Chapter 38: Style Conventions

Section 38.1: Fluent Usage

Using natural language
Functions calls should be close to natural English language.

Example:

```
list.insert(element, at: index)
```

instead of

```
list.insert(element, position: index)
```

Naming Factory Methods
Factory methods should begin with the prefix `make`.

Example:

```
factory.makeObject()
```

Naming Parameters in Initializers and Factory Methods
The name of the first argument should not be involved in naming a factory method or initializer.

Example:

```
factory.makeObject(key: value)
```

Instead of:

```
factory.makeObject(havingProperty: value)
```

Naming according to side effects

- Functions with side effects (mutating functions) should be named using verbs or nouns prefixed with `form-` .
- Functions without side effects (nonmutating functions) should be named using nouns or verbs with the suffix `-ing` or `-ed`.

Example: Mutating functions:

```
print(value)
array.sort()                // in place sorting
list.add(value)             // mutates list
set.formUnion(anotherSet)   // set is now the union of set and anotherSet
```

Nonmutating functions:

```
let sortedArray = array.sorted()     // out of place sorting
let union = set.union(anotherSet)    // union is now the union of set and another set
```

Boolean functions or variables
Statements involving booleans should read as assertions.

Example:

```
set.isEmpty
line.intersects(anotherLine)
```

Naming Protocols

- Protocols describing what something is should be named using nouns.
- Protocols describing capabilities should have `-able`, `-ible` or `-ing` as suffix.

Example:

```
Collection        // describes that something is a collection
ProgressReporting // describes that something has the capability of reporting progress
Equatable         // describes that something has the capability of being equal to something
```

Types and Properties

Types, variables and properties should read as nouns.

Example:

```
let factory = ...
let list = [1, 2, 3, 4]
```

Section 38.2: Clear Usage

Avoid Ambiguity

The name of classes, structures, functions and variables should avoid ambiguity.

Example:

```
extension List {
    public mutating func remove(at position: Index) -> Element {
        // implementation
    }
}
```

The function call to this function will then look like this:

```
list.remove(at: 42)
```

This way, ambiguity is avoided. If the function call would be just `list.remove(42)` it would be unclear, if an Element equal to 42 would be removed or if the Element at Index 42 would be removed.

Avoid Redundancy

The name of functions should not contain redundant information.

A bad example would be:

```
extension List {
    public mutating func removeElement(element: Element) -> Element? {
        // implementation
    }
}
```

A call to the function may look like `list.removeElement(someObject)`. The variable `someObject` already indicates, that an Element is removed. It would be better for the function signature to look like this:

```
extension List {
    public mutating func remove(_ member: Element) -> Element? {
        // implementation
    }
}
```

The call to this function looks like this: `list.remove(someObject)`.

Naming variables according to their role
Variables should be named by their role (e.g. supplier, greeting) instead of their type (e.g. factory, string, etc..)

High coupling between Protocol Name and Variable Names
If the name of the type describes its role in most cases (e.g. Iterator), the type should be named with the suffix `Type`. (e.g. IteratorType)
Provide additional details when using weakly typed parameters
If the type of an object does not indicate its usage in a function call clearly, the function should be named with a preceding noun for every weakly typed parameter, describing its usage.
Example:

```
func addObserver(_ observer: NSObject, forKeyPath path: String)
```

to which a call would look like `object.addObserver(self, forKeyPath: path)

instead of

```
func add(_ observer: NSObject, for keyPath: String)
```

to which a call would look like `object.add(self, for: path)`

Section 38.3: Capitalization

Types & Protocols

Type and protocol names should start with an uppercase letter.

Example:

```
protocol Collection {}
struct String {}
class UIView {}
struct Int {}
enum Color {}
```

Everything else...

Variables, constants, functions and enumeration cases should start with a lowercase letter.

Example:

```
let greeting = "Hello"
let height = 42.0

enum Color {
```

```
    case red
    case green
    case blue
}

func print(_ string: String) {
    ...
}
```

Camel Case:

All naming should use the appropriate camel case. Upper camel case for type/protocol names and lower camel case for everything else.

Upper Camel Case:

```
protocol IteratorType { ... }
```

Lower Camel Case:

```
let inputView = ...
```

Abbreviations

Abbreviations should be avoided unless commonly used (e.g. URL, ID). If an abbreviation is used, all letters should have the same case.

Example:

```
let userID: UserID = ...
let urlString: URLString = ...
```