A Brief Introduction

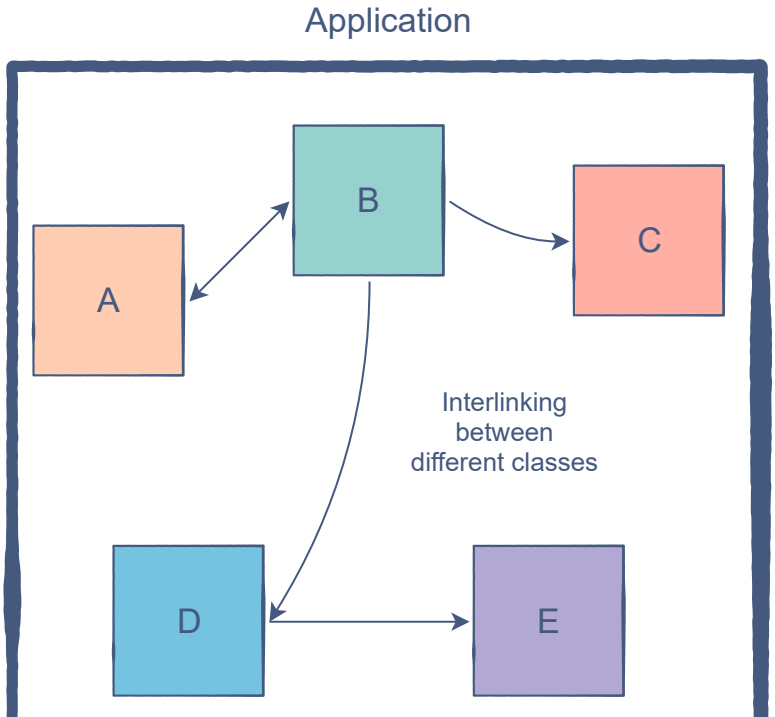
In this lesson, we will take a look at why composition, aggregation, and association are useful concepts in OOP.

WE'LL COVER THE FOLLOWING

Interaction Between Classes

By now, we've learned all we need to know about the creation and the behavior of a class. The concepts of **inheritance** and **polymorphism** taught us how to create dependent classes out of a base class.

The next step after object-oriented programming is **object-oriented design (OOD)**. OOD deals with the use of different class objects to create the design of an application. Naturally, this means that independent classes would have to find a way of interacting with each other.



Connecting OOP and OOD

That interaction is what this section is all about. Composition, aggregation, and association are the final concepts of object-oriented programming. They act as a bridge between OOP and OOD. These three techniques are used to link different classes together through a variety of relationships. Before we delve into these techniques, let's understand the different types of relationships they use

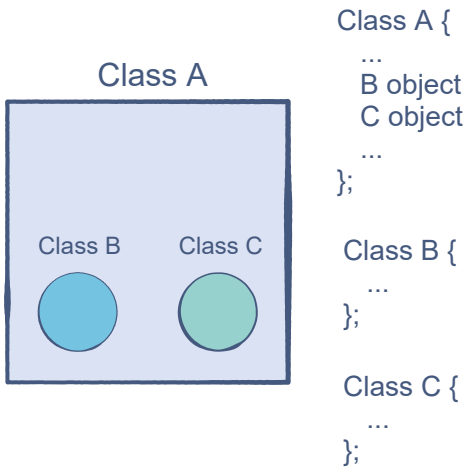
Relationships Between Classes

There are two main class relationships we need to know. We'll study them one by one.

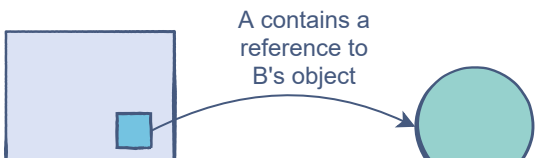
Part-of

In this relationship, one class is a **component** of another class. Given two classes, **class A** and **class B**, they are in a **part-of** relation if **class A** is a *part of* **class B** or vice-versa.

An instance of the component class can only be created inside the main class. In the example to the right, class B and class C have their own implementations, but their objects are only created once a class A object is created. Hence, **part-of** is a dependent relationship.



Class A contains objects of Class B and C.



Has-a

This is a slightly less concrete relationship between two classes. Class A and class B have a **has-a**

The object of class exists independent of A.

relationship if one or both need the other's object to perform an operation, but both classes can exist independently of each other.

This implies that a class *has a* reference to the object of the other class, but does not decide the lifetime of the other class's referenced object.

Now that we've understood the relationships relevant to this section, let's start off with composition.