Objective-C Foundation Framework

Khoai Nguyen

14 April, 2009

<CTO – iPhone Group>

< RnD>

Agenda

- ❑ Object Oriented Programming Overview.
- ❑ Objective-C Language
- ❑ Common Foundation Classes

Agenda

- ❑ Object Oriented Programming Overview.
- ❑ Objective-C Language
- ❑ Common Foundation Classes

Object Oriented Programming Overview

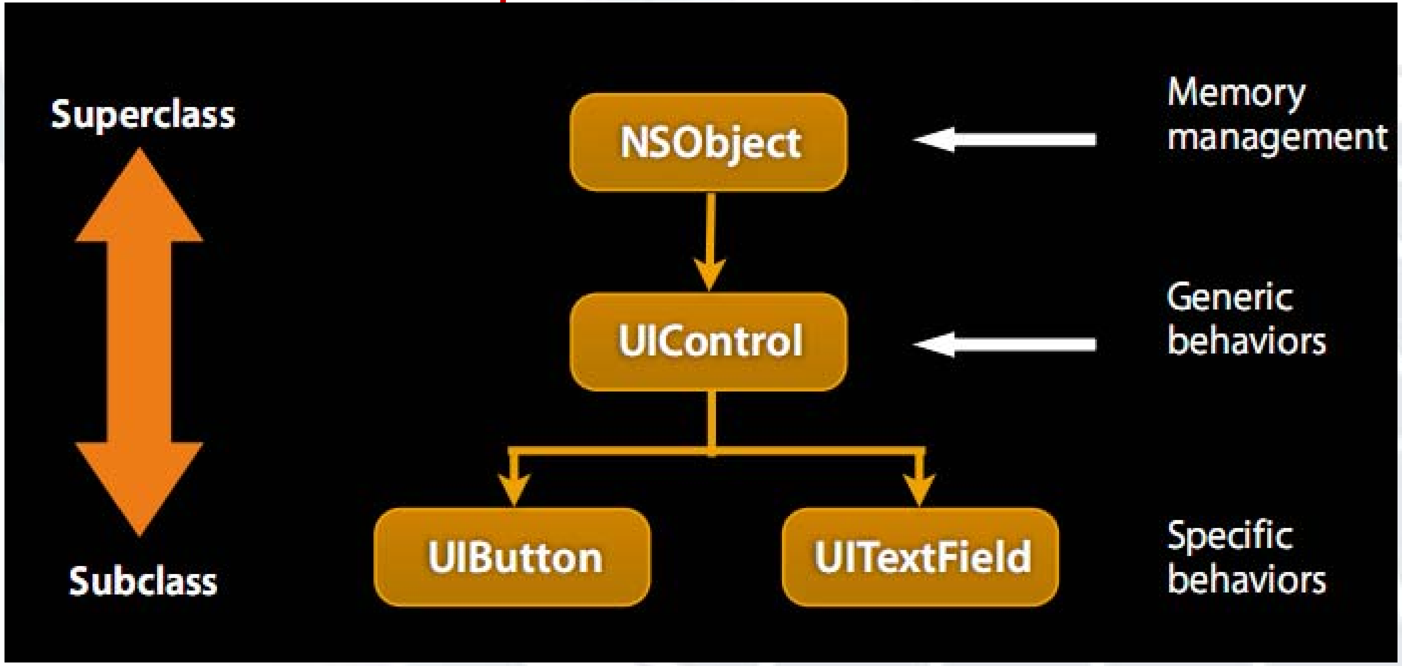
- Class : defines the grouping of data and code, the “type” of an object .
- Instance: a specific allocation of a class.
- Method: a “function” that an object knows how to perform.
- Instance Variable (or “ivar”): a specific piece of data belonging to an object.

Object Oriented Programming Overview (cont)

- Encapsulation : keep implementation private and separate
- Polymorphism : different objects, same interface.
- Inheritance : hierarchical organization, share code, customize or extend behaviors.

Object Oriented Programming Overview (cont)

Inheritance in Objective-C



Inheritance in Objective-C (cont)

- Hierarchical relation between classes
- Subclass “inherit” behavior and data from superclass
- Subclasses can use, augment or replace superclass methods

Agenda

- ❑ Object Oriented Programming Overview.
- ❑ Objective-C Language
- ❑ Common Foundation Classes

Objective-C Language

- Introduction.
- The main concepts behind classes.
- Memory management in iPhone OS.
- Objective-C Protocols.
- Properties.
- Extending existing classes with Categories.
- Exceptions and error handling.
- Using multithread.

Objective-C Language

- Introduction.
- The main concepts behind classes.
- Memory management in iPhone OS.
- Objective-C Protocols.
- Properties.
- Extending existing classes with Categories.
- Exceptions and error handling.
- Using multithread.

Introduce

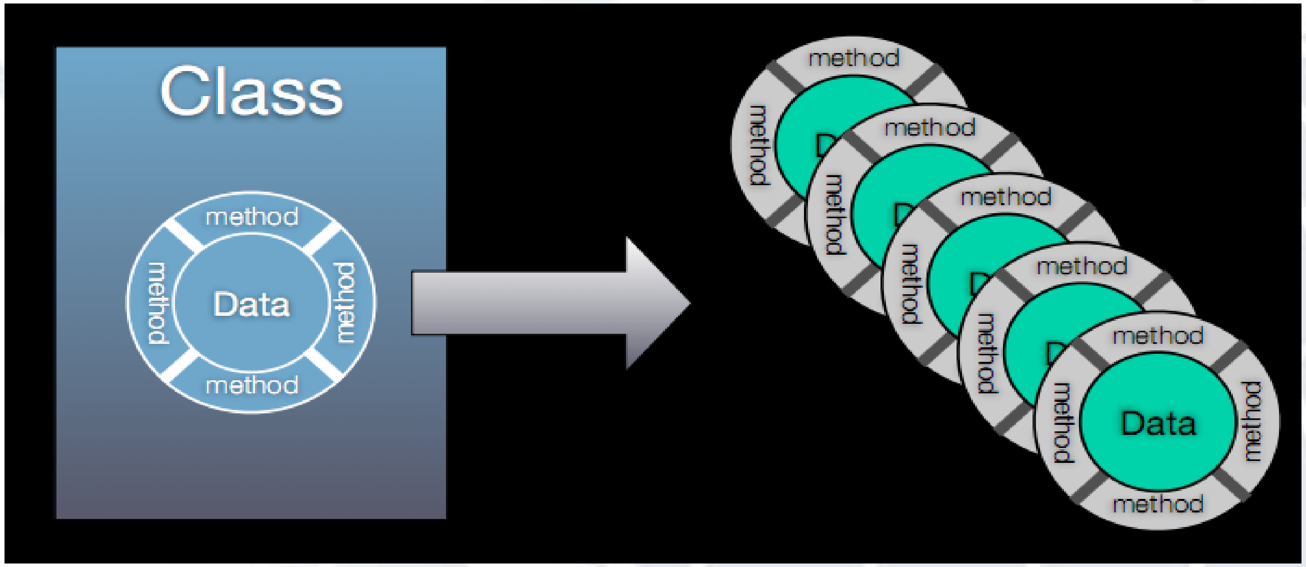
- Objective-C is an OOP language.
- Strict superset of C
  - Mix C with ObjC
  - Or even C++ with ObjC (usually referred to as ObjC++)
- A very simple language, but some new syntax
- Single inheritance, classes inherit from one and only one superclass
- Protocols define behavior that cross classes
- Dynamic runtime
- Loosely typed

Objective-C Language

- Introduction.
- The main concepts behind classes.
- Memory management in iPhone OS.
- Objective-C Protocols.
- Properties.
- Extending existing classes with Categories.
- Exceptions and error handling.
- Using multithread.

Objective-C Classes

- Instances that objects are created indirectly by class.
- Classes are objects too.



Objective-C Classes (cont)

- Type of a class object is Class . A null class pointer is Nil.
- Not support class variables.
- You should define a subclass base upon the Foundatiuon framework.
- Abstract classes aren't marked by syntax and can create instances.

Objective-C Classes (cont)

```
// ClassName.h
@interface ClassName : ItsSuperclass
{
    instance variable declarations
}
method declarations
@end
```

Objective-C Classes (cont)

```
// ClassName.m
#import "ClassName.h"
@implementation ClassName
Implement methods
@end
```

Objective-C Classes (cont)

- Using defined classes by declaring with *#import*
- Referring to classes by using @class directive.

Example :

```
@class  Address;
@interface Person
{
    Address *address;
}
@end
```

Objective-C Objects

- Object identifiers are a distinct data type : id.

```
typedef struct objc_object {
Class isa;
} *id;
typedef struct objc_class *Class;
```

- The keyword nil is defined as a null object. an id with a value of 0.
- Declare and use an object in two ways : static typing and dynamic binding.

Objective-C Objects (cont)

- Example :

```
Rectangle *thisObject; // static typing

id thisObject; // dynamic binding at run time.
```

Dynamic and static typing

- Dynamically-typed object.

  `id anObject;`

  - Just id
  - Not id * (unless you really, really mean it...)
- Statically-typed object.

  `Person * anObject;`
- Objective-C provides compile-time, not runtime, type checking
- Objective-C always uses dynamic binding.

Instance Variables

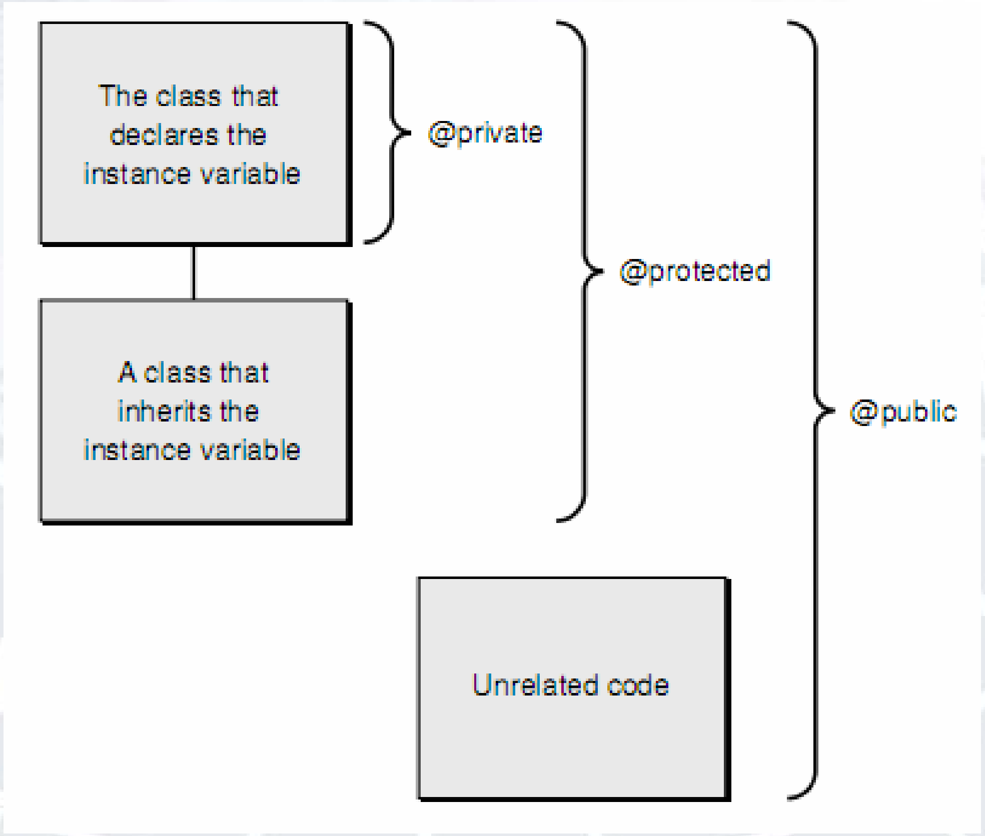
- Internal variables : define in *@interface* block.
- External variables : define for instances of a class shared data .

Example :

```
NSString *unknow; // external variable
@interface Tweet : NSObject {
 unsigned identifier; // internal variable
 NSString *sender; // internal variable
 NSString *text; // internal variable
}
…
@end
```

Instance Variables (cont)

- The scope of instance variables. (*@private*,*@protected*, *@public*).
- Default is *@protected*.


The class that declares the instance variable
A class that inherits the instance variable
Unrelated code
@private
@protected
@public

Send Messages

- Instances respond to instance methods

```
-(id)init;
-(float)height;
-(void)walk;
```

- Classes respond to class methods

```
+ (id)alloc;
+ (id)person;
+ (Person *)sharedPerson;
```

Send Messages (cont)

- Message is simply a method in receiver's repertoire.
- A message is composed of two parts :
  - Keywords
  - Parameters
- Message expression :

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

Send Messages (cont)

Example :

```
Rectangle * nObjective = [[Retangle alloc] init];
[nObject setOriginWithX:0.0 andWithY: 3.0];
[nObject display];
```

Selectors

- In the above example, representation setOriginWithX: andWithY: is called a *selector .*
- *A selector* is a unique name (within a class) of a method.
- SEL is a defined type that represent a *selector.*

```
SEL action = [button action];
[button setAction:@selector(start:)];
```

- Conceptually similar to function pointer.

Selectors (cont)

- You can determine if an object responds to a given selector.

```
id obj;
SEL sel = @selector(start:);
if ([obj respondsToSelector:sel]) {
    [obj performSelector:sel withObject:self];
}
```

- This sort of introspection and dynamic messaging underlies many Cocoa design patterns.

```
-(void)setTarget:(id)target;

-(void)setAction:(SEL)action;
```

Objective-C Language

- Introduction.
- The main concepts behind classes.
- Memory management in iPhone OS.
- Objective-C Protocols.
- Properties.
- Extending existing classes with Categories.
- Exceptions and error handling.
- Using multithread.

Memory management in iPhone OS

- Reference counting
  [NSObject retain];
  [NSObject release];
- Autorelease pools
  NSAutoreleasepool
- Simple rules
  -init and -get…methods return retained objects
  Everything else is autorelease
- Deallocation
  When your retain count hits 0, your –dealloc method is called

Objective-C Language

- Introduction.
- The main concepts behind classes.
- Memory management in iPhone OS.
- Objective-C Protocols.
- Properties.
- Extending existing classes with Categories.
- Exceptions and error handling.
- Using multithread.

Objective-C Protocols

- To declare methods that others are expected to implement
- To declare the interface to an object while concealing its class.
- To capture similarities among classes that are not hierarchically related.

Objective-C Protocols (cont)

- Formal protocols
  - There are two modal keywords : *@optional* and *@required* . (default is *@required*).

```
@protocol MyProtocol
- (void)requiredMethod;
@optional
- (void)anOptionalMethod;
- (void)anotherOptionalMethod;
@required
- (void)anotherRequiredMethod;
@end
```

Objective-C Protocols (cont)

- Informal protocols

Define an informal protocol by grouping the methods in a category (describe later) declaration:

```
@interface NSObject ( MyXMLSupport )
- initFromXMLRepresentation:(NSXMLElement *)XMLElement;
@property (nonatomic, readonly) (NSXMLElement*)
                        XMLRepresentation;
@end
```

Objective-C Protocols (cont)

- Adopting a protocol.
  - @interface ClassName : ItsSuperclass < protocol list >
  - @interface ClassName (CategoryName) < protocol list >
  - @interface Formatter : NSObject < Formatting, Prettifying >
- Source code can refer to a Protocol object using the *@protocol()* directive.

```
Protocol *myXMLSupportProtocol = @protocol(MyXMLSupport);
```

Objective-C Protocols (cont)

- Protocols within protocols.

  @protocol ProtocolName < protocol list >

Example :

```
@protocol Paging < Formatting >
```

- You can use type declaration.

  ```
  id<ProtocolName> someObject;
  ```

  to define a protocol.

Example :

```
@protocol Litigating
    - (int)sue : (id<Litigating>) someone;
@end
```

Objective-C Language

- Introduction.
- The main concepts behind classes.
- Memory management in iPhone OS.
- Objective-C Protocols.
- Properties.
- Extending existing classes with Categories.
- Exceptions and error handling.
- Using multithread.

Objective-C Properties

- Allowing you to generate setter/getter methods for your instance variables.

@property (attributes) type name;

- Keywords are used for declaring attributes : *nonatomic*, *readonly*, *readwrite*, *assign, retain, getter=getterName*, *setter=setterName*.

Example : @interface Tweet : NSObject {

```
        …
       }
       @property (readonly) NSString *text;
       @end
```

Objective-C Properties (cont)

- Ask compiler to generate setter/getter that correspond to attributes by using *@synthesize* in implementation file.

Ex :
```
@implementation
@synthesize text; // declared  in above example
@end
```

- Implement the methods yourself by using *@dynamic* directive.

Ex :
```
@implementation
@dynamic text;
-(NSString*)getText {
        // implement it
        …
}
@end
```

Objective-C Properties (cont)

- For mutable collections such as : NSMutableArray , NSMutableString,… , the compiler-provided setter/getter might not be appropriate.
- Solution : return a autoreleased copy of collection.

Ex:
```
@interface Employee
@dynamic achievement;
-(void) setAchievement : (NSMutableArray *) newAchievements {
        if(achievements != newAchievements) {
                [achievements release];
                achievements = [newAchievements mutableCopy];
        }
}
@end
```

Objective-C Language

- Introduction.
- The main concepts behind classes.
- Memory management in iPhone OS.
- Objective-C Protocols.
- Properties.
- Extending existing classes with Categories.
- Exceptions and error handling.
- Using multithread.

Extending existing classes with Categories

- A category allows you to add methods to an existing class - even to one to which you do not have the source.
- Cannot use a category to add additional instance variables to an existing class.
- By using category, we can define a class in many files.

Extending existing classes with Categories (cont)

- Define a category

```
#define TWOOSH_LENGTH (140)
@interface Tweet (TwooshSupport)
        - (BOOL)isTwoosh;
@end
@implementation Tweet (TwooshSupport)
        - (BOOL)isTwoosh {
                return [text length] == TWOOSH_LENGTH;
        }
@end
```

Objective-C Language

- Introduction.
- The main concepts behind classes.
- Memory management in iPhone OS.
- Objective-C Protocols.
- Properties.
- Extending existing classes with Categories.
- Exceptions and error handling.
- Using multithread.

Exceptions and error handling

- Exceptions: the developer’s fault.
- Using try block to capture a possible exception.

```
@try {
    // statements that may cause an exception
}
@catch (NSException *e) {
    // statements handle  an exception
    @throw; // optionally re-throwing the exception
}
@finally{
    // statements that should be executed regardless of having an
    //exception or not
}
```

Exceptions and error handling

- Exceptions: the developer's fault.
- Using NSException.

```
-(void) myMethod: (NSString*) str {
    if(str = nil){
      NSException *ex = [NSException
              exceptionWithName:@"NSInvalidArgument"
              reason:@"Argument is nil"
              userInfo:nil];
      @throw ex;
      // or @rise ex;
    }
}
```

Exceptions and error handling

- Errors : the user's fault
- Using error codes to conveying runtime errors to users.
    - Using objects of type NSError (or its subclass).

```
NSError *myError = nil;
NSURL *myURL = [NSURL URLWithString:@"http://fox.gov"];
NSString *str = [NSString StringWithContentsOfURL:myURL
                    encoding:NSUTF8StringEncoding
                    error:&myError];
```

Exceptions and error handling

- An NSError object stores three important attributes:
  - domain : a string representing the error domain.
  - code : an integer error code that has meaning within the domain.
  - userInfo : a dictionary containing objects related to the error.

Objective-C Language

- Introduction.
- The main concepts behind classes.
- Memory management in iPhone OS.
- Objective-C Protocols.
- Properties.
- Extending existing classes with Categories.
- Exceptions and error handling.
- Using multithread.

Using multithread

- Multithread in Cocoa is very simple to achieve.
- Using *operation objects* that created by NSOperation or its subclass NSInvocationOperation and NSOperationQueue*.*
- Beside, we must use *@synchronized()* directive to guarantee excluse access (like semaphore in C).

Using multithread (cont)

- Example :

```
@interface MyClass : NSObject {
        …
        NSInvocationOperation *comOp;
        NSOperationQueue *opQueue;
}
…
- (void) computeInBackground:(id) data;
- (BOOL) computationFinished;
- (DS*) computationResult;
@end
```

Using multithread (cont)

```
@interface MyClass
        - (void) computeInBackground: (id) data         {
                comOb = [[[NSInvocationOperation alloc]
initWithTarget:self
                        selector:@selector(compute:)
                        object: data] autorelease];
                [opQueue addOperation: comOb];
        }
…
```

Using multithread (cont)

```
- (BOOL) computationFinished {
    @synchronized(ds) {
        // if ds is complete return YES,
        // else return NO
    }
}
- (DS*) computationResult {
    if([self computationFinished] == YES){
        return ds;
    }else {
        return nil;
    }
}
```

Using multithread (cont)

```
-(void) compute: (id)data {
        NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
        // do something
        @synchronized(ds){
                // store result in ds
        }
        [pool release];
}
…
```

Using multithread (cont)

```
-(void) someOtherMethod {
        MyClass *anObject;
        …
        [anObject computeInBackground: data];
        // be responsive to user GUI
        …
        if([anObject computationFinished] == YES) {
                result = [anObject computationResult];
        }
}
@end
```

Agenda

- ❑ Object Oriented Programming Overview.
- ❑ Objective-C Language
- ❑ Common Foundation Classes

Common Foundation Classes

Foundation Framework

- Value and collection classes
- User defaults
- Archiving
- Notifications
- Undo manager
- Tasks, timers, threads
- File system, pipes, I/O, bundles

NSObject

- Root class
- Implements many basics
  - Memory management
  - Introspection
  - Object equality

NSString

- General-purpose Unicode string support
- Consistently used throughout Cocoa Touch instead of “char *”
- Without doubt the most commonly used class
- Easy to support any language in the world with Cocoa

String constants

- In C, constant strings are “string”
- In ObjC, constant strings are @”string”

Common NSString methods

- - (BOOL)isEqualToString:(NSString *)string;
- - (BOOL)hasPrefix:(NSString *)string;
- - (int)intValue;
- - (double)doubleValue;

NSMutableString

- NSMutableString subclasses NSString
- Allows a string to be modified
- Common NSMutableString methods

```
+ (id)string;
- (void)appendString:(NSString *)string;
- (void)appendFormat:(NSString *)format, ...;
```

NSArray

- Common NSArray methods
  - + arrayWithObjects: (id)firstObj, ...; // nil terminated!!!
  - - (unsigned)count;
  - - (id)objectAtIndex:(unsigned)index;
  - - (unsigned)indexOfObject:(id)object;
- NSNotFound returned for index if not found

NSMutableArray

- NSMutableArray subclasses NSArray
- So, everything in NSArray
- Common NSMutableArray Methods

```
+ (NSMutableArray *)array;
- (void)addObject:(id)object;
- (void)removeObject:(id)object;
- (void)removeAllObjects;
- (void)insertObject:(id)object atIndex:(unsigned)index;
```

NSDictionary

- Common NSDictionary methods

```
+ dictionaryWithObjectsAndKeys: (id)firstObject, ...;
- (unsigned)count;
- (id)objectForKey:(id)key;
```

NSMutableDictionary

- NSMutableDictionary subclasses NSDictionary
- Common NSMutableDictionary methods

```
+ (NSMutableDictionary *)dictionary;
- (void)setObject:(id)object forKey:(id)key;
- (void)removeObjectForKey:(id)key;
- (void)removeAllObjects;
```

NSSet

- Unordered collection of objects
- Common NSSet methods

```
+ setWithObjects:(id)firstObj, ...; // nil terminated
- (unsigned)count;
- (BOOL)containsObject:(id)object;
```

NSMutableSe

- NSMutableSet subclasses NSSet
- Common NSMutableSet methods

```
+ (NSMutableSet *)set;
- (void)addObject:(id)object;
- (void)removeObject:(id)object;
- (void)removeAllObjects;
- (void)intersectSet:(NSSet *)otherSet;
- (void)minusSet:(NSSet *)otherSet;
```

Enumeration

- Consistent way of enumerating over objects in collections
- Use with NSArray, NSDictionary, NSSet, etc.

NSNumber

- In Objective-C, you typically use standard C number types
- NSNumber is used to wrap C number types as objects
- Subclass of NSValue
- No mutable equivalent!
- Common NSNumber methods

```
+ (NSNumber *)numberWithInt:(int)value;
+ (NSNumber *)numberWithDouble:(double)value;
- (int)intValue;
- (double)doubleValue;
```

Other classes

- NSData/NSMutableData
- NSDate/NSCalendarDate
- NSThread
- NSTimer
- NSFileHandle
- NSStream

References

- http://developer.apple.com/iphone

Q & A

THANK YOU!!