

## Table of Contents

<b>De mobiele markt.....</b>	<b>4</b>
<b>Android vs. iOS.....</b>	<b>4</b>
<b>Ontwikkelen voor iOS.....</b>	<b>4</b>
<b>Foundation Kit.....</b>	<b>5</b>
<b>Variabelen.....</b>	<b>5</b>
<b>Pointers.....</b>	<b>6</b>
<b>Klassen.....</b>	<b>6</b>
<b>Methoden .....</b>	<b>7</b>
<b>Nil .....</b>	<b>8</b>
<b>Id .....</b>	<b>8</b>
<b>Self.....</b>	<b>8</b>
<b>Init methoden .....</b>	<b>8</b>
<b>Dynamic vs. static typing .....</b>	<b>10</b>
<b>NSLog .....</b>	<b>10</b>
<b>Properties .....</b>	<b>10</b>
Kenmerken.....	10
Properties definiëren .....	11
Synthesizing properties .....	11
Property attributen .....	11
Strong .....	11
Weak .....	11
Copy.....	12
Strong reference cycle.....	12
Synthesize instantievars .....	12
Dot syntax en self.....	12
Gedistribueerd programmeren (non-atomic/atomic) .....	13
Private properties en methoden.....	13
Visibiliteit datamembers (niet properties).....	14
<b>Memory management.....</b>	<b>14</b>
<b>Protocols .....</b>	<b>15</b>
<b>Collections .....</b>	<b>16</b>
Mutable vs. immutable .....	16
Arrays (initialisatie) .....	16
Collection overlopen .....	17
Arrays (contains object) .....	17
NSMutableArray .....	17
Dictionary .....	18
NSMutableDictionary .....	18
Set .....	19

<b>NSMutableSet .....</b>	<b>19</b>
<b>Model View Controller .....</b>	<b>20</b>
<b>Model .....</b>	<b>20</b>
<b>View.....</b>	<b>20</b>
<b>Controller.....</b>	<b>20</b>
<b>Werking .....</b>	<b>20</b>
<b>Levenscyclus van ViewControllers .....</b>	<b>20</b>
<b>PrepareForSegue (project).....</b>	<b>21</b>
<b>Aanmaken project.....</b>	<b>21</b>
<b>Opbouw schermen .....</b>	<b>21</b>
<b>Voorzie seques .....</b>	<b>21</b>
<b>Segue identifier.....</b>	<b>21</b>
<b>ViewController klassen.....</b>	<b>21</b>
<b>IBOutlets .....</b>	<b>22</b>
<b>ViewControllerGroen.h .....</b>	<b>22</b>
<b>ViewController .....</b>	<b>22</b>
<b>ViewControllerGroen en Rood.m.....</b>	<b>22</b>
<b>TableViewController (project) .....</b>	<b>23</b>
<b>Storyboard .....</b>	<b>23</b>
<b>Code.....</b>	<b>23</b>
<b>Eindresultaat.....</b>	<b>24</b>
<b>Uitbreiding .....</b>	<b>24</b>
<b>Location &amp; Maps .....</b>	<b>25</b>
<b>Location Services.....</b>	<b>25</b>
<b>Maps .....</b>	<b>25</b>
<b>Gestures.....</b>	<b>26</b>
<b>NSURLSession.....</b>	<b>27</b>
<b>Kenmerken.....</b>	<b>27</b>
<b>Codevoorbeeld.....</b>	<b>28</b>
<b>Blocks.....</b>	<b>28</b>
<b>Kenmerken.....</b>	<b>28</b>
<b>Codevoorbeeld.....</b>	<b>28</b>
<b>NSJSONSerialization .....</b>	<b>28</b>
<b>Kenmerken.....</b>	<b>28</b>
<b>Codevoorbeeld.....</b>	<b>29</b>
<b>Probleem .....</b>	<b>29</b>
<b>Oplossing .....</b>	<b>29</b>
<b>Onveilige communicatie – Opmerking .....</b>	<b>29</b>
<b>Core Data .....</b>	<b>30</b>
<b>ManagedObjectContext .....</b>	<b>30</b>
<b>Beheren van entiteiten.....</b>	<b>31</b>
<b>DIY .....</b>	<b>31</b>
<b>Internationalization &amp; Localization .....</b>	<b>32</b>
<b>Hoe? .....</b>	<b>32</b>
<b>Taal toevoegen.....</b>	<b>32</b>
<b>Getallen .....</b>	<b>32</b>

<b>Storyboards.....</b>	<b>33</b>
<b>Afbeeldingen.....</b>	<b>33</b>

# Mobile Apps: iOS

## De mobiele markt

### Kenmerken

- Smartphone markt: 80% Android, 17% iOS
- Gedownloade apps (geïndexeerd): 100 iOS App Store, 190 Google Play
- Omzet (geïndexeerd): 100 iOS App Store, 60 Google Play

## Android vs. iOS

### Kenmerken

- Op 9 november 2007 kwam Apple op de markt met iPhone
- Op 21 oktober 2008 lanceerde de Open Handset Alliance Android
- Beiden hebben een Linux/UNIX-achtergrond
- Echter belangrijk verschil wat betreft bescherming van intellectueel eigendom
  - iOS volledig gesloten
    - Ontwikkeld door en voor Apple
    - Geen licenties voor derden
  - Android volledig open systeem

### iOS – Gesloten systeem

- Voordelen
  - Ontwikkeling wordt gestuurd door één bedrijf
  - Ontwikkelaars weten op welke apparaten hun besturingssysteem zal werken, ze kunnen het systeem perfect afstemmen op de hardware
- Nadelen
  - Keuzevrijheid voor consument is beperkt
  - Ontwikkeling wordt gestuurd door één bedrijf

### Android – Open systeem

- Ontwikkeld door Google en hun partners binnen Open Handset Alliance
- Voordeel
  - Samenwerking tussen groot aantal mensen en bedrijven
  - Eender welk bedrijf kan een op Android gebaseerde smartphone ontwikkelen
- Nadeel
  - Samenwerking tussen groot aantal mensen en bedrijven
  - Aanpassingen aan de software zijn nodig om op specifieke platformen te werken + aanleveren van SDK voor alle platformen = kost tijd en energie

## Ontwikkelen voor iOS

### Kenmerken

- Noodzakelijk: Intel Mac, iOS SDK en gebruik maken van informatie op iOS Dev Center
- Ontwikkelen kan zonder problemen op virtuele iPhone of iPad
- Testen doe je bij voorkeur op een iPhone, iPod Touch of iPad

## iOS SDK

- Bestaat uit Xcode IDE en iOS Simulator
- Alternatieve IDE ontwikkeld door JetBrains genaamd AppCode
- Interface Builder: ontwerpen interface van applicatie
  - Werkt met .xib-bestanden en sinds iOS 5 storyboards
  - Op XML-gebaseerde bestanden
  - Worden “Nib files” genoemd (NextStep Interface Builder)
- Xcode: instruments
  - iOS heeft geen automatisch geheugenbeheer (geen garbage collection)
  - Tools om te testen op memory leaks, netwerk gebruik, CPU-gebruik, performantie (profiler), ...

## Objective-C

- iOS applicaties worden geschreven in Objective C, een superset van C
- Alles wat in C voorkomt is ook geldig in Objective-C

## Swift

- Geïntroduceerd juni 2014
- Stable release 2.1.1 December 2015
- Vanaf versie 2.2 open source!

## Foundation Kit

De **Foundation Kit** of gewoon **Foundation** is een Objective-C **framework**.

- Voorziet ons in de basis klassen zoals wrapper klassen en datastructuur klassen
- Gebruikt de prefix NS (van NeXTSTEP)

**NeXTStep** is een besturingssysteem ontwikkeld door Steve Jobs' bedrijf NeXT

- Uitgebracht in 1989
- Het besturingssysteem van de Apple Macintosh, Mac OS X, is een directe afstammeling van NeXTStep
- De eerste webbrowser ooit was het NeXTStep-programma WorldWideWeb, geïntroduceerd door Tim Berners-Lee in 1991. Berners-Lees NeXTcube- computer met NeXTStep was de eerste webserver ter wereld.

## Variabelen

### Datatypes

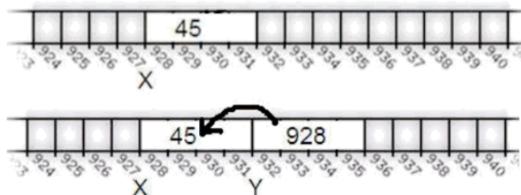
- int
- char
- float
- double
- BOOL (YES/NO)

```
int myFavNumber = 3;
```

## Pointers

- Pointer is een variabele die als waarde een geheugenadres bevat
- Wanneer je een variabele als pointer wil declareren wordt deze voorafgegaan door een \* in de declaratie

```
int x = 45;  
int* y;  
y = &x;
```



```
int x = 45;  
int *y = &x;  
NSLog(@"De waarde is %d", *y);
```

```
int x = 45;  
int *y = &x;  
*y = 90;  
NSLog(@"De waarde is %d", x);  
NSLog(@"De waarde is %d", *y);
```

## Klassen

### NSObject

- Basisklasse van Objective-C hiërarchie
- Basisvoorziening zoals memory management en introspection (bevragen objecten)

### NSString

- Klasse gebruikt voor string manipulaties
- Objecten van deze klasse stellen een unicode string voor

```
NSString tmp = @"Dit is een NSString";  
NSLog(@"%@", tmp);
```

### Klasse aanmaken

- Twee bestanden
  - Headerbestand met extensie .h
  - Implementatiebestand met extensie .m
- Headerbestand bevat interface van klasse; In deze public interface beschrijven we hoe objecten kunnen interageren met instanties van je klasse
- Implementatiebestand bevat uiteindelijke implementatie van de klasse

```
#import <Foundation/Foundation.h>  
  
@interface Teller : NSObject  
{  
    int totaal;  
}  
  
- (void) voegToe:(int) aantal;  
- (int) totaal;  
- (void) reset;  
@end
```

```
#import "Teller.h"  
  
@implementation Teller  
  
- (void) voegToe:(int) aantal  
{  
    totaal = totaal + aantal;  
}  
- (int) totaal  
{  
    return totaal;  
}  
- (void) reset  
{  
    totaal = 0;  
}  
@end
```

## Instantie creëren

- Instantie van klasse creëren is proces uit twee stappen
  - Alloceren van geheugen (alloc)
  - Vervolgens initialiseren we het object met een van de init methodes; De init methode geeft het object terug als waarde

```
Teller *teller = [[Teller alloc] init];  
[teller reset];  
[teller voegToe:100];  
NSLog(@"%@", [teller totaal]);
```

## Methoden

### Definiëren

```
#import "Teller.h"  
@implementation Teller  
  
- (void) voegToe:(int) aantal  
{  
    totaal = totaal + aantal;  
}  
- (void) voegToe:(int) aantal1 enOok:(int) aantal2;  
{  
    totaal = totaal + aantal1 + aantal2;  
}  
- (int) totaal  
{  
    return totaal;  
}  
- (void) reset  
{  
    totaal = 0;  
}  
@end
```

```
#import <Foundation/Foundation.h>  
  
@interface Teller : NSObject  
{  
    int totaal;  
}  
  
- (void) voegToe:(int) aantal;  
- (void) voegToe:(int) aantal1 enOok:(int) aantal2;  
- (int) totaal;  
- (void) reset;  
@end
```

### Methoden aanroepen

- [receiver message]
- [receiver message:argument]
- [receiver message:arg1 andArg:arg2]

```
Teller *teller = [[Teller alloc] init];  
[teller reset];  
[teller voegToe:100];  
[teller voegToe:100 enOok:100];  
NSLog(@"%@", [teller totaal]);
```

### Terminologie

- Message expression
  - [receiver message:argument]
- Message
  - [receiver message:argument]
- Selector
  - [receiver message:argument]
- Method
  - De code geselecteerd door een message

## Nil

- Het nil-object is een speciaal object dat alle oproepen absorbeert
- Het nil-object zal nil of 0 als resultaat opleveren in plaats van een fout te genereren wanneer men er een boodschap naar stuurt

```
Dog *dog = nil;  
[dog bark];  
// geen uitvoer, geen foutmelding  
  
if ([object isGood]) {  
    // Indien object niet nil is,  
    // wordt de opdracht uitgevoerd  
    // Indien object nil is, gebeurt er niets  
}
```

## Id

- Id is een type dat kan verwijzen naar ieder type object
- In het id datatype kan je een referentie naar eender welk object opslaan

```
Teller *teller = [[Teller alloc] init];  
id temp = teller;
```

## Self

- Objecten hebben een impliciete variabele genaamd “self”
  - Zoals “this” in Java
- Via self kan je in een methode van een object een andere methode van hetzelfde object oproepen

```
#import "Teller.h"  
@implementation Teller  
  
...  
  
- (void) reset  
{  
    totaal = 0;  
}  
- (BOOL) resetBis  
{  
    [self reset];  
    return YES;  
}  
@end
```

## Init methoden

### Initializer implementeren

- Wanneer je één init methode voorziet
  - Roep de initializer van de superklasse aan
  - Controleer het object dat teruggestuurd wordt door de superklasse: indien deze nil oplevert kan de init methode niet verder uitgevoerd worden, dan wordt nil gereturnd
  - Voer initialisatie van datamembers uit
  - Return self

- Meerdere initializers
  - Designated initializer is de meest volledige initializer, de initializer die andere init methoden oproepen
  - Designated initializer roept initializer van superklasse op

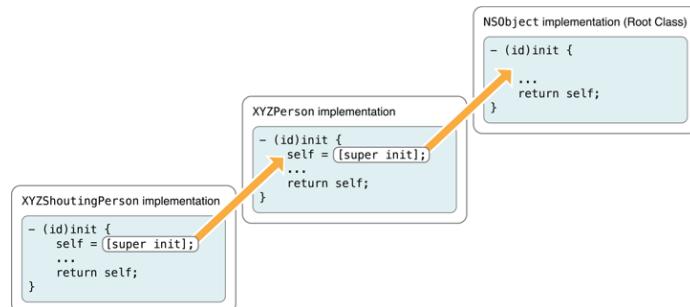
## Designated initializer

Iedere klasse heeft een designated initializer

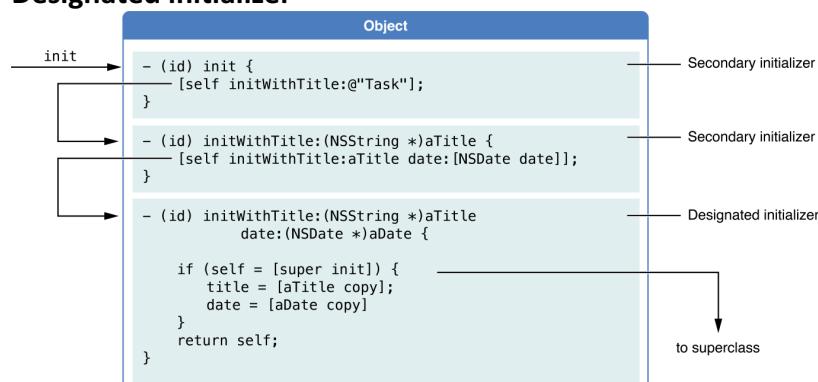
- Alle andere init-methoden roepen deze designated initializer op.
- De designated initializer is de init-methode die de 'designated initializer' oproept van de superklasse.
- De 'designated initializer' laat meestal toe om alle initialisatieparameters mee te geven die nodig zijn voor de aanmaak van het object.

```
#import "Person.h"
@implementation Person
- (id)init
{
    if (self = [super init])
    {
        age = 0;
        name = @"bob";
    }
    return self;
}
@end
```

## Init methoden



## Designated initializer



## Dynamic vs. static typing

### Dynamic typing

- Een programmeertaal wordt dynamisch getypeerd genoemd als type checking plaatsvindt tijdens de uitvoering van het programma in plaats van tijdens het compileren van het programma
- Alias late-binding

### Static typing

- Als een programmeertaal statisch getypeerd is, vindt type checking plaats tijdens het compileren (compile-time), en niet tijdens het uitvoeren (run-time) van het programma
- Als de compiler een fout vindt wordt de compilatie afgebroken en de programmeur wordt op de hoogte gesteld van het probleem
- Bijvoorbeeld: Java

```
BankAccount *account1;
account1 = [[BankAccount alloc] init];
```

### Id en dynamic typing

- Dynamic typing en het id type laten toe een variabele aan te maken waarin eender welk type object kan worden opgeslagen tijdens de uitvoer van het programma
- Dynamic binding maakt het mogelijk om methoden op te roepen op een object dat bewaard werd in een id variabele zonder kennis over het type object dat op dat moment toegewezen werd

```
id object1;

object1 = [[SavingsAccount alloc] init];
[object1 setAccount: 4543 andBalance: 310.10];

object1 = [[CustomerInfo alloc] init];
[object1 displayInfo];
```

## NSLog

```
 NSLog(@"%@", somenumber);
```

### Format specifiers:

- %i Integer
- %c Character
- %d Signed decimal Integer
- %f Floating-point decimal
- %u Unsigned decimal integer
- %@ Object (bv NSString)

## Properties

### Kenmerken

- Voorzien toegang tot instantievariabelen
- Als vervanging van het zelf implementeren van getter/setter methoden
- Mogelijkheden: read-only/read-write access, memory management policy

## Properties definiëren

```
#import <Foundation/Foundation.h>

@interface Person : NSObject
{
    // Enkel data members
}
@property int age;
@property (copy) NSString *name;
@property (readonly) BOOL canLegallyVote;
- (void)castBallot;
@end
```

## Synthesizing properties

Niet meer verplicht, kan gebruik worden om bijvoorbeeld Nederlands binnen de klasse te gebruiken, en Engels buiten de klasse

```
@implementation Person
@synthesize age = _age; // Default, niet vereist
@synthesize name = myName;
- (BOOL)canLegallyVote
{
    return (_age > 17);
}
@end
```

## Property attributen

- Read-only vs. read-write
  - @property int age → read-write by default
  - @property (**readonly**) BOOL → canLegallyVote;
- Memory management policies
  - Enkel bij objecten!
  - @property (**strong**) NSString \*name; // taking ownership
  - @property(**weak**)NSString\*name; //weak reference
  - @property(**copy**)NSString\*name; //copy called

## Strong

- Sterke relatie: je bent (mede)eigenaar van het object.
- Indien de originele eigenaar van het object zijn referentie loslaat, blijft het object nog steeds behouden.
- Default relatie voor object typen.

## Weak

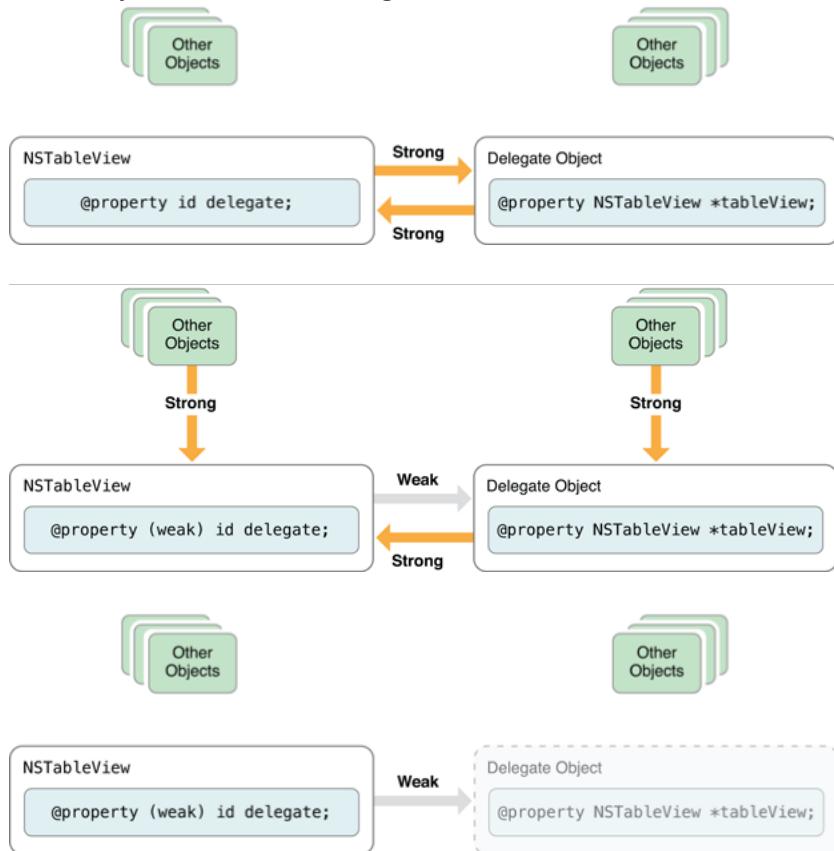
- Zwakke relatie: indien het object gedealloceerd wordt, zal de property onmiddellijk op nil worden gezet.

## Copy

- Zorgt dat kopie van object wordt gemaakt.
- Enkel geldig bij objecten die NSCopying protocol implementeren.

## Strong reference cycle

Twee objecten die een strong reference naar elkaar hebben!



## Synthesize instantievars

- De property implementatie wordt automatisch voorzien door de compiler, maar kan ook expliciet gebeuren door middel van het `@synthesize` keyword in het implementatie bestand.
- `@synthesize property = _property;`

## Dot syntax en self

Opletten met de dot syntax voor properties!

```
@implementation Person
- (void)setAge:(int)newAge
{
    self.age = newAge; // Infinite loop, idem [self setAge:newAge]
}
@end
```

## Gedistribueerd programmeren (non-atomic/atomic)

- Atomic: default, gesynthetiseerde getters zullen een lock gebruiken
- Gebruik steeds non-atomic properties behalve wanneer klasse thread-safe moet zijn
- Wanneer properties niet gedeclareerd worden als non-atomic wordt de indruk gegeven dat ze ontworpen werden met voorzieningen voor thread-safety

```
@interface Foo : NSObject  
@property(strong, nonatomic) NSObject* myObj;  
@end
```

## Niet automatisch volledig thread-safe

- Atomic properties zorgen niet automatisch dat object volledig thread-safe is
- Bijvoorbeeld
  - Object Persoon heeft een naam en voornaam die aangepast worden vanuit een bepaalde thread.
  - Een andere thread die toegang wil tot deze namen op net hetzelfde moment zal gegarandeerd een volledige string terugkrijgen, maar er is geen garantie of de naam ook bij de voornaam hoort of visa versa.

## Gebruik steeds self voor toegang tot properties

- Ook binnen eigen implementatie
- Enige uitzondering: init methode, deallocate of custom getter/setter methoden

## Private properties en methoden

- Wanneer je een private methode of property wenst, declareer je deze steeds in de "class extension" in het **implementatie bestand**

```
// Foo.m  
@interface Foo ()  
@property(strong, nonatomic) Bar* myPrivateProperty;  
- (int)myPrivateMethod;  
@end  
@implementation Foo  
// ...  
@end
```

## Visibiliteit datamembers (niet properties)

Standaard is **protected**

```
@interface Foo : NSObject
{
    NSString naam;
    @private int a;
    NSObject* b;
    @protected
    int b;
    @public
    int c;
}
@end
```

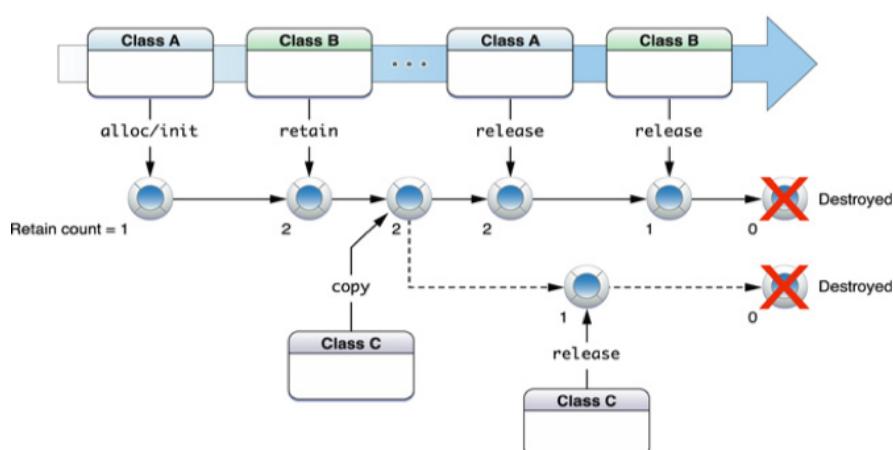
## Memory management

### Twee soorten

- Manual Retain-Release (MRR)
- Automatic Reference Counting (ARC)

### Manual Retain-Release (MRR)

- We bekijken dit omdat nog legacy code is met MRR
- We gebruiken echter ARC voor alle nieuwe projecten



### Automatic Reference Counting

- Ieder object heeft een retain count
  - Gedefinieerd in NSObject
  - Zolang retain count > 0, wordt het object behouden
- Via +alloc en -copy maken we objecten aan met een retain count van 1
- Retain: increments retain count
- Release: decrements retain count
- Wanneer retain count = 0, wordt object vernietigd
  - -dealloc wordt automatisch aangeroepen

## Dealloc methode

- MMR: voorzie nooit –dealloc methode behalve wanneer je bijzondere resources moet vrijgeven zoals bestanden, timers, ...
- ARC: ook niet gebruiken. Objective-C ARC zal normaal automatisch al het geheugen vrijgeven.

## Protocols

Vergelijkbaar met **Java Interfaces**

Je kan een klasse een protocol laten implementeren.

- Protocol heet **verplichte** en **optionele** methoden

```
//Drivable.h
@protocol Drivable
// @required is the default
-(void) drive: (int) distance;
@optional
-(void) reverse;
@required
-(int) miles;
@end
```

```
#import <Foundation/Foundation.h>
#import "Drivable.h"

@interface Car : NSObject<Drivable> {} // Meerdere via komma
@end

@implementation Car
-(void) drive: (int) distance {
    NSLog("Driving %d miles\n", distance);
}
-(void) reverse {
    NSLog("Reversing\n");
}
-(int) miles {
    return 0;
}
@end
```

```
Car* car = [[Car alloc] init];
[car drive: 10];

id<Drivable> drivable = car;
[drivable reverse];
```

## Collections

### Drie grote groepen

- Arrays
- Dictionaries
- Sets

### Steeds in twee soorten

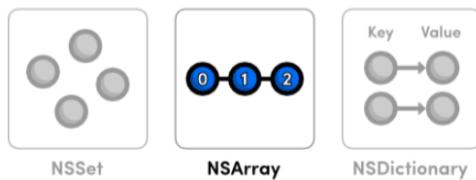
- Mutable (aanpasbaar)
- Immutable (niet aanpasbaar)

### Mutable vs. immutable

- Niet aanpasbare objecten zorgen ervoor dat een object **niet onverwachts wijzigt**
- Aanpasbare objecten voor basisbouwstenen zoals strings en verzamelingen zorgen voor **overhead**
- **Mutable** indien je **frequent en incrementeel de inhoud van het object wil aanpassen**
- In sommige gevallen is het interessant om een immutable object te vervangen door een ander immutable object
- Indien je twijfelt, kies dan steeds voor een immutable object

### Arrays (initialisatie)

Een **geordende** verzameling van objecten



#### Array initialiseren met **objecten**

```
NSArray* friends = [[NSArray alloc] initWithObjects: @"Joe", @"Jim", nil];
```

#### Array initialiseren met **objecten** (init wordt in de methode uitgevoerd)

```
NSArray *ukMakes = [NSArray arrayWithObjects:@"Aston Martin",@"Lotus",
                     @"Jaguar", @"Bentley", nil];
```

#### Array initialiseren met **NSString**

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",@"Opel",
                         @"Volkswagen", @"Audi"];
```

#### Waarde opvragen

```
NSLog(@"First german make: %@", germanMakes[0]);
NSLog(@"First U.K. make: %@", [ukMakes objectAtIndex:0]);
```

## Collection overlopen

### Enumerator

```
NSEnumerator* friendsEnumerator = [friends objectEnumerator];  
  
id aFriend;  
while ((aFriend = [friendsEnumerator nextObject])) {  
    NSLog(@"%@", aFriend);  
}
```

### For-loop

```
int friendsCount = [friends count];  
for(int i = 0; i < friendsCount; i++) {  
    NSLog(@"%@", [friends objectAtIndex: i]);  
}
```

### Foreach

```
for(NSString* aFriend in friends) {  
    NSLog(@"%@", aFriend);  
}
```

## Arrays (contains object)

### BOOL checking

```
if ([germanMakes containsObject:@"BMW"]) {  
    NSLog(@"BMW is a German auto maker");  
}
```

### Index checking

```
NSUInteger index = [germanMakes indexForObject:@"BMW"];  
if (index == NSNotFound) {  
    NSLog(@"Well that's not quite right...");  
} else {  
    NSLog(@"BMW is a German auto maker and is at index %ld", index);  
}
```

## NSMutableArray

### Mogelijkheid elementen **toevoegen/verwijderen na creatie**

```
NSMutableArray *brokenCars = [NSMutableArray arrayWithObjects:@"Audi A6",  
    @"BMW Z3", @"Audi Quattro", @"Audi TT", nil];
```

### Objecten toevoegen

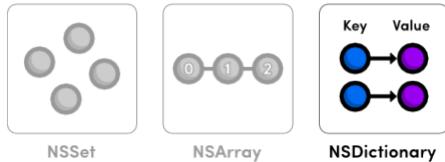
```
[brokenCars addObject:@"BMW F25"];  
[brokenCars insertObject:@"BMW F25" atIndex:0];
```

## Objecten verwijderen

```
[brokenCars removeLastObject];  
[brokenCars removeObjectAtIndex:0];  
[brokenCars removeObject:@"Audi Quattro"];
```

## Dictionary

Een dictionary is een **ongeordende** verzameling van objecten waarbij **sleutels** (key) worden geassocieerd met **waarden** (value)



**NSDictionary** is onveranderbaar daar waar **NSMutableDictionary** de mogelijkheid biedt om elementen toe te voegen en te verwijderen na creatie.

Als sleutel wordt in de volgende voorbeelden een NSString gebruikt, maar ieder object dat het **NSCopying** protocol implementeert, kan hiervoor worden aangewend.

**Initialisatie:** eerst waarde, vervolgens sleutel

```
NSDictionary* friends = [NSDictionary dictionaryWithObjectsAndKeys: @"44", @"Joe",  
    @"43", @"Jim", nil];
```

Waarde opvragen

```
for(NSString* aFriend in friends) {  
    NSLog(@"%@", aFriend, [friends objectForKey: aFriend]);  
}
```

Arrays gebruiken voor initialisatie

```
NSArray *models = @[@"Mercedes-Benz SLK250", @"Mercedes-Benz E350", @"BMW M3 Coupe", @"BMW X6"];  
NSArray *stock = @[[NSNumber numberWithInt:13], [NSNumber numberWithInt:22],  
    [NSNumber numberWithInt:19], [NSNumber numberWithInt:16]];  
  
inventory = [NSDictionary dictionaryWithObjects:stock forKeys:models];  
  
NSLog(@"There are %@ X6's in stock", inventory[@"BMW X6"]); NSLog(@"There are  
    %@ E350's in stock", [inventory objectForKey:@"Mercedes-Benz E350"]);
```

## NSMutableDictionary

**Initialisatie**

```
NSDictionary* friends = [NSDictionary dictionaryWithObjectsAndKeys: @"44", @"Joe",  
    @"43", @"Jim", nil];
```

Waarde aan **bestaande sleutel** koppelen

```
[jobs setObject:@"Mary" forKey:@"Audi TT"];
```

Waarde **verwijderen**

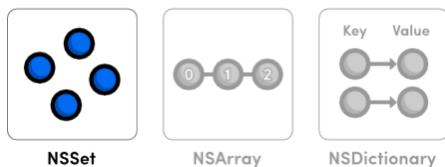
```
[jobs removeObjectForKey:@"Audi A7"];
```

Nieuwe waarde **toevoegen**

```
jobs[@"Audi R8 GT"] = @"Jack";
```

## Set

Een set is een **ongeordende lijst van unieke objecten**. NSSet is onveranderbaar.



**Initialisatie** (ook mogelijk met Array)

```
NSSet *americanMakes = [NSSet setWithObjects:@"Chrysler", @"Ford", @"General Motors", nil];
```

## NSMutableSet

**Initialisatie**

```
NSMutableSet *brokenCars = [NSMutableSet setWithObjects:@"Honda Civic", @"Nissan Versa", nil];
```

**Ruimte voorzien**

```
NSMutableSet *repairedCars = [NSMutableSet setWithCapacity:5];
```

**Objecten toevoegen/verwijderen**

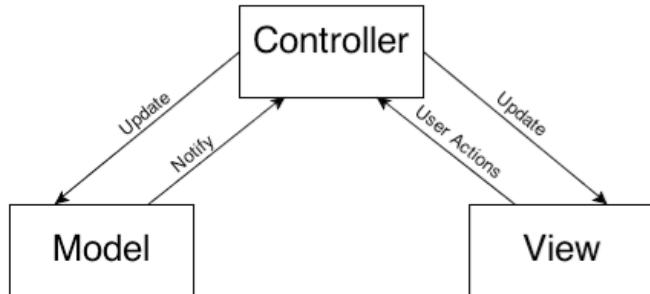
```
[brokenCars removeObject:@"Honda Civic"];
[repairedCars addObject:@"Honda Civic"];
```

**Afdrukken**

```
NSLog(@"Broken cars: %@", brokenCars); // Nissan Versa
NSLog(@"Repaired cars: %@", repairedCars); // Honda Civic
```

## Model View Controller

Een van meest gebruikte design patterns in iOS



### Model

Object dat **data** van applicatie **bevat** en aangeeft op welke manier deze data **gemanipuleerd** kan worden.

### View

Object verantwoordelijk voor de visuele representatie van het Model en mogelijkheden aanbiedt aan de gebruiker om ermee te interageren.

### Controller

Coördineert al het werk: zoekt toegang tot de data van het Model en toont deze data via de Views.

Luistert naar events en manipuleert data indien nodig.

### Werkings

- Model verwittigt Controller wanneer de data verandert, Controller zal op zijn beurt de data in Views aanpassen
- View zal Controller waarschuwen wanneer gebruiker bepaalde acties verricht, Controller zal indien nodig het Model aanpassen of bijkomende data opvragen
- Doel: herbruikbaarheid
- View moet idealiter volledig gescheiden zijn van het Model zodat de View kan herbruikt worden om andere data weer te geven; hetzelfde voor het Model

### Levenscyclus van ViewControllers

- `ViewDidLoad` – Uitgevoerd wanneer view in geheugen werd geladen
  - Hier plaats je de initialisatie van de view
- `ViewWillAppear` – View wordt zodra getoond aan gebruiker
  - Niet te veel code plaatsen (lag)
- `ViewDidAppear` – View werd juist getoond aan gebruiker
  - Eventueel kan je bijkomende views laten zien of bepaalde zaken animeren
- `ViewWillDisappear` – View zal zodra verwijderd worden
  - Op dit moment bepaalde waarden bewaren
- `ViewDidDisappear` – View werd verwijderd
  - Op dit moment kan je een volgende view tonen

Niet nodig om alle methoden te overschrijven!

## PrepareForSegue (project)

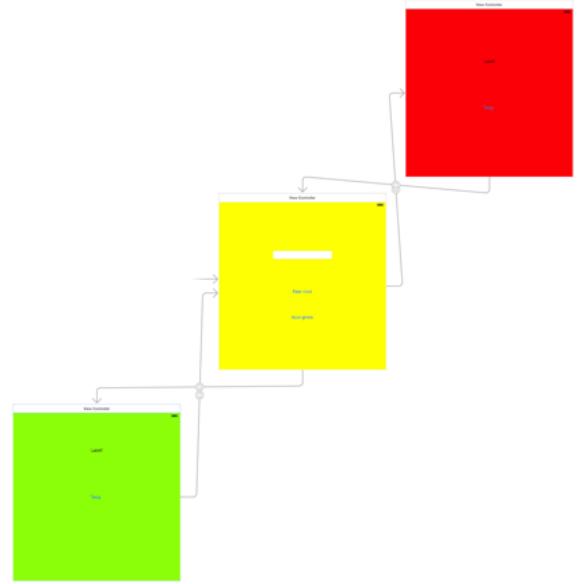
**Doelstelling** voorbeeld: aanleren hoe we informatie doorgeven van de ene viewcontroller naar de andere viewcontroller.

### Aanmaken project

- Nieuw project aanmaken
- Twee ViewControllers toevoegen aan storyboard

### Opbouw schermen

- Textfield en twee Buttons toevoegen aan eerste scherm
- Label en Button toevoegen aan tweede en derde scherm



### Voorzie seques

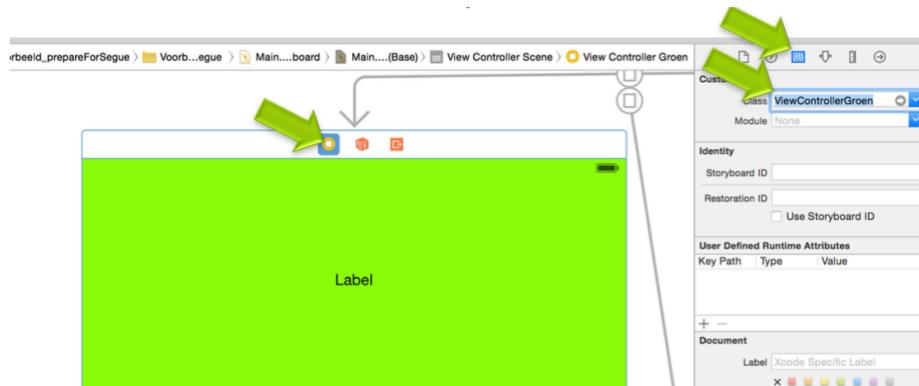
- Voorzie seques tussen de schermen door een knop te selecteren en vervolgens via ctrl-klik-sleep een link te leggen tussen de schermen in je storyboard
- In dit geval kiezen we voor "Present modally"
- Wanneer we gebruik maken van een Navigation View Controller kiezen "Show"

### Segue identifier

- We geven de verschillende seques namen. Vanuit het eerste scherm vertrekken er immers twee seques en we willen ieder segue op een aparte manier behandelen
- Selecteer de segue in het storyboard en geef de segue in identifier via de Attributes Inspector (waar je voordien ook de achtergrond kleur van de View kon aanpassen). Bijvoorbeeld: naarRood en naarGroen

### ViewController klassen

- Twee ViewController klassen (ViewControllerGroen en ViewControllerRood), subklassen van UIViewController
- Daarna verbinden met het storyboard



## IBOutlets

- Voorzie Outlets voor de UILabels in beide schermen
- Voorzie een Outlet voor het UITextField in het eerste scherm
- Doe dit steeds in de “class extension” in het implementatiebestand (.m)

## ViewControllerGroen.h

- Voeg een property toe waar je zo dadelijk een string aan kan doorgeven. We kunnen het label niet rechtstreeks aanpassen want vanaf het moment dat we overgaan naar het volgende scherm, is het userinterface element nog niet beschikbaar.

## ViewController

- In de eerste viewcontroller maken we de **prepareForSegue** methode aan
- We gaan na welke segue gevolgd zal worden (naarGroen of naarRood) en geven de inhoud van het UITextField door aan het property (temp) dat we daarstraks voorzien hadden

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender{
    if ([segue.identifier isEqualToString:@"naarRood"]){
        ViewControllerRood *vcRood = (ViewControllerRood*)segue.destinationViewController;
        vcRood.temp = self.myTextField.text;
    } else if ([segue.identifier isEqualToString:@"naarGroen"]){
        ViewControllerGroen *vcGroen = (ViewControllerGroen*)segue.destinationViewController;
        vcGroen.temp = self.myTextField.text;
    }
}
```

## ViewControllerGroen en Rood.m

```
@interface ViewControllerGroen ()
@property (weak, nonatomic) IBOutlet UILabel *myLabel;
@end

@implementation ViewControllerGroen

- (void)viewDidLoad {
    [super viewDidLoad];
    self.myLabel.text = self.temp;
}
```

## TableViewController (project)

**Doelestell** voorbeeld: aanleren hoe we een TableViewController gebruiken in een app

### Storyboard

- Sleep een TableViewController via de Object Library in het Storyboard
- Maak TableViewController de initial view controller
  - Via attribute inspector, wanneer je de TableViewController selecteert

**Opgelet:** wanneer je klikt in storyboard op een element, ben je niet altijd zeker wat je geselecteerd hebt → toon daarom de Document Outline links beneden in je Storyboard

### TableViewCell

- Selecteer de TableViewCell
- Pas de **stijl** aan
- Stel **identifier** in waarmee we deze cel identificeren in de code

Afhankelijk van de doelstellingen kan je de TableViewController inbedden in een **NavigationController**. Selecteer de controller, ga naar het Editor menu, en druk Embed In Navigation Controller.

### Code

Maak subklasse van **UITableViewController** aan

- Subklasse verbinden met TableViewController in Storyboard
- Selecteer in Storyboard de TableViewController
- Selecteer rechts bovenaan de Identity Inspector

### Bron van informatie

```
@interface TableViewController ()  
@property (nonatomic) NSDictionary *books;  
@end  
  
@implementation TableViewController  
  
- (void)viewDidLoad {  
    [super viewDidLoad];  
    self.books = [NSDictionary dictionaryWithObjectsAndKeys:@"How open is  
        the future?", @"Marleen Wynants", @"No logo", @"Naomi Klein",  
        @"Eating Animals", @"Jonathan Safran Foer", @"The Ancestor's Tale",  
        @"Richard Dawkins", @"The Spirit Level", @"Richard Wilkinson", @"The  
        World is Flat", @"Thomas L. Friedman", @"Amsterdam", @"Ian McEwan",  
        @"Free Software Free Society", @"Richard M. Stallman",  
        nil];
```

**numberOfSectionsInTable**-methode → geeft aan hoeveel secties in de TableView aanwezig zijn, minstens één!

**tableView:numberOfRowsInSection:-methode** → hoeveel rijen in tabel aanwezig, grootte van dictionary gebruiken [self.books count]

### tableView:cellForRowAtIndexPath:-methode

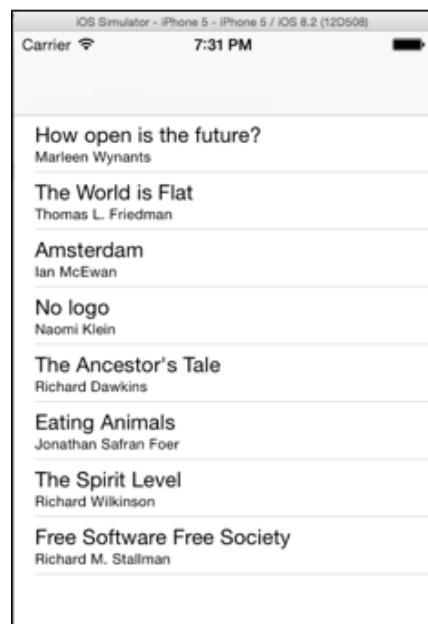
- Zal voor iedere rij in tabel opgeroepen worden, de methode maakt iedere rij aan
- We passen deze methode aan
  - Gebruik **identifier** uit Storyboard om naar TableViewCell uit te verwijzen
  - We gebruiken vervolgens de dictionary om iedere rij (TableViewCell) op te vullen
  - Via het argument indexPath (**indexPath.row**) van deze methode, kan je achterhalen welke rij wordt behandeld

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"myCell"
        forIndexPath:indexPath];

    cell.textLabel.text = [self.books objectForKey:self.books.allKeys[indexPath.row]];
    cell.detailTextLabel.text = self.books.allKeys[indexPath.row];

    return cell;
}
```

### Eindresultaat



### Uitbreiding

Wat indien we een volgend scherm met **meer info** willen, na **tap** op een rij in TableView?

- Maak een segue tussen de rij (cel) in de TableView en het volgende scherm
- Voorzie een **prepareForSegue**-methode in de **TableViewController**-klasse
  - Hierin kunnen we informatie doorgeven naar het volgende scherm, gebaseerd op de geselecteerde cel
- De **TableView** heeft een methode **indexPathForSelectedRow**, die het **indexPath** van de geselecteerde rij weergeeft

```

- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    ItemViewController *itemVC = (ItemViewController *)[segue destinationViewController];
    //geselecteerde rij
    NSIndexPath *indexPath = [[self tableView] indexPathForSelectedRow];
    // we selecteren het juiste element uit de array en geven deze waarde
    // mee aan de destination \viewcontroller
    itemVC.item = self.items[indexPath.row];
}

```

## Location & Maps

**Location services:** Core Location framework

**Maps:** Map Kit framework → Aanzetten in projecteigenschappen > Capabilities > Maps

### Location Services

#### Locatie van de gebruiker

- **CLLocationManager** klasse
  - Centrale punt voor configuratie van wijze waarop locatie moet opgehaald worden, en om deze te verkrijgen
- Belangrijk: eerst **toegang** vragen om locatie services te gebruiken

#### Permissie

- Maak **CLLocationManager** object aan, en ken er een delegate aan toe
- Permissie opvragen via **requestWhenInUseAuthorization** of **requestAlwaysAuthorization** methode  
`[[self locationManager] requestWhenInUseAuthorization];`
- Voeg **NSLocationAlwaysUsageDescription** of **NSLocationWhenInUseUsageDescription** key toe aan de Info.plist van je project en plaats daarin de boodschap die getoond moet worden aan de gebruiker wanneer de vraag gesteld wordt om toegang te krijgen tot de gebruikerslocatie

#### Locatie?

- **CLLocationManager**: `startUpdatingLocation`
- Delegate van dit object: `locationManager:didUpdateLocations`

### Maps

1. Map toevoegen aan view
2. Map rapporteert alle interacties aan zijn delegate: **MKMapViewDelegate** protocol

#### **MKMapViewDelegate**

- - `mapView:didUpdateUserLocation:`
- Wanneer de Shows: User Location (Attribute Inspector) aanstaat, zal deze methode opgeroepen worden wanneer de locatie verandert; kan ook via het storyboard

## Gebied tonen

- MKCoordinateRegionMakeWithDistance
    - // bij iedere beweging passen we de getoonde regio aan, een gebied van 2km op 2km rond de locatie van de gebruiker
- ```
MKCoordinateRegion region =
MKCoordinateRegionMakeWithDistance(userLocation.coordinate,
2000, 2000);

[mapView setRegion:region animated:YES];
```

## Annotation

- Een Annotation object moet het MKAnnotation protocol implementeren
- Je moet properties voorzien zoals title, coordinate
- Tip: bekijk documentatie van dit protocol
- Toevoegen aan MKMapView door methode addAnnotation(s)

## Coördinaten

- Hoe voorzie je coördinaten voor annotation?
- Type: CLLocationCoordinate2D  
CLLocationCoordinate2D location;  
location.latitude = (double) 50.850833;  
location.longitude = (double) 4.346983;

## GPS-coördinaten

- Open Google Maps
- Klik met rechtermuisknop op de locatie of gebied op de kaart
- Selecteer “wat is hier”
- Coördinaten worden weergegeven in de infokaart

## Gestures

Eenvoudig gestures toevoegen via storyboard (**Object Library**)

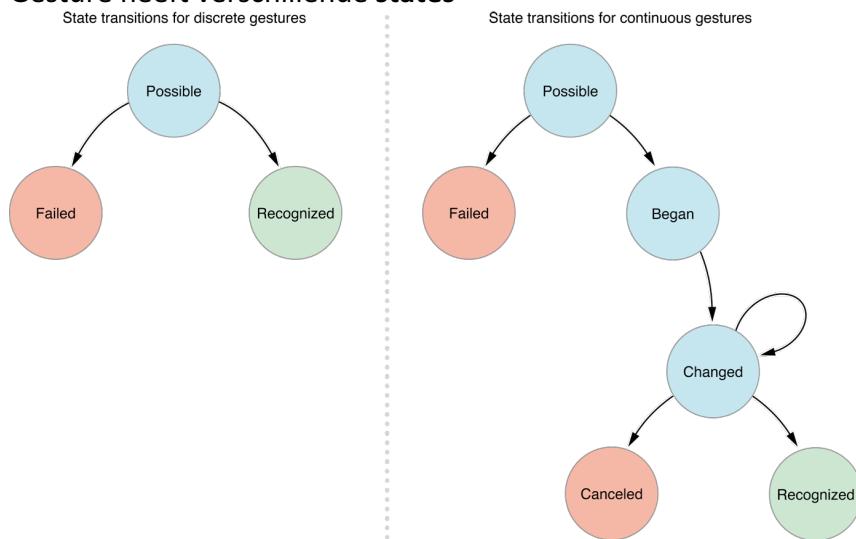
Vervolgens moet je een **actie koppelen** aan deze gesture

- Sleep naar je implementatiefile en voeg een **IBAction** (selector) toe

## Soorten gestures

- **Discrete gestures:** bestaan uit eenvoudige handelingen en zijn onmiddellijk afgelopen (bv. tap); triggeren één actie
- **Continuous gestures:** zijn complexer (multitouch) en spelen zich af over een langere tijd (pinching, long press); sturen meerdere actie boodschappen

## Gesture heeft verschillende states



Het is dus te adviseren om na te gaan wat de **staat** is van de gesture wanneer de actie opgeroepen wordt; controleer dus de `UIGestureRecognizerState`

```
- (IBAction)handleLongPress:(UILongPressGestureRecognizer *)sender {  
    if (sender.state == UIGestureRecognizerStateRecognized){
```

Je kan via de `UILongPressGestureRecognizer` de locatie opvragen waar de vinger zich bevindt

```
self.touchPoint = [sender locationInView:self.myMapView];
```

X en Y-coördinaat waar vinger terecht komt **omzetten** naar **map coördinaten**

```
CLLocationCoordinate2D location = [self.myMapView convertPoint:self.touchPoint toCoordinateFromView:self.myMapView];
```

## NSURLSession

### Kenmerken

- Klasse voorziet API om **inhoud te downloaden** door middel van http-protocol
- Klasse biedt veel methoden aan via **delegatie**; voorzien ondersteuning voor authenticatie, downloads in achtergrond, etc.
- Om klasse te gebruiken maakt je applicatie een **serie van sessies** aan
  - Sessie coördineert gerelateerde data transfer
- Zoals de meeste netwerk API's is de `NSURLSession` API **asynchroon**
  - Indien je een **default delegate object** gebruikt: onmiddellijk code voorzien voor de afhandeling van de ontvangen data wanneer transfer werd beëindigd
  - **Custom delegate object**: data manipuleren of verwerken vanaf het moment dat deze binnenkomt van de server

## Code voorbeeld

```
NSURLSession *session = [NSURLSession sharedSession];  
[[session dataTaskWithURL:[NSURL URLWithString:aUrl]  
completionHandler:^(NSData *data,  
NSURLResponse *response,  
NSError *error) {  
// handle response  
}]; resume];
```

## Blocks

### Kenmerken

- Stuk code dat kan uitgevoerd worden op later tijdstip
- Objective-C object
  - Kan doorgegeven worden als parameter of return waarde
- Alternatief voor delegatie, maar handige is dat alle gerelateerde code zich op één plaats bevindt

## Code voorbeeld

```
int multiplier = 7;  
We're declaring a variable "myBlock."  
The "^" declares this to be a block.  
This is a literal block definition,  
assigned to variable myBlock.  
  
int (^myBlock)(int) = ^(int num) { return num * multiplier; };
```

Annotations:

- myBlock is a block that returns an int.
- It takes a single argument, also an int.
- The argument is named num.
- This is the body of the block.

```
[theArray enumerateObjectsUsingBlock:^(id obj, NSUInteger idx,  
BOOL *stop){  
    NSLog(@"The object at index %d is %@",idx,obj);  
}];
```

- Toegang tot object, index en stop zonder code te schrijven = minder kans op fouten  
→ Snelheid: echter enkel merkbaar in complexe cases

## NSJSONSerialization

### Kenmerken

- Converteert JSON naar Foundation objecten (NSString, NSArray, NSDictionary, etc) en visa versa
- Alle data is steeds voorzien in vorm van key/value paren (Dictionary)
- Ook array's worden voorzien (herkenbaar aan [])

## Codevoorbeeld

### JSON

```
{  
  "employees": [  
    { "firstName":"John" , "lastName":"Doe" },  
    { "firstName":"Anna" , "lastName":"Smith" },  
    { "firstName":"Peter" , "lastName":"Jones" }  
  ]  
}
```

### Objective-C

```
NSError* error;  
NSDictionary* json = [NSJSONSerialization  
  JSONObjectWithData:responseData  
  options:kNilOptions  
  error:&error];  
  
NSArray* employees = [json objectForKey:@"employees"];  
  
 NSLog(@"%@", employees );  
  
NSDictionary* employee= [employees objectAtIndex:0];  
  
NSString* firstName = [employee objectForKey:@"firstName"];
```

### Probleem

- Ophalen van data via NSURLConnection gebeurt in een **aparte thread**, maar het updateen van de userinterface dient steeds via de **main thread** te gebeuren
- Ook indien gebruik gemaakt van delegate

### Oplossing

- **Opdracht toevoegen** aan **main thread** wanneer NSURLConnection de nodige gegevens heeft binnengehaald
- Block via **dispatch\_async** meegeven, zo wordt deze doorgestuurd naar de queue van de main thread

```
dispatch_async(dispatch_get_main_queue(), ^{  
  // do stuff  
});
```

### Onveilige communicatie – Opmerking

- iOS laat niet zomaar onveilige communicatie via het http-protocol toe
- Toevoegen aan **Info.plist**
  - Onder “Bundle OS Type code”
  - Add row “App Transport Security Settings”
    - “Allow Arbitrary Loads” = YES

## Core Data

**Core Data** = persistence framework

- Data opslaan in XML, binair of SQLite
- Data wordt beheerd door middel van objecten die entiteiten en hun relaties voorstellen

**Vereiste objecten:**

- Managed Object Model
  - Vergelijkbaar met database schema
  - Klasse die alle definities bevat van de entiteit die je wenst op de te slaan
- Persistent Store Coordinator
  - Vergelijkbaar met database connectie
  - Bevat naam en locatie van databank die je wilt gebruiken
- Management Object Context
  - Soort kladblok voor objecten die van de databank komen
  - Wanneer we objecten willen toevoegen, ophalen of verwijderen gebruiken we (meestal) de Managed Object Context

## Core Data en Xcode

Xcode helpt ons op weg: "Use Core Data" aanduiden bij het aanmaken van een project zal alle objecten die we net bespraken aanmaken

### .xcdatamodeld

- Automatisch voorzien
- Bestand openen zal een Visual Model Editor opstarten waarin we ons model kunnen aanmaken
- Via Xcode kunnen we ook bijhorende klassen genereren: Editor > Create NSManagedObject Subclass...

### ManagedObjectContext

Nodig in **elke viewcontroller** waarin we entiteiten wensen op te slaan, op te vragen en te verwijderen.

- Aangemaakt in AppDelegate, kan via dit object steeds opgevraagd worden
- Best practice: niet steeds opvragen bij AppDelegate, maar steeds doorgeven aan de viewcontroller

### Voorbeeld via AppDelegate

```
AppDelegate * appDelegate =  
[[UIApplication sharedApplication] delegate];  
  
NSManagedObjectContext * context = [appDelegate  
managedObjectContext];
```

### Voorbeeld best practice

In viewcontroller die nood heeft aan een ManagedObjectContext maak je een property om naar dit soort object te refereren, bij overgaan naar nieuwe view, geef je object mee

```

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    UINavigationController *navigationController =
(UINavigationController *)self.window.rootViewController;
    MyTableViewController *rootViewController = (MyTableViewController
*)[[navigationController viewControllers] objectAtIndex:0];
    rootViewController.managedObjectContext =
self.managedObjectContext;

    return YES;
}

```

## Beheren van entiteiten

### Aanmaken van een entiteit

```

Persoon *nieuwePersoon = [NSEntityDescription
    insertNewObjectForEntityForName:@"Persoon"
inManagedObjectContext:self.managedObjectContext];

// ...

NSError *error;
[self.managedObjectContext save:&error];

```

### Ophalen van een entiteit

```

NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];
NSEntityDescription *entity = [NSEntityDescription
    entityForName:@"Persoon"
inManagedObjectContext:context];

[fetchRequest setEntity:entity];

NSError *error;

NSArray *personen = [context executeFetchRequest:fetchRequest
error:&error];

```

### Verwijderen van een entiteit

```
[aContext deleteObject:aManagedObject];
```

## DIY

Maak een app waarmee je **persoonsgegevens** (naam, voornaam, telefoonnummer) kan opslaan.



## Internationalization & Localization

Software applicatie ontwerpen zodat hij kan aangepast worden aan **verschillende talen en regio's**

- Aanpassen van de taal
- Aangepaste weergave van datum, tijd en getallen

### Hoe?

#### Aandachtspunten

- Belangrijk: gebruik van **Auto Layout**, aangezien tekst verandert wanneer je taal selecteert
- **Tekst scheiden** van code door in apart bestand te plaatsen, zowel **tekst** in **klassen** als in **storyboard**
- Xcode gebruikt bestanden met extensie **".strings"** voor iedere ondersteunde taal

**File aanmaken:** File > New > iOS > Resource > Strings File

**Naam:** Localizable, andere naam vermijden anders steeds moeten vermelden

- Formaat: "KEY" = "CONTENT";

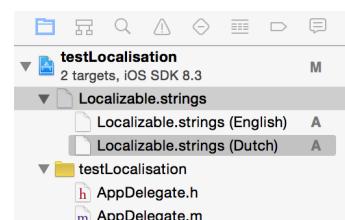
#### Klasse

```
mijnLabel.text = [NSString stringWithFormat:@"Hi %@", persoon.naam];
⇒ mijnLabel.text = [NSString stringWithFormat:NSLocalizedString(@"Hi %@", nil),
persoon.naam];
```

#### Taal toevoegen

Selecteer het project in "Project Navigator" > Project > **Localizations** > Add

- Aangeven dat we Localized.strings willen localiseren: File Inspector > Selecteer Localize > Selecteer talen in welke je het bestand wilt aanbieden
- Resultaat = verschillende versies van dit bestand
- Nu kan je verschillende talen aanbieden



**Testen in simulator:** Settings - General – Language & Region – Nederlands

#### Getallen

- Om getallen te formateren gebruik je **NSNumberFormatter**
- NSNumberFormatter past de weergave van getallen steeds aan aan je regio
- **Testen in simulator:** Settings - General – Language & Region – Region Formats

```
NSNumberFormatter *numberFormatter = [[NSNumberFormatter alloc] init];
[numberFormatter setNumberStyle:NSNumberFormatterDecimalStyle];
NSString *nummer = [numberFormatter stringFromNumber:@(250000)];
self.mijnLabel.text = [NSString stringWithFormat:NSLocalizedString(@"Totaal: %@", nil),
nummer];
```

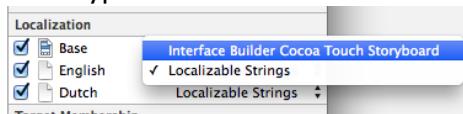
## Storyboards

Tekst van UI-elementen kan via **code** aangepast worden,  
maar ook via **gelokaliseerde tekstbestanden**



Twee opties om deze tekstbestanden te **refreshen**

- Event type veranderen



- Terminal: `ibtool MainStoryboard.storyboard --generate-strings-file file_name.strings`

## Afbeeldingen

```
[_imageView setImage:[UIImage imageNamed: NSLocalizedString(@"imageName", nil)]];
```