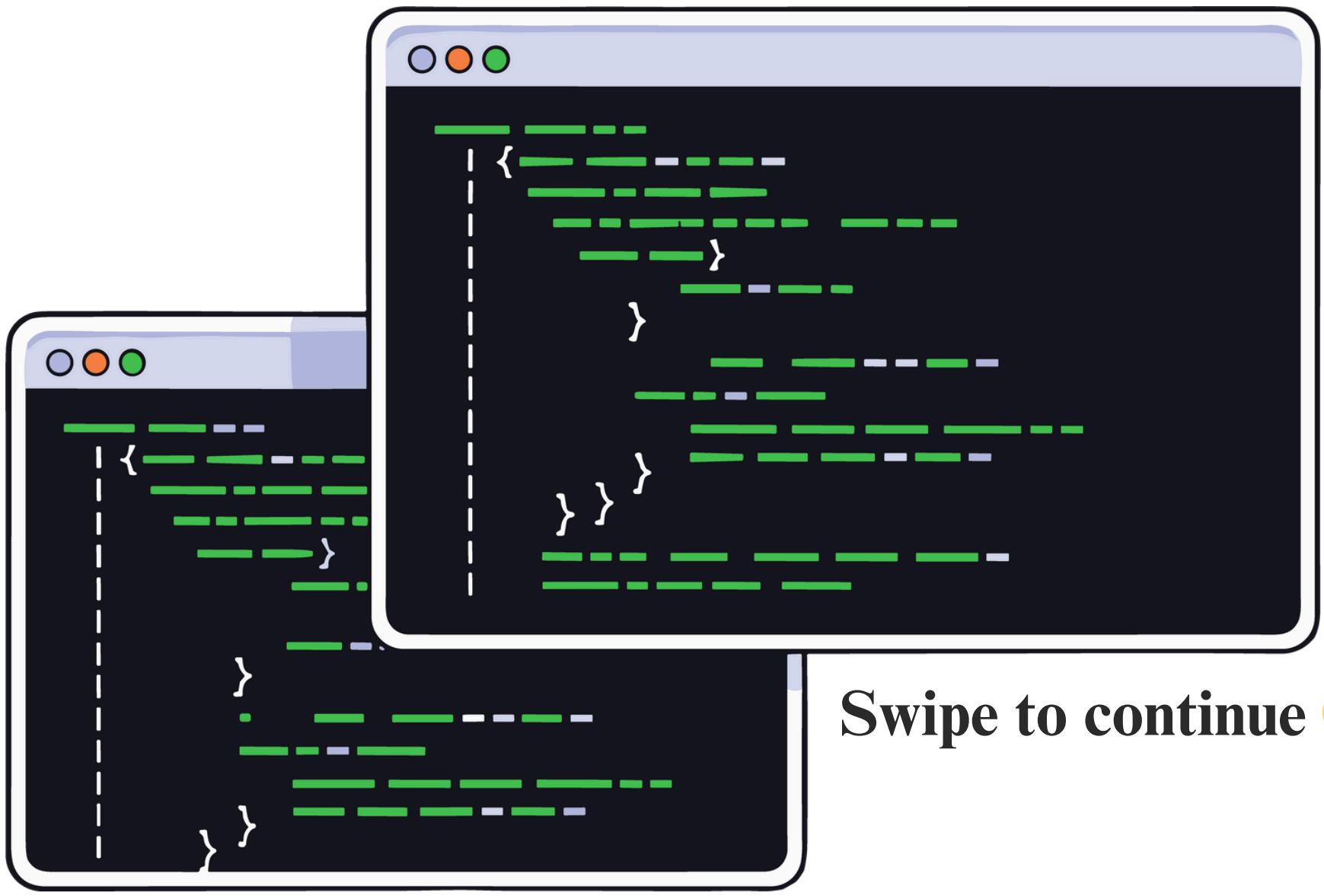
SINGLETON

PATTERN



Swipe to continue 👉

- Only one **instance** of a class exists
- That instance is **globally** accessible

by Shubam Gupta

1. What is a Singleton?

A Singleton is a **design pattern** that ensures:

- Only one **instance** of a class exists
- That instance is **globally accessible**

2. Why do we need Singleton?

Singleton is useful when:

- The same data or service is needed **across the app**
- Creating **multiple instances** may cause inconsistency

3. Common use cases in iOS:

- Network Manager
- API Service
- App Configuration
- User Session
- Database / Cache Manager

4. Key Rules of a Singleton

To create a Singleton, we must:

- *Restrict object creation*
- *Provide a single shared instance*
- *Prevent external initialization*

5. Basic Singleton Implementation in Swift

```swift
final class NetworkManager {

    // Step 1: Shared instance
    static let shared = NetworkManager()

    // Step 2: Private initializer
    private init() { }

    func fetchData() {
        print("Fetching data from API")
    }
}
```

6. How to Use It?

```swift
NetworkManager.shared.fetchData()
```

NOTE:

- No need to create an object
- Same instance used throughout the app

7. Why static let is Important?

- **static let** is thread-safe in Swift
- Instance is created only once
- Lazy-loaded (created when first accessed)

👉 **Swift handles synchronization internally** 💯

8. Advantages of Singleton

✅ Single source of truth

✅ Easy global access

✅ Memory efficient

✅ Thread-safe (with static let)

9. Disadvantages (Important for Interviews)

❌ Global state (harder to track changes)

❌ Difficult to unit test

❌ Tight coupling

❌ Can be overused if not careful

👉 **Use Singleton only when it truly makes sense**

10. One-Line Summary 📌

Singleton ensures one **shared instance of a class** across the entire iOS app using **static let** and a **private init()**.

"Singleton is not bad,

overusing Singleton is bad."

Thank you

by Shubam Gupta