Praktikum Entwicklung von Mediensystemen mit iOS

WS 2011

Prof. Dr. Michael Rohs
michael.rohs@ifi.lmu.de
MHCI Lab, LMU München

Today

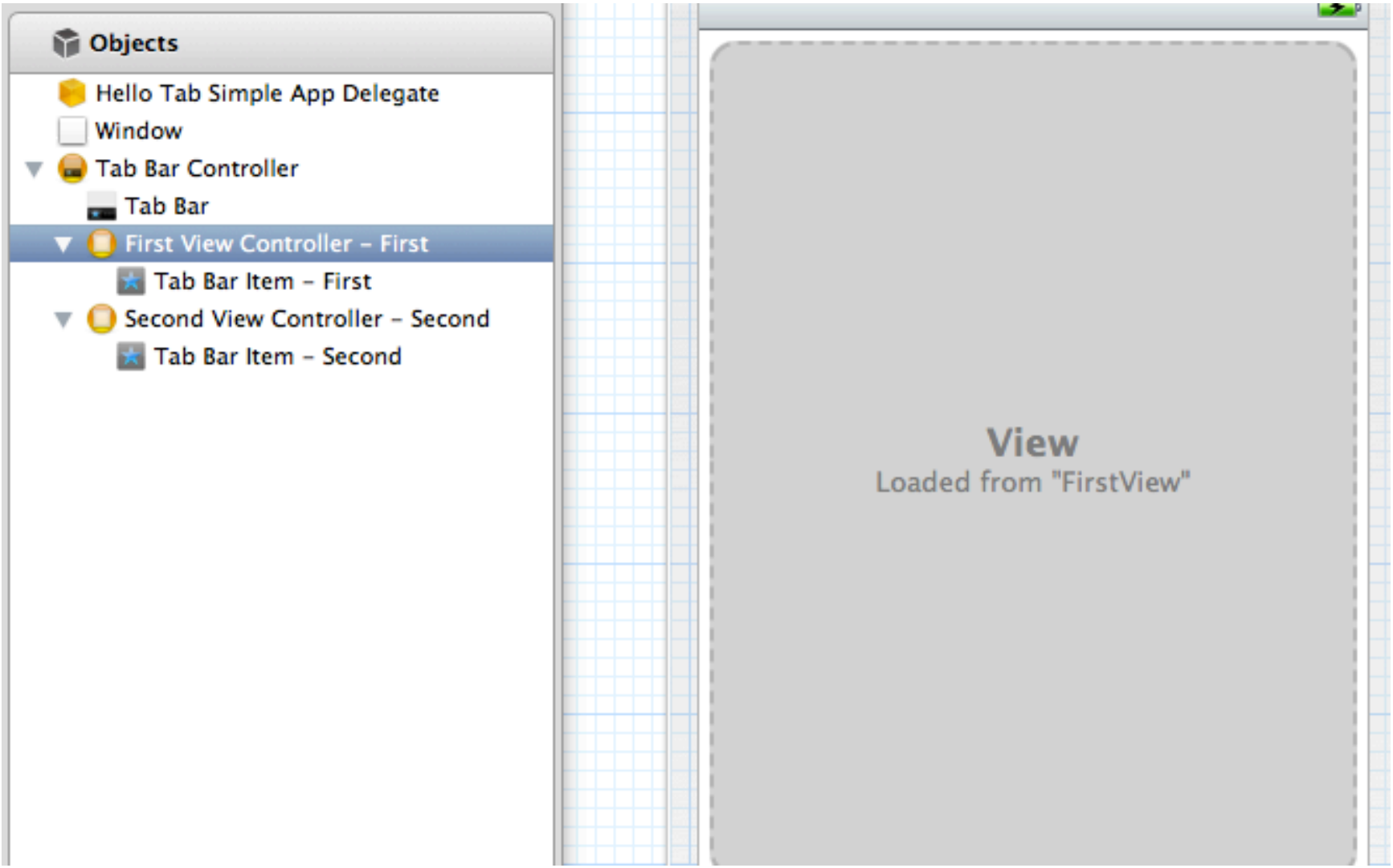
- Saving data
- Networking
- Location
- Sensors

- Exercise 2

Timeline

#	Date	Topic
1	19.10.2011	Introduction and overview of iOS
2	26.10.2011	App architecture, touch input, saving data
3	2.11.2011	Location, networking, sensors
4	16.11.2011	Interviews, storyboarding; brainstorming
5	30.11.2011	Paper prototyping test, start of software prototype
6	14.12.2011	Heuristic evaluation of software prototype
7	11.1.2012	Think-aloud user study
8	25.1.2012	Completion of software prototype
9	1.2.2012	Final presentation

Exercise 2: Multi Tab Application



DATA MANAGEMENT

Accessing Application Directories

- Application sandbox: can only access own app folder

```
NSString *homeDir = NSHomeDirectory();
NSString *tmpDir = NSTemporaryDirectory();
NSArray *paths = NSSearchPathForDirectoriesInDomains
(NSDocumentDirectory, NSUserDomainMask, YES);
path = [paths objectAtIndex:0];
```

- Accessing data bundled as an application resource

```
NSString *fileName = [homeDir
stringByAppendingPathComponent:@"/Test.app/mydata.dat"];
```

Loading and Saving Binary Data

- NSData is a container for bytes
- Loading arbitrary binary data

```
NSData *d = [[NSData alloc] initWithContentsOfFile:fileName];
NSMutableData *m = [NSData dataWithContentsOfFile:fileName];
```

- Accessing the data

```
const char* b = [d bytes]; // d is immutable → cannot be modified
char* c = [m mutableBytes]; // m is mutable → can be modified
```

- Saving arbitrary binary data

```
[d writeToFile:fileName atomically:YES];
```

- Appending to mutable data object

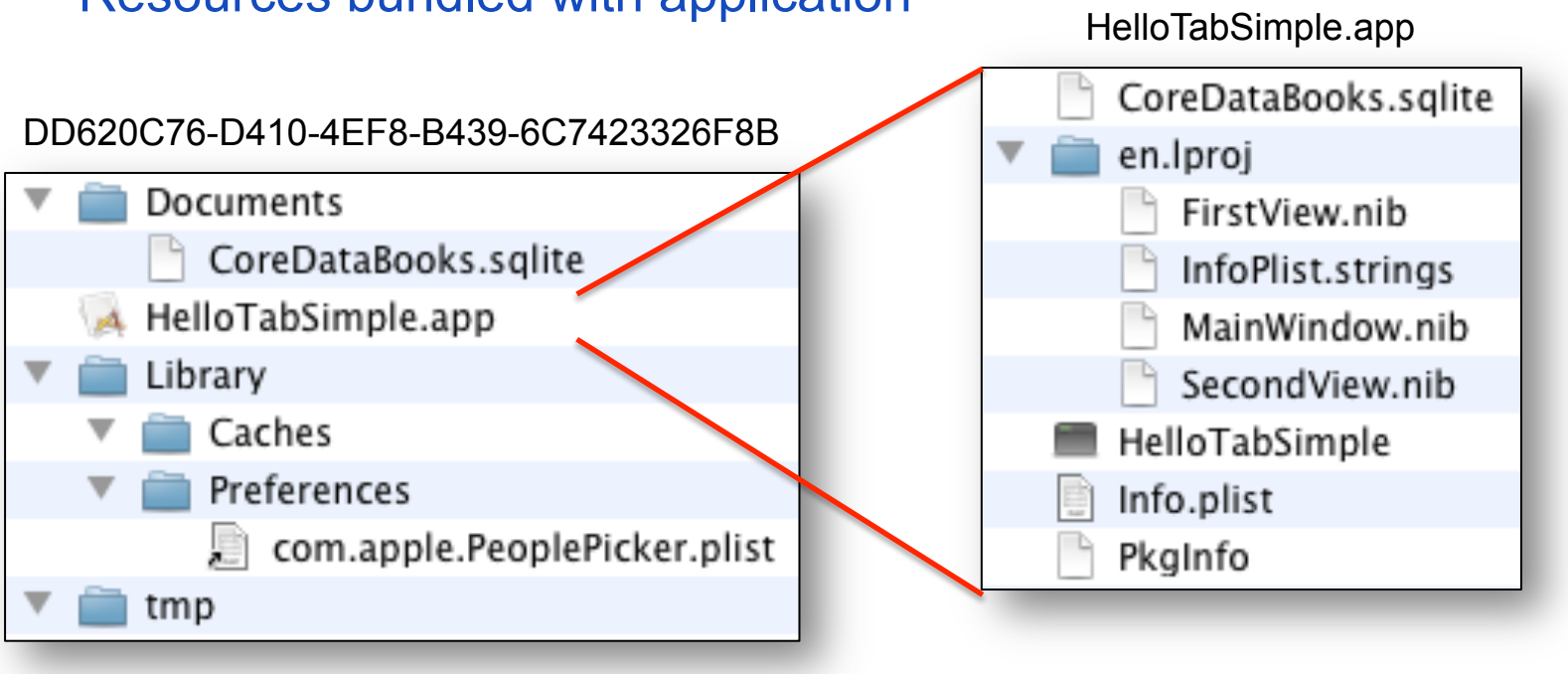
```
[m appendBytes:myBytes length:myBytesCount];
```

Binary Loading & Saving Code Snippet

```
NSString *fileName = @"/.../myTestFile.dat";
NSData *d = [[NSData alloc] initWithContentsOfFile:fileName];
const char* b = [d bytes];
// use the data, cannot modify
[d release];
NSMutableData *m = [[NSMutableData alloc]
                                initWithContentsOfFile:fileName];
char* c = [m mutableBytes];
c[0] = 42; // modify the data (direct access to data)
char *myBytes = "123";
int myBytesCount = strlen(myBytes);
[m appendBytes:myBytes length:myBytesCount];
[m writeToFile:fileName atomically:YES];
[m release];
```

Directory Structure of an App

- Each app has a unique identifier
  - Identifier is name of directory in “Applications” directory
- .app is a directory itself
  - Cannot be modified (signed)
  - Resources bundled with application



Copying Resources to Documents Directory

- Copying a resource file to documents directory

```
NSArray *paths = NSSearchPathForDirectoriesInDomains
                        (NSDocumentDirectory, NSUserDomainMask, YES);
NSString *documentPath = ([paths count] > 0) ? [paths objectAtIndex:0] : nil;

NSString *filePath = [documentPath stringByAppendingPathComponent:
                            @"CoreDataBooks.sqlite"];

NSFileManager *fileManager = [NSFileManager defaultManager];
if (![fileManager fileExistsAtPath:filePath]) {
    NSString *defaultFilePath = [[NSBundle mainBundle]
                            pathForResource:@"CoreDataBooks" ofType:@"sqlite"];
    if (defaultFilePath) {
        [fileManager copyItemAtPath:defaultFilePath toPath:filePath error:NULL];
    }
}
```

Loading XML Data

- XML data and property lists for structured data

```
NSDictionary *d = [NSDictionary dictionaryWithContentsOfFile:f];
```

- Predefined elements dict, array, string, key, integer, etc.
- Example (a dictionary containing an array of dictionaries)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>images</key>
        <array>
                <dict>
                        <key>title</key><string>My Image Title</string>
                        <key>image</key><string>MyImage.png</string>
                </dict>
                <dict>
                        <key>title</key><string>Another Title</string>
                        <key>image</key><string>AnotherImage.png</string>
                </dict>
        </array>
</dict>
</plist>
```

Loading and Saving Object Hierarchies

- Declaring objects as archiveable by implementing NSCopying protocol
  - initWithCoder, encodeWithCoder
- Handle archiving in these methods
  - All objects handled by coder need to conform to NSCopying
- NSKeyedArchiver to save object hierarchy
- NSKeyedUnarchiver to load object hierarchy

Declaring Classes as Archiveable

```
@interface MyClass : NSObject <NSCopying> {
    NSString *lastName;
    NSMutableArray *firstNames;
}
- (id) initWithCoder:(NSCoder *)decoder {
    self = [super init];
    self.lastName = [decoder decodeObjectForKey:@”lastName"];
    self.firstNames = [decoder decodeObjectForKey:@”firstNames"];
    return self;
}
- (void) encodeWithCoder:(NSCoder *)encoder {
    [encoder encodeObject:lastName forKey:@”lastName"];
    [encoder encodeObject:firstNames forKey:@"firstNames"];
}
```

Saving Object Hierarchies

- Archiving (simple version, one root)

```
[NSKeyedArchiver archiveRootObject:myRoot toFile:myFile];
```

- Archiving (complex version, multiple roots)

```
NSMutableData *data = [[NSMutableData alloc] init];
NSKeyedArchiver *archiver = [[NSKeyedArchiver alloc]
                                    initForWritingWithMutableData:data];
[archiver encodeObject:myRoot1 forKey:@”myRoot1"];
[archiver encodeObject:myRoot2 forKey:@”myRoot2"];
[archiver finishEncoding];
[data writeToFile:myFile atomically:YES];
[archiver release];
[data release];
```

Loading an Object Hierarchy

- Unarchiving an object hierarchy

```
self.object = [NSKeyedUnarchiver unarchiveObjectWithFile:fileName];
```

- Root object needs to be retained after unarchiving

  In the example above it is a retained property

CoreData

- Creation of data models (entities, relationships)
- Persistent storage of object graphs
- Undo management
- Version management

- Good for complex data models
- Data storage can be database or flat file

Core Data Stack

- Persistent storage (e.g. database)
- Object store maps records to objects
- Store coordinator aggregates stores
- Managed object model contains entity descriptions
- Managed object context contains managed objects
- Managed objects are part of the persistent object graph

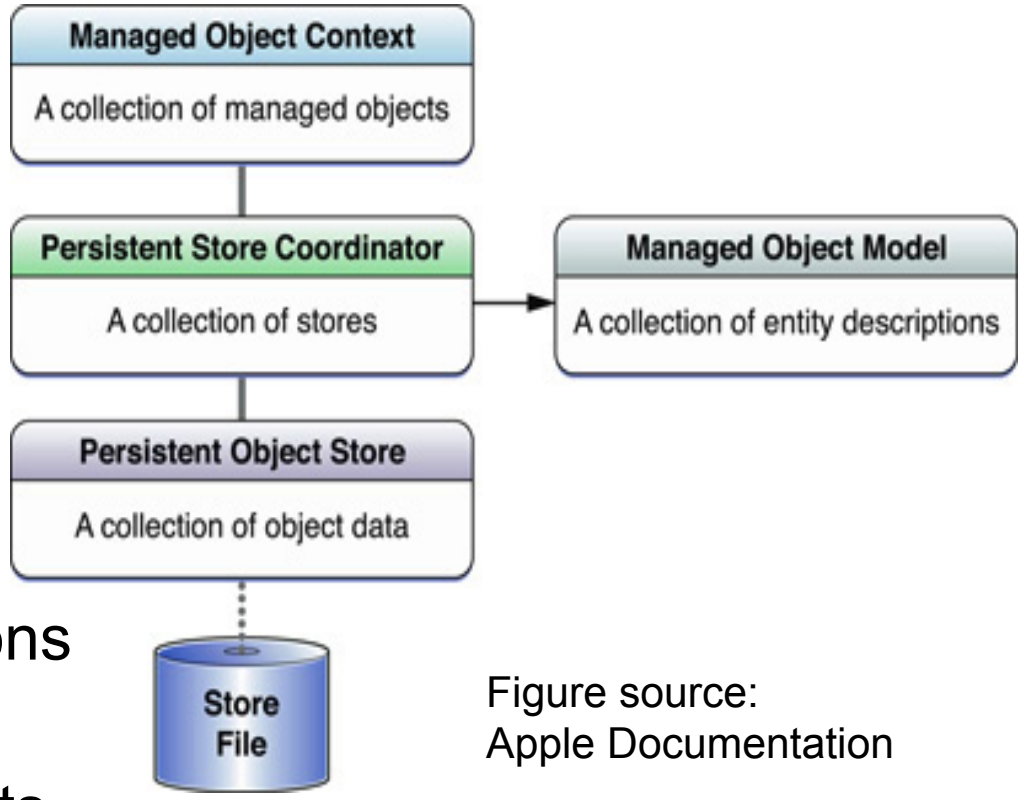


Figure source:
Apple Documentation

Managed Object

- Objective C object that represents a data record of the persistent store (e.g. row of a database)
- Has a reference to its entity description
- Part of a managed object context

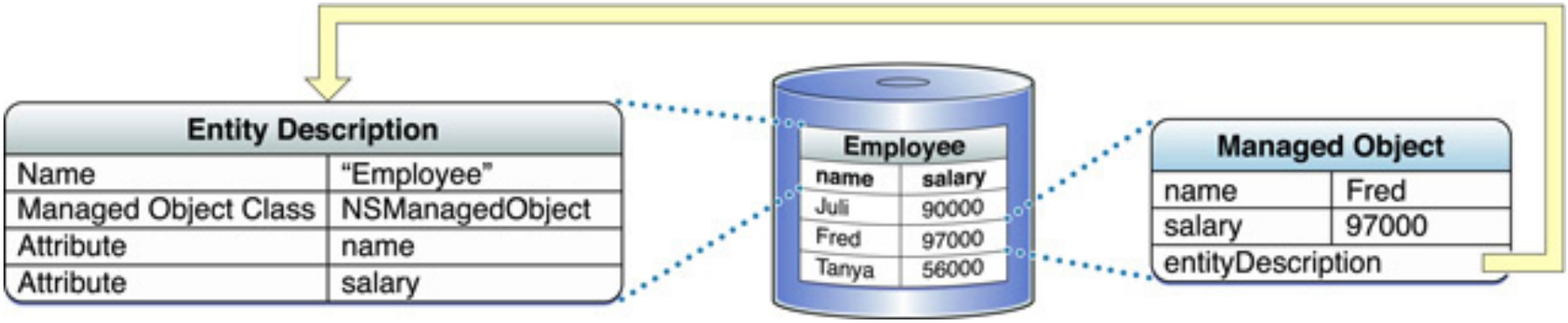
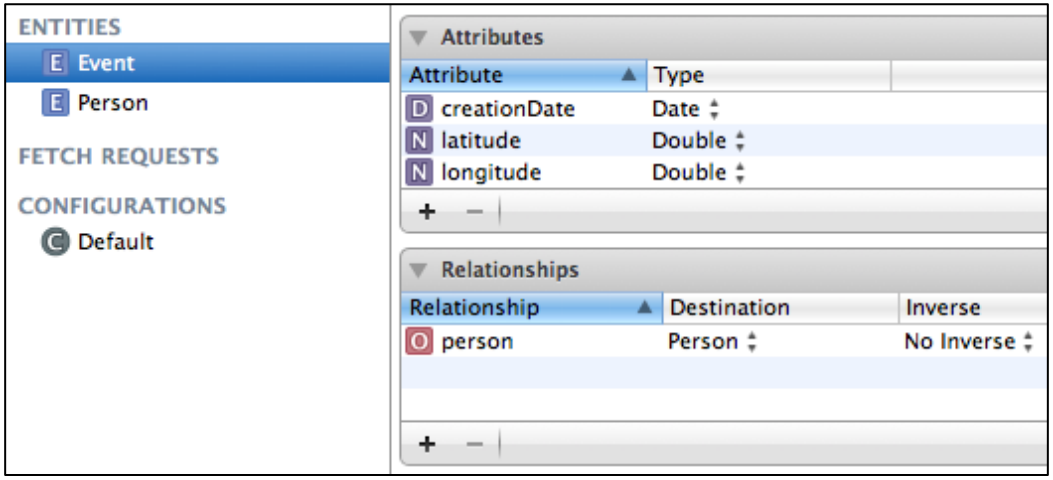
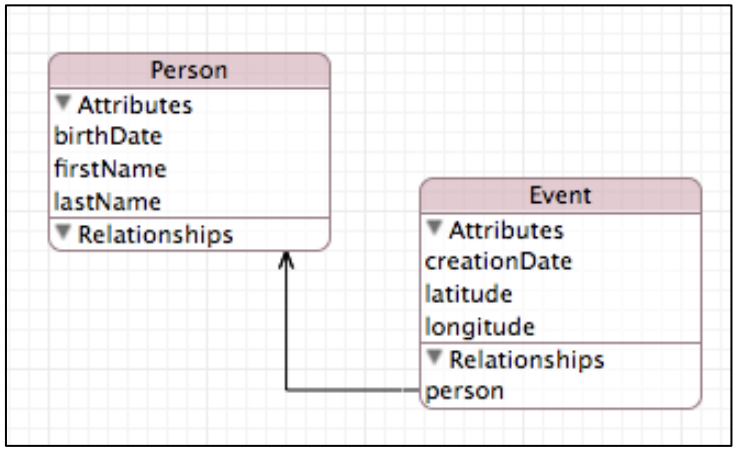


Figure source: Apple Documentation

Defining the Object Model

- Xcode data model





- Automatic creation of model class

```
@interface Event : NSManagedObject { }
@property (nonatomic, retain) NSNumber * latitude;
@property (nonatomic, retain) NSNumber * longitude;
@property (nonatomic, retain) NSDate * creationDate;
@property (nonatomic, retain) Person * person;
@end
```

Creating NSPersistentStoreCoordinator

```
NSURL *storeUrl = [NSURL fileURLWithPath:
    [[self applicationDocumentsDirectory]
        stringByAppendingPathComponent:@"Locations.sqlite"]];

managedObjectModel = [[NSManagedObjectModel
                              mergedModelFromBundles:nil] retain];

persistentStoreCoordinator = [[NSPersistentStoreCoordinator alloc]
    initWithManagedObjectModel: [self managedObjectModel]];

[persistentStoreCoordinator
        addPersistentStoreWithType:NSSQLiteStoreType
                     configuration:nil
                               URL:storeUrl
                           options:nil
                             error:nil];
```

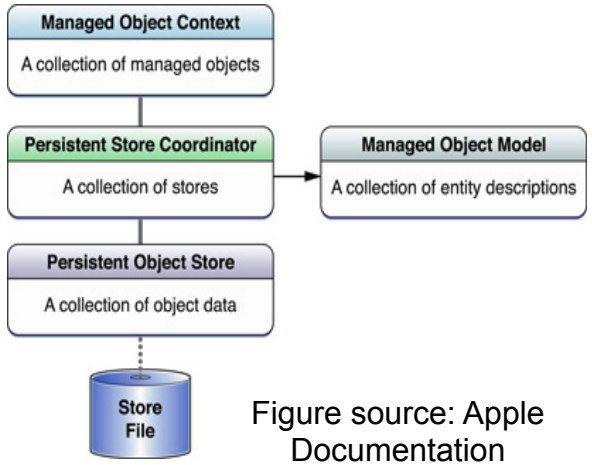


Figure source: Apple Documentation

NSManagedObjectContext

- Creation

```
managedObjectContext = [[NSManagedObjectContext alloc] init];
[managedObjectContext setPersistentStoreCoordinator: coordinator];
```

- Persistent saving of object graph

```
NSError *error;
if (![[self managedObjectContext]
            save:&error]) {
    // handle error
}
```

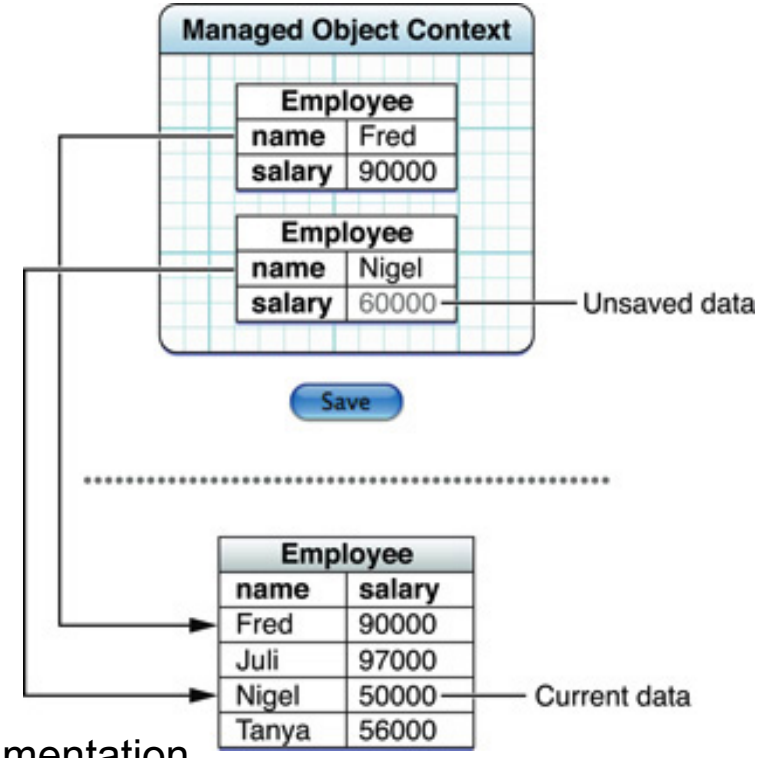


Figure source: Apple Documentation

Loading objects from store

- Fetch requests

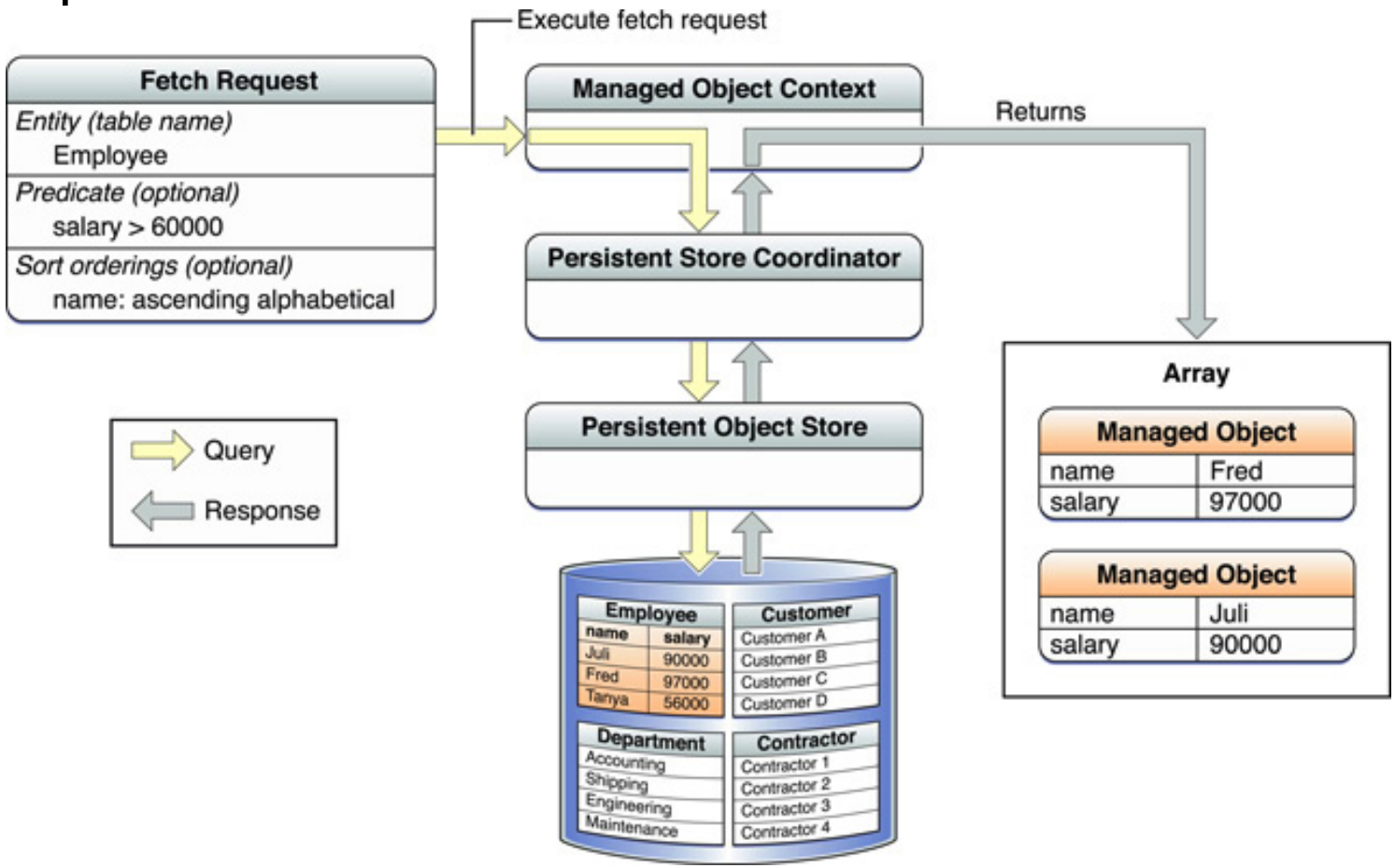


Figure source: Apple Documentation

Loading objects from store

from Apple example "Locations", Core Data Tutorial for iPhone OS

```
NSFetchRequest *request = [[NSFetchRequest alloc] init];
NSEntityDescription *entity = [NSEntityDescription entityForName:@"Event"
                                        inManagedObjectContext:managedObjectContext];
[request setEntity:entity];

NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc]
initWithKey:@"creationDate" ascending:NO];
NSArray *sortDescriptors = [NSArray arrayWithObject:sortDescriptor];
[request setSortDescriptors:sortDescriptors];
[sortDescriptor release];

NSError *error = nil;
NSArray *results = [managedObjectContext
                                          executeFetchRequest:request error:&error];
NSMutableArray *mutableResults = [results mutableCopy];
```

Removing an object from the Context

- “release” does not remove managed object
- need to explicitly remove from context and commit

```
[managedObjectContext deleteObject:eventToDelete];
[managedObjectContext save:nil];
```

- also remove from local data structure and GUI element

```
[eventsArray removeObjectAtIndex:indexPath.row];
[tableView deleteRowsAtIndexPaths:[NSArray
                    arrayWithObject:indexPath]
                  withRowAnimation:YES];
```

Creating a new managed object

```
Event *event = (Event *)[NSEntityDescription
                    insertNewObjectForEntityForName:@"Event"
                    inManagedObjectContext:managedObjectContext];

[event setLatitude:<my latitude>];
[event setLongitude:<my longitude>];
[event setCreationDate:[NSDate date]];

// Save persistently
[managedObjectContext save:nil];

// Insert into array
[eventsArray insertObject:event atIndex:0];
```

LOCATION & SENSORS

Location

- Location manager provides location information

```
CLLocationManager *locationManager;
```

- Configuration

```
locationManager = [[CLLocationManager alloc] init];
locationManager.delegate = self;
locationManager.desiredAccuracy = [[setupInfo
objectForKey:@"SetupInfoKeyAccuracy"] doubleValue];
[locationManager startUpdatingLocation];
```

- Update location

```
(void)locationManager:(CLLocationManager *)manager
didUpdateToLocation:(CLLocation *)newLocation fromLocation:
(CLLocation *)oldLocation { … }
```

- Details: LocateMe example application

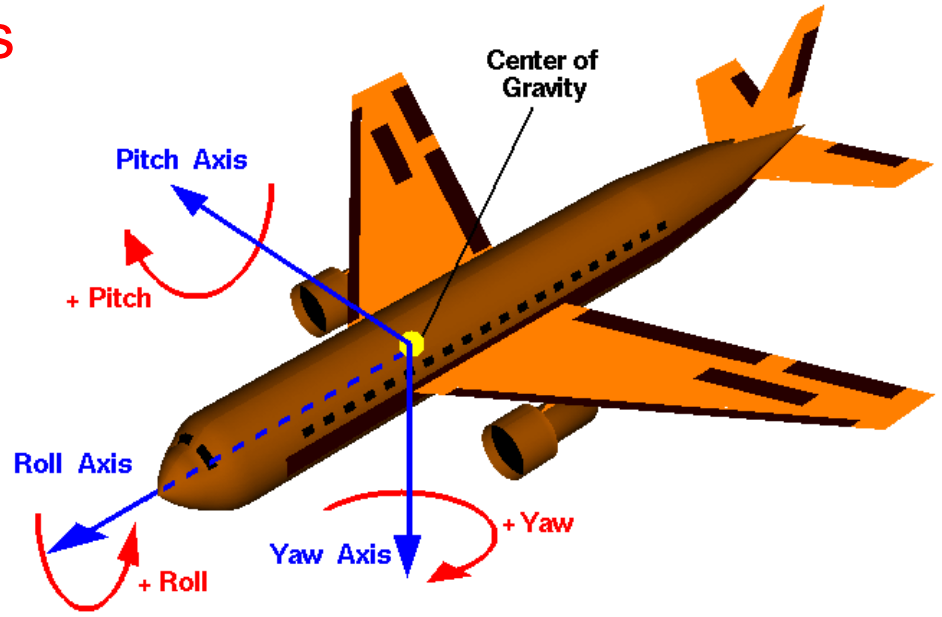
Latitude and Longitude

- Location manager state

  manager.locationServicesEnabled
- Location manager desiredAccuracy constants
  - kCLLocationAccuracyBestForNavigation
  - kCLLocationAccuracyBest
  - kCLLocationAccuracyNearestTenMeters
  - kCLLocationAccuracyHundredMeters
  - kCLLocationAccuracyKilometer
  - kCLLocationAccuracyThreeKilometers
- CLLocation
- CLLocationDegrees
- CLLocationDistance (from location … to location)

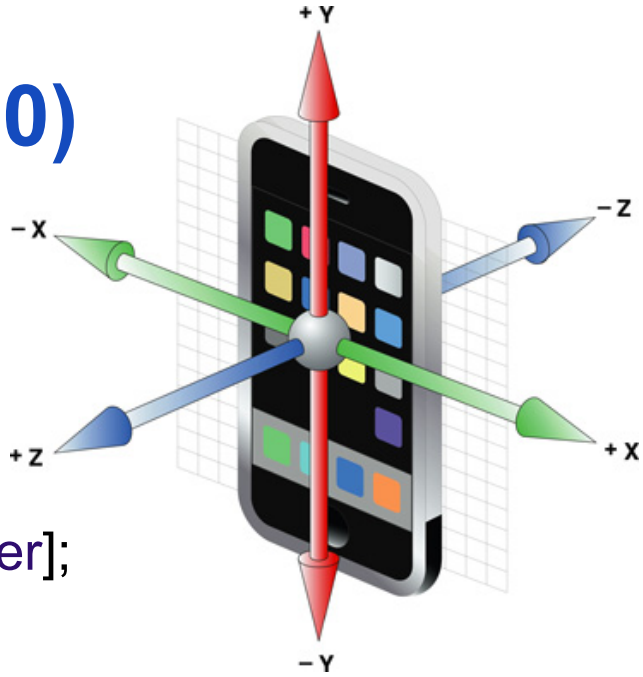
Sensor Coordinate Systems

"Orientation Sensor" = Accelerometer + Magnetometer



Axis	Zero position	Range	Details
Yaw	north	-π.. π	0=north, -π/2=east, ±π=south, π/2=west
Pitch	horizontal	-π/2.. π/2	0=horizontal, >0: up, <0: down
Roll	horizontal	-π.. π	0=horizontal, >0: right, <0: left

Accelerometer (since iOS 2.0)



Source: iOS Documentation, Apple

- UIAccelerometer

```
UIAccelerometer *acc;
```

- Configure accelerometer

```
acc = [UIAccelerometer sharedAccelerometer];
acc.updateInterval = 1.0; // sec
acc.delegate = self;
```

- Receive updates

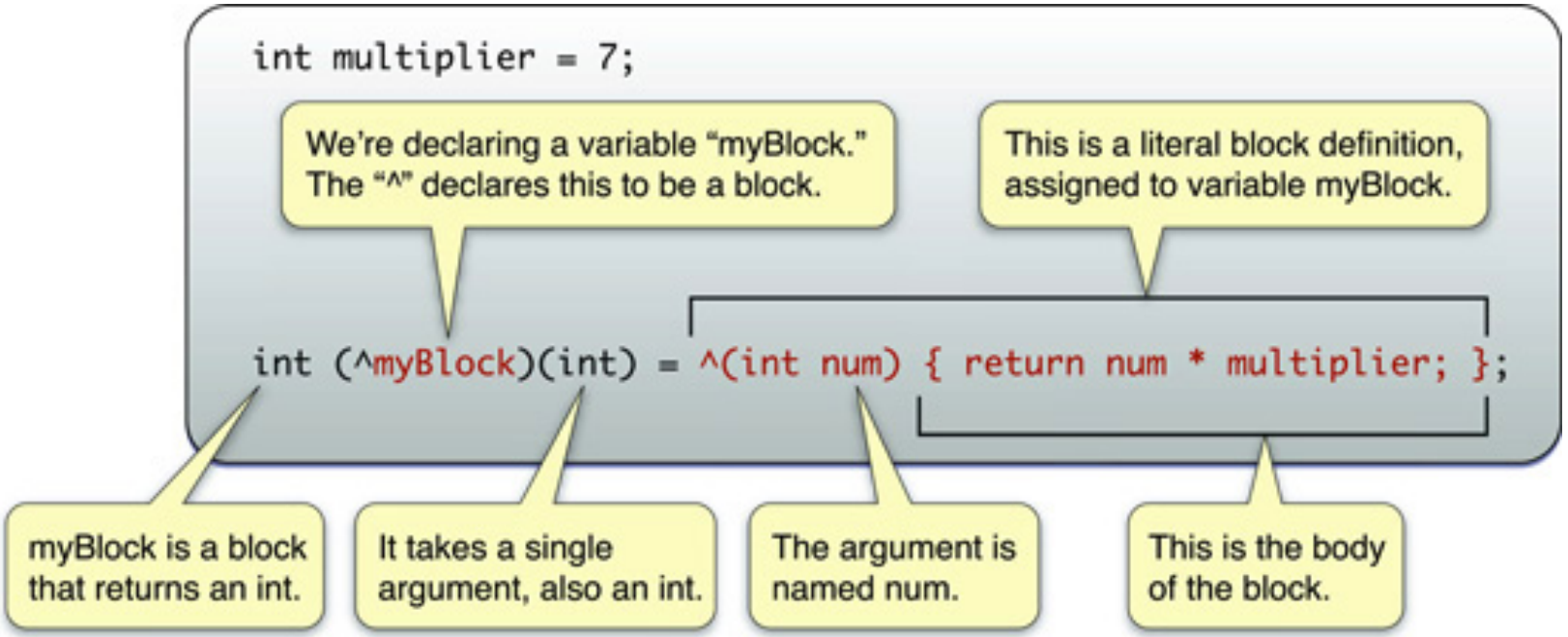
```
- (void)accelerometer:(UIAccelerometer *)a
        didAccelerate:(UIAcceleration *)acceleration
{
    NSLog(@"%.3f, %.3f, %.3f, %f",
          acceleration.x, acceleration.y,
          acceleration.z, acceleration.timestamp);
}
```

Motion Sensors (since iOS 4.0)

- Core Motion Framework
- CMMotionManager provides access to
  - raw accelerometer data (accelerometer)
  - raw rotation-rate data (gyroscope)
  - processed device-motion data (fused sensor data)
- Receiving sensor data
  - Specific intervals
    - accelerometerUpdateInterval
    - startAccelerometerUpdatesToQueue:withHandler:
  - On demand
    - startAccelerometerUpdates
    - accelerometerData

Code Blocks

- Blocks: conceptually similar to function pointers in C
  - Can access variables from function in which they are defined
- Passing a code block as argument



Source: iOS Documentation, Apple

- Calling: [someObject someFunction:myBlock];

Getting Accelerometer Values

- Create and configure motion manager

```
motionManager = [[CMMotionManager alloc] init];
motionManager.accelerometerUpdateInterval = 1.0; // sec
```

- Register handler and start updates

```
void (^accHandler)(CMAccelerometerData*, NSError*) =
        ^(CMAccelerometerData *accData, NSError *error)
    {
        NSLog(@"%f, %f, %f", accData.acceleration.x,
        accData.acceleration.y, accData.acceleration.z);
    };
 [motionManager startAccelerometerUpdatesToQueue:
      [NSOperationQueue mainQueue] withHandler:accHandler];
```

- Stop updates

```
[motionManager stopAccelerometerUpdates];
```

should not use main queue

Getting Accelerometer Values

- Create new queue

```
queue = [[NSOperationQueue alloc] init];
```

- Register handler and start updates

```
void (^accHandler)(CMAccelerometerData*, NSError*) =
        ^(CMAccelerometerData *accData, NSError *error)
    {
        NSLog(@"%f, %f, %f", accData.acceleration.x,
        accData.acceleration.y, accData.acceleration.z);
    };
[motionManager startAccelerometerUpdatesToQueue:queue
                                    withHandler:accHandler];
```

- Need to use main queue to update UI

Update UI on Main Queue

```
void (^accHandler)(CMAccelerometerData*, NSError*) =

^(CMAccelerometerData *accData, NSError *error)
{
    dispatch_async(dispatch_get_main_queue(), ^{
        NSString *s;
        s = [NSString stringWithFormat:@"%.3f", accData.acceleration.x];
        labelX.text = s;
        s = [NSString stringWithFormat:@"%.3f", accData.acceleration.y];
        labelY.text = s;
        s = [NSString stringWithFormat:@"%.3f", accData.acceleration.z];
        labelZ.text = s;
    });
};
```

Operation Queues

- NSOperationQueue objects store and execute NSOperation objects
- Operation queues
  - Store and execute operation objects
  - Provide their own threads to execute operations
  - Encapsulate threads
- Operations
  - Abstract base classes for tasks to perform
  - Have priorities
  - One operation may depend on completion of other operations

NETWORKING

Synchronous Download of Data

- Waits until data completely downloaded

```
- (UIImage*) imageFromUrlString:(NSString*)urlString;
{
    NSURL *url = [NSURL URLWithString:urlString];
    NSData *data = [NSData dataWithContentsOfURL:url];
    UIImage *image = [UIImage imageWithData:data];
    return image;
}
```

- Variant: specify options (e.g. caching) and error variable

```
NSDataReadingOptions o = NSDataReadingUncached;
NSError *e = nil;
NSData *data = [NSData dataWithContentsOfURL:url options:o error:&e];
    UIImage *image = [UIImage imageWithData:data];
    if (e) {...}
```

Synchronous Downloads with NSURLConnection

- Synchronous request with error response

```
NSMutableURLRequest *req = [NSMutableURLRequest
                                    requestWithURL:url];
NSURLResponse *res = nil;
NSError *e = nil;
NSData *data = [NSURLConnection sendSynchronousRequest:req
                                    returningResponse:&res error:&e];
```

- NSURLResponse contains content length, MIME type, etc.

Asynchronous Downloads with NSURLConnection

- Put request on a second thread

```
[NSThread detachNewThreadSelector:@selector(downloadImage:)
                         toTarget:self withObject:urlString];
```

- Update GUI on main (GUI-safe) thread

```
[imageView performSelectorOnMainThread:@selector(setImage:)
                            withObject:image waitUntilDone:NO];
```

MEMORY MANAGEMENT & INSTRUMENTS

Reference Counting

- Object reference life cycle:

```
myobject = [[MyClass alloc] init];          // reference count = 1 after alloc
[myobject retain];       // increment reference count (retainCount == 2)
[myobject release];      // decrement reference count (retainCount == 1)
[myobject release];      // decrement reference count (retainCount == 0)
// at this point myobject is no longer valid, memory has been reclaimed
[myobject someMethod]; // error: this will crash!
```

- Can inspect current reference count:

```
NSLog(@"retainCount = %d", [textField retainCount]);
```

- Can autorelease (system releases at some point in future)

```
[myobject autorelease];
```

Used when returning objects from methods.

Rules

- Memory rule: You are responsible for objects you allocate or copy (i.e. “allocate” or “copy” is some part of the name)!

- Not responsible:

```
NSData *data = [NSData dataWithContentsOfFile:@"file.dat"];
```

- Responsible:

```
NSData *data = [[NSData alloc] initWithContentsOfFile:@"file.dat"];
```

- Responsible:

```
NSData *data2 = [data copy];
```

- Never release objects you are not responsible for!

Objective C - Class

In .h file:

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

@interface Employee : NSObject
{ //Instance vars here
    NSString *name;
    int salary;
    int bonus;
}
// methods outside curly brackets
-  (void)setSalary:(int)cash withBonus:(int)extra
@end
```

Objective C Properties

- .h file:

```
@interface MyDetailViewController : UIViewController {
        NSString *labelText;
}
@property (nonatomic, retain) NSString *labelText;
@end
```

- .m file:

```
@synthesize labelText;
        -(void)someMethod {
        self.labelText = @”hello”;
}
```

creates accessor methods: setLabelText (retains/releases) and getLabelText.

dot-syntax means: use property’s setLabelText accessor method, will retain the object

equivalent to [self setLabelText:@”hello”];

Implicit Setter/Getter Accessor Methods

- .h file: @property (nonatomic, retain) NSString *labelText;
- .m file: @synthesize labelText;
- Automatic creation of accessor methods:

```
- (void) setLabelText:(NSString *)newLabelText {
    [labelText release];
    labelText = newLabelText;
    [labelText retain];
}
- (NSString*) getLabelText {
    return labelText;
}
```

decrement reference counter on old object (if any)

increment reference counter on new object (if any)

- Properties are accessible from other classes, data members only if declared @public

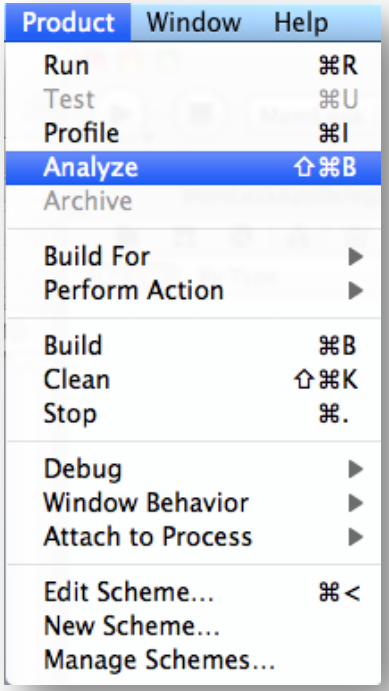
Property Attributes

- Writability: readwrite (default), readonly
- Setter semantics: assign, retain, copy
- Atomicity: atomic (default), nonatomic

- “readonly” means only a getter, but no setter accessor method is generated by @synthesize

Analyzing Code

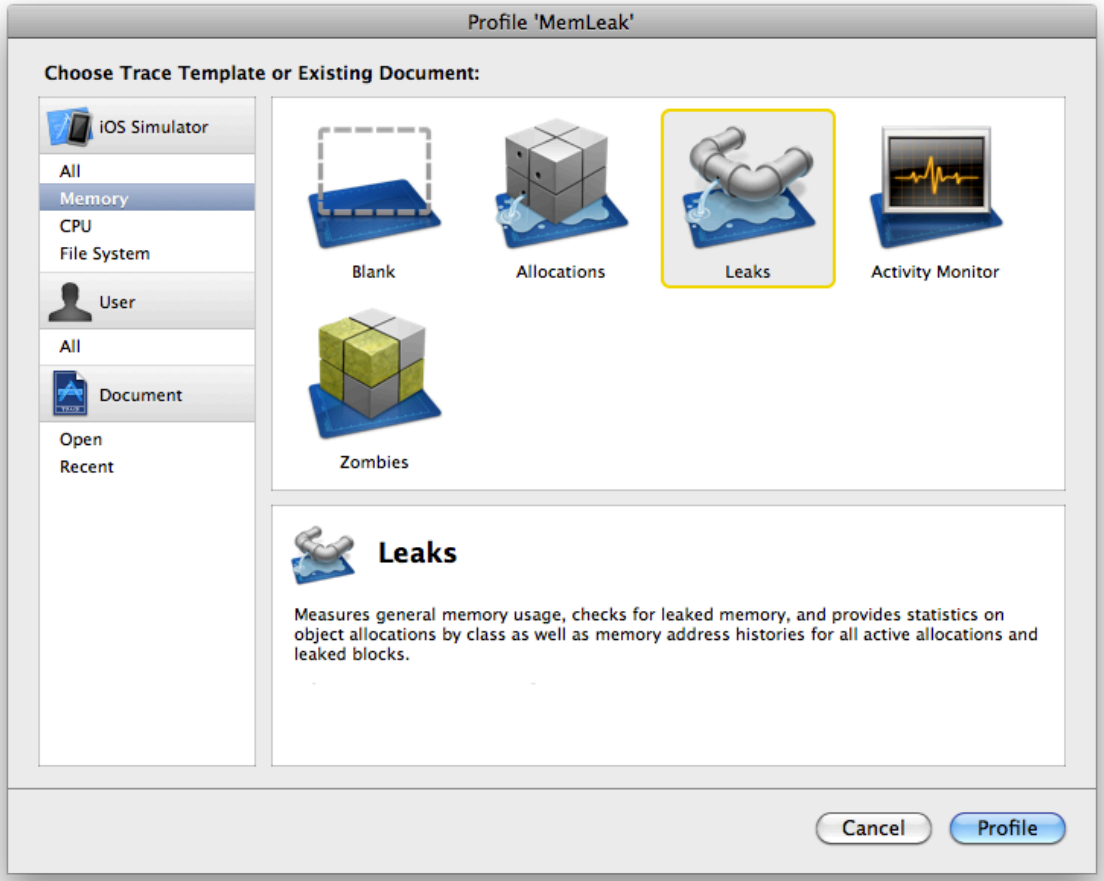
- Xcode static analysis for simple problems

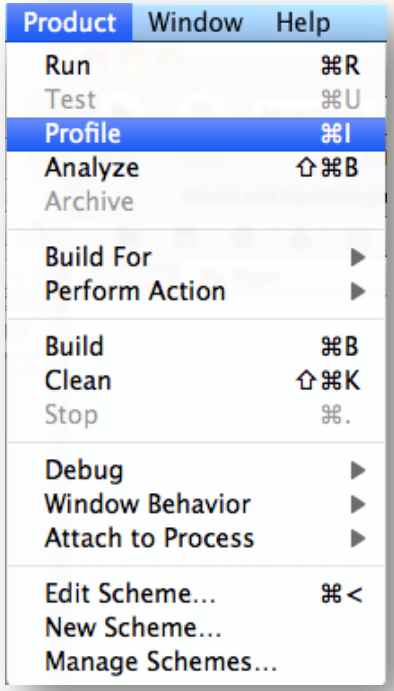



```
    NSString *s = [[NSString alloc] initWithFormat:@"%Number is %d", 123];

    NSLog(@"%@", s);

    return YES;                    Potential leak of an object allocated on line 28 and stored into 's'
}
```

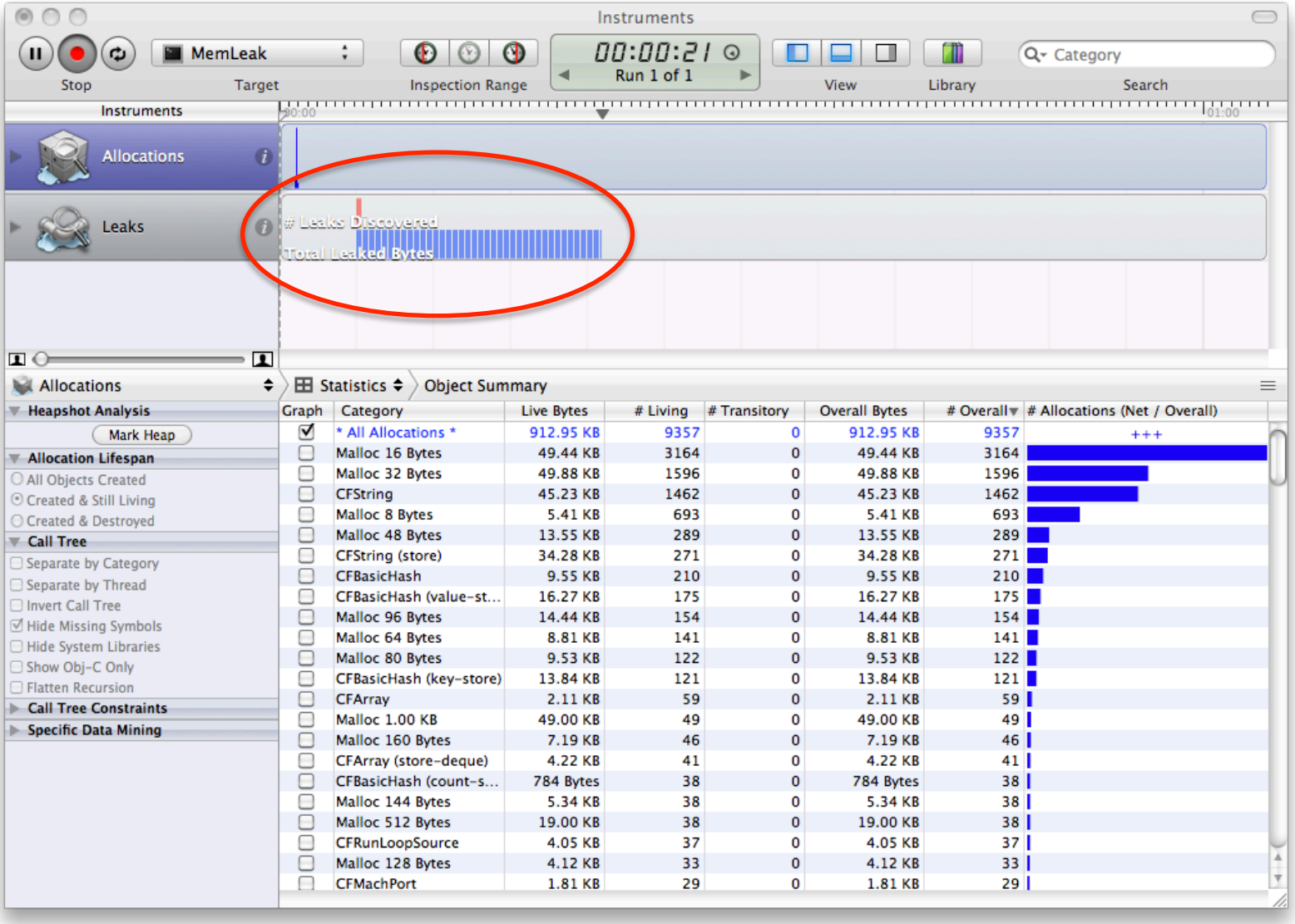
Profiling Code

- Analyzing runtime behavior



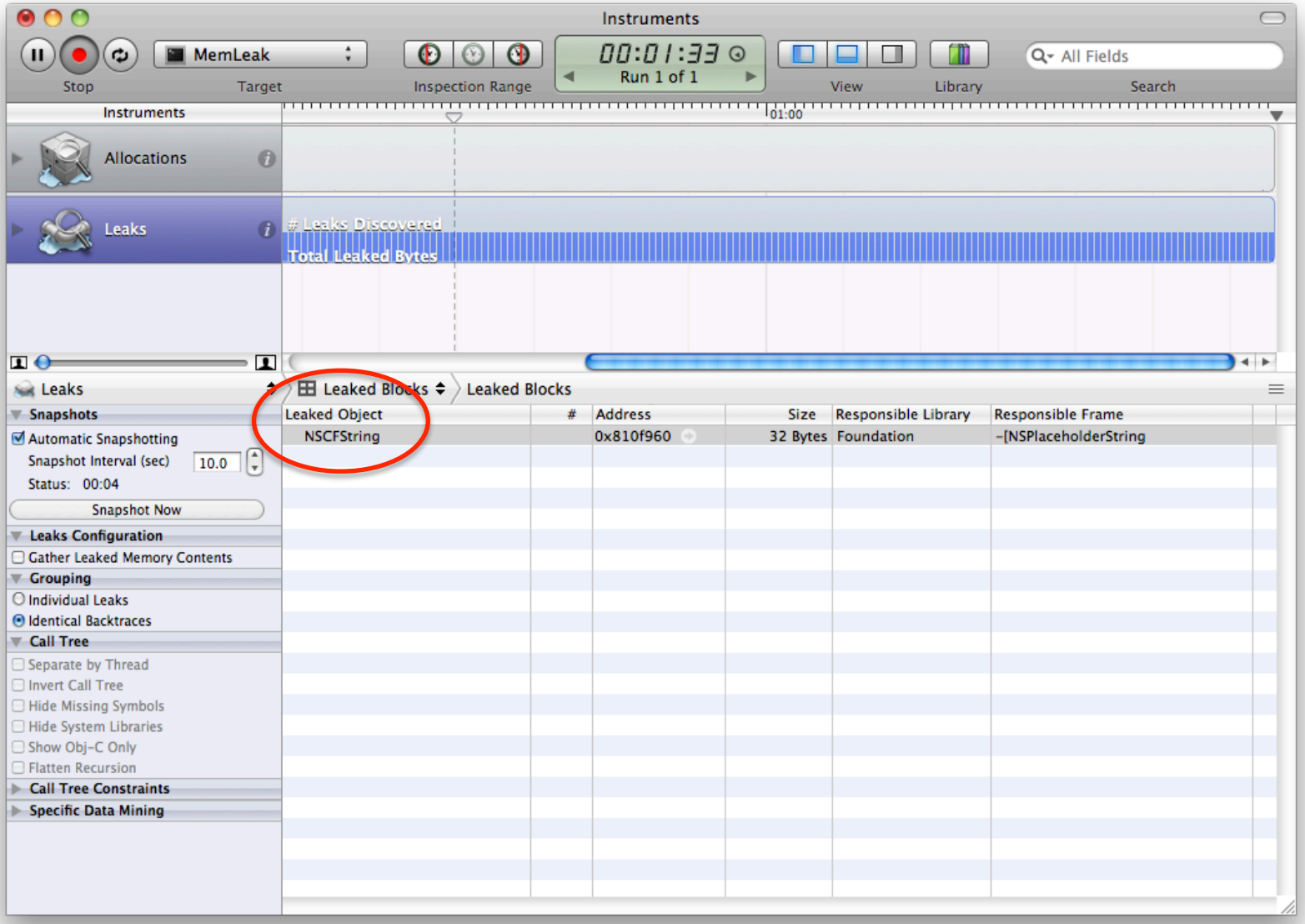


Profiling Code



Graph	Category	Live Bytes	# Living	# Transitory	Overall Bytes	# Overall	# Allocations (Net / Overall)
☑	* All Allocations *	912.95 KB	9357	0	912.95 KB	9357	+++
☐	Malloc 16 Bytes	49.44 KB	3164	0	49.44 KB	3164	
☐	Malloc 32 Bytes	49.88 KB	1596	0	49.88 KB	1596	
☐	CFString	45.23 KB	1462	0	45.23 KB	1462	
☐	Malloc 8 Bytes	5.41 KB	693	0	5.41 KB	693	
☐	Malloc 48 Bytes	13.55 KB	289	0	13.55 KB	289	
☐	CFString (store)	34.28 KB	271	0	34.28 KB	271	
☐	CFBasicHash	9.55 KB	210	0	9.55 KB	210	
☐	CFBasicHash (value-st...	16.27 KB	175	0	16.27 KB	175	
☐	Malloc 96 Bytes	14.44 KB	154	0	14.44 KB	154	
☐	Malloc 64 Bytes	8.81 KB	141	0	8.81 KB	141	
☐	Malloc 80 Bytes	9.53 KB	122	0	9.53 KB	122	
☐	CFBasicHash (key-store)	13.84 KB	121	0	13.84 KB	121	
☐	CFArray	2.11 KB	59	0	2.11 KB	59	
☐	Malloc 1.00 KB	49.00 KB	49	0	49.00 KB	49	
☐	Malloc 160 Bytes	7.19 KB	46	0	7.19 KB	46	
☐	CFArray (store-deque)	4.22 KB	41	0	4.22 KB	41	
☐	CFBasicHash (count-s...	784 Bytes	38	0	784 Bytes	38	
☐	Malloc 144 Bytes	5.34 KB	38	0	5.34 KB	38	
☐	Malloc 512 Bytes	19.00 KB	38	0	19.00 KB	38	
☐	CFRunLoopSource	4.05 KB	37	0	4.05 KB	37	
☐	Malloc 128 Bytes	4.12 KB	33	0	4.12 KB	33	
☐	CFMachPort	1.81 KB	29	0	1.81 KB	29	

Profiling Code



Best Method Avoiding Memory Leaks

- Program carefully, think hard
- Follow the memory management rules

- Ugly truth:
  Some leaks are in the frameworks as well!