

JEE - Java Enterprise Edition

Primary-Key Classes and Entity Relationships



Java Persistence API Primary-Key Class

- Sometimes a **primary key** has to be composed of **multiple persistent properties**
- These keys are called **composite keys**
- Composed primary keys can be declared using the **@IdClass** annotation
- The bean class does **not** use the key class **directly**, but it is used when **interacting with** the **EntityManager**
- @IdClass is a **class-level annotation**
- Designate **one or more properties**, that make out up your primaryKey using the **@Id annotation**. They must map **exactly** to the properties in the keyClass

Java Persistence API Primary-Key Class

```
public class ProtocolPK implements Serializable {  
  
    private String name;  
    private int version;  
  
    public ProtocolPK() {  
    }  
  
    public ProtocolPK(String name, int version) {  
        setName(name);  
        setVersion(version);  
    }  
  
    // ... get and set methods  
  
    public int hashCode() {  
        return getName().hashCode() + version;  
    }  
  
    public boolean equals(Object obj) {  
        if(obj == this) return true;  
        if(!obj instanceof ProtocolPK) return false;  
        ProtocolPK other = (ProtocolPK)obj;  
        if(!getName().equals(other.getName())) return false;  
        if(getVersion() != other.getVersion()) return false;  
        return true;  
    }  
}
```

implement
Serializable !

non-argument
constructor !

override hashCode()
and equals() !

Java Persistence API Primary-Key Class

```
@Entity
@IdClass(ProtocolPK.class)
public class ProtocolEntity implements Serializable {

    @Id
    private String name;
    @Id
    private int version;

    private Field[] fields;
    private Object payload;

    public ProtocolEntity() {
    }

    public String getName() {
        return name;
    }
}
```

← @IdClass Annotation

← Fields for composite key

- Primary-Key **autogeneration** is **NOT** supported for **composite keys**

Java Persistence API Primary-Key Class

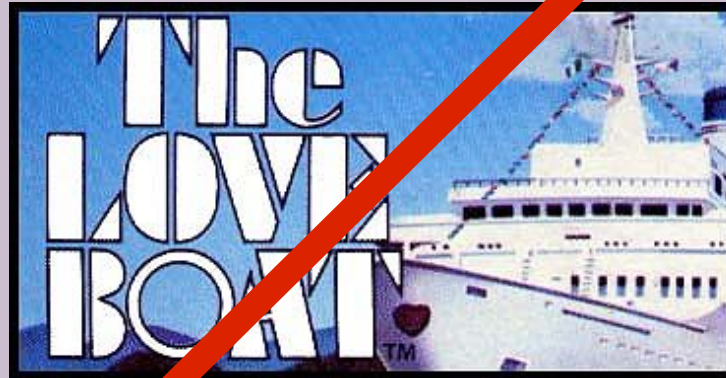
- **Use the Primary-Key Class** whenever you **query for the bean**

```
ProtocolPK primaryKey = new ProtocolPK("IP",4);  
ProtocolEntity ipProtocol = entityManager.find(ProtocolEntity.class,primaryKey);
```

- You must always use the newly created **Composite-Key Class** to **identify the entity**



Entity-Relationships



Java Persistence API Entity Relationships

- As **entity beans model real-world objects** they must be **capable of forming relationships**

| Entity A | Entity B | Relationship |
|-------------|------------|--------------|
| Course | Student | |
| Student | Address | |
| PCN | Student | |
| Car | Wheel | |
| Credit-Card | Customer | |
| Wife | Husband | |
| Wife | Ex-Husband | |



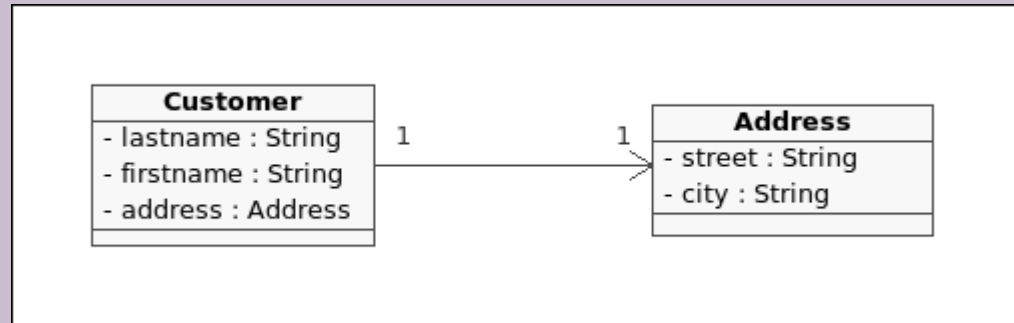
Java Persistence API Entity Relationships

- There are **seven types** of **relationships**
- **Four types** of *cardinality*
 - *one-to-one*
 - *one-to-many*
 - *many-to-one*
 - *many-to-many*
- Each relationship can be *unidirectional* or *bidirectional*
- $4 * 2 = 8$? *Many-to-one* and *One-to-Many bidirectional* are *the same!*



Java Persistence API

One-to-One unidirectional



- Tagged with **@OneToOne{CascadeType.ALL}**

```
@Entity
public class CustomerEntity implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @OneToOne(cascade={CascadeType.ALL})
    private AddressEntity address;

    /** Creates a new instance of CustomerEntity */
    public CustomerEntity() {
    }
}
```

Java Persistence API

One-to-One bidirectional

“mappedBy”-
attribute



- Tagged with **@OneToOne** on the **Owner-Side** and
- with **@OneToOne(mappedBy="creditCard")** on the other side

```
@Entity
public class CreditCardEntity implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @OneToOne(mappedBy = "creditCard")
    private CustomerEntity customer;

    /** Creates a new instance of CreditCardEntity */
    public CreditCardEntity() {
    }
}
```

Java Persistence API

One-to-One bidirectional

- **Note for all bidirectional relationships:**
Always wire both sides when changing relationships. Entities are POJOs! You have to set the values of both sides of the relationships in memory

```
CustomerEntity newCustomer = entityManager.find(CustomerEntity.class, newCustId);
CreditCardEntity card = oldCustomer.getCreditCard();
oldCustomer.setCreditCard(null);
newCustomer.setCreditCard(card);

// or ...

CustomerEntity customer = entityManager.find(CustomerEntity.class, custId);
entityManager.remove(customer.getCreditCard());
customer.setCreditCard(null);
```

Java Persistence API

One-to-Many unidirectional

- tagged with @OneToMany
- use **templated Collection** - definitions

```
@Entity
public class CustomerEntity implements Serializable {

    @OneToMany(cascade={CascadeType.ALL})
    private Collection <SaleEntity> sales;
```

- **without Generics the EntityManager cannot find the related Entity**

Java Persistence API

One-to-Many bidirectional

- same as Many-to-One bidirectional
- tagged with **@ManyToOne** on the **owner side** (required by JPA!)
- and **@OneToMany(mappedBy="...")** on the other side

```
@Entity
public class SaleEntity implements Serializable {

    @ManyToOne
    private CustomerEntity customer;
```

“mappedBy” on other side

```
@Entity
public class CustomerEntity implements Serializable {

    @OneToMany(mappedBy="customer")
    private Collection <SaleEntity> sales;
```



Java Persistence API

Other relationships

- All other relationships work corresponding to the presented relationships
 - One-To-Many unidirectional
 - Many-To-Many unidirectional
 - Many-To-Many bidirectional
- Be sure to **always wire both directions** regardless of what you are doing to avoid side-effects!!!



JEE - Java Enterprise Edition

Primary-Key Classes and
Entity Relationships -
Questions?

