CS 696 Mobile Application Development
Fall Semester, 2010
Doc 3 Objective C - Class Basics
Sep 2, 2010

References

The Objective-C 2.0 Programming Language, http://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html#//apple_ref/doc/uid/TP30001163

Classes & Instance

Class

- Template for objects
- Defines methods and instance variables for objects

Defining a Class

Rectangle.h

```
@interface Rectangle : NSObject
{
      int width;
      int height;
}

- (void) setWidth: (int) newWidth;
- (int) width;
- (void) setHeight: (int) newHeight;
- (int) height;
- (int) area;
@end
```

Rectangle.m

```
#import "Rectangle.h"
@implementation Rectangle

- (void) setWidth:(int)newWidth {
      width = newWidth;
}

- (int) width { return width;}

- (void) setHeight:(int)newHeight {
      height = newHeight;
}

- (int) height {return height; }
- (int) area { return width * height; }
@end
```

Using the Class

```
#import "Rectangle.h"

int main (int argc, const char * argv[]) {
      Rectangle * sample = [[Rectangle alloc] init];
   int initialValue = [sample width];
   NSLog(@"initialValue: %i", initialValue);
      [sample setWidth:4];
      [sample setHeight:5];
      int area = [sample area];
      NSLog(@"area %i", area);
      [sample release];
   return 0;
}
```


One can use printf("area %i", area) if you add "#import <stdio.h>"
NSLog provides a bit more information and is the standard in Objective-C

Header File Format

```
#import Imports here

@interface ClassName : ParentClass
{
      // Declare instance variables
}
   // Declare instance and class methods
- (returnType) instanceMethod;

+ (returnType) classMethod;
@end
```

Topics

New
NSObject
Multiple arguments
self, super
Instance and Class methods
Overloading
Overriding
Access levels
Constructors
Convenience constructors
Destructors
Description

new & alloc - init

[Rectangle new] same as [[Rectangle alloc] init]

Personal style on which to use

Most references use alloc & init

NSObject

Single root of class inheritance tree

Ancestor class of all Objective C classes

```
@interface Rectangle : NSObject
{
        int width;
        int height;
}
```

No implicit parent

```
@interface Rectangle
{
	int width;
	int height;
}
```


This rectangle class does not have a parent class. It does not inherit from NSObject and basically will not work

What is with the NS prefix?

No namespaces

Use package abbreviation as prefix for class names

NS - NextStep

Multiple Arguments

Rectangle.h

```
@interface Rectangle : NSObject
{
      int width;
      int height;
}
- (void) setHeight: (int) newHeight width: (int) newWidth;
@end
```

Rectangle.m

```
#import "Rectangle.h"
@implementation Rectangle

- (void) setHeight: (int) newHeight width: (int) newWidth {
       height = newHeight;
    width = newWidth;
}
@end
```

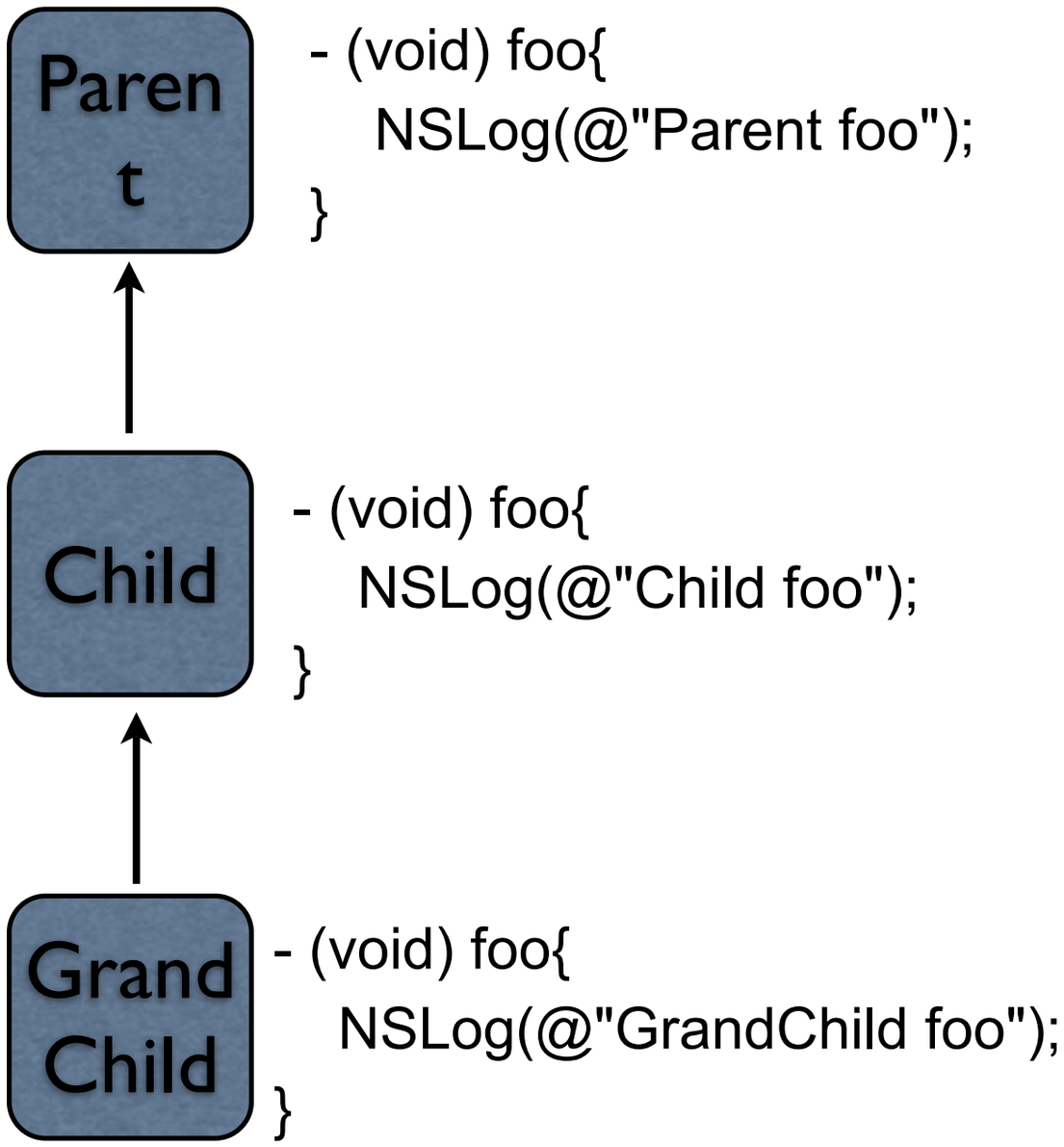
self & super

self - same as Java's this
super - same as Java's super

```
@implementation Rectangle

- (int) width { return width; }

- (int) height { return height; }

- (int) area {
      return [self width] * [self height];
}

@end
```

Example

```
Parent * example = [GrandChild new];
[example testSelf];
[example testSuper];
```



Child Methods

```
- (void) testSuper{
    NSLog(@"child testSuper");
    [super foo];
}
- (void) testSelf{
    NSLog(@"child testSelf");
    [self foo];
}
```

Instance & Class Methods

```
@interface ClassInstanceExample :
                                NSObject {

}
- (void) instanceMethod;
- (void) testMethod;
+ (void) classMethod;
@end
```

```
#import "ClassInstanceExample.h"

@implementation ClassInstanceExample

- (void) instanceMethod {
    NSLog(@"instance");
}

- (void) testMethod {
    [[self class] classMethod];
}

+ (void) classMethod {
    NSLog(@"class");
}
@end
```

Calling the methods

```
[ClassInstanceExample classMethod];
ClassInstanceExample * instance = [ClassInstanceExample new];
[instance instanceMethod];
[instance testMethod];
```

Static Variables

```
@interface Counter : NSObject
{
      int count;
}

- (int) instanceNext;
- (int) instanceGlobalAccess;
+ (int) classNext;
@end
```

```
#import "Counter.h"

static int gCount = 0;

@implementation Counter

-(int) instanceNext {
      return count++;
}

- (int) instanceGlobalAccess {
    return gCount++;
}

+ (int) classNext {
    return gCount++;
}
@end
```


Naming convention: start global variables with "g"
Static variables act like private class variables.

Initial Values

Instance variables & Globals

- initialized to 0 or null
- Except unions

Local variables

- not initialized

No Overloading methods

```
@interface OverLoadExample : NSObject {

}

- (void) method: (int) cat;
- (void) method: (NSString *) dog;  //Compile error
@end
```

Overriding Methods

```
@interface Parent : NSObject {
}
- (void) hello;
+ (void) bye;
@end
```

```
#import "Parent.h"

@implementation Parent

- (void) hello {
    NSLog(@"Parent Hello");
}

+ (void) bye {
    NSLog(@"Parent bye");
}
@end
```

Child Class

```
#import "Parent.h"

@interface Child : Parent {
}

@end
```

```
#import "Child.h"


@implementation Child

- (void) hello {
    [super hello];
    NSLog(@"Child Hello");
}

+ (void) bye {
    [super bye];
    NSLog(@"Child bye");
}

@end
```

Example

```
[Parent bye];                          //Parent bye
[Child bye];                           //Parent bye \n Child bye
Parent * example = [Child new];
[example hello];                       //Parent Hello \n Child hello
```

Instance Variable Access Levels

public
- All code can access

protected
- Access restricted to class & subclasses
- Default

private
- Access restricted to class

package
- Only available on 64-bit platforms

```
@interface Sample : NSObject
{
    int x;
    int y;
@private
    int a;
    int b;
@public
    int c;
@protected
    int d;
@private
    int e;
}
```

Method Access Levels

All methods are public

there are ways to fake private methods if you are so inclined

Dynamic Type Checking

Method calls are checked at runtime

```
@interface Foo : NSObject {
}

- (void) bar;

@end
```

```
Foo * test = [[Foo alloc] init];
[test catMan];
// Compile warning
// Runtime error
```


The compiler will look at methods you call and warn you if it thinks you are making a mistake

id

Pointer to object

```
Rectangle * justRectangle = [[Rectangle alloc] init];

id anyThing = [[Rectangle alloc] init];
anyThing = @"a String";
```


Recall that all objects are on the heap, so we always need a pointer to an object. id does not give the compiler any information about the type of the object.

Identity versus Equality

Identity - pointer comparison

```
if (objectA == objectB )
```

Equality - object attribute values are same

```
if ([objectA isEqual: objectB] )
```

description

Like Java's toString

```
@implementation Rectangle

- (NSString*) description {
    return [NSString stringWithFormat:@"Rect(%i,%i)", width, height];
}
//rest of class not shown
@end
```

```
Rectangle * sample = [[Rectangle alloc] init];
[sample setWidth:4];
[sample setHeight:5];
NSLog(@"%@", sample);
```

Output

Rect(4,5)

Simple Constructor

By convention constructors names start with init

Call [super init]

Normal return - self

On error -

Return nil

release self

```
@implementation Rectangle

- (id) init {
    if (self = [super init]) {
        width = 0;
        height = 0;
    }
    return self
}
```

[[Rectangle alloc] initWithWidth: 3 height: 4]

Constructors with Arguments

```
@interface Rectangle : NSObject { int width; int height; }

- (id) initWithWidth: (int) newWidth height: (int) newHeight;
@end
```

```
@implementation Rectangle

- (id) init { return [self initWithWidth: 0 height: 0]; }

- (id) initWithWidth: (int) newWidth height: (int) newHeight {
        if (self = [super init]) {
                width = newWidth;
                height = newHeight;
        }
        return self;
}
```