EJB

@Resource

- JNDI resource je možné získat pomocí anotace @Resource. Například Wildfly má DataSource (H2) s JNDI názvem:
  - `java:jboss/datasources/ExampleDS`
- Pro získání tohoto datasource přidejte atribut:

```
@Resource(name = "java:jboss/datasources/ExampleDS")
private DataSource dataSource;
```

Jak přidat Oracle JDBC ovladač do WildFly I.

- 1. stáhněte ojdbc7.jar do:
  - `modules/system/layers/base/com/oracle/main`
- Ve stejném adresáři vytvořte module.xml:

```
<module xmlns="urn:jboss:module:1.3" name="com.oracle">
  <resources>
    <resource-root path="ojdbc7.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

Jak přidat Oracle JDBC ovladač do WildFly III.

- Do `standalone/configuration/standalone.xml` přidejte:

```
<driver name="oracle" module="com.oracle">
  <driver-class>oracle.jdbc.OracleDriver</driver-class>
</driver>
```

Jak přidat Oracle JDBC ovladač do WildFly II.

```
<datasource jndi-name="java:jboss/datasources/hr" pool-name="OracleDS"
enabled="true">
  <connection-url>jdbc:oracle:thin:@localhost:1521:xe</connection-url>
  <driver>oracle</driver>
  <pool>
    <min-pool-size>1</min-pool-size>
    <max-pool-size>5</max-pool-size>
    <prefill>true</prefill>
  </pool>
  <security>
    <user-name>hr</user-name>
    <password>hr</password>
  </security>
</datasource>
```

Poznámka: Také by šlo přidat pomocí jboss-cli:
https://docs.jboss.org/author/display/AS71/CLI+Recipes

CDI (Contexts and Dependency Injection)

- CDI je náhrada EJB. Pomocí CDI je možné injectovat jakoukoli Java beanu. U CDI je důležitá anotace @Inject. Pokud injectovaná beana nemá žádný scope, pak je scope „dependent pseudo scope“ (podobné jako u Springu „prototype“):
  - http://stackoverflow.com/questions/10293510/what-is-the-default-scope-of-a-named-cdi-bean
- Lepší je nastavit nějaký scope:
  - @RequestScoped, @SessionScoped, @ApplicationScoped, @Dependent, @ConversationScoped (JSF)
  - http://docs.oracle.com/javaee/6/tutorial/doc/gjbbk.html

@Named

- Když se CDI používá ve spojení s JSF, pak se ještě setkáte s anotací @Named, která pojmenovává beanu pro potřeby EL (Expression Language) v JSF.

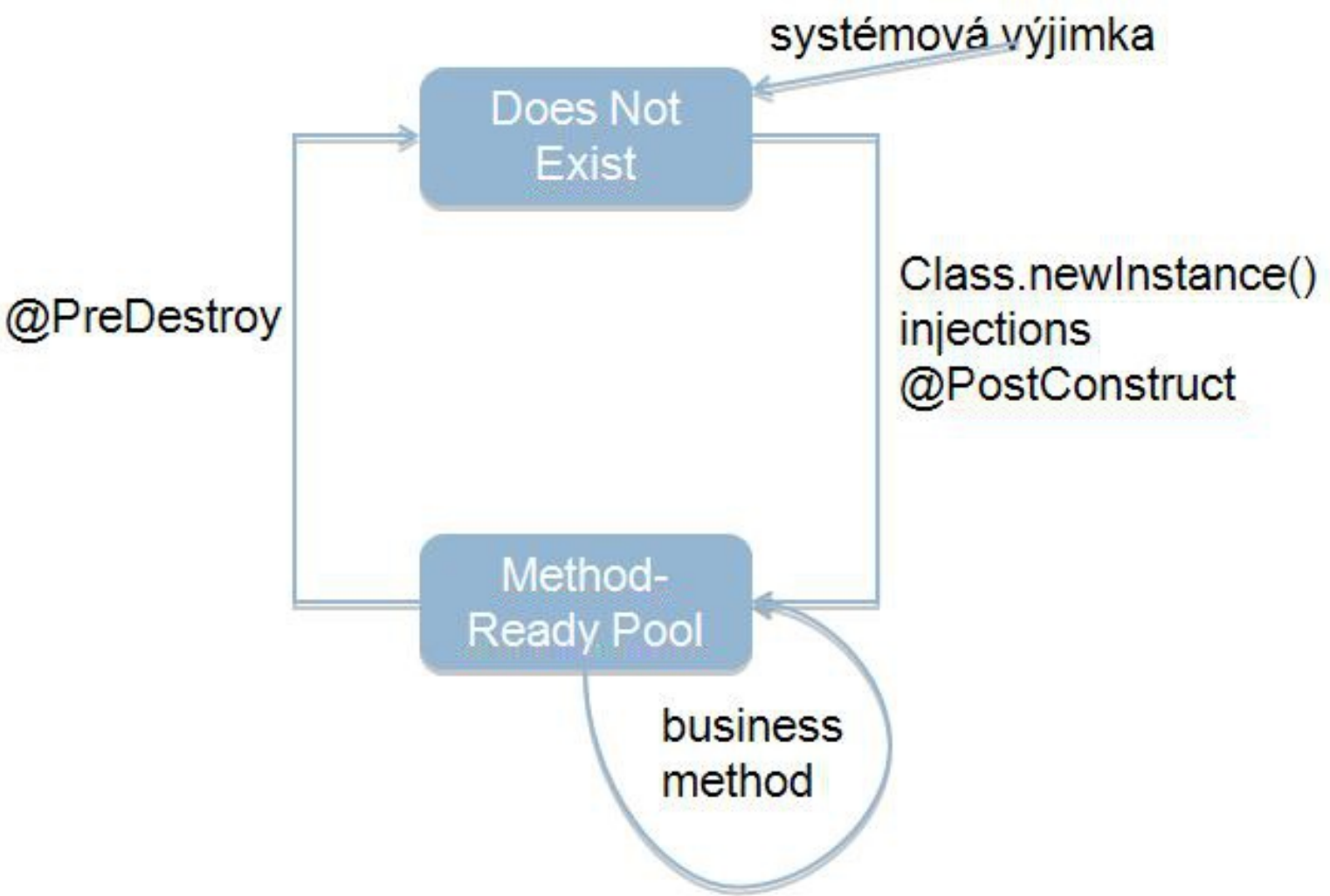
EJB

- EJB jsou třídy, které se používají pro tvorbu servisní vrstvy.
- Pro běh EJB je nutné mít aplikační server, ve webovém kontejneru (Tomcat, Jetty) fungovat nebudou!
  - Poznámka: Existuje projekt TomEE, kde je Tomcat + OpenEJB: http://tomee.apache.org/
- EJB jsou použitelné od EJB 3.0 (Java EE 5), kde přibyla možnost konfigurace EJB pomocí anotací.
- Jako alternativa EJB vznikl framework Spring, který umožňuje to samé a daleko víc (navíc k běhu Springu není zapotřebí aplikační server).

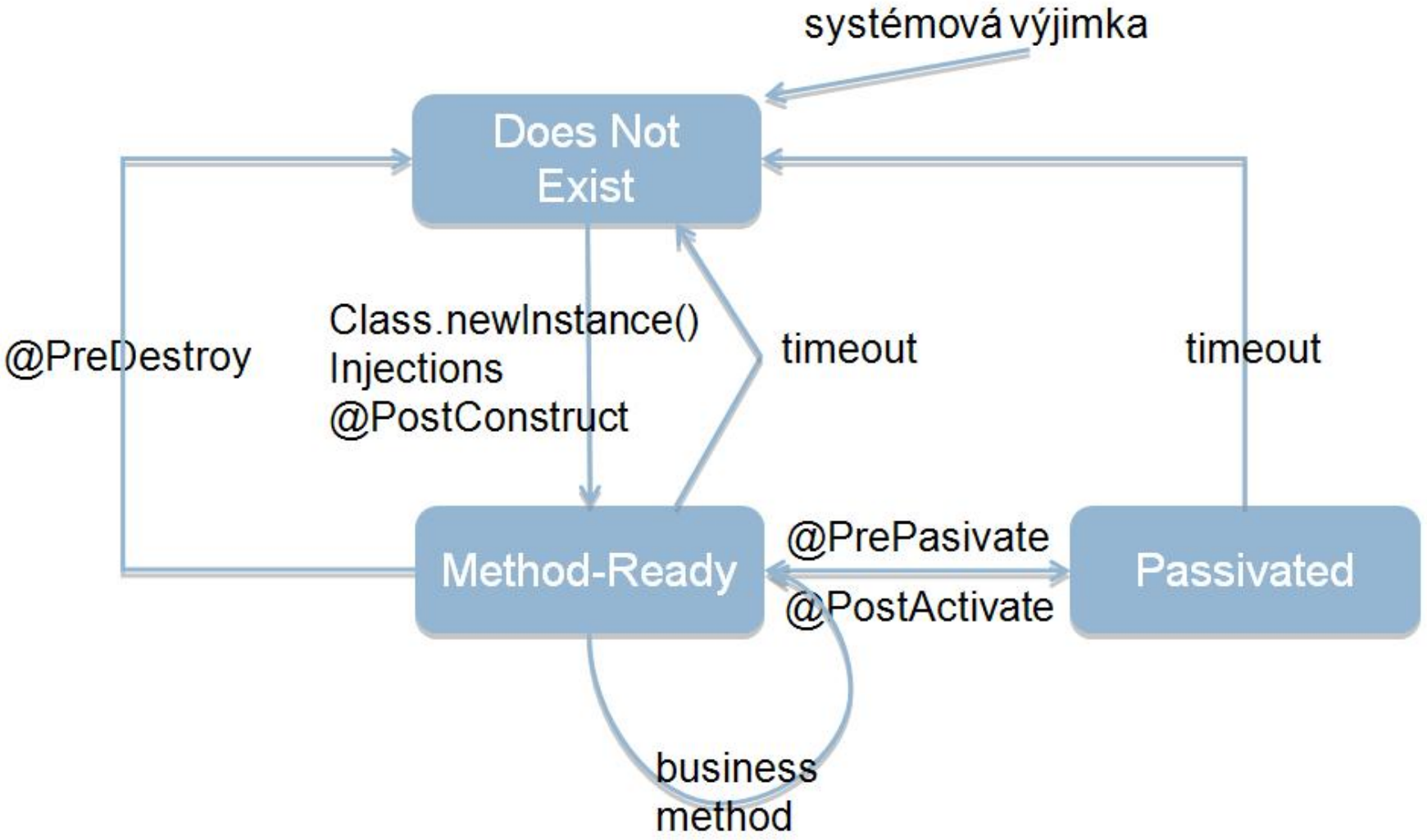
Typy EJB

- **Session beans**
  - **Stateless (@Stateless)**: neuchovávají stav klienta mezi obsluhou jeho jednotlivých požadavků. Není vhodné, aby tyto beany měly atributy, protože není zaručené, že se jejich obsah při dalším volání nezmění.
  - **Stateful (@Stateful)**: uchovávají stav klienta mezi obsluhou jeho jednotlivých požadavků v rámci jedné session klienta. Pro každého unikátního klienta se vytvoří v paměti na serveru samostatná instance stavové session beany, která patří pouze tomuto klientovi a při dalším volání metod se použije vždy stejný objekt obsahující data z předchozí interakce klienta.
  - **Singleton (@Singleton)** (od EJB 3.1)
- **Message Driven Beans**
  - Pro práci s JMS zprávami, pomocí kterých mohou komunikovat Java systémy.

Životní cyklus stateless bean



Životní cyklus stateful bean



Bean

- Bean musí mít interface s anotací `@Local` nebo `@Remote`:

Poznámka: @Local znamená, že je beana lokálního typu (běží v jedné instanci JVM). @Remote znamená, že může běžet na více JVM – například v distribuovaném prostředí

```
@Local
public interface ItemService { /* metody beany ... */ }
```

- Implementace beany musí mít anotaci `@Stateless`, `@Stateful`, nebo `@Singleton`:

```
@Stateless
public class ItemServiceImpl implements ItemService {

  @PersistenceContext
  private EntityManager entityManager;
  /* metody beany ... */
}
```

Bean bez interface

- Pokud nechcete vytvářet interface, pak od EJB 3.1 můžete použít:

```
@Stateless
@LocalBean
public class ItemServiceImpl implements ItemService {

  @PersistenceContext
  private EntityManager entityManager;
  /* metody beany ... */
}
```

Získání instance EJB

- Pomocí anotace @EJB:

```
@EJB

private ItemService itemService;
```

- Pomocí JNDI:

```
ItemService itemService =
  (ItemService)
  new InitialContext().lookup("ItemServiceImpl");
```

Všimněte si, že se získává interface – implementaci doplní EJB kontejner

Název třídy s anotací @Stateless / @Stateful / @Singleton

@Startup

- Můžete mít Singleton bean s metodou s anotací `@PostConstruct`, která se zavolá po konstrukci Singleton objektu, který se vytvoří při startování webové aplikace:

```
@Startup @Singleton
public class InitDbServiceImpl {

   @PersistenceContext private EntityManager entityManager;

   @PostConstruct
   public void init() {
      System.out.println("*** START INIT DATABASE ***");
      // TODO ...
      System.out.println("*** FINISH INIT DATABASE ***");
   }
}
```

Transakce

- Všechny metody EJB jsou automaticky transakční.
- Pokud zavoláte jednu EJB metodu a ta zavolá další, pak se nevytváří další transakce, ale rozšiřuje se stávající. Toto chování můžete změnit anotací `@TransactionAttribute`:
  - http://docs.oracle.com/javaee/7/tutorial/doc/transactions003.htm
- V případě, že se z transakční metody vyhodí výjimka typu `RuntimeException` (nebo její potomek), pak se provede rollback celé transakce. V opačném případě se provede commit.