Chapter 27: Reflection

Section 27.1: Basic Usage for Mirror

Creating the class to be the subject of the Mirror

```
class Project {
    var title: String = ""
    var id: Int = 0
    var platform: String = ""
    var version: Int = 0
    var info: String?
}
```

Creating an instance that will actually be the subject of the mirror. Also here you can add values to the properties of the Project class.

```
let sampleProject = Project()
sampleProject.title = "MirrorMirror"
sampleProject.id = 199
sampleProject.platform = "iOS"
sampleProject.version = 2
sampleProject.info = "test app for Reflection"
```

The code below shows the creating of Mirror instance. The children property of the mirror is a `AnyForwardCollection<Child>` where `Child` is typealias tuple for subject's property and value. `Child` had a `label: String` and `value: Any`.

```
let projectMirror = Mirror(reflecting: sampleProject)
let properties = projectMirror.children

print(properties.count)        //5
print(properties.first?.label) //Optional("title")
print(properties.first!.value) //MirrorMirror
print()

for property in properties {
    print("\(property.label!):\(property.value)")
}
```

Output in Playground or Console in Xcode for the for loop above.

```
title:MirrorMirror
id:199
platform:iOS
version:2
info:Optional("test app for Reflection")
```

Tested in Playground on Xcode 8 beta 2

Section 27.2: Getting type and names of properties for a class without having to instantiate it

Using the Swift class `Mirror` works if you want to extract *name*, *value* and *type* (Swift 3: `type(of: value)`, Swift 2: `value.dynamicType`) of properties for an **instance** of a certain class.

If you class inherits from `NSObject`, you can use the method `class_copyPropertyList` together with `property_getAttributes` to find out the *name* and *types* of properties for a class - **without having an instance of it**. I created a project on Github for this, but here is the code:

```
func getTypesOfProperties(in clazz: NSObject.Type) -> Dictionary<String, Any>? {
    var count = UInt32()
    guard let properties = class_copyPropertyList(clazz, &count) else { return nil }
    var types: Dictionary<String, Any> = [:]
    for i in 0..<Int(count) {
        guard let property: objc_property_t = properties[i], let name = getNameOf(property:
property) else { continue }
        let type = getTypeOf(property: property)
        types[name] = type
    }
    free(properties)
    return types
}

func getTypeOf(property: objc_property_t) -> Any {
    guard let attributesAsNSString: NSString = NSString(utf8String:
property_getAttributes(property)) else { return Any.self }
    let attributes = attributesAsNSString as String
    let slices = attributes.components(separatedBy: "\"")
    guard slices.count > 1 else { return getPrimitiveDataType(withAttributes: attributes) }
    let objectClassName = slices[1]
    let objectClass = NSClassFromString(objectClassName) as! NSObject.Type
    return objectClass
}

   func getPrimitiveDataType(withAttributes attributes: String) -> Any {
        guard let letter = attributes.substring(from: 1, to: 2), let type =
primitiveDataTypes[letter] else { return Any.self }
        return type
    }
```

Where `primitiveDataTypes` is a Dictionary mapping a letter in the attribute string to a value type:

```
let primitiveDataTypes: Dictionary<String, Any> = [
    "c" : Int8.self,
    "s" : Int16.self,
    "i" : Int32.self,
    "q" : Int.self, //also: Int64, NSInteger, only true on 64 bit platforms
    "S" : UInt16.self,
    "I" : UInt32.self,
    "Q" : UInt.self, //also UInt64, only true on 64 bit platforms
    "B" : Bool.self,
    "d" : Double.self,
    "f" : Float.self,
    "{" : Decimal.self
]

   func getNameOf(property: objc_property_t) -> String? {
        guard let name: NSString = NSString(utf8String: property_getName(property)) else { return
nil }
        return name as String
    }
```

It can extract the `NSObject.Type` of all properties which class type inherits from `NSObject` such as `NSDate` (Swift3: `Date`), `NSString`(Swift3: `String`?) and `NSNumber`, however it is store in the type `Any` (as you can see as the type of the value of the Dictionary returned by the method). This is due to the limitations of `value types` such as Int, Int32,

Bool. Since those types do not inherit from NSObject, calling `.self` on e.g. an Int - `Int.self` does not return NSObject.Type, but rather the type `Any`. Thus the method returns `Dictionary<String, Any>?` and not `Dictionary<String, NSObject.Type>?`.

You can use this method like this:

```
class Book: NSObject {
    let title: String
    let author: String?
    let numberOfPages: Int
    let released: Date
    let isPocket: Bool

    init(title: String, author: String?, numberOfPages: Int, released: Date, isPocket: Bool) {
        self.title = title
        self.author = author
        self.numberOfPages = numberOfPages
        self.released = released
        self.isPocket = isPocket
    }
}

guard let types = getTypesOfProperties(in: Book.self) else { return }
for (name, type) in types {
    print("'\(name)' has type '\(type)'")
}
// Prints:
// 'title' has type 'NSString'
// 'numberOfPages' has type 'Int'
// 'author' has type 'NSString'
// 'released' has type 'NSDate'
// 'isPocket' has type 'Bool'
```

You can also try to cast the `Any` to `NSObject.Type`, which will succeed for all properties inheriting from `NSObject`, then you can check the type using standard `==` operator:

```
func checkPropertiesOfBook() {
    guard let types = getTypesOfProperties(in: Book.self) else { return }
    for (name, type) in types {
        if let objectType = type as? NSObject.Type {
            if objectType == NSDate.self {
                print("Property named '\(name)' has type 'NSDate'")
            } else if objectType == NSString.self {
                print("Property named '\(name)' has type 'NSString'")
            }
        }
    }
}
```

If you declare this custom `==` operator:

```
func ==(rhs: Any, lhs: Any) -> Bool {
    let rhsType: String = "\(rhs)"
    let lhsType: String = "\(lhs)"
    let same = rhsType == lhsType
    return same
}
```

You can then even check the type of `value types` like this:

```
func checkPropertiesOfBook() {
    guard let types = getTypesOfProperties(in: Book.self) else { return }
    for (name, type) in types {
        if type == Int.self {
            print("Property named '\(name)' has type 'Int'")
        } else if type == Bool.self {
            print("Property named '\(name)' has type 'Bool'")
        }
    }
}
```

LIMITATIONS This solution does not work when `value types` are optionals. If you have declared a property in you NSObject subclass like this: `var myOptionalInt: Int?`, the code above won't find that property because the method `class_copyPropertyList` does not contain optional value types.