deliveroo

Coordinators

Managing lifetime and preventing memory leaks

14 / 03 / 2023

What is the current situation?

1. Multiple coordinator implementations with differing designs

2. Manual lifetime clean-up of coordinators

3. Missing handlers for automatic navigation (pop / dismiss)

Manual cleanup 🧼

- With our current pattern, when all the view-controllers managed by a coordinator have been deallocated, the coordinator needs to be manually deallocated too by calling **`didFinish`**

- It is very easy to forget or miss instances where we need to call **`didFinish`**, resulting in the coordinator being leaked

Example 🧑‍💻

- In the following example, we can see that **`didFinish`** is not called resulting in **`YellowCoordinator`** and **`GreenCoordinator`** being leaked
- Every time we launch either of these two flows, we bloat the memory footprint of the app
- As coordinators often manage expensive resources, this can be detrimental to our our apps performance and battery usage

Missing cleanup 😬

- With final CTA actions, or custom back/close buttons it's fairly easy to spot where **`didFinish`** needs to be called, however when hidden actions occur such as interactive dismiss & pop, it's much harder to spot leaks...

- It is very easy to forget or miss instances where we need to call **`didFinish`**, resulting in the coordinator being leaked

Common Solutions 💡

- One common solution is adding a hooks to the lifecycle events of the root view controller in the Coordinator's flow - **`viewDidDisappear`** or **`deinit`**

- Another option is to add hooks to **`UINavigationControllerDelegate`** or **`UIPopoverPresentationControllerDelegate`**

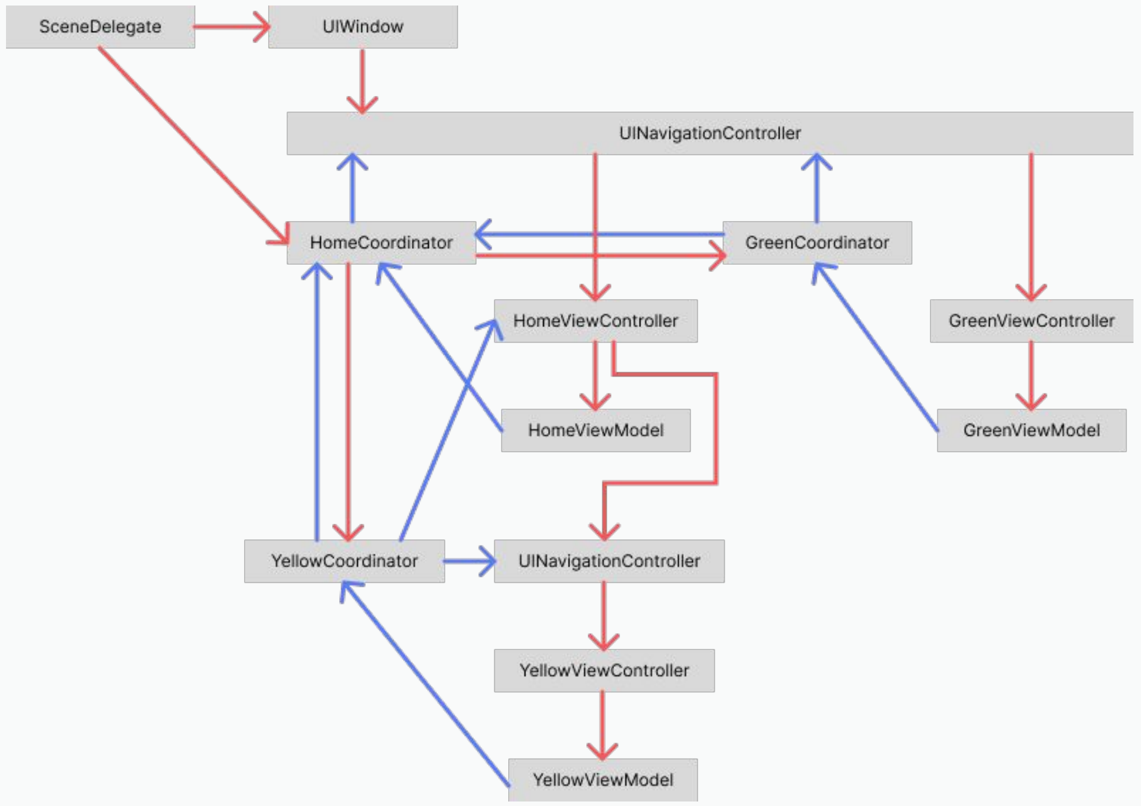
- Both of these techniques add additional boilerplate to the codebase...

A simpler approach 💪

- If we invert the ownership of Coordinators, we can let ARC do the hard work for us. Coordinators can be automatically deallocated when they are no-longer needed.

- This reduces the amount of boilerplate code and reduces the chance for memory leaks. It’s also what the Rider app does - so it’s proven in practice!

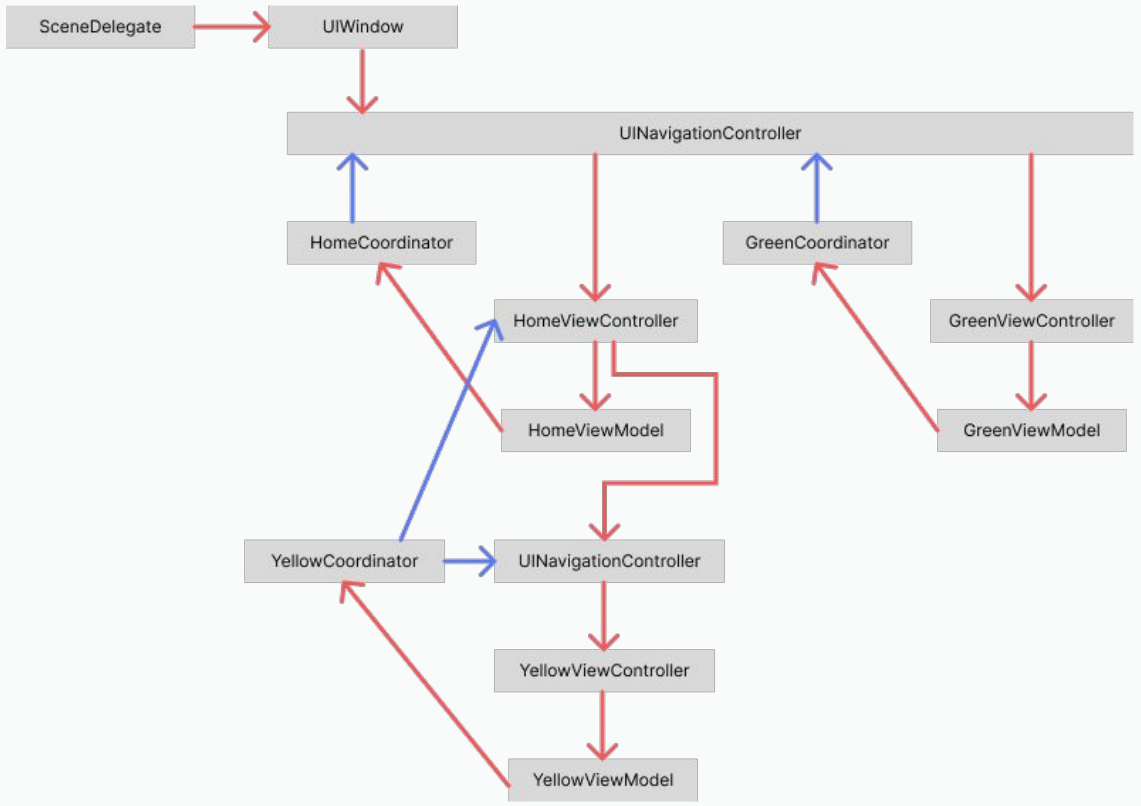
Ownership model



🟥 Strong reference

🟦 Weak reference

Ownership model



🟥 Strong reference

🟦 Weak reference

Example 🧑‍💻

- In the following example, we can see how to refactor the demo app to the new Coordinator pattern

- We're able to remove **`deinit`** handlers, calls to **`didFinish`** and parent / child Coordinator relationships, all reducing complexity

- We're also able to ensure correct memory management to help prevent memory leaks

Closures vs Protocols ✍️

To keep the Coordinator alive we can follow one of 2 patterns:

- Closures:
    - Removes need for protocol definitions
    - Coordinators functions can stay private
    - Using strong self in closures - can seem counterintuitive, especially for junior devs
- Coordinating protocols
    - Additional boilerplate around defining protocols
    - Coordinator functions cannot be private for protocol conformance
    - No strong self in closures, more developer-friendly

Composability 🤝

- When starting a new Coordinator from a traditional one, **`store(coordinator:)`** can simply be omitted as this reference is not needed.

- When starting a traditional Coordinator from a new one, simply hold a strong reference to it in the view model (either via a closure or protocol conformance). You won’t need to call **`didFinish`** either.

Testing 📄

One advantage of the Coordinator pattern presented, is that it highly testable. By introducing protocols for **`UIViewController`** and **`UINavigationController`**, we can even test Coordinators without any reference to UIKit

As can be seen in the example, the introduction of **`ScreenBuilder`** types allow tests to intercept view models, so the whole flows can be validated

Further Reading 📖

- Albert Montserrat Gambus - Self-deallocated Coordinator pattern in Swift
- Soroush Khanlou - Back Buttons and Coordinators
- Toby O’Connell - Coordinators, the back button problem and a simple way to fix it

Any questions?

github.com/tobyoconnelldeliveroo/CoordinatorDemo