Multiple Inheritance

In some programming languages, a class can inherit the interface of multiple base classes. Known as multiple inheritance, this feature can add significant complexity to the language and is unsupported in Swift. Instead, Swift allows composition of interfaces using protocols.

Consider the following example:

```
protocol Utensil {
    var name: String { get }
}

protocol ServingUtensil: Utensil {
    func serve()
}

extension ServingUtensil {
    func serve() { /* Default implementation. */ }
}

protocol Fork: Utensil {
    func spear()
}

protocol Spoon: Utensil {
    func scoop()
}

struct CarvingFork: ServingUtensil, Fork { /* ... */ }
struct Spork: Spoon, Fork { /* ... */ }
struct Ladle: ServingUtensil, Spoon { /* ... */ }
```

Swift protocols can declare interfaces that must be implemented by each conforming type (like abstract class members in other programming languages such as C# or Java), and they can also provide overridable default implementations for those requirements in protocol extensions.

When class inheritance and protocol conformances are used together, subclasses inherit protocol conformances from base classes, introducing additional complexity. For example, the default implementation of a protocol requirement not overridden in the conforming base class also cannot be overridden in any subclass (SR-103).

To learn more about defining and adopting protocols, see the Protocols section in *The Swift Programming Language*. To learn more about class inheritance, see the Inheritance section in *The Swift Programming Language*.