iPhone Application Programming
Lecture 3: Foundation Classes

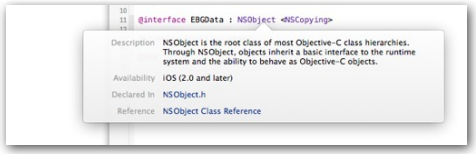
Prof. Jan Borchers
Media Computing Group
RWTH Aachen University

Winter Semester 2013/2014

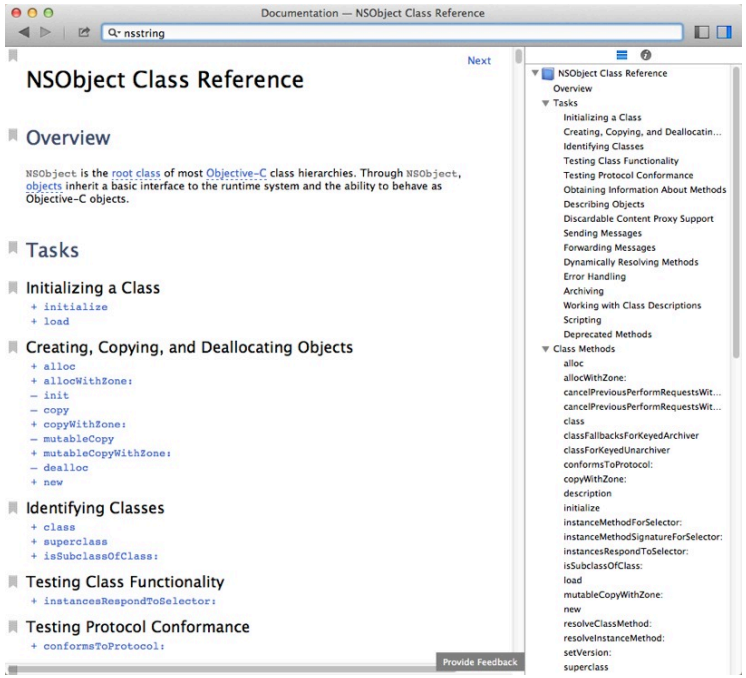
http://hci.rwth-aachen.de/iphone

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



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];

// 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

```
// 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;
```


Review

- Mutable vs. Immutable objects
- NSNumber vs. NSValue
- NSArray vs. NSDictionary
- NSArray vs. NSSet

NSDictionary

- Direct object access through key
  - All keys, all objects
- Sorting, Filtering
- Enumeration
  - Keys / keys and objects
- Reading from / writing to file / URL

Collections and modern Obj-C

```
id first, second, third;

// The list of objects is nil-terminated
NSArray *anArray = [NSArray arrayWithObjects:first,second,third,nil]
NSArray *anArray = @[first, second, third];

// get the first object in the array
id numberOne = [anArray objectAtIndex:0];
id numberOne = anArray[0];
aMutableArray[1] = second;

// also works for dictionaries
NSArray *lastNames = @[@"Borchers", @"Heller"];
NSArray *firstNames = @[@"Jan", @"Florian"];

// classic style:
NSDictionary *iPhoneTeachers = [NSDictionary
                                dictionaryWithObjects: firstNames
                                forKeys: lastNames];
NSString *firstName = [iPhoneTeachers objectForKey:@"Heller"];

// modern style:
NSDictionary *iPhoneTeachers = @{   @"Borchers" : @"Jan",
                                    @"Heller" : @"Florian",   };
```


More Foundation Topics!

- Bundles and Files
- Exceptions and Errors
- Notifications
- Key-Value Coding and Observing
- Archive and Serialization
- Timers and Threads
- Localization
- Caches

Bundles and Files

Bundles

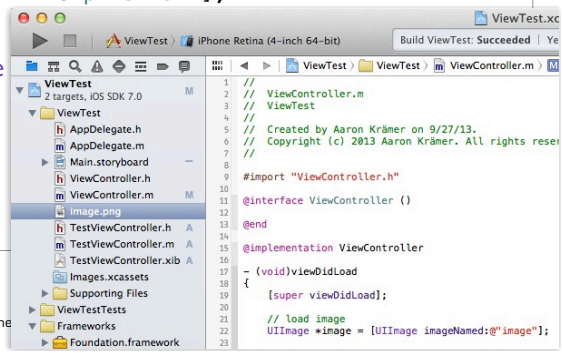
- A bundle is a folder that contains executable code and resources in a standardized structure
  - Applications are bundles
- NSBundle is used to access bundle resources
  - Access by name/type — resources must be unique
  - Can load resource or extract file path

Bundles

```objc
// get the application bundle
NSBundle *mainBundle = [NSBundle mainBundle];

// load a plist
NSString *plistPath = [mainBundle pathForResource:@"data"
  ofType:@"plist"];
NSDictionary *plist = [NSDictionary
  dictionaryWithContentsOfFile:plistPath];

// load image
UIImage *image = [UIImage
  imageNamed:@"image"];
```



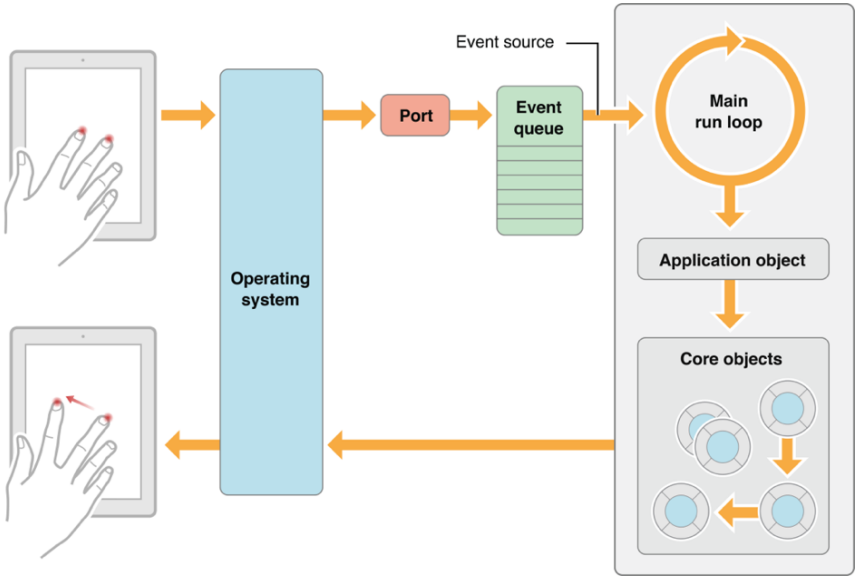
System Folders

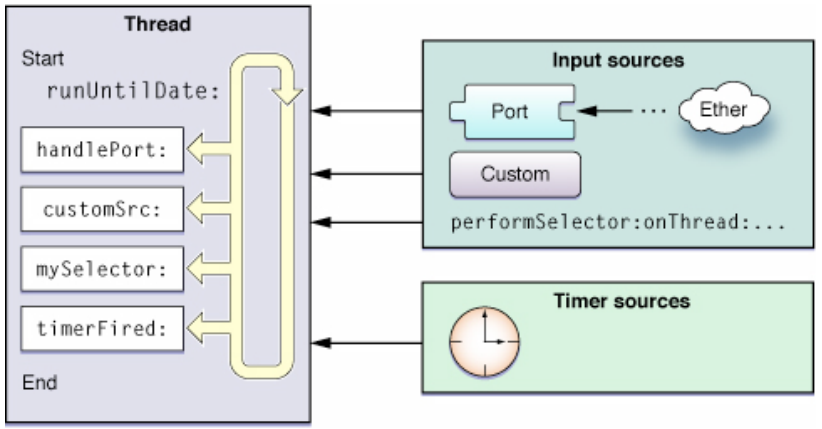
User Home	`NSHomeDirectory()`
Application	`[[NSBundle mainBundle] resourcePath]`
Temporary	`NSTemporaryDirectory()`

File Management

- NSFileManager
  - Copy / move files and folders
  - Discover files and folders
  - Manage links
  - Change attributes
  - Create files / folders
  - ...

Timer and Threads



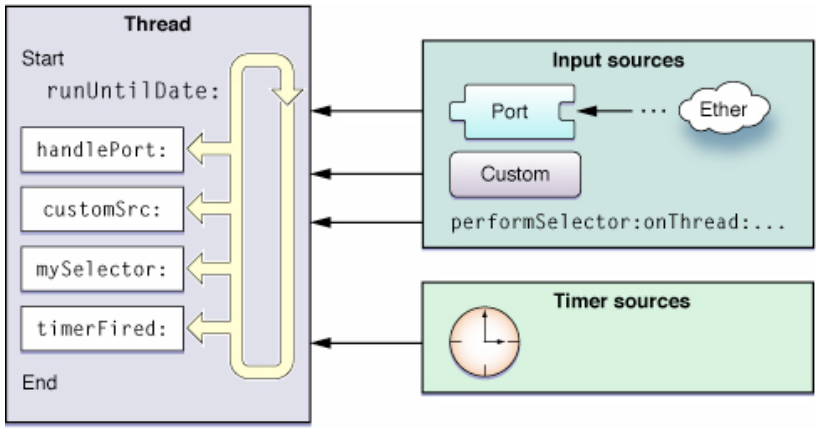


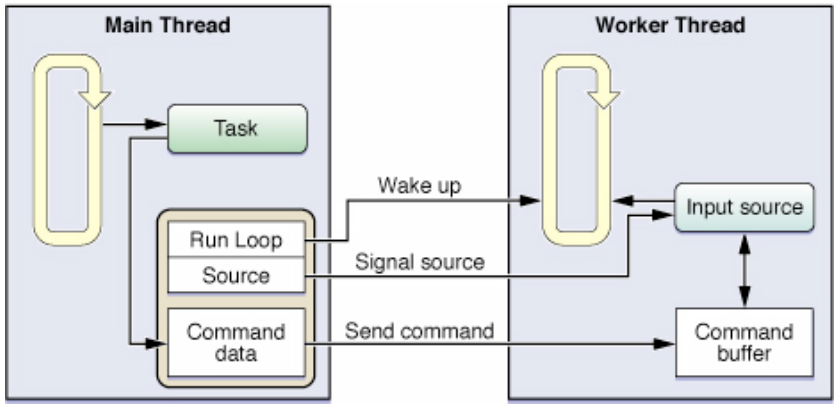
Timer

```
- (void) setupTimer {
  // schedule a new timer and attach to run loop
  timer = [[NSTimer alloc] initWithFireDate:[NSDate date]
     interval:1.0 target:self selector:@selector(onTimer:)
     userInfo:nil repeats:YES];
  NSRunLoop *runLoop = [NSRunLoop mainRunLoop];
  [runLoop addTimer:timer forMode:NSDefaultRunLoopMode];
}

- (void) onTimer:(NSTimer *)theTimer {
  // react to the timer
}

// clean up
- (void) dealloc {
  [timer invalidate];
  [super dealloc];
}
```





Thread

```
// detach a new thread
thread = [[NSThread alloc] initWithTarget:self
  selector:@selector(threadMain) object:nil];
  [thread start];

- (void) threadMain {
  while(![[NSThread currentThread] isCancelled]) {
     @autoreleasepool {
        // do something
        [NSThread sleepForTimeInterval:0.1];
        [pool drain];
     }
  }
}

// clean up
[thread cancel];
```


NSOperations

```
NSOperationQueue *queue = [[NSOperationQueue alloc] init];
NSBlockOperation *operation = [NSBlockOperation
blockOperationWithBlock:^{
     NSLog(@"Doing something...(this may take some time)");
}];

// you can add more blocks
[operation addExecutionBlock:^{ NSLog(@"Another block"); }];

// dispatch it
[operation setCompletionBlock:^{
     NSLog(@"Doing sth. once the operation has finished...");
}];

[queue addOperation:operation];
```


Threads vs. Timers

- Timers
  - Bound to run loop of the current thread
  - Fires once or regularly
- Thread
  - Executable code runs in parallel
  - Must create its own run loop

Exceptions and Errors

Exceptions vs. Errors

- Exception
  - Only for semantic errors in your code
  - Used for debugging
  - Examples: wrong parameters, assertion failures
- Error
  - For erroneous user actions or unavoidable exceptions
  - Reported to the user
  - Examples: validation errors, connectivity problems

Exceptions

```
// raise an exception
[NSException raise:@"Short Name"
  format:@"Description with debugging information"];

// simple assertion
NSAssert(param != nil, @"Parameter must be set");

// catching exceptions
@try {
  // code that might raise an exception
}
@catch(NSException *exception) {
  // handle the exception
}
@finally {
  // code that is always executed
}
```


Errors

```
   // method with potential error
- (BOOL)performWithError:(NSError **)error {
  if(somethingFails()) {

     // create the error
     *error = [NSError errorWithDomain:@"Domain"
       code:1 userInfo:nil];
     return NO;
  }
  return YES;
}

// run a method with potential error
NSError *error;
if(![self performWithError:&error]) {
  NSLog(@"%@", error);
}
```

Debugging and Performance Tweaking

Debug Logging

- Log important / problematic operations
- Implement description method:

```
- (NSString *)description {
    return [NSString stringWithFormat:
      @"<Person name:%@>", self.name];
}
```

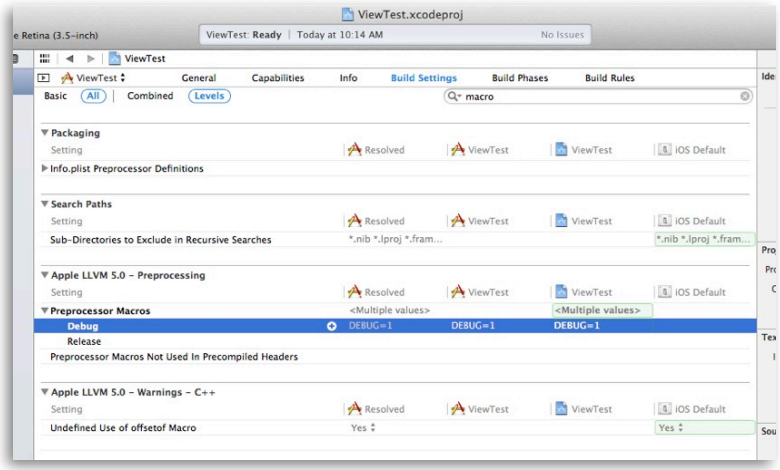
- Use preprocessor macro for logging:
  - Log to console in Debug mode
  - Ignore in Release mode

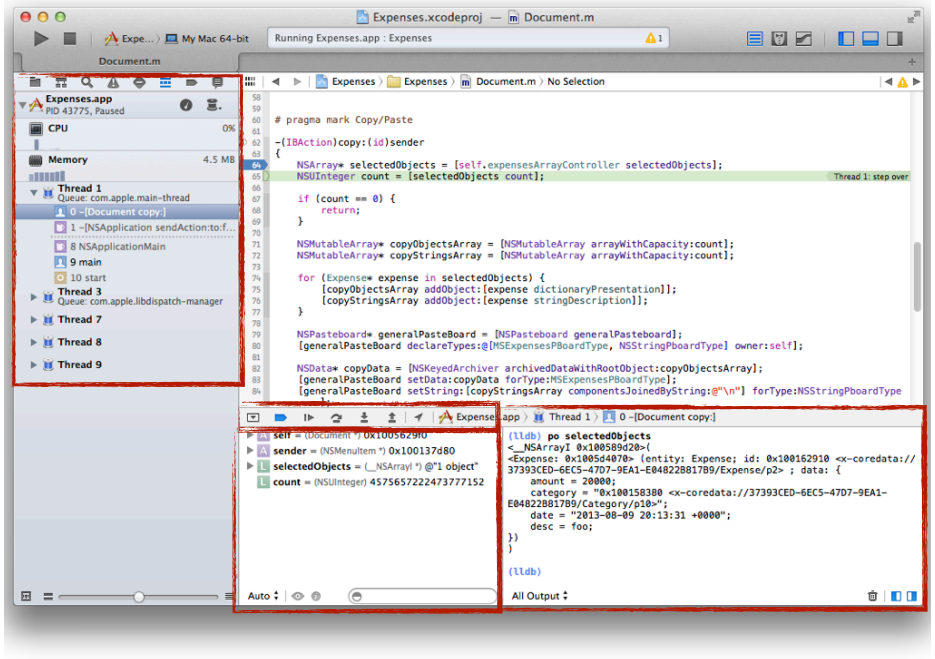
Debug Logging

```
  // define DLog function for DEBUG and non-DEBUG
#ifdef DEBUG
#  define DLog(fmt, ...) NSLog((@"%s [Line %d] " fmt),
__PRETTY_FUNCTION__, __LINE__, ##__VA_ARGS__);
#else
#  define DLog(...)
#endif

// Debug Logging
DLog(@"Something did not work: %@", object);
```


Debug Logging

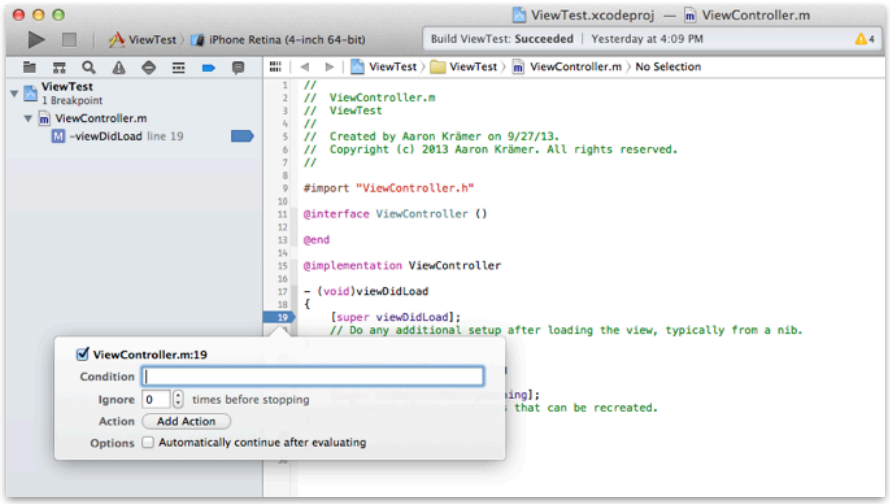




Console

- Dump variables
  - po for objects, p for C-types
  - Perform method calls
- Handle breakpoints
  - Conditional breakpoints
  - Breakpoints on symbols
  - Watchpoints, catchpoints
- Examine the call stack and source files
- Much more...

Breakpoints



Debugging Exceptions

- When an exception is raised:
  - The application is already dead
  - Cannot navigate call stack to check the problems
- Solution: Add breakpoint on objc_exception_throw

Archive and Serialization

NSCoding

- Formal protocol for en- /decoding objects
  - `- (void)encodeWithCoder:(NSCoder *)encoder`
  - `- (id)initWithCoder:(NSCoder *)decoder`
- NSCoder provides methods for (re)storing all basic data structures
- NSKeyed(Un)Archiver to (un)archive objects into data streams (NSData)

Property Lists

- Property lists are serialized collections
  - NSArray or NSDictionary
  - XML or binary
- Can be read/written directly
- Many APIs for plist handling available
  - PHP, Ruby, Java
- Plists can be included as resources in a project

Notifications

Notifications

Publish-subscribe information sharing

1. Observers register with the notification center
2. Senders post notifications to the notification center
3. Notification center forwards notification to observers

Notifications

```objc
// get a reference to the default notification center
NSNotificationCenter *notificationCenter =
  [NSNotificationCenter defaultCenter];

// register observer to receive my notification
[notificationCenter addObserver:observer
  selector:@selector(didNotify:) name:@"MyNotification"
  object:nil];

// create and post notification
NSNotification *notification = [NSNotification
  notificationWithName:@"MyNotification" object:object];
[notificationCenter postNotification:notification];

// remove observer
[notificationCenter removeObserver:observer];
```


Notifications

- When to use?
  - System-wide (UI) events
  - Inter-system communication
  - Key-value observing
  - Some frameworks use notifications
- When not to use?
  - To delegate behavior
  - High-performance situations
- Don't forget to remove your observers!

Key-Value Coding and Observing

Key-value Coding vs. Observing

Coding	Observing
Access object properties indirectly through path	Notify observers about changes of object properties
`person.supervisor.name`	`person.name = ...` ↓ inform observers

Key-Value Observing

```
// create person
person = [[Person alloc] init];
[person setName:@"Paul"];

// add observer
[person addObserver:self forKeyPath:@"name"
  options:NSKeyValueObservingOptionNew +
          NSKeyValueObservingOptionOld
  context:nil];

// fires KVO notification (in runloop)
[person setName:@"Peter"];

// receive KVO notification
- (void)observeValueForKeyPath:(NSString *)keyPath
  ofObject:(id)object change:(NSDictionary *)change
  context:(void *)context { ... }
```


Key-Value Coding

read	`- (id)<key>`
write	`- (void)set<Key>:(id)value`
validate	`- (BOOL)validate<Key>:(id *)value error:(NSError **)error`

Key-Value Coding

```
// create susan with parent michael
Person *susan = [Person personNamed:@"Susan"];
susan.parent = [Person personNamed:@"Michael"];

// reports Susan and Michael
NSLog(@"%@", [susan valueForKey:@"name"]);
NSLog(@"%@", [susan valueForKeyPath:@"parent.name"]);

// validate and change name
NSError *error;
NSString *newName = @"Sarah";
if([susan validateValue:&newName forKey:@"name" error:&error]) {
  [susan setValue:newName forKey:@"name"];
} else {
  NSLog(@"%@", error);
}
```

To-Many Relationships

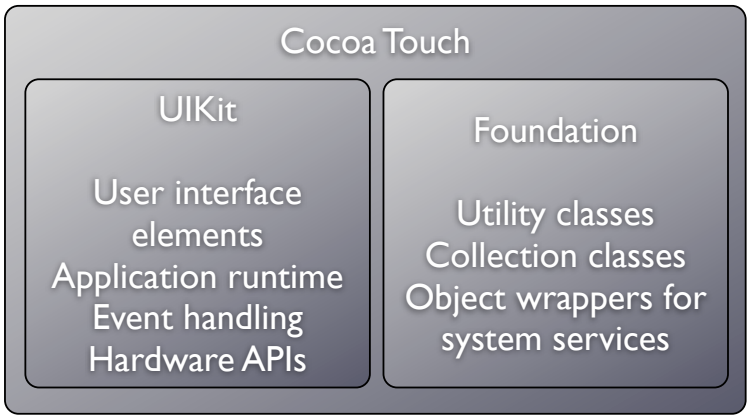
- KVC variable can be collection
- Problem: collection changes are not propagated

Insert	`- (void)insertObject:(id)object in<Key>AtIndex:(int)index`
Remove	`- (void)removeObjectFrom<Key>AtIndex:(int)index`
Replace	`- (void)replaceObjectIn<Key>AtIndex:(int)index withObject:(id)object`

Key-Value Observing Purpose

- Observe and react to changes of your data
  - Automatic UI update (Bindings)
  - Synchronization to persistent store (Core Data)
  - Consistency checking

Cocoa Touch Architecture



Summary

- Fundamentals of iOS programming
- Debugging and performance tweaking
- References
  - Foundation Guides
  - Xcode Debugging Guide / Debugging with LLDB
  - Aaron Hillegass: Cocoa Programming for Mac OS
  - http://www.cocoadevcentral.com