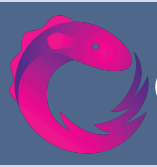
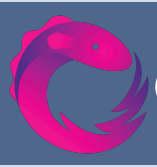
Advanced RxSwift – Day 2

Younghwan Kim



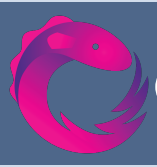
Grouping – Protocol Extension, associatetype

```
public struct Extension<Base> {
    public let base: Base // Generic Type (Base)
    public init(_ base: Base) {
        self.base = base
    }
}
```



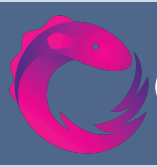
Grouping – Protocol Extension, associatetype

```
public struct Extension<Base> {
    public let base: Base // Generic Type (Base)
    public init(_ base: Base) {
        self.base = base
    }
}

//  Associated Types
//
//  When defining a protocol, it's sometimes useful to declare one or more associated types as part of the protocol's definition.
//  An associated type gives a placeholder name to a type that is used as part of the protocol. The actual type to use for
//  that associated type isn't specified until the protocol is adopted. Associated types are specified with the associatedtype keyword.

public protocol ExtensionCompatible {
    associatedtype Compatible
    var ex: Extension<Compatible> { get }
}
```



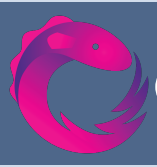
Grouping – Protocol Extension, associatetype

```
//  Protocol Associated Type Declaration
//
//  Protocols declare associated types using the associatedtype keyword. An associated type provides an alias for a type that is used
//    as part of a protocol's declaration. Associated types are similar to type parameters in generic parameter clauses, but they're
//    associated with Self in the protocol in which they're declared. In that context, Self refers to the eventual type that conforms
//    to the protocol. For more information and examples, see Associated Types.

// Generic Conforming
public extension ExtensionCompatible {
    //Self refers to the eventual type that conforms to the protocol
    public var ex: Extension<Self> {
        get {
            return Extension(self)
        }
    }
}
```



Grouping – Protocol Extension, associatetype

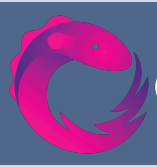
```
extension Int: ExtensionCompatible { }

public extension Extension where Base == Int {
    public var cube: Int {
        return base * base * base
    }
}


 print("\(4.ex.cube)")
```



Grouping – Protocol Extension, associatetype

```
public struct ExtensionTwo<Base> {
    public let base: Base
    public init(_ base: Base) {
        self.base = base
    }
}

public protocol ExtensionTwoCompatible {
    associatedtype Compatible
    var ext: ExtensionTwo<Compatible> { get set }
    static var ext: ExtensionTwo<Compatible>.Type
    { get set }
}
```

```
public extension ExtensionTwoCompatible {
    public var ext: ExtensionTwo<Self> {
        get {
            return ExtensionTwo(self)
        }
        set {
            // this enables using Extension to "mutate" base object
        }
    }
    public static var ext: ExtensionTwo<Self>.Type {
        get {
            return ExtensionTwo<Self>.self
        }
        set {
            // this enables using Extension to "mutate" base type
        }
    }
}
```



Grouping – Protocol Extension, associatetype

```
extension Int: ExtensionTwoCompatible { }

public extension ExtensionTwo where Base == Int {
    public var cube: Int {
        return base * base * base
    }
    public static var almostMax: Int {
        return Int.max - 1
    }
}


print("\(4.ext.cube)")
print("\(Int.ext.almostMax)")
```