Chapter 26/28

Data Persistence

Scene Storage

- used to store small amounts of data within the scope of individual app scene instances using `@SceneStorage`
- saves/restores the state of a scene when an app is terminated while in the background so it can be retrieved the next time the scene is loaded
- limited to ***primitive*** data types

> creates a binding between the variable and the associated view; stores data persistently so that it is accessible from the view it is declared on even after the application closes.

```
struct AuthenticationView: View {
    @SceneStorage("token") private var token: String = ""

    var body: some View {
        TextEditor(text: $token)
            .padding()
    }
}


var body: some View {
    AuthenticationView()
}
```

App Storage

- used to store small amounts of data that is universally available throughout the entire app using `@AppStorage`

- data is stored in real-time (when the application is in the foreground)
- limited to ***primitive*** data types

> creates a binding between the variable and the associated view; stores data persistently so that it is accessible from any view even after the application closes.

```
struct AuthenticationView: View {
    @AppStorage("token") var token: String = ""

    var body: some View {
        TextEditor(text: $token)
            .padding()
    }
}

var body: some View {
    AuthenticationView()
}
```

Object Storage

- object's type must conform to the `Encodable` and `Decodable` protocols

```
struct UserName: Encodable, Decodable {
    var firstName: String
    var secondName: String
}

@AppStorage("username") var namestore: Data = Data()

if let data = try? JSONEncoder().encode(username) {
    namestore = data
}

if let name = try? JSONDecoder().decode(UserName.self, from: namestore) {
    username = name
}
```

Lists and Navigation

List View

- describes what elements are shown in the list of the user interface

```
List {
    HStack {
        Image(systemName: "trash.circle.fill")
        Text("Take out the trash")
    }
    HStack {
        Image(systemName: "person.2.fill")
        Text("Pick up the kids")
    }
    HStack {
```

```
        Image(systemName: "car.fill")
        Text("Wash the car")
    }
}
```

Section View

- creates a separate section of list elements whose title is taken from the header parameter as a way to differentiate between lists

```
Section(header: Text("Section Name")) {
    ...
}
```

DisclosureGroup View

- creates a collapsible list of elements where the label is taken from the second closure parameter and the content parameter contains the list of elements

```
DisclosureGroup(content: {
    ...
}) {
    Text("Sub-Section Name")
}
```

List Navigation

```
NavigationView {
    List {
        Section(header: Text("Nutrition Facts")) {
            NavigationLink(destination: ...) {
                Text("Chipotle")
            }
        }
    }
}
```

Outline Group

- the `List` initializer creates a `DisclosureGroup` for an `Element` whose `childNodes` property is not `nil`

```
struct Element: Identifiable {
    var id = UUID()
    var name: String
    var description: String?
    var childNodes: [Element]?
}

struct CrossWalkInfoOutlineGroup: View {
    @State private var info = [
        Element(name: "Crosswalk name", description: "Name of the crosswalk"),
        Element(name: "Crosswalk address", description: "Address of the crosswalk"),
        Element(name: "Volunteer", description: "Name of the volunteer"),
        Element(name: "Maximum hours", childNodes: [
```

```
            Element(name: "Minors", description: "Minors can only volunteer for 1 hour and accompanied by an adult"),
            Element(name: "Adults", description: "Adults can volunteer for a maximum of 3 hours."),
            Element(name: "Seniors", description: "Seniors can volunteer for a maximum of 2 hours.")
        ])
    ]

    var body: some View {
        NavigationView {
            VStack {
                // Version 1
                // the "\" symbols is used to define the location/path of a property in reference to an object
                List(info, children: \.childNodes) { element in
                    if let description = element.description {
                        NavigationLink(destination: Text(description)) {
                            Text("\(element.name)")
                        }
                    } else {
                        Text("\(element.name)")
                    }
                }

                // Version 2
                ForEach (info, id: \.childNodes) { element in
                    if let description = element.description {
                        NavigationLink(destination: Text(description)) {
                            Text("\(element.name)")
                        }
                    } else {
                        Text("\(element.name)")
                    }
                }
            }
        }
    }
}
```