

# App Development & Modeling

BSc in Applied Computing

---

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 LÁIRCE



# UML & JPA Modeling

---

# Agenda

---

- Introduce UML Class Diagram modeling using Visual Paradigm
- Define a simple model and implement it in Play
- Write comprehensive unit tests to exercise the model

## JPA Model Project





















---

Start by creating a brand new Play project. Do this by determining the parent folder (most likely your workspace) and running a command prompt. Then type:

```
play new jpamodel
```

Once this has completed, change into the folder just created (jpamodel) and run the eclipsify command:

```
cd jpamodel  
play eclipsify
```

- ▼  jpamodel
  - ▼  app
    - ▶  controllers
    - ▶  models
    - ▶  views
  - ▼  test
    - ▶  (default package)
      -  Application.test.html
      -  data.yml
    - ▶  crud
    - ▶  JRE System Library [Java SE 7 (MacOS
    - ▶  Referenced Libraries
  - ▼  conf
    - ▶  crud
      -  application.conf
      -  dependencies.yml
      -  messages
      -  routes
    -  eclipse
    - ▶  public

# Visual Paradigm



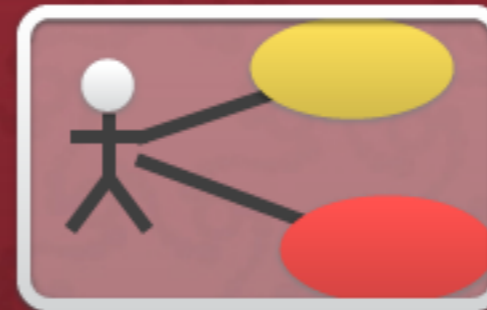
Visual Paradigm  
for UML

## Model-Code-Deploy Platform

- An intuitive interface helps hit the ground running with deliverables
- Able to scale to best fit your needs
- Effortless translation between design and code

What VP-UML Provides

Tutorials



### 🔍 Requirements Capturing

Capture system requirements with use case diagram, SysML requirement diagrams and textual analysis.



### 🔍 Software Design

Design system structure with Class Diagram, composite structure diagram. Model interactions with Sequence Diagram.



### 🔍 Database and Code Generation

Design database with entity relationship diagram. Generate UML class diagram.

# Create new Model

UML version: UML 2.x

Language: UML

Properties | Template | Other CASE Tools Project

UML version: UML 2.x

Project name: jpamodel

Author: edeleastar

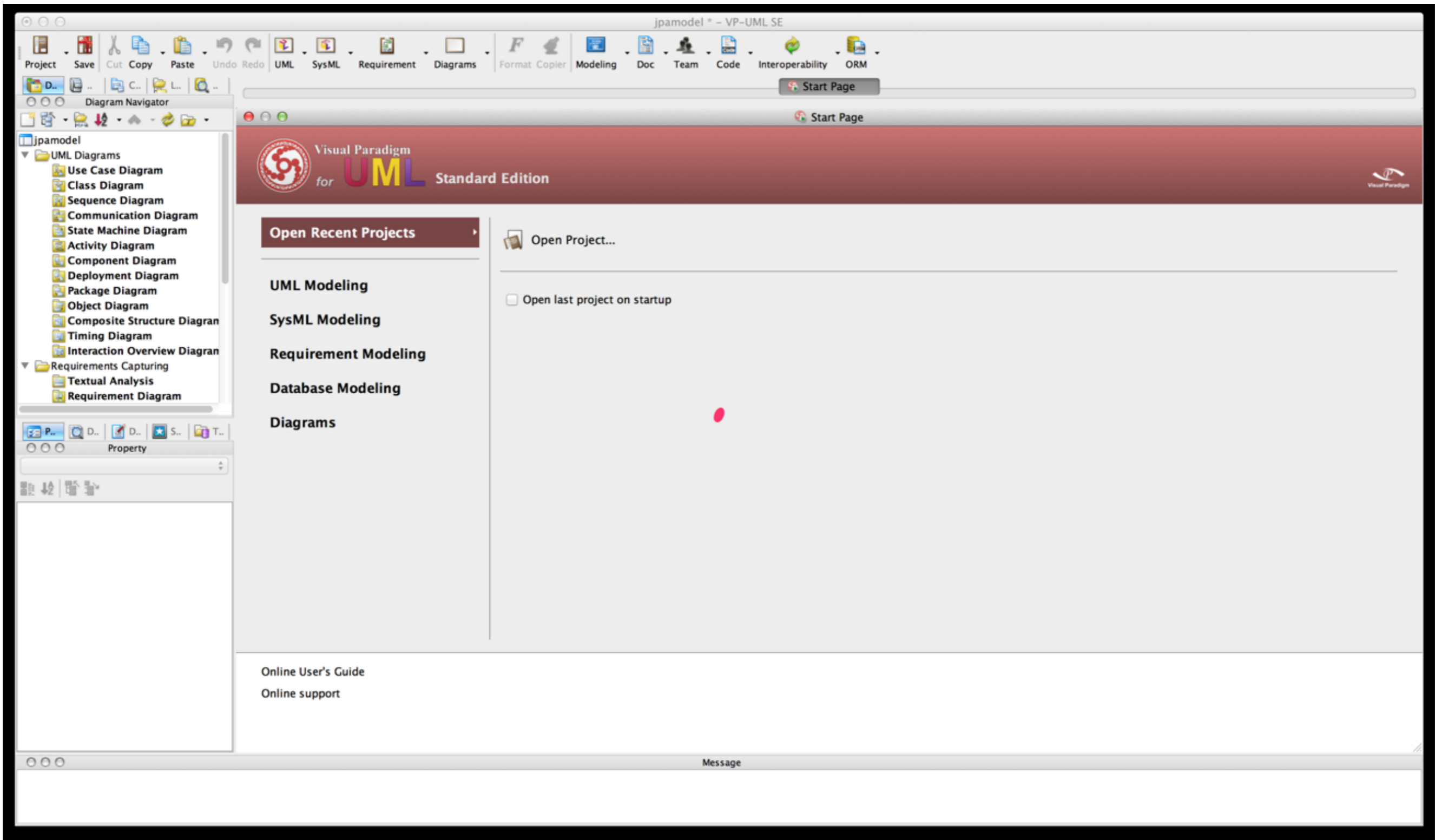
Company:

Create Project Management Lookup

Project description:

B I U F F

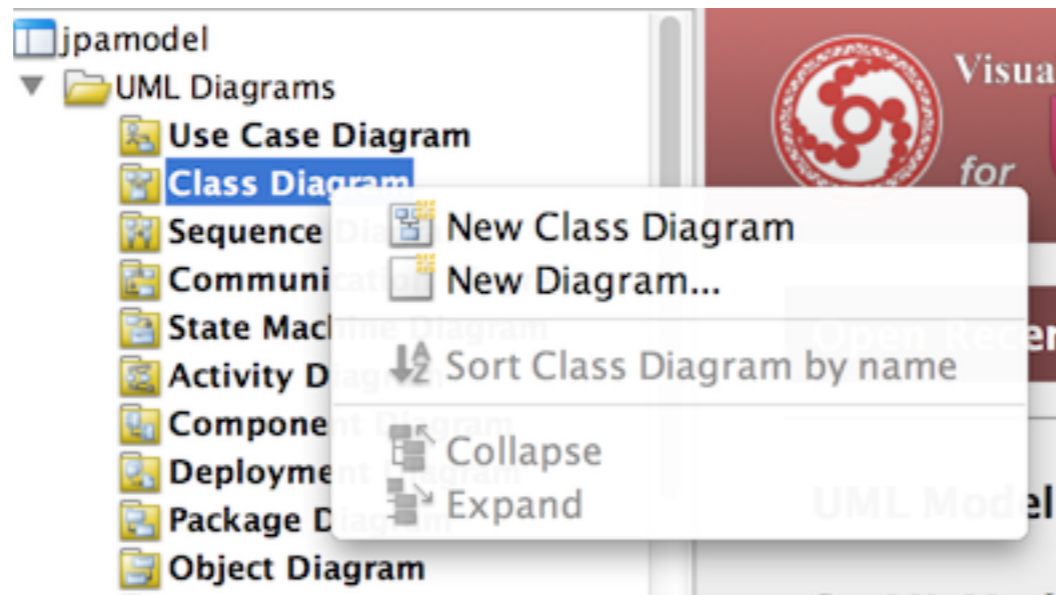
Create Blank Project Cancel Help



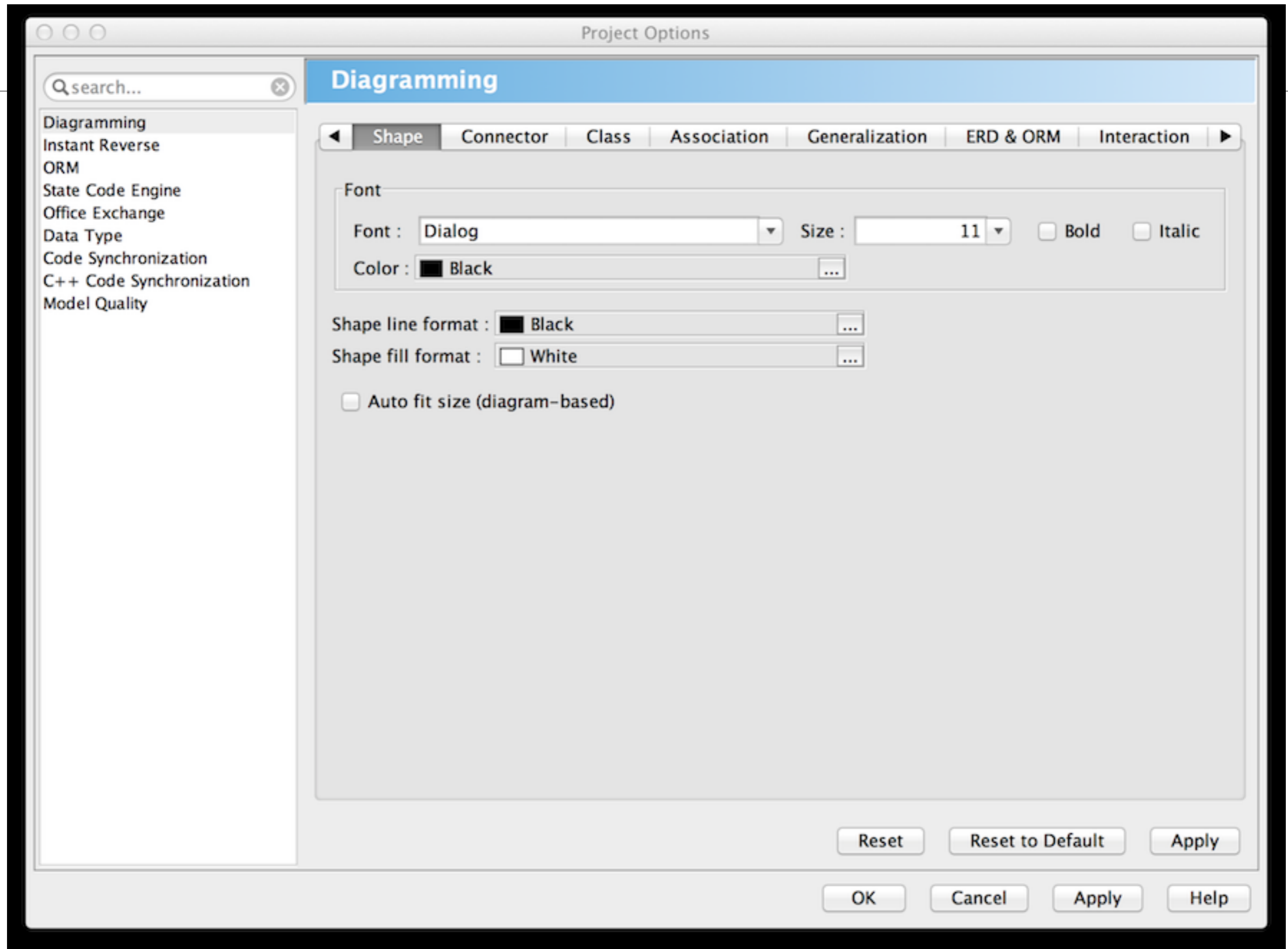


# Create New Class Diagram

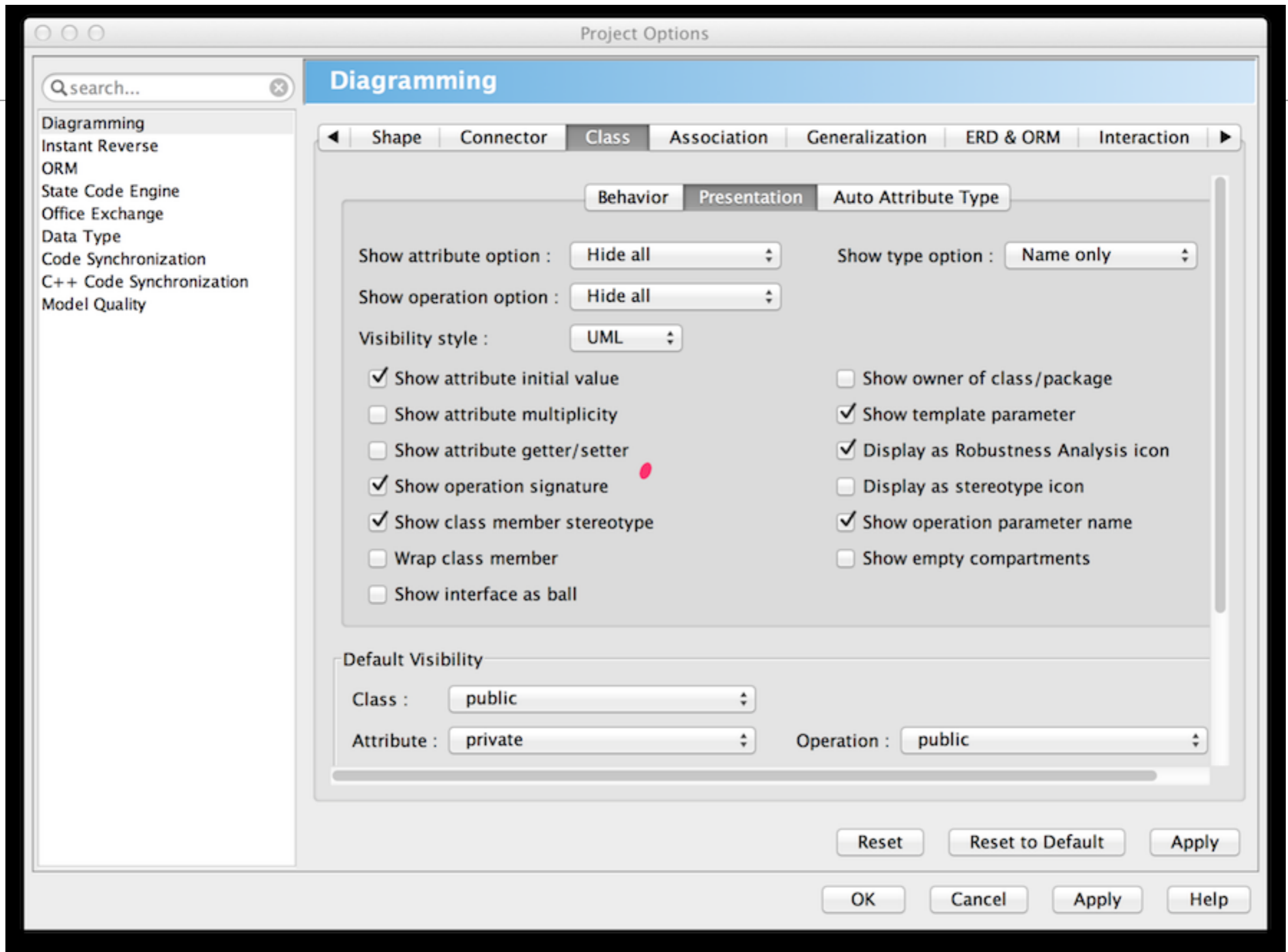
---



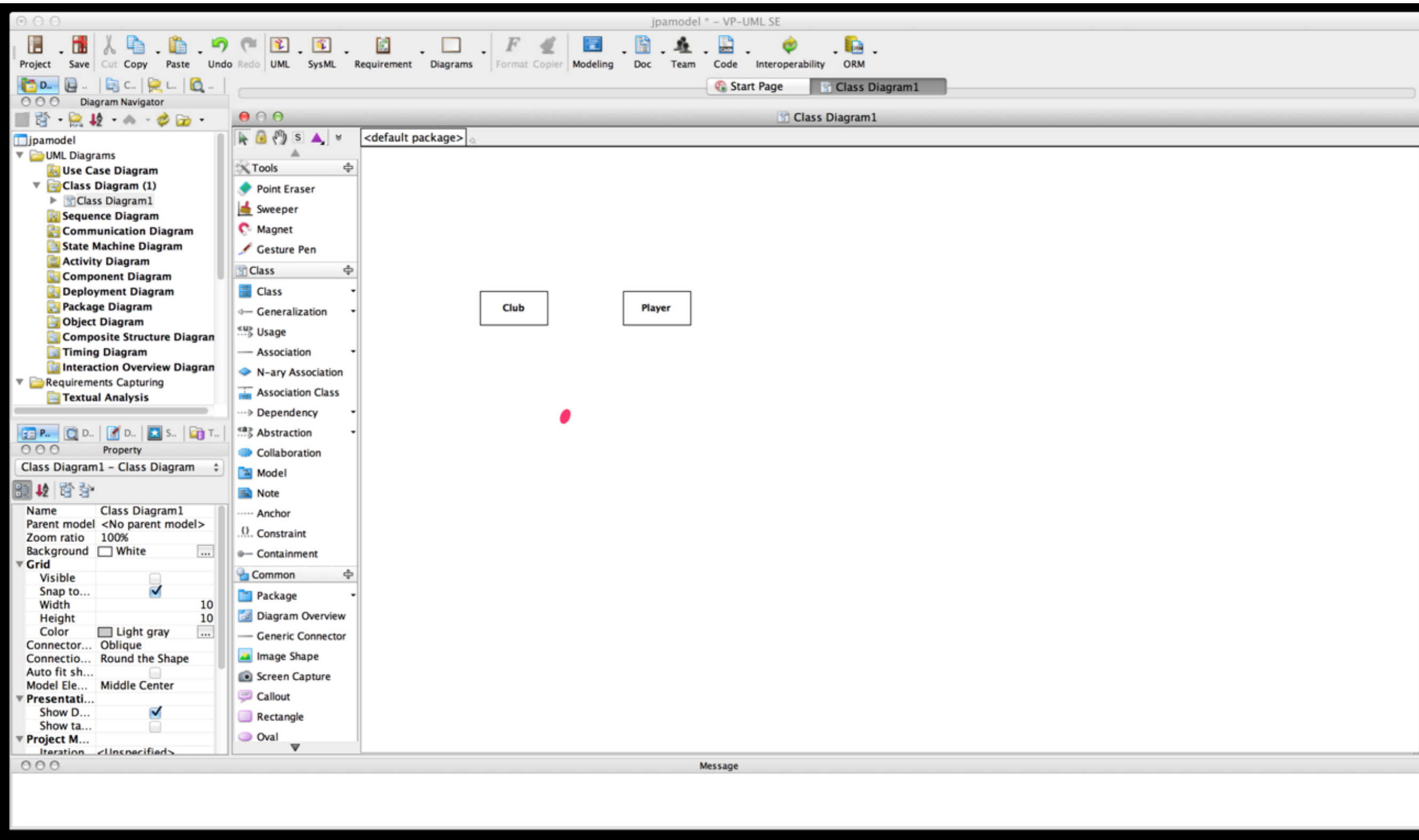
# Customize Visual Paradigm - Shape Fill Format



# Customize Visual Paradigm - 'Show' options...



# Create 2 classes



# Club Class

```
package models;

import javax.persistence.Entity;
import play.db.jpa.Model;

@Entity
public class Club extends Model
{
    public String name;

    public Club(String name)
    {
        this.name = name;
    }
}
```

# Player Class

```
package models;

import javax.persistence.Entity;
import play.db.jpa.Model;

@Entity
public class Player extends Model
{
    public String name;

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

# ClubTest

---

```
import org.junit.*;

import java.util.*;
import play.test.*;
import models.*;

public class ClubTest extends UnitTest
{
    @Before
    public void setup()
    {
    }

    @After
    public void teardown()
    {
    }

    @Test
    public void testCreate()
    {
    }

}
}
```

# PlayerTest

---

```
import org.junit.*;

import java.util.*;
import play.test.*;
import models.*;

public class PlayerTest extends UnitTest
{
    @Before
    public void setup()
    {
    }

    @After
    public void teardown()
    {
    }

    @Test
    public void testCreate()
    {
    }

}
}
```



Run the app now in 'test' mode:

```
play test
```

...and navigate to the test runner page:

- <http://localhost:9000/@tests>

Select the Club and Player tests - and they should be green.

Also try the database interface:

- <http://localhost:9000/@db>

# PlayerTest

---

```
public class PlayerTest extends UnitTest
{
    private Player p1, p2, p3;

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

    @After
    public void teardown()
    {
        p1.delete();
        p2.delete();
        p3.delete();
    }

    @Test
    public void testCreate()
    {
    }
}
```

Auto commit  | Max rows: 1000 | Auto complete Normal

- jdbc:h2:mem:play
  - club
  - player
  - information\_schema
  - Sequences
  - Users
- H2 1.3.166 (2012-04-08)

Run (Ctrl+Enter) Clear SQL statement:

---

---

There is currently no running statement

# toString + //@After

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

    @ManyToOne
    public Club club;

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

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

- We can use db interface while project is in 'test' mode
- Enables us to understand model as we evolve classes and their relationships

```
public class PlayerTest extends UnitTest
{
    private Player p1, p2, p3;

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

//@After

```
public void teardown()
{
    p1.delete();
    p2.delete();
    p3.delete();
}

@Test
public void testCreate()
{
    Player a = Player.findByName("mike");
    assertNotNull(a);
    assertEquals("mike", a.name);
    Player b = Player.findByName("jim");
    assertNotNull(b);
    assertEquals("jim", b.name);
    Player c = Player.findByName("frank");
    assertNotNull(c);
    assertEquals("frank", c.name);
}
```

- jdbc:h2:mem:play
- club
- player
- information\_schema
- Sequences
- Users
- H2 1.3.166 (2012-04-08)

Run (Ctrl+Enter) Clear SQL statement:

SELECT \* FROM PLAYER

SELECT \* FROM PLAYER;

ID	NAME	CLUB_ID
1	mike	<i>null</i>
2	jim	<i>null</i>
3	frank	<i>null</i>
4	mike	<i>null</i>
5	jim	<i>null</i>
6	frank	<i>null</i>

(6 rows, 2 ms)

Edit

```
private Player p1, p2, p3;  
  
public void setup()  
{  
    p1 = new Player("mike");  
    p2 = new Player("jim");  
    p3 = new Player("frank");  
    p1.save();  
    p2.save();  
    p3.save();  
}
```

# Some Player Tests

---

```
@Test
public void testCreate()
{
    Player a = Player.findByName("mike");
    assertNotNull(a);
    assertEquals("mike", a.name);
    Player b = Player.findByName("jim");
    assertNotNull(b);
    assertEquals("jim", b.name);
    Player c = Player.findByName("frank");
    assertNotNull(c);
    assertEquals("frank", c.name);
}

@Test
public void testNotThere()
{
    Player a = Player.findByName("george");
    assertNull(a);
}
```

# ClubTest

---

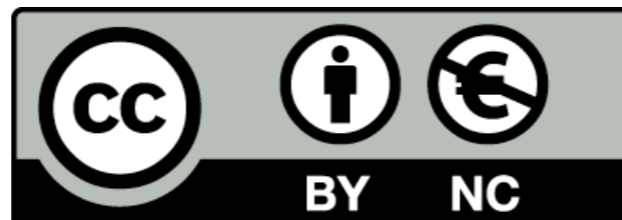
```
public class ClubTest extends UnitTest
{
    private Club c1, c2, c3;

    @Before
    public void setup()
    {
        c1 = new Club("tramore");
        c2 = new Club("dunmore");
        c3 = new Club("fenor");
        c1.save();
        c2.save();
        c3.save();
    }

    @After
    public void teardown()
    {
        c1.delete();
        c2.delete();
        c3.delete();
    }
}
```

```
@Test
public void testCreate()
{
    Club a = Club.findByName("tramore");
    assertNotNull(a);
    assertEquals("tramore", a.name);
    Club b = Club.findByName("dunmore");
    assertNotNull(b);
    assertEquals("dunmore", b.name);
    Club c = Club.findByName("fenor");
    assertNotNull(c);
    assertEquals("fenor", c.name);
}

@Test
public void testNotThere()
{
    Club a = Club.findByName("bunmahon");
    assertNull(a);
}
}
```



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/>

