Concurrency

Juggling Tasks with Elegance and Efficiency

What is Concurrency?

1. Managing multiple tasks over overlapping time periods
2. True parallelism: Simultaneous execution on multiple cores
3. Interleaved execution: Rapid switching between tasks on a single core
4. Dividing a program into independently executing components

Why is Concurrency needed?

1. Improve app responsiveness
2. Utilize multi-core processors effectively
3. Handle I/O-bound and CPU-bound tasks efficiently
4. Support modern app architectures and user expectations

Concurrency Solutions in iOS

1. Grand Central Dispatch (GCD)
2. Operation and OperationQueue
3. Async/Await
4. Thread and NSThread

Grand Central Dispatch (GCD)

1. Low-level C-based API
2. Manages dispatch queues
3. Automatic thread pool management
4. Efficient work distribution

Async/Await

1. Modern Swift feature (Swift 5.5+)
2. Simplifies asynchronous code
3. Uses 'async' and 'await' keywords
4. Integrates with GCD

MainActor

1. Swift 5.5+ feature
2. Ensures code runs on the main thread
3. Helps prevent data races
4. Integrates with Async/Await

GCD Queues

1. Serial Queues
2. Concurrent Queues
3. Main Queue
4. Global Queues tasks in parallel

```
// Serial Queue
let serialQueue = DispatchQueue(label:
    "com.app.serialQueue")

// Concurrent Queue
let concurrentQueue = DispatchQueue(label:
    "com.app.concurrentQueue", attributes: .concurrent)

// Main Queue
DispatchQueue.main.async { /* UI updates */ }

// Global Queue
DispatchQueue.global(qos: .userInitiated).async { /*
    Background work */ }
```

Operation Queues

1. Manages Operation objects
2. Allows dependencies between operations
3. Supports cancellation and prioritization

```
let queue = OperationQueue()
queue.maxConcurrentOperationCount = 3

let op1 = BlockOperation { print("Operation 1") }
let op2 = BlockOperation { print("Operation 2") }
op2.addDependency(op1)

queue.addOperations([op1, op2], waitUntilFinished: false)

// Custom Operation
class MyOperation: Operation {
    override func main() {
        // Perform work here
    }
}
```

Quality of Service (QoS)

QoS Levels (Highest to Lowest Priority):

1. .userInteractive
2. .userInitiated
3. .default
4. .utility
5. .background

```
//Sample Code for GCD:
let queue = DispatchQueue(label: "com.app.queue", qos:
    .userInitiated)

DispatchQueue.global(qos: .utility).async {
    // Perform long-running task
}

//Sample Code for OperationQueue:
let operation = BlockOperation { /* work */ }
operation.qualityOfService = .userInitiated

let operationQueue = OperationQueue()
operationQueue.qualityOfService = .utility
```

Operation States

1. Ready: Initial state, operation can begin
2. Executing: Operation is currently running
3. Finished: Operation has completed
4. Cancelled: Operation was cancelled before completion

Operation Dependencies

1. Define execution order of operations
2. Use addDependency(_:) method
3. Create complex workflows
4. Helps in managing task relationships

```
func fetchData() {
    let operationQueue = OperationQueue()

    let reachabilityOp = ReachabilityOperation(queue: operationQueue, delegate: self)
    let networkOp = NetworkOperation(queue: operationQueue, urlString: urlString,
        delegate: self)
    jsonParsingOp = JSONParsingOperation(queue: operationQueue, delegate: self)

    // set the depedancy
    networkOp.addDependency(reachabilityOp)
    jsonParsingOp!.addDependency(networkOp)

    let operations:[Operation] = [reachabilityOp, networkOp, jsonParsingOp!]

    operationQueue.addOperations(operations, waitUntilFinished: false)
}
```

Cancelling Operations

1. Use cancel() method on Operation
2. Check isCancelled in operation code
3. Allows graceful termination of tasks
4. Important for responsive UI and resource management

Concurrency Approaches

Grand Central Dispatch (GCD):

- Simple API for common use cases
- Automatic thread management
- Manual dependency and synchronization management
- Complex error handling with nested closures

Async/Await:

- Syntactic clarity (reads like synchronous code)
- Native error handling and propagation
- Automatic cancellation propagation
- Structured concurrency
- Requires iOS 15+ for full support

Operation Queues:

- Object-oriented approach
- Built-in support for dependencies
- Cancellation support
- More verbose for simple tasks
- Higher overhead compared to GCD