Niraj Bhatt – Architect's Blog

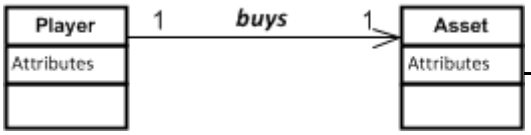
Ruminations on .NET, Architecture & Design

Association vs. Dependency vs. Aggregation vs. Composition

❑ nirajrules ☐ Architecture Design, Food For Thought, Visio ☐ July 15, 2011July 17, 2011 ☐ 2 Minutes

This might sound like a preliminary topic but recently I had a healthy discussion around them and thought of sharing few thoughts here. The description below is in context with Class Diagrams and this blog post uses UML to express the relationship in form a diagram.

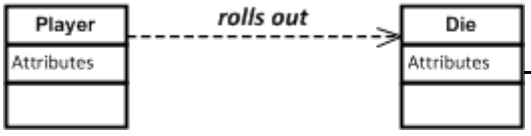
Association is reference based relationship between two classes. Here a class A holds a class level reference to class B. Association can be represented by a line between these classes with an arrow indicating the navigation direction. In case arrow is on the both sides, association has bidirectional navigation.



(https://nirajrules.files.wordpress.com/2011/07/association.png)

```
class Asset { ... }
class Player {
Asset asset;
public Player(Assest purchasedAsset) { ... } /*Set the asset via Constructor
or a setter*/
}
```

Dependency is often confused as Association. Dependency is normally created when you receive a reference to a class as part of a particular operation / method. Dependency indicates that you may invoke one of the APIs of the received class reference and any modification to that class may break your class as well. Dependency is represented by a dashed arrow starting from the dependent class to its dependency. Multiplicity normally doesn't make sense on a Dependency.

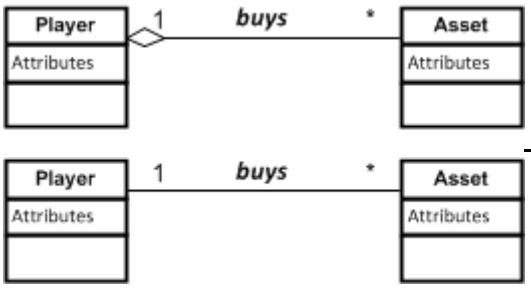


(https://nirajrules.files.wordpress.com/2011/07/dependency.png)

```
class Die { public void Roll() { ... } }
class Player
{
public void TakeTurn(Die die) /*Look ma, I am dependent on Die and it's Roll
```

```
method to do my work*/
{ die.Roll(); ... }
}
```

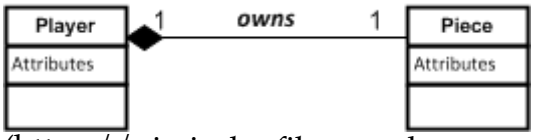
Aggregation is same as association and is often seen as redundant relationship. A common perception is that aggregation represents one-to-many / many-to-many / part-whole relationships (i.e. higher multiplicity), which of course can be represented by via association too (hence the redundancy). As aggregation doesn't convey anything more effective about a software design than an association, there is no separate UML representation for it (though some developers use a hollow diamond to indicate aggregation). You can give aggregation a miss unless you use it to convey something special.



(https://nirajrules.files.wordpress.com/2011/07/aggregation.png)

```
 class Asset { ... }
class Player {
List assets;
public void AddAsset(Asset newlyPurchasedAsset) {
assets.Add(newlyPurchasedAssest); ... }
...
}
```

Composition relates to instance creational responsibility. When class B is composed by class A, class A instance owns the creation or controls lifetime of instance of class B. Needless to say when class instance A instance is destructed (garbage collected), class B instance would meet the same fate. Composition is usually indicated by line connecting two classes with addition of a solid diamond at end of the class who owns the creational responsibility. It's also a perceived wrong notion that composition is implemented as nested classes. Composition binds lifetime of a specific instance for a given class, while class itself may be accessible by other parts of the system.



(https://nirajrules.files.wordpress.com/2011/07/composition.png)

```
 public class Piece { ... }
public class Player
{
Piece piece = new Piece(); /*Player owns the responsibility of creating the
Piece*/
...
}
```

Though elementary, I hope above distills your thinking 🙂