

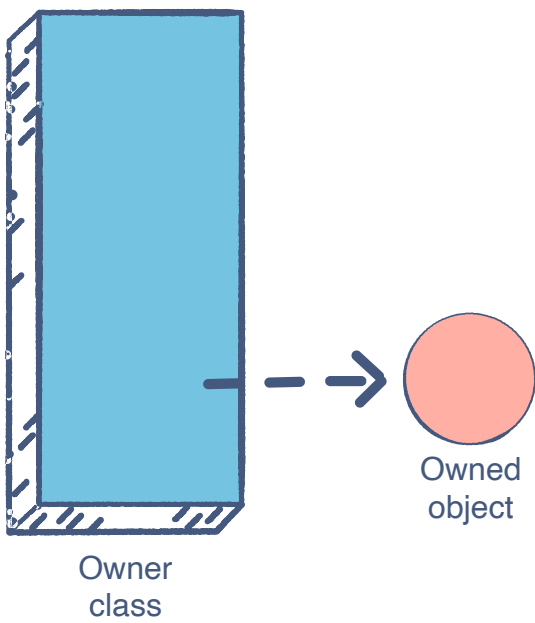
Aggregation follows the **has-A** model. This creates a parent-child relationship between two classes, with one class owning the object of another.

So, what makes aggregation unique?

Independent Lifetimes#

In **aggregation**, the lifetime of the owned object does not depend on the lifetime of the owner.

The owner object could get deleted, but the owned object can continue to exist in the program. In aggregation, the parent only contains a **reference** to the child, which removes the child’s dependency.



The owner (parent) simply points to the owned object (child)

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 people and their country of origin. Each person is associated with a country, but the country can exist without that person:

```
1 class Country {
2
3     private String name;
4     private int population;
5
6     public Country(String n, int p) {
7         name = n;
8         population = p;
9     }
10    public String getName() {
11        return name;
12    }
13
14 }
15
16 class Person {
17
18     private String name;
19     private Country country; // An instance of Country class
20
21     public Person(String n, Country c) {
22         name = n;
23         country = c;
24     }
25
26     public void printDetails() {
27         System.out.println("Name: " + name);
28         System.out.println("Country: " + country.getName());
29     }
30 }
```

Run Save Reset

As we can see, the **country** object lives on even after the **user** goes out of scope. This creates a loosely coupled relationship between the two classes.