

PG5600

iOS programming

Forelesning 7

Først: Adaptive layouts

<http://mathewsanders.com/designing-adaptive-layouts-for-iphone-6-plus/>

Sist gang

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

Agenda

- Debugging
- Swift og gjenbruk av kode
 - Bundles
 - Rammeverk
- Tråder og asynkronitet
- Snakke med internett
- Cloud Kit

Debugging

Sørg for god informasjon på forhånd

- Bruk logging
- Unit tests
- Assertions

Debugging og informasjon

Logging

```
println("Logg en linje")
```


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

Snakket om dette i forelesning 5

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:

- Cocoa Pods, bruker static libraries enn så lenge og har mest Objective-C libs

Cocoa Touch Frameworks

- Tilgjengelig fra og med Xcode 6
- Brukes i forbindelse med:
 - Live views (som dere snakket om sist)
 - Extensions (snakkes om i senere forelesninger)
 - Lage gjenbrukbare moduler/rammeverk på tvers av prosjekter

Runtime deling av rammeverk

- Rammeverk blir delt på tvers av applikasjoner runtime
- Som betyr at versjonering blir viktig

Gjenbruk av kode demo

Cocoa Touch Frameworks

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

Pre-definerte køer

QOS_CLASS_USER_INTERACTIVE - Main Queue

QOS_CLASS_USER_INITIATED - Høy prioritet -
Concurrent

QOS_CLASS_UTILITY - Lav prioritet - Concurrent

QOS_CLASS_BACKGROUND - Lavere prioritet - Concurrent

```
// vanligvis

dispatch_queue_t aQueue = dispatch_get_global_queue(QOS_CLASS_BACKGROUND, 0);

// bug i swift 1.0

extension qos_class_t {

    public var id:Int {
        return Int(self.value)
    }

}

let aQueue = dispatch_get_global_queue(QOS_CLASS_BACKGROUND.id, 0)
```

QOS_CLASS_USER_INTERACTIVE

- UI tråden
- Brukes ved event håndtering
- Tegning av UI

QOS_CLASS_USER_INITIATED

- UI som har satt i gang
- Men er asynkron for UI
- Brukeren venter på resultat
- Kreves for å fortsette brukerinteraksjon

QOS_CLASS_UTILITY

- Langtkjørende, men brukeren skal kunne se at noe skjer
- Kalkuleringer, I/O, nettverk
- Vil oppdatere data i UI
- Energi effektiv

QOS_CLASS_BACKGROUND

- Brukeren har ikke peiling på hva som skjer her
- Hente data på forhånd
- Cache oppdateringer

Når skal du bruke hvilke?

- `QOS_CLASS_USER_INTERACTIVE` - Handler arbeidet om å oppdatere UI?
- `QOS_CLASS_USER_INITIATED` - Må man gjøre dette arbeidet for at brukeren skal kunne forsette?
- `QOS_CLASS_UTILITY` - Er brukeren klar over at dette arbeidet går fremover?
- `QOS_CLASS_BACKGROUND` - Kan dette arbeidet gjøres en annen gang, når brukeren ikke er aktiv?

Det finnes flere default køer også, men vi har gått igjennom de som er mest brukt

Lag dine egne køer

```
dispatch_queue_t newQueue =  
    dispatch_queue_create("com.my.utility", DISPATCH_QUEUE_CONCURRENT)
```

```
// FIFO
```

```
dispatch_queue_t newQueue =  
    dispatch_queue_create("com.my.utility", DISPATCH_QUEUE_SERIAL)
```

Bruk køene til å utføre oppgaver

Asynkrone funksjoner

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

Synkrone funksjoner

- `dispatch_once`
- `dispatch_sync`

Bruker disse mest

```
// Oppdater noe på main køer for å oppdatere UI
dispatch_async(dispatch_get_main_queue(), ui_update_closure);
```

```
// legg til en closure til en kø om 2 sekunder
dispatch_after(dispatch_time(DISPATCH_TIME_NOW, 2 * NSEC_PER_SEC), some_queue, check_for_updates_closure);
```

```
// legg til arbeid på køen N ganger
dispatch_apply(i, some_queue, work_unit_closure);
```

```
// gjør en oppgave kun en gang
static dispatch_once_t onceToken = 0; // Må være static
dispatch_once(&onceToken, only_once_closure);
```

```
// legg til noe på en bakgrunnskø og vent til den er ferdig
dispatch_sync(background_queue, blocking_closure);
```

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

// i stedet for

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_BACKGROUND, 0), {  
    println("Kjører på bakgrunnskøen")  
  
    dispatch_async(dispatch_get_main_queue(), 0), {  
        println("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
    println(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 = session.dataTaskWithRequest(request, completionHandler: { (data, response, error) -> Void in
    println(data)
})
```

```
task2.resume()
```

Playground og Nettverk

For å kjøre asynkron kode i playground må man gjøre følgende

```
import XCPPlayground  
XCPSetExecutionShouldContinueIndefinitely()
```


Alamofire og REST

```
Alamofire.request(.GET, "http://mobile-course.herokuapp.com/message")
    .response { (request, response, data, error) in
        println(request)
        println(response)
        println(error)
    }
```

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

Starscream og WebSocket

```
import Starscream

var socket = WebSocket(url: NSURL(scheme: "ws", host: "localhost:8080", path: "/"))
socket.delegate = self
socket.connect()

// delegate

func websocketDidConnect()
func websocketDidDisconnect(error: NSError?)
func websocketDidWriteError(error: NSError?)
func websocketDidReceiveMessage(text: String)
func websocketDidReceiveData(data: NSData)

// skriv data

self.socket.writeData(data)
self.socket.writeString("Hi Server!")
```

<https://github.com/daltoniam/starscream>

JSON

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

    if let json = NSJSONSerialization.JSONObjectWithData(data, options: nil, error: &jsonError) as? NSDictionary {
        if let feed = json["feed"] as? NSDictionary {
            if let entries = feed["entry"] as? NSArray {
                for entry in entries {
                    if let name = entry["im:name"] as? NSDictionary {
                        if let label = name["label"] as? String {
                            titles.append(label)
                        }
                    }
                }
            }
        }
    }
    else {
        if let unwrappedError = jsonError {
            println("json error: \(unwrappedError)")
        }
    }

    return titles
}
```

Litt usikkert alternativ

```
var jsonError: NSError?  
let json = NSJSONSerialization.JSONObjectWithData(data, options: nil, error: &jsonError) as NSDictionary  
  
if let unwrappedError = jsonError {  
    println("json error: \(unwrappedError)")  
} else {  
    self.titles = json.valueForKeyPath("feed.entry.im:name.label") as [String]  
}
```

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

```
[
  {
    .....
    "text": "just another test",
    .....
    "user": {
      "name": "OAuth Dancer",
      "favourites_count": 7,
      "entities": {
        "url": {
          "urls": [
            {
              "expanded_url": null,
              "url": "http://bit.ly/oauth-dancer",
              "indices": [
                0,
                26
              ],
              "display_url": null
            }
          ]
        }
      }
    },
    .....
    "in_reply_to_screen_name": null,
  },
  .....]
```

Hente ut navn med vanlig Swift kode

```
let jsonObject : AnyObject! = NSJSONSerialization.JSONObjectWithData(dataFromTwitter,
    options: NSJSONReadingOptions.MutableContainers, error: nil)

if let statusesArray = jsonObject as? NSArray{
    if let aStatus = statusesArray[0] as? NSDictionary{
        if let user = aStatus["user"] as? NSDictionary{
            if let userName = user["name"] as? NSDictionary{

                println(userName)

            }
        }
    }
}
```

Hente ut kode med SwiftyJSON

```
let json = JSON(data: dataFromNetworking)
if let userName = json[0]["user"]["name"].string {
    println(userName)
}
```


CloudKit

- Ektern lagring for apps
- Ganske rimelig når det gjelder pris
- Har et web grensesnitt for å se og håndtere data
- Inneholder et set med API for å overføre data mellom iCloud og device

Kan brukes om du lager en app som bare skal fungere på iOS og kun trenger lagring og sync av data mellom enheter

Hvordan ser det ut?

- Man har en kontainer: `CKContainer`
- En kontainer består av en `public` og `private`:
`CKDatabase`
- `Public` er tilgjengelig for alle iCloud brukere, mens `privat` er kun for en bruker
- Man kan lytte på `notifications` på data endringer i iCloud og reagere på disse

CKRecord

CKRecord er en slags dictionary som er raden i databasen, en CKRecord kan inneholde

- NSString
- NSNumber
- NSData
- NSDate
- CLLocation
- NSArray<T>
- CKReference
- CKAsset

CKReference - Knytt en annen child record til denne recorden, har cascade delete

CKAsset - Lar deg lagre en blob av noe som er knyttet til en record

CKRecord kan maks være 1 MB i størrelse

Lag en CKRecord og lagre den

```
var publicDB = CKContainer.defaultContainer().publicCloudDatabase

var record = CKRecord(recordType: "channelType")
record.setValue("NRK", forKey: "channelName")

publicDB.saveRecord(record, completionHandler: { (record, error) -> Void in
    println("Record er lagret")
})
```

Fetching

```
var publicDB = CKContainer.defaultContainer().publicCloudDatabase

var predicate = NSPredicate()
var query = CKQuery(recordType: "channelType", predicate: predicate)

publicDB.performQuery(query, inZoneWithID: nil) { (result, error) -> Void in
    println("Hentet ned alle channelTypes")
}
```

Deleting

```
let predicateDelete = NSPredicate(format: "channelName = %@", "NRK")
let queryDelete = CKQuery(recordType: "channelType", predicate: predicateDelete)
publicDB.performQuery(query, inZoneWithID: nil) { (result, error) -> Void in
    for record in result {
        publicDB.deleteRecordWithID(record.recordID, completionHandler: { (deletedId, error) -> Void in
            println("\(deletedId)")
        })
    }
}
```


**Dette var bare en forsmak på CloudKit
og det finnes alternativer.**

Facebook sitt Parse er en av dem

Nyttig lenker

- **Swift design patterns:**

<https://github.com/ochococo/Design-Patterns-In-Swift>

Smidig 2014

- Sitte vakt av og til
- Gratis inngang
- Lære mer om smidig metoder
- Middag og billig øl
- Ta kontakt med: frivillig@smidig.no

Oppgaver

Se Øvingsoppgavene

<https://github.com/hinderberg/ios-swift-kurs>