

Overview over EJB3



info@ejb3workshop.com

http://www.ejb3workshop.com



Current State of EJB3

- *Current specification is up for public review, which closed on 15th August 2005*
- *Implementations already available from various vendors, e.g. JBoss*
- *<http://java.sun.com/products/ejb/docs.html>*





What's new ?

- *Simplification*
- *Based on standards and existing practices*
- *RAD cycle*
- *Focus on business functionality rather than piping and framework*
- *Uses many “new” Java 5 features*

Java 5 features

- *Annotations*
- *Generics*
- *Auto boxing*





Java 5 Annotations

- *Defined in an interface*
- *Declared using @Annotation*
- *Accessible via reflection API*
- *Used to add meta information e.g. deployment descriptor or bean info*
- *See : <http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html>*

Java 5 Generics for Collections

- *Generics allow for typed collection which only contain objects of a particular class. A collection of Integers, People, Vehicle or Accounts.*

```
ArrayList list = new ArrayList();  
list.add(0, new Integer(42));  
int total = ((Integer)list.get(0)).intValue();
```

```
ArrayList<Integer> list = new ArrayList<Integer>();  
list.add(0, 42);  
int total = list.get(0);
```



Java 5 Autoboxing

- *Automatic conversion between primitive data types (int, long, boolean) and their respective Object counterparts (Integer, Long and Boolean)*

```
ArrayList<Integer> list = new ArrayList<Integer>();  
list.add(0, new Integer(42));  
int total = (list.get(0)).intValue();
```

```
ArrayList<Integer> list = new ArrayList<Integer>();  
list.add(0, 42);  
int total = list.get(0);
```



Aspect Oriented Programming

- *Separation of functional and non-functional concerns or aspects*
- *Non functional concerns are removed from non-functional*
- *Additional feature can be added at a later on by added new aspects without affecting functional aspects*



Example uses of AOP

- *Logging*
- *Security*
- *Testing*
- *Transformations*
- *Any other concern which does not directly impact on the underlying concern.*





EJB3 and META data

- *Inline META data*
- *External META data (deployment descriptor)*
- *Both have pro's and con's*
 - Lost ability to deploy same bean in multiple contexts*
 - Gained maintainability and portability*

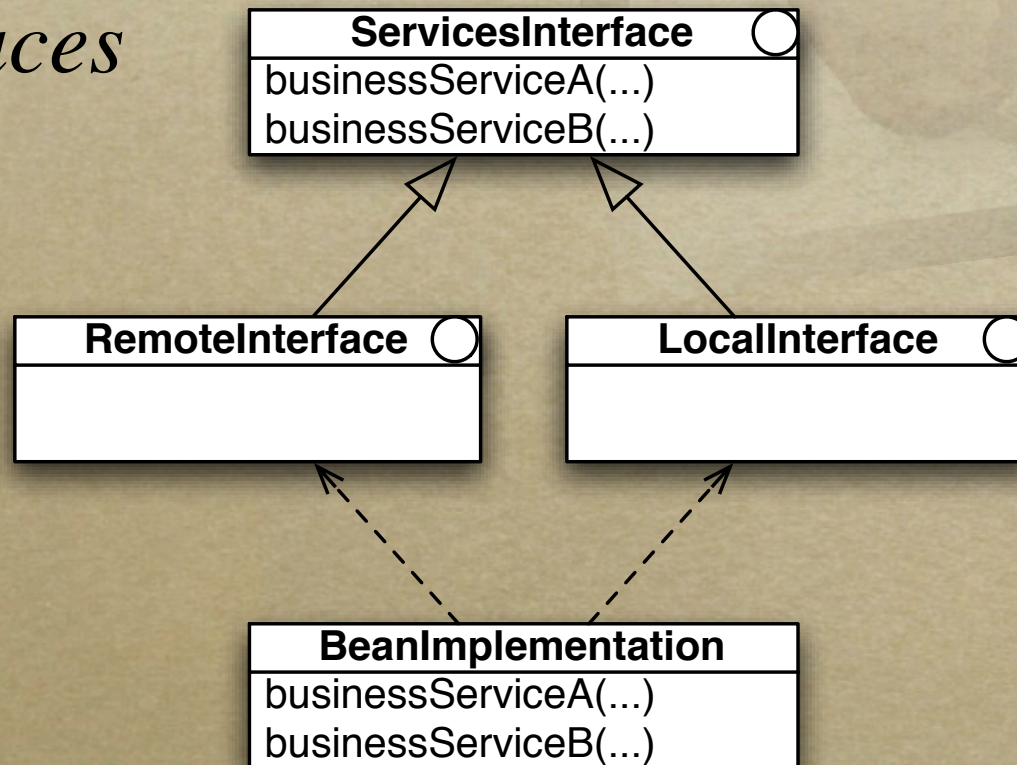
EJB3 - Session Beans

- *Stateful and Stateless*
- *Use of vanilla POJO's plus META data*
- *Annotation of call back methods to replace `ejbXYZ(...)`*



EJB3 - Session Bean Pattern

- *Ensures that the common services are supported by both Remote and Local interfaces*

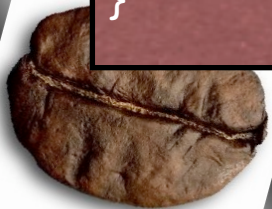


EJB3 - Stateless Session Bean

- *The Services Interface*

```
package com.ejb3workshop.sessionbean;
import java.io.Serializable;

public interface CalculatorServices extends Serializable
{
    public double add(double a, double b);
    public double subtract(double a, double b);
    public double multiply(double a, double b);
    public double divide(double a, double b) throws
        CalculatorException;
}
```



EJB3 - Stateless Session Bean

- *Remote Interface*

```
package com.ejb3workshop.sessionbean;  
import javax.ejb.Local;
```

```
@Local
```

```
public interface CalculatorLocal extends CalculatorServices  
{  
}
```



EJB3 - Stateless Session Bean

- *Remote Interface*

```
package com.ejb3workshop.sessionbean;  
import javax.ejb.Remote;
```

@Remote

```
public interface CalculatorRemote extends CalculatorServices  
{  
}
```



EJB3 - Stateless Session Bean

o *Bean Implementation*

```
package com.ejb3workshop.sessionbean;
import java.util.logging.Logger;
import javax.ejb.*;
@Stateless
public class CalculatorBean implements CalculatorRemote,
CalculatorLocal
{
    ...
    public double add(double a, double b)
    {
        ...
    }
    ...
}
```

EJB3 - Stateless Session Bean

- *Bean Implementation continued*

```
@PostConstruct
```

```
public void postConstruct()  
{  
    logger.info("POST CONSTRUCT");  
}
```

```
@PreDestroy
```

```
public void preDestroy()  
{  
    logger.info("PRE DESTROY");  
}  
}
```

EJB3 Client Code

```
package com.ejb3workshop.sessionbean;
import java.util.*;
import javax.ejb.EJBException;
import javax.naming.*;

public class Client
{
    public void runTest()
    {
        try
        {
            InitialContext ctx = new InitialContext();
            CalculatorServices calculator = (CalculatorServices) ctx.lookup(CalculatorRemote.class.getName());
            System.out.println("ADD      : "+calculator.add(5,4));
        }
        catch (Exception e){e.printStackTrace();}
    }

    public static void main (String args[])
    {
        new Client().runTest();
    }
}
```

Stateless Lifecycle



EJB3 - Stateful Session Bean

o *Bean Implementation*

```
package com.ejb3workshop.sessionbean;
import java.util.logging.Logger;
import javax.ejb.*;
@Stateful
public class CalculatorBean implements CalculatorRemote,
CalculatorLocal
{
    ...
    public double add(double a, double b)
    {
        ...
    }
    ...
}
```

EJB3 - Stateful Session Bean

- *Bean Implementation continued*

`@Remove`

```
public double getResult()  
{  
    ...  
}
```

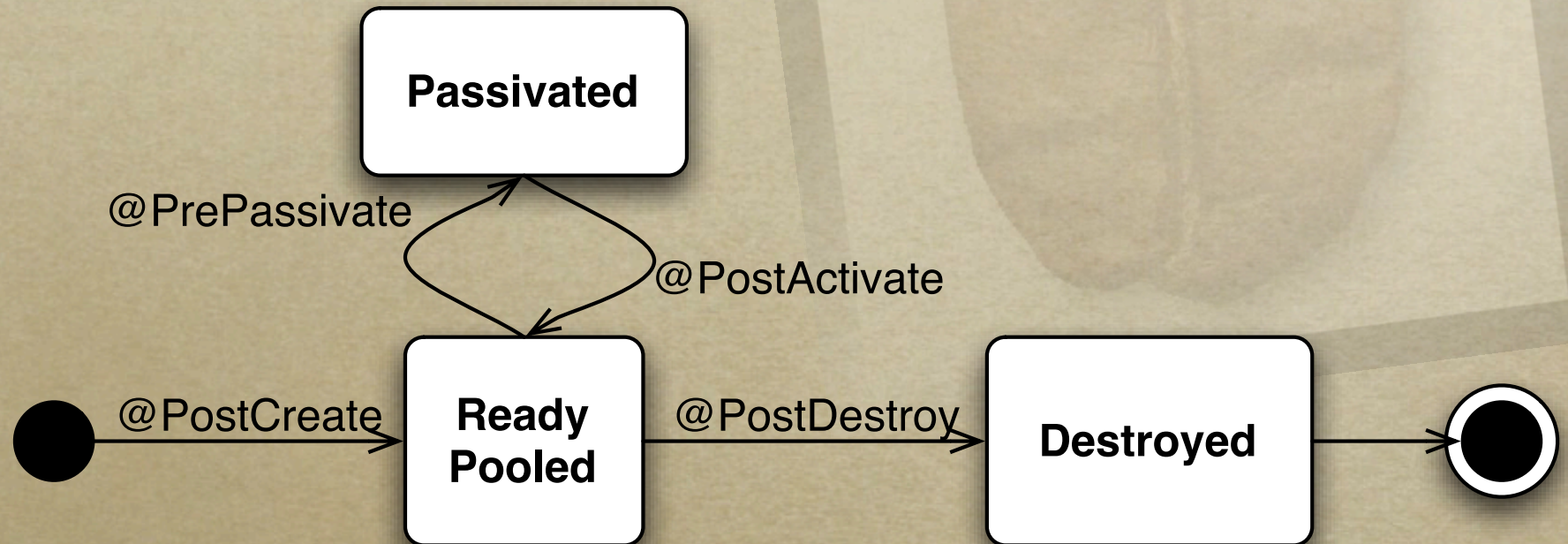
`@PostActivate`

```
public void postActivate()  
{  
    ...  
}
```

`@PrePassivate`

```
public void prePassivate()  
{  
    ...  
}
```

Stateful Lifecycle



EJB Interceptors

- *Implementation of basic AOP*
- *Less capable than AspectJ or AspectWerks*
- *Allows only for interceptor delegation to other methods and / or other classes.*
- *Affects all services offered by the bean*



EJB3 Interceptors

- *Specifies a single interceptor method per bean via **@AroundInvoke** in the bean itself*
- *Specific method signature*

@AroundInvoke

```
public Object customInterceptor(InvocationContext ctx) throws  
Exception  
{  
    System.out.println("*** BEFORE INTERCEPTION ***");  
    Object object = ctx.proceed();  
    System.out.println("*** AFTER INTERCEPTION ***");  
    return object;  
}
```

EJB3 Interceptors

- *Specifies a series of interceptor which are invoked sequentially*

```
import javax.ejb.*;
```

```
@Interceptors
```

```
({com.ejb3workshop.sessionbean.interceptor.Dog.class})
```

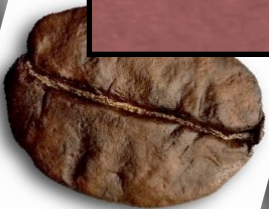
```
@Stateless
```

```
public class MailManBean implements MailManRemote,
```

```
MailManLocal
```

```
{
```

```
...
```

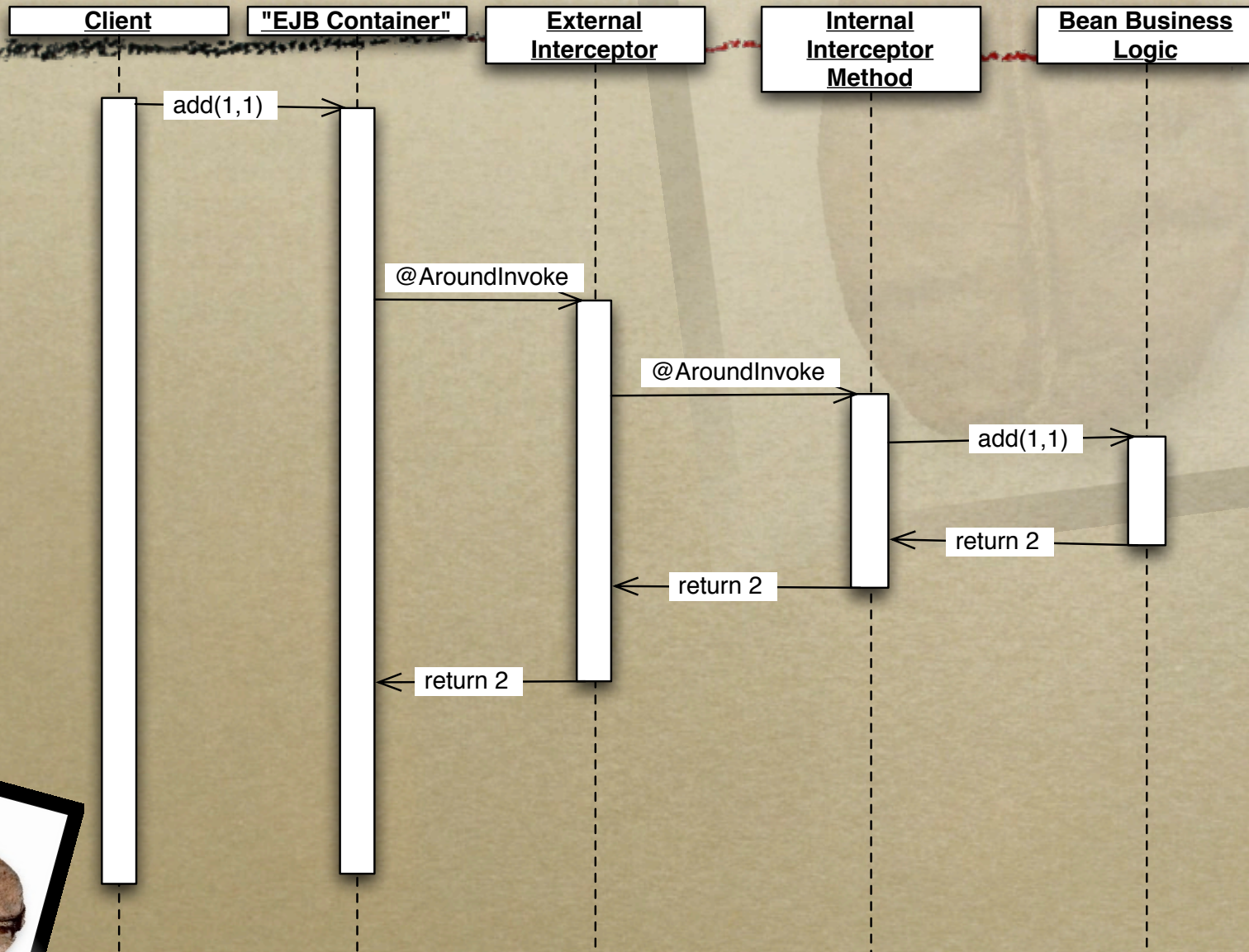


EJB3 Interceptors

- *Interception handled in external class*
- *Same method signature as before*

```
package com.ejb3workshop.sessionbean.interceptor;
import javax.ejb.*;
public class Dog
{
    @AroundInvoke
    public Object intercept(InvocationContext ctx) throws
Exception
    {
        System.out.println("*** BEFORE INTERCEPTION ***");
        Object object = ctx.proceed();
        System.out.println("*** AFTER INTERCEPTION ***");
        return object;
    }
}
```

Interception Sequence



EJB3 Interceptors

- *Only provide interceptors for all methods of a particular bean*
- *Consider and investigate AspectJ, AspectWerks or alternate AOP implementation*




Q & A



Next Section Entity Beans...





EJB3 - Entity Beans

- *Single defined layer for persistence*
- *Reuse of persistent classes in business and presentation layer*
- *Use POJO as persistent component, with minor modifications*
- *Ability to map to existing data model / database schema*



EJB3 - Entity Beans

- *Optimistic Locking via versioning*
- *Support for relationships*
- *Support for association and composition*
- *Support for inheritance and polymorphic collections*
- *Primary Key generation*
- *Query language and SQL support*



EJB - Entity Example

- *Standard POJO*
- *Added Annotation*
- *Should provide implementation for **equals** and **hashCode**.*
- *Can be specialisation of common “**super**” entity.*



Entity Example

```
package com.ejb3workshop.addressbook;
import java.util.*;
import javax.persistence.*;

@Entity
@Table(name="Contacts")
public class Contact implements java.io.Serializable
{
    private String name;
    private String surname;
    @Column(name="name")
    public String getName(){return name;}
    public void setName(String name){this.name = name;}

    @Column(name="surname")
    public String getSurname(){return surname;}
    public void setSurname(String surname){this.surname = surname;}
    /*...equals and hash should also be implemented */
}
```



Entity Manager

- *Once the entities have been defined the “Entity Manger” is used to interact with the “Persistence Context”.*
- *The entity manager is responsible for the persistence of the entity and provides services for : persisting (creation), updating (merging), deleting (remove)*
- *Annotated in Session Beans as @PersistenceContext*

Persistence Context

- *Persistence context represent a collection of persistent entities*
- *Scoped around a single transaction, but can be extended to beyond that, but never multiple concurrent transactions.*



Entity Manager Example

```
@Stateful
public class AddressBookBean implements AddressBookRemote,
AddressBookLocal
{
    @PersistenceContext
    private EntityManager manager;
    public void addContact(Contact contact)
    {
        manager.persist(contact);
    }
    public void updateContact(Contact contact)
    {
        manager.merge(contact);
    }
    public void deleteContact(Contact contact)
    {
        manager.remove(contact);
    }
}
```



Entity Manager

- *Useful to return entity after create / update*

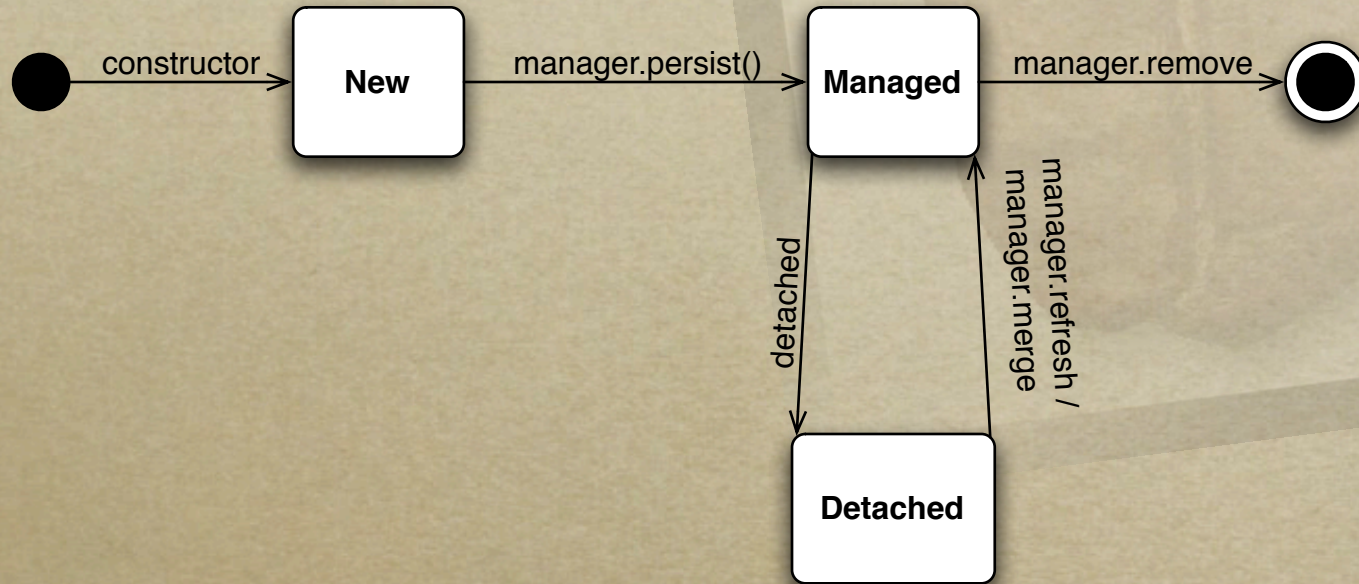
```
public Contact addContact(Contact contact)
{
    manager.persist(contact);
    return contact;
}
public Contact updateContact(Contact contact)
{
    manager.merge(contact);
    return contact;
}
public void deleteContact(Contact contact)
{
    manager.remove(contact);
}
```

Entity Manager - Queries

- *Support for EQL and SQL queries via `manager.createQuery` and `manager.createNativeQuery`*

```
public Collection<Contact> findContact(String name)
{
    Query query = null;
    query = manager.createQuery("Select c from Contact c where
c.name = :contactName");
    query = query.setParameter("contactName", name);
    List contacts = query.getResultList();
    return contacts;
}
```

Entity Lifecycle



Entity States

- *New: The entity is created but not attached to the persistence context*
- *Managed: The entity has been attached and is now added to the persistence context*
 - *State changes synchronised with backend*
 - *Transparent association are fetched*





Entity States

- ***Detached:*** *The entity has been passed outside the container*
 - *No longer managed by container*
 - *State changes to be managed manually*
 - *Transparent fetch of associations no longer supported*
- ***Removed:*** *Deleted from persistence context*

EJB3 - Entity Relationships

- *One - One*
- *One - Many*
- *Many - Many*
- *Inheritance, including **Abstract** entities*
- *Support for embedded objects*





One - One

```
@Entity
public class A
{
    private B b;
    @OneToOne
    public B getB()
    {
        return b;
    }

    public void setB(B b)
    {
        this.b = b;
    }
}
```

```
@Entity
public class B
{
    private A a;
    @OneToOne(mappedBy="b")
    public A getA()
    {
        return a;
    }

    public void setA(A a)
    {
        this.a = a;
    }
}
```

A is the owner of the relationship as B contains the **mappedBy** annotation
For **one-to-one bidirectional relationships**, the owning side corresponds to the side that contains the corresponding foreign key.



One - Many

```
@Entity
public class A
{
    private Collection<B> bees;
    @OneToMany (mappedBy="a")
    public Collection<B> getBs()
    {
        return bees;
    }

    public void setBs(Collection<B> b)
    {
        this.bees = b;
    }
}
```

```
@Entity
public class B
{
    private A a;
    @ManyToOne
    public A getA()
    {
        return a;
    }

    public void setContact(A a)
    {
        this.a = a;
    }
}
```

B must be the owner of the relationship as it is the many side of the relationship

The many side of one-to-many / many-to-one bidirectional relationships must be the owning side, hence the mappedBy element cannot be specified on the ManyToOne annotation.



Many - Many

```
@Entity
public class A
{
private Collection<B> bees;
@ManyToMany
public Collection<B> getBs()
{
return bees;
}

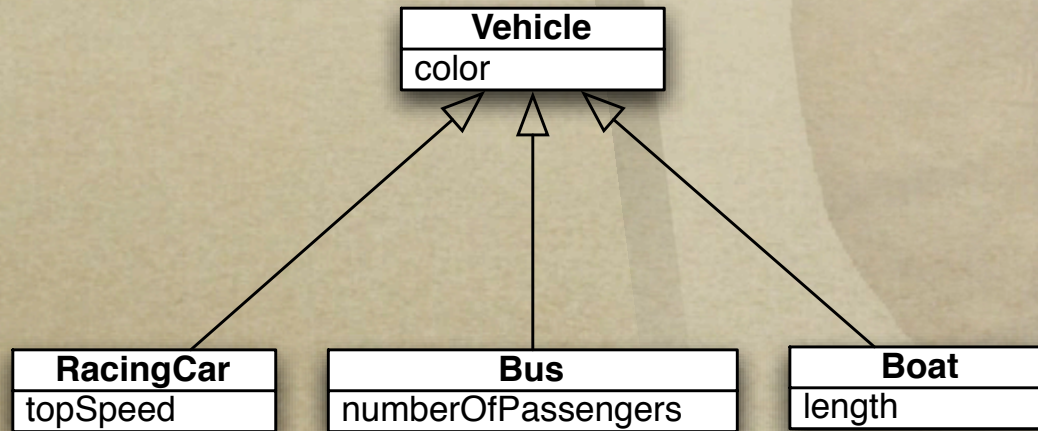
public void setBs(Collection<B> b)
{
this.bees = b;
}
}
```

```
@Entity
public class B
{
private Collection<A> as;
@ManyToMany (mappedBy="b")
public Collection<A> getAs()
{
return as;
}

public void setAs(Collection<A> a)
{
this.as = a;
}
}
```

A is the owner of the relationship as B contains the **mappedBy** annotation
For **many-to-many bidirectional relationships** either side may be the owning side.

EJB3 - Single Table Strategy



Vehicle Table				
Color	Top Speed	Number of Passengers	Length	Vehicle Type



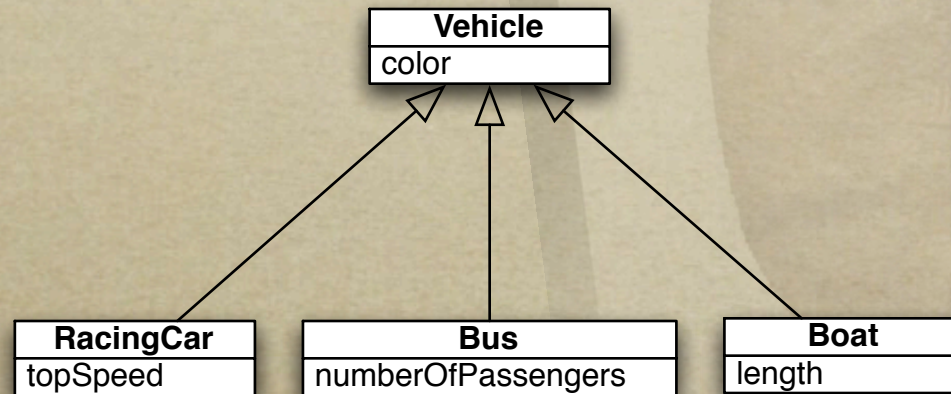
Single Table Strategy

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE,
discriminatorType = DiscriminatorType.STRING)
@DiscriminatorColumn(name = "VEHICLETYPE")
public class Vehicles implements java.io.Serializable
```

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE,
discriminatorType = DiscriminatorType.STRING, discriminatorValue
= "CAR")
public class Car extends Vehicles
```



EJB3 - Table per Class Strategy



Vehicle Table	
Color	

RacingCar Table	
Color	TopSpeed

Bus Table	
Color	Number Of Passengers

Boat Table	
Color	Length



Table per Class Strategy

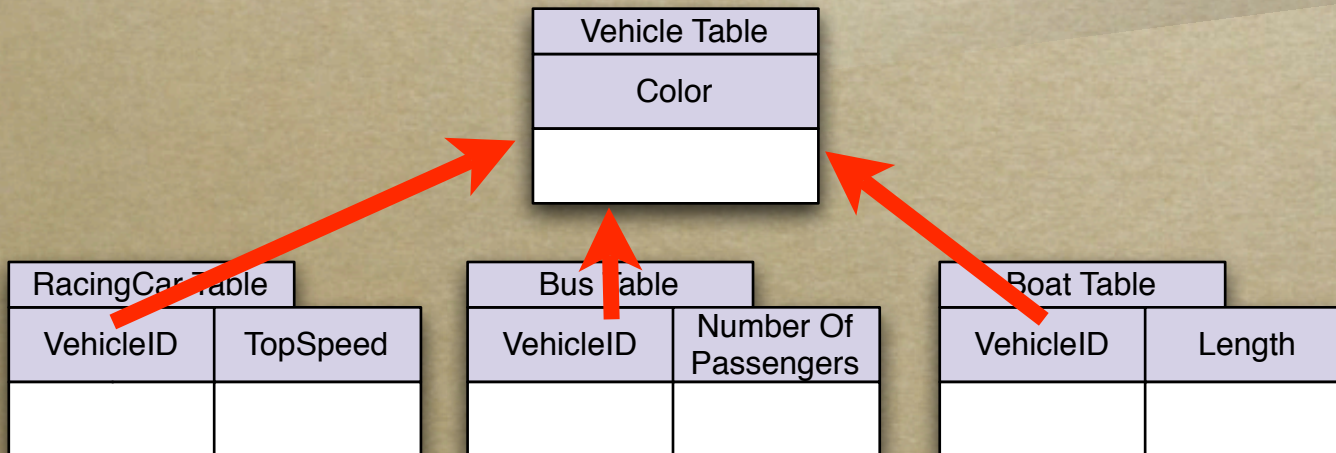
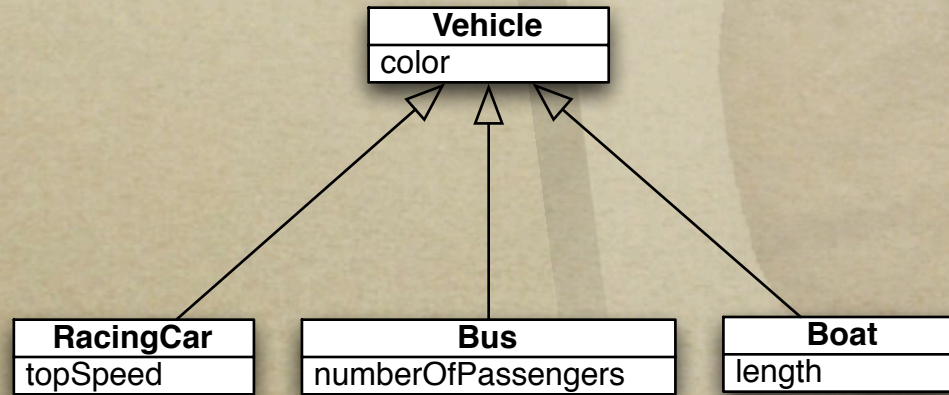
```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Vehicles implements java.io.Serializable
```

```
@Entity
public class Car extends Vehicles
```





EJB3 - Join Strategy



Join Strategy

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Vehicle implements java.io.Serializable
```

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Car extends Vehicle
```



EJB3 - Fetch Strategy

- *Two fetch strategies are available for relationships.*
 - *Eager - The related object is also fetched from the database. Allows for prefetching of frequently used objects.*
 - *Lazy - The related object is loaded on demand from the database. Allows for on-demand loading of rarely used objects*

@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, mappedBy="contact")



EJB3 - Cascade Strategy

- *The Cascade Strategy can be customised for entity relationships by including the cascade constraint in the the relationship annotation.*
- *The default is no cascading behaviour*
- *ALL, PERSIST, MERGE, REMOVE and REFRESH*

```
@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, mappedBy="contact")
```





Entity Callbacks

- *Similar to the callback's used on the session beans we can annotate methods using the following:*
- *@PrePersist*
- *@PostPersist*
- *@PreRemove*
- *@PostRemove*

Entity Callbacks

- *@PreUpdate*
- *@PostUpdate*
- *@PostLoad*





EJB3 - Primary Key Generators

- *A primary key which can now be generated as a new instance is created.*

```
@Id(generate = GenerationType.AUTO)
@Column(name="id")
public int getId()
{
    return id;
}

public void setId(int id)
{
    this.id = id;
}
```

EJB3 - Optimistic Locking

- *Optimistic Locking is achieved via the addition of a version attribute to each bean. Each instance has a specific version. The version is incremented during each update.*
- *An exception is thrown when the version of the instance has been updated (incremented) by another party.*

Optimistic Locking Example

```
@Version  
public int getVersion()  
{  
    return version;  
}  
public void setVersion(int version)  
{  
    this.version=version;  
}
```

EJB3 - Message Driven Beans

- *Asynchronous Messaging*

```
@MessageDriven(activateConfig =  
{  
    @ActivationConfigProperty(propertyName="destinationType",  
                               propertyValue="javax.jms.Queue"),  
    @ActivationConfigProperty(propertyName="destination",  
                               propertyValue="queue/ejb3/demo")  
})
```

```
public class ExampleMDB implements MessageListener  
{  
    public void onMessage(Message recvMsg)  
    ...  
}
```



Useful references

<http://java.sun.com/developer/technicalArticles/releases/j2se15/>

<http://eclipse.org/aspectj/>

<http://www.jboss.com/products/ejb3>

<http://today.java.net/pub/a/today/2004/06/15/ejb3.html>

<http://ejb3workshop.com>

Q & A



Thank you

