Aggregation

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

We'll cover the following

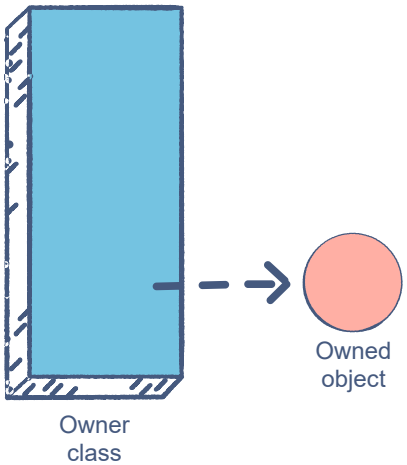
Aggregation follows the **Has-A** model. This creates a parent-child relationship between two classes, with one class owning the object of another.

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)

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:

```
class Country:
    def __init__(self, name=None, population=0):
        self.name = name
        self.population = population

    def printDetails(self):
        print("Country Name:", self.name)
        print("Country Population", self.population)


class Person:
    def __init__(self, name, country):
        self.name = name
        self.country = country

    def printDetails(self):
        print("Person Name:", self.name)
        self.country.printDetails()


c = Country("Wales", 1500)
p = Person("Joe", c)
p.printDetails()

# deletes the object p
del p
print("")
c.printDetails()

```

As we can see, the `Country` object, `c`, lives on even after we delete the `Person` object, `p`. This creates a weaker relationship between the two classes.

In the next lesson, you will learn about another technique for relating objects in Python: composition.