Relationships

- one-to-one
- one-to-many
- many-to-one
- many-to-many
- compositions?
One-to-One

- Simple relationship
- Use a property of the type of another domain class
- Changes and updates will be cascaded automatically
One-to-One

class Face{
    Nose nose
}

class Nose{
    Face face
}

• Unidirectional relationship
• Deletion is not cascaded (the Nose will NOT be deleted when the face is deleted)
class Face{
    Nose nose
}

class Nose{
    static belongsTo = Face
    Face face
}
Two way relationships

Two way relationships are NOT resolved automatically

myFace.nose = bigNose

...will NOT automatically do

bigNode.face = myFace

!! YOU HAVE TO IMPLEMENT THIS YOURSELF !!!
class Author{
    static hasMany = [ books : Book]
    String name
}

• add the mapping to the hasMany map
• results in an implicit property
• you can define the property explicitly (optional)
• the owning side is assumed to be the ”one” side
Many-to-One

class Book{
    Author author
    String name
}

• many-to-one relationships are defined implicitly
Multiple one2many

class Author{
    static hasMany = [ books : Book, coBooks: Book ]
    static mappedBy = [ books : ”mainAuthor”,
                        coBooks : ”coAuthor” ]

    String name
}

class Book{
    static belongsTo = Author
    Author mainAuthor
    Author coAuthor

    String title
}
In other words

- an owner can have belongings
- `belongsTo` property refers to the owner(s)
- if an owner dies, all belongings die if the owner has a direct reference or an explicit `hasMany` relationship
Many-to-Many

- Use ‘hasMany’ on both sides of the relationship

```ruby
class Book{
    static belongsTo = Author
    static hasMany = [authors:Author]
}
class Author{
    static hasMany = [books:Book]
}
```
Many2Many updates

- Updates are cascaded only on the owning side!

new Author(...).addToBooks(new Book()).save() works!

myBook.addToAuthors(myAuthor); myBook.save() will NOT update the Author!

- Only save and update are cascaded - no deletes!