iPhone Application Programming
Lecture 3: Foundation Classes

Prof. Jan Borchers
Media Computing Group
RWTH Aachen University

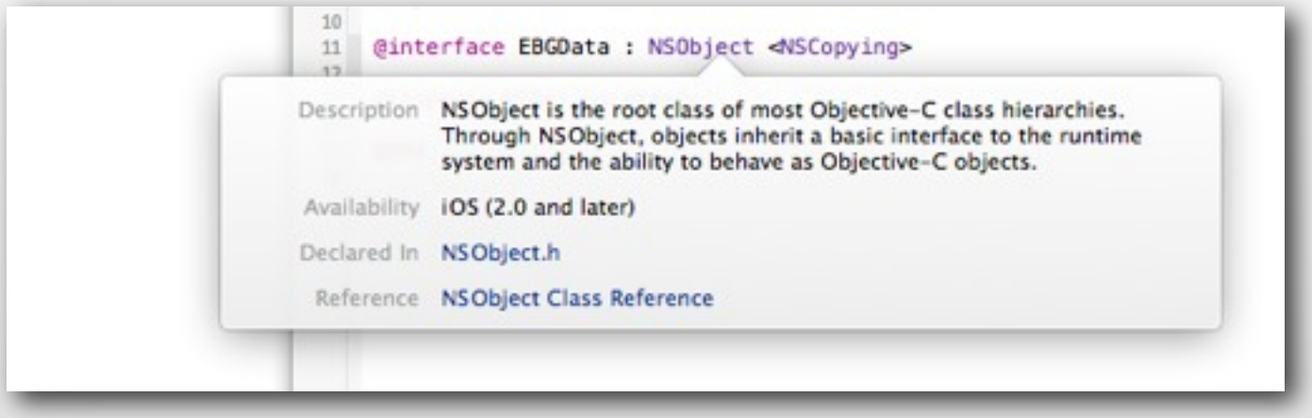
Winter Semester 2013/2014

http://hci.rwth-aachen.de/iphone

RWTH AACHEN UNIVERSITY

Getting Help

- Online Documentation
  - http://developer.apple.com/library/ios
  - Getting Started Videos
- Developer Forums
  - http://devforums.apple.com
- Cocoa
  - http://www.cocoadevcentral.com
- Google
- StackOverflow.com



alt + click for help

Documentation — NSObject Class Reference

nsstring

Next

NSObject Class Reference

Overview

`NSObject` is the root class of most Objective-C class hierarchies. Through `NSObject`, objects inherit a basic interface to the runtime system and the ability to behave as Objective-C objects.

Tasks

Initializing a Class

+ initialize
+ load

Creating, Copying, and Deallocating Objects

+ alloc
+ allocWithZone:
– init
– copy
+ copyWithZone:
– mutableCopy
+ mutableCopyWithZone:
– dealloc
+ new

Identifying Classes

+ class
+ superclass
+ isSubclassOfClass:

Testing Class Functionality

+ instancesRespondToSelector:

Testing Protocol Conformance

+ conformsToProtocol:

Provide Feedback

Data Structures in Objective C

Primitives

Numbers	int, float, double, NSNumber
Boolean	BOOL
String	NSString, NSMutableString
Date	NSDate
binary data	NSData
other	NSIndex, NSValue

Collections

Array	NSArray, NSMutableArray
Set	NSSet, NSMutableSet
Dictionary	NSDictionary, NSMutableDict...

Special

executable code	block

Mutable vs. Immutable

- Mutable: value can be changed after initialization
- Foundation classes are immutable
  - Exception: classes explicitly named “Mutable”
  - NSMutableString, NSMutableArray, NSMutableSet, NSMutableDictionary...
- Advantage of Immutable Objects
  - No side-effects when passing objects to methods

Mutable vs. Immutable

```
- (void) sort:(                    ) array;
```

NSArray or NSMutableArray ?

```
- (void) renderToScreen:(          ) data;
```

NSData or NSMutableData ?

Mutable vs. Immutable

```
- (void) sort:(NSMutableArray *) array;
```

```
// Common pattern: return a copy with the result
- (NSArray *) arraySortingArray:(NSArray *) array;
```

```
- (void) renderToScreen:(NSData *) data;
```

```
// This will not work!! Why?
- (void) renderToScreen:(const NSData *) data;
```

C Types

```
// Boolean
BOOL isPeter = [person.name isEqualToString:@"Peter"];
BOOL yesNo = YES ? YES : NO;

// Integer
int count = [people count];                 // 32 bit
long sum = x * y;                           // 64 bit

// Float
float average = sum / count;                // 32 bit
double precise_average = sum / count; // 64 bit

// Array and Pointers
float values[5];
char* string;
```

Pointers

```
// Pointer to C Type variable
int number = 1;
int *pointerToNumber = &number;
*pointerToNumber = 2;
NSLog(@"%d", number);

// Pointer to Objective-C object
NSString *hello = @"Hello";
NSString **pointerToHello = &hello;
*pointerToHello = @"Welcome";
NSLog(@"%@", hello);

// C Type arrays are based on pointers
int *intArray = malloc(sizeof(int) * 5);
for(int i=0; i<5; i++) intArray[i] = i;
NSLog(@"%d = %d", *intArray, intArray[0]);
```

NSNumber / NSValue

- Encode C-type variables as objects for inclusion in collections
- Numeric types: NSNumber
- Other types: NSValue
- Binary data: NSData

NSNumber

- Encapsulation of numerical values
- Provides compare: method to determine the ordering of two NSNumbers

```
+ numberWithBool:                    - boolValue
+ numberWithChar:                    - charValue
+ numberWithDouble:                  - doubleValue
+ numberWithFloat:                   - floatValue
+ numberWithInt:                     - intValue
+ numberWithInteger:                 - integerValue
+ numberWithLong:                    - longValue
+ numberWithShort:                   - shortValue
+ numberWithUnsignedChar:            - unsignedCharValue
+ numberWithUnsignedInt:             - unsignedIntegerValue
+ numberWithUnsignedInteger:         - unsignedIntValue
+ numberWithUnsignedLong:            - unsignedLongLongValue
+ numberWithUnsignedLongLong:        - unsignedLongValue
+ numberWithUnsignedShort:           - unsignedShortValue
```

NSNumber

```
NSNumber *aNumber;
aNumber = [NSNumber numberWithInt:42];      // 42
aNumber = [NSNumber numberWithDouble:3.14];// 3.14
aNumber = [NSNumber numberWithBool:YES];    // 1
aNumber = [NSNumber numberWithChar:'X'];    // 88
```

```
// Shorthand notation in modern objective-c
NSNumber *aNumber;
aNumber = @42;     // 42
aNumber = @3.14;   // 3.14
aNumber = @YES;    // 1
aNumber = @'X';    // 88
```

NSValue

- Container class
- Allows to use floats, chars, pointers,... in collection classes
- NSValue objects are always immutable

NSString

```objc
// Initialize string with unicode support
NSString *name = @"Rüdiger";
NSString *name2 = @"中島康祐";

// Create a message
NSString *message = [NSString stringWithFormat:
                     @"Hello %@, you are the %dth visitor.", name, 5];

// Append a string
NSString *messageLong = [message stringByAppendingString:
                         @" Thank you! ♥"];

// Parsing for int-value
int code = [@"404" intValue];

// Log to the console
NSLog(@"%d: %@", code, messageLong);

// Mutable strings can be modified
NSMutableString *helloWorld = @"Hello";
[helloWorld appendString:@" World"];
```

NSString

- String formatting
- Searching / comparing / sorting
  - NSScanner: value scanner
  - Line / paragraph / path separation
- Appearance
  - NSAttributedString: string with attributes
- Reading from / writing to file / URL

NSDate

```objc
// create dates
NSDate *now = [NSDate date];

// date for the first monday of May 2008
NSDateComponents* components = [[NSDateComponents alloc] init];
[components setWeekday:2];          // Monday
[components setWeekdayOrdinal:1]; // The first day in the month
[components setMonth:5];            // May
[components setYear:2008];
NSCalendar *gregorian = [[NSCalendar alloc]
                 initWithCalendarIdentifier:NSGregorianCalendar];
NSDate *date = [gregorian dateFromComponents:components];
```

```objc
// bad way to do it, reports 86400 only if you are lucky
NSDate *tomorrow = [now addTimeInterval:24 * 60 * 60];
NSLog(@"%.0f", [tomorrow timeIntervalSinceNow]);

// right way to do it, works on all dates
NSDateComponents* offset = [[NSDateComponents alloc] init];
[offset setDay:1];
NSDate *tomorrow = [gregorian dateByAddingComponents:offset];
NSLog(@"%.0f", [tomorrow timeIntervalSinceNow]);
```

NSDateFormatter

```
// create a format template for dates
NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
[dateFormatter setTimeStyle:NSDateFormatterNoStyle];
[dateFormatter setDateStyle:NSDateFormatterShortStyle];

// 10/23/12
NSLog(@"Date: %@", [dateFormatter stringFromDate:date]);

NSLocale *deLocale = [[NSLocale alloc]
                      initWithLocaleIdentifier:@"de_DE"];
[dateFormatter setLocale:deLocale];

// 23.10.12
NSLog(@"Date (DE): %@", [dateFormatter stringFromDate:date]);
```

Collection Classes

- NSArray – NSMutableArray
- NSDictionary – NSMutableDictionary
- NSSet – NSMutableSet – NSCountedSet
- NSIndexSet – NSMutableIndexSet

NSArray

```objc
// Initialize array with multiple objects
NSArray *array = [NSArray arrayWithObjects:
                            @"first", @"second", nil];

// Count elements in the array
int arrayCount = [array count];

// Access object at index
id firstArrayElement = [array objectAtIndex:0];

// Extend array
NSArray *extendedArray = [array arrayByAddingObject:@"third"];

// Iterate over array
for (id object in extendedArray)
{
  [object doSomething];
}

// Simple saving to disk
[array writeToFile:@"/Users/alice/debug.plist" atomically:YES];
```

NSArray

- Direct element objects through index
- Searching, sorting, filtering
  - NSSortDescriptor: sort rules
  - NSPredicate: filter conditions
- Enumeration
  - NSEnumerator / Fast Enumeration
- Joined string, paths
- Reading from / writing to file / URL

NSSet

- Unique objects
- No direct access to objects
  - Only random object
  - Enumeration
  - Set operations
  - Conversion to Array

NSDictionary

```
// Initialize dictionary with keys and values
NSDictionary *dict = [NSDictionary dictionaryWithObjectsAndKeys:
                      @"Peter", @"first_name",
                      @"Smith", @"last_name", nil];

// Access objects by key
id first_name = [dict objectForKey:@"first_name"];

// Enumerate over keys
NSString *key
for(key in dict) {
  id object = [dict objectForKey:key];
  ...
}

// Check for key existence
BOOL keyExists = [dict objectForKey:key] != nil;
```

To be continued in the next lecture