10.1 — Object relationships

BY ALEX ON AUGUST 10TH, 2016 | LAST MODIFIED BY ALEX ON SEPTEMBER 27TH, 2016

Life is full of recurring patterns, relationships, and hierarchies between objects. By exploring and understanding these, we can gain insight into how real-life objects behave, enhancing our understanding of those objects.

For example, let's say one day you're walking down the street, and you see a bright yellow object attached to a green shrubby object. You'd probably recognize that the bright yellow thing is a flower, and the green shrubby thing is a plant. Even though you'd never seen this particular type of plant before, you'd know that the green things are leaves, collecting sunlight. You'd know that the flower helps the plant propagate itself. You'd also know that if you killed the plant, the flower would die too.

But how can you know all of this without ever encountering a plant of this type before? You know this because you understand the abstract concept of plants, and recognize that this plant is an instantiation of that abstraction. You know that most plants are composed (in part) of leaves, and some have flowers. You know that the leaves interact with the sunlight (even if you don't know how, exactly), and that the flower's existence depends on the plant. Because you know all of these things about plants in general, you can infer a lot about this plant.

Similarly, programming is also full of recurring patterns, relationships and hierarchies. Particularly when it comes to programming objects, the same patterns that govern real-life objects are applicable to the programming objects we create ourselves. By examining these in more detail, we can better understand how to improve code reusability and write classes that are more extensible.

In previous chapters, we've already explored some ideas around recurring patterns: we've created loops and functions to allow us to do a particular task many times. Additionally, we've created our own enums, structs, and classes to allow us to instantiate objects of a given type.

We've also explored some primitive forms of hierarchy, such as arrays (which allow us to group elements into a larger structure) and recursion, where a function calls a derivative version of itself.

However, we haven't yet focused much on the relationship between objects, particularly as it relates to programming.

Relationships between objects

There are many different kinds of relationships two objects may have in real-life, and we use specific "relation type" words to describe these relationships. For example: a square "is-a" shape. A car "has-a" steering wheel. A computer programmer "uses-a" keyboard. A flower "depends-on" a bee for pollination. A student is a "member-of" a class. And your brain exists as "part-of" you (at least, we can reasonably assume so if you've gotten this far).

All of these relation types have useful analogies in C++.

In this chapter, we'll explore the nuances of the relation types "part-of", "has-a", "uses-a", "depends-on", and "member-of", and show how they can be useful in the context of C++ classes. We'll also explore a couple of related topics that don't fit nicely anywhere else.

Then we'll devote the following two chapters to exploring "is-a" relationships, via C++'s inheritance model and virtual functions. Yup, it's a biggie.

Alright, enough context setting. Let's get to it.

 **10.2 -- Composition**

 **Index**

 **9.x -- Chapter 9 comprehensive quiz**