Overview:

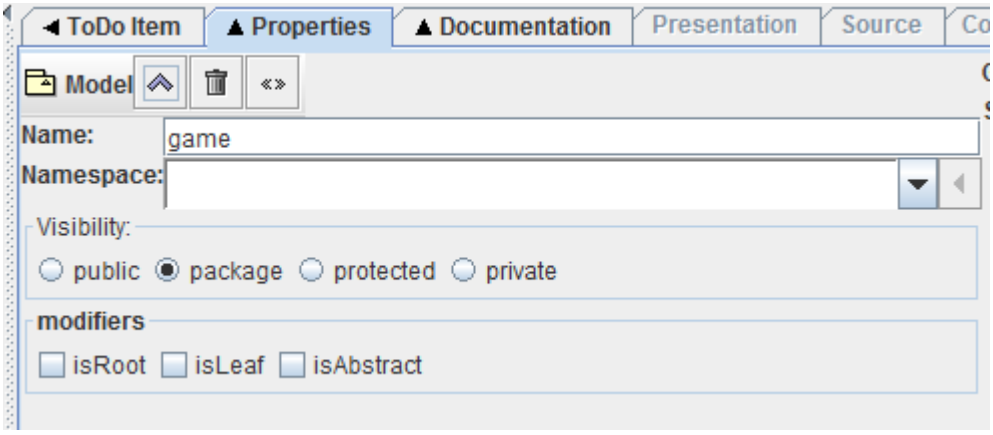
In this project we are going to model a basic game based on two teams and one arena/board. The arena/board determines the type, the size (big, medium, small), the level of the bots (novice, normal, expert and insane) and the quantity of players per team suitable for the arena. One game starts with one arena/board and one real player (the remaining is filled with bots), the game holds the timestamp of the game, the score and the state of the game (waiting, playing, finished). A real player only can join a game during waiting or playing state and replaces one of the bots. Each player has a name, a class, and a total punctuation of experience. When a player joins the game, he can either create a new game or join one. When creating a new game, the player creates the arena/board or chooses one of the available. Once the game is finished each player can choose if quit the game or continue playing.

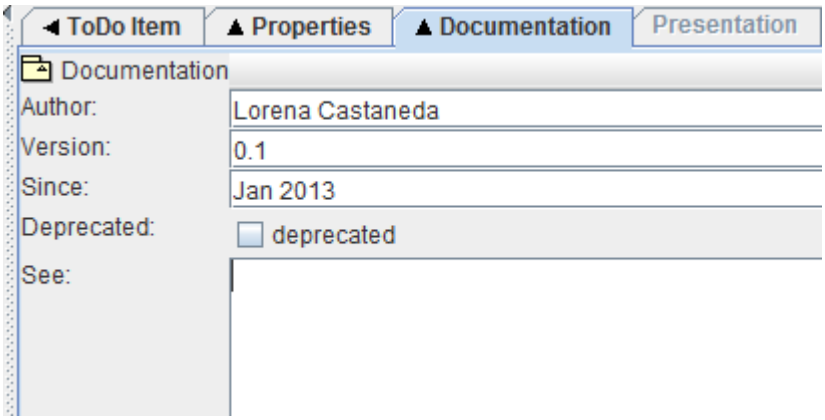
Lab Activities:

1. Use Case Diagram
2. Sequence Diagram
3. Class Diagram
4. UML to Code
5. Reverse Engineering: Code to UML

Activity 1: Use Case Diagram (using Argo UML)

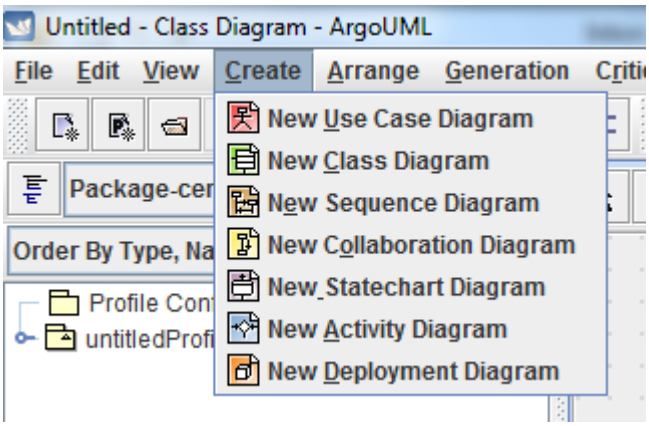
1. Configuration of your profile





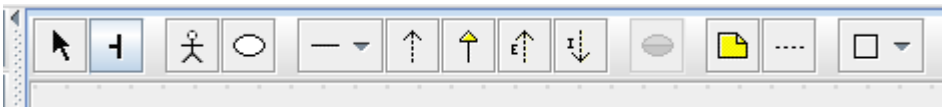
2. Select the type of diagram to use

   >Use Case Diagram

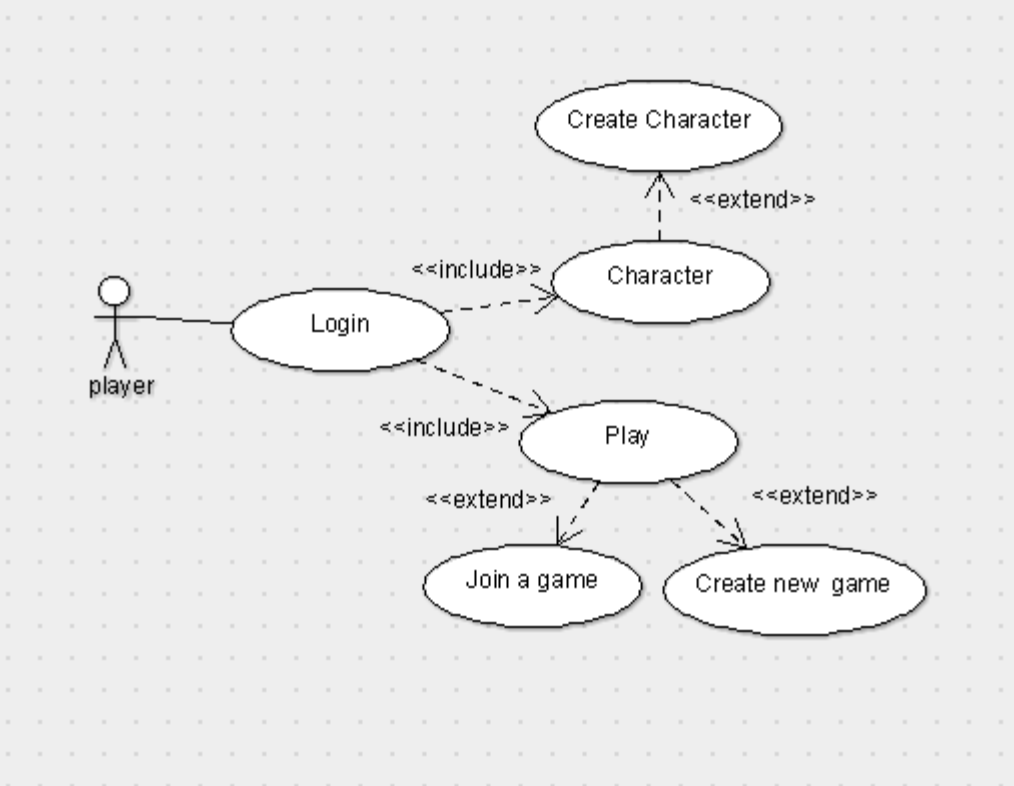


3. Complete the properties
   a. Name: ucd_game
   b. Home Model: Game profile

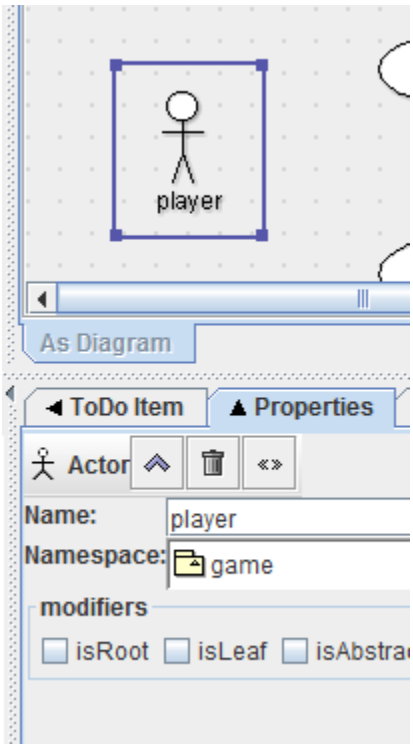
4. Review the toolbar with the elements of an *Use Case Diagram*



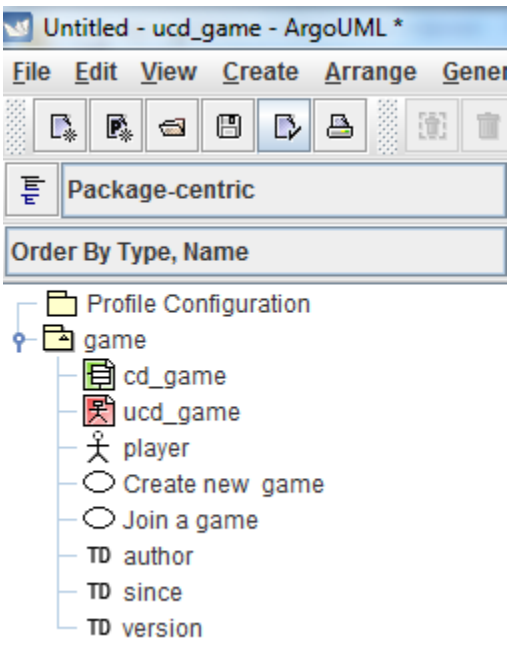
5. Create the Use Case corresponding with the *"start game"* requirement.



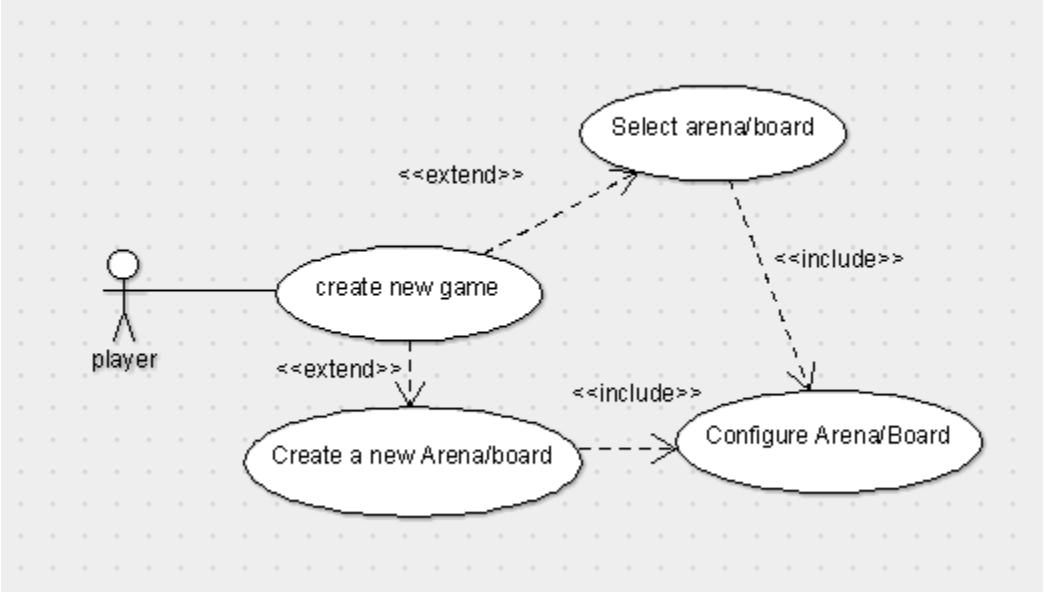
a. Explore the properties for every element,
   - Visit the documentation http://argouml.tigris.org/tours/index.html



b. Delete the arrows and check in the explorer panel. See the difference between deleting from the visual panel and the explorer. Be careful when you want to delete something.



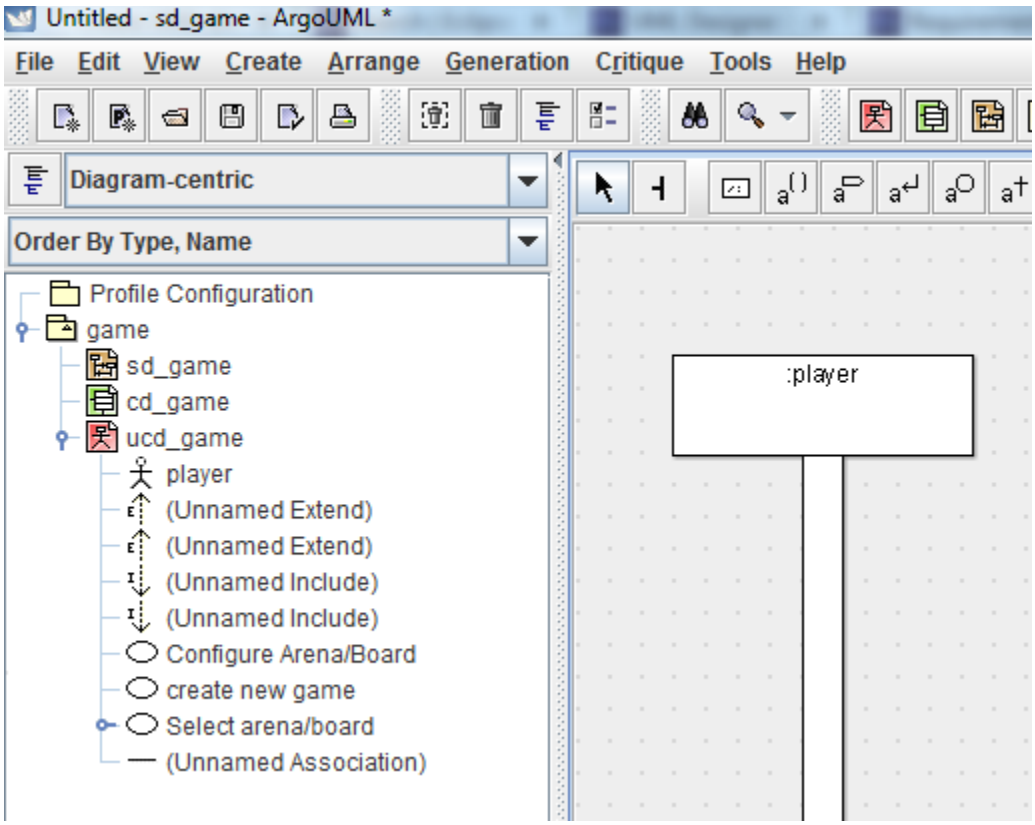
6. Create the Use Case corresponding with the *"create new game"* requirement.



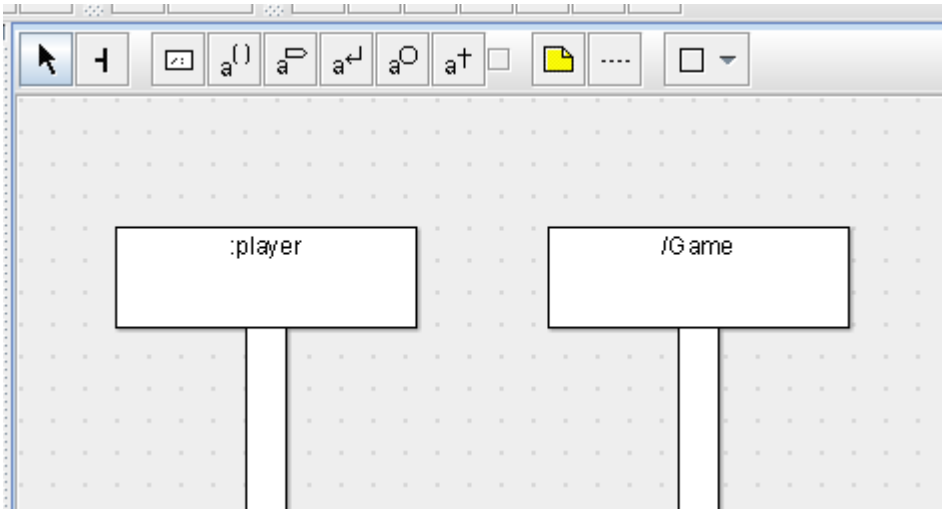
a. Reuse the elements in the panel, for example the **actor** in this case is the same.

Activity 2: Sequence Diagram (using Argo UML)

1. Use the menu to create a Sequence Diagram for the same project and configure the basic properties
2. Drag the actor from the use case to the sequence panel.

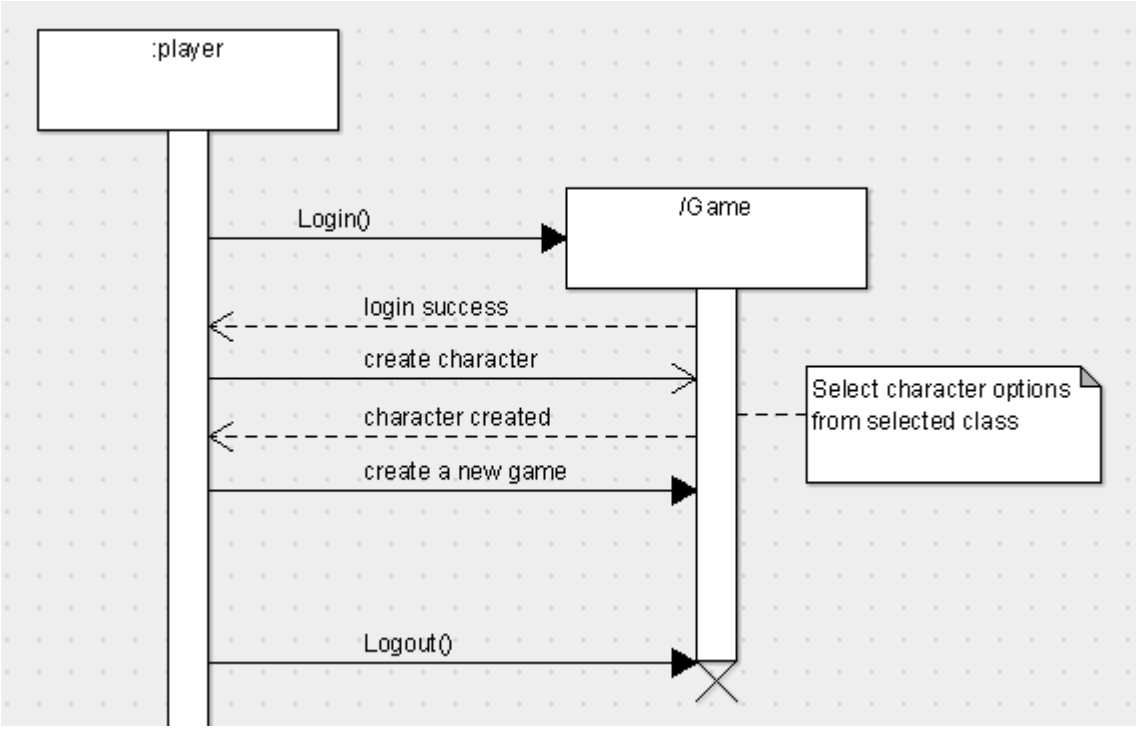


3. Create a new Role for the system named Game.



4. Model a simple sequence for a player that logins into the game, creates a character, create new game, and plays the game until the game finishes and then goes logs off. Don't forget to also

specify relevant activities in the Game role (e.g. create bots, filter class) use comments. (*The next figure is incomplete*)



a. Make a new model for a player who goes online to join a current game.

Activity 3: Class Diagram (PART 1 yUML)

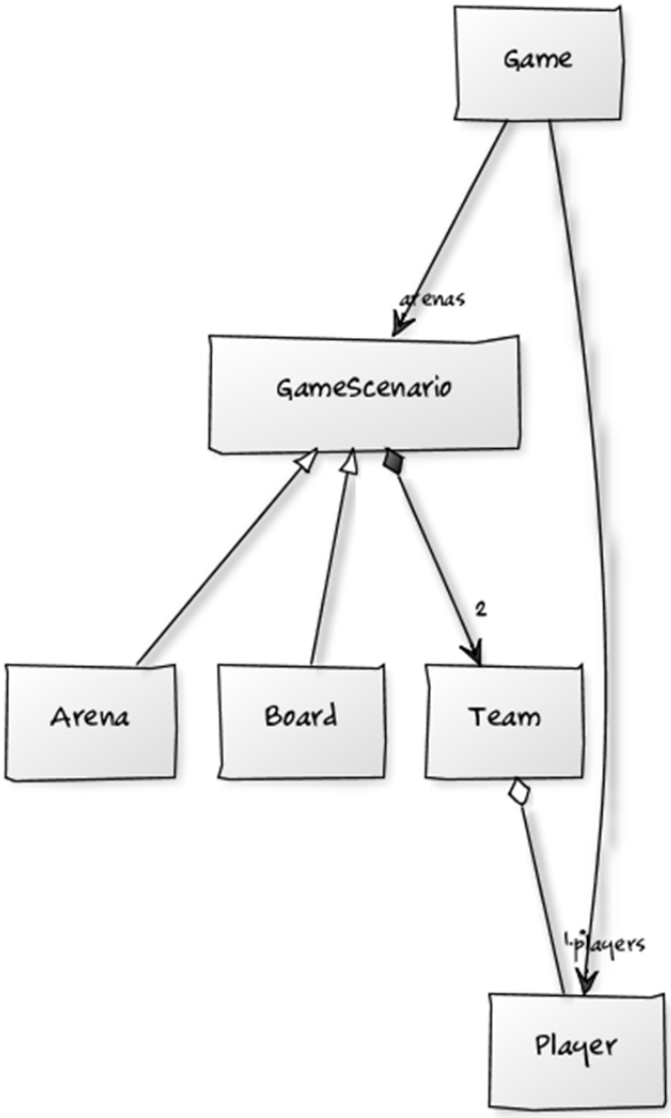
1. Open http://yuml.me/
   (Check out the **Tools** option in the menu for "3rd Party Tools For Using yUML With Your Apps")

2. Click on the "View Samples" button and review the syntax of yUML to create the UML class diagram.

3. Create your own class diagram for our online gaming example. This is a way to model it, but yours can be different

yUML code:

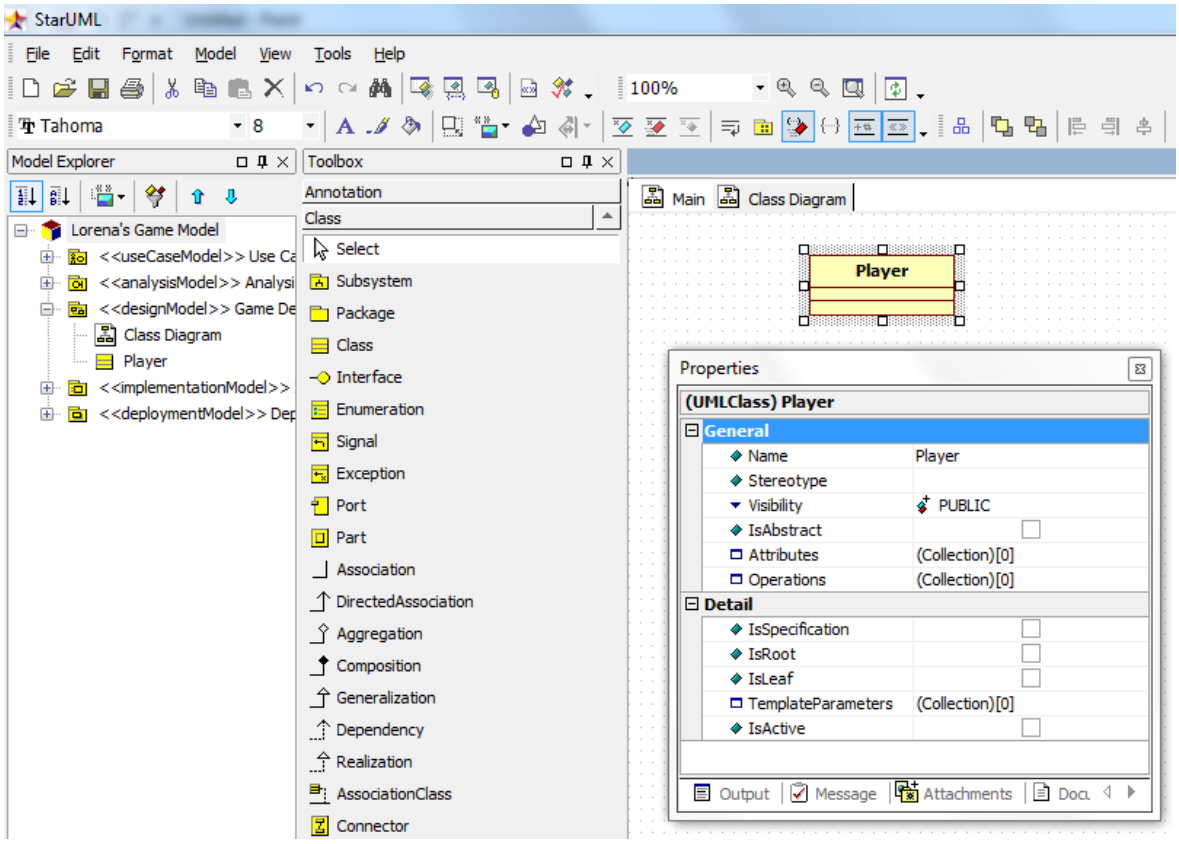
```
[Game]-players>[Player]
[Team]<>-1..*[Player]
[GameScenario]++-2>[Team]
[Game]-arenas>[GameScenario]
[GameScenario]^-[Board]
[GameScenario]^-[Arena]
```


Game
arenas
GameScenario
2
Arena
Board
Team
1..*players
Player

Activity 3: Class Diagram (PART 2 StarUML)

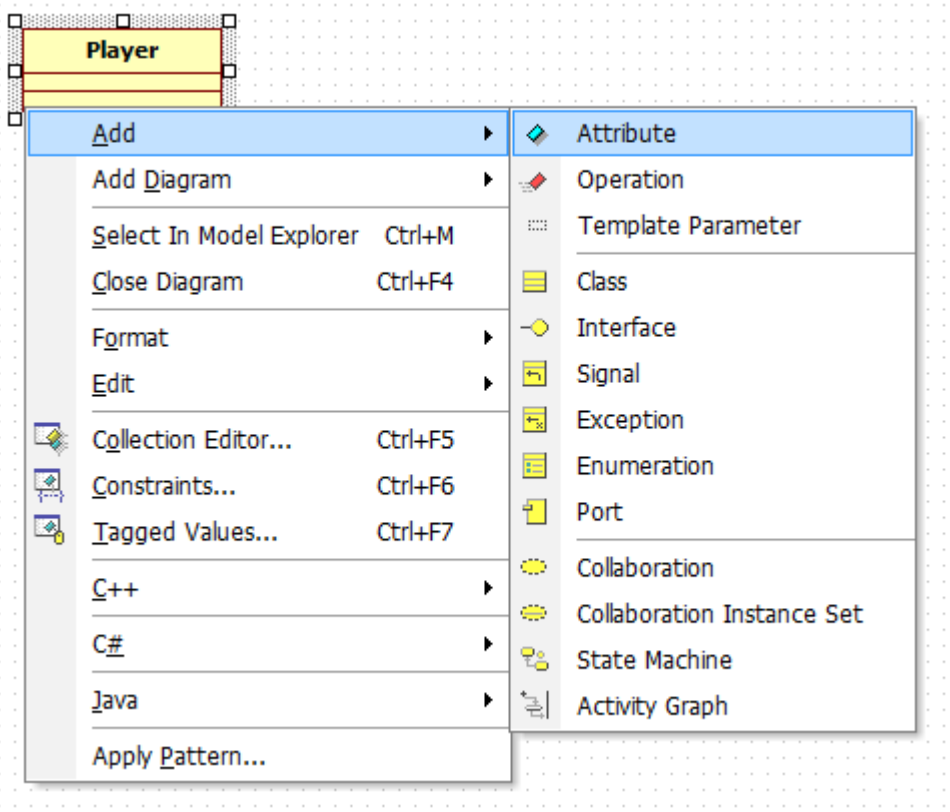
Open a new project and select the default approach. For this activity you can create your own version of the class diagram as in the previous activity. Follow the next steps with your own classes if you like.

1. Create a new class element by dragging it from the toolbox bar

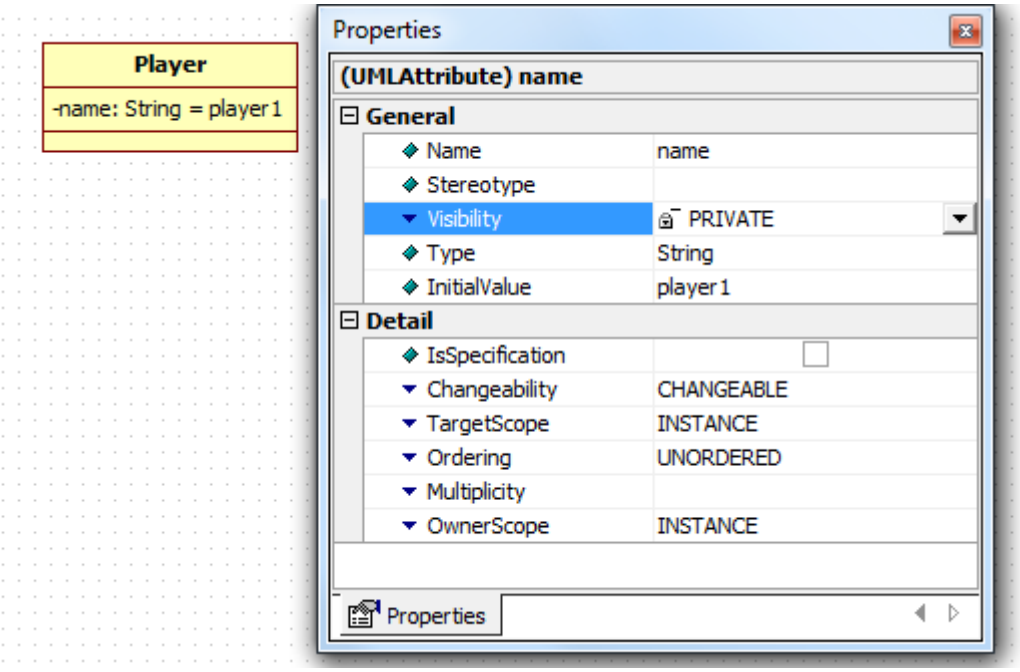


   a. The **Model Explorer** panel displays the elements in the diagram
   b. The **Toolbox** panel displays all the elements to create the Class Diagram UML. The tools will be updated according the type of diagram.
   c. The **Properties** panel displays all the configurations for a specific element. You can dock all these panels as you feel more comfortable.

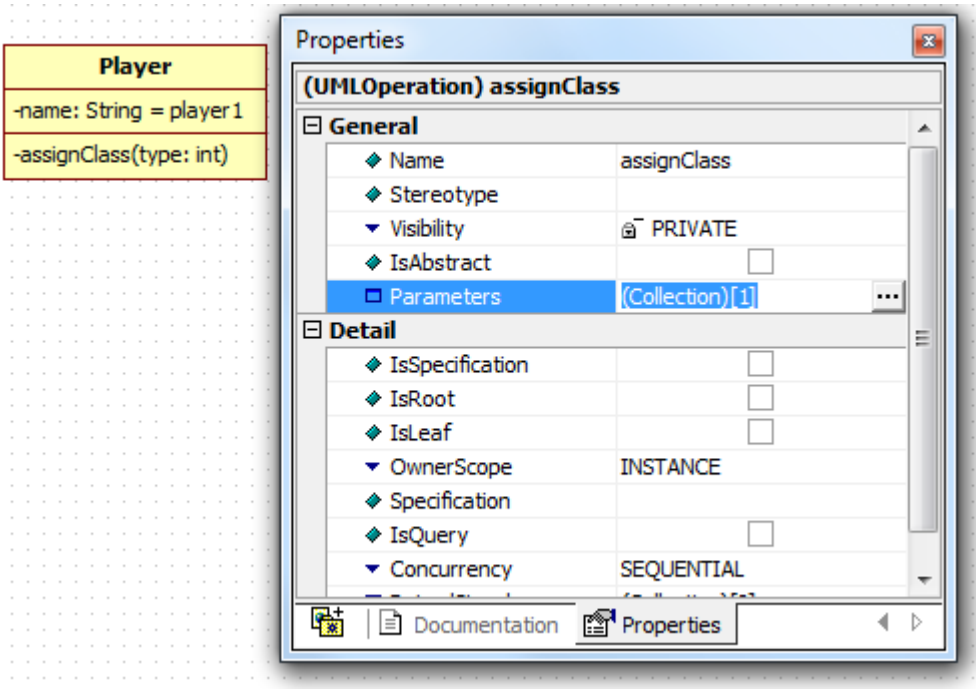
2. Add attributes and operations to a class. While stepping on the Class **use right click** and add the elements, and then configure in the Properties panel the values for each.



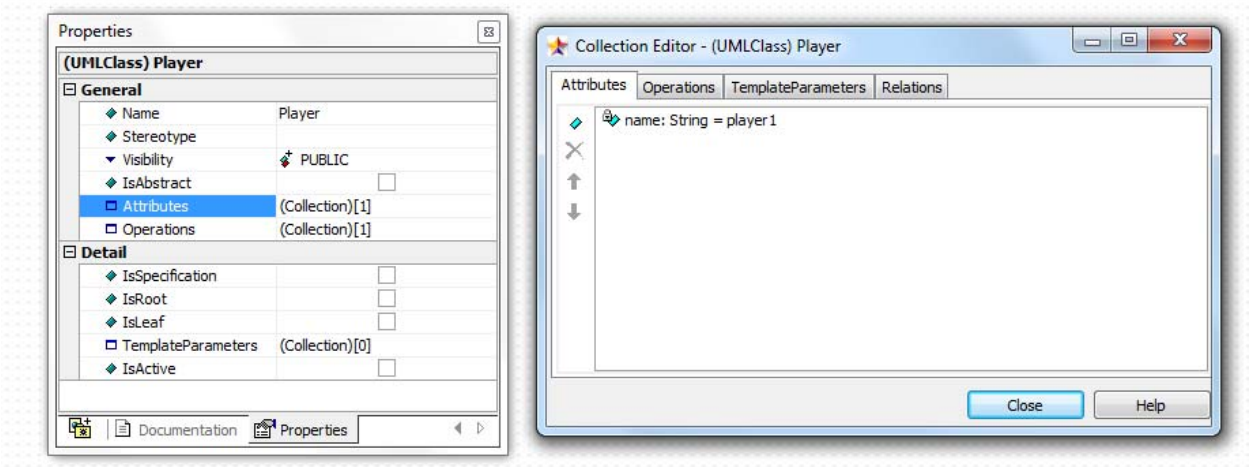
a. Attribute **name**



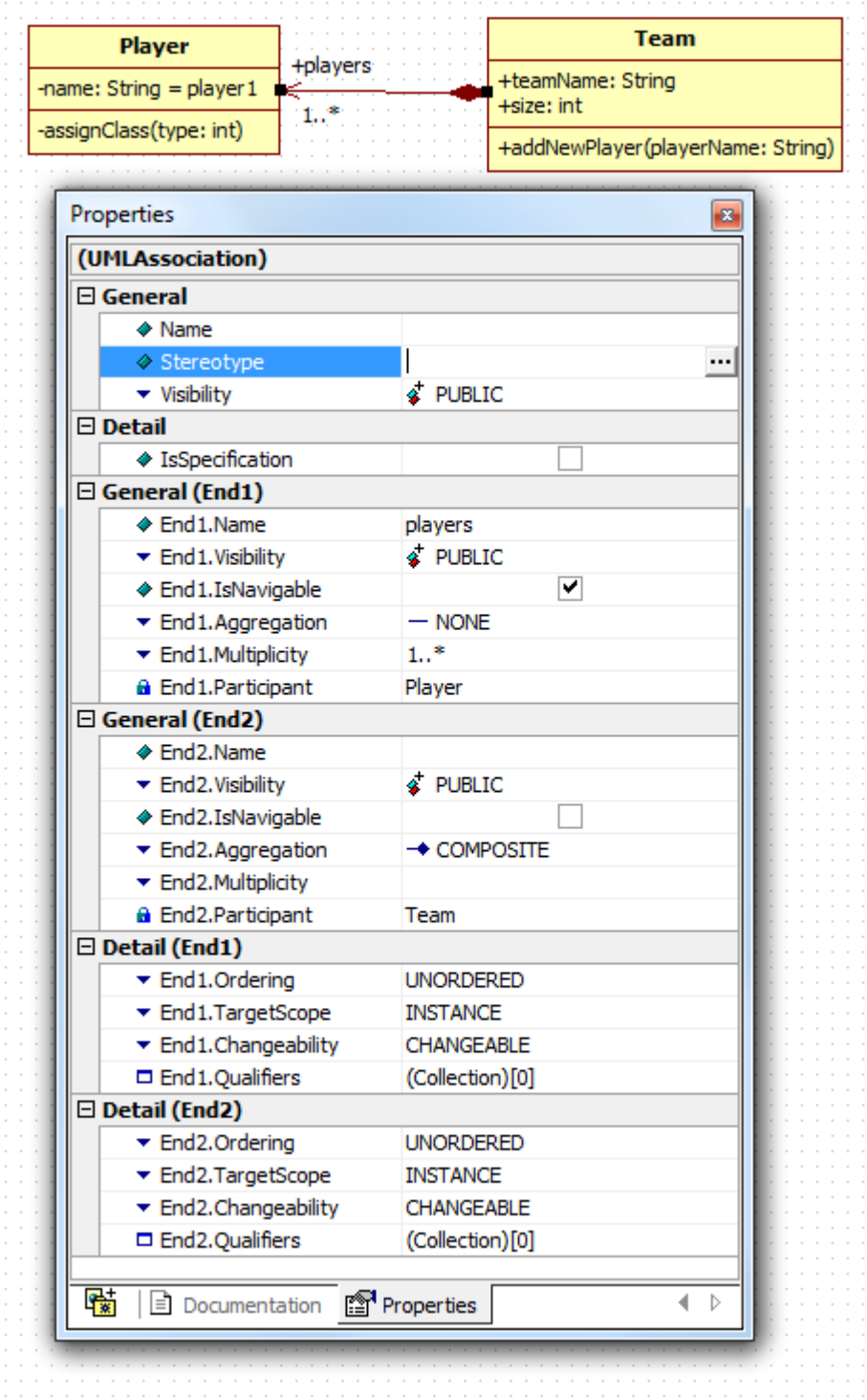
b. Operation **asignClass(type:int)**



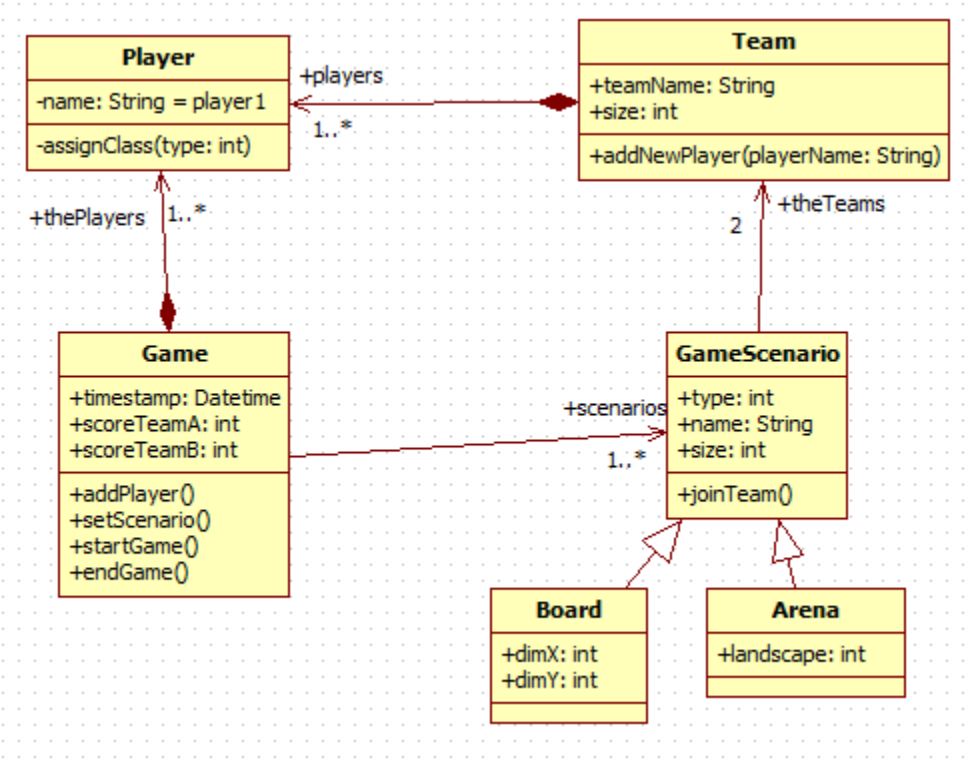
3. You can also add attributes and operations in the Properties panel for the class by double clicking the *Attributes* and *Operations* Collection link



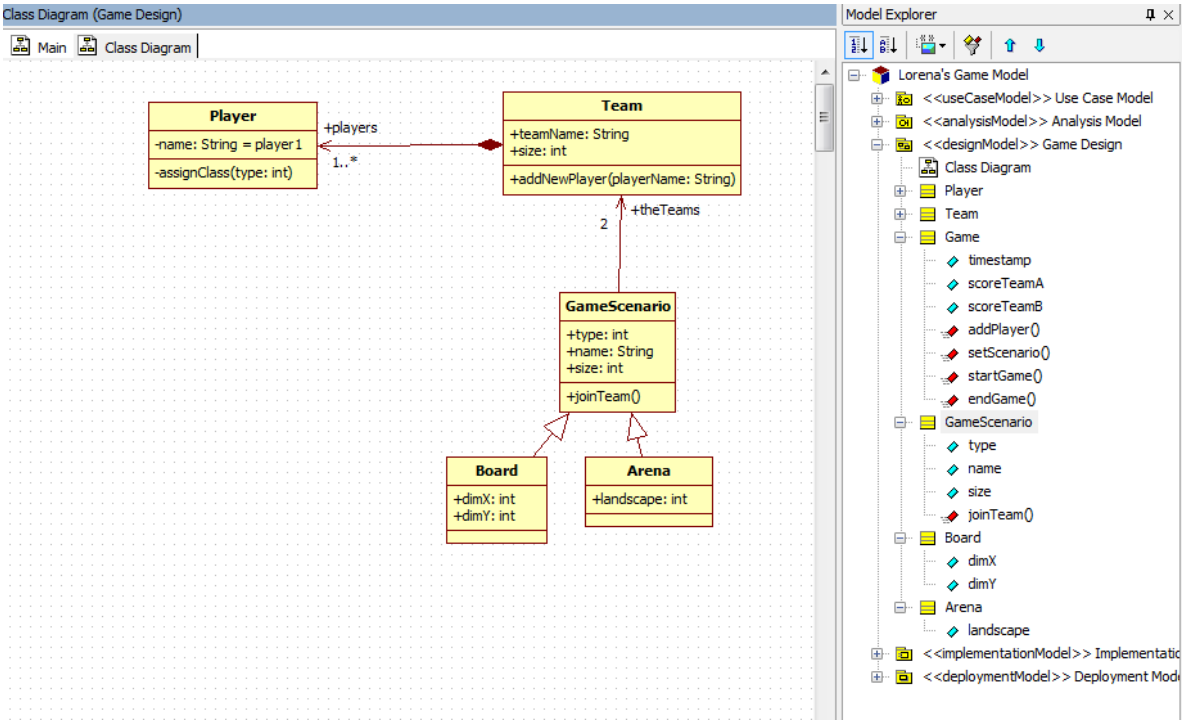
4. Create a second class and add a relationship for them.
   For this, you can just make a simple association and then configure all the specification in the *Properties* panel.



5. Complete you class diagram with all your elements.

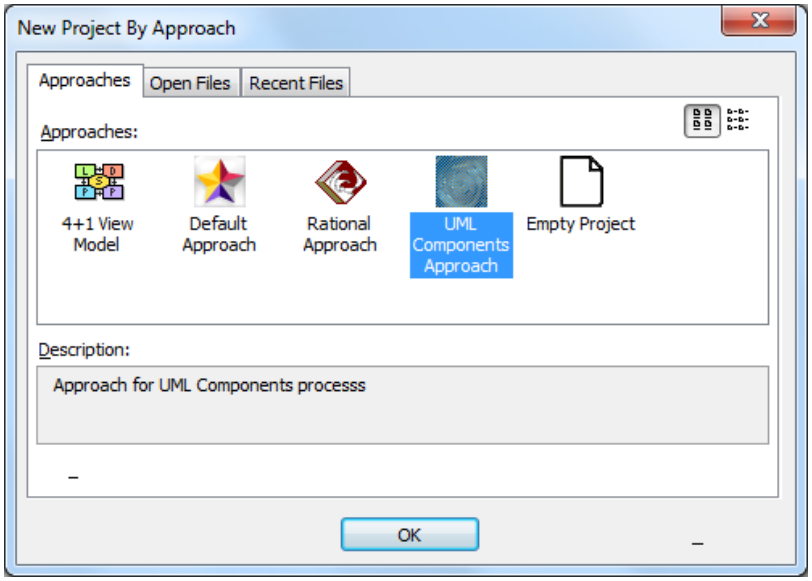


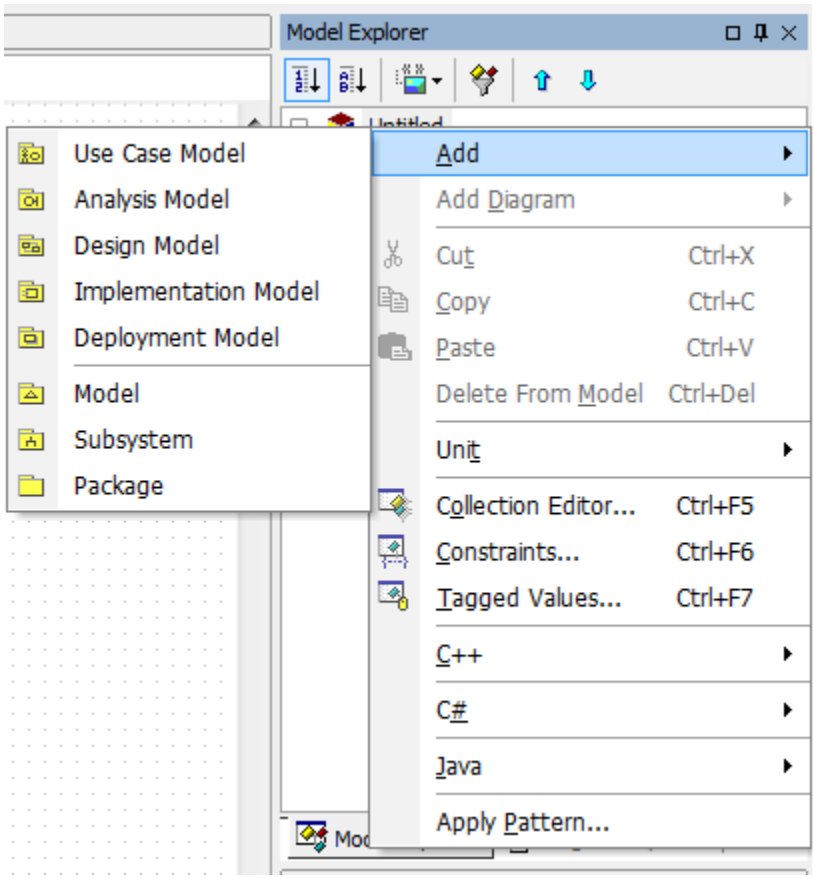
6. Delete an element from the visualization panel and compare with the explorer view. Now, bring back the element from the Explorer to the visualization panel. All the relations and attributes are in the model.



Extended Activity:

1. Create a new project approach and select de **UML Components**.

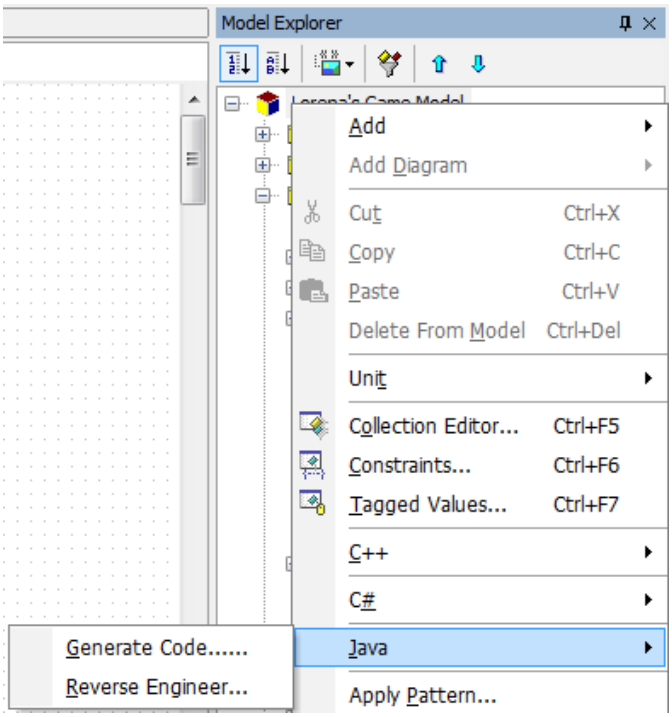




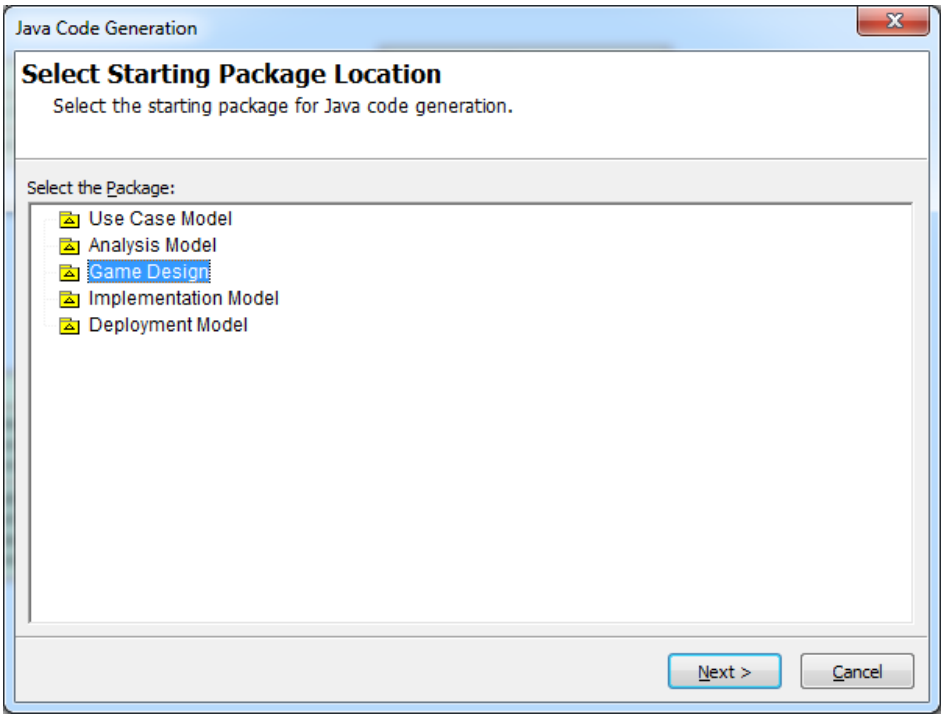
2. Explore those models in the software engineering process and documentation.

Activity 4: UML to Code (StarUML)

1. Go the **Model Explorer** view and right click on the project. Chose the language you want to generate the code based on the class diagram.

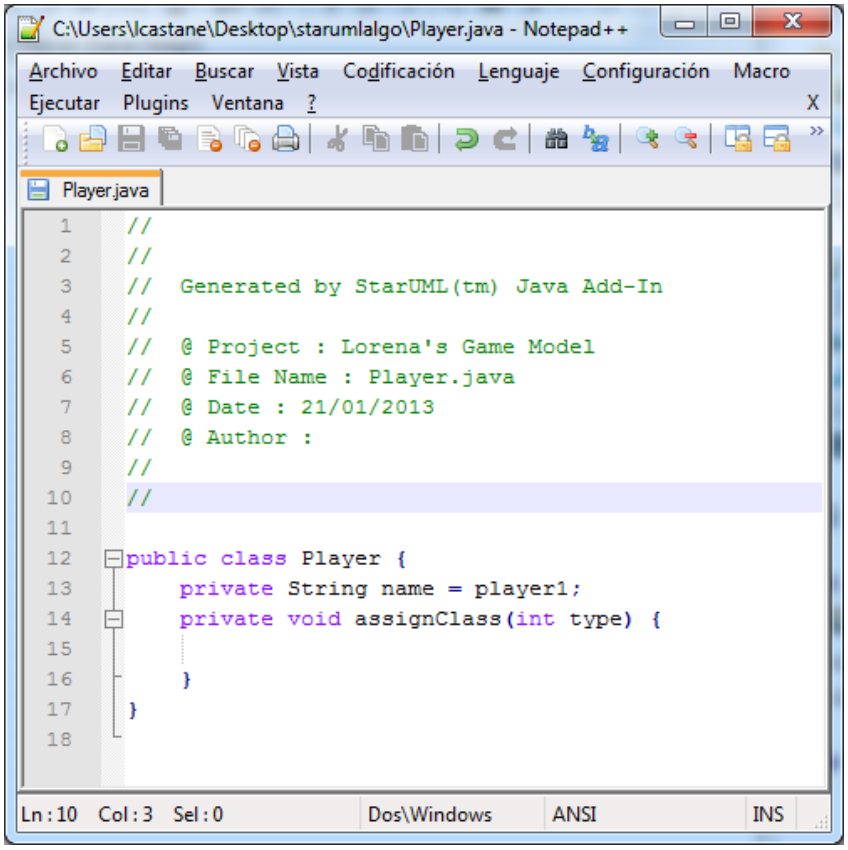


   a. Select the package, in my case is the **Game Design**



b. Complete the wizard according the language specifications.

2. Open the generated code in a IDE and evaluate the transformation.

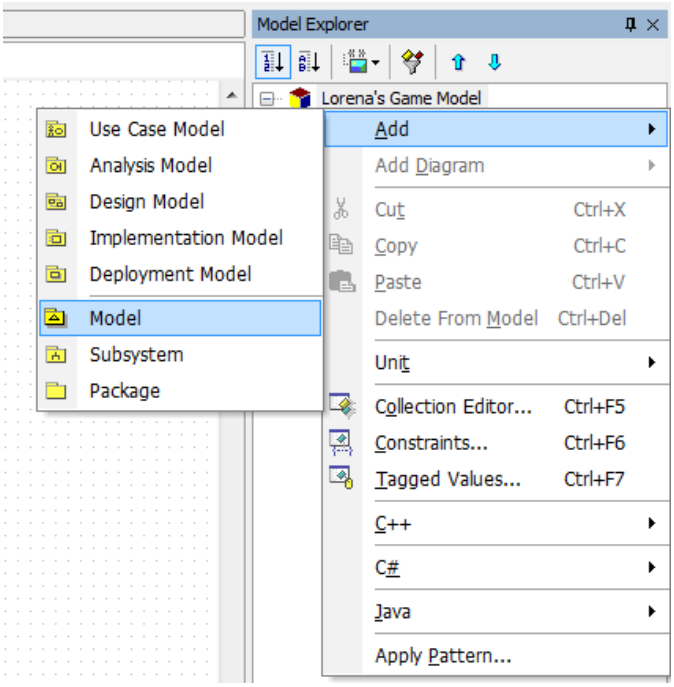


Extended Activity:

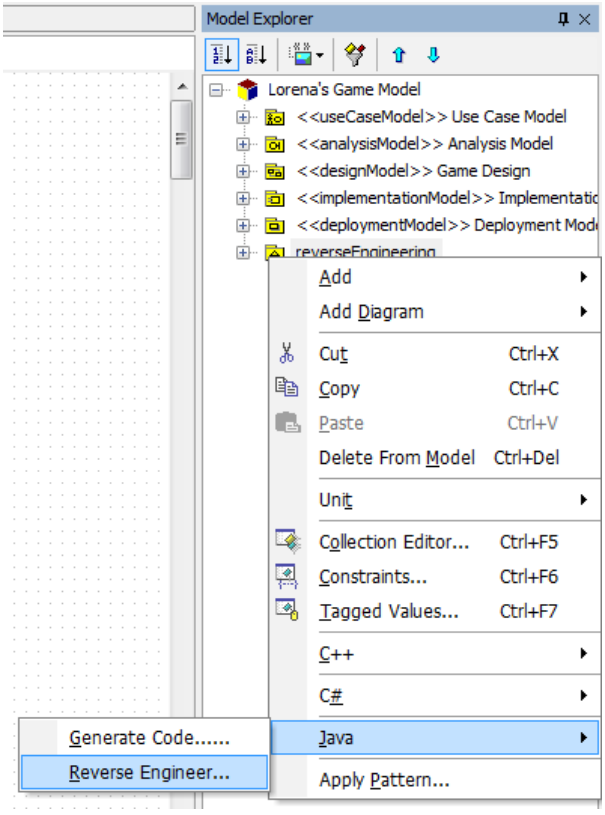
1. Explore the other modeling project types and compare it with ArgoUML.

Activity 5: Reverse Engineering: Code to UML (StarUML)

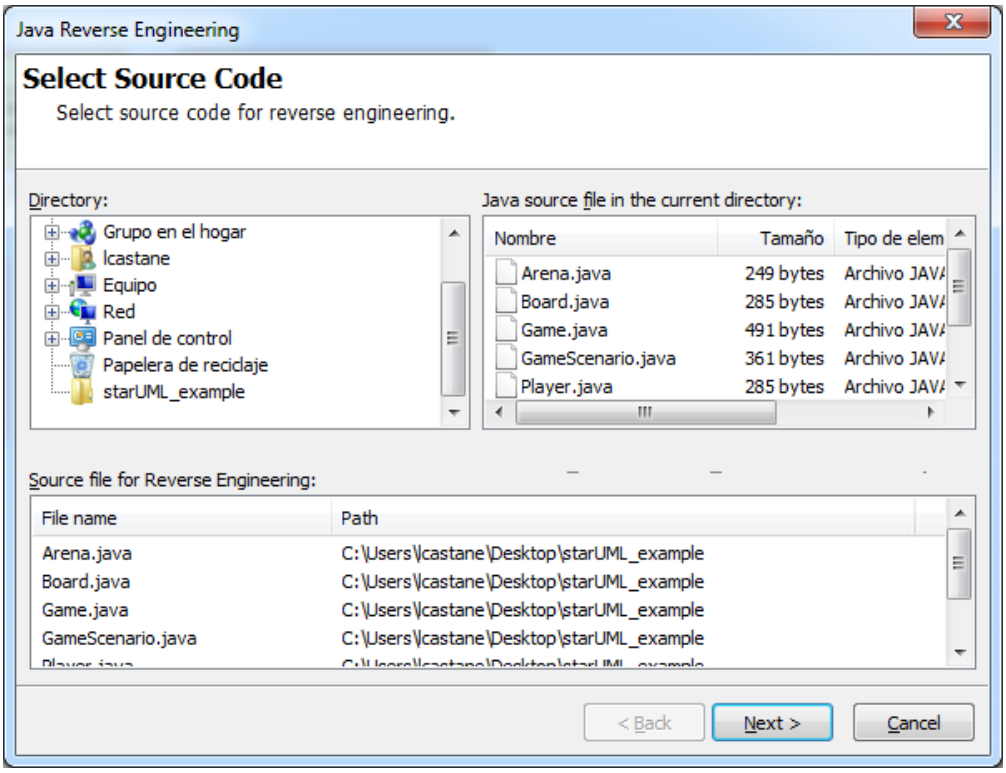
1. Create a new model from the Model Explorer view with the name “Reverse Engineering”



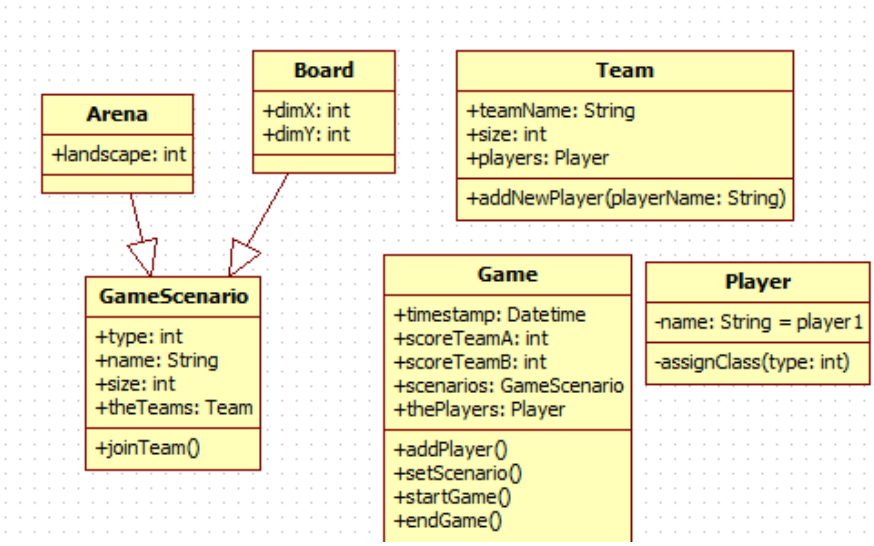
2. In the new model right click and choose the language you generated the previous model, we are going to reverse the code we just generated.



3. Reverse all the classes



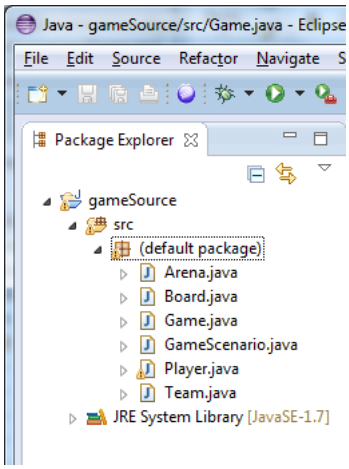
4. Compare the new diagram with the original, the one you created.



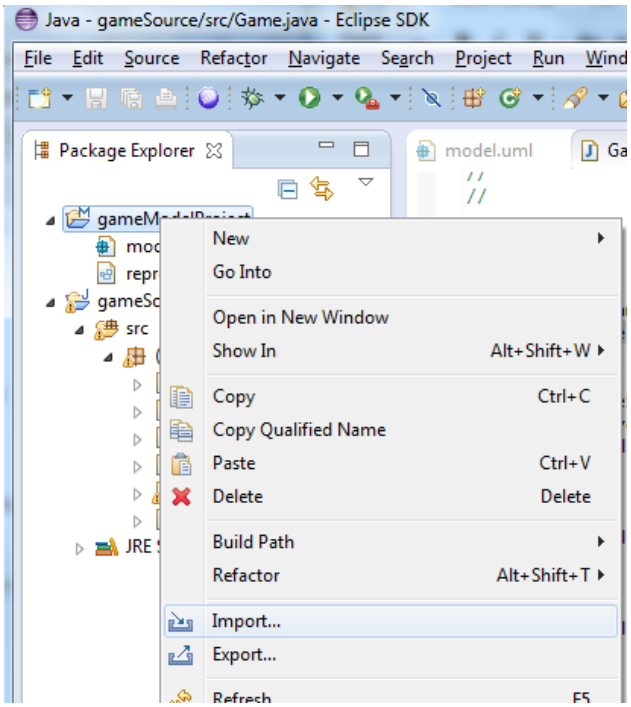
Extended Activity:

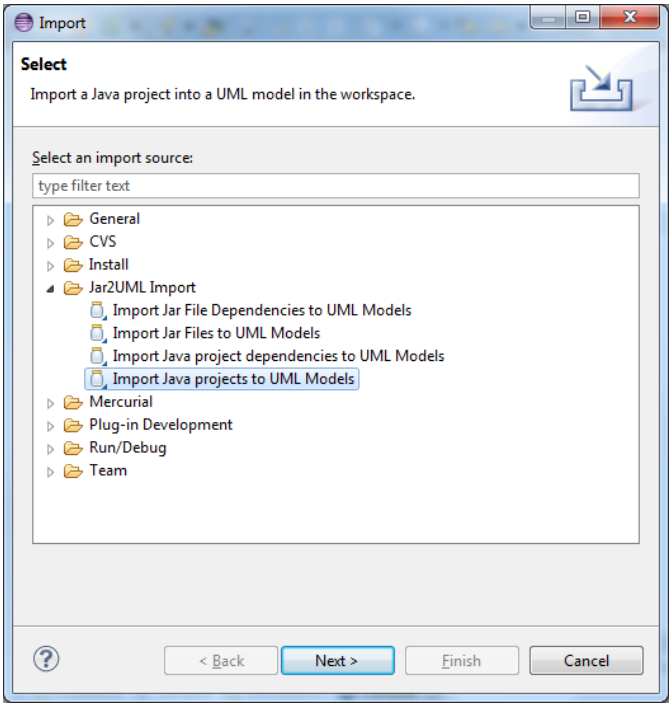
Install the eclipse plugin Jar2UML and generate the diagram from the same source code and compare them

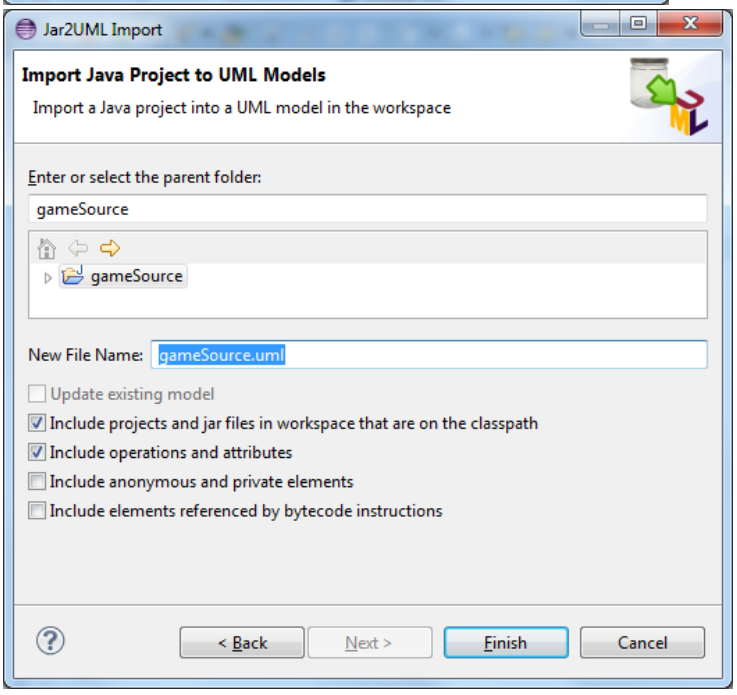
1. Import the Java source code in a Java project (this plugin only works for java, but there are others)



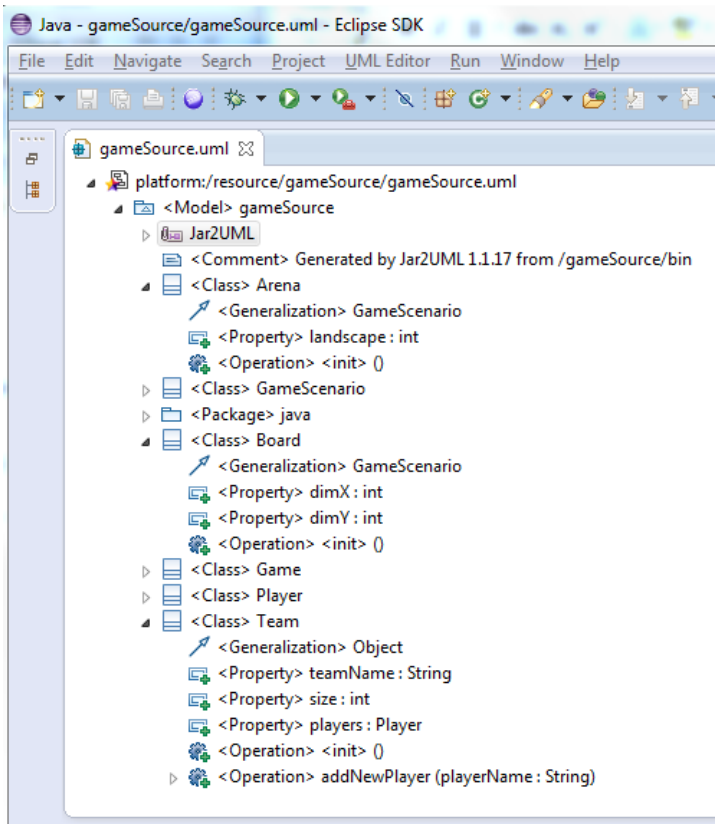
2. Right click on the project and import java projects to UML Model, create a name for the model and finish the import







3. Compare the model generated with the starUML original. What are the differences?


Java - gameSource/gameSource.uml - Eclipse SDK
File Edit Navigate Search Project UML Editor Run Window Help
gameSource.uml
platform:/resource/gameSource/gameSource.uml
<Model> gameSource
Jar2UML
<Comment> Generated by Jar2UML 1.1.17 from /gameSource/bin
<Class> Arena
<Generalization> GameScenario
<Property> landscape : int
<Operation> <init> ()
<Class> GameScenario
<Package> java
<Class> Board
<Generalization> GameScenario
<Property> dimX : int
<Property> dimY : int
<Operation> <init> ()
<Class> Game
<Class> Player
<Class> Team
<Generalization> Object
<Property> teamName : String
<Property> size : int
<Property> players : Player
<Operation> <init> ()
<Operation> addNewPlayer (playerName : String)