Advanced Java technologies: JBoss

Část 2.

Contexts and Dependency Injection (CDI)

Enterprise JavaBeans

October 2017
Matěj Novotný

redhat

@Inject

@SessionScoped

redhat

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

- Java EE specifikace
  - Od Java EE 6 (2009)
- Několik implementací
  - **Weld** (JBoss)
  - **OpenWebBeans** (Apache)
- Aktuální verze 1.2 (Java EE 7) & 2.0 (Java EE 8)

redhat

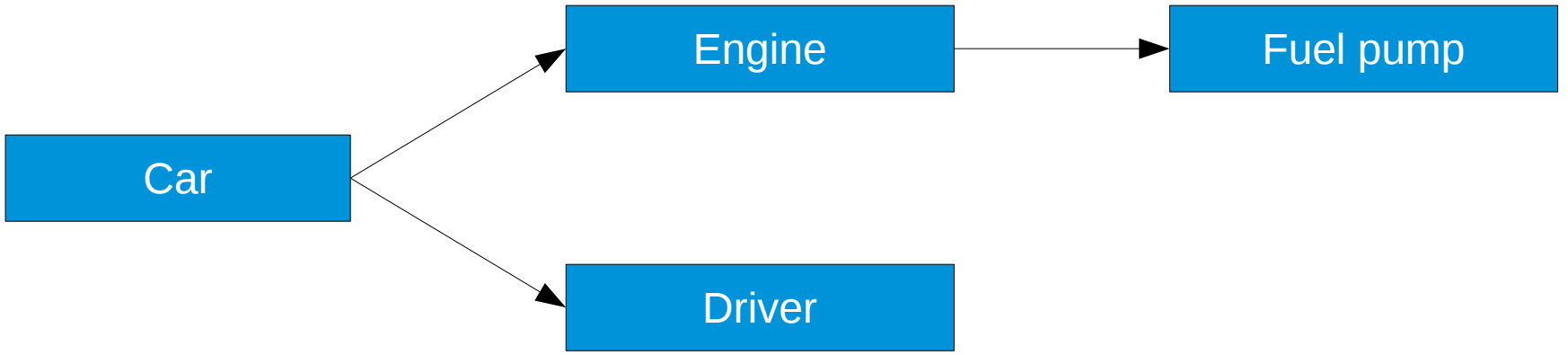
Contexts and Dependency Injection for the Java EE platform (CDI)

- Definuje komponentový model
  - komponenty spravuje server
  - komponentám jsou poskytované služby
    - správa životního cyklu
    - správa závislostí
    - etc.
- Integrace v rámci Java EE

redhat

Dependency Injection (@Inject)

- Inversion of control



redhat

Loose coupling (Volné propojení)

- Rozšiřitelnost a změny aplikace
- Testovaní

Strong typing

- Java (rozhraní, třídy, anotace)
- Refactoring
- Kompilátor

redhat

Charakteristika CDI Beany

- Třída
- Není abstraktní
- Má vhodný konstruktor
  - Bezparametrický
  - @Inject (viz. dále)
- Obsahuje **“bean defining annotation”**
  - Scope, @Interceptor, @Decorator
- V případě, že využívá proxy
  - Není final
  - Nemá final metody

- Session Beany (EJB) jsou zároveň CDI Beany

redhat

Vlastnosti CDI Beany

- Scope (viz. dále)
- Množina typů (Bean type closure)
  - Všechny nadtypy a všechny (i nepřímo) implementované rozhraní
- Množina kvalifikátorů (Qualifiers – viz. dále)
- Volitelně
  - Jméno (@Named)
  - Množina stereotypů (Stereotypes)
  - Množina interceptor vazeb (Interceptor bindings)

redhat

Příklad CDI Beany

```
@RequestScoped
public class Car {

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

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

redhat

Příklad 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;
```

redhat

Životní cyklus

- Inversion of control – o životní cyklus se stará kontejner

- **Vytvoření nové instance** (viz. constructor injection)
- Field injection (naplnění atributů objektu)
- @PostConstruct / @Inject initializer
- **Uložení do kontextu** (scope)
- @PreDestroy / @Disposer method (viz. dále)

redhat

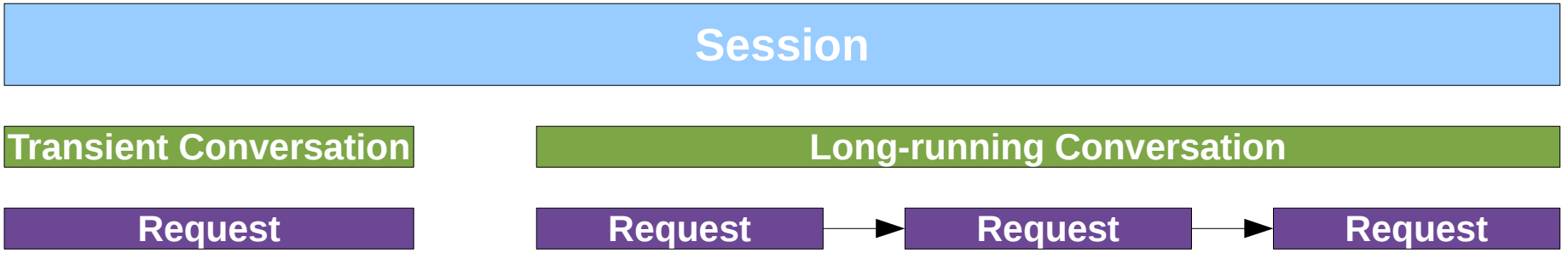
Kontexty - Scopes

- Servlet
  - Request Scope- @RequestScoped
  - Conversation Scope - @ConversationScoped
  - Session Scope - @SessionScoped
  - Application Scope - @ApplicationScoped
- Dependent Scope - @Dependent
  - Vždy se vytváří nová instance
  - Její životní cyklus je svázaný s Beanou která ji požaduje
- EE
  - @TransactionScoped (JTA)
  - @ViewScoped / @FlowScoped (JSF)

redhat

Conversation scope

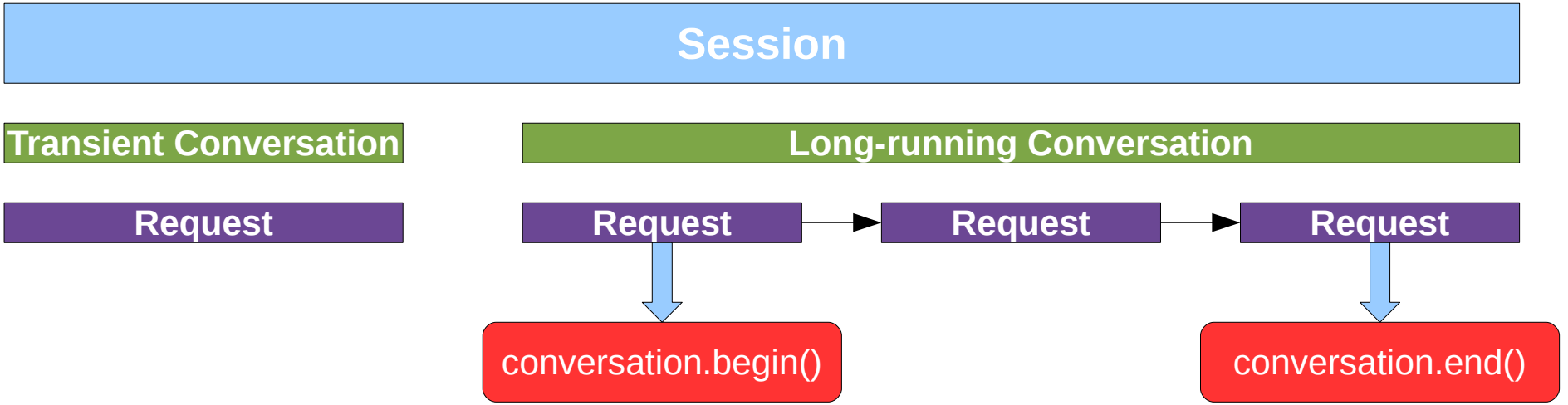
- Uchovává stav sekvence po sobě jdoucích logicky souvisejících requestů
- Samotná konverzace má 2 stavy
  - **Transient** – jeden Servlet request
  - **Long-running** – sekvence Servlet requestů



redhat

Ovládaní konverzací

```
public class NewAuctionWizzard {

    @Inject
    private Conversation conversation;
```



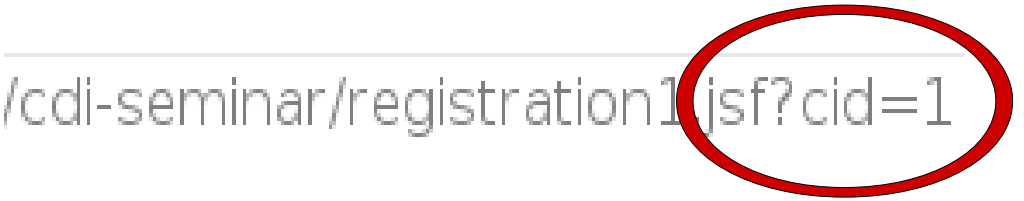
redhat

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();
}
```

Propagace konverzací

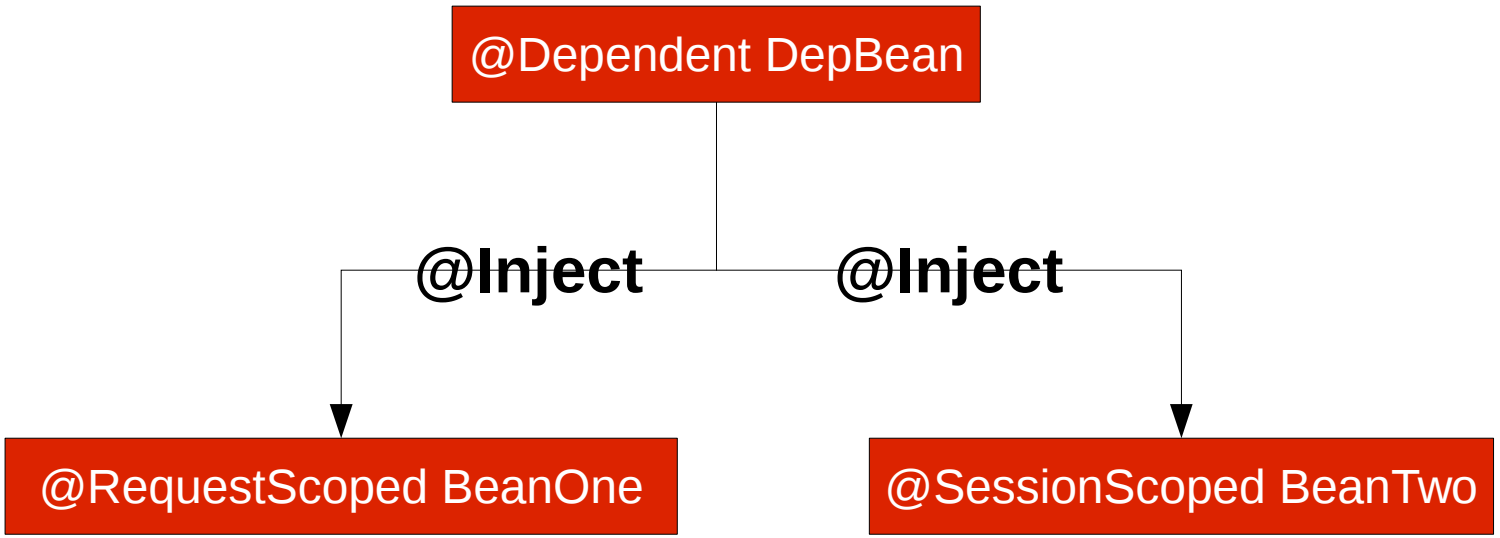
- String identifier
  - Set by application
  - Generated by container
- V rámci session
- Propagace pomocí **cid** parametru
  - Automatická propagace při odesílaní JSF formuláře
  - Manuální při plain Servlet request



redhat

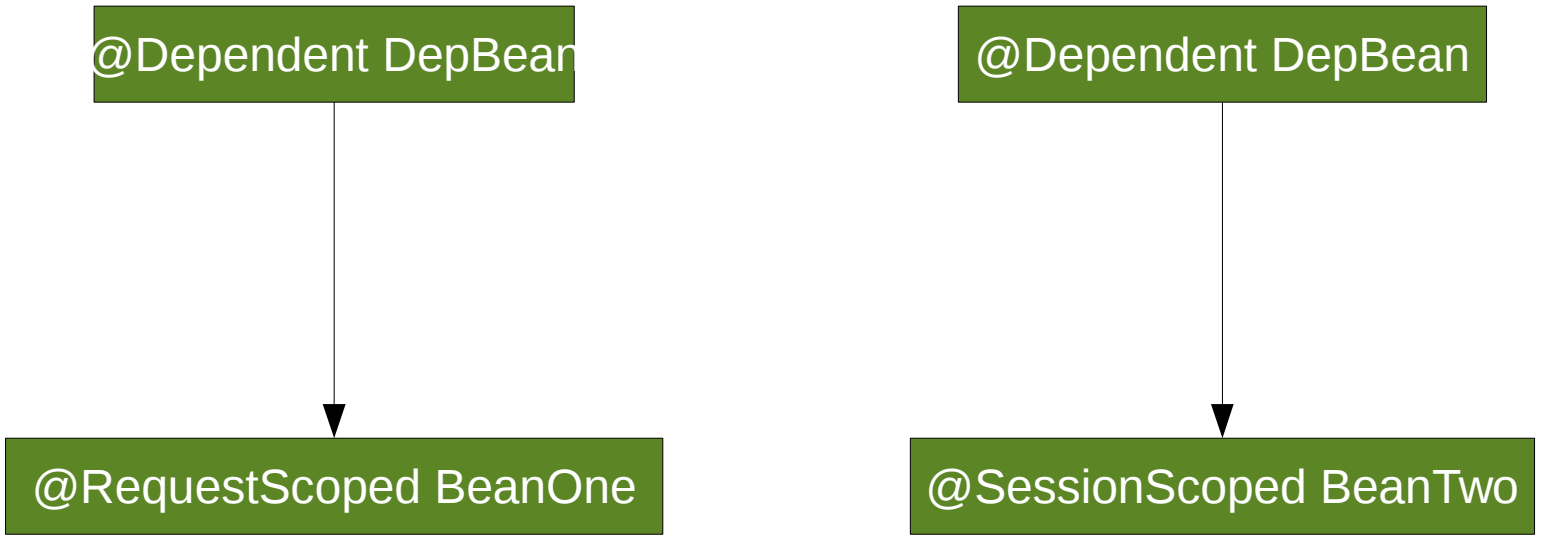
Dependent Pseudo Scope

- Na úrovni tříd



Dependent Pseudo Scope

- Na úrovni objektů



redhat

Passivation

- Velikost paměti potřebné pro uložení sessions roste lineárně s počtem existujících sessions
- Session a Conversation – passivating scopes
- Odkladaní neaktivních sessions na sekundární úložiště
- Replikace sessions
  - Load balancing
  - Failover
- Každá @SessionScoped a @ConversationScoped komponenta musí být serializovatelná
  - java.io.Serializable
  - java.io.Externalizable

redhat

Dependency Injection

- Mechanismus získavaní závislostí
- Spouští se explicitně pomocí anotace @Inject

```
@RequestScoped
public class Car {

    @Inject
    private Driver driver;
```

redhat

Pravidla DI – Typesafe resolution

- Výběr na základě
  - Typů
  - Kvalifikátorů (Qualifiers)
    - Slouží primárně na odlíšení více instancí daného typu
- Na dependency injection se použije objekt daného typu
  - Z kontextu (pokud objekt existuje v kontextu)
  - Nový
    - vytvoří jej container
    - při @Dependent se vytváří vždy nový objekt

redhat

Qualifiers

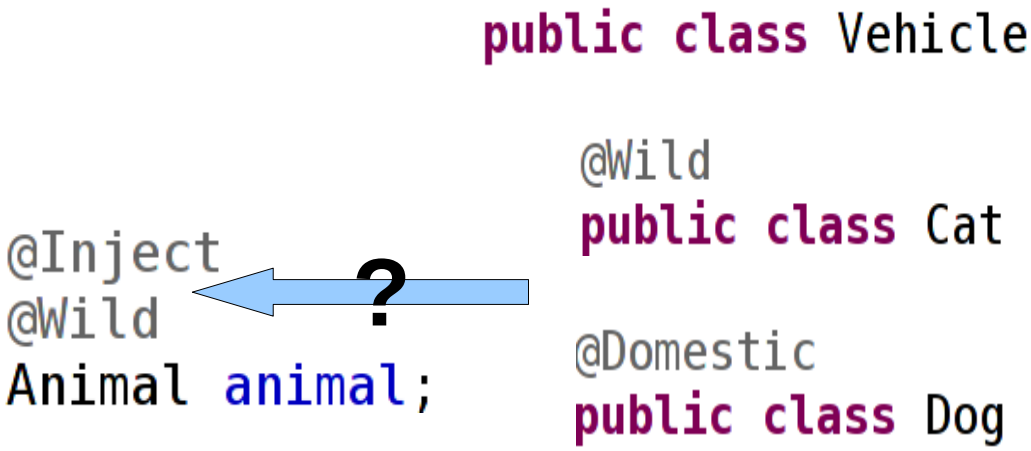
```
@Qualifier
@Retention(RetentionPolicy.RUNTIME)
public @interface LoggedIn {

}
```

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

redhat

Pravidla DI – Typesafe resolution

- Injection point X vyžaduje
  - Typ (Z)
  - Množinu kvalifikátorů
- Bean Y poskytuje
  - Množinu typů
  - Množinu kvalifikátorů
- Bean Y je vhodným kandidátem pokud
  - Vyžadovaný typ se nachází v množině poskytovaných typů
  - Všechny požadované kvalifikátory se nachází v množině poskytovaných kvalifikátorů



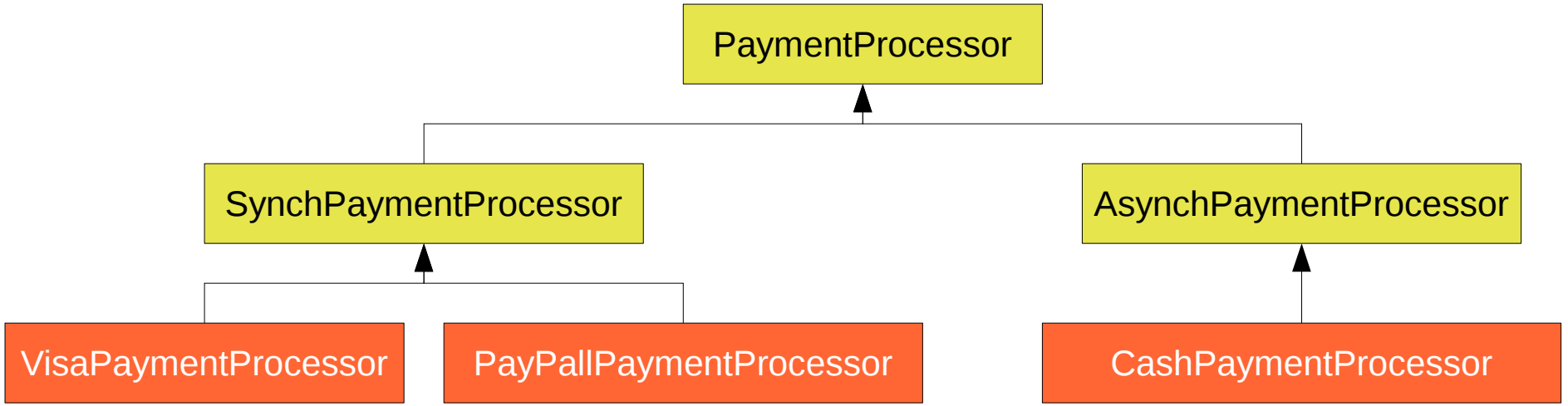
redhat

Poskytované typy (Bean types)

- Tranzitivní uzávěr rozšiřovaných a implementovaných typů
- java.lang.Object je vždy typem
- java.io.Serializable není nikdy typem

redhat

Příklad typové rezoluce (typy)




```
@Inject
private PaymentProcessor pp;

@Inject
private SynchPaymentProcessor spp;

@Inject
private VisaPaymentProcessor vpp;
```

redhat

Příklad typové rezoluce (kvalifikátory)

```
@Inject
@Any
private Animal animal;

@Inject
@Domestic
private Animal domestic;

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


```
public interface Animal

public class Sheep

@Domestic
public class Dog

@Wild
@Hungry
public class Lion

@Wild
public class Elephant

@Domestic
@Lazy
public class Cat
```

redhat

Speciální kvalifikátory

- @Default – implicitní kvalifikátor v případě, že daná komponenta / injection point nedefinuje jiný kvalifikátor
- @Any - implicitní kvalifikátor každé komponenty
- @Named

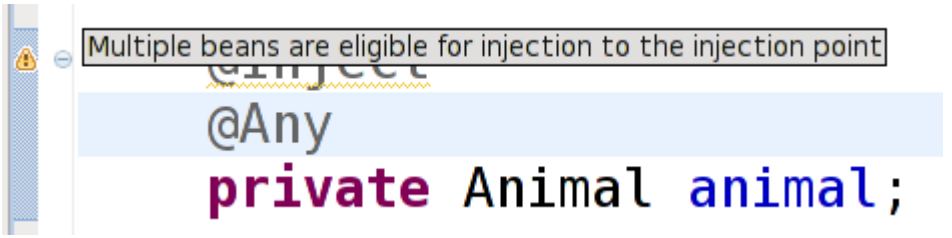
```
public class Shelter {

    @Inject
    Animal animal1;

    @Inject @Default
    Animal animal2;

    @Inject @Any
    Animal animal3;
}
```

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

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

Pravidla DI – Typesafe resolution

- 1 injection point = právě jedna vhodná Bean
- Kontrola při deploy (nebo tooling)



redhat

Typy DI

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

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

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

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

redhat

Producer method

- Factory - pro případy kdy volaní konstruktoru třídy nestačí na vytvoření objektu
- Rozšírená kontrola
  - Výsledek JPA dotazu
  - Náhodné číslo
  - ...
- Konkrétní metoda (není abstraktní)
  - Umístěná na CDI Bean
  - Tranzitivní uzávěr typu návratové hodnoty = typ objektu
  - Kvalifikátory, Scope a jméno objektu se definují na metodě

redhat

Producer method

```
@ApplicationScoped
public class RandomNumberGenerator {

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

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

redhat

Producer method

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

redhat

Producer field

- Zjednodušení producer metody

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

redhat

Disposer method

- Některé objekty vyžadují explicitní zničení
- @PreDestroy nefunguje na produkovaných objektech

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

redhat

Alternatives

- @Alternative
- Nahrazení komponenty jinou na základě prostředí
  - Konfigurace
  - Testovaní
- Alternativa musí být explicitně aktivovaná (enabled) pomocí anotace **@Priority**
- Rozšíření **Typesafe resolution** algoritmu
  1. Výběr vhodných kandidátů na základě typu a kvalifikátorů
  2. V případě, že existuje víc jak jeden kandidát jsou eliminovaní všichni kandidáti kromě aktivovaných alternatives
  3. V případě, že všichni kandidáti mají definovanou prioritu, je vybraný ten s nejvyšší prioritou

redhat

Integrace s prezentační vrstvou

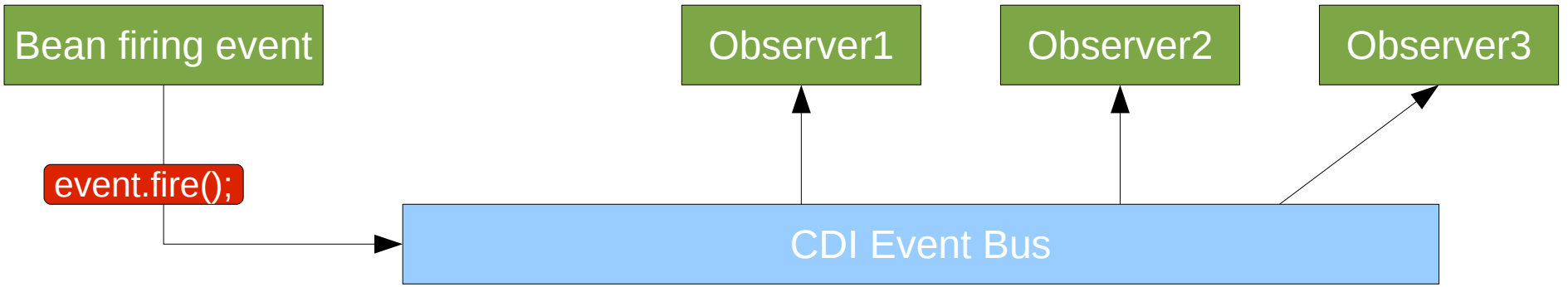
- Přístup k CDI komponentám pomocí jména
- @Named

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

redhat

Události (Events)

- Komunikace pomocí odesílání zpráv
- Oddělení producentů zpráv od jejich konzumentů
- Loose coupling – další observery můžeme přidat později bez zásahu do producenta událostí
- Synchronní (Asynchronní k dispozici v CDI 2.0)
- Event = message (payload)



redhat

Události (Events) – Odesílání zprávy

- In-built Event bean

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

- Vyvolání události

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

redhat

Vlastnosti zprávy

- Typ zprávy
- Qualifiers

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

redhat

Události (Events) – Doručení zprávy

- Observer method
- Pravidla typové rezoluce stejné jako pro DI

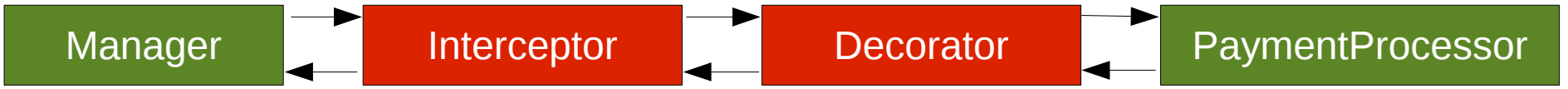
```
public void log(@Observes User user) {
    System.out.println("Hello " + user.getName());
}
```

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

redhat

Dekorátory a Interceptory

- Aspect-oriented programming
- Odchytávaní volaní metod
- Izolovaní kódu, který bychom jinak opakovali v metodách
- Změna chování v závislosti na prostředí



redhat

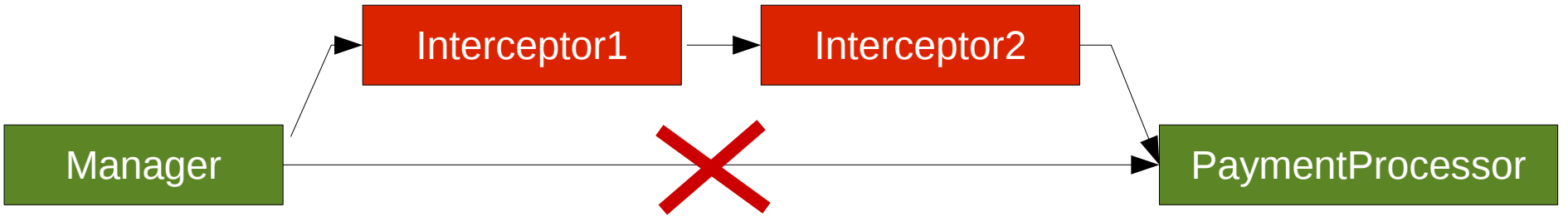
Interceptory

- Oddělení technických detailů od logiky aplikace
- Cross-cutting concerns (logovaní, zabezpečení, caching)
- Můžu odchytávat
  - Volaní metod (@AroundInvoke)
  - Životní cyklus
    - @AroundConstruct
    - @PostConstruct
    - @PreDestroy

redhat

Interceptory

```
public class PaymentManager {

    @Inject
    private PaymentProcessor pp;

    pp.foo();
```



redhat

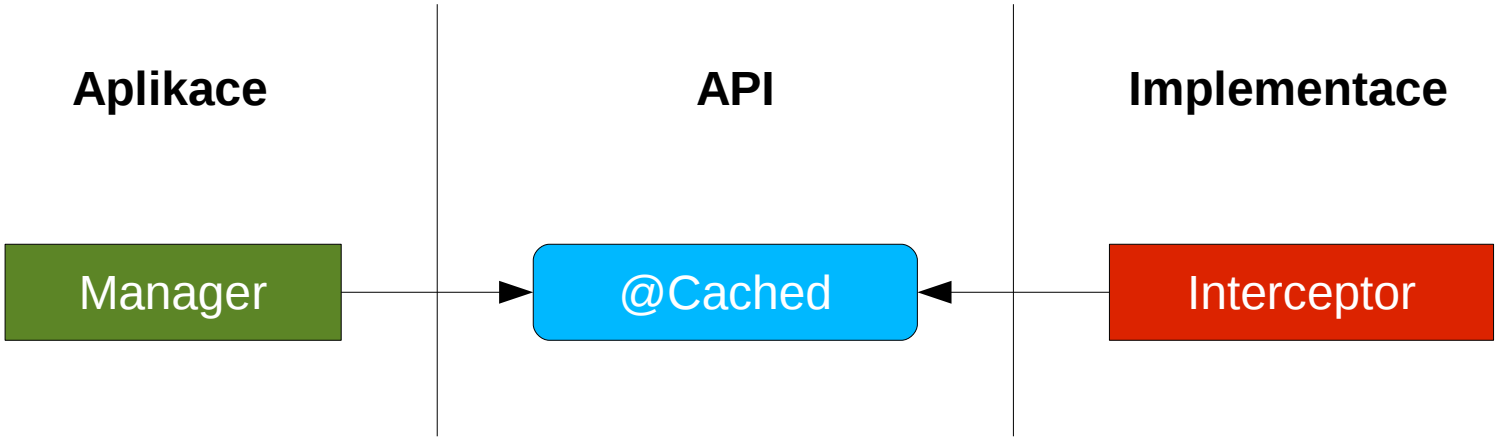
Interceptory

```
@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

- Anotace reprezentující funkcionalitu poskytovanou interceptorem (loose coupling)

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

}
```



redhat

Decorator

- “Typovaný” interceptor – “pozná” sémantiku volaní které obaluje
- Není nutné anotovat odchytávaný objekt
- Implementuje přímo metody rozhraní které obaluje
  - Nemusí implementovat všechny metody rozhraní (abstraktní třída)
- Speciální injection point (**@Delegate**)

redhat

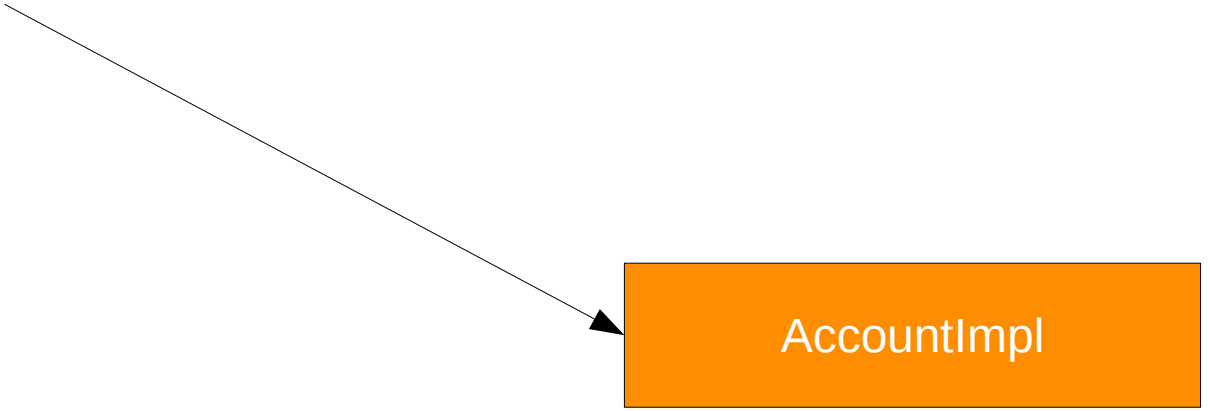
Decorator

```
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);
    }
}
```

redhat

Decorator

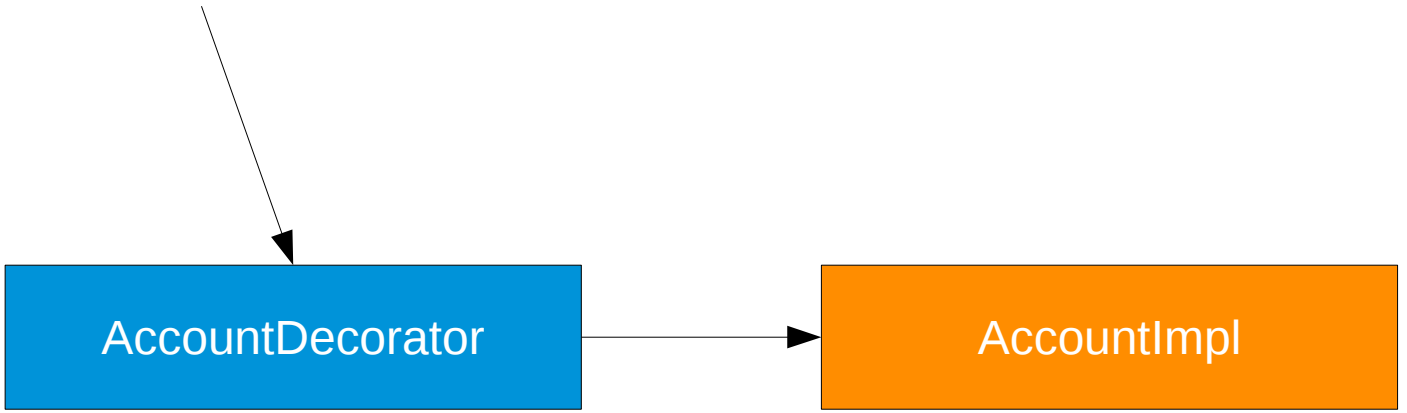
```
@Inject
private Account account;

account.withdraw(BigDecimal.TEN);
```



redhat

Decorator

```
@Inject
private Account account;

account.withdraw(BigDecimal.TEN);
```



redhat

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);
    }
```

redhat

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);
    }
```

redhat

Aktivace a pořadí

- @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)

redhat

Stereotypy

- Obrana před “annotation hell”
- Anotace sdružující
  - Scope
  - Interceptor bindings
  - @Named
- Vestavěný stereotyp **@Model**

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

redhat

Integrace

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

redhat

Integrace – 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

redhat

Contexts and Dependency Injection

- Vlastnosti CDI komponent
- Životní cyklus (Scopes)
- Dependency Injection
- Události (Events)
- Dekorátory a Interceptory (Decorators and Interceptors)
- Integrace

redhat

Tipy na závěr

- Nezapomenout **scope annotation**
- Pozor na správné 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;

redhat

Enterprise JavaBeans

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

redhat

Enterprise JavaBeans

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

redhat

Enterprise JavaBeans 3.1

- Zjednodušení existujícího komponentového modelu
  - Rozhraní (views) nejsou povinná
  - Session beans jsou automaticky CDI komponentami
    - Scopes (Stateful session beany)
    - Dependency injection
    - Events
    - Interceptory / Dekorátory
  - Možnost používat v jednoduché webové aplikaci (.war)
- **@Singleton** session bean
- Asynchronní volaní metod

redhat

Singleton session bean

- Jediná instance sdílená v rámci celé aplikace
  - V rámci jedné JVM
- V porovnaní s **@ApplicationScoped** navíc
  - Inicializovaná při startu aplikace (**@Startup**)
  - Podporuje paralelní přístup
    - @Lock(LockType.READ)
    - @Lock(LockType.WRITE)
    - @AccessTimeout
    - @ConcurrencyManagement
- DEMO na cvičení

redhat

Asynchronní volaní metod

- Abstrakce od přímého použití vláken, exekutorů a thread pools
- Použití
  - Asynchronní vykonávaní dlouhotrvající úlohy
    - Odesílání potvrzujícího e-mailu
    - Návratová hodnota **void**
  - Paralelizace výpočtu
    - Návratová hodnota **Future<V>** - **AsyncResult<V>**
- **@Asynchronous**
- DEMO na cvičení

redhat

Enterprise JavaBeans 3.2

- Převážně kosmetické změny
  - @Stateful(passivationCapable = false)
  - @PostConstruct/@PreDestroy volitelně transakční
  - TimerService.getAllTimers()
    - Všechny Timers pro daný modul

redhat

Odkazy

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

redhat

CDI 2.0 – přehled novinek

- **SE Bootstrap**
- Manuální aktivace kontextu
- Řazení observerů
- **Asynchronní události**
- Configurators SPI
- Interceptory pro producery
- A další...

redhat

CDI 2.0 – SE Bootstrap

- CDI je možné použít v Java SE
- Start/stop CDI kontejneru
  - Extends AutoCloseable
- Volitelné, co je na classpath
  - Tzv. synthetic bean archive
- Dostupné scopes – Application, Dependent (, Request)
- beans.xml je vyžadováno
- Krátké demo na cvičeních

redhat

CDI 2.0 – Asynchronní události

- Opět z rozhraní Event, pomocí metody fireAsync(…)
- Observer musí být speficky monitorovat asynchronní události
  - @ObservesAsync @MyQualifier Payload payload
- Nelze mít observer pro sync i async události zároveň
- Demo na cvičeních

	callMe(@Observes payload)	callMe(@ObservesAsync payload)
event.fire(payload)	Sync. call	**Not notified**
event.fireAsync(payload)	**Not notified**	Async call

manovotn@redhat.com | www.redhat.com