@Embeddable
@Access(AccessType.FIELD)
public class Address {
    private String street;
    private String city;
    private String state;
    @Column(name="ZIP_CODE")
    private String zip;
}

@Entity
public class Employee {
    @Id private int id;
    private String name;
    private long salary;
    @Embedded private Address address;
}

Embedded Objects

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK</td>
</tr>
<tr>
<td>NAME</td>
</tr>
<tr>
<td>STREET</td>
</tr>
<tr>
<td>STATE</td>
</tr>
</tbody>
</table>

Employee
- id : int
- name : String
- salary : long

Address
- street : String
- city : String
- state : String
- zip : String

KBSS 2010
```java
@Embeddable
@Access(AccessType.FIELD)
public class Address {
    private String street;
    private String city;
    private String state;
    @Column(name="ZIP_CODE")
    private String zip;
}
```
@Entity
public class Employee {
  @Id private int id;
  private String name;
  private long salary;
  @Embedded
  @AttributeOverrides(
    @AttributeOverride(name="state", column=@Column(name="PROVINCE")),
    @AttributeOverride(name="zip", column=@Column(name="POSTAL_CODE"))
  )
  private Address address;
}

KBSS 2010
@Entity
public class Company {
  @Id private String name;
  @Embedded
  private Address address;
}
Cascade Persist

@javax.persistence.Entity
public class Employee {
    // ...
    @javax.persistence.ManyToOne(cascade=javax.persistence CascadeType.PERSIST)
    private Address address;
    // ...
}

Employee emp = new Employee();
emp.setId(2);
emp.setName("Rob");
Address addr = new Address();
addr.setStreet("164 Brown Deer Road");
addr.setCity("Milwaukee");
addr.setState("WI");
emp.setAddress(addr);
em.persist(addr);
em.persist(emp);
Persisting bidirectional relationship

```java
Department dept = em.find(Deptment.class, 101);
Employee emp = new Employee();
emp.setId(2);
emp.setName("Rob");
emp.setSalary(25000);
department.employees.add(emp); // @ManyToMany(cascade=cascadeType.PERSIST)
em.persist(dept);

!!! emp.departments still doesn't contain dept !!!

em.refresh(dept);

!!! emp.departments does contain dept now !!!
```
Cascade

List of operations supporting cascading:

- cascadeType.ALL
- cascadeType.DETACH
- cascadeType.MERGE
- cascadeType.PERSIST
- cascadeType.REFRESH
- cascadeType.REMOVE
Collection Mapping

- Collection-valued relationship (above)
  - @OneToMany
  - @ManyToMany

- Element collections
  - @ElementCollection
  - Collections of Embeddable (new in JPA 2.0)
  - Collections of basic types (new in JPA 2.0)

- Specific types of Collections are supported
  - Lists
  - Maps
@Entity
public class Employee {
    @Id private int id;
    private String name;
    private long salary;
    // ...
    @ElementCollection(targetClass=VacationEntry.class);
    private Collection vacationBookings;

    @ElementCollection
    private Set<String> nickName;
    // ...
}

@Embeddable
public class VacationEntry {
    @Temporal(TemporalType.DATE)
    private Calendar startDate;
    @Column(name="DAYS")
    private int daysTaken;
    // ...
}
@Entity
public class Employee {
    @Id private int id;
    private String name;
    private long salary;
    // ...
    @ElementCollection(targetClass=VacationEntry.class);
    private Collection vacationBookings;

    @ElementCollection
    private Set<String> nickName;
    // ...
}
@Entity
public class Employee {
    @Id private int id;
    private String name;
    private long salary;
    // ...
    @ElementCollection(targetClass=VacationEntry.class);
    @CollectionTable(
        name="VACATION",
        joinColumn=@JoinColumn(name="EMP_ID")
    )
    @AttributeOverride(name="daysTaken", column="DAYS_ABS")
    private Collection vacationBookings;

    @ElementCollection
    @Column(name="NICKNAME")
    private Set<String> nickName;
    // ...
}

@Embeddable
public class VacationEntry {
    @Temporal(TemporalType.DATE)
    private Calendar startDate;
    @Column(name="DAYS")
    private int daysTaken;
    // ...
}
Collection Mapping

@Entity
public class Employee {
    @Id private int id;
    private String name;
    private long salary;
    // ...
    @ElementCollection(targetClass=VacationEntry.class);
    @CollectionTable(
        name="VACATION”,
        joinColumn=@JoinColumn(name="EMP_ID”);
    @AttributeOverride(name="daysTaken", column="DAYS_ABS")
    private Collection vacationBookings;

    @ElementCollection
    @Column(name="NICKNAME")
    private Set<String> nickName;
    // ...
}

@Embeddable
public class VacationEntry {
    @Temporal(TemporalType.DATE)
    private Calendar startDate;
    @Column(name="DAYS")
    private int daysTaken;
    // ...
}
Collection Mapping

Interfaces:  
- Collection may be used for mapping purposes.
- Set
- List
- Map

An instance of an appropriate implementation class (HashSet, OrderedList, etc.) will be used to implement the respective property initially (the entity will be unmanaged).

As soon as such an Entity becomes managed (by calling em.persist(...)), we can expect to get an instance of the respective interface, not an instance of that particular implementation class.

When we get it back (em.find(..)) to the persistence context. The reason is that the JPA provider may replace the initial concrete instance with an alternate instance of the respective interface (Collection, Set, List, Map).
Collection Mapping – ordered List

• Ordering by Entity or Element Attribute
  ordering according to the state that exists in each entity
  or element in the List

• Persistently ordered lists
  the ordering is persisted by means of an additional
  database column(s)
  typical example – ordering = the order in which the entities
  were persisted
Collection Mapping – ordered List
(Ordering by Entity or Element Attribute)

@Entity
public class Department {
    // ...
    @OneToMany(mappedBy="department")
    @OrderBy("name ASC")
    private List<Employee> employees;
    // ...
}
Collection Mapping – ordered List
(Ordering by Entity or Element Attribute)

```java
@Entity
public class Department {
    // ...
    @OneToMany(mappedBy="department")
    @OrderBy("info.name ASC")
    private List<Employee2> employees;
    // ...
}
```
@Entity
public class PrintQueue {
    @Id private String name;
    // ...
    @OneToMany(mappedBy="queue")
    @OrderColumn(name="PRINT_ORDER")
    private List<PrintJob> jobs;
    // ...
}
Collection Mapping – ordered List
(Persistently ordered lists)

@Entity
public class PrintQueue {
    @Id private String name;
    // ...  
    @OneToMany(mappedBy="queue")
    @OrderColumn(name="PRINT_ORDER")
    private List<PrintJob> jobs;
    // ...
}

This annotation need not be necessarily on the owning side
Collection Mapping – Maps

Map is an object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value.

**Keys:**
- Basic types (stored directly in the table being referred to)
  - Target entity table
  - Join table
  - Collection table
- Embeddable types ( - “ - )
- Entities (only foreign key is stored in the table)

**Values:**
- Values are entities => Map must be mapped as a one-to-many or many-to-many relationship
- Values are basic types of embeddable types => Map is mapped as an element collection
Collection Mapping – Maps
(keying by basic type – key is String)

@Entity
public class Employee {
    @Id private int id;
    private String name;
    private long salary;

    @ElementCollection
    @CollectionTable(name="EMP_PHONE")
    @MapKeyColumn(name="PHONE_TYPE")
    @Column(name="PHONE_NUM")
    private Map<String, String> phoneNumbers;
    // ...
}

KBSS 2010
@Entity
public class Employee {
    @Id private int id;
    private String name;
    private long salary;

    @ElementCollection
    @CollectionTable(name="EMP_PHONE")
    @MapKeyEnumerated(EnumType.String)
    @MapKeyColumn(name="PHONE_TYPE")
    @Column(name="PHONE_NUM")
    private Map<
            PhoneType,
            String>
            phoneNumbers;

    // ...
}

Public enum PhoneType {
    Home,
    Mobile,
    Work
}
Collection Mapping – Maps
(keying by basic type – 1:N relationship using a Map with String key)

@Entity
public class Department {
    @Id private int id;
    private String name;

    @OneToMany(mappedBy=“department“)
    @MapKeyColumn(name=“CUB_ID“)
    private Map<String, Employee> employeesByCubicle;
    // ...
}
Collection Mapping – Maps
(keying by basic type – N:M relationship using a Map with String key)

@Entity
public class Department {
    @Id private int id;
    private String name;

    @ManyToMany
    @JoinTable(name="DEPT_EMP",
                joinColumns=@JoinColumn(name="DEPT_ID"),
                inverseJoinColumns=@JoinColumn(name="EMP_ID"))
    @MapKeyColumn(name="CUB_ID")
    private Map<String, Employee> employeesByCubicle;

    // ...
}
@Entity
public class Department {
    // ...
    @OneToMany(mappedBy="department")
    @MapKey(name="id")
    private Map<Integer, Employee> employees;
    // ...
}
Collection Mapping – Maps
(keying by embeddable type)

@Entity
public class Department {
    // ...
    @OneToMany(mappedBy="department")
    @MapKey(name="id")
    private Map<EmployeeName, Employee> employees;
    // ...
}

@Entity
public class Employee {
    @Id private int id;
    @Column(name="F_NAME")
    private String firstName;
    @Column(name="L_NAME")
    private String lastName;
    private long salary;
    // ...
}

@Embeddable
public class EmployeeName {
    @Column(name="F_NAME", insertable=false, updatable=false)
    private String first_Name;
    @Column(name="L_NAME", insertable=false, updatable=false)
    private String last_Name;
    // ...
}

Sharing columns => insertable=false and updatable = false
@Entity
public class Employee {
    @Id private int id;

    @Embedded
    private EmployeeName name;
    private long salary;
    // ...
}

@Embeddable
public class EmployeeName {
    @Column(name="F_NAME", insertable=false, updateable=false)
    private String firstName;

    @Column(name="L_NAME", insertable=false, updateable=false)
    private String lastName;
    // ...
}

@Entity
public class Department {
    // ...
    @OneToMany(mappedBy="department")
    @MapKey(name="id")
    private Map<EmployeeName, Employee> employees;
    // ...
}
Collection Mapping – Maps
(keying by embeddable type)

@Entity
public class Department {
    @Id private int id;
    @ManyToMany
    @JoinTable(name="DEPT_EMP",
        joinColumns=@JoinColumn(name="DEPT_ID"),
        inverseJoinColumns=@JoinColumn(name="EMP_ID"))
    @AttributeOverrides({
        @AttributeOverride(
            name="first_Name",
            column=@Column(name="EMP_FNAME")),
        @AttributeOverride(
            name="last_Name",
            column=@Column(name="EMP_LNAME"))},
    private Map<EmployeeName, Employee> employees;
    // …
@Entity
public class Department {
    @Id private int id;

    @ManyToMany
    @JoinTable(name="DEPT_EMP",
                joinColumns=@JoinColumn(name="DEPT_ID"),
                inverseJoinColumns=@JoinColumn(name="EMP_ID"))
    @AttributeOverrides({
        @AttributeOverride(
            name="first_Name",
            column=@Column(name="EMP_FNAME")),
        @AttributeOverride(
            name="last_Name",
            column=@Column(name="EMP_LNAME"))
    })
    private Map<EmployeeName, Employee> employees;
    // ...
}
We have to distinguish, if we are overriding embeddable attributes of the key or the value.

```java
@Entity
public class Department {
    @Id private int id;
    @AttributeOverrides({
        @AttributeOverride(name="key.first_Name", column=@Column(name="EMP_FNAME")),
        @AttributeOverride(name="key.last_Name", column=@Column(name="EMP_LNAME"))
    })
    private Map<EmployeeName, EmployeeInfo> employees;
    // ...
}
```

The embeddable attributes will be stored in the collection table (rather than in a join table As it was on the previous slide).
Collection Mapping – Maps
(keying by entity)

@Entity
class Department {
    @Id private int id;
    private String name;
    // ...
    @ElementCollection
    @CollectionTable(name="EMP SENIORITY")
    @MapKeyJoinColumn(name="EMP_ID")
    @Column(name="SENIORITY")
    private Map<Employee, Integer> employees;
    // ...
}