Coding Exercise

Part 1

The code below is more than suboptimal in terms of testability, modernity of the employed concepts, and API. Worse than that, the code is even incorrect when being wrongly used.

Please rewrite the code such that it becomes **correct**, **fully testable**, **thread-safe**, and **easy to use** and **reuse**. In particular, your code should not rely on antiquated concepts such as `NotificationCenter` but leverage modern concepts such as `async`/`await`, `actor`s, and structured concurrency, where applicable.

After rewriting the code, the code snippet below should be usable to run the code you wrote. There is no need to provide tests for your code but please write your code in a way in which the following test cases could easily be implemented:

- If no network is available, the given closure is not invoked
- If the network is initially available, the given closure is invoked
- If the network is initially not available but becomes available within the given timeout duration, the given closure is invoked
- If the network is initially not available and becomes available only after the given timeout duration, the given closure is not invoked

Problematic code

```
import Foundation
import Network

public class NetworkOperationPerformer {
  private let networkMonitor: NetworkMonitor
  private var timer: Timer?
  private var closure: (() -> Void)?

  public init() {
    self.networkMonitor = NetworkMonitor()
  }

  /// Attempts to perform a network operation using the given `closure`, within the given `timeoutDuration`.
  /// If the network is not accessible within the given `timeoutDuration`, the operation is not performed.
  public func performNetworkOperation(
    using closure: @escaping () -> Void,
    withinSeconds timeoutDuration: TimeInterval
  ) {
    self.closure = closure

    if self.networkMonitor.hasInternetConnection() {
      closure()
    } else {
      tryPerformingNetworkOperation(withinSeconds: timeoutDuration)
    }
  }

  private func tryPerformingNetworkOperation(withinSeconds timeoutDuration: TimeInterval) {
    NotificationCenter.default.addObserver(
      self,
      selector: #selector(networkStatusDidChange(_:)),
      name: .networkStatusDidChange,
      object: nil
    )

    self.timer = Timer.scheduledTimer(withTimeInterval: timeoutDuration, repeats: false) { _ in
      self.closure = nil
      self.timer = nil

      NotificationCenter.default.removeObserver(self)
    }
  }
```

```
  @objc func networkStatusDidChange(_ notification: Notification) {
    guard
      let connected = notification.userInfo?["connected"] as? Bool,
      connected,
      let closure
    else {
      return
    }

    closure()
  }
}

private class NetworkMonitor {
  private let monitor = NWPathMonitor()

  init() {
    startMonitoring()
  }

  private func startMonitoring() {
    monitor.pathUpdateHandler = { _ in
      NotificationCenter.default.post(
        name: .networkStatusDidChange,
        object: nil,
        userInfo: ["connected": self.hasInternetConnection()]
      )
    }

    monitor.start(queue: DispatchQueue(label: "NetworkMonitor"))
  }

  func hasInternetConnection() -> Bool {
    return monitor.currentPath.status == .satisfied
  }
}

private extension Notification.Name {
  static let networkStatusDidChange = Notification.Name("NetworkStatusDidChange")
}
```

Snippet to use for running your code

```
let networkOperationClosure: () async -> SomeType = {
  // Long-lasting network operation.
  return result
}

let result = await NetworkOperationPerformer().perform(withinSeconds: 3) {
  return await networkOperationClosure()
}
```

Bonus

- Describe the problems existing in the provided problematic code
- Improve your code such that is supports cancellation. Make sure to reason about which cancellation behavior is most desirable for the `perform` method.
- Add proper documentation to the `public` method of your code

Part 2

In this exercise, we ask you to build a simple `SwiftUI`-based app showing two screens, a loading screen and subsequently a screen depicting an image. As part of this exercise, you will first need to solve the “Network Operation” exercise and then use the written code as part of this exercise.

The behavior of the desired app is as follows:

- Upon app start, the discussed loading screen is displayed, depicting a spinning wheel on top of a background which is black when dark mode is on and white, otherwise. If the internet is not accessible within half a second after starting to display the loading screen, an additional text is displayed on the loading screen, indicating that there is currently no network connection.
- While the loading screen is being displayed, an image download is performed in the background. You should trigger the image download operation within the `perform` method of the `NetworkOperationPerformer` you wrote as part of “Network Operation” exercise, using a timeout duration of 2 seconds.
- After the image was downloaded or once the timeout duration has been reached without successful image download (whatever happens first), the second screen should be displayed. If the image has been downloaded successfully, it should be displayed on the second screen. Otherwise, a text stating that the download failed should be displayed.

Points to Consider

- Think about how you want the app to manage its state
- Make sure to separate logic and UI such that writing integration tests for the app logic is feasible, testing the entire control flow, including everything from the state changes which are responsible for UI changes to the successful or unsuccessful download of the image

Explanatory Comments

- The loading screen is shown for 5 seconds to allow for enough time to visually test the behavior. Obviously the loading screen would only be shown as long as necessary in a real world scenario.