**UNO A UNO – BIDIRECCIONAL – 01**

|  |  |
| --- | --- |
| **SOURCE** | **TARGET** |
|  | |
| * The entity table that contains the join column is the **owner** of the relationship. * In a bidirectional one to one relationship, either side can be the **owner**, so the join column might end up being on one side or the other. * This would be a data modeling decision, not a Java programming decision, and it would likely be decides based on the most frequent direction of traversal. |  |
|  | |
| * This example assumes that **Employee** is the **owning** side of the relationship. |  |
|  |  |
|  | * We now have to add a reference from **ParkingSpace** back to Employee. * This is achived by adding the @OneToOne relationship annotation on the **employee** field. * As part of the annotation, we must add a **mappedBy** element to indicate that **ParkingSpace** is the **inverse side**, is the **non-owning** side of the relationship. * Because ParkingSpace is the inverse side on the reñationship, is does not have to supply the join column information. |
|  |  |
|  | * The **mappedBy** element in the one to one mapping of the **employee** attribute of **ParkingSpace** is needed to refer to the **parkingSpace** attribute in the Employee class. * The value of **mappedBy** is the name of the attribute in the owning entity that points back to the inverse entity. |