

PG5600

iOS programming

Forelesning 7

Sist gang

- Viewkonsepter
- Å instansiere views
- Å lage custom views
- Eventhåndtering
- Gestures
- Animasjoner

Agenda

- Debugging
- Testing
- Swift og gjenbruk av kode
 - Rammeverk
 - Cocoapods & Carthage
- Tråder og asynkronitet
- Snakke med internett
- try & json

Sørg for god informasjon på forhånd

- Bruk logging
- Unit tests
- Assertions (ikke så vanlig lenger)

Debugging

Breakpoint Logging

579

```
}
```

580

```
}
```

581

582

```
/**
```

583

```
Called when
```

584

```
*/
```

585

```
private func
```

586

```
let appD
```

☒ BaseTestCase.swift:58

Condition

Ignore



times before stopping

Action

Debugger Command



po person

Options



Automatically continue after evaluating actions

Logging

```
// Til console i XCode  
print("Logg en linje")
```

```
// Til console på device  
NSLog("Logg objekter")
```


Logging med swell

<https://github.com/hubertr/Swell>

```
class ContactService {  
    let logger = Swell.getLogger("ContactService")  
  
    func getContact(name: String) {  
        Swell.info("Retrieving contact for \$(name)")  
        ...  
  
        //named logger  
        logger.debug("Retrieving contact for \$(name)")  
  
        //complex logger  
        logger.trace {  
            let city = getCityFor(name)  
            return "Retrieving contact for \$(name) of \$(city)"  
        }  
    }  
}  
  
//Swell.plist for å konfigurere
```

Debugging og informasjon

Unit tests

Apple sitt XCTest kan brukes til å skrive tester

```
import XCTest
import SwiftFonts

class FontSorterTests: XCTestCase {

    let sorter = FontSorter()

    func testCompareHyphenWithNoHyphen() {
        let fonts = ["Arial-ItalicMT", "ArialMT"]
        let expected = ["ArialMT", "Arial-ItalicMT"]
        let sorted = sorter.sortFontNames(fonts)
        XCTAssertEqual(expected[0], sorted[0], "the array should be sorted properly")
        XCTAssertEqual(expected[1], sorted[1], "the array should be sorted properly")
    }

    func testCompareHyphenWithHyphen() {
        let fonts = ["Avenir-Roman", "Avenir-Oblique"]
        let expected = ["Avenir-Oblique", "Avenir-Roman"]
        let sorted = sorter.sortFontNames(fonts)
        XCTAssertEqual(expected[0], sorted[0], "when two fonts contain a hyphen, they should be sorted alphabetically")
        XCTAssertEqual(expected[1], sorted[1], "when two fonts contain a hyphen, they should be sorted alphabetically")
    }
}
```

Viktige test assertions, det finnes flere

```
XCTAssert(expression, format...) // hvis expression = true, så er testen ok
```

```
XCTAssertTrue(expression, format...) // lik som den over
```

```
XCTAssertFalse(expression, format...) // hvis false så er testen ok
```

```
XCTAssertEqual(expression1, expression2, format...) // lik så er testen ok
```

```
XCTAssertNotEqual(expression1, expression2, format...) // ulike så er testen ok
```

```
XCTAssertEqualWithAccuracy(expression1, expression2, accuracy, format...) // kan brukes på nummer som ikke må være helt lik
```

```
XCTAssertNotEqualWithAccuracy(expression1, expression2, accuracy, format...) // kan brukes på nummer som ikke må være helt lik
```

```
CTAssertNil(expression, format...) // teste optionals
```

```
XCTAssertNotNil(expression, format...) // teste optionals
```

Async testing

```
func testAsynchronousURLConnection() {
    let URL = "http://mobile-course.herokuapp.com/message"
    let expectation = expectationWithDescription("GET \(URL)")

    let session = NSURLSession.sharedSession()
    let task = session.dataTaskWithURL(NSURL(string: URL), completionHandler: {(data, response, error) in
        expectation.fulfill()

        XCTAssertNotNil(data, "data should not be nil")
        XCTAssertNil(error, "error should be nil")

        if let HTTPResponse = response as? NSHTTPURLResponse {
            XCTAssertEqual(HTTPResponse.URL!.absoluteString!, URL, "HTTP response URL should be equal to original URL")
            XCTAssertEqual(HTTPResponse.statusCode, 200, "HTTP response status code should be 200")
            XCTAssertEqual(HTTPResponse.MIMETYPE as String!, "application/json", "HTTP response content type should be text/html")
        } else {
            XCTFail("Response was not NSHTTPURLResponse")
        }
    })

    task.resume()

    waitForExpectationsWithTimeout(task.originalRequest.timeoutInterval, handler: { error in
        task.cancel()
    })
}
```

Performance testing

```
func testPerformanceExample() {  
    // Tester performance med self.measureBlock  
    self.measureBlock() {  
        // Her puttes koden du ønsker å teste tiden på  
    }  
}
```

Debugging og informasjon

Assertion i kode

- Optionals lar oss sjekke om verdien ikke eksiterer, derav skrive kode som ikke feiler eller knekker appen
- Noen ganger så kan det skje at man ikke kan gå videre hvis verdier ikke finnes eller verdier har feil verdi
- I slike situasjoner kan man bruke assertions for å gi en feilmelding til utvikleren slik at det blir lettere å debugge
- En assertion sjekker om noe er sant og hvis <false> så avsluttes applikasjonen

```
let age = 12  
assert(age >= 13, "Personen må være 13 eller eldre")
```

Når skal man bruke assertions?

- Bruk det når noe kan være feil, men koden din er avhengig av at det alltid er riktig. Eksempler:
 - Når subscript index er utenfor mulige verdier
 - Når en verdi er sendt inn til en funksjon, men funksjonen kan ikke utføre oppgaven, da verdien er umulig å bruke
 - Når en optional må være satt for at koden skal kunne kjøre

Debugging

View

Trykk pause når applikasjonen din kjører

Test

PID 61254, Paused

CPU0%

Memory19.8 MB

DiskZero KB/s

NetworkZero KB/s

Thread 1

Queue: com.apple.main-thread (serial)

0 mach_msg_trap

6 UIApplicationMain

7 top_level_code

8 main

9 start

10 start

Thread 2

Queue: com.apple.libdispatch-mana...

Thread 3

Thread 4

Thread 5

Thread 6

Test > Thread 1 > 0 mach_msg_trap

libsystem_kernel.dylib`mach_msg_trap:
0x10429ca10: movq %rcx, %r10
0x10429ca13: movl \$0x100001f, %eax
0x10429ca18: syscall
0x10429ca1a: retq
0x10429ca1b: nop

Thread 1: signal SIGSTOP

Test > Thread 1 > 0 mach_msg_trap

(lldb)

Quick Help

No Quick Help

View Controller – A controller that supports the fundamental view-management model in iOS.

Navigation Controller – A controller that manages navigation through a hierarchy of views.

Table View Controller – A controller that manages a table view.

Debugging

Playground

Bruk playground til å debugge kodesnutter

Debugging Demo

Breakpoints

Swift og gjenbruk av kode

Det finnes tre måter å dele kode på i Swift

- Importer filene inn til ditt prosjekt direkte
- Cocoa Touch Static Library
- Cocoa Touch Framework (nytt og bedre)

Pakke-manager:

- CocoaPods - [Cocoapods.org](https://cocoapods.org)
- Carthage - Mindre intrusive for prosjekt men mer jobb

Cocoa Touch Frameworks

- Tilgjengelig fra og med Xcode 6
- Brukes i forbindelse med:
 - Extensions (brukes i Extensions og Apple Watch)
 - Lage gjenbrukbare moduler/rammeverk på tvers av prosjekter

Asynkronitet

- Hovedtråden er den som man vanligvis er på og som tegner GUI
- Andre tråder brukes når man ønsker å gjøre tyngre jobber som ikke skal blokkere GUI
- Vi har tre måter å lage tråder på:
 - NSThread
 - Grand Central Dispatch
 - NSOperationQueue

NSThread

- Lite brukt i iOS verden, men kjekk å vite om
- Krever manuell håndtering
- Apple anbefaler å bruke GCD eller NSOperationQueues

```
// Lag en ny tråd
```

```
NSThread.detachNewThreadSelector("someMethod", toTarget: self, withObject: nil)
```

```
var thread = NSThread(target: self, selector: "testMethod", object: nil)
```

```
thread.start()
```

```
thread.cancel()
```

```
thread.isMainThread
```

Grand Central Dispatch

- Håndterer trådene for deg
- Baserer seg på køer med oppgaver
- Det finnes to typer køer
 1. Serial – En oppgave av gangen
 2. Concurrent – Kan utføre flere oppgaver samtidig

Bruk køene til å utføre oppgaver

Asynkrone funksjoner

- `dispatch_async`
- `dispatch_after`
- `dispatch_apply`

Synkrone funksjoner

- `dispatch_once`
- `dispatch_sync`

Bruker disse mest

```
// asynkron jobb så tilbake på main
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    // do some task
    image = generateComplexImage()
    dispatch_async(dispatch_get_main_queue(), ^{
        self.view.addSubview(UIImageView(image))
    });
});

// Delay

let delayTime = dispatch_time(DISPATCH_TIME_NOW, Int64(1 * Double(NSEC_PER_SEC)))
dispatch_after(delayTime, dispatch_get_main_queue()) {
    print("After 1 second")
}
```

GCD og Async

<https://github.com/duemunk/async>

- En abstraksjon av Grand Central Dispatch api'et
- Lettere syntak, mer man bør kunne GCD fra før av
- Mindre verbost


```
Async.background {  
    print("Kjører på bakgrunnskøen")  
}.main {  
    print("Kjører på hovedtråden etter bakgrunnsjobben er ferdig")  
}
```

// i stedet for

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_BACKGROUND, 0), {  
    print("Kjører på bakgrunnskøen")  
  
    dispatch_async(dispatch_get_main_queue(), 0), {  
        print("Kjører på hovedtråden etter bakgrunnsjobben er ferdig")  
    })  
})
```

NSOperation og NSOperationQueue

NSOperation

- En enhet av arbeid
- En abstrakt klasse som man arver fra

Alternativer

NSBlockOperation - Lag en closure som du ønsker å gjøre i bakgrunn

NSInvocationOperation - Kjører en metode i bakgrunn

For å starte NSOperation

```
var operation = NSOperation()  
operation.start()
```

```
operation.cancel()
```

eller legg det i en kø

NSOperationQueue

- Håndterer kjøringen av et sett med NSOperation, NSBlockOperation eller NSInvocationOperation
- First-In-First-Out med mindre man prioriterer oppgavene eksplisitt
- Man kan bestemme maks antall samtidige jobber med `maxConcurrentOperationCount`
- Bruker Grand Central Dispatch i bakgrunnen

- QOS_CLASS_USER_INTERACTIVE =
NSQualityOfServiceUserInteractive
- QOS_CLASS_USER_INITIATED =
NSQualityOfServiceUserInitiated
- QOS_CLASS_UTILITY = NSQualityOfServiceUtility
- QOS_CLASS_BACKGROUND = NSQualityOfServiceBackground

```
let backgroundOperation = NSOperation()  
backgroundOperation.qualityOfService = .Background
```

```
let operationQueue = NSOperationQueue()  
operationQueue.addOperation(backgroundOperation)
```

NSThread vs GCD vs NSOperationQueue

- NSThread når du trenger full kontroll over trådene du lager
- GCD når du trenger enkel parallellisering (kast denne jobben inn i en bakgrunnstråd)
- NSOperationQueue når du har mer komplekse jobber du vil parallelisere

Snakke med internett

Http metoder

GET - Hente ned data

POST - Sende ny data

PUT - Oppdatere all eksisterende data

PATCH - Oppdatere eksisterende data med bare noen felter

DELETE - Slette data

```
let url = NSURL(string: "http://mobile-course.herokuapp.com/message")
let session = NSURLSession.sharedSession()
let task = session.dataTaskWithURL(url, completionHandler: { (data, response, error) -> Void in
    print(data)
})
```

```
task.resume()
```

```
let url2 = NSURL(string: "http://mobile-course.herokuapp.com/message")
let request = NSMutableURLRequest(URL: url2)
request.HTTPMethod = "POST"
let session2 = NSURLSession.sharedSession()
let task2 = session2.dataTaskWithRequest(request, completionHandler: { (data, response, error) -> Void in
    print(data)
})
```

```
task2.resume()
```

Playground og Nettverk

For å kjøre asynkron kode i playground må man gjøre følgende

```
import XCPplayground  
XCPSetExecutionShouldContinueIndefinitely()
```

**[https://github.com/Alamofire/
Alamofire](https://github.com/Alamofire/Alamofire)**

Alamofire og REST

```
Alamofire.request(.GET, "http://jsonplaceholder.typicode.com/posts")
    .responseJSON { ( response) -> Void in

        if let responseJSONArray = response.result.value as? [[String : AnyObject]] {

            for post in responseJSONArray {
                print(post)
            }
        }

        if let responseError = response.result.error {
            print(response.result.error)
        }
    }
```

**[https://github.com/Alamofire/
Alamofire](https://github.com/Alamofire/Alamofire)**

try

```
do {  
    try expression  
    statements  
} catch pattern 1 {  
    statements  
} catch pattern 2 where condition {  
    statements  
}
```



```
func testStuff() {  
    do {  
        try login()  
    } catch LoginError.NoUserName {  
        print("wrong username")  
    } catch LoginError.WrongPassword {  
        print("wrong password")  
    } catch {  
        print("some other error")  
    }  
}
```

Try

```
enum LoginError: ErrorType {  
    case NoUserName  
    case WrongPassword  
}  
  
func login() throws {  
    defer {  
        print("an error happened")  
    }  
  
    let userText : String? = "John Snow"  
    let passWordIsCorrect = false  
  
    guard let actualUserName = userText else {  
        throw LoginError.NoUserName  
    }  
  
    guard passWordIsCorrect else {  
        throw LoginError.WrongPassword  
    }  
}
```

Try!

```
// krasjer hvis throws error  
try! login()
```

JSON

Swift 3

```
func titlesFromJSON(data: NSData) -> [String] {
    var titles = [String]()

    do {
        let jsonDictionary = try NSJSONSerialization.JSONObjectWithData(data, options: nil, error: &jsonError) as? [String : AnyObject] {
            guard let feed = jsonDictionary["feed"] as? [String : AnyObject] else {
                // throw ?
            }

        } catch let error as NSError {
            // Do something with error
        }

        return titles
    }
}
```

SwiftyJSON

Som du så i det første eksempelet, så er Swift nøye med typer

— SwiftyJSON prøver å hjelpe oss med akkurat dette

<https://github.com/SwiftyJSON/SwiftyJSON>

Eksempelvis denne json strukturen

```
{ "name" : "Matrix",  
  "genre" : "Sci-fi",  
  "year" : 2003,  
  "rating" : 9.8  
}
```

Hente ut navn med vanlig Swift kode

```
struct Movie {
    let name: String
    let genre: String?
    let year : Int
    let rating: Double

    init?(attributes: [String : Any]) {
        guard let name = attributes["name"] as? String, let year = attributes["year"] as? Int, let rating = attributes["rating"] as? Double else {
            return nil
        }
        self.name = name
        self.genre = attributes["genre"] as? String
        self.year = year
        self.rating = rating
    }
}

let jsonDict = NSJSONSerialization.JSONObjectWithData(data,
    options: NSJSONReadingOptions.MutableContainers, error: nil) as? [String : Any]

Movie(attributes: jsonDict)
```

Swift 4

```
struct Movie : Decodable{
    let name: String
    let genre: String?
    let year : Int
    let rating: Double

    enum Keys: String, CodingKey {
        case name
        case genre
        case year
        case rating
    }

    public init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: Keys.self)
        self.name = try container.decode(String.self, forKey: Keys.name)
        self.rating = try container.decode(Double.self, forKey: Keys.rating)
        self.year = try container.decode(Int.self, forKey: Keys.year)
        self.genre = try container.decode(String?.self, forKey: Keys.genre)
    }
}
```



```
let json =  
"""  
{ "name" : "Matrix",  
  "genre" : "Sci-fi",  
  "year" : 2003,  
  "rating" : 9.8  
}  
"""
```

```
let movie = try JSONDecoder().decode(Movie.self, from: json.data(using: String.Encoding.utf8)!)
```

Videre lesning

- <https://github.com/ochococo/Design-Patterns-In-Swift>
- <http://www.raywenderlich.com/79149/grand-central-dispatch-tutorial-swift-part-1>
- Error handling i boka
- Basics i iOS-boka
- Cocoapods.org

Oppgaver

Se GitHub