

EJB 3.0

Johannes Carlén - Jan Västernäs

Callista Enterprise AB

Vad tycker vi ?

- Min ödmjukhet är byggd på 35 års framgångar
 - *Bert Karlsson*

- Det är min uppfattning. Och jag delar den.
 - *Alf Svensson*

- Kronan har flera mynt
 - *Gunnar Persson, Brynäscoach*

Outline

- Current Situation (pre EJB 3.0)
- Java 5.0 Annotations
- EJB 3.0 (based on early draft – not complete)
- Demo
- Conclusion

Current situation

Current situation

- EJB used to be the "silver bullet"
- Many people disagree
 - *Need Application Server*
 - *Deployment Descriptors*
 - *Callbacks*
 - *Too much overhead*
 - *Testing is difficult*
 - *Persistence model (Entity Beans) sucks*

Current situation

□ EJB also bring values

- *Declarative Transactions & Security, Threading, Resource Pooling etc*
- *May not be needed by all applications*
- *Component Programming model*

□ Strong Open Source alternate products has emerged

- *Spring – IoC and much more*
- *Hibernate - persistence*
- *xDoclet – javadoc-driven source/configuration files generation*

Java 5.0

Java 5.0

□ Annotations

□ Generics

- *Map<Person, Integer> points = new HashMap<Person, Integer>*

□ Autoboxing

- *points.put(jan,5);*

□ Enums

- *enum Season { WINTER, SPRING, SUMMER, FALL }*

□ Enhanced for-loop

```
List<String> = . . . .  
for ( String s : list ) {  
    System.out.println(s);  
}
```

Java 5.0 Annotations definitions

- Use @ to define

```
@Unittest public OrderImplTest {  
@WebService public OrderService  
@Async public shipOrder() }
```

- Can be used on class, interface, method, method parameter, field.
- Annotation must be defined in an interface

```
public @interface Unittest {
```
- Can have arguments
 - *@Copyright("Callista Enterprise AB")*
- java.lang.annotation package

Java 5.0 Annotations usage

- Parse source code
- Runtime via new Class, Method and Field class methods

```
Class myclass = OrderImplTes.class;  
for (Annotation a : myclass.getAnnotations()) {  
    System.out.println("annotation: " + a.annotationType());  
}
```

```
annotation: interface se.callista.annotate.Unittest
```

EJB 3.0

Extreme makeover

Overall goals with EJB 3.0

- Decrease the number of required artifacts (some are optional)
 - *Home, LocalHome, Local Interface*
 - *Deployment Descriptors*
 - *Vendor Deployment Descriptors*
- Use of annotations
- POJO's and POJI's
- Transparent persistence (POJO-based a la Hibernate)

Overall goals with EJB 3.0 (cont'd)

- Enhancements of entity beans
 - *Inheritance and polymorphism*
 - *Metadata annotations for ORM*
 - *Enhancements of EJB QL – explicit inner and outer joins, subqueries, group-by, dynamic queries...*
- No callbacks
- Dependency Injection
- Decrease the requirements of checked exceptions
- Out of container testing

Interface

□ DD.... **D**eclare **D**irectly

Declare your interface to be remote or local directly in your class file:

`@Remote`

```
public interface OrderService {  
    public OrderDTO findByOrderNo(Long orderNo);  
}
```

Local interface declaration by `@Local`

Implementation

- Set the the type of your EJB the same way:

`@Stateless`

```
public class OrderServiceBean implements OrderService {  
    ...  
}
```

Big difference compared with todays decoupled impl/interface

Transaction Handling

- Set the the transaction attribute of the method:

```
@Tx(TxType.REQUIRESNEW)
```

```
public void updateOrder (OrderDTO order) {  
  
    ...  
  
}
```

Message Driven Beans

```
@MessageDriven
```

```
@ConnectionFactory( destinationType = javax.jms.Queue.class,  
    destinationJndiName = "jms/orderconfirmation",  
    durable = true, subscriptionId = "queueExample")
```

```
public class ConfirmationMDB implements MessageListener {  
    public void onMessage(Message recvMsg) {
```

```
...
```

Dependency Injection

@Inject

- *anything from JNDI*
- *Access objects like SessionContext, TimerService, UserTransaction, EntityManager, EJB's, data sources, environment variables, JMS Queues etc*

@Inject

```
private DataSource ds;
```

Dependency Injection

□ @EJB

- *inject EJB references*

```
private OrderBean orderBean;
```

```
@EJB(jndiName = "se.callista.Orderbean")
```

```
public void setOrderBean(OrderBean orderBean) {  
    this.orderBean = orderBean;  
}
```

□ @Resource

- *access data sources*
- *other*

Entity Beans

- ❑ **Plain Old Java Objects**
- ❑ **Inheritance and polymorphism**
- ❑ **O/R mapping with annotations**
- ❑ **Allocate with `new`.**
- ❑ **Accessed through the EntityManager**

Entity Bean Definition

```
@Entity
```

```
@Table(name = "CALLISTA_PRODUCT")
```

```
public class CallistaProduct implements
```

```
    java.io.Serializable {
```

```
    private Long id;
```

```
    private String name;
```

```
    . . .
```

```
}
```

EntityManager

□ `javax.ejb.EntityManager`

□ **Possible to attach, detach, reattach to/from EntityManager**

```
public void create(Object entity); // attach
public <T> T find(Class<T> entityClass, Object primaryKey);
public <T> T merge(T entity);      // reattach
public void remove(Object entity);
public Query createQuery(String ejbqlString);
public void flush();
public void evict(Object entity); // detach
...
```

Making objects persistent

@Inject

```
private EntityManager entityManager;
```

```
public void create(CallistaProduct product) {  
    entityManager.create(product);  
    // entityManager.flush();  
}
```

Deleting objects

@Inject

```
private EntityManager entityManager;
```

```
public void delete(CallistaProduct product) {  
    entityManager.remove(product);  
    // entityManager.flush();  
}
```

Updating objects

@Inject

```
private EntityManager entityManager;
```

```
public void update(CallistaProduct product) {  
    entityManager.merge(product);  
    // entityManager.flush();  
}
```

Entity Beans – mapping tables & attributes

```
@Entity
```

```
@Table(name = "PURCHASE_ORDER")
```

```
public class Order implements java.io.Serializable {
```

```
    private int id;
```

```
    private String customerName
```

```
    private Date orderDate
```

```
@Id(generate = GenerationType.AUTO)
```

```
public int getId(){return id;}
```

```
@Column(name="CUSTOMER_NAME" , length=80)
```

```
public String getCustomerName(){ return customerName; }
```

```
public Date getOrderDate(){ return orderDate; }
```

Entity Beans – one-to-many (1)

Order:

```
@OneToMany(cascade = CascadeType.ALL,  
           fetch = FetchType.EAGER)  
@JoinColumn(name = "order_id")  
public Collection<OrderItem> getItems() {  
    return items;  
}
```

Entity Beans – one-to-many (2)

OrderItem:

`@Entity`

```
public class OrderItem implements java.io.Serializable {  
    private Order order;  
  
    @ManyToOne  
    @JoinColumn(name = "order_id")  
    public Order getOrder(){ return order;}  
}
```

Queries

@Inject

```
private EntityManager manager;
```

```
public List<Customer> findByLastName(String lastName) {  
    Query query = manager.createQuery(  
        "from Customer c where c.lastname = :lastname");  
    query.setParameter("lastname", lastName);  
    return (List<Customer>) query.listResults();  
}
```

Calling the service

```
List<Customer> customerList =  
    service.findByLastName("Svensson");  
  
for (Customer c : customerList) {  
    System.out.println(c.getName());  
}
```

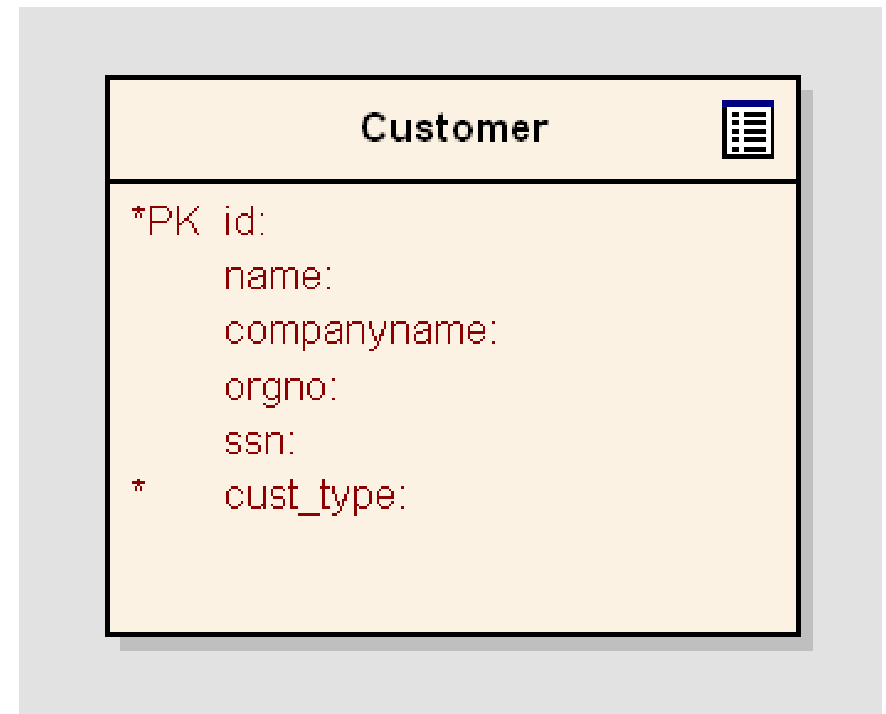
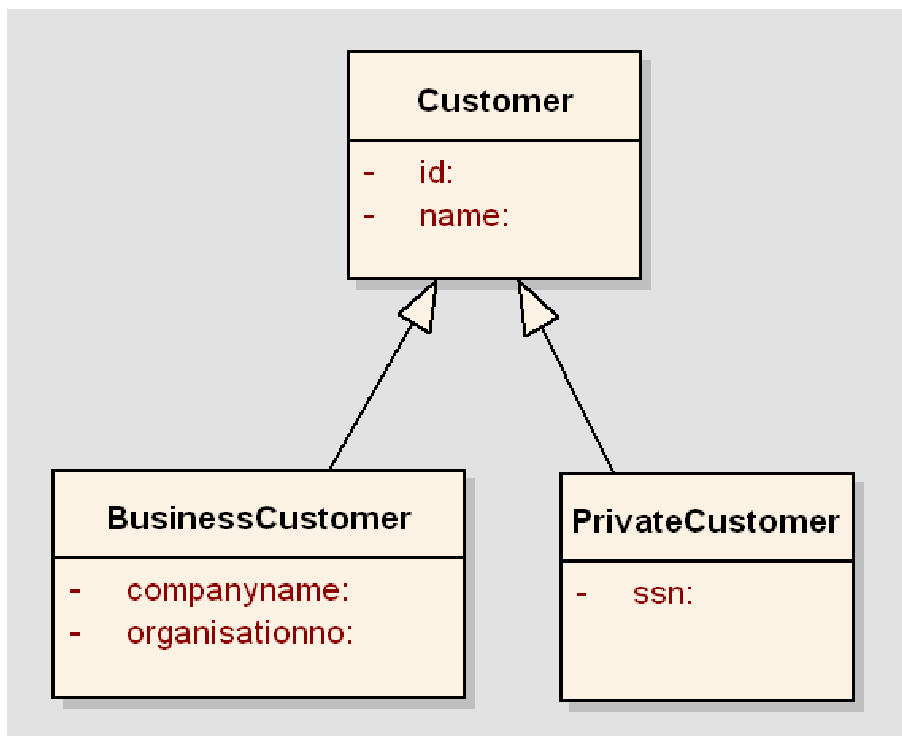
Object Relational Mapping

- Object model is NOT equal to Database schema
- Fine-grained Objects, aka Complex attributes
 - *Company having a Billing Address and Home Address, all attributes in one Table – three Objects*
- Inheritance
 - *See coming slides*
- Derived properties
 - *Exists in the Object but not in the Database*

Entity Beans - inheritance

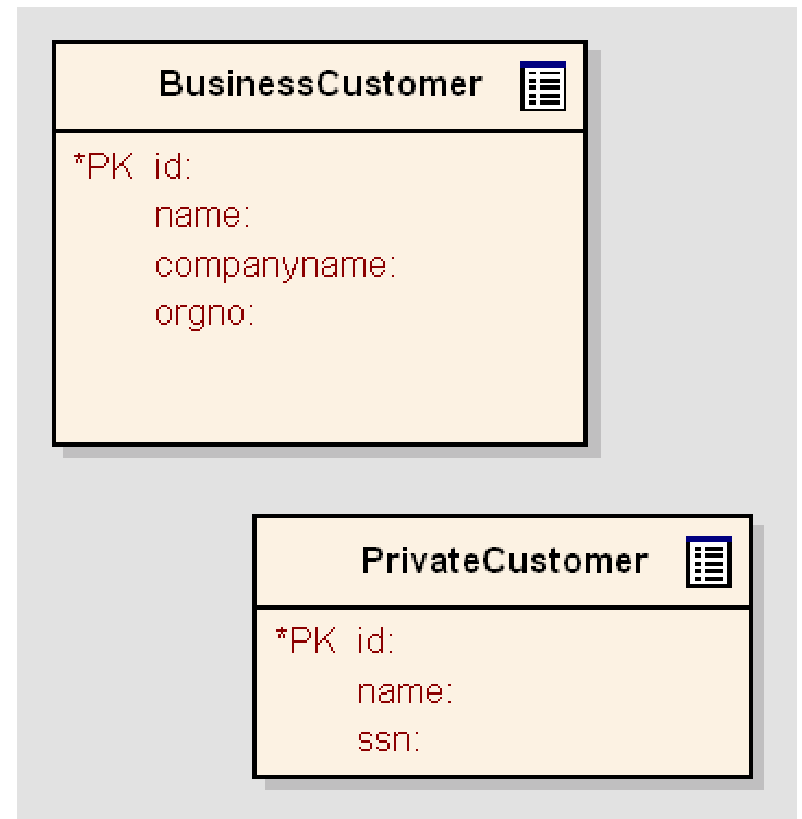
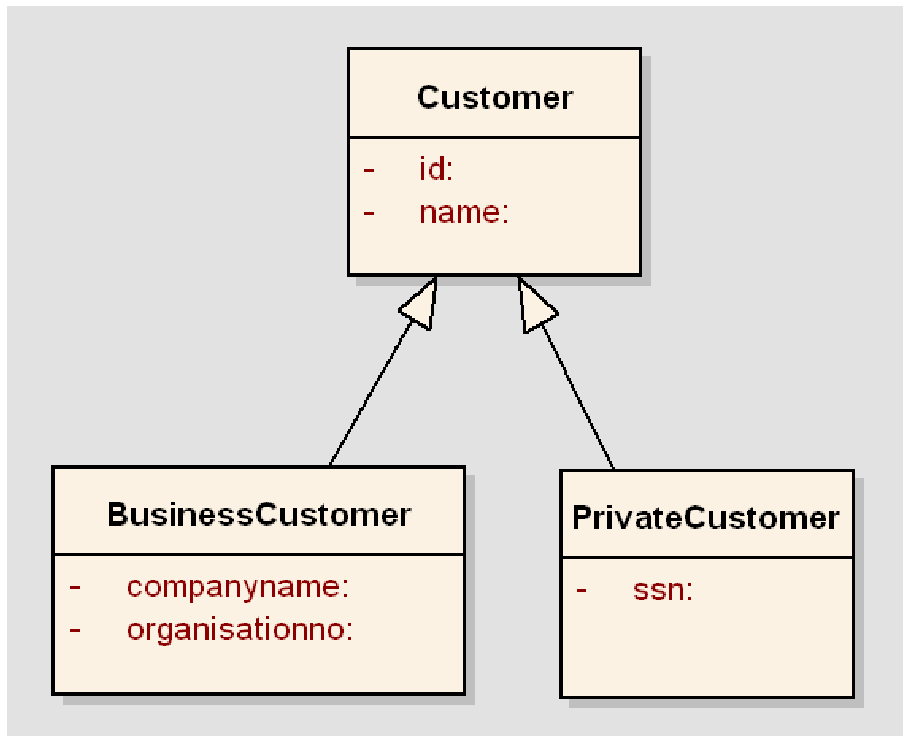
□ Three ways to implement inheritance. First up...

□ Single table strategy



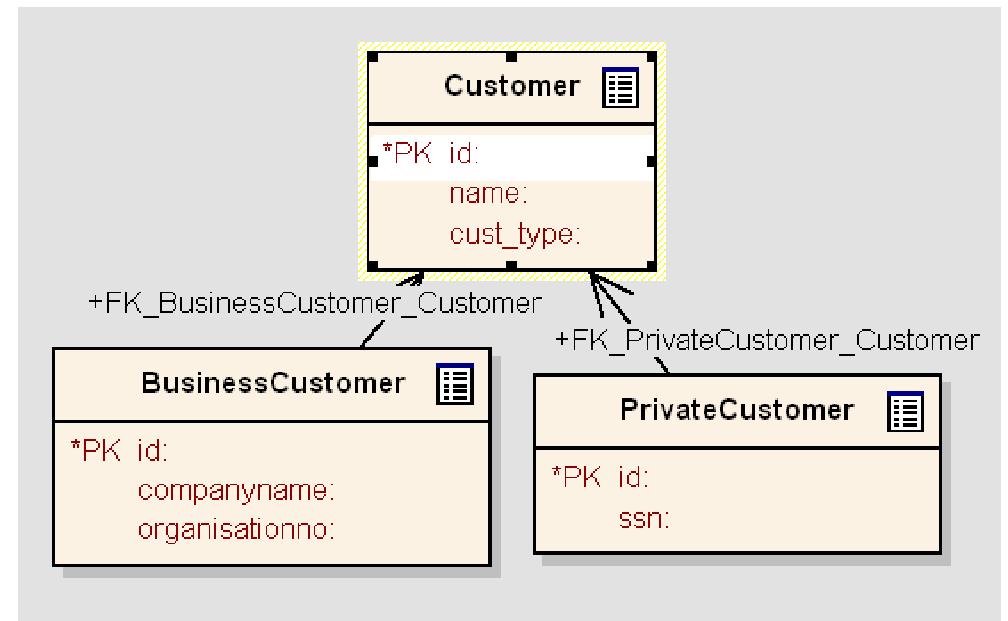
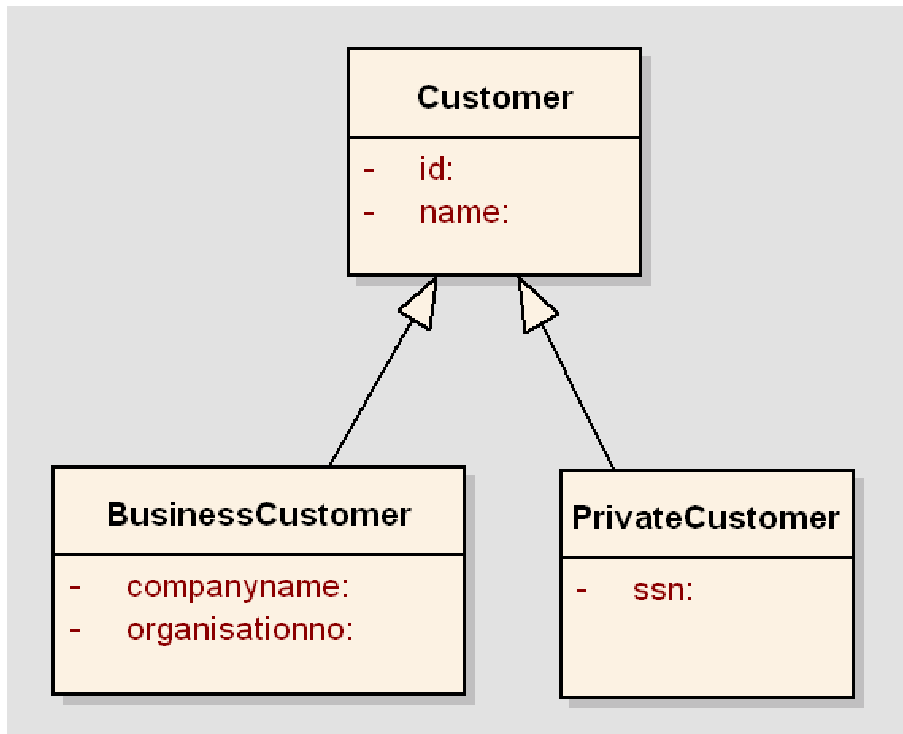
Entity Beans - inheritance

□ One table per subclass



Entity Beans - inheritance

□ Join table strategy



Join table strategy

```
@Entity
```

```
@Inheritance(strategy = InheritanceType.JOINED,  
    discriminatorType = DiscriminatorType.STRING)
```

```
@DiscriminatorColumn(name = "CUSTOMER_TYPE")
```

```
public class Customer implements java.io.Serializable {
```

```
@Entity
```

```
@Inheritance(strategy = InheritanceType.JOINED,  
    discriminatorType = DiscriminatorType.STRING,  
    discriminatorValue = "B_CUST")
```

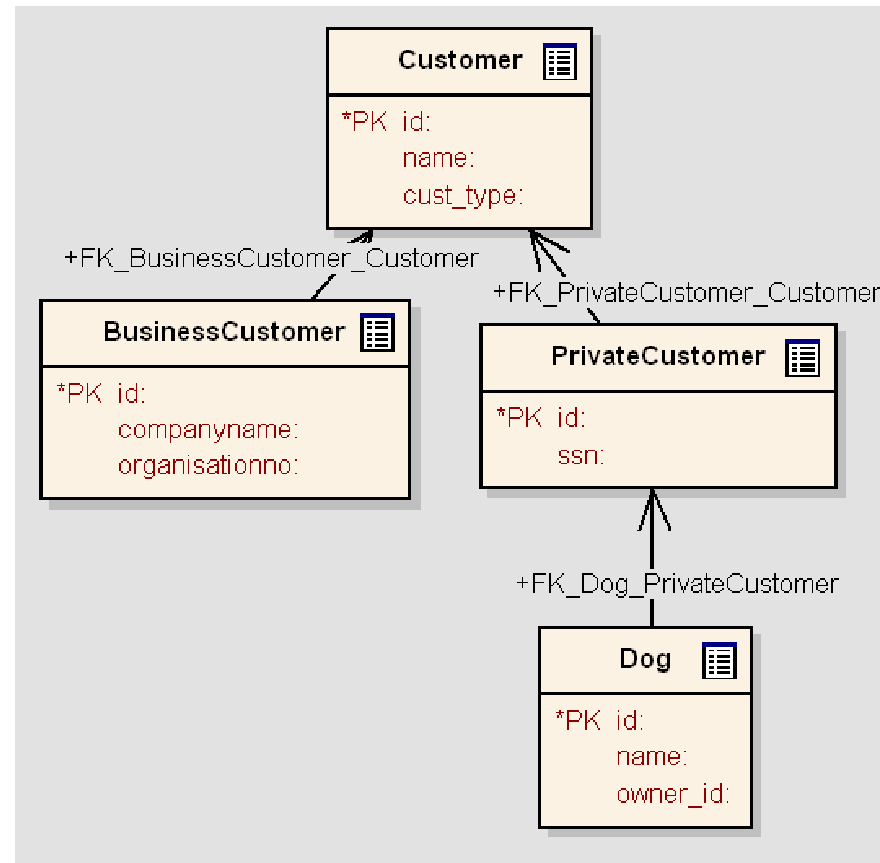
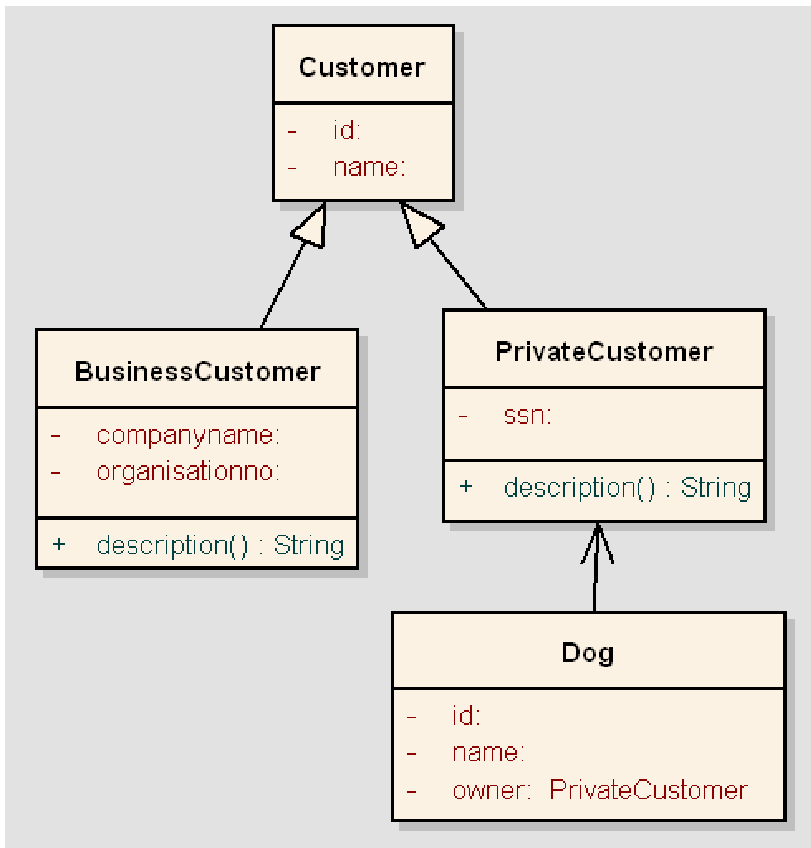
```
public class BusinessCustomer extends Customer {
```

Demo - Callista Store EJB3.0

- Eclipse 3.1 M4, compile-checking favorite Notepad
- JBoss 4.0.1 with EJB3 Preview 2
- Ant build
- Changes – less code
 - *No DAO:s*
 - *No SQL*
 - *No Deployment Descriptors*
 - *No Home Interfaces*
 - *No callbacks*
- Changes – more code
 - *Annotations*

Demo content

- Polymorfistic Query on Customer
 - *Find all customers*
 - *call description() abstract method for each hit*



Conclusion

Good news

- Easier development – less artifacts
 - *POJO + POJI + Annotations*
- Entity beans – completely new
 - *Transparent, POJO-based*
- Dependency Injection
 - *Better for testing*
- Is a specification – not some proprietary open source
 - *Vendor competition*
 - *Portability*

Concerns

- Timing – when will it be ready ?
 - *Spring+Hibernate is gaining momentum*
- Out of container testing. Not in the spec yet.
 - *Maybe "next release"*
- Dependency Injection – not a full implementation
 - *Inject non-JNDI POJO:s ?*

Questions ?

EJB 3.0

- Coming to an Application Server near you soon
- Considerably easier to use

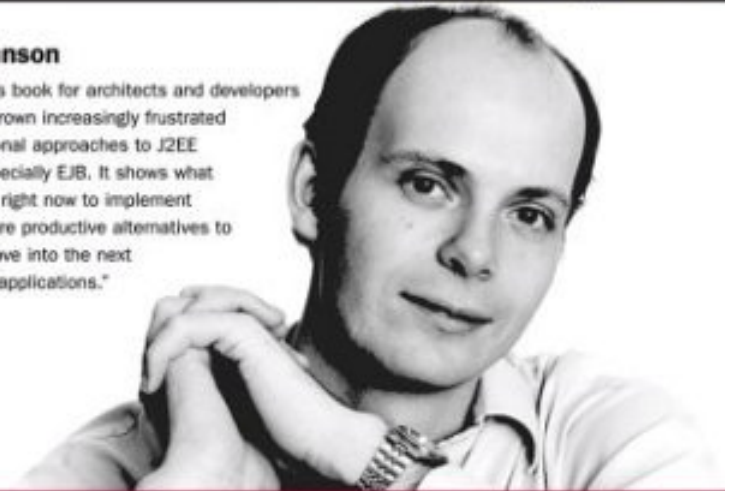
Rod Johnson 14 april

- Färgstark kille !
- Självkostnad c:a 1000:-
- Börja bearbeta din chef nu.
- Preliminärt intresse till Maria
- Välkommna!

Programmer to Programmer™

Rod Johnson

"I wrote this book for architects and developers who have grown increasingly frustrated with traditional approaches to J2EE design, especially EJB. It shows what you can do right now to implement cleaner, more productive alternatives to EJB and move into the next era of web applications."



expert one-on-one™

**J2EE™ Development
without EJB™**

Rod Johnson with Juergen Hoeller



Updates, source code, and Wrox technical support at www.wrox.com