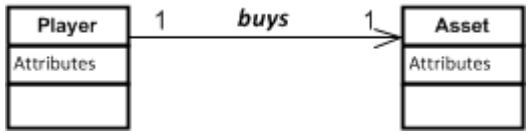
Niraj Bhatt – Architect's Blog

Ruminations on .NET, Architecture & Design

Association vs. Dependency vs. Aggregation vs. Composition

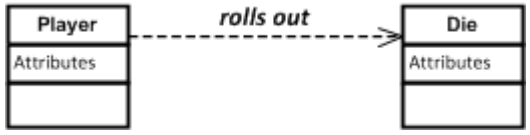
This might sound like a preliminary topic but recently I had a healthy discussion around them and thought of sharing few thoughts here. The description below is in context with Class Diagrams and this blog post uses UML to express the relationship in form a diagram.

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.




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

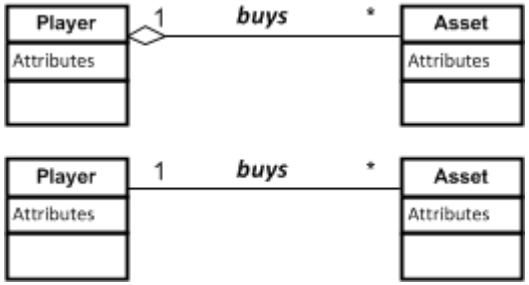
Dependency is often confused as Association. Dependency is normally created when you receive a reference to a class as part of a particular operation / method. Dependency indicates that you may invoke one of the APIs of the received class reference and any modification to that class may break your class 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.




```
class Die { public void Roll() { ... } }
class Player
{
```

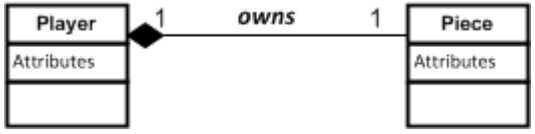
```
public void TakeTurn(Die die) /*Look ma, I am dependent on Die and it's Roll method to do my work*/
{ die.Roll(); ... }
}
```

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.




```
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.



```
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 🙂

Rate this:

54 Votes

Share this:

Email Print Facebook 36 LinkedIn 4 Reddit Twitter Google

★ Like

5 bloggers like this.

Related

Dependency Inversion, Dependency Injection, DI Containers and Service Locator
In ".NET"

NHibernate - Lessons Learned
In "NHibernate"

MVC vs. MVP vs. MVVM
In "Architecture Design"

Architecture Design, Food For Thought, Visio OOAD, UML

← Setting Auto Commit Off SQL Server

ASP.NET Session Timeout Not Working →

22 responses to "*Association vs. Dependency vs. Aggregation vs. Composition*"

johnysputnik July 15, 2011 at 7:05 pm

I always view association as something to be used in an early stage of analysis. Once a model has been created, the associations indicate potential dependencies. It then becomes an exercise in reducing dependencies to decrease coupling and increase orthogonality.

The next stage would then be to determine whether an object needs a member reference to the object it depends on or if it is just transient for a particular method or methods. An exercise in managing dependency injection I guess 🙂

If it is decided that the reference needs to be a member of the dependent object, then the dependency becomes an aggregation.

Then if the dependent object needs to own the object that it depends on and control its lifetime then the aggregation becomes a composition.

So I guess I see all the relationships as extension of each other. Composition extends Aggregation, Aggregation extends Dependency and Dependency extends Association.

Reply

nirajrules July 15, 2011 at 9:11 pm

@johnysputnik, thanks for your comment. Excellent thoughts. Do you use DI for all of them (composition, aggregation, dependency)? Just wondering since you mentioned DI in middle rather than in end.

Reply

johnysputnik July 16, 2011 at 1:38 pm

@nirajrules. If I was using Dependency Injection as my inversion of control strategy, I would probably inject the dependency in the case of Aggregation (injection in the constructor) and a basic Dependency (injection in the dependent method). In the case of composition, I would probably inject some kind of factory method/class in the constructor to allow the constructor to create an instance. An alternative would be to create an instance externally, inject it into the constructor and take ownership of it. I think this is a little messy, although it works well for the Qt framework, so who am I to argue 🙂

Of course DI is not the only IOC pattern. For example, if the initial associations highlighted some classes, with few or even 1 instance and many dependents, it may be more appropriate to use some kind of Service Locator pattern to manage dependencies.

Reply

nirajrules July 19, 2011 at 1:00 am

@johnysputnik I guess constructor injection would make you the sole owner of the instance (default configuration). Also, not sure if one would use DI / Service Locator for all the dependencies. At times we may have to resort to manual wire up with appropriate factories (protected variation).

Pingback: Link Resource # 2 July 21 « Dactylonomy of Web Resource

Sam March 2, 2012 at 3:42 pm

Most simple explanation on this topic I ever found on any site or in any thread,
gr8fully explained and conceptually elaborated.

Reply

gbushuev January 21, 2013 at 2:38 am
The best explanation about the topic I've ever seen on the internet. Thanks.

Reply

everson April 5, 2014 at 11:11 pm
i have to agree with messages before. The best explanation!

Reply

ashis mandal (@ashis_man) July 5, 2014 at 9:01 am
Thanks for your good representation .Very helpful.

Reply

Vishal November 11, 2014 at 2:37 pm
Thanks for such a nice explanation. Although I still have some queries.
Is dependency a special case of Association?
Can any type of relationship be an association?

Reply

siamak April 4, 2015 at 4:31 am
I am wondering how its possible to implement an association with higher multiplicities than 1 for example suppose a specific domain that there is this business rule on there.
| person| 23|car|
A person can have more than one car , the maximum is 3 cars.
A car can be owned by more than one person , the maximum is 2 person

Reply

siamak April 4, 2015 at 4:39 am
I am wondering how its possible to implement an association with higher multiplicities than 1 for example suppose a specific domain that there is this business rule on there.
| person| 2.....3|car|
A person can have more than one car , the maximum is 3 cars.
A car can be owned by more than one person , the maximum is 2 person

Reply

Nezar Fadle May 19, 2015 at 12:12 pm
What you think about this Diagram (Please correct if needed) :

https://drive.google.com/file/d/0B9P9VIVjqj-AWkFfLVNIdGs0Mlk/view?pli=1

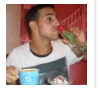
Explanation
#########
1. The Customer could have more than one Address and the Address can be belongs to multiple Customers.
2. The Customer could have multiple Orders and the Order belongs to one Customer.
3. The Order could have multiple Products and the Product belongs to multiple Orders.

Reply

Pavan Tiwari May 30, 2015 at 7:30 pm
Explanation of Association is not correct. it has been explained same like Aggregation, because in association instance are independent to each other. But in you example class Player having an global instance of Assets. It means Player class has a assets, which is Aggregation.

Reply

Matan June 1, 2015 at 7:08 pm
Thank you for your knowledge sharing, straight forward and simple, exactly what I was looking for 🙂



Reply

kante January 6, 2016 at 11:02 pm
Hello, thanks. I liked your clear explanation. It would be great if you also point out what's the difference between drawing an association "arrowed" and without any arrow . Draw.io (section uml) gives you chances of Association(unarrowed solid line), Relation (arrowed line with 1 cardinality), Association with double arrow. I read from somewhere that arrow means navigability. So if A points to B (A _____>B) A will have in the code implementation a reference to B, but B won't contain reference to A (A is not navigable from B). This is what I understood until now. I need a confirmation. Thanks for your posts.

Reply

alex June 2, 2016 at 7:27 pm
the best explanation!

Reply

paulo August 29, 2016 at 1:50 am
Excellent. Simple and right to the point.

Reply

SS December 14, 2016 at 10:52 am
Great explanation! Thanks.

Reply

sasanka ghosh January 2, 2017 at 5:52 pm
Why it is a wrong notion that Composition is not nested class. Composition of restrictive form of aggregation. With your example anyone can instantiate the class piece but that is not the purpose of composition. I think u r not correct in this case.

Reply

nirajrules January 2, 2017 at 8:26 pm
Sasanka, the point of emphasis is creational responsibility, which is the main difference when compared to association. Rest is contextual, including use of nested classes for implementation.

Reply

Rajesh Patel January 26, 2017 at 10:18 am
The best and most simplest explanation about the topic I've ever seem
. Thanks.

Reply

↑ Top