Override and Final

In this lesson, we'll discuss override and final in detail.

override

The `override` keyword in a method declaration expresses that the method should override a virtual method of a base class. The compiler checks this assertion. It checks the parameter of the method, the return type of the method, and qualifiers like const and volatile. Of course, the compiler notices if the overridden method is not virtual.

The compiler verifies if the `override` annotated method actually overrides a virtual method of a base class.

- The compiler checks for
  - The parameters and the return type.
  - The constness of the method.
  - The virtuality of the method.

The compiler ensures that the programmer obeys the contract.

By using the context-sensitive keywords `override` and `final`, we can explicitly manage the overriding of virtual functions. In particular, the keyword `override` solves a common bug present in object hierarchies: methods that should override methods of base classes but do not. The result is a syntactically but not semantically correct program. The program performs the wrong actions in the right way.

`override`

To override a method, the signature of the overridden method of the base class has to match exactly. Although this sounds easy in theory, it is often not in practice. If the signature of the method does not match exactly, the program will compile but have the wrong behavior… A different method than intended will be invoked.

`final`

`final` supports two use cases. First, we can declare a method that cannot be overridden; second, we can define a class that cannot be derived from. The compiler uses the same rules to determine if a method of child class overrides a method of a base class. Of course, the strategy is inverted because the final specifier should disallow the overriding of a method. Therefore, the compiler checks the parameters of the method, its return type, and any `const/volatile` qualifiers.

A virtual method declared `final` must not be overridden.

- The compiler checks for
  - The parameter.
  - The return type.
  - The constness of the method.
  - Methods and classes declared as `final` are an optimization opportunity for the compiler.
  - Both variants are equivalent:

```
void func() final;
virtual void func() final override;
```

The compiler ensures that the programmer obeys the contract.

To learn more about **override**, click here.

To learn more about **final**, click here.

In the next lesson, we'll look at some examples of override and final.