PG5600

iOS programmering

Forelesning 10 - Persistering

Agenda

- Hvor filer lagres
- Enkel lesing og skriving til disk
- NSUserDefaults
- NSKeyedArchiver / NSKeyedUnarchiver
- Core Data

Hvor filer lagres

```
let fm = NSFileManager.defaultManager()

// <app home>/Documents, backes opp, kan bli vist til bruker
// For brukerens datafiler
println(fm.URLsForDirectory(.DocumentDirectory, inDomains: .UserDomainMask)[0])

// <app home>/Library, backes opp, skjult
// For det som ikke er brukerens datafiler
println(fm.URLsForDirectory(.LibraryDirectory, inDomains: .UserDomainMask)[0])

// <app home>/Library/Caches, backes IKKE opp, skjult
// F.eks. for caching av bilder
println(fm.URLsForDirectory(.CachesDirectory, inDomains: .UserDomainMask)[0])

// <app home>/tmp, backes IKKE opp
// For midlertidige filer som ikke trenger å eksistere mellom launches
println(NSTemporaryDirectory())
```

Skriv og les string til disk

```
let dir = NSFileManager.defaultManager().URLsForDirectory(.DocumentDirectory,
        inDomains: .UserDomainMask)[0] as NSURL

let string: NSString = "Hello world"

let path = dir.URLByAppendingPathComponent("file.txt").path!
var writeError: NSError?
if(!string.writeToFile(path, atomically: true, encoding: NSUTF8StringEncoding, error: &writeError)) {
    println("Write failed: \(writeError)")
}

var readError: NSError?
if let savedString = NSString(contentsOfFile: path, encoding: NSUTF8StringEncoding, error: &readError) {
    println(savedString)
} else {
    println("Read failed: \(readError)")
}
```

Skriv og les NSDictionary til plist

```
let dir = NSFileManager.defaultManager().URLsForDirectory(.DocumentDirectory,
                  inDomains: .UserDomainMask)[0] as NSURL
let dict = ["workouts": 23] as NSDictionary
println(dir)

let path = dir.URLByAppendingPathComponent("file.plist").path!

// Skriv
dict.writeToFile(path, atomically: true)

// Les
println(NSDictionary(contentsOfFile: path))
```

NSUserDefaults

NSUserDefaults

- For enkle verdier og caser
- Eksempel: lagre brukerens preferanser
- Verdier caches slik at man slipper diskaksess ved hver henting
- Synkronisering av verdier mot disk skjer automatisk ved jevne mellomrom

```swift
let userDefaults = NSUserDefaults.standardUserDefaults()
userDefaults.setObject("Tim Cook", forKey: "name")

if let name = userDefaults.stringForKey("name") {
    println("Got Name: \(name)")
}

// Tving synkronisering mot disk
userDefaults.synchronize()
```

NSKeyedArchiver / NSKeyedUnarchiver

NSKeyedArchiver / NSKeyedUnarchiver

- For serialisering av objekter til disk
- Klasser som skal serialiseres må implementere NSCoding-protokollen. Mange standard datatyper gjør dette allerede.
- Pass på fremover og bakoverkompatibilitet

Klasse som implementerer NSCoding

```
class Workout : NSObject, NSCoding {
    var name: String!
    var entries: Int = 0

    required convenience init(coder aDecoder: NSCoder) {
        self.init()
        name = aDecoder.decodeObjectForKey("name") as String?
        entries = aDecoder.decodeIntegerForKey("entries")
    }

    func encodeWithCoder(aCoder: NSCoder) {
        aCoder.encodeObject(self.name, forKey: "name")
        aCoder.encodeInteger(self.entries, forKey: "entries")
    }
}
```

Eksempel serialisering og deserialisering

```
let fm = NSFileManager.defaultManager()

let libDir = fm.URLsForDirectory(.LibraryDirectory, inDomains: .UserDomainMask)[0] as NSURL
println(libDir)

let workout = Workout()
workout.entries = 14
workout.name = "Pull-ups"

let path = libDir.URLByAppendingPathComponent("workouts").path!

// Serialisere til disk
NSKeyedArchiver.archiveRootObject(workout, toFile: path)

// Deserialisere fra disk
let savedWorkout = NSKeyedUnarchiver.unarchiveObjectWithFile(path) as Workout
println("\(savedWorkout.name), entries: \(savedWorkout.entries)")
```

Core Data

Core Data

For mer komplekse behov, når du trenger:

- Å lagre objektgrafer (med relasjoner)
- Å gjøre spørringer mot objektgrafer
- Støtte undo/redo
- Lagre objektgrafer i iCloud

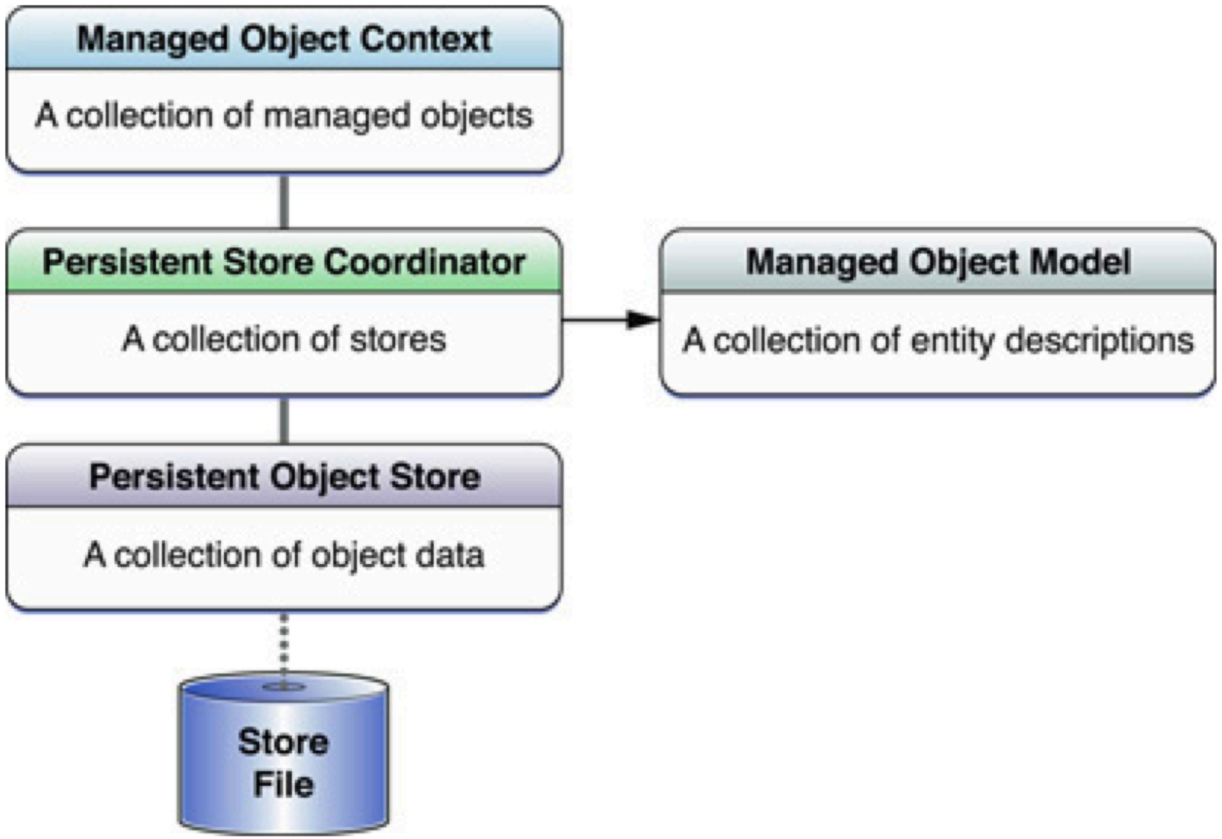
	Core Data	NSKeyedArchiver
Entity Modeling	Yes	No
Querying	Yes	No
Speed	Fast	Slow
Serialization Format	SQLite, XML, or NSData	NSData
Migrations	Automatic	Manual
Undo Manager	Automatic	Manual

nshipster.com/nscoding/

	Core Data	NSKeyedArchiver
Persists State	Yes	Yes
Pain in the Ass	Yes	No

Terminologi

- **Managed Object Context** - "hovedobjektet" mot Core Data. Håndterer et sett Managed Objects og deres lifssyklus
- **Persistent Store Coordinator** - abstraherer bort underliggende store. Implementerer henting/lagring/sletting/ m.m. av MOM mot store
- **Managed Object Model** - som et slags databaseskjema
- **Persistent Object Store** - XML, SQLite, Binær, custom



ManagedObjectContext

```
lazy var managedObjectContext: NSManagedObjectContext? = {
    let coordinator = self.persistentStoreCoordinator
    if coordinator == nil {
        return nil
    }
    var managedObjectContext = NSManagedObjectContext()
    managedObjectContext.persistentStoreCoordinator = coordinator
    return managedObjectContext
}()
```

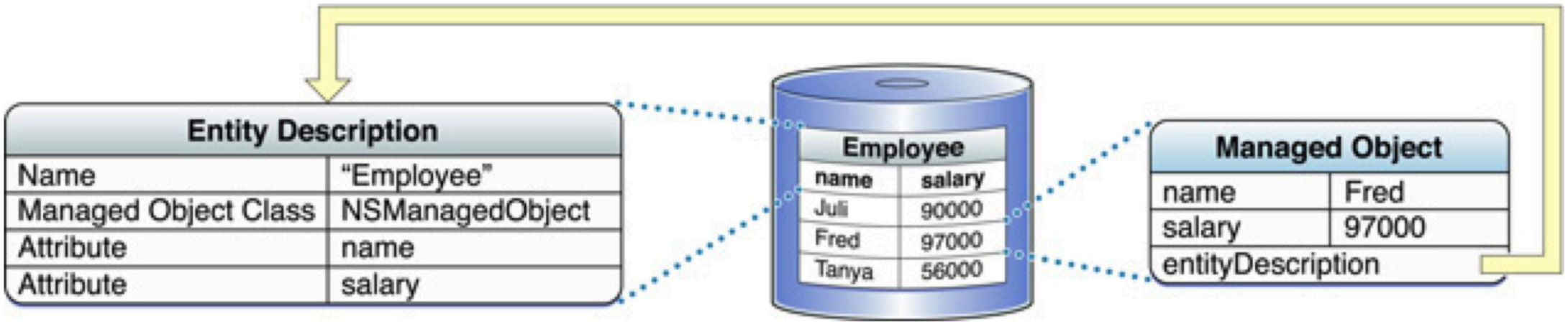
PersistentStoreCoordinator

```
lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator? = {

    var coordinator: NSPersistentStoreCoordinator? = NSPersistentStoreCoordinator(
            managedObjectModel: self.managedObjectModel)
    let url = self.applicationDocumentsDirectory.
                    URLByAppendingPathComponent("Workouts.sqlite")
    var error: NSError? = nil

    if coordinator!.addPersistentStoreWithType(NSSQLiteStoreType, configuration: nil,
        URL: url, options: nil, error: &error) == nil {
        coordinator = nil
        //... feilhåndtering, se XCode template
    }

    return coordinator
}()
```

ManagedObjectModel

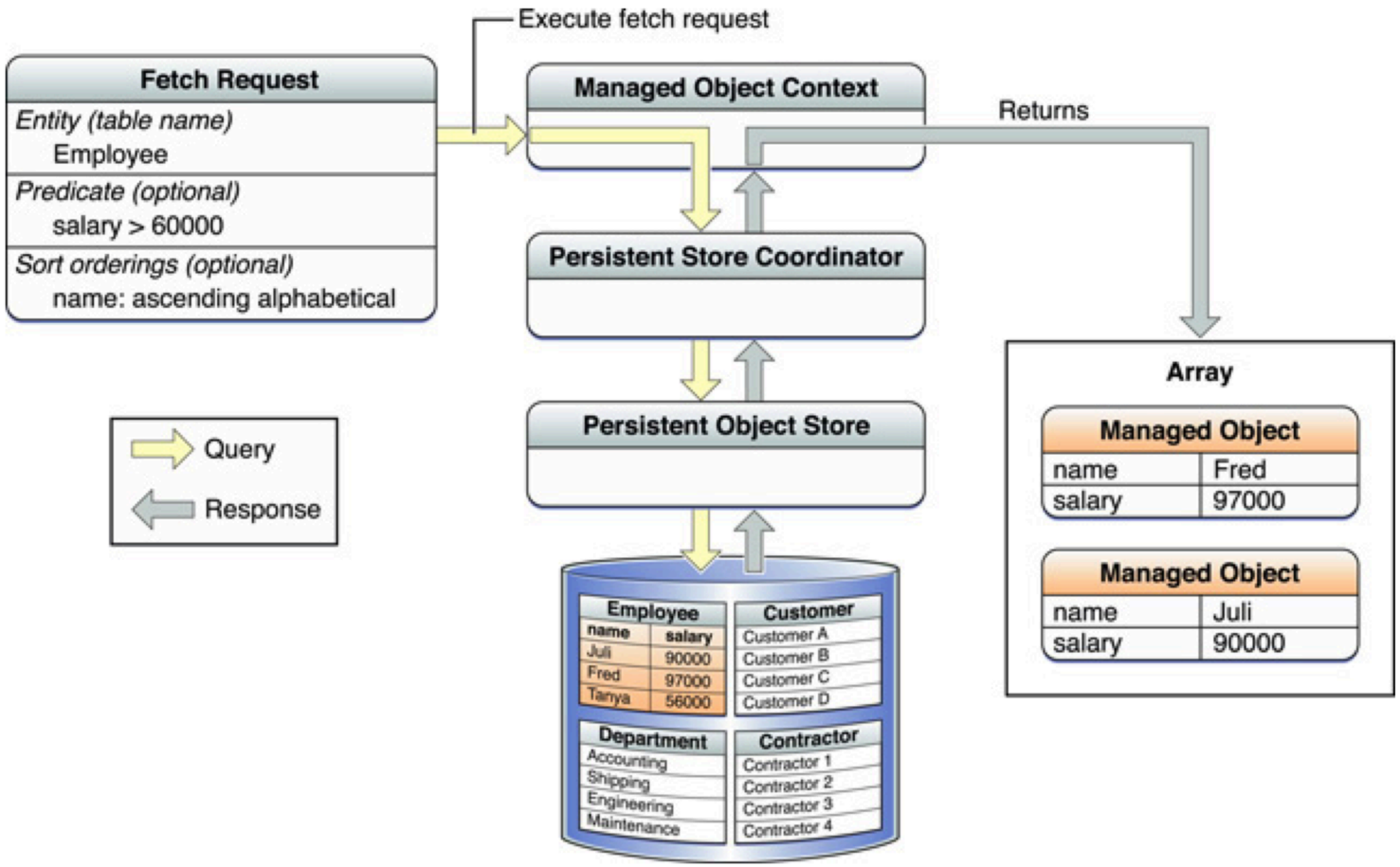
```
lazy var managedObjectModel: NSManagedObjectModel = {
    let modelURL = NSBundle.mainBundle().URLForResource("Workouts",
        withExtension: "momd")!
    return NSManagedObjectModel(contentsOfURL: modelURL)!
}()
```

Managed Objects



I utgangspunktet NSManagedObject's med key/value. Disse **kan** og **bør** ofte ha en subklasse for renere kode og mer funksjonalitet, men må ikke

Fetch Requests



Hent alle

```
let query = NSFetchRequest(entityName: "Workout")
var error: NSError?
if let results = moc.executeFetchRequest(query, error: &error) as? [Workout] {
    workouts = results
    //tableView.reloadData()
} else {
    println("Query failed: \(error)")
}
```

Fetch med predikat

```
var query = NSFetchRequest(entityName: "Workout")
query.predicate = NSPredicate(format: "entries >= %d", 5)

var error: NSError?
if let results = moc.executeFetchRequest(query, error: &error) as?
    [Workout] {
    workouts = results
    //tableView.reloadData()
} else {
    println("Query failed: \(error)")
}
```

Sortering og limit

```
// Hent topp 3
var fetchRequest = NSFetchRequest(entityName: "Workout")
fetchRequest.sortDescriptors = [NSSortDescriptor(key: "entries", ascending: false)]
fetchRequest.fetchLimit = 3
```

Opprett

```
let entity = NSEntityDescription.entityForName("Workout",
                    inManagedObjectContext: moc)
var workout = Workout(entity: entity!,
                    insertIntoManagedObjectContext: moc)
workout.name = excersice
workout.entries = 0

var error: NSError?
if moc.save(&error) {
    workouts.append(workout)
} else {
    println("Save failed \(error)")
}
```

Endre

```
//let workout = workouts[indexPath.row]
workout.entries = workout.entries.integerValue + 1

var error: NSError?
if(!moc.save(&error)) {
    println("Failed to update: \(error)")
}
```

Slett

```
//let workoutToRemove = workouts[indexPath.row]
moc.deleteObject(workoutToRemove)
if(!moc.save(&error)) {
    println("Failed to delete: \(error)")
}
```

Telle

```
let fetchRequest = NSFetchRequest(entityName: "Workout")
let count = moc.countForFetchRequest(fetchRequest, error: nil)
println("\(count) øvelser registrert")
```

NSFetchedResultsController

- No more `tableView.reloadData()`!
- Går hånd i hanske med UITableView
- Gir gruppering, caching, synkronisering av tableview mot data
- Abstraksjon rundt fetchRequest og resultatene
- Må ha minst en sortDescriptor

NSFetchedResultsController API

```
// Instansier (eks. i viewDidLoad)
NSFetchedResultsController(fetchRequest: query,
        managedObjectContext: moc, sectionNameKeyPath: nil, cacheName: nil)

// Hent data
fetchedResultsController.performFetch(&error)

// Få antall rader i section:
fetchedResultsController.sections![section].numberOfObjects

// Hent ut objekt med:
fetchedResultsController.objectAtIndexPath(indexPath)
```

NSFetchedResultsController - automatisk oppdatering av tableview

```
class ViewController: UIViewController, UITableViewDataSource,
    UITableViewDelegate,NSFetchedResultsControllerDelegate {

    // I ViewDidLoad, new opp NSFetchedResultsController og sett delegate:
    // fetchedResultsController.delegate = self

    func controllerWillChangeContent(controller: NSFetchedResultsController) {
        tableView.beginUpdates()
    }

    func controllerDidChangeContent(controller: NSFetchedResultsController) {
        tableView.endUpdates()
    }
}
```

```
func controller(controller: NSFetchedResultsController,
    didChangeObject anObject: AnyObject,
    atIndexPath indexPath: NSIndexPath,
    forChangeType type: NSFetchedResultsChangeType,
    newIndexPath: NSIndexPath) {

    switch type {
        case .Insert:
            self.tableView.insertRowsAtIndexPaths([newIndexPath],
                withRowAnimation: UITableViewRowAnimation.Automatic)
        case .Update:
            if let cell = self.tableView.cellForRowAtIndexPath(indexPath) {
                self.configureCell(cell, indexPath: indexPath)
            }
        case .Move:
            tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation: .Automatic)
            tableView.insertRowsAtIndexPaths([newIndexPath], withRowAnimation: .Automatic)
        case .Delete:
            tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation: .Automatic)
        default:
            break
    }

}
```

Mer

- Validering - sett i datamodell. Populerer `error` ved feil
- Relasjoner - sett i datamodell. En-en, en-mange, mange-mange
- Versjonering - når du endrer datamodellen mellom releaser
- Core Data i iCloud

Keychain

Keychain

- For lagring av sensitive data (passord m.m.)
- C API
- Se "iOS Keychain Services Tasks" i dokumentasjonen for mer info

Oppgaver

Se Øvingsoppgavene

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