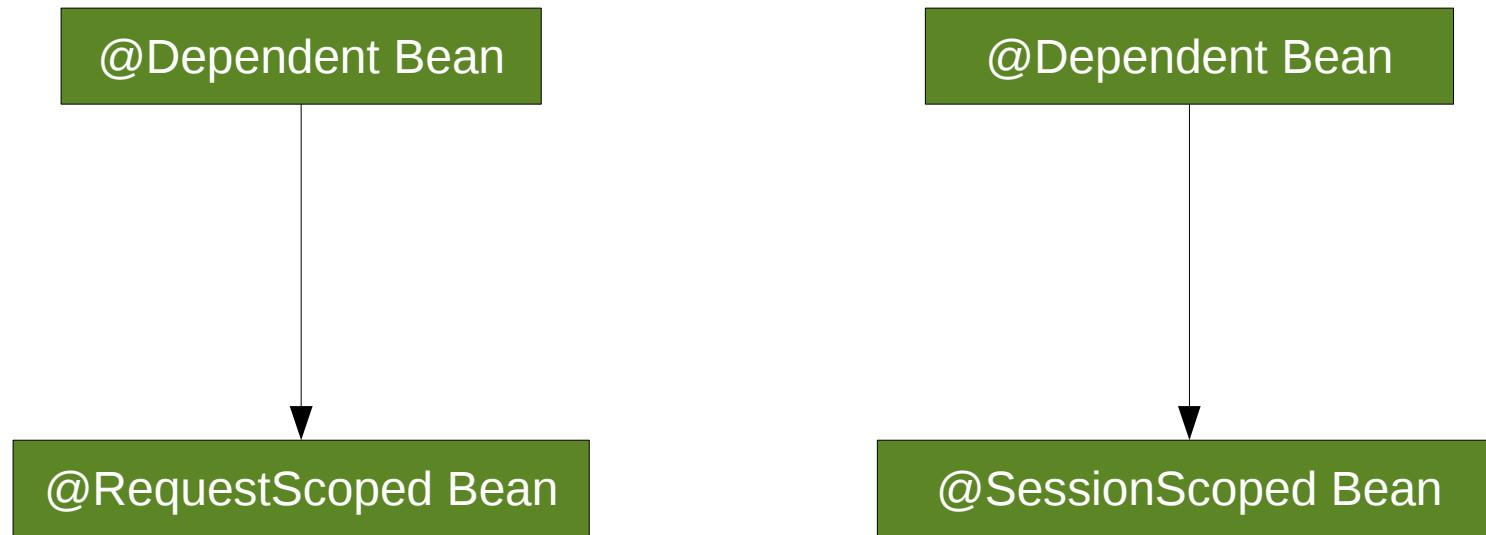
Dependent Pseudo Scope

- 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

```
@RequestScoped  
public class Car {  
  
    @Inject  
    private Driver driver;
```



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  
@Retention(RetentionPolicy.RUNTIME)  
public @interface LoggedIn {  
  
}
```


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

Pravidlá DI - Typesafe resolution

- Injection point 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

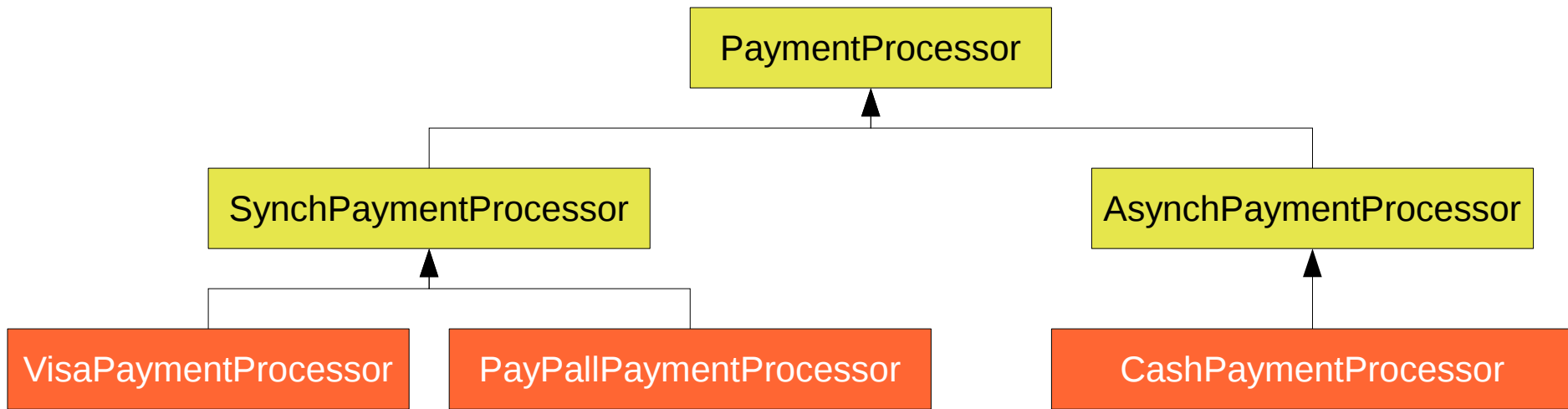
@Inject
@Wild
Animal animal;
```



Poskytované typy (Bean types)

- 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;
```

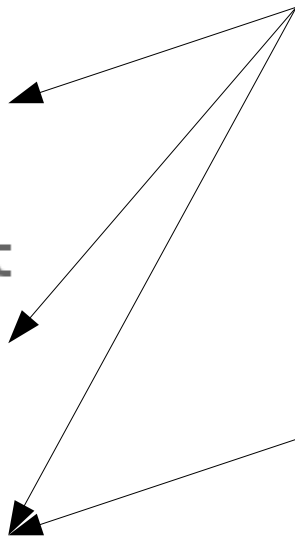
```
    @Inject @Default  
    Animal animal2;
```

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

```
}
```

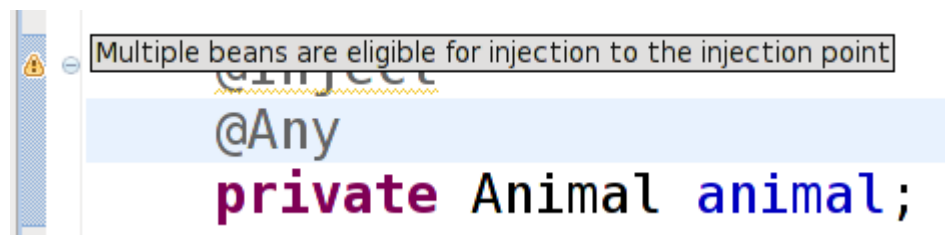
```
public class Dog  
    implements Animal {  
}
```

```
@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  
@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

- Factory - pre prípady keď volanie konštruktoru 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 produkovanych 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) pomocou **@Priority**
- 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
 3. V prípade, že všetci kandidáti majú definovanú prioritu, je vybraný ten s najvyššou prioritou

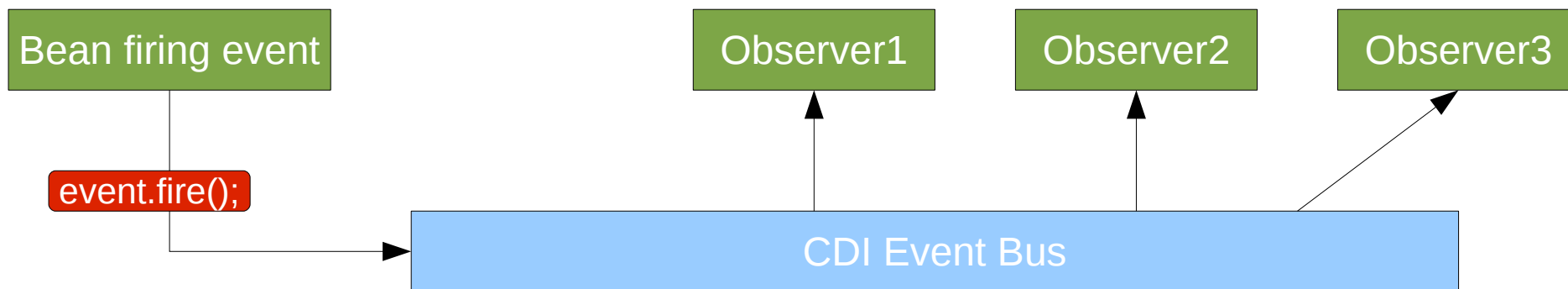
Integrácia s prezentačnou vrstvou

- 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>
```

Udalosti (Events)

- Komunikácia pomocou odoslania správ
- Oddelenie producentov správ od ich konzumentov
- Loose coupling - ďalšie observery sa môžu pridať neskôr bez zásahu do producenta udalostí
- Synchronne
- Event = message (payload)



Udalosti (Events) - Odoslanie správy

- Vstavovaný Event bean

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

- Vyvolanie udalosti

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

Vlastnosti správy

- Typ správy
- Qualifiers

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

Udalosti (Events) - Doručenie správy

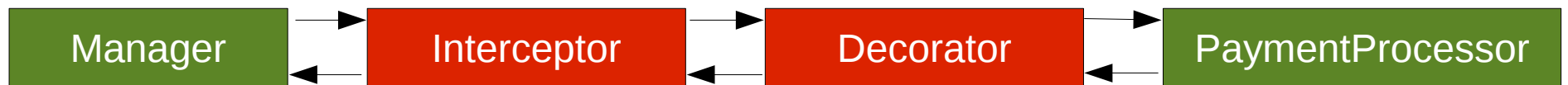
- Observer method
- Pravidlá typovej rezolúcie rovnaké ako pre DI

```
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

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

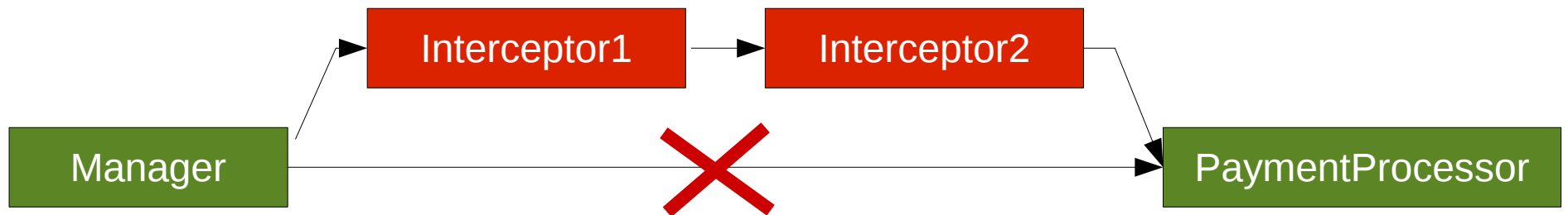


Interceptory

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

Interceptory

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



Interceptory

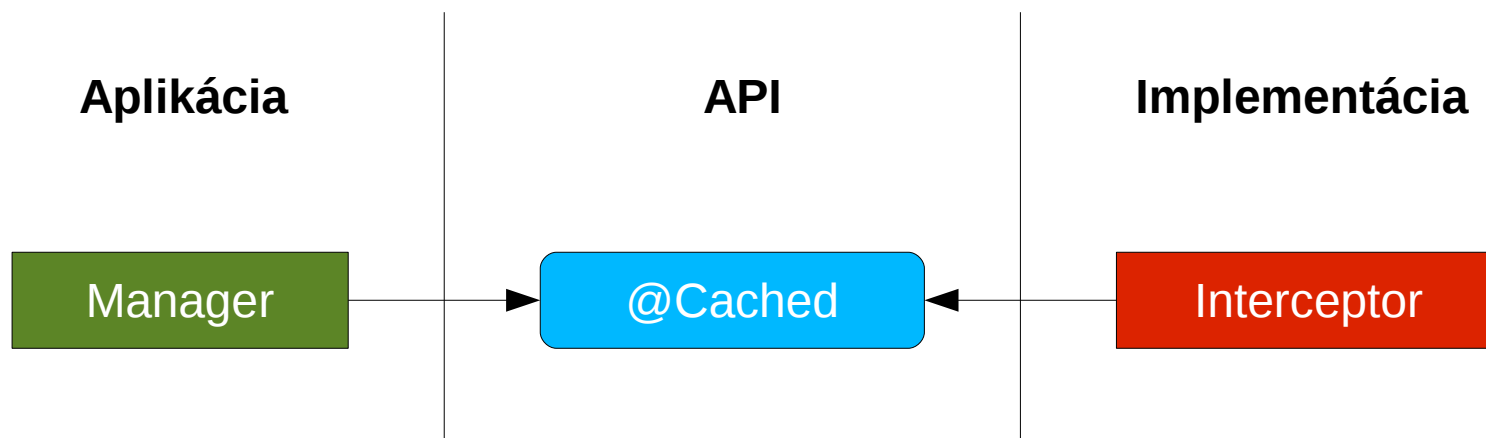
```
@Interceptor
@Cached
@Priority(Interceptor.Priority.APPLICATION)
public class CachingInterceptor {

    @AroundInvoke
    Object intercept(InvocationContext ctx) throws Exception {
        // do something before the call
        Object result = ctx.proceed();
        // do something after the call
        return result;
    }
}
```

Interceptor binding

- Anotácia reprezentujúca funkcionálnosť poskytovanú interceptorm (loose coupling)

```
@InterceptorBinding  
@Retention(RetentionPolicy.RUNTIME)  
public @interface Cached {  
  
}
```



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 interface Account {  
    void withdraw(BigDecimal amount);  
}
```

@Dependent

```
public class AccountImpl implements Account {  
  
    private BigDecimal balance;  
  
    public AccountImpl(BigDecimal initialBalance) {  
        this.balance = initialBalance;  
    }  
  
    public void withdraw(BigDecimal amount) {  
        balance.subtract(amount);  
    }  
}
```

Decorator

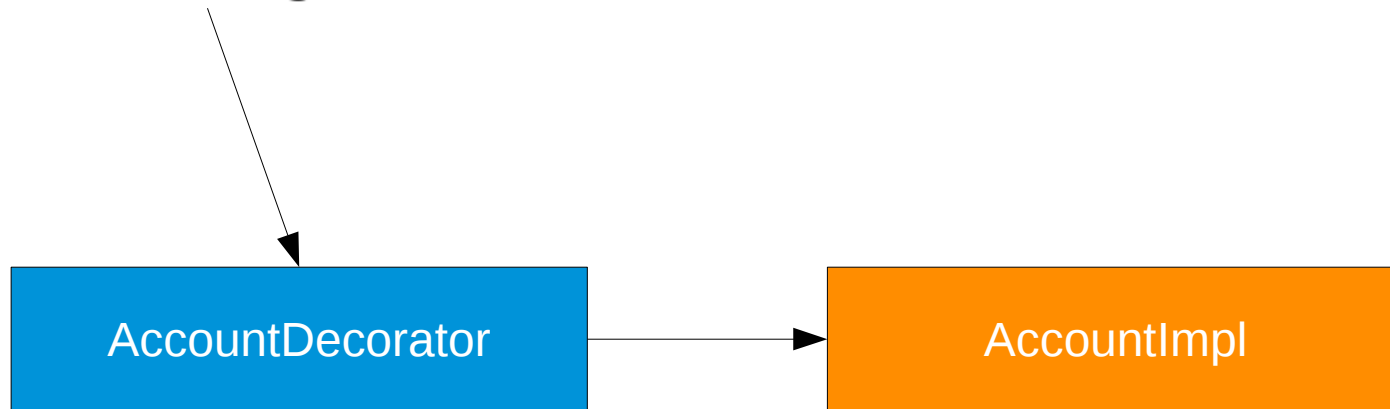
```
@Inject  
private Account account;  
account.withdraw(BigDecimal.TEN);
```



AccountImpl

Decorator

```
@Inject  
private Account account;  
account.withdraw(BigDecimal.TEN);
```



Decorator

```
public class AccountDecorator implements Account {  
  
    private final Account delegate;  
  
    public AccountDecorator(Account delegate) {  
        this.delegate = delegate;  
    }  
  
    @Override  
    public void withdraw(BigDecimal amount) {  
        if (amount.compareTo(LIMIT) > 0 &&  
            !isAuthorizedForLargeTransactions()) {  
            throw new SecurityException("Amount too high!");  
        }  
        delegate.withdraw(amount);  
    }  
}
```

Decorator

```
@Decorator
@Priority(Interceptor.Priority.APPLICATION)
public class AccountDecorator implements Account {

    private final Account delegate;

    @Inject
    public AccountDecorator(@Delegate @Any Account delegate) {
        this.delegate = delegate;
    }

    @Override
    public void withdraw(BigDecimal amount) {
        if (amount.compareTo(LIMIT) > 0 &&
            !isAuthorizedForLargeTransactions()) {
            throw new SecurityException("Amount too high!");
        }
        delegate.withdraw(amount);
    }
}
```

Aktivácia a poradie

- @Priority(2000) - magic number
- Interceptors, Decorators, Alternatives

Interceptor.PRIORITY.

0 - 999	PLATFORM_BEFORE
1000 - 1999	LIBRARY_BEFORE
2000 - 2999	APPLICATION
3000 - 3999	LIBRARY_AFTER
4000 - 4999	PLATFORM_AFTER

- Transactional interceptor (200)
- Bean Validation interceptor (4800)

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
{
}
```

Integrácia

- Beans
 - @Inject **HttpServletRequest**
 - @Inject **UserTransaction**
 - @Inject **javax.security.Principal**
- Events
 - @**Initialized(SessionScoped.class) HttpSession**
- Scopes
 - @**TransactionScoped**

Integrácia - non-contextual components

- Servlets / Filters
- JPA Entity listeners
- JSF converters / validators
- JAX-WS / JAX-RS / WebSocket endpoints

- @Inject (the component itself cannot be injected anywhere)
- Firing events (no observing support)
- Interceptors

Contexts and Dependency Injection

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

Tipy na záver

- Nezabúdať **scope annotation**
- Pozor na správny FQCN!
 - javax.inject.Inject
 - javax.enterprise.inject.Produces
 - javax.enterprise.context.*Scoped
 - javax.enterprise.event.Event
 - javax.enterprise.event.Observes
 - javax.enterprise.inject.spi.Extension;

Enterprise JavaBeans

- Komponentový model poskytující služby:
 - Resource injection
 - Transactions
 - **Security** (separátna prednáška)
 - Remote method invocation (RMI) - @Remote
 - Naming and directory services - JNDI
 - **Messaging** (JMS)
 - **Asynchronous invocations**
 - **Scheduling** (Timer service)
 - Concurrency control

Enterprise JavaBeans

- Session beans
 - Stateless
 - Stateful
 - Singleton
- Message-driven beans

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 v rámci celej aplikácie
 - V rámci 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

Enterprise JavaBeans 3.2

- Prevažne kozmetické zmeny
 - `@Stateful(passivationCapable = false)`
 - `@PostConstruct/@PreDestroy` voliteľne transakčné
 - `TimerService.getAllTimers()`
 - Všetky Timers pre daný modul

Odkazy

- <http://cdi-spec.org/>
- <http://docs.jboss.org/cdi/spec/1.2/>
- <http://docs.oracle.com/javaee/7/tutorial/doc/>



Questions?

jharting@redhat.com | www.redhat.com