

# Mobile Application Development

Higher Diploma in Science in Computer Science

---

Produced  
by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics  
Waterford Institute of Technology

<http://www.wit.ie>

<http://elearning.wit.ie>



Waterford Institute of Technology  
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE



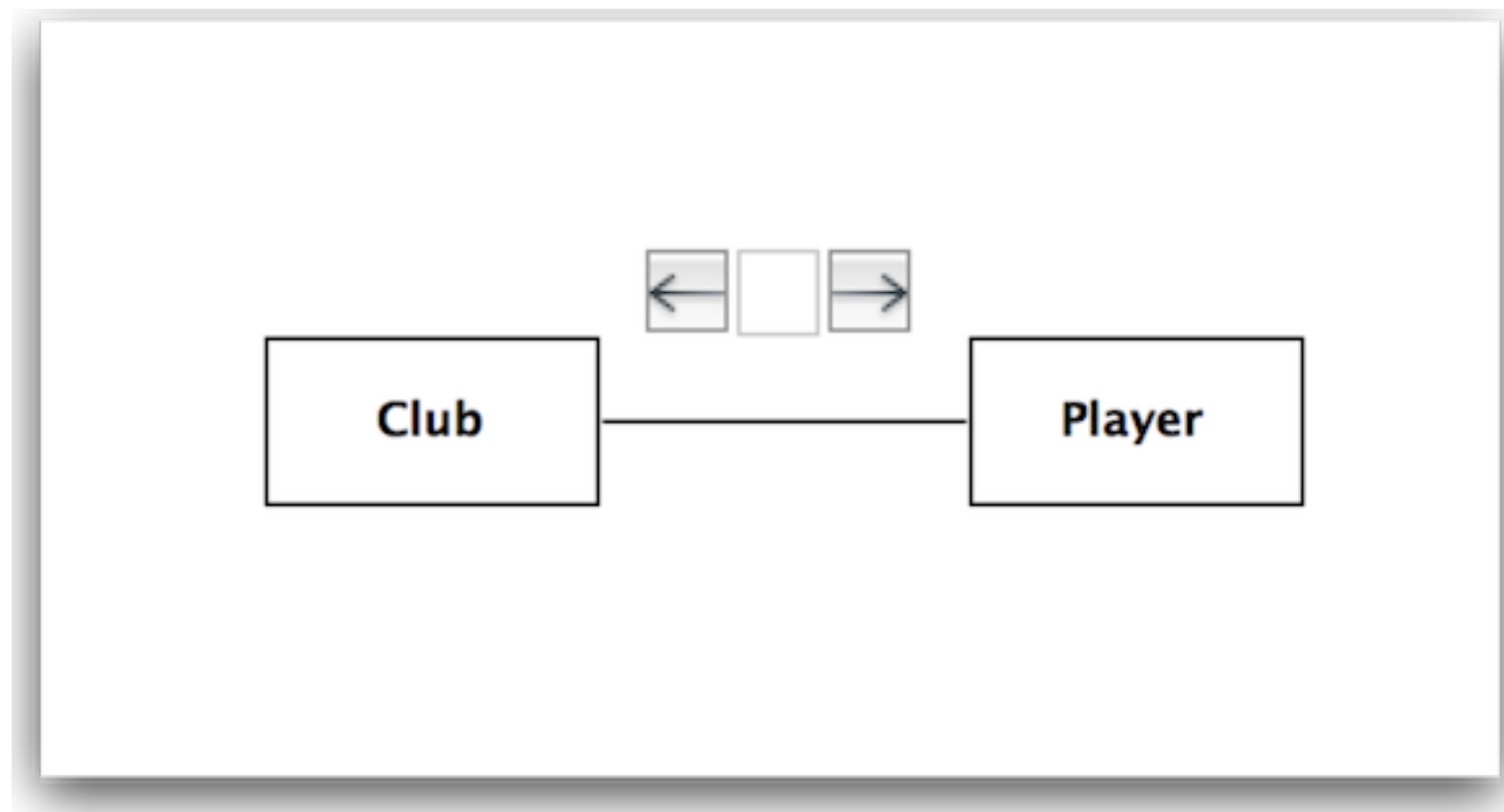
# Modeling Relationships

---

# Associations

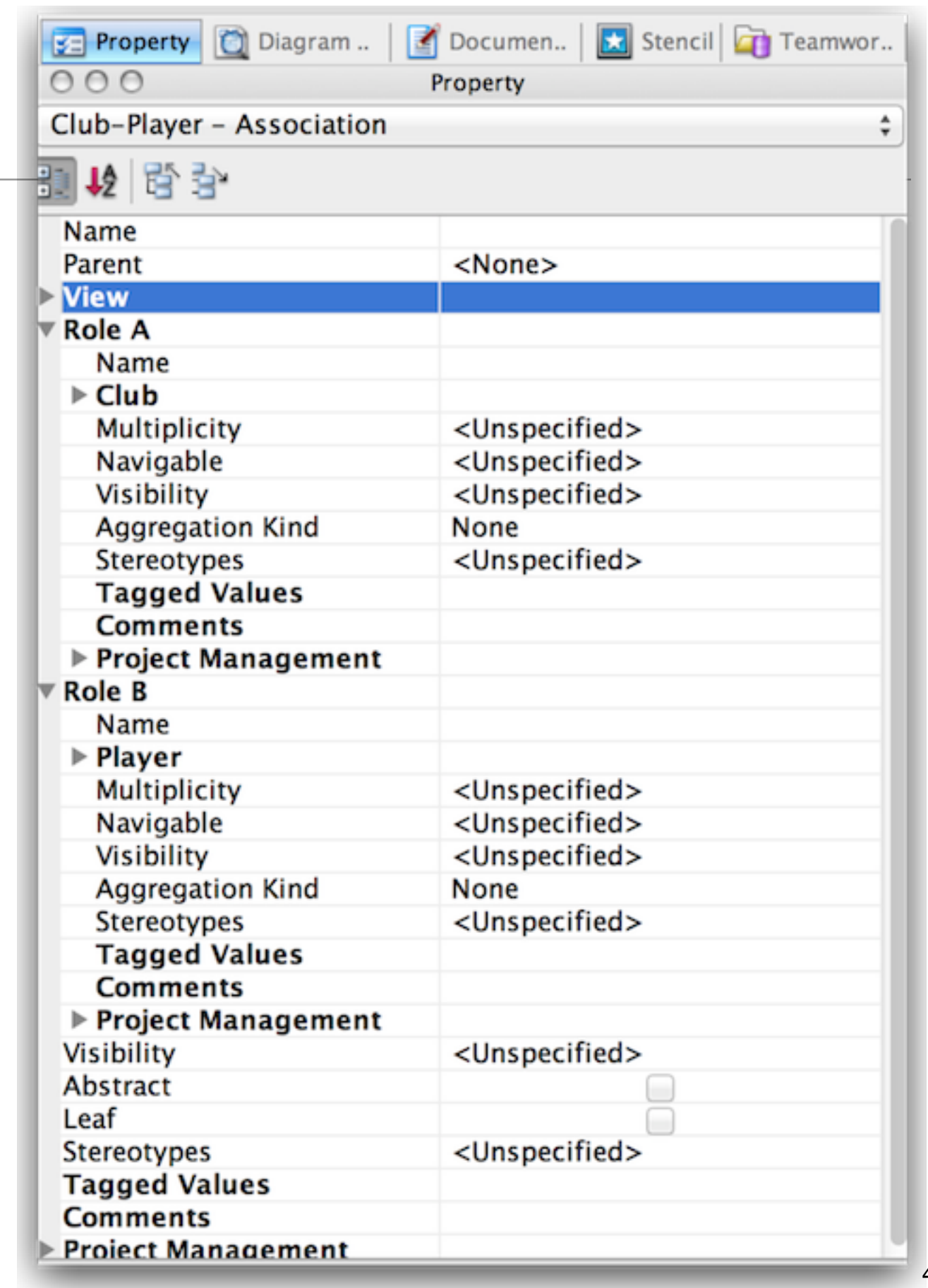
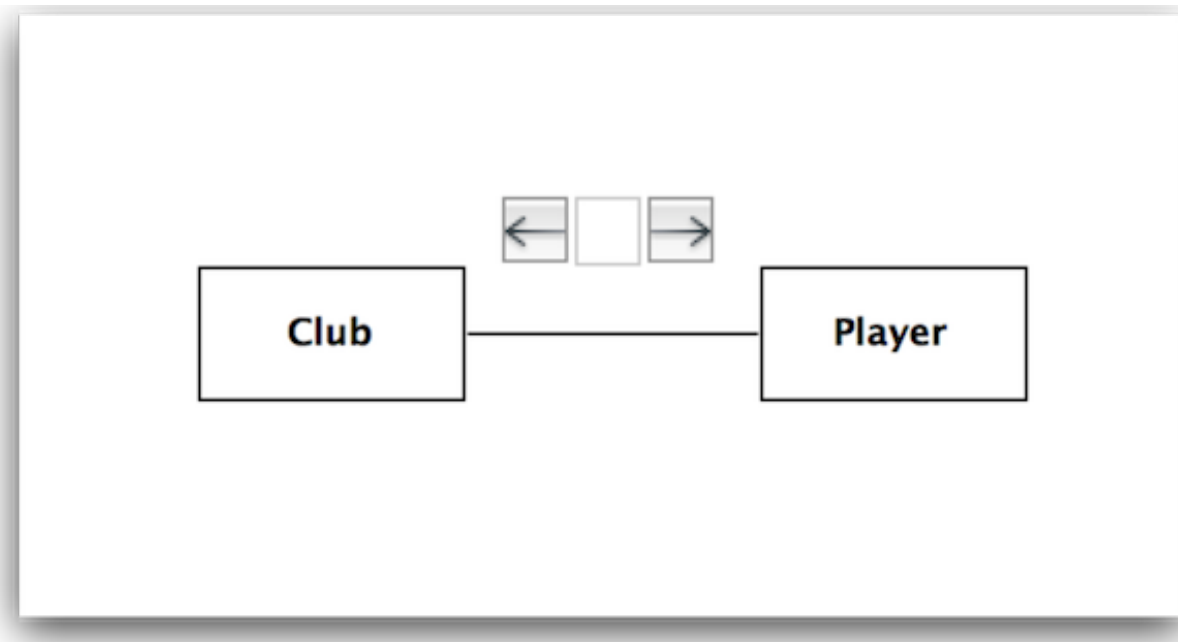
---

- In Visual Paradigm, on the palette on the left, select the 'association' element and use it to connect Club and Player.



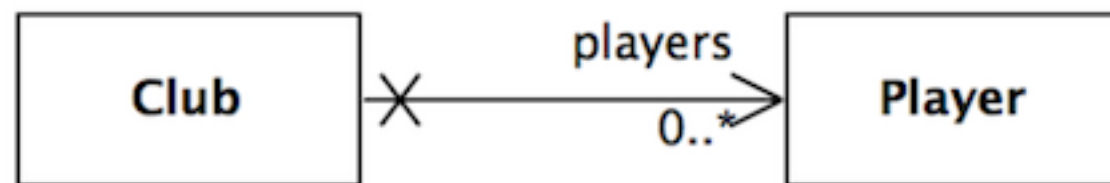
# Association Attributes

- Select the association (the line), and locate the following panel:

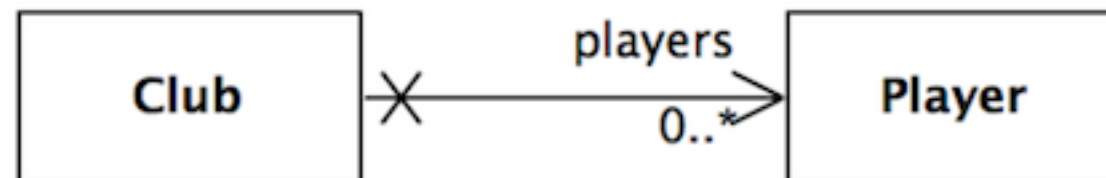


# Multiplicity & Navigation

---



- Club has a collection of zero or more players
- Players are unaware of Club



▼ <b>Role A</b>	
Name	
► <b>Club</b>	
Multiplicity	<Unspecified>
<b>Navigable</b>	<b>False</b>
Visibility	<Unspecified>
Aggregation Kind	None
Stereotypes	<Unspecified>
<b>Tagged Values</b>	
<b>Comments</b>	

▼ <b>Project management</b>	
▼ <b>Role B</b>	
Name	players
► <b>Player</b>	
Multiplicity	0..*
Navigable	True
Visibility	<Unspecified>
Aggregation Kind	None
Stereotypes	<Unspecified>
<b>Tagged Values</b>	
<b>Comments</b>	

# Implementation Relationship in Java Classes

```
public class Club extends Model
{
    public String name;

    @OneToMany(cascade=CascadeType.ALL)
    public List<Player> players;

    public Club(String name)
    {
        this.name = name;
        this.players = new ArrayList<Player>();
    }

    public String toString()
    {
        return name;
    }

    public void addPlayer(Player player)
    {
        players.add(player);
    }
}
```

```
public class Player extends Model
{
    public String name;

    public Player(String name)
    {
        this.name = name;
    }

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

# Testing the Player / Club Relationship

---

- Use the fixture to set up some club / relationships

```
@Before
public void setup()
{
    p1 = new Player("mike");
    p2 = new Player("jim");
    p3 = new Player("frank");

    c1 = new Club("tramore");
    c2 = new Club("dunmore");
    c3 = new Club("fenor");

    c1.addPlayer(p1);
    c1.addPlayer(p2);

    c1.save();
    c2.save();
    c3.save();
}
```



# testPlayers

---

- In the test, see if these relationship have been established

```
@Test
public void testPlayers()
{
    Club tramore = Club.findByName("tramore");

    assertEquals (2, tramore.players.size());

    Player mike  = Player.findByName("mike");
    Player jim   = Player.findByName("jim");
    Player frank = Player.findByName("frank");

    assertTrue (tramore.players.contains(mike));
    assertTrue (tramore.players.contains(jim));
    assertFalse (tramore.players.contains(frank));
}
```

# testRemovePlayers

---

- Removing relationships must also be tested

```
@Test
public void testRemovePlayer()
{
    Club tramore = Club.findByName("tramore");
    assertEquals(2, tramore.players.size());

    Player mike = Player.findByName("mike");
    assertTrue(tramore.players.contains(mike));
    tramore.players.remove(mike);
    tramore.save();

    Club c = Club.findByName("tramore");
    assertEquals(1, c.players.size());

    mike.delete();
}
```

# Explore the Relationship in the Database

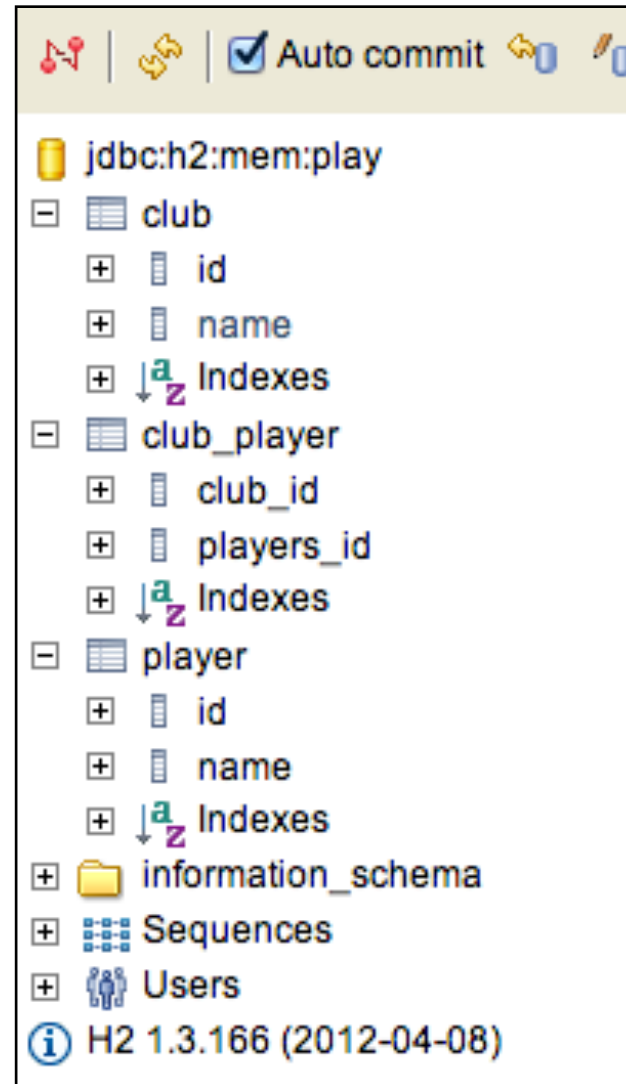
@Before

```
public void setup()
{
    p1 = new Player("mike");
    p2 = new Player("jim");
    p3 = new Player("frank");

    c1 = new Club("tramore");
    c2 = new Club("dunmore");
    c3 = new Club("fenor");

    c1.addPlayer(p1);
    c1.addPlayer(p2);

    c1.save();
    c2.save();
    c3.save();
}
```



SELECT \* FROM CLUB;

ID	NAME
1	tramore
2	dunmore
3	fenor

(3 rows, 3 ms)

SELECT \* FROM PLAYER;

ID	NAME
1	mike
2	jim

(2 rows, 2 ms)

Edit

SELECT \* FROM CLUB\_PLAYER;

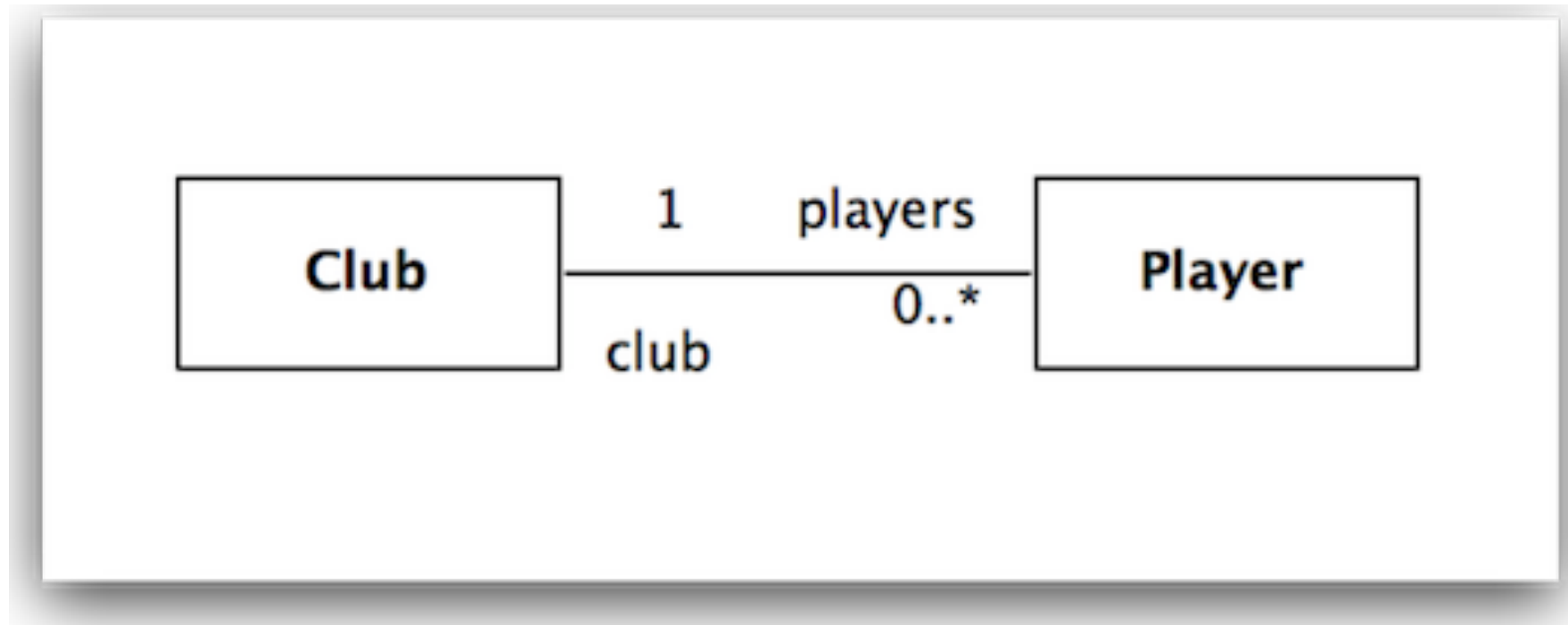
CLUB_ID	PLAYERS_ID
1	1
1	2

(2 rows, 4 ms)

Edit

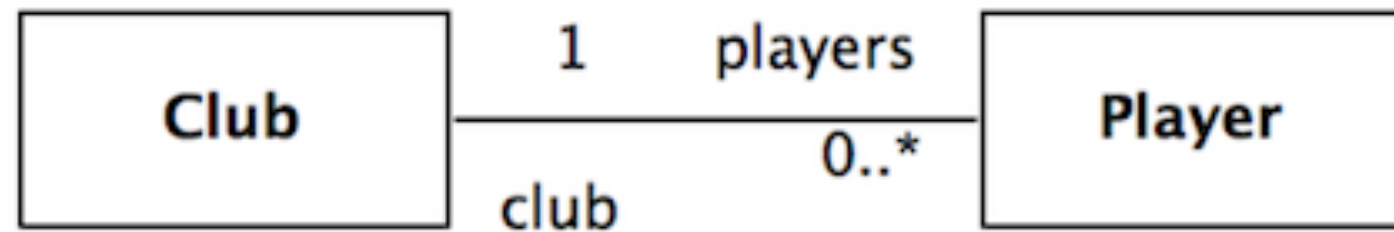
# Bidirectional Relationship

---



- Club has a 'one to many' relationship with players
- Player has a 'many to one' relationship with club

# Bidirectional Relationship



▼ <b>Role A</b>	
Name	club
▶ <b>Club</b>	
Multiplicity	1
Navigable	True
Visibility	<Unspecified>
Aggregation Kind	None
Stereotypes	<Unspecified>
Tagged Values	
Comments	

# Bidirectional Relationship in Java Classes

```
public class Club extends Model
{
    public String name;

    @OneToMany(mappedBy="club", cascade=CascadeType.ALL)
    public List<Player> players;

    public Club(String name)
    {
        this.name = name;
        this.players = new ArrayList<Player>();
    }

    public String toString()
    {
        return name;
    }

    public void addPlayer(Player player)
    {
        player.club = this;
        players.add(player);
    }
}
```

```
public class Player extends
Model
{
    public String name;

    @ManyToOne
    public Club club;

    public Player(String name)
    {
        this.name = name;
    }

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

| Auto commit

jdbc:h2:mem:play

- club
  - + id
  - + name
  - + Indexes
- player
  - + id
  - + name
  - + club\_id
  - + Indexes
- + information\_schema
- + Sequences
- + Users

H2 1.3.166 (2012-04-08)

SELECT \* FROM CLUB;

ID	NAME
1	tramore
2	dunmore
3	fenor

(3 rows, 3 ms)

Edit

SELECT \* FROM PLAYER;

ID	NAME	CLUB_ID
1	mike	1
2	jim	1

(2 rows, 2 ms)

Edit

# Unidirectional Relationship in Java Classes

```
public class Club extends Model
{
    public String name;

    @OneToMany(cascade=CascadeType.ALL)
    public List<Player> players;

    public Club(String name)
    {
        this.name = name;
        this.players = new ArrayList<Player>();
    }

    public String toString()
    {
        return name;
    }

    public void addPlayer(Player player)
    {
        players.add(player);
    }
}
```

```
public class Player extends Model
{
    public String name;

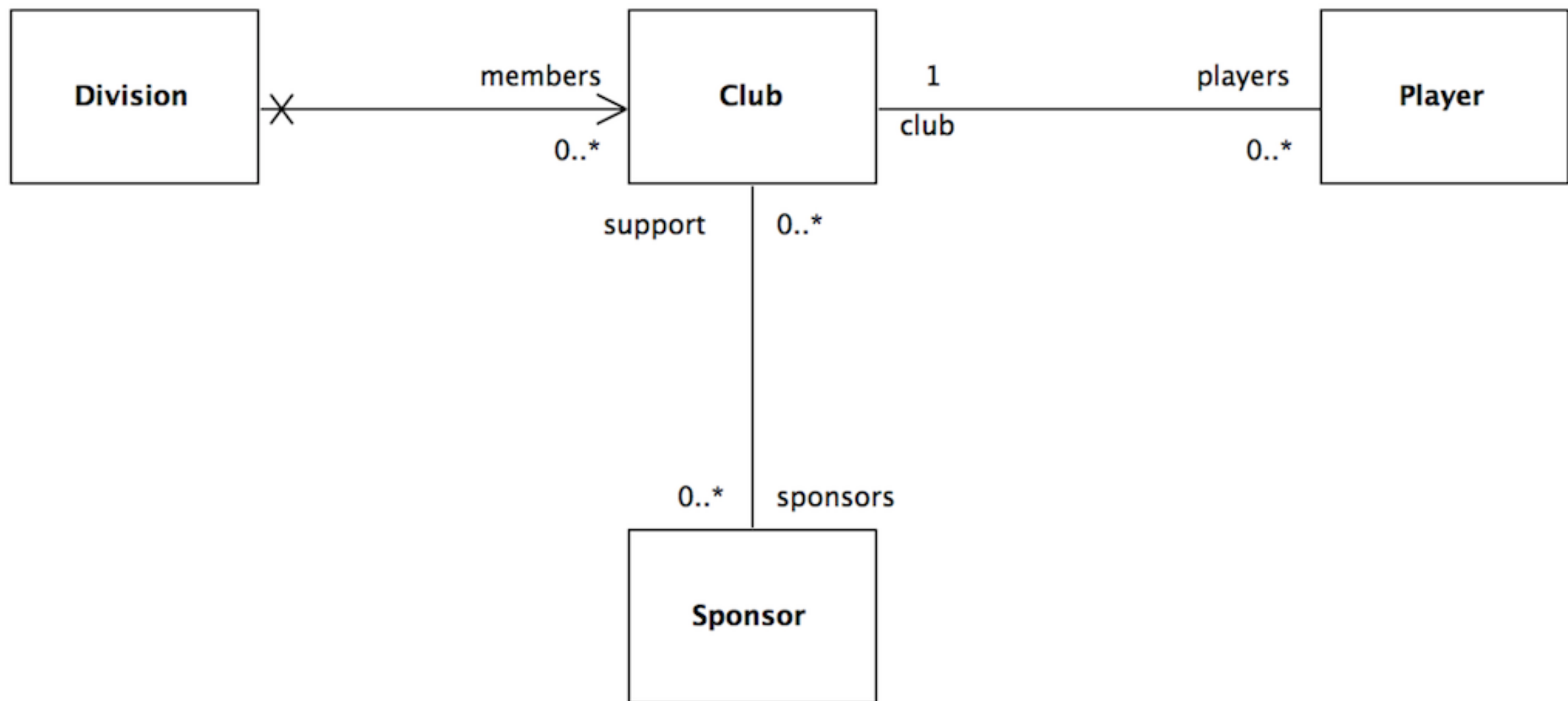
    public Player(String name)
    {
        this.name = name;
    }

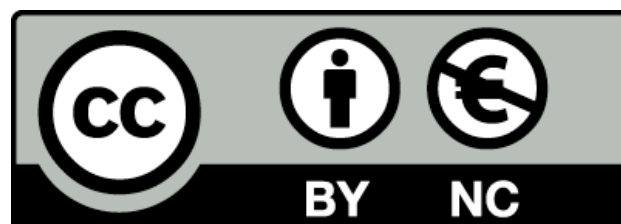
    public String toString()
    {
        return name;
    }
}
```



# Exercise: Model This:

---





Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see <http://creativecommons.org/licenses/by-nc/3.0/>



Waterford Institute of Technology  
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

