Aggregation

In this lesson, you'll get familiar with a new way of linking different classes.

WE'LL COVER THE FOLLOWING

Aggregation is a specialized form of *association*. It follows the **Has-A** model. In aggregation, a class **uses** the objects of other classes. Here, we will refer to the class, **using** the objects of other classes, as the *container* class, and the classes whose objects are being used as the *contained* classes.

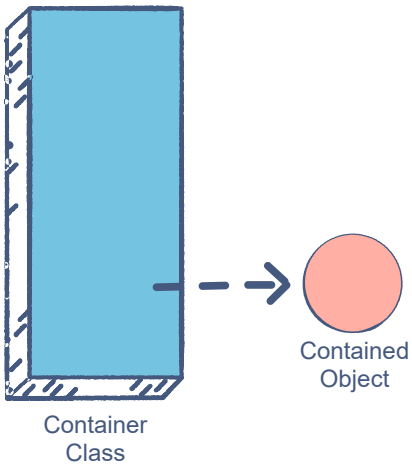
The *container* class contains the references to the objects of the *contained* classes.

Let’s see what makes aggregation unique.

Independent Lifetimes

> In aggregation, the lifetime of the contained object does not depend on the lifetime of the container object.

The *container* object could get deleted but the *contained* object can continue to exist in the program. The container only contains a **reference** to the contained, which removes the latter’s dependence on the former.



The container simply references to the contained object

Aggregation

You can probably guess from the illustration above that we'll need object references to implement aggregation.

Example

Let's take the example of the vending machine and the products inside it. A vending machine contains products but these products will still exist even if the vending machine ceases to exist. Let's assume, just for ease of our understanding, we have a vending machine with a capacity of 5 products and it's our duty to refill this machine daily. We are going to place the products in the machine with the help of an `InsertProduct()` method. Let's see this in our code:

```
class VendingMachine
{

    private Product[] _productShelf;
    private int _capacity;
    private static int _productCount;


    public VendingMachine()
    {
        this._productShelf = new Product[5];
        this._capacity = 5;
        _productCount = 0;
    }
    public bool InsertProduct(Product item)
    {
        if (_productCount < _capacity)
        {
```

```
            {
                this._productShelf[_productCount] = item;
                _productCount++;
                return true;
            }
            else
            {
                return false;
            }
        }

        public void PrintProducts()
        {
            for (int i = 0; i < this._capacity; i++)
            {
                _productShelf[i].PrintDetails();
            }
        }

    }

    public class Product
    {

        private string _name;
        private double _price;

        public Product(string name, double price)
        {
            this._name = name;
            this._price = price;
        }

        public void PrintDetails()
        {
            Console.Write("[{0}  {1}]\t\t", this._name, this._price);
        }

    }

    class Demo
    {

        public static void Main(string[] args)
        {
            // Creating Vending Machine
            VendingMachine myVendy = new VendingMachine();
            // Creating Products
            Product snack = new Product("Rango Tango", 0.5);
            Product drink = new Product("Mineral Water", 0.7);
            Product chocolate = new Product("Crunchy", 1);
            Product energyDrink = new Product("Red Bull", 3);
            Product popcorn = new Product("Caramel pop", 0.5);
            // Inserting Products into machine
            myVendy.InsertProduct(snack);
            myVendy.InsertProduct(drink);
            myVendy.InsertProduct(chocolate);
            myVendy.InsertProduct(energyDrink);
            myVendy.InsertProduct(popcorn);
            // Printing the product in vending machine
            myVendy.PrintProducts();
            // Now if we null the myVend
            myVendy = null;
```

```
        // The products still exist and are still usable
        Console.WriteLine("\nThe Products still exist!");
        snack.PrintDetails();
        drink.PrintDetails();
        chocolate.PrintDetails();
        energyDrink.PrintDetails();
        popcorn.PrintDetails();
    }

}
```

Products can exist independently even if the vending machine ceases to exist

Aggregation

As we can see on **lines 83-87**, the `Product` objects live on even after the `VendingMachine` is set to `null` on **line 80**. This creates a looser relationship between the two.

In the next lesson, we will explore the third type of linkage between classes, i.e. composition.