11.1 — Introduction to inheritance

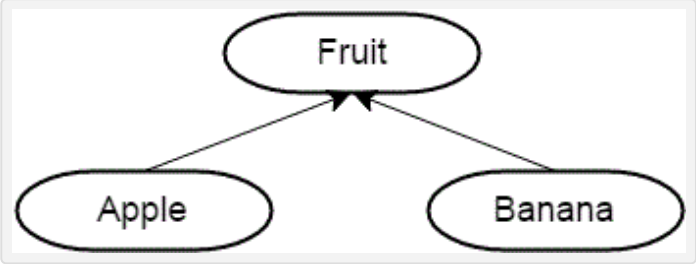
BY ALEX ON DECEMBER 19TH, 2007 | LAST MODIFIED BY ALEX ON JUNE 27TH, 2017

In the last chapter, we discussed object composition, where complex classes are constructed from simpler classes and types. Object composition is perfect for building new objects that have a “has-a” relationship with their parts. However, object composition is just one of the two major ways that C++ lets you construct complex classes. The second way is through inheritance, which models an “is-a” relationship between two objects.

Unlike object composition, which involves creating new objects by combining and connecting other objects, inheritance involves creating new objects by directly acquiring the attributes and behaviors of other objects and then extending or specializing them. Like object composition, inheritance is everywhere in real life. When you were conceived, you inherited your parents genes, and acquired physical attributes from both of them -- but then you added your own personality on top. Technological products (computers, cell phones, etc…) inherit features from their predecessors (often used for backwards compatibility). For example, the Intel Pentium processor inherited many of the features defined by the Intel 486 processor, which itself inherited features from earlier processors. C++ inherited many features from C, the language upon which it is based, and C inherited many of its features from the programming languages that came before it.

Consider apples and bananas. Although apples and bananas are different fruits, both have in common that they *are* fruits. And because apples and bananas are fruits, simple logic tells us that anything that is true of fruits is also true of apples and bananas. For example, all fruits have a name, a color, and a size. Therefore, apples and bananas also have a name, a color, and a size. We can say that apples and bananas inherit (acquire) these all of the properties of fruit because they *are* fruit. We also know that fruit undergoes a ripening process, by which it becomes edible. Because apples and bananas are fruit, we also know that apples and bananas will inherit the behavior of ripening.

Put into a diagram, the relationship between apples, bananas, and fruit might look something like this:

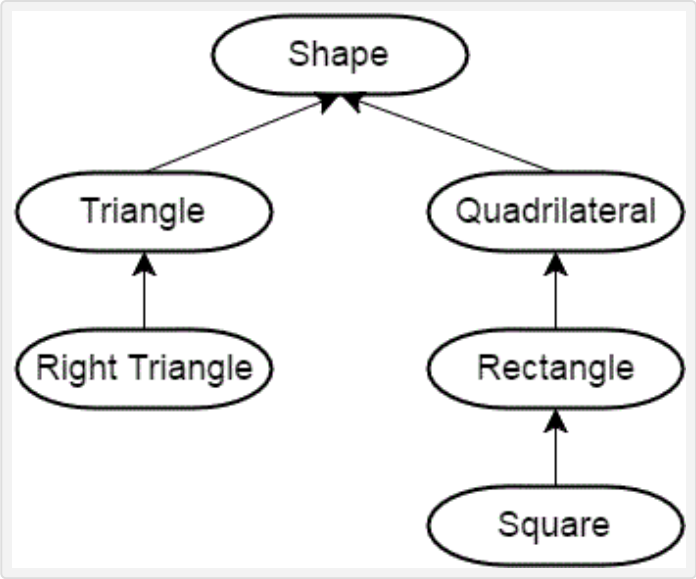


This diagram defines a hierarchy.

Hierarchies

A hierarchy is a diagram that shows how various objects are related. Most hierarchies either show a progression over time (386 -> 486 -> Pentium), or categorize things in a way that moves from general to specific (fruit -> apple -> red delicious). If you’ve ever taken biology, the famous domain, kingdom, phylum, class, order, family, genus, and species ordering defines a hierarchy (from general to specific).

Here’s another example of a hierarchy: a square is a rectangle, which is a quadrilateral, which is a shape. A right triangle is a triangle, which is also a shape. Put into a hierarchy diagram, that would look like this:



This diagram goes from general (top) to specific (bottom), with each item in the hierarchy inheriting the properties and behaviors of the item above it.

A look ahead

In this chapter, we'll explore the basics of how inheritance works in C++.

Next chapter, we'll explore how inheritance enables polymorphism (one of object-oriented programming's big buzzwords) through virtual functions.

As we progress, we'll also talk about inheritance's key benefits, as well as some of the downsides.

11.2 -- Basic inheritance in C++

Index

10.x -- Chapter 10 comprehensive quiz

Share this:

C++ TUTORIAL | PRINT THIS POST

16 comments to 11.1 — Introduction to inheritance

Cosmin
May 26, 2018 at 5:44 am · Reply

Hello,
I want to build a binary tree class and I'm trying to use a template so I can have different data-types binary trees. I have created first a Node class, it looks like this:

```
1 template<class T>
```