Early & Late Binding

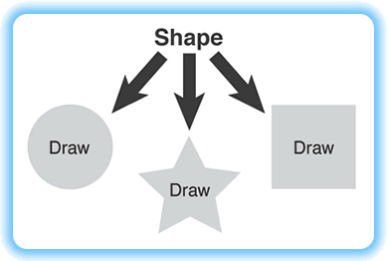
What would happen if you were to remove the keyword `virtual` from the definition of the `toString()` member function in the `Person` class? Your code would still compile, but the `toString()` function would no longer be overridden; it **would be hidden** in the derived class `Student`.

Functions are **bound** to an object depending on how they are declared. A non-virtual function is **bound at compile time** to the class that it is defined in. A non-virtual function defined in the `Person` class (such as `getName()`), will always be bound to the `Person` class, and **cannot be overridden** in any subsequent classes.

This is called **early binding** (or compile-time binding).

When you add the keyword `virtual` to a function, the function call binding is not determined at compile time, but **when the program is run**. Instead of looking at the type of the pointer or reference used in the function call, **the actual object pointed to** is used to decide which function to call.

This decision is made when your program runs. If your `Shape*` points to a `Circle` object, then `Circle::draw()` will be called, but **only if** `draw()` is a `virtual` function.



This is called **late-binding** or **dynamic dispatch**. In Java, **all** methods use late binding, but in C++ you, as the base-class designer get to decide which version to use, through the application of the keyword `virtual`.