Observable object, ObservableObject and @Published are all related to reactive programming in SwiftUI.

Observable object is a protocol that allows objects to be observed for changes. When an Observable object changes, all of its observers are notified. This makes it easy to create user interfaces that automatically update when the data they are displaying changes.

ObservableObject is a class that conforms to the observable object protocol. It provides a number of features that make it easy to implement reactive programming, including:

- **Published properties:** ObservableObject classes can publish properties, which are properties that automatically notify observers when they change.
- **ObjectWillChange:** ObservableObject classes can send an objectWillChange notification before they change. This allows observers to prepare for the change and update their state accordingly.
- **ObjectDidChange:** ObservableObject classes can send an objectDidChange notification after they change. This allows observers to update their state accordingly.

@Published is a property wrapper that can be used to publish properties of ObservableObject classes. Published properties are automatically observed by any views that bind to them.

To use ObservableObject and @Published in SwiftUI, you first need to create a class that conforms to the ObservableObject protocol. You can then publish any properties that you want to observe for changes by using the @Published property wrapper.

For example, the following code shows a simple ObservableObject class with a published property:

Swift .

```swift
class ViewModel: ObservableObject {

    @Published var name: String = ""

    init() {}

    func updateName(to name: String) {
        self.name = name
    }
}
```

This class publishes a property called **name**. When the **updateName()** method is called, the value of the **name** property is updated. This change is automatically notified to any views

that are observing the **name** property.

To use the **ViewModel** class in a SwiftUI view, you can simply bind to the **name** property:

```Swift
struct ContentView: View {

    @ObservedObject private var viewModel = ViewModel()

    var body: some View {
        Text(viewModel.name)
    }
}
```

This view will automatically update whenever the **name** property of the **ViewModel** changes.

You can also bind to multiple published properties in a single view. For example, the following code shows a view that binds to the **name** and **age** properties of a **ViewModel** class:

```Swift
struct ContentView: View {

    @ObservedObject private var viewModel = ViewModel()

    var body: some View {
        Text("Name: \(viewModel.name), Age: \(viewModel.age)")
    }
}
```

This view will now update whenever either the **name** or **age** property of the **ViewModel** changes.

Here are some additional benefits of using ObservableObject, @Published, and @ObservedObject in SwiftUI:

- **Code readability:** ObservableObject, @Published, and @ObservedObject can help to make your code more readable and easier to understand.
- **Maintainability:** ObservableObject, @Published, and @ObservedObject can help to make your code more maintainable by making it easier to change and evolve your user interface.
- **Testability:** ObservableObject, @Published, and @ObservedObject can help to make your code more testable by providing a clear separation of concerns.

Here is a summary of the differences between observable object, ObservableObject, and @Published:

Term	Definition
observable object	A protocol that allows objects to be observed for changes.
ObservableObject	A class that conforms to the observable object protocol.
@Published	A property wrapper that can be used to publish properties of ObservableObject classes.

Overall, ObservableObject, @Published, and @ObservedObject are valuable tools that can help you to write better SwiftUI code.