Inheritance
- Defining a Base Class
- Subclassing
- Overriding
  - Accessing Superclass Methods, Properties, and Subscripts
  - Overriding Methods
  - Overriding Properties
    - Overriding Property Getters and Setters
    - Overriding Property Observers
- Preventing Overrides

Inheritance

Defining a Base Class

> Swift classes do not inherit from a universal base class. Classes you define without specifying a superclass automatically become base classes for you to build upon.

```
class Vehicle {
    var currentSpeed = 0.0
    var description: String {
        return "traveling at \(currentSpeed) miles per hour"
    }
    func makeNoise() {
        // do nothing - an arbitrary vehicle doesn't necessarily make a noise
    }
}
```

Subclassing

To indicate that a subclass has a superclass, write the subclass name before the superclass name, separated by a colon:

```
class SomeSubclass: SomeSuperclass {
    // subclass definition goes here
}
```

Overriding

A subclass can provide its own custom implementation of **an instance method, type method, instance property, type property, or subscript** that it would otherwise inherit from a superclass. This is known as *overriding*.

To override a characteristic that would otherwise be inherited, you prefix your overriding definition with the `override` keyword. Doing so clarifies that you intend to provide an override and have not provided a matching definition by mistake. Overriding by accident can cause unexpected behavior, and any overrides without the `override` keyword are diagnosed as an error when your code is compiled.

The `override` keyword also prompts the Swift compiler to check that your overriding class's superclass (or one of its parents) has a declaration that matches the one you provided for the override. This check ensures that your overriding definition is correct.

Accessing Superclass Methods, Properties, and Subscripts

When you provide a method, property, or subscript override for a subclass, it is sometimes useful to use the existing superclass implementation as part of your override. For example, you can refine the behavior of that existing implementation, or store a modified value in an existing inherited variable.

Where this is appropriate, you access the superclass version of a method, property, or subscript by using the `super` prefix:

- An overridden method named `someMethod()` can call the superclass version of `someMethod()` by calling `super.someMethod()` **within the overriding method implementation.**
- An overridden property called `someProperty` can access the superclass version of `someProperty` as `super.someProperty` **within the overriding getter or setter implementation.**
- An overridden subscript for `someIndex` can access the superclass version of the same subscript as `super[someIndex]` from **within the overriding subscript implementation.**

Overriding Methods

```
class Train: Vehicle {
    override func makeNoise() {
        print("Choo Choo")
    }
}
```

Overriding Properties

You can override an inherited instance or type property to provide your own custom getter and setter for that property, or to add property observers to enable the overriding property to observe when the underlying property value changes.

Overriding Property Getters and Setters

You can provide a custom getter (and setter, if appropriate) to override *any* inherited property, regardless of whether the inherited property is implemented as a stored or computed property at source. **The stored or computed nature of an inherited property is not known by a subclass—it only knows that the inherited property has a certain name and type.** You must always state both the name and the type of the property you are overriding, to enable the compiler to check that your override matches a superclass property with the same name and type.

You can present an inherited read-only property as a read-write property by providing both a getter and a setter in your subclass property override. You cannot, however, present an inherited read-write property as a read-only property.

> If you provide a setter as part of a property override, you **must** also provide a getter for that override. If you don't want to modify the inherited property's value within the overriding getter, you can simply pass through the inherited value by returning `super.someProperty` from the getter, where `someProperty` is the name of the property you are overriding.

Overriding Property Observers

You can use property overriding to add property observers to an inherited property. This enables you to be notified when the value of an inherited property changes, regardless of how that property was originally implemented. For more information on property observers, see Property Observers.

> You cannot add property observers to inherited constant stored properties or inherited read-only computed properties. The value of these properties cannot be set, and so it is not appropriate to provide a `willSet` or `didSet` implementation as part of an override.
>
> Note also that you cannot provide both an overriding setter and an overriding property observer for the same property. If you want to observe changes to a property's value, and you are already providing a custom setter for that property, you can simply observe any value changes from within the custom setter.

Preventing Overrides

You can prevent a method, property, or subscript from being overridden by marking it as *final*. Do this by writing the `final` modifier before the method, property, or subscript's introducer keyword (such as `final var`, `final func`, `final class func`, and `final subscript`).

Any attempt to override a final method, property, or subscript in a subclass is reported as a compile-time error. Methods, properties, or subscripts that you add to a class in an extension can also be marked as final within the extension's definition.

You can mark an entire class as final by writing the `final` modifier before the `class` keyword in its class definition (`final class`). Any attempt to subclass a final class is reported as a compile-time error.