

# Enterprise Java Beans 3 (EJB)



# EJB 2.x'in Kısıtları

- Remote ve Home Interface tanımlamaları **çok fazla**
- EJB sınıfı Remote interface'i implement etmemesine rağmen metotlarını **implement etmek zorunda**
- EJB instance'ı yaratma, erişim ve kullanım adımları tamamen **gereksiz, tek adımda olabilir**
- Checked exception'ların kullanımı **oldukça fazla**

# EJB 2.x'in Kısıtları

- Bean'ın container tarafından devreye alınması için deployment descriptor'lara **ihtiyaç yok**
- Container ortamına erişim, diğer bean ve resource'lara lookup oldukça **hataya açık ve standart değil**
- Entity Bean ile persistence işlemleri **oldukça karmaşık**
- Container Managed Persistence ve Relationship işlemleri **oldukça kısıtlı**

# EJB 2.x'in Kısıtları

- Polymorphism ve Inheritance kabiliyetleri **yok veya standart dışı** yöntemlerle yapılıyor
- EJB QL **yetersiz**, çoğu zaman bean managed persistence ve SQL kullanılıyor

# EJB 3.x ile Gelen Yenilikler

- Home ve remote interface tanımlarına **gerek kalmamıştır**
- EJB sınıfının da herhangi bir **interface**'i implement etmesine **gerek yoktur**
- Session ve entity bean tanımları için **deployment descriptor**'a da **gerek kalmamıştır**
- Callback metotlarından **sadece ihtiyaç duyulanlar** anotasyonlar ile tanımlanabilir
- Application server'daki diğer **resource**'lara **erişim** kolaylaşmıştır

# EJB 3.x ile Gelen Yenilikler

- EJB içerisinde ihtiyaç duyulan resource instance'ları **container tarafından enjekte** edilmektedir
- Bu işleme **dependency injection** adı verilir
- Session ve message bean'ların metotlarından önce ve sonra devreye girecek **interceptor tanımlanabilir**
- EJB instance'ını **yaratma ve erişim adımları basitleşmiştir**, instance'a doğrudan erişip metotlarını çağırmak yeterlidir

# EJB 3.x ile Gelen Yenilikler

- EJB nesneleri tekrar normal/sıradan **Java nesnelerine** benzemiştir
- Buna **POJO** adı verilir
- Entity – tablo, property – sütun **eşleştirmeleri de annotasyonlarla yapılabilir** hale gelmiştir
- Persistence işlemleri **Java Persistence API (JPA)** ismi ile ayrı bir spesifikasyon olmuştur
- Entity **ilişkilerinin yönetimi ve transaction yönetimi basitleşmiştir**

# Stateless Session Bean Örneği

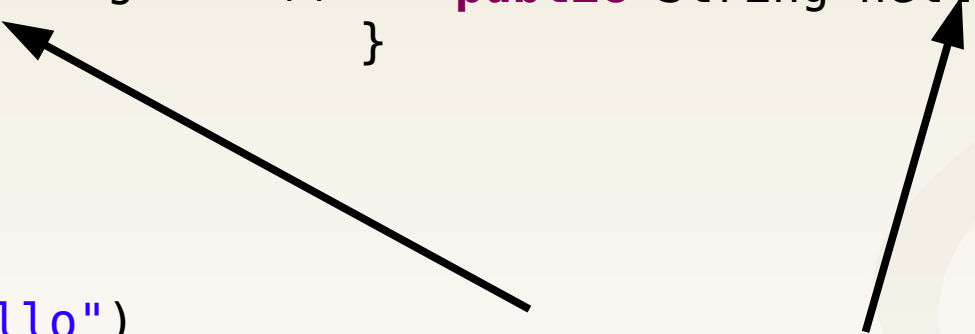
```
@Remote
public interface HelloRemote {
    public String hello(String name);
}

@Local
public interface HelloLocal {
    public String hello(String name);
}

@Stateless(name="hello")
public class HelloBean implements HelloRemote, HelloLocal {

    @Resource(mappedName="java:/DefaultDS")
    private DataSource dataSource;

    public String hello(String name) {
        return "Hello " + name;
    }
}
```





# Callback Metotlar

```
@Stateless(name="hello")
public class HelloBean implements HelloRemote, HelloLocal {

    ...

    @PostConstruct
    public void initializeCountryWineList() {

    }

    @PreDestroy
    public void destroyWineList() {

    }

}
```

# Stateful Session Bean Örneği

```
@Stateful(name = "ShoppingCart")
public class ShoppingCartBean implements ShoppingCart,
ShoppingCartLocal {

    private List cartItems;

    @PostConstruct
    public void initialize() {
        cartItems = new ArrayList();
    }

    @PreDestroy
    public void exit() {
        System.out.println("Save items list into database");
    }

    @Remove
    public void stopSession() {
        System.out.println("Session bean instance is removed");
    }
}
```

# Message Driven Bean Örneği

```
@MessageDriven (activationConfig = {
    @ActivationConfigProperty(propertyName="destinationType",
        propertyValue="javax.jms.Queue"),
    @ActivationConfigProperty(propertyName="destination",
        propertyValue="queue/SimpleMessageQ")})
public class SimpleMessageBean implements MessageListener {

    public void onMessage (Message msg) {
        TextMessage txtMsg = (TextMessage)msg;

        try {
            String message = txtMsg.getText();

            System.out.println(message);

        } catch (JMSEException e) {
            e.printStackTrace();
        }
    }
}
```

# EJB Instance'larına Erişim

```
@Stateless
public class GoldBidderManagerBean {

    @EJB
    private BidManager bidManager;

    ...

}

public class HelloServlet extends HttpServlet {

    @EJB
    private HelloRemote hello;

    ...

}
```

# EJB 3 ve Entity Bean

- Persistence işlemleri **EJB 3** **spesifikasyonundan** çıkarılmıştır
- EJB 2.x'deki gibi **distributed bileşen** olarak entity bean kavramı ortadan kalkmıştır
- Persistence işlemleri **Java Persistence API** (JPA) spesifikasyonu ile belirlenmiştir

- **Sıradan Java nesnelerinin** persistence işlemlerinde kullanılmasını ve
- Metadata'nın **Java annotasyonları** ile tanımlanmasını temel almıştır
- **JPA** bir spesifikasyondur
- Hibernate, OpenJPA, EclipseLink gibi **implementasyonları** mevcuttur

# JPA ile Entity Tanımı

```
@Entity
@Table(name="pets")
public class Pet extends BaseEntity {

    @Id
    @GeneratedValue
    private Long id;

    @Column(name="name")
    private String name;

    @ManyToOne
    @JoinColumn(name="owner_id")
    private Owner owner;

    @OneToMany
    @JoinColumn(name="pet_id")
    @IndexColumn(name="pos_index")
    private List<Visit> visits = new ArrayList<Visit>();

    //getter ve setter tanımları, diğer business metotlar...
}
```

# JPA ile Persistence İşlemleri

```
EntityManagerFactory emf =  
Persistence.createEntityManagerFactory("test");
```

Öncelikle META-INF/persistence.xml'de Tanımlı persistence unit ismi ile belirtilerek EntityManagerFactory nesnesi yaratılır

EntityManagerFactory uygulama genelinde Bir defalık yaratılan bir nesnedir

```
<persistence>  
  <persistence-unit name="test"  
    transaction-type="RESOURCE_LOCAL">  
    <property name="hibernate.connection.driver_class"  
      value="org.hsqldb.jdbcDriver" />  
    <property name="hibernate.connection.url"  
      value="jdbc:hsqldb:mem:mem:mydb" />  
    <property name="hibernate.connection.username" value="sa" />  
    <property name="hibernate.dialect"  
      value="org.hibernate.dialect.HSQLDialect" />  
  </properties>  
</persistence-unit>  
</persistence>
```



# JPA ile Persistence İşlemleri

```
EntityManager em = emf.createEntityManager();
```

Persistence işlemlerini gerçekleştirmek için EntityManager nesnesi oluşturulur

Persistence işlemleri Persistence context üzerinden gerçekleştirilir. Persistence context Veritabanı tablo karşılığı olan entity'lerin persistent state yönetimlerinin gerçekleştirildiği nesnedir

```
EntityTransaction tx = em.getTransaction();  
tx.begin();
```

Persistence context üzerinde çalışmak için mutlaka bir transaction'ın başlatılması gerekir

```
try {  
    Pet pet = new Pet("minnoş");  
    em.persist(pet);  
    tx.commit();  
} catch (Exception ex) {  
    tx.rollback();  
    throw new RuntimeException(ex);  
}
```

# JPA ile Persistence İşlemleri

```
finally {  
    em.close();  
}
```

Persistence işlemler gerçekleştirildikten sonra Persistence Context'in kapatılması gerekir

Persistence context'in çok uzun süre açık kalması da istenilen Bir durum değildir

```
emf.close();
```

EntityManagerFactory ise uygulama her ne zaman kapatılacak ise O zaman sonlandırılır

# Container Managed Persistence Context

- EJB instance'larına **@PersistenceContext** anotasyonu ile container tarafından yönetilen **EntityManager** instance'ı enjekte edilebilir
- Bunun için mevcut bir transaction mutlaka olmalıdır
- Servlet instance'larına ise **@PersistenceUnit** anotasyonu ile **EntityManagerFactory** enjekte edilebilir

# EJB 3 ve Transaction Yönetimi

- EJB 3'de transaction yönetimi **default olarak JTA** üzerine kuruludur
- JTA'da bir transaction, **birden fazla veritabanı** ile ilişkili olabilir
- Ancak **non-JTA** (resource local) transaction yönetimi de kullanılabilir
- Resource local transaction ise sadece **tek bir veritabanı** ile ilişkilidir
- CMT ve BMT imkanı sunar

# Container Managed Transaction

@Stateless

Container Managed Transaction yönetimini  
Aktive der

```
@TransactionManagement(TransactionManagementType.CONTAINER);
```

```
public class OrderManagerBean {
```

```
@Resource
```

```
private SessionContext context;
```

→ EJBContext'i enjekte eder

```
@PersistenceContext
```

```
private EntityManager entityManager;
```

→ Container tarafından yönetilen  
EntityManager'i enjekte eder

```
@TransactionAttribute(TransactionAttributeType.REQUIRED);
```

```
public void placeOrder(Item item, Customer customer) {
```

```
try {
```

```
    //transactional işlemler...
```

```
} catch (CreditValidationException cve) {
```

```
    context.setRollbackOnly();
```

```
}
```

```
}
```

↙ Metodun TX davranışını  
düzenler

Normalde RuntimeException metod dışına çıktığı anda TX rollback olur, Herhangi bir exception fırlatılmadan TX'i metod sonlandığında rollback etmek için setRollbackOnly() metodu kullanılabilir

# Transaction Yönetimi ve Exception'lar

```
@ApplicationException(rollback=true)  
public class CreditValidationException extends Exception {  
    ...  
}
```

rollback=true attribute ile bu checked  
Exception'ı fırlatan metotlardaki TX  
Rollback edilir hale gelir

Default olarak checked exception fırlatıldığında  
TX rollback edilmez

```
@ApplicationException(rollback=false)  
public class DatabaseException extends RuntimeException {  
    ...  
}
```

rollback=false attribute ile bu unchecked  
Exception'ı fırlatan metotlardaki TX  
Rollback edilmez hale gelir

Default olarak unchecked exception  
Fırlatıldığında TX rollback edilir

# Bean Managed Transaction

```
@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class OrderManagerBean {

    @Resource
    private UserTransaction userTransaction;

    public void placeOrder(Item item, Customer customer) {
        try {
            userTransaction.begin();
            //transactional isler gercekleştirilir...
            userTransaction.commit();
        } catch (CreditValidationException cve ) {
            userTransaction.rollback();
        }
    }
}
```

# EJB 3.x Interceptor

```
public class LoggingInterceptor {
    @AroundInvoke
    public Object log(InvocationContext context) throws Exception {
        System.out.println("Metot oncesi :" +
                           context.getMethod().getName());
        Object val = context.proceed();
        System.out.println("Metot sonrasi :" +
                           context.getMethod().getName());
        return val;
    }
}
```

```
@Stateless(name="hello")
public class HelloBean implements HelloRemote, HelloLocal {
```

```
    @Interceptors(LoggingInterceptor.class)
    public String hello(String name) {
        return "Hello " + name;
    }
}
```

→  
Sınıf veya metot düzeyinde tanımlanabilir



# İletişim



[www.harezmi.com.tr](http://www.harezmi.com.tr)

[www.java-egitimleri.com](http://www.java-egitimleri.com)



[info@harezmi.com.tr](mailto:info@harezmi.com.tr)

[info@java-egitimleri.com](mailto:info@java-egitimleri.com)



[@HarezmiBilisim](https://twitter.com/HarezmiBilisim)

[@JavaEgitimleri](https://twitter.com/JavaEgitimleri)