Nominal Types

In Swift, a type is considered a nominal type if it has been explicitly named by a declaration somewhere in code. Examples of nominal types include classes, structures and enumerations. Nominal types are an important concept in Swift because they may conform to protocols, be extended, and have values created using the initializer syntax `MyType()`.

In contrast, non-nominal types do not have these capabilities. Many are obtained by composing other types. Examples include function types like `(Int) -> (String)`, tuple types like `(Int, String)`, metatypes like `Int.Type`, and special types like `Any` and `AnyObject`.

Since a protocol is named by a declaration in code, it may conform to (in other words, refine) other protocols and it may be extended. However, when written as the type of a constant or variable such as `let value: MyProtocol`, the name refers to a distinct, non-nominal existential type that provides a "box" for a value of any concrete type that conforms to the protocol. The existential type itself does not conform to any protocols and cannot be extended, and a value cannot be created using the initializer syntax `MyProtocol()`.

For more on using existential types, see Protocols as Types in *The Swift Programming Language*.