Aggregation

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

WE'LL COVER THE FOLLOWING

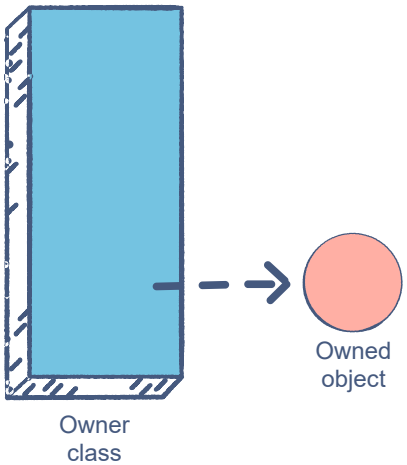
Aggregation follows the **has-A** model. This creates a parent-child relationship between two classes, with one class owning the object of another.

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)

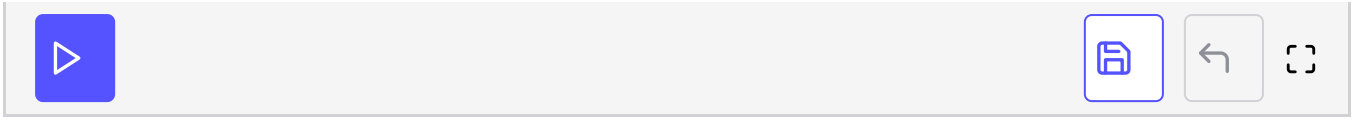
Aggregation

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:

```
class Country {

    private String name;
    private int population;

    public Country(String n, int p) {
      name = n;
      population = p;
    }
    public String getName() {
      return name;
    }

}

class Person {

    private String name;
    private Country country; // An instance of Country class

    public Person(String n, Country c) {
      name = n;
      country = c;
    }

    public void printDetails() {
      System.out.println("Name: " + name);
      System.out.println("Country: " + country.getName());
    }

}

class Main {

  public static void main(String args[]) {
    Country country = new Country("Utopia", 1);
    {
      Person user = new Person("Darth Vader", country);
      user.printDetails();
    }
    // The user object's lifetime is over

    System.out.println(country.getName()); // The country object still exists!
  }

}
```



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.

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