Aggregate Root Pattern in C#

Shivprasad koirala

13 Nov 2015 CPOL

Rate this: ★★★★★ 4.88 (12 votes)

Aggregate root pattern in C#

Aggregate root are cluster / group of objects that are treated as a single unit of data.

I am sure lots of developers are already using this pattern unknowingly, via this short note I would like to inform you formally what you are doing.

Let us try to understand the above definition with an example. Consider the below "`Customer`" class which has the capability to add multiple "`Address`" objects to it. In order to achieve the same, we have exposed an `address` collection from the `customer` class to represent the 1 to many relationships.

Hide Copy Code

```
class Customer
{
        public string CustomerName { get; set; }
        public DateTime DateofBirth { get; set; }

        public List<Address> Addresses { get; set; }

}

class Address
{
        public string Address1 { get; set; }
        public string Type { get; set; }
}
```

The above class structure works perfectly well. You can create object of `customer` and add multiple `addresses` object to it.

Hide Copy Code

```
Customer cust = new Customer();
cust.CustomerName = "Shiv koirala";
cust.DateofBirth = Convert.ToDateTime("12/03/1977");

Address Address1 = new Address();
Address1.Address1 = "India";
Address1.Type = "Home";
cust.Addresses.Add(Address1);

Address Address2 = new Address();
Address2.Address1 = "Nepal";
Address2.Type = "Home";
cust.Addresses.Add(Address2);
```

Now let's say we want to implement the following validations:

“Customer can only have one address of Home type”.

At this moment, the address collection is a NAKED LIST COLLECTION which is exposed directly to the client. In other words, there are no validations and restrictions on the “`Add`” method of the list. So you can add whatever and how much ever `address` objects as you wish.

Hide Copy Code

```
cust.Addresses.Add(Address2);
```

So how to address this problem in a clean and logical way. If you think logically, “`Customer`” is composed of `Addressescollection`, so `Customer` is like a main root. So rather than allowing DIRECT NAKED ACCESS to `Addresses` list, how about accessing the address list from the `customer` class. In other words, centralizing access to `address` objects from the `customer` class.

So below are three steps which I have implemented to put a centralize address validation.

Step 1: I have made the address list private. So no direct access to the collection is possible.

Step 2: Created a “Add” method in the “Customer” class for adding the “Address” object. In this add method, we have put the validation that only one “Home” type address can be added.

Step 3: Clients who want to enumerate through the address collection for them we have exposed “IEnumerable” interface.

Hide Shrink Copy Code

```
class Customer
    {
    // Code removed for clarity
        private List<Address> _Addresses; // Step 1 :- Make list private


        public void Add(Address obj) // Step 2 :- Address objects added via customer
        {
            int Count=0;
            foreach (Address t in _Addresses)
            {
                if (t.Type == "Home")
                {
                    Count++;
                    if (Count > 1)
                    {
                        throw new Exception("Only one home address is allowed");
                    }
                }
            }
            _Addresses.Add(obj);
        }

        public IEnumerable<Address> Addresses // Step 3 :- To browse use enumerator
        {
            get { return _Addresses; }
        }

    }
```

If you analyze the above solution closely, the customer is now the root and the address object is manipulated and retrieved via the customer class.

Why did we do this? Because “Customer” and “Address” object is one logical data unit. To maintain integrity of address validation, we need to go via the “Customer” class. In the same way loading of data, updation, deletion, etc. should all happen via the “Customer” class so that we have proper data integrity maintained.

So when we say load customer from database, all the respective address objects should also get loaded.

So when a group of objects which form one logical unit should have centralized root via which the manipulation of the contained object should happen. This kind of arrangement is terms as “Aggregate Root”.

The best way to Learn Design pattern is by doing a project. So if are interested, you can start from the below youtube video which demonstrates Design pattern by doing a project.

License

Share

About the Author

Shivprasad koirala

Architect https://www.questpond.com
India

Do not forget to watch my Learn step by step video series.

Learn MVC 5 step by step in 16 hours
Learn MVC Core step by step
Learn Angular tutorials step by step for beginners
Learn Azure Step by Step
Learn Data Science Step by Step

Step by Step Mathematics for Data Science
Learn MSBI in 32 hours
Learn Power BI Mobile Step by Step
Learn SQL Server in 16 hours ser...

show more

Comments and Discussions

You must Sign In to use this message board.

Search Comments

Spacing Relaxed ▼ Layout Normal ▼ Per page 25 ▼ Update

First Prev Next

I think this way is much better	**Charles Schneider Pereira**	**16-Nov-15 23:13**
If you want to avoid people accessing your private list, one way is that you can copy it out every time when asked	**Song Tiansheng**	**16-Nov-15 14:39**
Re: If you want to avoid people accessing your private list, one way is that you can copy it out every time when asked	**Shivprasad koirala**	16-Nov-15 14:47
Re: If you want to avoid people accessing your private list, one way is that you can copy it out every time when asked	Song Tiansheng	16-Nov-15 15:11
[My vote of 2] Some changes...	**Jono Stewart**	**15-Nov-15 21:24**
Good material, but the example could use	**AFell2**	**13-Nov-15 12:31**
i may be wrong	**lucwuyts**	**13-Nov-15 10:55**

Last Visit: 27-Aug-19 20:26 Last Update: 27-Aug-19 20:26 Refresh 1

General News Suggestion Question Bug Answer Joke Praise Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.