Classes

Name: Mahaboobsab Nadaf

* As in many other object-oriented programming languages, Objective-C classes provide the blueprint for creating objects.
* First, you define a reusable set of properties and behaviors inside of a class then, you instantiate objects from that class to interact with those properties and behaviors.
* Objective-C is abstracts a class’s interface from its implementation.
* An **interface** declares the public properties and methods of a class, and the corresponding **implementation** defines the code that actually makes these properties and methods work.
* In object-oriented programming terms, an object is an instance of a class.
* A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package.
* The data and methods within a class are called members of the class.

## Objective-C characteristics:

* The class is defined in two different sections namely **@interface** and **@implementation**.
* Almost everything is in form of objects.
* Objects receive messages and objects are often referred as receivers.
* Objects contain instance variables.
* Objects and instance variables have scope.
* Classes hide an object's implementation.
* Properties are used to provide access to class instance variables in other classes.

## Interfaces:

* An interface is created with the @interface directive, after which come the class and the super class name, separated by a colon.
* Protected variables can be defined inside of the curly braces, but most developers treat instance variables as implementation details and prefer to store them in the .m file instead of the interface.
* The @property directive declares a public property, and the (copy) attribute defines its memory management behavior.
* In this case, the value assigned to model will be stored as a copy instead of a direct pointer.
* The - (void) call line declares a method called drive that takes no parameters, and the (void) portion defines its return type.
* The minus sign indicates the method marks it as an instance method (opposed to a [class method](http://rypress.com/tutorials/objective-c/classes#class-methods-and-variables)).

**Example:**

// Mobile.h

#import <Foundation/Foundation.h>

@interface Mobile : NSObject {

// protected instance variables (not recommended)

}

@property (copy) NSString \*model;

- (void)call;

@end

Mobile.h contains some template code; this declares a property called model and a method called drive.

## Implementations

* The first thing any class implementation needs to do is import its corresponding interface.
* The @implementation directive is similar to @interface, except you don’t need to include the super class.
* Private instance variables can be stored between curly braces after the class name:

**Example:**

// Mobile.m

#import "Mobile.h"

@implementation Mobile {

// Private instance variables

double \_callTimer;

}

@synthesize model = \_model; // Optional for Xcode 4.4+

- (void) call {

NSLog(@"Calling to a number using %@ model", self.model);

}

@end

* @synthesize is a convenience directive that automatically generates accessor methods for the property.
* By default, the getter is simply the property name (model), and the setter is the capitalized name with the set prefix (setModel).
* This is much easier than manually creating accessor for every property.
* The \_model portion of the synthesize statement defines the private instance variable name to use for the property.

## Instantiation and Usage:

* A class provides the blueprints for objects, so basically an object is created from a class.
* Any files that need access to a class must import its header file (Mobile.h)—they should never, ever try to access the implementation file directly.

**Example:**

// main.m

#import <Foundation/Foundation.h>

#import "Mobile.h"

int main(int argc, const char \* argv[]) {

@autoreleasepool {

Mobile \*motorola = [[Mobile alloc] init];

[motorola setModel:@"motorola Corolla"];

NSLog(@"Created a %@", [motorola model]);

motorola.model = @"motorola Camry";

NSLog(@"Changed the Mobile to a %@", motorola.model);

[motorola call];

}

return 0;

}

* After the interface has been imported with the #import directive, you can instantiate objects with the alloc/init pattern shown above.
* As you can see, instantiation is a two-step process: first you must allocate some memory for the object by calling the alloc method, then you need to initialize it so it’s ready to use.
* You should never use an uninitialized object.
* To call a method on an Objective-C object, you place the instance and the method in square brackets, separated by a space.
* Arguments are passed after the method name, preceded by a colon.

**Example:**

motorola.setModel("3rd Gen");

## Class Methods and Variables:

* In Objective C it’s also possible to define class-level Methods and Variables.
* These are commonly called “static” methods/properties in other programming languages.
* Class method declarations look just like instance methods, except they are prefixed with a plus sign instead of a minus sign.

**Example:**

Let’s add the following class-level method to Mobile.h:

// Mobile.h

+ (void)setDefaultModel:(NSString \*)aModel;

* Similarly, a class method implementation is also preceded by a plus sign.

**Example:**

// Mobile.m

#import "Mobile.h"

static NSString \*\_defaultModel;

@implementation Mobile {

...

+ (void)setDefaultModel:(NSString \*)aModel {

\_defaultModel = [aModel copy];

}

@end

* The [aModel copy] call creates a copy of the parameter instead of assigning it directly.
* Class methods use the same square-bracket syntax as instance methods, but they must be called directly on the class, as shown below.

**Example:**

// Mobile.m

[Car setDefaultModel:@"Motorola 2nd Gen"];s