

HEIG-VD / TIC

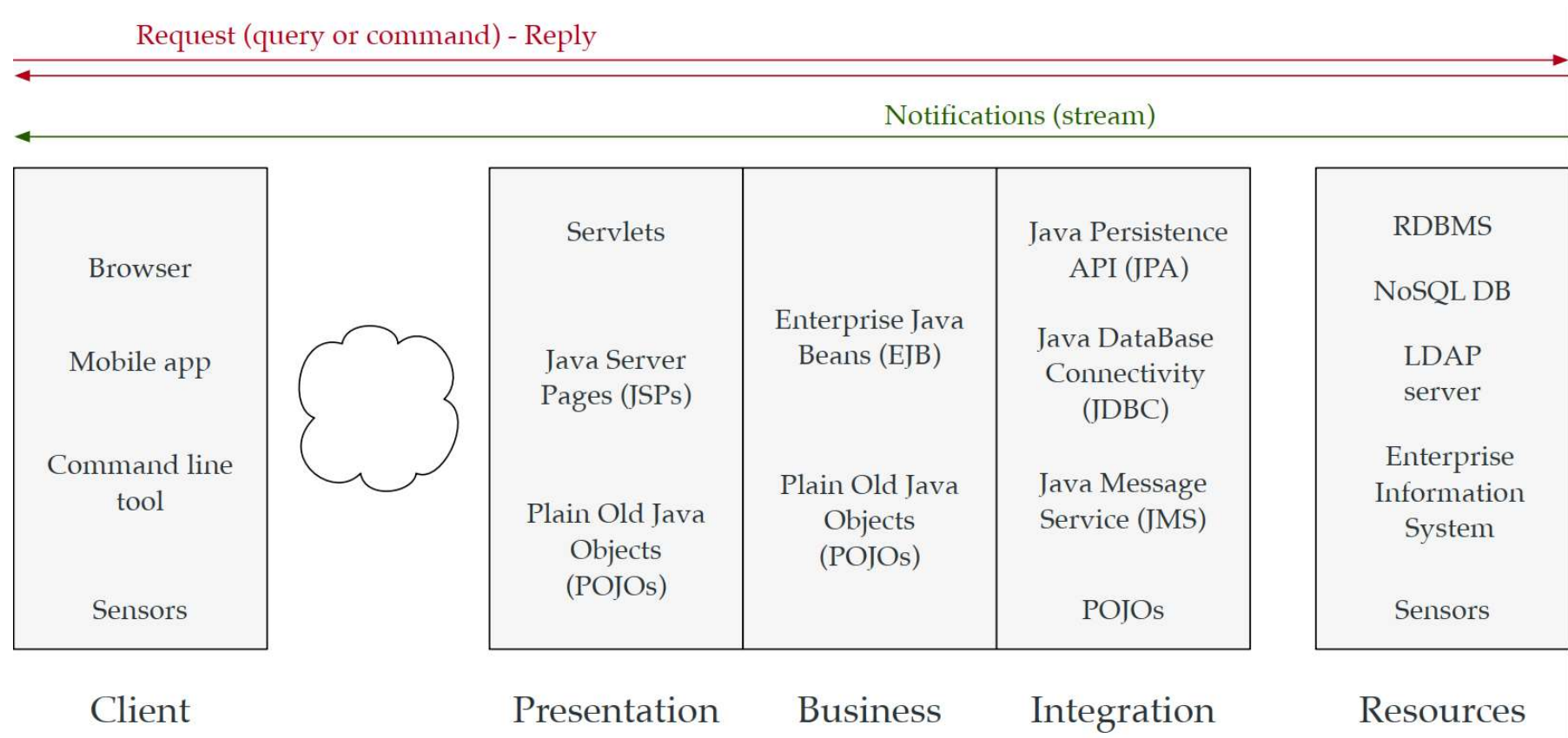
Applications multi-tiers

Tiers Business, Tiers Intégration

Entreprise Java Bean, Data Access Object

Injection de dépendance

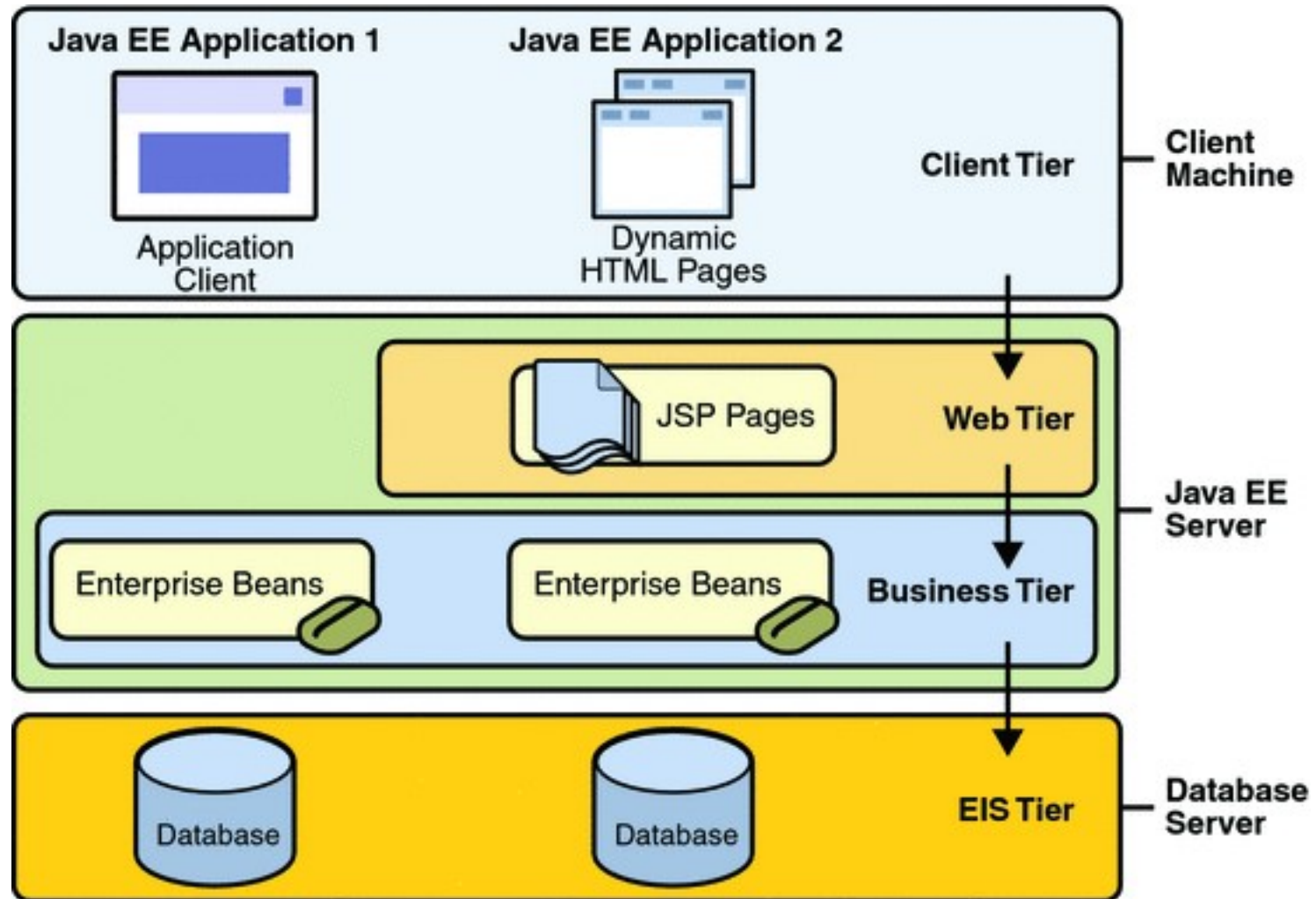
Rappel



Tiers Business / Intégration

- Business / Intégration
 - *Ne connaît pas l'interface utilisateur*
 - Logique service
 - Business : appliquer la logique métier, contrôle l'intégration
 - Intégration : fournir et obtenir des données
- Exemple
 - Intégration : obtenir une voiture en cours de fabrication, créer un moteur, des roues
 - Business : assembler le tout dans la voiture
 - Intégration : enregistrer la voiture modifiée
 - Intégration : déclarer l'opération dans une autre appli
- Un seul composant dans les cas simples
 - Pas de données partagées, crud
- Implémentation
 - Objets java standards (POJO)
 - Entreprise Java Bean (EJB)

Conteneurs Java EE



Entreprise Java Bean (EJB)

- Conteneur d'EJB, séparé du conteneur Web
 - Cycle de vie des composants business
 - Pool, intercepteurs, transactions ...
- Une des spécifications de Java EE
- Evolution de la spécification
 - EJB 3.0 / Java EE 5 (2006), EJB 3.1 / Java EE 6 (2009)
 - Moins de configuration (annotations)
 - Abandon progressifs des EJB Entité → JPA, EJB Lite
 - Performances
- Packaging historique : archive séparée .jar
 - .war et .jar → .ear (entreprise archive)
 - .war et .jar déployés séparément
- Les EJB peuvent exposer des interfaces locales ou remotes
 - Implémentation d'interfaces annotées @Local et @Remote
 - Si seulement interface locale, optionnelle depuis Java EE 6
- Les EJB peuvent être packagés dans le .war (Servlet 2.5, Java EE 5)
 - WEBINF/classes ou WEBINF/lib

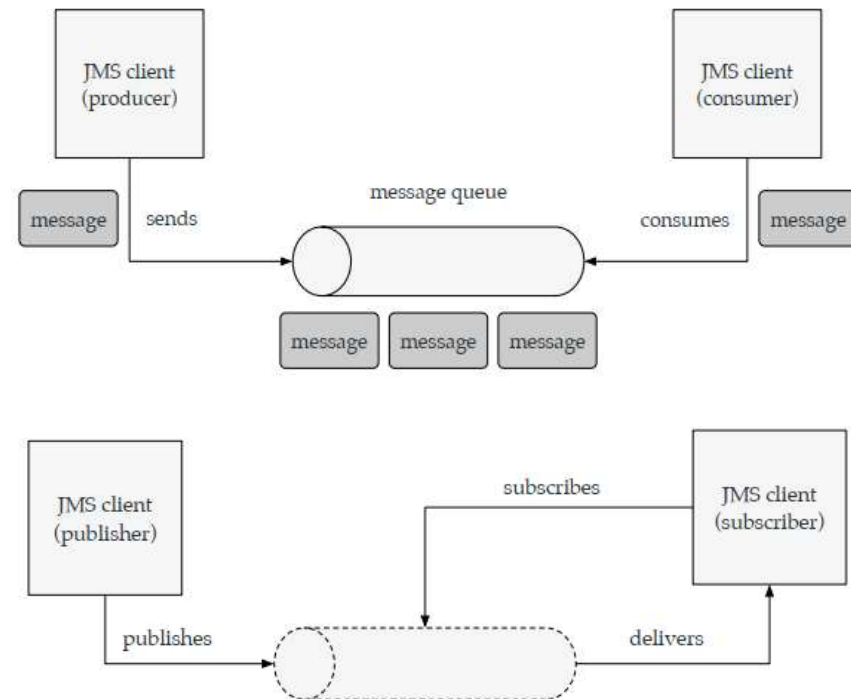
Entreprise Java Bean (EJB)

- Stateless Session Bean (SLSB)
 - Pas d'état : pas de données conservées entre 2 appels de méthode
 - Annotation @Stateless
- Singleton Session Bean
 - Session bean pour données partagées (cache, ...)
 - Annotation @Singleton
- StateFull Session Bean (SFSB)
 - Données conservées entre appels de méthode (panier, ...)
 - Attention : performances et perte
 - Annotation @Statefull

Entreprise Java Bean (EJB)

Message Driven Bean (MDB)

- Java Message Service (JMS) spécification
- Communication en mode messages (queues)
- Communication en mode diffusion (publish-subscribe, topics)
- Communication systèmes tiers
- Méthode onMessage (Message message)
- Annotation @MessageDriven, @Resources
- Implémentations OpenMQ, ActiveMQ, IBM MQ, ...



Stateless Session Bean

- **@Stateless**

```
public class ContactService {
```

```
    @Inject
```

```
    private ContactMemory contactMemory;
```

```
    public List<Contact> getContacts() {  
        return contactMemory.getContacts();  
    }
```

```
    public void add(Contact contact) {  
        contactMemory.add(contact);  
    }
```

Les méthodes sont automatiquement exportées en local

- **@Local**

```
public interface ContactServiceLocal {
```

```
    List<Contact> getContacts();
```

```
    void add(Contact contact);
```

Interface @Local (ou @Remote) explicite

- **@Stateless**

```
public class ContactService implements ContactServiceLocal
```


Java Naming Directory Interface (JNDI)

- Obtention du bean : méthode 1, JNDI

```
@WebServlet("/liste")
public class Liste extends HttpServlet {
    private ContactService service;

    public void init() throws ServletException {
        super.init();
        try {
            Context ctx = new InitialContext();
            service = (ContactServiceLocal)
                ctx.lookup("java:module/ContactService");
        } catch (NamingException ex) {
            Logger.getLogger(Liste.class.getName())
                .log(Level.SEVERE, null, ex);
        }
    }
}
```

Injection de dépendance

Context & Dependency Injection (CDI)

- Obtention du bean : méthode 2, CDI

```
@WebServlet("/liste")  
public class Liste extends HttpServlet {  
  
    @Inject  
    private ContactService service;
```

- On peut aussi utiliser @EJB (pré-CDI)
- Attention : le conteneur ne peut injecter une ressource que dans un composant managé ou sous contrôle de CDI :
 - Ici un servlet : ok
 - Avant-dernier slide, un EJB : ok
 - Dans un POJO : non
 - C'est le conteneur qui crée votre objet et il en profite pour injecter

Singleton Session Bean

@Singleton

```
public class ContactMemory {
```

Partagé par tous les
beans : données de l'appli

```
    private List<Contact> contacts;
```

@PostConstruct

```
public void init() {
```

Activé par le conteneur juste
après la création du bean

```
    contacts = new ArrayList<>();
```

```
    contacts.add(new Contact("Pierre", 1234));
```

```
    contacts.add(new Contact("Sylvie", 1111));
```

```
}
```

@Lock(LockType.READ)

```
public List<Contact> getContacts() {
```

```
    return contacts;
```

```
}
```

Le conteneur gère les accès plus
finement que synchronize : il peut y avoir
plusieurs lecteurs simultanément

@Lock(LockType.WRITE)

```
public void add(Contact contact) {
```

```
    contacts.add(contact);
```

```
}
```

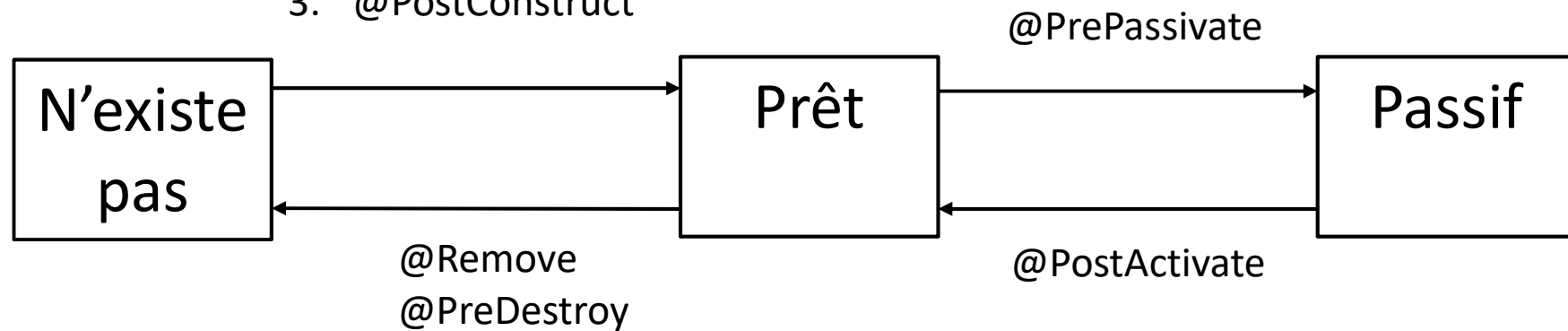
```
}
```

EJB : managed par un conteneur

- qui gère son cycle de vie
- qui intercepte les appels de méthode
 - AOP: Aspect Oriented Programming
- qui gère les besoins transversaux :
intercepteurs: transaction, sécurité, audit, ...
- qui gère un pool d'instances d'EJB
 - optimisation des performances
- EJB 3.2 / JSR 345

Cycle de vie EJB statefull

1. Création
2. Injection de dépendance
3. @PostConstruct



sans conteneur :

```
private Context ctxt;
public void transferMoney(Account a, Account b, double amount) {
// authentication concern
    if (!ctxt.isAuthenticated()) {
        throw new UnauthorizedException();
    }
// authorization concern
    Principal p = ctxt.getPrincial();
    if (!a.getOwner().equals(p)) {
        throw new UnauthorizedException();
    }
// audit concern
    AuditLog.getLog().record('transfer', p, a, b);
// tx concern
    ctxt.getTxManager().begin();
// business logic concern
    try {
        a.debit(a, amount);
        a.credit(b, amount);
    } catch (Exception e) {
// tx concern
        ctxt.getTxManager().rollback();
    }
// tx concern
    ctxt.getTxManager().commit();
}
```

avec conteneur :

```
private Context ctxt;  
@RolesAllowed("customer")  
@TransactionAttribute(REQUIRED);  
@Interceptors(AuditInterceptor.class);  
public void transferMoney(Account a, Account b, double amount) {  
    // authorization concern  
    Principal p = ctxt.getPrincial();  
    if (!a.getOwner().equals(p)) {  
        throw new UnauthorizedException();  
    }  
    // business logic concern  
    a.debit(a, amount);  
    a.credit(b, amount);  
}
```



Interception par proxy

Caused by: `java.lang.RuntimeException`: just kidding

```
at ch.heigvd.amt.lab1.services.CollectorService.submitMeasure(CollectorService.java:15)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:483)
at org.glassfish.ejb.security.application.EJBSecurityManager.runMethod(EJBSecurityManager.java:1081)
at org.glassfish.ejb.security.application.EJBSecurityManager.invoke(EJBSecurityManager.java:1153)
at com.sun.ejb.containers.BaseContainer.invokeBeanMethod(BaseContainer.java:4786)
at com.sun.ejb.EjbInvocation.invokeBeanMethod(EjbInvocation.java:656)
at com.sun.ejb.containers.interceptors.AroundInvokeChainImpl.invokeNext(InterceptorManager.java:822)
at com.sun.ejb.EjbInvocation.proceed(EjbInvocation.java:608)
at
org.jboss.weld.ejb.AbstractEJBRequestScopeActivationInterceptor.aroundInvoke(AbstractEJBRequestScopeActivationInter
ceptor.java:46)
at org.jboss.weld.ejb.SessionBeanInterceptor.aroundInvoke(SessionBeanInterceptor.java:52)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:483)
at com.sun.ejb.containers.interceptors.AroundInvokeInterceptor.intercept(InterceptorManager.java:883)
at com.sun.ejb.containers.interceptors.AroundInvokeChainImpl.invokeNext(InterceptorManager.java:822)
at com.sun.ejb.EjbInvocation.proceed(EjbInvocation.java:608)
at com.sun.ejb.containers.interceptors.SystemInterceptorProxy.doCall(SystemInterceptorProxy.java:163)
at com.sun.ejb.containers.interceptors.SystemInterceptorProxy.aroundInvoke(SystemInterceptorProxy.java:140)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:483)
at com.sun.ejb.containers.interceptors.AroundInvokeInterceptor.intercept(InterceptorManager.java:883)
at com.sun.ejb.containers.interceptors.AroundInvokeChainImpl.invokeNext(InterceptorManager.java:822)
at com.sun.ejb.containers.interceptors.InterceptorManager.intercept(InterceptorManager.java:369)
at com.sun.ejb.containers.BaseContainer.__intercept(BaseContainer.java:4758)
at com.sun.ejb.containers.BaseContainer.intercept(BaseContainer.java:4746)
at com.sun.ejb.containers.EJBLocalObjectInvocationHandler.invoke(EJBLocalObjectInvocationHandler.java:212)
... 34 more
```


Pool d'EJB

User: admin Domain: production Server: localhost

 payara server 

- Concurrent Resources
- Connectors
- JDBC
- JMS Resources
- JNDI
- JavaMail Sessions
- Resource Adapter Configs
- Configurations
 - default-config
 - server-config
 - Admin Service
 - Availability Service
 - Batch
 - Connector Service
 - Data Grid
 - EJB Container**
 - HealthCheck
 - HTTP Service
 - JVM Settings
 - Java Message Service
 - Logger Settings
 - MicroProfile
 - Monitoring
 - Network Config
 - Notification

Commit Option:

☒ **Option B - Cache a ready instance between transactions**
The container caches a ready instance between transactions, but the container does not ensure that the instance has exclusive access to the state of the instance's state by invoking ejbLoad from persistent storage at the beginning of the next transaction.

☐ **Option C - Do not cache a ready instance between transactions**
The container does not cache a ready instance between transactions, but instead returns the instance to the pool of available instances after a transaction.

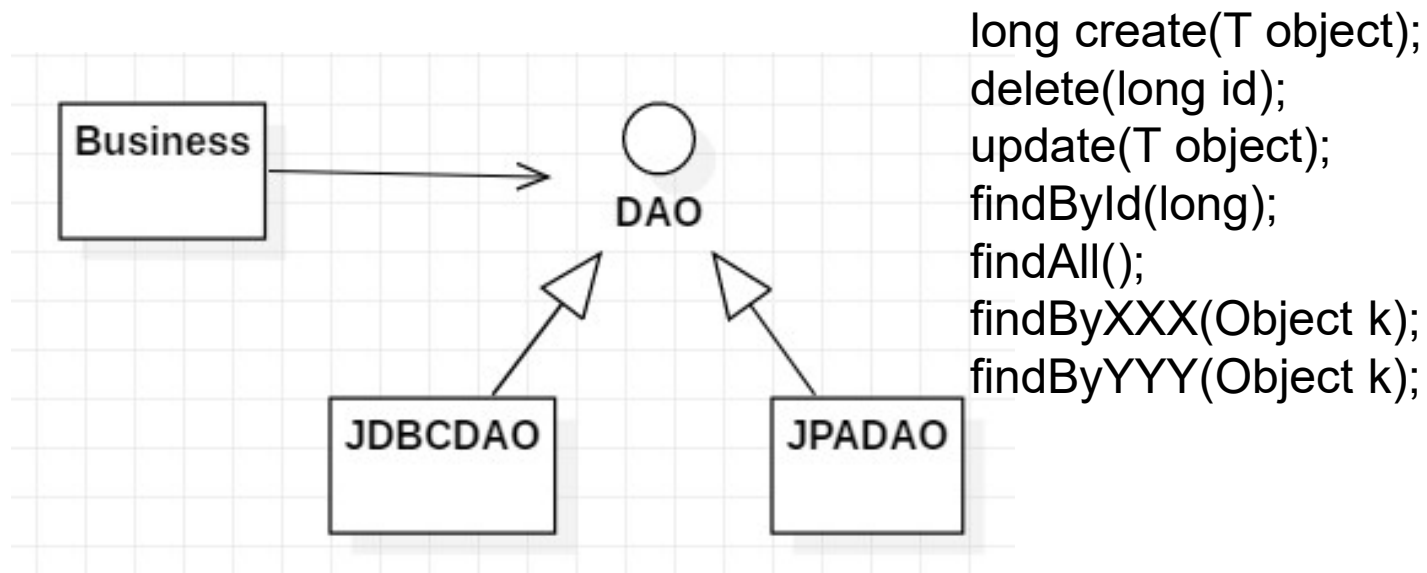
Pool Settings

Initial and Minimum Pool Size:	<input type="text" value="0"/>	Number of beans
Minimum and initial number of beans maintained in the pool		
Maximum Pool Size:	<input type="text" value="16"/>	Number of beans
Maximum number of beans that can be created to satisfy client requests		
Pool Resize Quantity:	<input type="text" value="8"/>	Number of beans
Number of beans to be removed when pool idle timeout expires		
Pool Idle Timeout:	<input type="text" value="600"/>	Seconds
Amount of time before pool idle timeout timer expires		
Limit Concurrent EJB Instances:	<input type="text" value=""/>	
Enable maximum allowable concurrent instances/threads for any particular stateless EJB		
Timeout to wait for EJB instance:	<input type="text" value="6000"/>	Milliseconds
In milliseconds, maximum time to wait for available EJB instance/thread. 0 (default) means indefinite.		

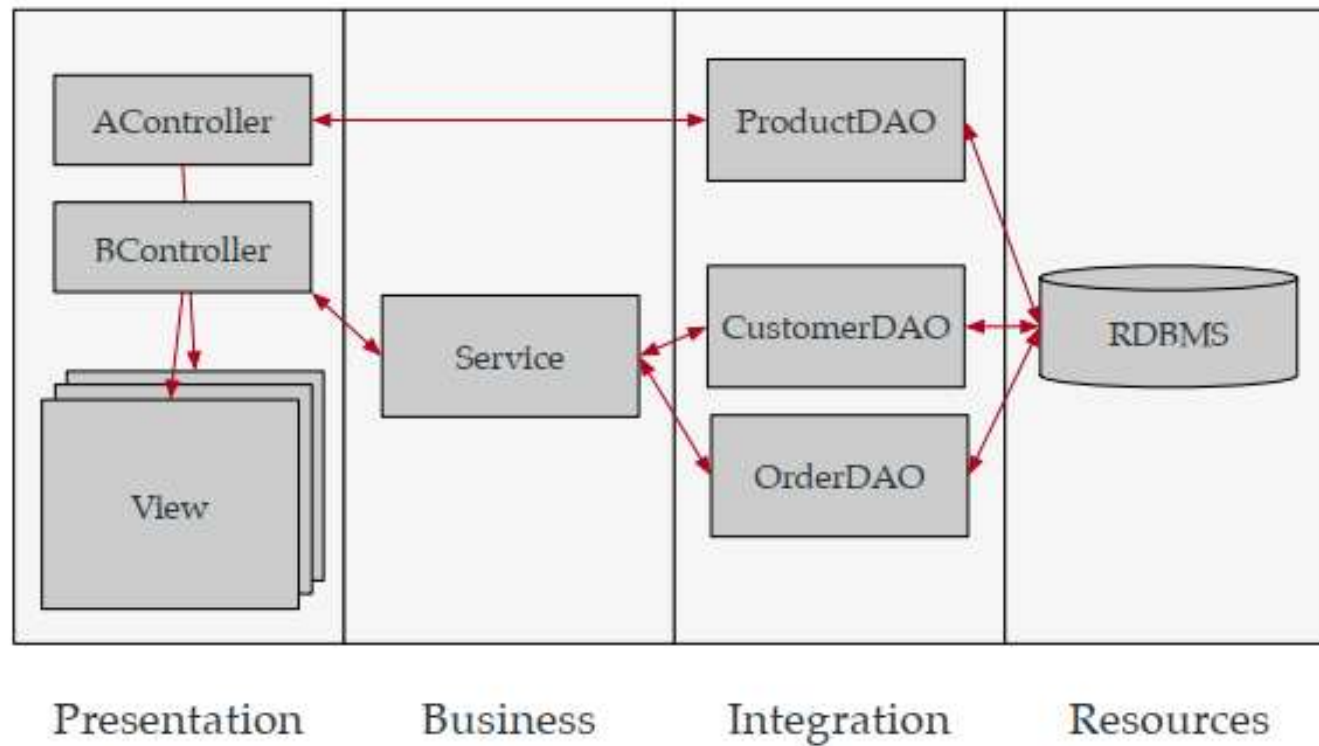
Tiers Intégration - Persistence

DAO : Data Access Object

- Le tiers Business ne doit pas se préoccuper de savoir comment les objets sont stockés : SGBDR, No-SQL, LDAP, file systems, ...
- Couche permettant de réduire le couplage entre le métier et le stockage des données



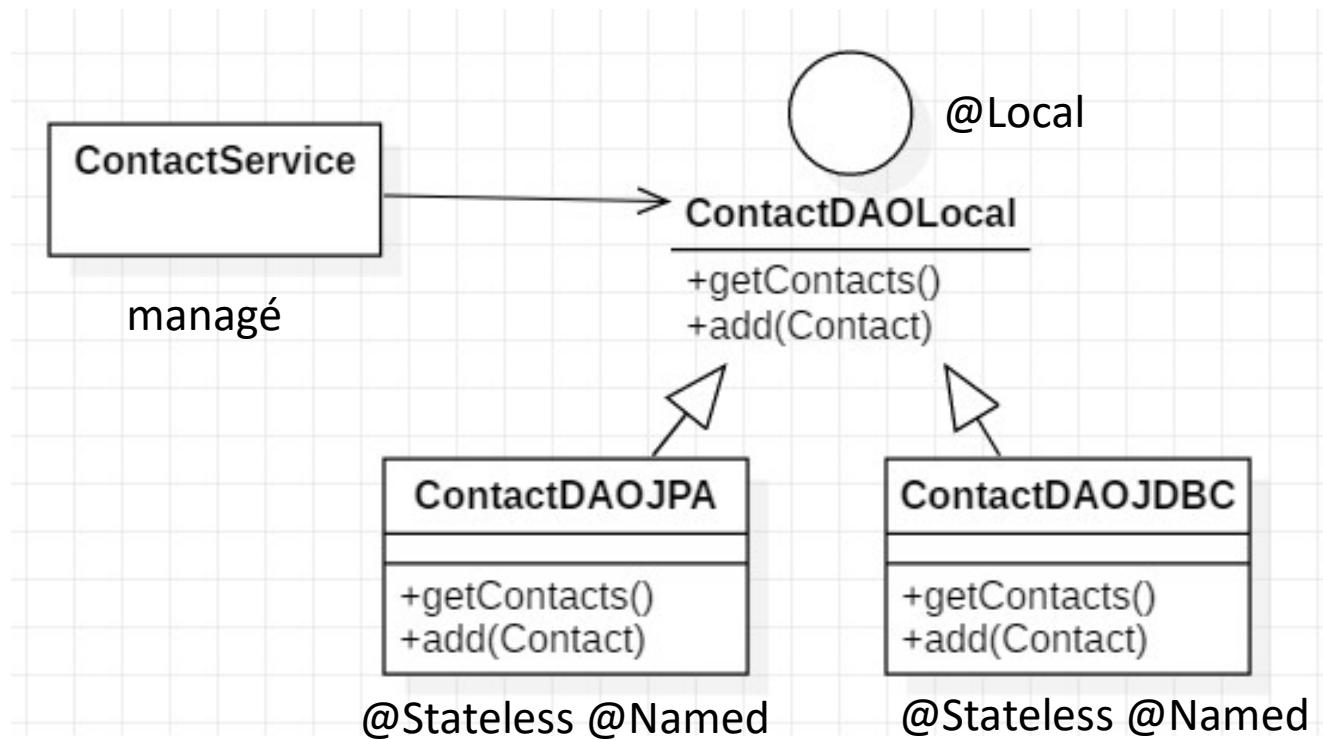
Pattern DAO



EJB DAO

@Inject

```
private @Named("ContactDAOJDBC") ContactDAOLocal contactDAO;
```



DAO JDBC

`@Stateless`

`public class ContactDAO {`

`@Resource(lookup = "jdbc/ContactsDS")`

`private DataSource dataSource;`

Injection du datasource : pool
de connections à une BD

`public List<Contact> getContacts() {`

`List<Contact> contacts = new ArrayList<>();`

`try {`

`Connection connection = dataSource.getConnection();`

`PreparedStatement pstmt = connection.prepareStatement
 ("SELECT * FROM contacts");`

`ResultSet rs = pstmt.executeQuery();`

`while (rs.next()) {`

`. . .`

`}`

`. . .`

`connection.close();`

Obtention d'une
connexion du pool

close : retour de la connexion au pool
ce n'est pas une déconnexion

Datasource et Driver JDBC

- Mécanisme: Service Provider Interface (SPI)
- JAVA EE / JDBC Service (`java.sql.DataSource`)
 - `registerDriver(url ..)` : chargement dynamique du driver spécifique
 - A déposer dans le dossier lib du serveur :
.../payara5/glassfish/lib/mysql-connector-java-8.0.17.jar
- Driver. `getConnection` : création du pool
 - appelle `getConnection` sur le driver spécifique et obtient des connexions à la BD

Configuration du pool (mysql 8)

General

Advanced

Additional Properties

Edit JDBC Connection Pool

Modify an existing JDBC connection pool. A JDBC connection pool is a group of reusable connections for a p

Load Defaults

Flush

Ping



* Indicates required field

General Settings

Pool Name:	ContactsPool
Resource Type:	<div>javax.sql.DataSource ▼</div> <p>Must be specified if the datasource class implements more than 1 of the inte</p>
Datasource Classname:	<div>com.mysql.cj.jdbc.MysqlDataSource</div>

Configuration du pool (mysql 8)

Additional Properties (6)



Add PropertyDelete Properties

Select	Name	Value
<input type="checkbox"/>	useSSL	false
<input type="checkbox"/>	url	jdbc:mysql://localhost:3306/contacts
<input type="checkbox"/>	password	adm
<input type="checkbox"/>	user	adm
<input type="checkbox"/>	serverTimezone	UTC
<input type="checkbox"/>	allowPublicKeyRetrieval	true

Configuration du datasource (mysql 8)

Edit JDBC Resource

Edit an existing JDBC data source.

Load Defaults

JNDI Name:	jdbc/ContactsDS
Pool Name:	ContactsPool ▼

Use the [JDBC Connection Pools](#) page to create new pools