**Design Patterns**

(<http://www.raywenderlich.com/46988/ios-design-patterns>, <http://www.cs.colorado.edu/~kena/classes/5448/f12/presentation-materials/myrose.pdf>, https://developer.apple.com/legacy/library/documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaDesignPatterns/CocoaDesignPatterns.html)

**Design Patterns:**

Design patterns are reusable solutions to common problems in software design. They’re templates designed to help you write code that’s easy to understand and reuse. They also help you create loosely coupled code so that you can change or replace components in your code without too much of a hassle.

Most common design patterns are:

* *Structural*: MVC, Decorator, Adapter, Facade and Composite.
* *Creational*: Singleton, Abstract Factory and Cluster Class.
* *Behavioral*: Observer, Decorator Memento, Chain of Responsibility and Command.

1. **MVC – The King of Design Patterns**

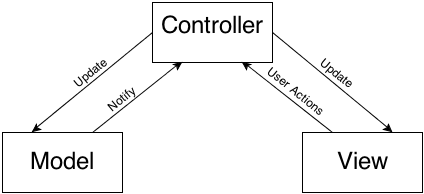
Model View Controller (MVC) is one of the building blocks of Cocoa and is undoubtedly the most-used design pattern of all. It classifies objects according to their general role in your application and encourages clean separation of code based on role.

The three roles are:

* *Model*: The object that holds your application data and defines how to manipulate it.
* *View*: The objects that are in charge of the visual representation of the Model and the controls the user can interact with; basically, all the UIViews and their subclasses.
* *Controller*: The controller is the mediator that coordinates all the work. It accesses the data from the model and displays it with the views, listens to events and manipulates the data as necessary.

A good implementation of this design pattern in your application means that each object falls into one of these groups.

The communication between View to Model through Controller can be best described with the following diagram:



The Model notifies the Controller of any data changes, and in turn, the Controller updates the data in the Views. The View can then notify the Controller of actions the user performed and the Controller will either update the Model if necessary or retrieve any requested data.

You might be wondering why you can’t just ditch the Controller, and implement the View and Model in the same class, as that seems a lot easier.

It all comes down to code separation and reusability. Ideally, the View should be completely separated from the Model. If the View doesn’t rely on a specific implementation of the Model, then it can be reused with a different model to present some other data.

1. **The Decorator Design Patterns**

The Decorator pattern dynamically adds behaviors and responsibilities to an object without modifying its code. It’s an alternative to subclassing where you modify a class’ behavior by wrapping it with another object.

In Objective-C there are two *very* common implementations of this pattern: *Category* and *Delegation*.

### Category:

Category is an extremely powerful mechanism that allows you to add methods to existing classes without subclassing. The new methods are added at compile time and can be executed like normal methods of the extended class. It’s slightly different from the classic definition of a decorator, because a Category doesn’t hold an instance of the class it extends.

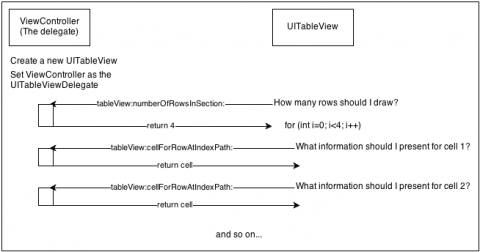
*Note:* Besides extending your own classes, you can also add methods to any of Cocoa’s own classes!

### Delegation (Q. What is Delegation in iOS? -> It is a design pattern)

The other Decorator design pattern, Delegation, is a mechanism in which one object acts on behalf of, or in coordination with, another object. For example, when you use a UITableView, one of the methods you must implement is *tableView:numberOfRowsInSection:*.

You can’t expect the UITableView to know how many rows you want to have in each section, as this is application-specific. Therefore, the task of calculating the amount of rows in each section is passed on to the UITableView delegate. This allows the UITableView class to be independent of the data it displays.

Here’s a pseudo-explanation of what’s going on when you create a new UITableView:



The UITableView object does its job of displaying a table view. However, eventually it will need some information that it doesn’t have. Then, it turns to its delegates and sends a message asking for additional information. In Objective-C’s implementation of the delegate pattern, a class can declare optional and required methods through a protocol.

It might seem easier to just subclass an object and override the necessary methods, but consider that you can only subclass based on a single class. If you want an object to be the delegate of two or more other objects, you won’t be able to achieve this by subclassing.

*Note:* This is an important pattern. Apple uses this approach in most of the UIKit classes: UITableView, UITextView, UITextField, UIWebView, UIAlert, UIActionSheet, UICollectionView, UIPickerView, UIGestureRecognizer, UIScrollView. The list goes on and on.

1. **The Adapter Pattern**

An Adapter allows classes with incompatible interfaces to work together. It wraps itself around an object and exposes a standard interface to interact with that object.

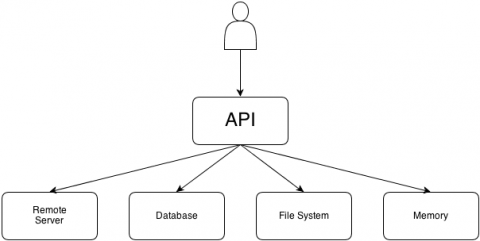
If you’re familiar with the Adapter pattern then you’ll notice that Apple implements it in a slightly different manner – Apple uses protocols to do the job. You may be familiar with protocols like UITableViewDelegate, UIScrollViewDelegate, NSCoding and NSCopying. As an example, with the NSCopying protocol, any class can provide a standard copy method.

A simple example of an Adapter using a category that I use in my projects is as follows: Interface Builder (IB) includes a feature called "User Defined Runtime Attributes" that lets you set properties on your custom interface objects using Key Value Coding (KVC). It lets you specify a limited number of data types (ints, floats, bools, points, rects, UIColors, and a few others.) You can use User Defined Runtime Attributes to set the border width and corner radius on a view's layer, and you SHOULD be able to use it to change the layer's border color or background color. However, layer colors are specified as CGColors, and UIViews use UIColors. Since IB only accepts UIColors in a User Defined Runtime Attribute, it doesn't work.

To fix this, I created a category of CALayer called CALayer+setUIColor. It has 2 methods, setBorderUIColor and setBackgroundUIColor. Those methods are very simple. They take UIColors as input, and simply convert the UIColor to a CGColor and set the layer's border color or background color.

1. **The Façade Design Pattern**

The Facade design pattern provides a unified interface to a set of interfaces in a subsystem. The pattern defines a higher-level interface that makes the subsystem easier to use by reducing complexity and hiding the communication and dependencies between subsystems. The following image explains this concept:



It allows for a subsystem to be accessed through one entry point, allowing the systems using it to be unaware of the classes in the subsystem. One example of this in iOS is the *NSImage* class

The NSImage class of the AppKit framework provides a unified interface for loading and using images that can be bitmap-based (such as those in JPEG, PNG, or TIFF format) or vector-based (such as those in EPS or PDF format). NSImage can keep more than one representation of the same image; each representation is a kind of NSImageRep object. NSImage automates the choice of the representation that is appropriate for a particular type of data and for a given display device. It also hides the details of image manipulation and selection so that the client can use many different underlying representations interchangeably.

##### Uses and Limitations: Because NSImage supports several different representations of what an image is, some requested attributes might not apply. For example, asking an image for the color of a pixel does not work if the underlying image representation is vector-based and device-independent.

1. **Composite Design Patterns**

The Composite design pattern composes related objects into tree structures to represent part-whole hierarchies. The pattern lets clients treat individual objects and compositions of objects uniformly. The Composite pattern is part of the Model-View-Controller aggregate pattern,

1. **The Singleton Design Pattern**

The Singleton design pattern ensures that only one instance exists for a given class and that there’s a global access point to that instance. It usually uses lazy loading to create the single instance when it’s needed the first time.

*Note:* Apple uses this approach a lot. For example: [NSUserDefaults standardUserDefaults], [UIApplication sharedApplication], [UIScreen mainScreen], [NSFileManager defaultManager] all return a Singleton object.

You’re likely wondering why you care if there’s more than one instance of a class floating around. Code and memory is cheap, right?

There are some cases in which it makes sense to have exactly one instance of a class. For example, there’s no need to have multiple Logger instances out there, unless you want to write to several log files at once. Or, take a global configuration handler class: it’s easier to implement a thread-safe access to a single shared resource, such as a configuration file, than to have many class modifying the configuration file possibly at the same time.

1. **Abstract Factory Design Pattern**

The Abstract Factory pattern provides an interface for creating families of related or dependent objects without specifying their concrete classes. The client is decoupled from any of the specifics of the concrete object obtained from the factory.

#### Class Cluster

A class cluster is an architecture that groups a number of private concrete subclasses under a public abstract superclass. The abstract superclass declares methods for creating instances of its private subclasses. The superclass dispenses an object of the proper concrete subclass based on the creation method invoked. Each object returned may belong to a different private concrete subclass.

Class clusters in Cocoa can generate only objects whose storage of data can vary depending on circumstances. The Foundation framework has class clusters for [NSString](https://developer.apple.com/legacy/library/documentation/LegacyTechnologies/WebObjects/WebObjects_3.5/Reference/Frameworks/ObjC/Foundation/Classes/NSStringClassCluster/Description.html#//apple_ref/occ/cl/NSString), [NSData](https://developer.apple.com/legacy/library/documentation/LegacyTechnologies/WebObjects/WebObjects_3.5/Reference/Frameworks/ObjC/Foundation/Classes/NSDataClassCluster/Description.html#//apple_ref/occ/cl/NSData), [NSDictionary](https://developer.apple.com/legacy/library/documentation/LegacyTechnologies/WebObjects/WebObjects_3.5/Reference/Frameworks/ObjC/Foundation/Classes/NSDictionaryClassClstr/Description.html#//apple_ref/occ/cl/NSDictionary), [NSSet](https://developer.apple.com/legacy/library/documentation/LegacyTechnologies/WebObjects/WebObjects_3.5/Reference/Frameworks/ObjC/Foundation/Classes/NSSetClassCluster/Description.html#//apple_ref/occ/cl/NSSet), and [NSArray](https://developer.apple.com/legacy/library/documentation/LegacyTechnologies/WebObjects/WebObjects_3.5/Reference/Frameworks/ObjC/Foundation/Classes/NSArrayClassCluster/Description.html#//apple_ref/occ/cl/NSArray) objects. The public superclasses include these immutable classes as well as the complementary mutable classes [NSMutableString](https://developer.apple.com/legacy/library/documentation/LegacyTechnologies/WebObjects/WebObjects_3.5/Reference/Frameworks/ObjC/Foundation/Classes/NSStringClassCluster/Description.html#//apple_ref/occ/cl/NSMutableString), [NSMutableData](https://developer.apple.com/legacy/library/documentation/LegacyTechnologies/WebObjects/WebObjects_3.5/Reference/Frameworks/ObjC/Foundation/Classes/NSDataClassCluster/Description.html#//apple_ref/occ/cl/NSMutableData), [NSMutableDictionary](https://developer.apple.com/legacy/library/documentation/LegacyTechnologies/WebObjects/WebObjects_3.5/Reference/Frameworks/ObjC/Foundation/Classes/NSDictionaryClassClstr/Description.html#//apple_ref/occ/cl/NSMutableDictionary), [NSMutableSet](https://developer.apple.com/legacy/library/documentation/LegacyTechnologies/WebObjects/WebObjects_3.5/Reference/Frameworks/ObjC/Foundation/Classes/NSSetClassCluster/Description.html#//apple_ref/occ/cl/NSMutableSet), and [NSMutableArray](https://developer.apple.com/legacy/library/documentation/LegacyTechnologies/WebObjects/WebObjects_3.5/Reference/Frameworks/ObjC/Foundation/Classes/NSArrayClassCluster/Description.html#//apple_ref/occ/cl/NSMutableArray).

##### ***Uses and Limitations***

You use one of the public classes of a class cluster when you want to create immutable or mutable objects of the type represented by the cluster. With class clusters there is a trade-off between simplicity and extensibility. A class cluster simplifies the interface to a class and thus makes it easier to learn and use the class. However, it is generally more difficult to create custom subclasses of the abstract superclass of a class cluster.

1. **Observer Design Pattern**

The Observer design pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. The Observer pattern is essentially a publish-and-subscribe model in which the subject and its observers are loosely coupled. Communication can take place between the observing and observed objects without either needing to know much about the other. This pattern’s most often used to notify interested objects when a property has changed.

The usual implementation requires that an observer register interest in the state of another object. When the state changes, all the observing objects are notified of the change. Apple’s Push Notification service is a global example of this.

If you want to stick to the MVC concept (hint: you do), you need to allow Model objects to communicate with View objects, but without direct references between them. And that’s where the Observer pattern comes in.

Cocoa implements the observer pattern in two familiar ways: *Notifications* and *Key-Value Observing (KVO)*.

The observer pattern is similar to the delegate pattern, however one key difference is that observable objects support multiple observers, while a delegate is just one object. However, with this expanded possibility comes one big pitfall: you must remember to remove an object as an observer when that object is deallocated , otherwise there will be a code leak

1. **Memento Design Pattern**

The memento pattern captures and externalizes an object’s internal state. In other words, it saves your stuff somewhere. Later on, this externalized state can be restored without violating encapsulation; that is, private data remains private.

- (void)saveCurrentState {

*// When the user leaves the app and then comes back again, he wants it to be in the exact same state*

*// he left it. In order to do this we need to save the currently displayed album.* *// Since it's only one piece of information we can use NSUserDefaults.*

[[[NSUserDefaults](http://developer.apple.com/documentation/Cocoa/Reference/Foundation/Classes/NSUserDefaults_Class/) standardUserDefaults] setInteger:currentAlbumIndex forKey:*@*"currentAlbumIndex"];

}

- (void)loadPreviousState {

currentAlbumIndex = [[[NSUserDefaults](http://developer.apple.com/documentation/Cocoa/Reference/Foundation/Classes/NSUserDefaults_Class/) standardUserDefaults] integerForKey:*@*"currentAlbumIndex"];

[self showDataForAlbumAtIndex:currentAlbumIndex];

}

saveCurrentState saves the current album index to NSUserDefaults – NSUserDefaults is a standard data store provided by iOS for saving application specific settings and data.

loadPreviousState loads the previously saved index. This isn’t quite the full implementation of the Memento pattern, but you’re getting there.

### Archiving:

One of Apple’s specialized implementations of the Memento pattern is Archiving. This converts an object into a stream that can be saved and later restored without exposing private properties to external classes.

1. **Chain of Responsibility Design Pattern**

The Chain of Responsibility design pattern decouples the sender of a request from its receiver by giving more than one object a chance to handle the request. The pattern chains the receiving objects together and passes the request along the chain until an object handles it. Each object in the chain either handles the request or passes it to the next object in the chain

1. **Command Design Pattern**

The Command design pattern encapsulates a request or action as an object. The encapsulated request is much more flexible than a raw request and can be passed between objects, stored for later, modified dynamically, or placed into a queue. Apple has implemented this pattern using the *Target-Action* mechanism and *Invocation*.

Invocation uses the *NSInvocation* class which contains a target object, a method selector and some parameters. This object can be changed dynamically and executed when needed. It’s a perfect example of the Command pattern in action. It decouples the sending object from the receiving object or objects and can persist a request or a chain of requests.

**Sub Class, Category and Extensions**

**Sub Class:**

Sub classing in simple words is **changing the behavior of properties or methods of an existing class** or in other words sub classing is inheriting a class and modifying the methods or properties of super class however you want.

For example consider a UITextField class, by default the placeholder text of UITextField will be of light gray color with default system font. If we want to change this style, just create a subclass of UITextField and override drawPlaceholderInRect method. For that, Create a class of type UITextField and name it some thing like CustomUITextFieldPlaceholderAppearance

***CustomUITextFieldPlaceholderAppearance.h***

#import <UIKit/UIKit.h>

@interface CustomUITextFieldPlaceholderAppearance : UITextField

@end

***CustomUITextFieldPlaceholderAppearance.m***

#import "CustomUITextFieldPlaceholderAppearance.h"

@implementation CustomUITextFieldPlaceholderAppearance

// Override **drawPlaceholderInRect** method

- (void) drawPlaceholderInRect:(CGRect)rect {

   // Set color and font size and style of placeholder text

     [[UIColor redColor] setFill];//set placeholder text color to red

     [[self placeholder] drawInRect:rect withFont:[UIFont fontWithName:@"verdana" size:14.0]]; //set custom font style & size to placeholder

}

@end

Now in your application wherever you want this custom look and feel for placeholder text of textfield you can just import this subclass header file (#import "CustomUITextFieldPlaceholderAppearance.h*"*) and create an object of this class and you are done. In addition to this look and feel the default delegate methods and properties of UITextField will remain same

**Category:**

An Objective C category allows you **add your own methods to an existing class.** Categories are also called as "informal protocols".

For example, since Foundation Framework classes such as NSString, NSArray, NSDate etc… doesn’t have any access to modify, you can add your own methods in to these classes by the help of a category.

Consider NSString Class and if suppose we want to add a reverse string method to NSString class, so that in our application at any point of time any NSString object can call this category method and get a reversed string as a result. We can do this as below. Usually naming convention for category file is like **OriginalClassName+CategoryName**

Lets create a category class with a name something like NSString+NSString\_ReverseString

***NSString+NSString\_ReverseString.h***

#import <Foundation/Foundation.h>

@interface NSString (NSString\_ReverseString)

- (NSString \*)reverseString:(NSString \*)yourString;

@end

***NSString+NSString\_ReverseString.m***

#import "NSString+NSString\_ReverseString.h"

@implementation NSString (NSString\_ReverseString)

- (NSString \*)reverseString:(NSString \*)yourString {

NSMutableString \*reversedStr = [NSMutableString stringWithCapacity:[yourString length]];

[yourString enumerateSubstringsInRange:NSMakeRange(0,[yourString length])options:(NSStringEnumerationReverse | NSStringEnumerationByComposedCharacterSequences)usingBlock:^(NSString \*substring, NSRange substringRange, NSRange enclosingRange, BOOL \*stop) {

[reversedStr appendString:substring];

 }];

     return reversedStr;

}

@end

Now in your application wherever you want  to reverse a string then just import this category header file (#import "NSString+NSString\_ReverseString.h") and call our **reverseString:** method from any of NSString object and it will reverse and return you the reversed string.

In the above example we have added a custom method called **reverseString:**to NSString class from the help of a category.

**Difference between Sub Class and Category:**

Finally few simple points to remember is

**- Subclassing is better option if you want to customize an existing stuffs or functionalities, and**

**- Category is a best option if you want to add additional functionalities to an existing class**

**Extensions:**

Extensions are similar to categories but the need of extension is different.

- Class extensions are often used to extend the public interface with additional private methods or properties for use within the implementation of the class.

- Extensions can only be added to a class for which you have the source code at compile time (the class is compiled at the same time as the class extension).

- Extensions will be local to a class file.

The syntax to declare class extension looks like,

@interface ClassName()

@end

Since no name is given in the parentheses, class extensions are often referred to as **anonymous categories**. Note Extensions can add instance variables.

@interface ABCExtension()

@property NSObject \*yourProperty;

@end

The compiler will automatically synthesize the relevant accessor methods, as well as an instance variable, inside the primary class implementation. If you add any methods in a class extension, these must be implemented in the primary implementation for the class.

In any of your class in implementation file(.m), say ***ViewController.m***

@interface ViewController ()

-(void)printName:(NSString \*)name;

@end

@implementation ViewController

-(void)printName:(NSString \*)name {

NSLog(@"%@",name);

}

@end

In the above extension example printName: method is private to class ViewController, and cannot be accessed from outside the ViewController class. Even if you inherit since printName: is a private method there will not be any access to this method outside the class. You can call this extension method only inside ViewController class, as below

[self printName:@"MyName"];

Usually people will use extensions to hide private information of a class without exposing them to access from any other class.

**GCD (Grand Central Dispatch)**

(<http://www.raywenderlich.com/60749/grand-central-dispatch-in-depth-part-1>, https://developer.apple.com/library/mac/documentation/Performance/Reference/GCD\_libdispatch\_Ref/index.html)

**GCD:**

GCD (Grand Central Dispatch) provides and manages FIFO queues to which your application can submit tasks in the form of block objects. Blocks submitted to dispatch queues are executed on a set of threads completely managed by the system. No guarantee is made as to the thread on which a task executes. GCD allows three kinds of queues:

* + **Main:** tasks execute serially on your application’s main thread
  + **Concurrent:** tasks are dequeued in FIFO order, but run concurrently and can finish in any order.
  + **Serial:** tasks execute one at a time in FIFO order

GCD is the marketing name for *libdispatch*, Apple’s library that provides support for concurrent code execution on multicore hardware on iOS and OS X. It offers the following benefits:

* + GCD can improve your app’s responsiveness by helping you defer computationally expensive tasks and run them in the background.
  + GCD provides an easier concurrency model than locks and threads and helps to avoid concurrency bugs.
  + GCD can potentially optimize your code with higher performance primitives for common patterns such as singletons.

**Main & Serial:**

The more important functions you’ll need are for creating the queue:

dispatch\_queue\_t dispatch\_queue\_create(const char \*label, dispatch\_queue\_attr\_t attr);

and adding blocks to the queue:

void dispatch\_async(dispatch\_queue\_t queue, dispatch\_block\_t block);

There’s also a couple helper functions for retrieving specific queues such as:

dispatch\_queue\_t dispatch\_get\_current\_queue(void);

dispatch\_queue\_t dispatch\_get\_main\_queue(void);

The dispatch\_get\_current\_queue function will return the current queue from which the block is dispatched and the dispatch\_get\_main\_queue function will return the main queue where your UI is running.

The dispatch\_get\_main\_queue function is very useful for updating the iOS app’s UI, as UIKit methods are **not** thread safe (with a few exceptions) so any calls you make to update UI elements must always be done from the main queue.

A typical GCD call would look something like this:

// Doing something on the main thread

dispatch\_queue\_t myQueue = dispatch\_queue\_create("My Queue",NULL);

dispatch\_async(myQueue, ^{

// Perform long running process

dispatch\_async(dispatch\_get\_main\_queue(), ^{

// Update the UI

});

});

// Continue doing other stuff on the

// main thread while process is running.

**Concurrent:**

If you want to run a single independent queued operation and you’re not concerned with other concurrent operations, you can use the global concurrent queue:

dispatch\_queue\_t globalConcurrentQueue = dispatch\_get\_global\_queue(DISPATCH\_QUEUE\_PRIORITY\_DEFAULT, 0)

This will return a concurrent queue with the given priority

* + DISPATCH\_QUEUE\_PRIORITY\_HIGH
  + DISPATCH\_QUEUE\_PRIORITY\_DEFAULT
  + DISPATCH\_QUEUE\_PRIORITY\_LOW
  + DISPATCH\_QUEUE\_PRIORITY\_BACKGROUND

**Difference: GCD & NSOperation/NSOperationQueue**

GCD is a lightweight way to represent units of work that are going to be executed concurrently. You don’t schedule these units of work; the system takes care of scheduling for you. Adding dependency among blocks can be a headache. Canceling or suspending a block creates extra work for you as a developer!

NSOperation and NSOperationQueue add a little extra overhead compared to GCD, but you can add dependency among various operations. You can re-use operations, cancel or suspend them. NSOperation is compatible with Key-Value Observation (KVO); for example, you can have an NSOperation start running by listening to NSNotificationCenter.

GCD is indeed lower-level than NSOperationQueue, its major advantage is that its implementation is very light-weight and focused on lock-free algorithms and performance.

NSOperationQueue does provide facilities that are not available in GCD, but they come at non-trivial cost, the implementation of NSOperationQueue is complex and heavy-weight, involves a lot of locking, and uses GCD internally only in a very minimal fashion.

If you need the facilities provided by NSOperationQueue by all means use it, but if GCD is sufficient for your needs, I would recommend using it directly for better performance, significantly lower CPU and power cost and more flexibility.

**NSOperation & NSOperationQueue**

(<http://www.raywenderlich.com/19788/how-to-use-nsoperations-and-nsoperationqueues>, https://developer.apple.com/library/ios/documentation/General/Conceptual/ConcurrencyProgrammingGuide/OperationQueues/OperationQueues.html)

**NSOperation:**

NSOperation encapsulate the code and data associated with a single task. It's an abstract class, you do not use this class directly but instead subclass or use one of the system-defined subclasses to perform the actual task. The base implementation of NSOperation does include significant logic to coordinate the safe execution of your task. NSOperation offers a useful, thread-safe structure for modeling state, priority, dependencies, and management.

The presence of this built-in logic allows you to focus on the actual implementation of your task, rather than on the glue code needed to ensure it works correctly with other system objects.

For situations where it doesn't make sense to build out a custom NSOperation subclass, Foundation provides the concrete implementations [NSBlockOperation](https://developer.apple.com/library/ios/documentation/cocoa/reference/NSBlockOperation_class/Reference/Reference.html) and [NSInvocationOperation](https://developer.apple.com/library/mac/documentation/cocoa/reference/NSInvocationOperation_Class/Reference/Reference.html).

An operation object is a single-shot object—that is, it executes its task once and cannot be used to execute it again. You typically execute operations by adding them to an operation queue (an instance of the NSOperationQueue lass). An operation queue executes its operations either directly, by running them on secondary threads, or indirectly using the libdispatch library (also known as Grand Central Dispatch). For more information about how queues execute operations.

If you do not want to use an operation queue, you can execute an operation yourself by calling its start method directly from your code. Executing operations manually does put more of a burden on your code, because starting an operation that is not in the ready state triggers an exception. The ready property reports on the operation’s readiness.

**NSOperationQueue:**

The NSOperationQueue class regulates the execution of a set of [NSOperation](https://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSOperation_class/index.html#//apple_ref/occ/cl/NSOperation) objects. After being added to a queue, an operation remains in that queue until it is explicitly canceled or finishes executing its task. Operations within the queue (but not yet executing) are themselves organized according to priority levels and inter-operation object dependencies and are executed accordingly. An application may create multiple operation queues and submit operations to any of them.

Inter-operation dependencies provide an absolute execution order for operations, even if those operations are located in different operation queues. An operation object is not considered ready to execute until all of its dependent operations have finished executing. For operations that are ready to execute, the operation queue always executes the one with the highest priority relative to the other ready operations.

NSOperationQueue acts as a priority queue, such that operations are executed in a roughly First-In-First-Out manner, with higher-priority (NSOperation.queuePriority) ones getting to jump ahead of lower-priority ones. NSOperationQueue can also limit the maximum number of concurrent operations to be executed at any given moment, using the maxConcurrentOperationCount property.

## State:

NSOperation encodes a rather elegant state machine to describe the execution of an operation:

ready → executing → finished

State is determined implicitly by KVO notifications on those keypaths. When an operation is ready to be executed, it sends a KVO notification for the ready keypath, whose corresponding property would then return true.

Each property must be mutually exclusive from one another in order to encode a consistent state:

1. ready: Returns true to indicate that the operation is ready to execute, or false if there are still unfinished initialization steps on which it is dependent.
2. executing: Returns true if the operation is currently working on its task, or false otherwise.
3. finished Returns true if the operation's task finished execution successfully, or if the operation was cancelled. An NSOperationQueue does not dequeue an operation until finished changes to true, so it is *critical* to implement this correctly in subclasses to avoid deadlock.

## Priority:

All operations may not be equally important. Setting the queuePriority property will promote or defer an operation in an NSOperationQueue according to the following rankings:

typedef enum : NSInteger {

NSOperationQueuePriorityVeryLow = -8,

NSOperationQueuePriorityLow = -4,

NSOperationQueuePriorityNormal = 0,

NSOperationQueuePriorityHigh = 4,

NSOperationQueuePriorityVeryHigh = 8

} NSOperationQueuePriority;

**Size of all devices in inch and pixels**

<http://www.gsmarena.com/apple_iphone_3g-2424.php>

<http://www.gsmarena.com/apple_iphone_3gs-2826.php>

<http://www.iosres.com/>

<https://www.apple.com/in/iphone/compare/>

<http://www.apple.com/in/iphone-5s/specs/>

Blocks

Block objects are a C-level syntactic and runtime feature. Blocks are particularly useful as a callback because the block carries both the code to be executed on callback (on completion handler block) and the data needed during that execution.

**Simple Block declaration syntax**

returntype (^blockName)(argumentType);

Simple block implementation

returntype (^blockName)(argumentType)= ^{ };

**Here is a simple example**

void (^simpleBlock)(void) = ^{     NSLog(@"This is a block"); };

**We can invoke the block using**

simpleBlock();

1. Protocol
2. Blocks
3. Size of all devices in inch and pixels
4. Variables as block
5. SOAP
6. Distance between two locations
7. Thread
8. Asynchronous and Synchronous
9. XML Parsing
10. Auto Layout and Auto Sizing
11. Size Class
12. RSS Feed
13. Types of Layer
14. Lazy Loading
15. Find occupied memory of one View Controller
16. Perform selector
17. Atomic and nonatomic
18. Strong and weak
19. Retain and copy
20. Application State
21. In app purchase types
22. Copy and Mutable Copy
23. Not consume battery when app is in background
24. Delegates and notifications
25. What is the maximum no of UILocalNotifications

An app can have only a limited number of scheduled notifications; the system keeps the soonest-firing 64 notifications (with automatically rescheduled notifications counting as a single notification) and discards the rest.

1. Maximum time of Alert sound like alarm
   1. 30 seconds
2. Maximum length of a Push Notification Alert text

In iOS 8 and later, the maximum size allowed for a notification payload is 2 kilobytes; Apple Push Notification service refuses any notification that exceeds this limit. (Prior to iOS 8 and in OS X, the maximum payload size is 256 bytes.)

1. Maximum length of a Push Notification Alert text

* Does iOS support multitasking?

iOS 4 and above supports multi – tasking and allows apps to remain in the background until they are launched again or until they are terminated.

* Which JSON framework is supported by iOS?

SBJson framework is supported by iOS. It is a JSON parser and generator for Objective-C. SBJson provides flexible APIs and additional controls that makes JSON handling easier.

* What are the tools required to develop iOS applications?

iOS development requires Intel-based Macintosh computer and iOS SDK.

* Name the framework that is used to construct application’s user interface for iOS.

The UIKit framework is used to develop application’s user interface for iOS. UIKit framework provides handling, drawing model, windows, views, and controls specifically designed for a touch screen interface.

* Name the application thread from where UIKit classes should be used?

UIKit classes should be used only from an application’s main thread. Note: The derived classes of UIResponder and the classes, which manipulate application’s user interface, should be used from application’s main thread.

* Which API is used to write test scripts that help in exercising the application’s user interface elements?

UI Automation API is used to automate test procedures. Tests scripts are written in JavaScript to the UI Automation API. This in turn simulates user interaction with the applications and returns log information to the host computer.

* Why an app on iOS device behaves differently when running in foreground than in background?

An application behaves differently when running in foreground than in background because of the limitation of resources on iOS devices.

* How can an operating system improve battery life while running an app?

An app is notified whenever the operating system moves the apps between foreground and background. The operating system improves battery life while it bounds what your app can do in the background. This also improves the user experience with foreground app.

* Which framework delivers event to custom object when app is in foreground?

The UIKit infrastructure takes care of delivering events to custom objects. As an app developer, you have to override methods in the appropriate objects to process those events.

* When an app is said to be in not running state?

An app is said to be in ‘not running’ state when:

* It is not launched.
* It gets terminated by the system during running.
* Assume that your app is running in the foreground but is currently not receiving events. In which stare it would be?

An app will be in In-Active state if it is running in the foreground but is currently not receiving events. An app stays in In-Active state only briefly as it transitions to a different state.

* Give example scenarios when an application goes into In-Active state?

An app can get into In-Active state when the user locks the screen or the system prompts the user to respond to some event e.g. SMS message, incoming call etc.

* When an app is said to be in active state?

An app is said to be in active state when it is running in foreground and is receiving events.

* Name the app state, which it reaches briefly on its way to being suspended.

An app enters background state briefly on its way to being suspended.

* Assume that an app is not in foreground but is still executing code. In which state will it be?

Background state.

* An app is loaded into memory but is not executing any code. In which state will it be?

An app is said to be in suspended state when it is still in memory but is not executing any code.

* Assume that system is running low on memory. What can system do for suspended apps?

In case system is running low on memory, the system may purge suspended apps without notice.

* How can you respond to state transitions on your app?

On state transitions can be responded to state changes in an appropriate way by calling corresponding methods on app’s delegate object.

For Example:

applicationDidBecomeActive method can be used to prepare to run as the foreground app.

applicationDidEnterBackground method can be used to execute some code when app is running in the background and may be suspended at any time.

applicationWillEnterForeground method can be used to execute some code when your app is moving out of the background.

applicationWillTerminate method is called when your app is being terminated.

* List down app’s state transitions when it gets launched.

Before the launch of an app, it is said to be in not running state.

When an app is launched, it moves to the active or background state, after transitioning briefly through the in-active state.

* Who calls the main function of your app during the app launch cycle?

During app launching, the system creates a main thread for the app and calls the app’s main function on that main thread. The Xcode project's default main function hands over control to the UIKit framework, which takes care of initializing the app before it is run.

* What is the use of controller object UIApplication?

Controller object UIApplication is used without subclassing to manage the application event loop. It coordinates other high-level app behaviors. It works along with the app delegate object, which contains app-level logic.

* Which object is creating by UIApplicationMain function at app launch time?

The app delegate object is created by UIApplicationMain function at app launch time. The app delegate object's main job is to handle state transitions within the app.

* How is the app delegate is declared by Xcode project templates?

App delegate is declared as a subclass of UIResponder by Xcode project templates.

* What happens if Application object does not handle an event?

In such case the event will be dispatched to your app delegate for processing.

* Which app specific objects store the app's content?

Data model objects are app specific objects and store app’s content. Apps can also use document objects to manage some or all of their data model objects.

* Are document objects required for an application? What does they offer?

Document objects are not required but are very useful in grouping data that belongs in a single file or file package.

* Which object manage the presentation of app's content on the screen?

View controller object takes care of the presentation of app's content on the screen. A view controller is used to manage a single view along with the collection of sub views. It makes its views visible by installing them in the app’s window.

* Which is the super class of all view controller objects?

UIViewController class. UIViewController class provides functionality for loading views, presenting them, rotating them in response to device rotations, and several other standard system behaviors.

* What is the purpose of UIWindow object?

The presentation of one or more views on a screen is coordinated by UIWindow object.

* How do you change the content of your app in order to change the views displayed in the corresponding window?

To change the content of your app, you use a view controller to change the views displayed in the corresponding window. Remember, window itself is never replaced.

* Define view object?

Views along with controls are used to provide visual representation of the app content. View is an object that draws content in a designated rectangular area and it responds to events within that area.

* You wish to define your custom view. Which class will be sub classed?

Custom views can be defined by sub classing UIView.

* Apart from incorporating views and controls, what else an app can incorporate?

Apart from incorporating views and controls, an app can also incorporate Core Animation layers into its view and control hierarchies.

* What are layer objects and what do they represent?

Layer objects are data objects which represent visual content. Layer objects are used by views to render their content. Custom layer objects can also be added to the interface to implement complex animations and other types of sophisticated visual effects.

* If I call performSelector:withObject:afterDelay: – is the object retained?

Yes, the object is retained. It creates a timer that calls a selector on the current threads run loop. It may not be 100% precise time-wise as it attempts to dequeue the message from the run loop and perform the selector.

* Can you explain what happens when you call autorelease on an object?

When you send an object a autorelease message, its retain count is decremented by 1 at some stage in the future. The object is added to an autorelease pool on the current thread. The main thread loop creates an autorelease pool at the beginning of the function, and releases it at the end. This establishes a pool for the lifetime of the task. However, this also means that any autoreleased objects created during the lifetime of the task are not disposed of until the task completes. This may lead to the taskʼs memory footprint increasing unnecessarily. You can also consider creating pools with a narrower scope or use NSOperationQueue with itʼs own autorelease pool. (Also important – You only release or autorelease objects you own.)

* What is the NSCoder class used for?

NSCoder is an abstractClass, which represents a stream of data. They are used in Archiving and Unarchiving objects. NSCoder objects are usually used in a method that is being implemented so that the class conforms to the protocol. (Which has something like encodeObject and decodeObject methods in them).

* What is an NSOperationQueue and how/would you use it?

The NSOperationQueue class regulates the execution of a set of NSOperation objects. An operation queue is generally used to perform some asynchronous operations on a background thread so as not to block the main thread.

* Explain the correct way to manage Outlets memory?

Create them as properties in the header that are retained. In the viewDidUnload set the outlets to nil(i.e self.outlet = nil). Finally in dealloc make sure to release the outlet.

* What is sandbox?

For security reasons, iOS places each app (including its preferences and data) in a sandbox at install time. A sandbox is a set of fine-grained controls that limit the app’s access to files, preferences, network resources, hardware, and so on. As part of the sandboxing process, the system installs each app in its own sandbox directory, which acts as the home for the app and its data. To help apps organize their data, each sandbox directory contains several well-known subdirectories for placing files. Above Figure shows the basic layout of a sandbox directory.

* Is the delegate for a CAAnimation retained?

Yes it is!! This is one of the rare exceptions to memory management rules.

* What is ‘dynamic’?

You use the @dynamic keyword to tell the compiler that you will fulfill the API contract implied by a property either by providing method implementations directly or at runtime using other mechanisms such as dynamic loading of code or dynamic method resolution. It suppresses the warnings that the compiler would otherwise generate if it can’t find suitable implementations. You should use it only if you know that the methods will be available at runtime.

* Explain the difference between NSOperationQueue concurrent and non-concurrent?

In the context of an NSOperation object, which runs in an NSOperationQueue, the terms concurrent and non-concurrent do not necessarily refer to the side-by-side execution of threads. Instead, a non-concurrent operation is one that executes using the environment that is provided for it while a concurrent operation is responsible for setting up its own execution environment.

* What is Automatic Reference Counting (ARC)?

ARC is a compiler-level feature that simplifies the process of managing the lifetimes of Objective-C objects. Instead of you having to remember when to retain or release an object, ARC evaluates the lifetime requirements of your objects and automatically inserts the appropriate method calls at compile time.

* What is the difference between retain & assign?

**Assign** creates a reference from one object to another without increasing the source’s retain count.

if (\_variable != object) {

[\_variable release];

\_ variable = nil;

\_ variable = object;

}

**Retain** creates a reference from one object to another and increases the retain count of the source object

if (\_variable != object) {

  [\_variable release];

    \_variable = nil;

\_variable = [object retain];

}

* What is ‘categories’ in iOS?

We use categories to define additional methods of an existing class—even one whose source code is unavailable to you—without subclassing. You typically use a category to add methods to an existing class, such as one defined in the Cocoa frameworks. The added methods are inherited by subclasses and are indistinguishable at runtime from the original methods of the class. You can also use categories of your own classes to:

* Distribute the implementation of your own classes into separate source files—for example, you could group the methods of a large class into several categories and put each category in a different file.
* Declare private methods.

You add methods to a class by declaring them in an interface file under a category name and defining them in an implementation file under the same name. The category name indicates that the methods are an extension to a class declared elsewhere, not a new class.

* a

.

### \*Q: What is notification in iOS?

A:

The notification mechanism of Cocoa implements one-to-many broadcast of messages based on the Observer pattern. Objects in a program add themselves or other objects to a list of observers of one or more notifications, each of which is identified by a global string (the notification name). The object that wants to notify other objects—the observed object—creates a notification object and posts it to a notification center. The notification center determines the observers of a particular notification and sends the notification to them via a message. The methods invoked by the notification message must conform to a certain single-parameter signature. The parameter of the method is the notification object, which contains the notification name, the observed object, and a dictionary containing any supplemental information.Posting a notification is a synchronous procedure. The posting object doesn’t regain control until the notification center has broadcast the notification to all observers. For asynchronous behavior, you can put the notification in a notification queue; control returns immediately to the posting object and the notification center broadcasts the notification when it reaches the top of the queue.Regular notifications that is, those broadcast by the notification center—are intraprocess only. If you want to broadcast notifications to other processes, you can use the istributed notification center and its related API.

### \*Q:What is the difference between delegates and notifications?

A:

We can use notifications for a variety of reasons. For example, you could broadcast a notification to change how user-interface elements display information based on a certain event elsewhere in the program. Or you could use notifications as a way to ensure that objects in a document save their state before the document window is closed. The general purpose of notifications is to inform other objects of program events so they can respond appropriately.But objects receiving notifications can react only after the event has occurred. This is a significant difference from delegation. The delegate is given a chance to reject or modify the operation proposed by the delegating object. Observing objects, on the other hand, cannot directly affect an impending operation.

### \*Q:What is posing in iOS?

A:

Objective-C permits a class to **entirely replace another class** within an application. The replacing class is said to “pose as” the target class. All messages sent to the target class are then instead received by the posing class. There are some restrictions on which classes can pose:

* A class may only pose as one of its direct or indirect superclasses
* The posing class must not define any new instance variables which are absent from the target class (though it may define or override methods).
* No messages must have been sent to the target class prior to the posing.

Posing, similarly to categories, allows **globally augmenting existing classes**. Posing permits two features absent from categories:

* A posing class can call overridden methods through super, thus incorporating the implementation of the target class.
* A posing class can override methods defined in categories.

### \*Q:What is atomic and nonatomic? Which one is safer? Which one is default?

A:

You can use this attribute to specify that accessor methods are not atomic. (There is no keyword to denote atomic.)

**nonatomic**

Specifies that accessors are nonatomic. *By default, accessors are atomic.*

Properties are atomic by default so that synthesized accessors provide robust access to properties in a multithreaded environment—that is, the value returned from the getter or set via the setter is always fully retrieved or set regardless of what other threads are executing concurrently.

If you specify strong, copy, or retain and do not specify nontoxic, then in a reference-counted environment, a synthesized get accessor for an object property uses a lock and retains and autoreleases the returned value—the implementation will be similar to the following:

|  |
| --- |
| *[\_internal lock]; // lock using an object-level lock* |
| *id result = [[value retain] autorelease];* |
| *[\_internal unlock];* |
| *return result;* |

If you specify nonatomic, a synthesized accessor for an object property simply returns the value directly.

#### Markup and Deprecation

Properties support the full range of C-style decorators. Properties can be deprecated and support \_\_attribute\_\_ style markup:

|  |
| --- |
| *@property CGFloat x* |
| *AVAILABLE\_MAC\_OS\_X\_VERSION\_10\_1\_AND\_LATER\_BUT\_DEPRECATED\_IN\_MAC\_OS\_X\_VERSION\_10\_4;* |
| *@property CGFloat y \_\_attribute\_\_((…));* |

### \*Q: What is run loop in iOS ?

A:

Run loops are part of the fundamental infrastructure associated with threads. A **run loop** is an event processing loop that you use to schedule work and coordinate the receipt of incoming events. The purpose of a run loop is to keep your thread busy when there is work to do and put your thread to sleep when there is none.

Run loop management is not entirely automatic. You must still design your thread’s code to start the run loop at appropriate times and respond to incoming events. Both Cocoa and Core Foundation provide **run loop objects** to help you configure and manage your thread’s run loop. Your application does not need to create these objects explicitly; each thread, including the application’s main thread, has an associated run loop object. Only secondary threads need to run their run loop explicitly, however. In both Carbon and Cocoa applications, the main thread automatically sets up and runs its run loop as part of the general application startup process.

### \*Q: What isDynamic typing?

A:

A variable is dynamically typed when the type of the object it points to is not checked at compile time. Objective-C uses the id data type to represent a variable that is an object without specifying what sort of object it is. This is referred to as *dynamic typing*.

Dynamic typing contrasts with static typing, in which the system explicitly identifies the class to which an object belongs at compile time. Static type checking at compile time may ensure stricter data integrity, but in exchange for that integrity, dynamic typing gives your program much greater flexibility. And through object introspection (for example, asking a dynamically typed, anonymous object what its class is), you can still verify the type of an object at runtime and thus validate its suitability for a particular operation.

### \*Q: What is the configuration file name in iOS explain in brief ? (Or) What is plist file and explain about it is usage?

A:

A property list is a representation of a hierarchy of objects that can be stored in the file system and reconstituted later. Property lists give applications a lightweight and portable way to store small amounts of data. They are hierarchies of data made from specific types of objects—they are, in effect, an object graph. Property lists are easy to create programmatically and are even easier to serialize into a representation that is persistent. Applications can later read the static representation back into memory and recreate the original hierarchy of objects. Both Cocoa Foundation and Core Foundation have APIs related to property list serialization and deserialization.

#### Property List Types and Objects

Property lists consist only of certain types of data: dictionaries, arrays, strings, numbers (integer and float), dates, binary data, and Boolean values. Dictionaries and arrays are special types because they are collections; they can contain one or multiple data types, including other dictionaries and arrays. This hierarchical nesting of objects creates a graph of objects. The abstract data types have corresponding Foundation classes, Core Foundation types, and XML elements for collection objects and value objects.

### \*Q:When will be the autorelease object released?

A:

Once the pool recives drain message.

### \*Q:Consider we are implementing our own thread with lot of autoreleased object. Is it mandatory to use autorelease pool on this scenario if yes/no why?

A:

YES.

### \*Q:  Have you ever used automated unit test framework in iOS? Explain in short?

A:

### \*Q: What are all the difference between iOS3, iOS4 and iOS5?

A:

Refer:  Apple iOS Versions and diffrences

### \*Q: Is there any garbage collector concept available in iOS?

A:

No, Manual memory management or ARC.

### \*Q: What is difference between synchronous and asynchronous in web request?

A:

### \*Q:What are all the instruments available in Xcode?

A:

### \*Q:What is the difference between copy & retain? When can we go for copy and when can we go for retain?

A:

### \*Q:How would you create your own custom view?

**A:**By Subclassing the UIView class.

### \*Q:What is App Bundle?

A:When you build your iOS app, Xcode packages it as a bundle. A **bundle** is a directory in the file system that groups related resources together in one place. An iOS app bundle contains the app executable file and supporting resource files such as app icons, image files, and localized content.

### \*Q:Whats fast enumeration?

**A:**Fast enumeration is a language feature that allows you to enumerate over the contents of a collection. (Your code will also run faster because the internal implementation reduces message send overhead and increases pipelining potential.)

### \*Q:Whats a struct?

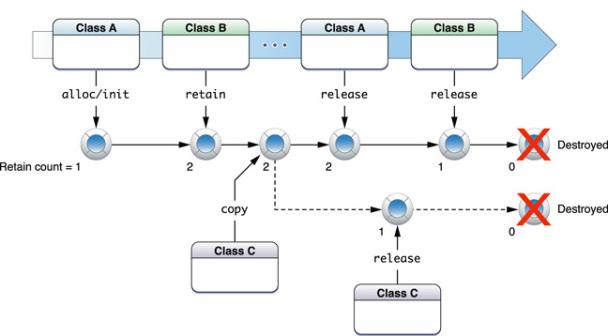
A:A struct is a special C data type that encapsulates other pieces of data into a single cohesive unit. Like an object, but built into C.

### \*Q:Whats the difference between  NSArray and  NSMutableArray?

A:NSArrayʼs contents can not be modified once itʼs been created whereas a NSMutableArray can be modified as needed, i.e items can be added/removed from it.

### \*Q:Explain retain counts.

A:Retain counts are the way in which memory is managed in Objective-C. When you create an object, it has a retain count of 1. When you send an object a retain message, its retain count is incremented by 1. When you send an object a release message, its retain count is decremented by 1. When you send an object a autorelease message, its retain count is decremented by 1 at some stage in the future. If an objectʼs retain count is reduced to 0, it is deallocated.



This will explain how the memory management is done in iOS

### \*Q:Whats the difference between frame and bounds?

A:The frame of a view is the rectangle, expressed as a location (x,y) and size (width,height) relative to the superview it is contained within. The bounds of a view is the rectangle, expressed as a location (x,y) and size (width,height) relative to its own coordinate system (0,0).

### \*Q:Is a delegate retained?

A:No, the delegate is never retained! Ever!

### \*Q:Outline the class hierarchy for a UIButton until NSObject.

A:UIButton inherits from UIControl, UIControl inherits from UIView, UIView inherits from UIResponder, UIResponder inherits from the root class NSObject.

**\*Q:**

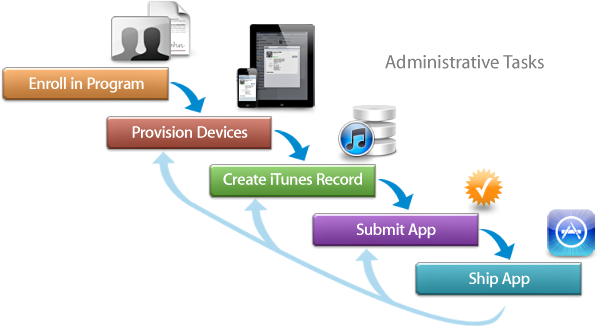
### What are the App states. Explain them?

**A:**

* **Not running State:**  The app has not been launched or was running but was terminated by the system.
* **Inactive state:** The app is running in the foreground but is currently not receiving events. (It may be executing other code though.) An app usually stays in this state only briefly as it transitions to a different state. The only time it stays inactive for any period of time is when the user locks the screen or the system prompts the user to respond to some event, such as an incoming phone call or SMS message.
* **Active state:** The app is running in the foreground and is receiving events. This is the normal mode for foreground apps.
* **Background state:**  The app is in the background and executing code. Most apps enter this state briefly on their way to being suspended. However, an app that requests extra execution time may remain in this state for a period of time. In addition, an app being launched directly into the background enters this state instead of the inactive state. For information about how to execute code while in the background, see “Background Execution and Multitasking.”
* **Suspended state**:The app is in the background but is not executing code. The system moves apps to this state automatically and does not notify them before doing so. While suspended, an app remains in memory but does not execute any code. When a low-memory condition occurs, the system may purge suspended apps without notice to make more space for the foreground app.

### \*Q:Explain the steps involved in submitting the App to App-Store.

A:



Apple provides the tools you need to develop, test, and submit your iOS app to the App Store. To run an app on a device, the device needs to be provisioned for development, and later provisioned for testing. You also need to provide information about your app that the App Store displays to customers and upload screenshots. Then you submit the app to Apple for approval. After the app is approved, you set a date the app should appear in the App Store as well as its price. Finally, you use Apple’s tools to monitor the sales of the app, customer reviews, and crash reports. Then you repeat the entire process again to submit updates to your app.

Ref: [App Store Review Guidelines](http://way2ios.com/development/ios-development-2/app-store-review-guidelines/)  
 

### \*Q:Why do we need to use @Synthesize?

A:

We can use generated code like nonatomic, atmoic, retain without writing any lines of code. We also have getter and setter methods. To use this, you have 2 other ways: @synthesize or @dynamic: @synthesize, compiler will generate the getter and setter automatically for you, @dynamic: you have to write them yourself.@property is really good for memory management, for example: retain.How can you do retain without @property?

if (\_variable != object)

{

    [\_variable release];

    \_variable = nil;

    \_variable = [object retain];

    }

How can you use it with @property?self.variable = object; When we are calling the above line, we actually call the setter like [self setVariable:object] and then the generated setter will do its job.

### \*Q:Multitasking support is available from which version?

A:

iOS 4.0.

#### \*Q:How many bytes we can send to apple push notification server?

A:

256bytes.

### \*Q:Can you just explain about memory management in iOS?

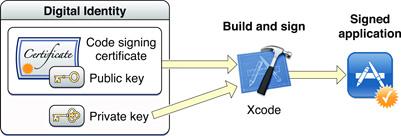
A:

Refer:[iOS Memory Management](http://faq)

### \*Q: What is code signing?

A:

Signing an application allows the system to identify who signed the application and to verify that the application has not been modified since it was signed. Signing is a requirement for submitting to the App Store (both for iOS and Mac apps). OS X and iOS verify the signature of applications downloaded from the App Store to ensure that they they do not run applications with invalid signatures. This lets users trust that the application was signed by an Apple source and hasn’t been modified since it was signed.



Xcode uses your digital identity to sign your application during the build process. This digital identity consists of a public-private key pair and a certificate. The private key is used by cryptographic functions to generate the signature. The certificate is issued by Apple; it contains the public key and identifies you as the owner of the key pair.

In order to sign applications, you must have both parts of your digital identity installed. Use Xcode or Keychain Access to manage your digital identities. Depending on your role in your development team, you may have multiple digital identities for use in different contexts. For example, the identity you use for signing during development is different from the identity you user for distribution on the App Store. Different digital identities are also used for development on OS X and on iOS.

An application’s executable code is protected by its signature because the signature becomes invalid if any of the executable code in the application bundle changes. Resources such as images and nib files are not signed; a change to these files does not invalidate the signature.

An application’s signature can be removed, and the application can be re-signed using another digital identity. For example, Apple re-signs all applications sold on the App Store. Also, a fully-tested development build of your application can be re-signed for submission to the App Store. Thus the signature is best understood not as indelible proof of the application’s origins but as a verifiable mark placed by the signer

Q. Consider the following UITableViewCell constructor:

- (id)initWithStyle:(UITableViewCellStyle)style reuseIdentifier:(NSString \*)reuseIdentifier

What is the purpose of the reuseIdentifier? What is the advantage of setting it to a non-nil value?

Ans. The reuseIdentifier is used to group together similar rows in an UITableView; i.e., rows that differ only in their content, but otherwise have similar layouts.

A UITableView will normally allocate just enough UITableViewCell objects to display the content visible in the table. If reuseIdentifier is set to a non-nil value, then when the table view is scrolled, UITableView will first attempt to reuse an already allocated UITableViewCell with the same reuseIdentifier. If reuseIdentifier has not been set, the UITableView will be forced to allocate new UITableViewCell objects for each new item that scrolls into view, potentially leading to laggy animations.

Q. What are different ways that you can specify the layout of elements in a UIView?

Ans. Here are a few common ways to specify the layout of elements in a UIView:

* Using InterfaceBuilder, you can add a XIB file to your project, layout elements within it, and then load the XIB in your application code (either automatically, based on naming conventions, or manually). Also, using InterfaceBuilder you can create a storyboard for your application.
* You can your own code to use NSLayoutConstraints to have elements in a view arranged by Auto Layout.
* You can create CGRects describing the exact coordinates for each element and pass them to UIView’s - (id)initWithFrame:(CGRect)frame method.

Q. What is the difference between atomic and nonatomic properties? Which is the default for synthesized properties? When would you use one vs. the other?

Ans. Properties specified as atomic are guaranteed to always return a fully initialized object. This also happens to be the default state for synthesized properties so, while it’s a good practice to specify atomic to remove the potential for confusion, if you leave it off, your properties will still be atomic. This guarantee of atomic properties comes at a cost to performance, however. If you have a property for which you know that retrieving an uninitialized value is not a risk (e.g. if all access to the property is already synchronized via other means), then setting it to nonatomic can gain you a bit of performance.

**1-How would you create your own custom view?**

*By Subclassing the UIView class.*

**2-Whats fast enumeration**?

*Fast enumeration is a language feature that allows you to enumerate over the contents of a collection. (Your code will also run faster because the internal implementation reduces  
message send overhead and increases pipelining potential.)*

**3-Whats a struct?**

*A struct is a special C data type that encapsulates other pieces of data into a single cohesive unit. Like an object, but built into C.*

**4-What are mutable and immutable types in Objective C?**

*Mutable means you can change its contents later but when you mark any object immutable, it means once they are initialized, their values cannot be changed. For example, NSArray, NSString values cannot be changed after initialized.*

**5-Explain retain counts**.

*Retain counts are the way in which memory is managed in Objective-C. When you create an object, it has a retain count of 1. When you send an object a retain message, its retain count is incremented by 1. When you send an object a release message, its retain count is decremented by 1. When you send an object a autorelease message, its retain count is decremented by 1 at some stage in the future. If an objectʼs retain count is reduced to 0, it is deallocated.*

**6-Whats the difference between frame and bounds**?

*The frame of a view is the rectangle, expressed as a location (x,y) and size (width,height) relative to the superview it is contained within. The bounds of a view is the rectangle, expressed as a location (x,y) and size (width,height) relative to its own coordinate system (0,0).*

**7-Is a delegate retained**?

*No, the delegate is never retained! Ever!*

**8-Outline the class hierarchy for a UIButton until NSObject.**

*UIButton inherits from UIControl, UIControl inherits from UIView, UIView inherits from UIResponder, UIResponder inherits from the root class NSObject*

### **9- What is**dynamic**?**

*You use the*@dynamic*keyword to tell the compiler that you will fulfill the API contract implied by a property either by providing method implementations directly or at runtime using other mechanisms such as dynamic loading of code or dynamic method resolution. It suppresses the warnings that the compiler would otherwise generate if it can’t find suitable implementations. You should use it only if you know that the methods will be available at runtime*

**10-If I call performSelector:withObject:afterDelay: – is the object retained?**

*Yes, the object is retained. It creates a timer that calls a selector on the current threads run loop. It may not be 100% precise time-wise as it attempts to dequeue the message from  
the run loop and perform the selector.*

**11-Can you explain what happens when you call autorelease on an object?**

*When you send an object a autorelease message, its retain count is decremented by 1 at some stage in the future. The object is added to an autorelease pool on the current thread. The main thread loop creates an autorelease pool at the beginning of the function, and release it at the end. This establishes a pool for the lifetime of the task. However, this also means that any autoreleased objects created during the lifetime of the task are not disposed of until the task completes. This may lead to the taskʼs memory footprint increasing unnecessarily. You can also consider creating pools with a narrower scope or use NSOperationQueue with itʼs own autorelease pool. (Also important – You only release or autorelease objects you own.)*

**12-Whats the NSCoder class used for?**

*NSCoder is an abstractClass which represents a stream of data. They are used in Archiving and Unarchiving objects. NSCoder objects are usually used in a method that is being implemented so that the class conforms to the protocol. (which has something like encodeObject and decodeObject methods in them).*

**13-Whats an NSOperationQueue and how/would you use it?**

*The NSOperationQueue class regulates the execution of a set of NSOperation objects. An operation queue is generally used to perform some asynchronous operations on a background thread so as not to block the main thread.*

**14-Explain the correct way to manage Outlets memory**

*Create them as properties in the header that are retained. In the viewDidUnload set the outlets to nil(i.e self.outlet = nil). Finally in dealloc make sure to release the outlet.*

**15-Is the delegate for a CAAnimation retained?**

*Yes it is!! This is one of the rare exceptions to memory management rules.*

**16-What happens when the following code executes?**

**Ball \*ball = [[[[Ball alloc] init] autorelease] autorelease];**

*It will crash because itʼs added twice to the autorelease pool and when it it dequeued the autorelease pool calls release more than once.*

**17-Explain the difference between NSOperationQueue concurrent and non-concurrent.**

*In the context of an NSOperation object, which runs in an NSOperationQueue, the terms concurrent and non-concurrent do not necessarily refer to the side-by-side execution of threads. Instead, a non-concurrent operation is one that executes using the environment that is provided for it while a concurrent operation is responsible for setting up its own execution environment.*

**18-Implement your own synthesized methods for the property NSString \*title.**

*Well you would want to implement the getter and setter for the title object. Something like this: view source print?*

*- (NSString\*) title // Getter method*

*{*

*return title;*

*}*

*- (void) setTitle: (NSString\*) newTitle //Setter method*

*{*

*if (newTitle != title)*

*{*

*[title release];*

*title = [newTitle retain]; // Or copy, depending on your needs.*

*}*

*}*

**19-Implement the following methods: retain, release, autorelease.**

*-(id)retain*

*{*

*NSIncrementExtraRefCount(self);*

*return self;*

*}*

*-(void)release*

*{*

*if(NSDecrementExtraRefCountWasZero(self))*

*{*

*NSDeallocateObject(self);*

*}*

*}*

*-(id)autorelease*

*{ // Add the object to the autorelease pool*

*[NSAutoreleasePool addObject:self];*

*return self****20-What are the App states. Explain them?***

* ***Not running State****: The app has not been launched or was running but was terminated by the system.*
* ***Inactive state:****The app is running in the foreground but is currently not receiving events. (It may be executing other code though.) An app usually stays in this state only briefly as it transitions to a different state. The only time it stays inactive for any period of time is when the user locks the screen or the system prompts the user to respond to some event, such as an incoming phone call or SMS message.*
* ***Active state****: The app is running in the foreground and is receiving events. This is the normal mode for foreground apps.*
* ***Background state****: The app is in the background and executing code. Most apps enter this state briefly on their way to being suspended. However, an app that requests extra execution time may remain in this state for a period of time. In addition, an app being launched directly into the background enters this state instead of the inactive state. For information about how to execute code while in the background, see “Background Execution and Multitasking.”*
* ***Suspended state:****The app is in the background but is not executing code. The system moves apps to this state automatically and does not notify them before doing so. While suspended, an app remains in memory but does not execute any code. When a low-memory condition occurs, the system may purge suspended apps without notice to make more space for the foreground app.*

**21-What is Automatic Reference Counting (ARC) ?**

*ARC is a compiler-level feature that simplifies the process of managing the lifetimes of Objective-C objects. Instead of you having to remember when to retain or release an object, ARC evaluates the lifetime requirements of your objects and automatically inserts the appropriate method calls at compile time.*

**22-Multitasking support is available from which version?**

*iOS 4 and above supports multi-tasking and allows apps to remain in the background until they are launched again or until they are terminated.*

**23-How many bytes we can send to apple push notification server.**

*256bytes.*

**24-What is the difference between retain & assign?**

***Assign****creates a reference from one object to another without increasing the source’s retain count.*

if (\_variable != object)

{

[\_variable release];

\_variable = nil;

\_variable = object;

}

***Retain****creates a reference from one object to another and increases the retain count of the source object.*

if (\_variable != object)

{ [\_variable release];

\_variable = nil;

\_variable = [object retain];

}

**25-Why do we need to use @Synthesize?**

*We can use generated code like nonatomic, atmoic, retain without writing any lines of code. We also have getter and setter methods. To use this, you have 2 other ways: @synthesize or @dynamic: @synthesize, compiler will generate the getter and setter automatically for you, @dynamic: you have to write them yourself.@property is really good for memory management, for example: retain.How can you do retain without @property?*

if (\_variable != object)

{

[\_variable release];

\_variable = nil;

\_variable = [object retain];

}

*How can you use it with @property?*self.variable = object;*When we are calling the above line, we actually call the setter like [self setVariable:object] and then the generated setter will do its job*

**26-What is categories in iOS?**

*A Category is a feature of the Objective-C language that enables you to add methods (interface and implementation) to a class without having to make a subclass. There is no runtime difference—within the scope of your program—between the original methods of the class and the methods added by the category. The methods in the category become part of the class type and are inherited by all the class’s subclasses.As with delegation, categories are not a strict adaptation of the Decorator pattern, fulfilling the intent but taking a different path to implementing that intent. The behavior added by categories is a compile-time artifact, and is not something dynamically acquired. Moreover, categories do not encapsulate an instance of the class being extended.*

*Delegation is a mechanism by which a host object embeds a weak reference (weak in the sense that it’s a simple pointer reference, unretained) to another object—its delegate—and periodically sends messages to the delegate when it requires its input for a task. The host object is generally an “off-the-shelf” framework object (such as an*NSWindow*or*[NSXMLParser](https://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSXMLParser_Class/Reference/Reference.html#//apple_ref/occ/cl/NSXMLParser)*object) that is seeking to accomplish something, but can only do so in a generic fashion. The delegate, which is almost always an instance of a custom class, acts in coordination with the host object, supplying program-specific behavior at certain points in the task (see Figure 4-3). Thus delegation makes it possible to modify or extend the behavior of another object without the need for subclassing.Refer: delegate pattern*

**30-What are all the difference between categories and subclasses?Why should we go to subclasses?**

*Category is a feature of the Objective-C language that enables you to add methods (interface and implementation) to a class without having to make a subclass. There is no runtime difference—within the scope of your program—between the original methods of the class and the methods added by the category. The methods in the category become part of the class type and are inherited by all the class’s subclasses.As with delegation, categories are not a strict adaptation of the Decorator pattern, fulfilling the intent but taking a different path to implementing that intent. The behavior added by categories is a compile-time artifact, and is not something dynamically acquired. Moreover, categories do not encapsulate an instance of the class being extended.The Cocoa frameworks define numerous categories, most of them informal protocols . Often they use categories to group related methods. You may implement categories in your code to extend classes without subclassing or to group related methods. However, you should be aware of these caveats:*

* *You cannot add instance variables to the class.*
* *If you override existing methods of the class, your application may behave unpredictably.*

**31-What is notification in iOS?**

*The notification mechanism of Cocoa implements one-to-many broadcast of messages based on the Observer pattern. Objects in a program add themselves or other objects to a list of observers of one or more notifications, each of which is identified by a global string (the notification name). The object that wants to notify other objects—the observed object—creates a notification object and posts it to a notification center. The notification center determines the observers of a particular notification and sends the notification to them via a message. The methods invoked by the notification message must conform to a certain single-parameter signature. The parameter of the method is the notification object, which contains the notification name, the observed object, and a dictionary containing any supplemental information.Posting a notification is a synchronous procedure. The posting object doesn’t regain control until the notification center has broadcast the notification to all observers. For asynchronous behavior, you can put the notification in a notification queue; control returns immediately to the posting object and the notification center broadcasts the notification when it reaches the top of the queue.Regular notifications—that is, those broadcast by the notification center—are intraprocess only. If you want to broadcast notifications to other processes, you can use the istributed notification center and its related API.*

**32-What is the difference between delegates and notifications?**

*We can use notifications for a variety of reasons. For example, you could broadcast a notification to change how user-interface elements display information based on a certain event elsewhere in the program. Or you could use notifications as a way to ensure that objects in a document save their state before the document window is closed. The general purpose of notifications is to inform other objects of program events so they can respond appropriately.But objects receiving notifications can react only after the event has occurred. This is a significant difference from delegation. The delegate is given a chance to reject or modify the operation proposed by the delegating object. Observing objects, on the other hand, cannot directly affect an impending operation.*

**33-What is posing in iOS?**

*Objective-C permits a class to****entirely replace another class****within an application. The replacing class is said to “pose as” the target class. All messages sent to the target class are then instead received by the posing class. There are some restrictions on which classes can pose:*

* *A class may only pose as one of its direct or indirect superclasses*
* *The posing class must not define any new instance variables which are absent from the target class (though it may define or override methods).*
* *No messages must have been sent to the target class prior to the posing.*

*Posing, similarly to categories, allows****globally augmenting existing classes****. Posing permits two features absent from categories:*

* *A posing class can call overridden methods through super, thus incorporating the implementation of the target class.*
* *A posing class can override methods defined in categories.*

**34-What is atomic and nonatomic? Which one is safer? Which one is default?**

*You can use this attribute to specify that accessor methods are not atomic. (There is no keyword to denote atomic.)*

nonatomic

*Specifies that accessors are nonatomic. By default, accessors are atomic.*

*Properties are atomic by default so that synthesized accessors provide robust access to properties in a multithreaded environment—that is, the value returned from the getter or set via the setter is always fully retrieved or set regardless of what other threads are executing concurrently.*

*If you specify*strong*,*copy*, or*retain*and do not specify*nonatomic*, then in a reference-counted environment, a synthesized get accessor for an object property uses a lock and retains and autoreleases the returned value—the implementation will be similar to the following:*

|  |
| --- |
| [\_internal lock]; // lock using an object-level lock |
| id result = [[value retain] autorelease]; |
| [\_internal unlock]; |
| return result; |

*If you specify*nonatomic*, a synthesized accessor for an object property simply returns the value directly.*

**35-Where can you test Apple iPhone apps if you don’t have the device?**

*iOS Simulator can be used to test mobile applications. Xcode tool that comes along with iOS SDK includes Xcode IDE as well as the iOS Simulator. Xcode also includes all required tools and frameworks for building iOS apps. However, it is strongly recommended to test the app on the real device before publishing it.*

**36-Which JSON framework is supported by iOS?**

*SBJson framework is supported by iOS. It is a JSON parser and generator for Objective-C. SBJson provides flexible APIs and additional control that makes JSON handling easier.*

**37-What are the tools required to develop iOS applications?**

*iOS development requires Intel-based Macintosh computer and iOS SDK.*

**38- Name the framework that is used to construct application’s user interface for iOS.**

*A. The UIKit framework is used to develop application’s user interface for iOS. UIKit framework provides event handling, drawing model, windows, views, and controls specifically designed for a touch screen interface.*

**39-Name the application thread from where UIKit classes should be used?**

*UIKit classes should be used only from an application’s main thread. Note: The derived classes of UIResponder and the classes which manipulate application’s user interface should be used from application’s main thread.*

**40- Which API is used to write test scripts that help in exercising the application’s user interface elements?**

*UI Automation API is used to automate test procedures. Tests scripts are written in JavaScript to the UI Automation API. This in turn simulates user interaction with the application and returns log information to the host computer.*

**41-Why an app on iOS device behaves differently when running in foreground than in background?**

*An application behaves differently when running in foreground than in background because of the limitation of resources on iOS devices.*

**42- How can an operating system improve battery life while running an app?**

*An app is notified whenever the operating system moves the apps between foreground and background. The operating system improves battery life while it bounds what your app can do in the background. This also improves the user experience with foreground app.*

**43-Which framework delivers event to custom object when app is in foreground?**

*The UIKit infrastructure takes care of delivering events to custom objects. As an app developer, you have to override methods in the appropriate objects to process those events.*

**44-When an app is said to be in not running state?**

*An app is said to be in ‘not running’ state when:  
- it is not launched.  
- it gets terminated by the system during running.*

**45-Assume that your app is running in the foreground but is currently not receiving events. In which sate it would be in?**

*An app will be in InActive state if it is running in the foreground but is currently not receiving events. An app stays in InActive state only briefly as it transitions to a different state.*

**46- Give example scenarios when an application goes into InActive state?**

*An app can get into InActive state when the user locks the screen or the system prompts the user to respond to some event e.g. SMS message, incoming call etc.*

**47-When an app is said to be in active state?**

*An app is said to be in active state when it is running in foreground and is receiving events.*

**48-Name the app sate which it reaches briefly on its way to being suspended**

*An app enters background state briefly on its way to being suspended.*

**49- Assume that an app is not in foreground but is still executing code. In which state will it be in?**

*Background state.*

**50-An app is loaded into memory but is not executing any code. In which state will it be in?**

*An app is said to be in suspended state when it is still in memory but is not executing any code.*

**51-Assume that system is running low on memory. What can system do for suspended apps?**

*In case system is running low on memory, the system may purge suspended apps without notice.*

**52- How can you respond to state transitions on your app?**

*On state transitions can be responded to state changes in an appropriate way by calling corresponding methods on app’s delegate object.*

*For example: applicationDidBecomeActive method can be used to prepare to run as the foreground app.  
applicationDidEnterBackground method can be used to execute some code when app is running in the background and may be suspended at any time.  
applicationWillEnterForeground method can be used to execute some code when your app is moving out of the background  
applicationWillTerminate method is called when your app is being terminated.*

**53-List down app’s state transitions when it gets launched.**

*Before the launch of an app, it is said to be in not running state.  
When an app is launched, it moves to the active or background state, after transitioning briefly through the inactive state.*

**54-Who calls the main function of you app during the app launch cycle?**

*During app launching, the system creates a main thread for the app and calls the app’s main function on that main thread. The Xcode project’s default main function hands over control to the UIKit framework, which takes care of initializing the app before it is run.*

**55-What is the use of controller object UIApplication?**

*Controller object UIApplication is used without subclassing to manage the application event loop.  
It coordinates other high-level app behaviors.  
It works along with the app delegate object which contains app-level logic.*

**56-Which object is create by UIApplicationMain function at app launch time?**

*The app delegate object is created by UIApplicationMain function at app launch time. The app delegate object’s main job is to handle state transitions within the app.*

**57- How is the app delegate is declared by Xcode project templates?**

*App delegate is declared as a subclass of UIResponder by Xcode project templates.*

**58-What happens if IApplication object does not handle an event?**

*In such case the event will be dispatched to your app delegate for processing.*

**59- Which app specific objects store the app’s content?**

*Data model objects are app specific objects and store app’s content. Apps can also use document objects to manage some or all of their data model objects.*

**60-Are document objects required for an application? What does they offer?**

*Document objects are not required but are very useful in grouping data that belongs in a single file or file package.*

**61- Which object manage the presentation of app’s content on the screen?**

*View controller objects takes care of the presentation of app’s content on the screen. A view controller is used to manage a single view along with the collection of subviews. It makes its views visible by installing them in the app’s window.*

**62- Which is the super class of all view controller objects?**

*UIViewController class. The functionality for loading views, presenting them, rotating them in response to device rotations, and several other standard system behaviors are provided by UIViewController class.*

**63-What is the purpose of UIWindow object?**

*The presentation of one or more views on a screen is coordinated by UIWindow object.*

**64-How do you change the content of your app in order to change the views displayed in the corresponding window?**

*To change the content of your app, you use a view controller to change the views displayed in the corresponding window. Remember, window itself is never replaced.*

**65-Define view object.**

*Views along with controls are used to provide visual representation of the app content. View is an object that draws content in a designated rectangular area and it responds to events within that area.*

**66-Apart from incorporating views and controls, what else an app can incorporate?**

*Apart from incorporating views and controls, an app can also incorporate Core Animation layers into its view and control hierarchies.*

**67- What are layer objects and what do they represent?**

*Layer objects are data objects which represent visual content. Layer objects are used by views to render their content. Custom layer objects can also be added to the interface to implement complex animations and other types of sophisticated visual effects.*

### **68-What is App Bundle?**

*When you build your iOS app, Xcode packages it as a bundle. A****bundle****is a directory in the file system that groups related resources together in one place. An iOS app bundle contains the app executable file and supporting resource files such as app icons, image files, and localized content.*

**69-Define property?**

*It is used to access instance variables outside of class.*

**70-Why synthesized is used?**

*After declaring property we will have to tell compiler instantly by using synthesize directive. This tells the compiler to generate setter and getter methods.*

**71-What is retaining?**

*It is reference count for an object.*

**72- What is webservice?**

*To get data in form of xml ,by using this we can get data from a server.*

**73-What is parsing?**

*To get data from web service we use parsing.*

**74-which xml parser we use on iphone?**

*“NSXML” Parser.*

**75-Which type of parse does iphone support?**

*“SAX” parser.*

**76-.Name those classes used to establish connection b/w application to webserver?**

*(a)NSURL (b)NSURL REQUEST (c)NSURL CONNECTION.*

**77-Tell the difference between DOM and SAX Parser?**

*(a)Dom is “documents based parser”.  
b)SAX is a event driven parser*

**78-Name three method of NSXML parser.**

*(1)did start element (2)did end element (3)found character.*

**79-Tell methods used in NSURLConnection**

*(1)Connection did receive Response  
(2)Connection did recevice Datat  
(3)Connection fail with error  
(4)Connection did finish loading.*

**80-.What is json-parser?**

*JSON(Java script object notation)is a parser used to get data from web Server.*

**81-.By default which things are in the application?**

*iPhone applications by default have 3 things  
1.main: entry point of application.  
2.Appdelegate: perform basic application and functionality.  
3.Window: provide uiinterface.*

**82-Tell me about tab bar controller?**

*It is used to display the data on the view.*

**83-Which are the protocols used in table view?**

*Table view contain two delegate protocols  
(1) Uitable view data source  
(2).Uitable view delegate.  
ui view table view data source three method namely  
(1)No of sections.  
(2)No of rows in sections.  
(3)Cell for row index path row.  
In ui table view delegate contain  
(1)Did select row at index path row*

**84-Name data base used in iphone?**

*(1)Sql lite (2)Plist 3)Xml (4)Core Data*

**85-Tell four frameworks used in iphone?**

*(1)Ui kit framework  
(2)Map kit framework  
(3)ADI kit framework  
(4)Core data framework  
(5)core foundation framework*

**86-Tell me about single inheritance in objective-c?**

*Objective c subclass can derived from a single parent class.It is called “single inheritance”.*

**87-Tell me about the MVC architecture?**

*M-model, V-view, C-controller*

*Main advantage of MVC architecture is to provide “reusability and security” by separating the layer by using MVC architecture.*

***Model:****it is a class model is interact with database.*

***Controller:****controller is used for by getting the data from model and controls the views.*

*Display the information in views. : View*

**88-What is the instance methods?**

*Instance methods are essentially code routines that perform tasks so instances of clases we create methods to get and set the instance variables and to display the current values of these variables.*

*Declaration of instance method :*

*– (void)click me: (id)sender;*

*Void is return type which does not giving any thing here.*

*Click me is method name.*

*Id is data type which returns any type of object.*

**89-What is the class method?**

*Class methods work at the class level and are common to all instance of a class these methods are specific to the class overall as opposed to working on different instance data encapsulated in each class instance.*

*@interface class name :ns object*

*{*

*}*

*+(class name \*)new alloc:*

*-(int)total open*

**90-What is data encapsulation?**

*Data is contained within objects and is not accessible by any other than via methods defined on the class is called data encapsulation.*

**91-What is accessor methods?**

*Accessor methods are methods belonging to a class that allow to get and set the values of instance valuables contained within the class.*

**92-What is synthesized accessor methods?**

*Objective-c provides a mechanism that automates the creation of accessor methods that are called synthesized accessor methods that are implemented through use of the @property and @synthesized.*

**93-How to access the encapsulated data in objective-c?**

*(a)Data encapsulation encourages the use of methods to +get and set the values of instance variables in a class.*

*(b)But the developer to want to directly access an instance variable without having to go through an accessor method.*

*(c) In objective-c syntax for an instance variable is as follow [class instance variable name]*

**94-What is dot notation?**

*Dot notation features introduced into version 2.0 of objective-c. Dot notation involves accessing an instance variable by specifying a class “instance” followed by a “dot” followed in turn by the name of instance variable or property to be accessed.*

**95-Difference between shallow copy and deep copy?**

*Shallow copy is also known as address copy. In this process you only copy address not actual data while in deep copy you copy data.*

*Suppose there are two objects A and B. A is pointing to a different array while B is pointing to different array. Now what I will do is following to do shallow copy. Char \*A = {‘a’,’b’,’c’}; Char \*B = {‘x’,’y’,’z’}; B = A; Now B is pointing is at same location where A pointer is pointing.Both A and B in this case sharing same data. if change is made both will get altered value of data.Advantage is that coping process is very fast and is independent of size of array.*

*while in deep copy data is also copied. This process is slow but Both A and B have their own copies and changes made to any copy, other will copy will not be affected.*

**96-Difference between categories and extensions?**

*Class extensions are similar to categories. The main difference is that with an extension, the compiler will expect you to implement the methods within your main @implementation, whereas with a category you have a separate @implementation block. So you should pretty much only use an extension at the top of your main .m file (the only place you should care about ivars, incidentally) — it’s meant to be just that, an extension.*

**97-What are KVO and KVC?**

***KVC:****Normally instance variables are accessed through properties or accessors but KVC gives another way to access variables in form of strings. In this way your class acts like a dictionary and your property name for example “age” becomes key and value that property holds becomes value for that key. For example, you have employee class with name property.*

*You access property like*

*NSString age = emp.age;*

*setting property value.*

*emp.age = @”20″;*

*Now how KVC works is like this*

*[emp valueForKey:@"age"];*

*[emp setValue:@"25" forKey:@"age"];*

***KVO :****The mechanism through which objects are notified when there is change in any of property is called KVO.*

*For example, person object is interested in getting notification when accountBalance property is changed in BankAccount object.To achieve this, Person Object must register as an observer of the BankAccount’s accountBalance property by sending an addObserver:forKeyPath:options:context: message.*

**98-Can we use two tableview controllers on one view controller?**

*Yes, we can use two tableviews on the same view controllers and you can differentiate between two by assigning them tags…or you can also check them by comparing their memory addresses.*

**99-Swap the two variable values without taking third variable?**

*int x=10; int y=5; x=x+y; NSLog(@”x==> %d”,x);*

*y=x-y; NSLog(@”Y Value==> %d”,y);*

*x=x-y; NSLog(@”x Value==> %d”,x);*

**100-What is push notification?**

*Imagine, you are looking for a job. You go to software company daily and ask sir “is there any job for me” and they keep on saying no. Your time and money is wasted on each trip.(Pull Request mechanism)*

*So, one day owner says, if there is any suitable job for you, I will let you know. In this mechanism, your time and money is not wasted. (Push Mechanism)*

**How it works?**

*This service is provided by Apple in which rather than pinging server after specific interval for data which is also called pull mechanism, server will send notification to your device that there is new piece of information for you. Request is initiated by server not the device or client.*

**Flow of push notification**

*Your web server sends message (device token + payload) to Apple push notification service (APNS) , then APNS routes this message to device whose device token specified in notification.*

**101-What is polymorphism?**

*This is very famous question and every interviewer asks this. Few people say polymorphism means multiple forms and they start giving example of draw function which is right to some extent but interviewer is looking for more detailed answer.*

*Ability of base class pointer to call function from derived class at runtime is called polymorphism.*

*For example, there is super class human and there are two subclasses software engineer and hardware engineer. Now super class human can hold reference to any of subclass because software engineer is kind of human. Suppose there is speak function in super class and every subclass has also speak function. So at runtime, super class reference is pointing to whatever subclass, speak function will be called of that class. I hope I am able to make you understand.*

**101-What is responder chain?**

*Suppose you have a hierarchy of views such like there is superview A which have subview B and B has a subview C. Now you touch on inner most view C. The system will send touch event to subview C for handling this event. If C View does not want to handle this event, this event will be passed to its superview B (next responder). If B also does not want to handle this touch event it will pass on to superview A. All the view which can respond to touch events are called responder chain. A view can also pass its events to uiviewcontroller. If view controller also does not want to respond to touch event, it is passed to application object which discards this event.*

**102-Can we use one tableview with two different datasources? How you will achieve this?**

*Yes. We can conditionally bind tableviews with two different data sources.*

**103-What is a protocol?**

*A protocol is a language feature in objective C which provides multiple inheritance in a single inheritance language. Objective C supports two types of protocols:*

* *Ad hoc protocols called informal protocol*
* *Compiler protocols called formal protocols*

*You must create your own autorelease pool as soon as the thread begins executing; otherwise, your application will leak objects*

**104-Three occasions when you might use your own autorelease pools:**

* *If you are writing a program that is not based on a UI framework, such as a command-line tool.*
* *If you write a loop that creates many temporary objects.You may create an autorelease pool inside the loop to dispose of those objects before the next iteration. Using an autorelease pool in the loop helps to reduce the maximum memory footprint of the application.*
* *If you spawn a secondary thread.*

**105- InApp purchase product type**

* ***Consumable****products must be purchased each time the user needs that item. For example, one-time services are commonly implemented as consumable products.*
* ***Non-consumable****products are purchased only once by a particular user. Once a non-consumable product is purchased, it is provided to all devices associated with that user’s iTunes account. Store Kit provides built-in support to restore non-consumable products on multiple devices.*
* ***Auto-renewable subscriptions****are delivered to all of a user’s devices in the same way as non-consumable products. However, auto-renewable subscriptions differ in other ways. When you create an auto-renewable subscription in iTunes Connect, you choose the duration of the subscription. The App Store automatically renews the subscription each time its term expires. If the user chooses to not allow the subscription to be renewed, the user’s access to the subscription is revoked after the subscription expires. Your application is responsible for validating whether a subscription is currently active and can also receive an updated receipt for the most recent transaction.*
* ***Free subscriptions****are a way for you to put free subscription content in Newsstand. Once a user signs up for a free subscription, the content is available on all devices associated with the user’s Apple ID. Free subscriptions do not expire and can only be offered in Newsstand-enabled apps*

**106-the advantages and disadvantages about synchronous versus asynchronous connections.**

*That’s it, pretty fast and easy, but there are a lot of caveats :*

*• The most important problem is that the thread which called this method will be blocked until the connection finish or timeout, so we surely don’t want to start the connection on the main thread to avoid freezing the UI. That means we need to create a new thread to handle the connection, and all programmers know that threading is hard.*

*• Cancellation, it’s not possible to cancel a synchronous connection, which is bad because users like to have the choice to cancel an operation if they think it takes too much time to execute.*

*• Authentication, there is no way to deal with authentication challenges.*

*• It’s impossible to parse data on the fly.*

*So let’s put it up straight, avoid using synchronousNSURLConnection, there is absolutely no benefit of using it.*

*It’s clear that****asynchronous connections****give us more control :*

*• You don’t have to create a new thread for the connection because your main thread will not be blocked.*

*• You can easily cancel the connection just by calling the cancelmethod.*

*• If you need authentication just implement the required delegate methods.*

*• Parsing data on the fly is easy.*

*So clearly we have a lot of more control with this, and the code is really not difficult.*

*Even better, we don’t have to handle the creation of a new thread, which is a good thing, because you know, threading is hard.*

*Well, if you read me until here, you should be convinced to use asynchronous connections, and forget about synchronous ones. They clearly give us more control and possibilities and, in some case can spare us to create new thread.*

*So I encourage you to move away from synchronous connections, just think of them as evil.*

**107-What is the navigation controller?**

*Navigation controller contains the stack of controllers every navigation controller  
must be having root view controller by default these controllers contain 2 method  
(a) push view (b) pop view  
By default navigation controller contain “table view”.*

**108- What is the split view controller?**

*This control is used for ipad application and it contain proper controllers by default  
split view controller contain root view controller and detail view controller.*

**109-Cocoa.**

*Cocoa is an application environment for both the Mac OS X operating system and iOS. It consists of a suite of object-oriented software libraries, a runtime system, and an integrated development environment. Carbon is an alternative environment in Mac OS X, but it is a compatibility framework with procedural programmatic interfaces intended to support existing Mac OS X code bases.*

**110- Frameworks that make Cocoa.**

*Appkit (Application Kit)*

*Foundation*

**111- Objective-C.**

*Objective-C is a very dynamic language. Its dynamism frees a program from compile-time and link-time constraints and shifts much of the responsibility for symbol resolution to runtime, when the user is in control. Objective-C is more dynamic than other programming languages because its dynamism springs from three sources:*

*Dynamic typing—determining the class of an object at runtime*

*Dynamic binding—determining the method to invoke at runtime*

*Dynamic loading—adding new modules to a program at runtime*

**112- Objective-C vs C/C++.**

*· The Objective-C class allows a method and a variable with the exact same name. In C++, they must be different.*

*· Objective-C does not have a constructor or destructor. Instead it has init and dealloc methods, which must be called explicitly.*

*· Objective-C uses + and – to differentiate between factory and instance methods, C++ uses static to specify a factory method.*

*· Multiple inheritance is not allowed in Obj-C, however we can use protocol to some extent.*

*· Obj-C has runtime binding leading to dynamic linking.*

*· Obj-C has got categories.*

*· Objective-C has a work-around for method overloading, but none for operator overloading.*

*· Objective-C also does not allow stack based objects. Each object must be a pointer to a block of memory.*

*· In Objective-C the message overloading is faked by naming the parameters. C++ actually does the same thing but the compiler does the name mangling for us. In Objective-C, we have to mangle the names manually.*

*· One of C++’s advantages and disadvantages is automatic type coercion.*

*· Another feature C++ has that is missing in Objective-C is references. Because pointers can be used wherever a reference is used, there isn’t much need for references in general.*

*· Templates are another feature that C++ has that Objective-C doesn’t. Templates are needed because C++ has strong typing and static binding that prevent generic classes, such as List and Array.*

**113-Appilcation Kit/App kit.**

*The Application Kit is a framework containing all the objects you need to implement your graphical, event-driven user interface: windows, panels, buttons, menus, scrollers, and text fields. The Application Kit handles all the details for you as it efficiently draws on the screen, communicates with hardware devices and screen buffers, clears areas of the screen before drawing, and clips views.*

*You also have the choice at which level you use the Application Kit:*

*· Use Interface Builder to create connections from user interface objects to your application objects.*

*· Control the user interface programmatically, which requires more familiarity with AppKit classes and protocols.*

*· Implement your own objects by subclassing NSView or other classes.*

**114-Foundation Kit.**

*The Foundation framework defines a base layer of Objective-C classes. In addition to providing a set of useful primitive object classes, it introduces several paradigms that define functionality not covered by the Objective-C language. The Foundation framework is designed with these goals in mind:*

*· Provide a small set of basic utility classes.*

*· Make software development easier by introducing consistent conventions for things such as deallocation.*

*· Support Unicode strings, object persistence, and object distribution.*

*· Provide a level of OS independence, to enhance portability.*

**115-Dynamic and Static Typing.**

*Static typed languages are those in which type checking is done at compile-time, whereas dynamic typed languages are those in which type checking is done at run-time.*

*Objective-C is a dynamically-typed language, meaning that you don’t have to tell the compiler what type of object you’re working with at compile time. Declaring a type for a varible is merely a promise which can be broken at runtime if the code leaves room for such a thing. You can declare your variables as type id, which is suitable for any Objective-C object.*

**116-Selectors**

*In Objective-C, selector has two meanings. It can be used to refer simply to the name of a method when it’s used in a source-code message to an object. It also, though, refers to the unique identifier that replaces the name when the source code is compiled. Compiled selectors are of type SEL. All methods with the same name have the same selector. You can use a selector to invoke a method on an object—this provides the basis for the implementation of the target-action design pattern in Cocoa.*

*[friend performSelector:@selector(gossipAbout:) withObject:aNeighbor];*

*is equivalent to:*

*[friend gossipAbout:aNeighbor];*

**117-Class Introspection**

*· Determine whether an objective-C object is an instance of a class*

*[obj isMemberOfClass:someClass];*

*· Determine whether an objective-C object is an instance of a class or its descendants*

*[obj isKindOfClass:someClass];*

*· The version of a class*

*[MyString version]*

*· Find the class of an Objective-C object*

*Class c = [obj1 class]; Class c = [NSString class];*

*· Verify 2 Objective-C objects are of the same class*

*[obj1 class] == [obj2 class]*

**118- Proxy**

*As long as there aren’t any extra instance variables, any subclass can proxy itself as its superclass with a single call. Each class that inherits from the superclass, no matter where it comes from, will now inherit from the proxied subclass. Calling a method in the superclass will actually call the method in the subclass. For libraries where many objects inherit from a base class, proxying the superclass can be all that is needed.*

**119- Why category is better than inheritance?**

*If category is used, you can use same class, no need to remember a new class-name. Category created on a base class is available on sub classes.*

**120-Formal Protocols**

*Formal Protocols allow us to define the interface for a set of methods, but implementation is not done. Formal Protocols are useful when you are using DistributedObjects, because they allow you to define a protocol for communication between objects, so that the DO system doesn’t have to constantly check whether or not a certain method is implemented by the distant object.*

**121- Formal vs informal protocol.**

*In addition to formal protocols, you can also define an informal protocol by grouping the methods in a category declaration:*

*@interface NSObject (MyProtocol)*

*//someMethod();*

*@end*

*Informal protocols are typically declared as categories of the NSObject class, because that broadly associates the method names with any class that inherits from NSObject. Because all classes inherit from the root class, the methods aren’t restricted to any part of the inheritance hierarchy. (It is also possible to declare an informal protocol as a category of another class to limit it to a certain branch of the inheritance hierarchy, but there is little reason to do so.)*

*When used to declare a protocol, a category interface doesn’t have a corresponding implementation. Instead, classes that implement the protocol declare the methods again in their own interface files and define them along with other methods in their implementation files.*

*An informal protocol bends the rules of category declarations to list a group of methods but not associate them with any particular class or implementation.*

*Being informal, protocols declared in categories don’t receive much language support. There’s no type checking at compile time nor a check at runtime to see whether an object conforms to the protocol. To get these benefits, you must use a formal protocol. An informal protocol may be useful when all the methods are optional, such as for a delegate, but (in Mac OS X v10.5 and later) it is typically better to use a formal protocol with optional methods.*

**122- Optional vs required**

*Protocol methods can be marked as optional using the @optional keyword. Corresponding to the @optional modal keyword, there is a @required keyword to formally denote the semantics of the default behavior. You can use @optional and @required to partition your protocol into sections as you see fit. If you do not specify any keyword, the default is @required.*

*@protocol MyProtocol*

*@optional*

*-(void) optionalMethod;*

*@required*

*-(void) requiredMethod;*

*@end*

**123- Memory Management**

*If you alloc, retain, or copy/mutablecopy it, it’s your job to release it. Otherwise it isn’t.*

**124-Copy vs assign vs retain**

*· Assign is for primitive values like BOOL, NSInteger or double. For objects use retain or copy, depending on if you want to keep a reference to the original object or make a copy of it.*

*·****assign****: In your setter method for the property, there is a simple assignment of your instance variable to the new value, eg:*

*(void)setString:(NSString\*)newString{*

*string = newString;*

*}*

*This can cause problems since Objective-C objects use reference counting, and therefore by not retaining the object, there is a chance that the string could be deallocated whilst you are still using it.*

*·****retain****: this retains the new value in your setter method. For example:*

*This is safer, since you explicitly state that you want to maintain a reference of the object, and you must release it before it will be deallocated.*

*(void)setString:(NSString\*)newString{*

*[newString retain];*

*[string release];*

*string = newString;*

*}*

*·****copy****: this makes a copy of the string in your setter method:*

*This is often used with strings, since making a copy of the original object ensures that it is not changed whilst you are using it.*

*(void)setString:(NSString\*)newString{*

*if(string!=newString){*

*[string release];*

*string = [newString copy];*

*}*

*}*

**125- alloc vs new**

*“alloc” creates a new memory location but doesn’t initializes it as compared to “new”.*

**126- release vs pool drain**

*“release” frees a memory. “drain” releases the NSAutoreleasePool itself.*

**127- NSAutoReleasePool : release vs drain**

*Strictly speaking, from the big picture perspective drain is not equivalent to release:*

*In a reference-counted environment, drain does perform the same operations as release, so the two are in that sense equivalent. To emphasise, this means you do not leak a pool if you use drain rather than release.*

*In a garbage-collected environment, release is a no-op. Thus it has no effect. drain, on the other hand, contains a hint to the collector that it should “collect if needed”. Thus in a garbage-collected environment, using drain helps the system balance collection sweeps.*

**128-autorelease vs release**

*Autorelase: By sending an object an autorelease message, it is added to the local AutoReleasePool, and you no longer have to worry about it, because when the AutoReleasePool is destroyed (as happens in the course of event processing by the system) the object will receive a release message, its RetainCount will be decremented, and the GarbageCollection system will destroy the object if the RetainCount is zero.*

*Release: retain count is decremented at this point.*

**129- Autorelease Pool**

*Autorelease pools provide a mechanism whereby you can send an object a “deferred” release message. This is useful in situations where you want to relinquish ownership of an object, but want to avoid the possibility of it being deallocated immediately (such as when you return an object from a method). Typically, you don’t need to create your own autorelease pools, but there are some situations in which either you must or it is beneficial to do so.*

**130- How autorelease pool is managed.**

*Every time -autorelease is sent to an object, it is added to the inner-most autorelease pool. When the pool is drained, it simply sends -release to all the objects in the pool.*

*Autorelease pools are simply a convenience that allows you to defer sending -release until “later”. That “later” can happen in several places, but the most common in Cocoa GUI apps is at the end of the current run loop cycle.*

**131-Memory Leak**

*If RetainingAndReleasing are not properly used then RetainCount for AnObject doesn’t reach 0. It doesn’t crash the application.*

**132- Event Loop**

*In a Cocoa application, user activities result in events. These might be mouse clicks or drags, typing on the keyboard, choosing a menu item, and so on. Other events can be generated automatically, for example a timer firing periodically, or something coming in over the network. For each event, Cocoa expects there to be an object or group of objects ready to handle that event appropriately. The event loop is where such events are detected and routed off to the appropriate place. Whenever Cocoa is not doing anything else, it is sitting in the event loop waiting for an event to arrive. (In fact, Cocoa doesn’t poll for events as suggested, but instead its main thread goes to sleep. When an event arrives, the OS wakes up the thread and event processing resumes. This is much more efficient than polling and allows other applications to run more smoothly).*

*Each event is handled as an individual thing, then the event loop gets the next event, and so on. If an event causes an update to be required, this is checked at the end of the event and if needed, and window refreshes are carried out.*

**133-Differnce between boxName and self.boxName.**

*boxName: Accessing directly.*

*self. boxName: Accessing boxName through accessors. If property/synthesize is not there it will throw error.*

**134-What it does “@synthesize boxDescription=boxName;” ?**

*Here you can use boxName or self.boxName. We cant use boxDescription.*

**135-Collection**

*In Cocoa and Cocoa Touch, a collection is a Foundation framework class used for storing and managing groups of objects. Its primary role is to store objects in the form of either an array, a dictionary, or a set.*

**136-Threads and how to use**

*Use this class when you want to have an Objective-C method run in its own thread of execution. Threads are especially useful when you need to perform a lengthy task, but don’t want it to block the execution of the rest of the application. In particular, you can use threads to avoid blocking the main thread of the application, which handles user interface and event-related actions. Threads can also be used to divide a large job into several smaller jobs, which can lead to performance increases on multi-core computers.*

*Two ways to create threads…*

*· detachNewThreadSelector:toTarget:withObject:*

*· Create instances of NSThread and start them at a later time using the “start” method.*

*NSThread is not as capable as Java’s Thread class, it lacks*

*· Built-in communication system.*

*· An equivalent of “join()”*

**137-Threadsafe**

*When it comes to threaded applications, nothing causes more fear or confusion than the issue of handling signals. Signals are a low-level BSD mechanism that can be used to deliver information to a process or manipulate it in some way. Some programs use signals to detect certain events, such as the death of a child process. The system uses signals to terminate runaway processes and communicate other types of information.*

*The problem with signals is not what they do, but their behavior when your application has multiple threads. In a single-threaded application, all signal handlers run on the main thread. In a multithreaded application, signals that are not tied to a specific hardware error (such as an illegal instruction) are delivered to whichever thread happens to be running at the time. If multiple threads are running simultaneously, the signal is delivered to whichever one the system happens to pick. In other words, signals can be delivered to any thread of your application.*

*The first rule for implementing signal handlers in applications is to avoid assumptions about which thread is handling the signal. If a specific thread wants to handle a given signal, you need to work out some way of notifying that thread when the signal arrives. You cannot just assume that installation of a signal handler from that thread will result in the signal being delivered to the same thread.*

**138-Notification and Observers**

*A notification is a message sent to one or more observing objects to inform them of an event in a program. The notification mechanism of Cocoa follows a****broadcast****model. It is a way for an object that initiates or handles a program event to communicate with any number of objects that want to know about that event. These recipients of the notification, known as observers, can adjust their own appearance, behavior, and state in response to the event. The object sending (or posting) the notification doesn’t have to know what those observers are. Notification is thus a powerful mechanism for attaining coordination and cohesion in a program. It reduces the need for strong dependencies between objects in a program (such dependencies would reduce the reusability of those objects). Many classes of the Foundation, AppKit, and other Objective-C frameworks define notifications that your program can register to observe.*

*The centerpiece of the notification mechanism is a per-process singleton object known as the notification center (****NSNotificationCenter****). When an object posts a notification, it goes to the notification center, which acts as a kind of clearing house and broadcast center for notifications. Objects that need to know about an event elsewhere in the application register with the notification center to let it know they want to be notified when that event happens. Although the notification center delivers a notification to its observers synchronously, you can post notifications asynchronously using a notification queue (NSNotificationQueue).*

**139-Delegate vs Notification**

*· The concept of notification differs from delegation in that it allows a message to be sent to more than one object. It is more like a broadcast rather than a straight communication between two objects. It removes dependencies between the sending and receiving object(s) by using a notification center to manage the sending and receiving of notifications. The sender does not need to know if there are any receivers registered with the notification center. There can be one, many or even no receivers of the notification registered with the notification center. Simply, Delegate is 1-to-1 object and Notification can be \*-to-\* objects.*

*· The other difference between notifications and delegates is that there is no possibility for the receiver of a notification to return a value to the sender.*

*· Typical uses of notifications might be to allow different objects with an application to be informed of an event such as a file download completing or a user changing an application preference. The receiver of the notification might then perform additional actions such as processing the downloaded file or updating the display.*

**140-Plist**

*Property lists organize data into named values and lists of values using several object types. These types give you the means to produce data that is meaningfully structured, transportable, storable, and accessible, but still as efficient as possible. Property lists are frequently used by applications running on both Mac OS X and iOS. The property-list programming interfaces for Cocoa and Core Foundation allow you to convert hierarchically structured combinations of these basic types of objects to and from standard XML. You can save the XML data to disk and later use it to reconstruct the original objects.*

*The user defaults system, which you programmatically access through the NSUserDefaults class, uses property lists to store objects representing user preferences. This limitation would seem to exclude many kinds of objects, such as NSColor and NSFont objects, from the user default system. But if objects conform to the NSCoding protocol they can be archived to NSData objects, which are property list–compatible objects*

**141-Helper Objects**

*Helper Objects are used throughout Cocoa and CocoaTouch, and usually take the form of a delegate or dataSource. They are commonly used to add functionality to an existing class without having to subclass it.*

**143-Differentiate Foundation vs Core Foundation**

*CoreFoundation is a general-purpose C framework whereas Foundation is a general-purpose Objective-C framework. Both provide collection classes, run loops, etc, and many of the Foundation classes are wrappers around the CF equivalents. CF is mostly open-source , and Foundation is closed-source.*

***Core Foundation****is the C-level API, which provides CFString, CFDictionary and the like.****Foundation****is Objective-C, which provides NSString, NSDictionary, etc. CoreFoundation is written in C while Foundation is written in Objective-C. Foundation has a lot more classes CoreFoundation is the common base of Foundation and Carbon.*

**144-Difference between coreData and Database**

|  |  |
| --- | --- |
| ***Database*** | ***Core Data*** |
| *Primary function is storing and fetching data* | *Primary function is graph management (although reading and writing to disk is an important supporting feature)* |
| *Operates on data stored on disk (or minimally and incrementally loaded)* | *Operates on objects stored in memory (although they can be lazily loaded from disk)* |
| *Stores “dumb” data* | *Works with fully-fledged objects that self-manage a lot of their behavior and can be subclassed and customized for further behaviors* |
| *Can be transactional, thread-safe, multi-user* | *Non-transactional, single threaded, single user (unless you create an entire abstraction around Core Data which provides these things)* |
| *Can drop tables and edit data without loading into memory* | *Only operates in memory* |
| *Perpetually saved to disk (and often crash resilient)* | *Requires a save process* |
| *Can be slow to create millions of new rows* | *Can create millions of new objects in-memory very quickly (although saving these objects will be slow)* |
| *Offers data constraints like “unique” keys* | *Leaves data constraints to the business logic side of the program* |

**145- Core data vs sqlite.**

*Core data is an object graph management framework. It manages a potentially very large graph of object instances, allowing an app to work with a graph that would not entirely fit into memory by faulting objects in and out of memory as necessary. Core Data also manages constraints on properties and relationships and maintains reference integrity (e.g. keeping forward and backwards links consistent when objects are added/removed to/from a relationship). Core Data is thus an ideal framework for building the “model” component of an MVC architecture.*

*To implement its graph management, Core Data happens to use sqlite as a disk store. Itcould have been implemented using a different relational database or even a non-relational database such as CouchDB. As others have pointed out, Core Data can also use XML or a binary format or a user-written atomic format as a backend (though these options require that the entire object graph fit into memory).*

**146-Retain cycle or Retain loop.**

*When object A retains object B, and object B retains A. Then Retain cycle happens. To overcome this use “close” method.*

*Objective-C’s garbage collector (when enabled) can also delete retain-loop groups but this is not relevant on the iPhone, where Objective-C garbage collection is not supported.*

**147-What is unnamed category.**

*A named category —****@interface Foo(FooCategory)****— is generally used to:*

*i. Extend an existing class by adding functionality.*

*ii. Declare a set of methods that might or might not be implemented by a delegate.*

*Unnamed Categories has fallen out of favor now that @protocol has been extended to support @optional methods.*

*A class extension —****@interface Foo()****— is designed to allow you to declare additional private API — SPI or System Programming Interface — that is used to implement the class innards. This typically appears at the top of the .m file. Any methods / properties declared in the class extension must be implemented in the @implementation, just like the methods/properties found in the public @interface.*

*Class extensions can also be used to redeclare a publicly readonly @property as readwrite prior to @synthesize’ing the accessors.*

*Example:*

***Foo.h***

*@interface Foo:NSObject*

*@property(readonly, copy) NSString \*bar;*

*-(void) publicSaucing;*

*@end*

***Foo.m***

*@interface Foo()*

*@property(readwrite, copy) NSString \*bar;*

*– (void) superSecretInternalSaucing;*

*@end*

*@implementation Foo*

*@synthesize bar;*

*…. must implement the two methods or compiler will warn ….*

*@end*

**148-Copy vs mutableCopy.**

*copy always creates an immutable copy.*

*mutableCopy always creates a mutable copy.*

**149- Strong vs Weak**

*The strong and weak are new ARC types replacing retain and assign respectively.*

*Delegates and outlets should be weak.*

*A****strong reference****is a reference to an object that stops it from being deallocated. In other words it creates a owner relationship.*

*A****weak reference****is a reference to an object that does not stop it from being deallocated. In other words, it does not create an owner relationship.*

**150-\_\_strong, \_\_weak, \_\_unsafe\_unretained, \_\_autoreleasing.**

*Generally speaking, these extra qualifiers don’t need to be used very often. You might first encounter these qualifiers and others when using the migration tool. For new projects however, you generally you won’t need them and will mostly use strong/weak with your declared properties.*

***\_\_strong****– is the default so you don’t need to type it. This means any object created using alloc/init is retained for the lifetime of its current scope. The “current scope” usually means the braces in which the variable is declared*

***\_\_weak****– means the object can be destroyed at anytime. This is only useful if the object is somehow strongly referenced somewhere else. When destroyed, a variable with \_\_weak is set to nil.*

***\_\_unsafe\_unretained****– is just like \_\_weak but the pointer is not set to nil when the object is deallocated. Instead the pointer is left dangling.*

***\_\_autoreleasing****, not to be confused with calling autorelease on an object before returning it from a method, this is used for passing objects by reference, for example when passing NSError objects by reference such as [myObject performOperationWithError:&tmp];*

**151-Types of NSTableView**

*Cell based and View based. In view based we can put multiple objects.*

**152-Abstract class in cocoa.**

*Cocoa doesn’t provide anything called abstract. We can create a class abstract which gets check only at runtime, compile time this is not checked.*

*@interface AbstractClass : NSObject*

*@end*

*@implementation AbstractClass*

*+ (id)alloc{*

*if (self == [AbstractClass class]) {*

*NSLog(@”Abstract Class cant be used”);*

*}*

*return [super alloc];*

*@end*

**153- Difference between HTTP and HTTPS.**

*· HTTP stands for*[*HyperText*](http://www.blogger.com/blogger.g?blogID=2378569646178591916)*Transfer*[*Protocol*](http://www.blogger.com/blogger.g?blogID=2378569646178591916)*, whereas, HTTPS is HyperText Transfer Protocol*[*Secure*](http://www.blogger.com/blogger.g?blogID=2378569646178591916)*.*

*· HTTP transmits everything as plan text, while HTTPS provides encrypted communication, so that only the recipient can decrypt and read the information. Basically, HTTPS is a combination of HTTP and*[*SSL*](http://www.blogger.com/blogger.g?blogID=2378569646178591916)*(Secure Sockets Layer). This SSL is that protocol which encrypts the data.*

*· HTTP is fast and cheap, where HTTPS is slow and expensive.*

*As, HTTPS is safe it’s widely used during payment transactions or any sensitive transactions over the internet. On the other hand, HTTP is used most of the sites over the net, even this blogspot sites also use HTTP.*

*· HTTP URLs starts with “http:// “ and use*[*port*](http://www.blogger.com/blogger.g?blogID=2378569646178591916)*80 by default, while HTTPS URLs stars with “https:// “ and use port 443.*

*· HTTP is unsafe from attacks like*[*man-in-the-middle*](http://en.support.wordpress.com/affiliate-links/)*and eavesdropping, but HTTPS is secure from these sorts of attacks.*

**154-GCD**

*Grand Central Dispatch is not just a new abstraction around what we’ve already been using, it’s an entire new underlying mechanism that makes multithreading easier and makes it easy to be as concurrent as your code can be without worrying about the variables like how much work your CPU cores are doing, how many CPU cores you have and how much threads you should spawn in response. You just use the Grand Central Dispatch API’s and it handles the work of doing the appropriate amount of work. This is also not just in Cocoa, anything running on Mac OS X 10.6 Snow Leopard can take advantage of Grand Central Dispatch ( libdispatch ) because it’s included in libSystem.dylib and all you need to do is include #import <dispatch/dispatch.h> in your app and you’ll be able to take advantage of Grand Central Dispatch.*

**155-How you attain the backward compatibility?**

* *Set the Base SDK to Current version of Mac (ex. 10.7)*
* *Set the Deployment SDK to older version (ex.1.4)*

**156-Call Back.**

*Synchronous operations are ones that happen in step with your calling code. Most of Cocoa works this way: you send a message to an object, say to format a string, etc, and by the time that line of code is “done”, the operation is complete.*

*But in the real world, some operations take longer than “instantaneous” (some intensive graphics work, but mainly high or variably latency things like disk I/O or worse, network connectivity). These operations are unpredictable, and if the code were to block until finish, it might block indefinitely or forever, and that’s no good.*

*So the way we handle this is to set up “callbacks”– you say “go off and do this operation, and when you’re done, call this other function”. Then inside that “callback” function, you start the second operation that depends on the first. In this way, you’re not spinning in circles waiting, you just get called “asynchronously” when each task is done.*

**Q:**   Which all frameworks you worked in ios ?

**Ans:**

Foundation Framework,

UIKit Framework

MapKit Framework

XCTest Framework (Unit testing)

CoreGraphics Framework

CFNetwork Framework

CoreLocation Framework

**Q:**     If you run an application on device , which has location services.Which location it will show first ?

**Ans:**

The answer depends on if the user is enabled the location services or not. If user enables the location services it will show the users current location.

**Q:**    If you use location services in your app, will it drain your battery.if yes? How can we avoid?

**Q:**    Did you used CLRegion? How its working?

**Ans:**

CLRegion class defines a geographical area that should be tracked After you create a region, you must register it with a CLLocationManager object. The location manager generates appropriate events whenever the user crosses the boundaries of the region.

You ca use the delegate to perform your  necessary operations

- (void)locationManager:(CLLocationManager\*)manager didExitRegion:(CLRegion \*)region

- (void)locationManager:(CLLocationManager \*)manager didFailWithError:(NSError \*)error

- (void)locationManager:(CLLocationManager\*)manager didExitRegion:(CLRegion \*)region

**Q:**   what is dispatchQueue?

**Ans:**

Dispatch queues are an easy way to perform tasks asynchronously and concurrently in your application. A task is simply some work that your application needs to perform.

**Q:**  Find the letter count in UItextfield? Including first character?

Q:  Tell about block and GCD

**Ans:**

Block:    Block objects are a C-level syntactic and runtime feature. Blocks are particularly useful as a callback because the block carries both the code to be executed on callback (on completion handler block) and the data needed during that execution.

GCD:    GCD (Grand Central Dispatch) provides and manages FIFO queues to which your application can submit tasks in the form of block objects. Blocks submitted to dispatch queues are executed on a set of threads completely managed by the system. No guarantee is made as to the thread on which a task executes. GCD allows three kinds of queues:

* **Main:** tasks execute serially on your application’s main thread
* **Concurrent:** tasks are dequeued in FIFO order, but run concurrently and can finish in any order.
* **Serial:** tasks execute one at a time in FIFO order

**Q:**  Difference between retain and assign ?

**Ans:**

Assign and retain was used in manual memory management (MRC) environment.

**assign** to set a property’s pointer to the address of the object without retaining it. use **weak** to have the property point to nil automatically if the object assigned to it is deallocated .

Use **retain** by default and will manage the object’s reference count automatically whether another object is assigned to the property or it’s set to nil; In ARC you will use **strong** instead of **retain**.

**Q:**   How can we use NSOperation queue? what is the advantage of using it ?

**Ans:**

NSOperationQueue can be more suitable for long-running operations that may need to be cancelled or have complex dependencies. It is possibles to cancel operations that have been enqueued in an NSOperationQueue (as far as the operations support it).  
NSOperationQueue gives you a lot more control over how your operations are executed. You can define dependencies between individual operations.

**Q:**   How to fix tableviewcell dequeue issue ?

**Q:**Difference between ScrollviewdidScroll and didendDecelerating delegates?

**Ans:**

scrollViewDidScroll:    Tells the delegate when the user scrolls the content view within the receiver.

scrollViewDidEndDecelerating:   Tells the delegate that the scroll view has ended decelerating the scrolling movement.

**Q:  M**ajor difference between delegate and notification?

**Ans:**

Delegate: One to one message passing between objects.

Notification:  One to many or broadcasting type of message passing  between objects.

**Q:  P**ush notification working in ios?

**Ans:**

Push notification is known as remote notifications—arrive from outside a device.  They originate on a remote server—the application’s provider—and are pushed to applications on devices (via the Apple Push Notification service) when there are messages to see or data to download.

**Q:**   A tableview with while tapping on cell show a popOver. and tapping on it move to another view controller, is it possible if yes? then how?

**Ans:**    Its Possible.

**Q:**   Bundle & document directory, sandbox ?

**Ans:**

Sandbox:  sandbox is a set of fine-grained controls that limit the app’s access to files, preferences, network resources, hardware, and so on. As part of the sandboxing process, the system installs each app in its own sandbox directory, which acts as the home for the app and its data.

Bundle :  A bundle is a directory with a standardized hierarchical structure that holds executable code and the resources used by that code.

Documents :  Documents directory allows you to store files and subdirectories your app creates or may need.

**Q:**   Difference b/w collectionView & uitableview?

**Ans:**

**What is UICollectionView.**  
The UICollectionView class manages an ordered collection of data items and presents them using customizable layouts. Collection views provide the same general function as table views except that a collection view is able to support more than just single-column layouts. Collection views support customizable layouts that can be used to implement multi-column grids, tiled layouts, circular layouts, and many more.

**What is UITableView.**

An instance of UITableView (or simply, a table view) is a means for displaying and editing hierarchical lists of information.

A table view displays a list of items in a single column. UITableView is a subclass of UIScrollView, which allows users to scroll through the table, although UITableView allows vertical scrolling only. The cells comprising the individual items of the table are UITableViewCell objects; UITableView uses these objects to draw the visible rows of the table.

**Q:**   DB updation when app new version comes ?

**Q:**   How to do upload of large sized video file to server ? while uploading user presses “Homebutton” How long will execute?

**Q:**  “BgTask “  ? how long execute?

**Ans:**

if you want to perform some long running in background you will do it with  the help **beginBackgroundTaskWithExpirationHandler**:

**Example**:

bgTask = [app beginBackgroundTaskWithExpirationHandler:^{

[app endBackgroundTask:bgTask];

bgTask = UIBackgroundTaskInvalid;

}];

**Q:**  UI updation in seperate thread is it possible ?

**Ans:**

UI Updating only possible using main() thread. If  you are doing operation in custom thread, and if you want to do some UI operation you should get main() thread and update the UI.

1.    Cocoa.

Cocoa is an application environment for both the Mac OS X operating system and iOS.  It consists of a suite of object-oriented software libraries, a runtime system, and an integrated development environment. Carbon is an alternative environment in Mac OS X, but it is a compatibility framework with procedural programmatic interfaces intended to support existing Mac OS X code bases.

2.     Frameworks that make Cocoa.

∙       Appkit (Application Kit)

∙       Foundation

3.     Objective-C.

Objective-C is a very dynamic language. Its dynamism frees a program from compile-time and link-time constraints and shifts much of the responsibility for symbol resolution to runtime, when the user is in control. Objective-C is more dynamic than other programming languages because its dynamism springs from three sources:

∙         Dynamic typing—determining the class of an object at runtime

∙         Dynamic binding—determining the method to invoke at runtime

∙         Dynamic loading—adding new modules to a program at runtime

4.     Objective-C vs C/C++.

∙       The Objective-C class allows a method and a variable with the exact same name. In C++, they must be different.

∙       Objective-C does not have a constructor or destructor. Instead it has init and dealloc methods, which must be called explicitly.

∙       Objective-C uses + and - to differentiate between class method (known as factory method in Java) and instance methods, C++ uses static to specify a factory method.

∙       Multiple inheritance is not allowed in Obj-C, however we can use protocol to some extent.

∙       Obj-C has runtime binding leading to dynamic linking.

∙       Obj-C has categories.

∙       Objective-C has a work-around for method overloading, but none for operator overloading.

∙       Objective-C also does not allow stack based objects. Each object must be a pointer to a block of memory.

∙       In Objective-C the message overloading is faked by naming the parameters. C++ actually does the same thing but the compiler does the name mangling for us. In Objective-C, we have to mangle the names manually. If you go in deep you will see a complete method names are as : `addA:withB:` which is a selector and `:` is used for parameter.

∙       One of C++'s advantages and disadvantages is automatic type coercion.

∙       Another feature C++ has that is missing in Objective-C is references. Because pointers can be used wherever a reference is used, there isn't much need for references in general.

∙       Templates are another feature that C++ has that Objective-C doesn't. Templates are needed because C++ has strong typing and static binding that prevent generic classes, such as List and Array.

5.     Appilcation Kit/App kit.

The Application Kit is a framework containing all the objects you need to implement your graphical, event-driven user interface: windows, panels, buttons, menus, scrollers, and text fields. The Application Kit handles all the details for you as it efficiently draws on the screen, communicates with hardware devices and screen buffers, clears areas of the screen before drawing, and clips views.

You also have the choice at which level you use the Application Kit:

∙         Use Interface Builder to create connections from user interface objects to your application objects.

∙         Control the user interface programmatically, which requires more familiarity with AppKit classes and protocols.

∙         Implement your own objects by subclassing NSView or other classes.

6.   Foundation Kit.

The Foundation framework defines a base layer of Objective-C classes. In addition to providing a set of useful primitive object classes, it introduces several paradigms that define functionality not covered by the Objective-C language. The Foundation framework is designed with these goals in mind:

∙         Provide a small set of basic utility classes.

∙         Make software development easier by introducing consistent conventions for things such as deallocation.

∙         Support Unicode strings, object persistence, and object distribution.

∙         Provide a level of OS independence, to enhance portability.

7.     Dynamic and Static Typing.

Static typed languages are those in which type checking is done at compile-time, whereas dynamic typed languages are those in which type checking is done at run-time.

Objective-C is a dynamically-typed language, meaning that you don't have to tell the compiler what type of object you're working with at compile time. Declaring a type for a varible is merely a promise which can be broken at runtime if the code leaves room for such a thing. You can declare your variables as type id, which is suitable for any Objective-C object.

8.     Selectors

In Objective-C, selector has two meanings. It can be used to refer simply to the name of a method when it’s used in a source-code message to an object. It also, though, refers to the unique identifier that replaces the name when the source code is compiled. Compiled selectors are of type SEL. All methods with the same name have the same selector. You can use a selector to invoke a method on an object—this provides the basis for the implementation of the target-action design pattern in Cocoa.

[friend performSelector:@selector(gossipAbout:) withObject:aNeighbor];

is equivalent to:

[friend gossipAbout:aNeighbor];

9.     Class Introspection

∙         Determine whether an objective-C object is an instance of a class

        [obj isMemberOfClass:someClass];

∙         Determine whether an objective-C object is an instance of a class or its descendants

        [obj isKindOfClass:someClass];

∙         The version of a class

        [MyString version]

∙         Find the class of an Objective-C object

        Class c = [obj1 class]; Class c = [NSString class];

∙         Verify 2 Objective-C objects are of the same class

[obj1 class] == [obj2 class]

10. Immutable vs Mutable

Immutable objects cant be changed. However they are just pointing to some location where stored values are made constant. You can change that reference to other location.  You can change the value of  mutable objects.

11. Category

In Objective-C, new functionality can be added to existing classes by adding a new category. Shared libraries that depend on a base class that has been extended will continue to work, and new classes created can call the additional methods. Category on NSString will become available on NSMutableString.

We can add new ivar to a category using these: (It is required when we have APIs of some other, don’t have code)

@interface NSFruit (Liking)

@property ( nonatomic ) BOOL liked ;

@end

@implementation NSFruit (Liking)

-(BOOL)liked{

    return [ objc\_getAssociatedObject( self, "\_abliked" ) boolValue ] ;

}

-(void)setLiked:(BOOL)b{

objc\_setAssociatedObject(self, "\_abliked",  [ NSNumber numberWithBool:b ], OBJC\_ASSOCIATION\_RETAIN\_NONATOMIC ) ;

}

@end

12. Proxy

As long as there aren't any extra instance variables, any subclass can proxy itself as its superclass with a single call. Each class that inherits from the superclass, no matter where it comes from, will now inherit from the proxied subclass. Calling a method in the superclass will actually call the method in the subclass. For libraries where many objects inherit from a base class, proxying the superclass can be all that is needed.

13. Category vs Inheritance

Category allows adding methods only; no data members can be added as in Inheritance both data and methods can be added. Category’s scope is full application whereas inheritance’s scope that particular file. However in new compilers using associatedObjects you can add data members using category.

14. Why category is better than inheritance?  
       If category is used, you can use same class, no need to remember a new class-name. Category created on a base class is available on sub classes.

15. Fast enumeration

Fast enumeration is a language feature that allows you to enumerate over the contents of a collection. (Your code will also run faster because the internal implementation reduces message send overhead and increases pipelining potential.)

Enum is preferred over loop for the same reason.

16. Protocol

∙  A Protocol in Objective-C is identical in functionality to an interface in Java, or a purely virtual class in C++.

∙  A protocol is means to define a list of required and/or optional methods that a class implements. If a class adopts a protocol, it must implement all required methods in the protocols it adopts.

∙  Cocoa uses protocols to support interprocess communication through Objective-C messages. In addition, since Objective-C does not support multiple inheritance, you can achieve similar functionality with protocols, as a class can adopt more than one protocol.

∙  A good example of a protocol is NSCoding, which has two required methods that a class must implement. This protocol is used to enable classes to be encoded and decoded, that is, archiving of objects by writing to permanent storage.

17. Formal Protocols

Formal Protocols allow us to define the interface for a set of methods, but implementation is not done. Formal Protocols are useful when you are using DistributedObjects, because they allow you to define a protocol for communication between objects, so that the DO system doesn't have to constantly check whether or not a certain method is implemented by the distant object.

18. Formal vs informal protocol.

In addition to formal protocols, you can also define an informal protocol by grouping the methods in a category declaration:

@interface NSObject (MyProtocol)

        //someMethod();

@end

**Informal protocols are typically declared as categories of the NSObject class**, because that broadly associates the method names with any class that inherits from NSObject. Because all classes inherit from the root class, the methods aren’t restricted to any part of the inheritance hierarchy. (It is also possible to declare an informal protocol as a category of another class to limit it to a certain branch of the inheritance hierarchy, but there is little reason to do so.)

When used to declare a protocol, a category interface doesn’t have a corresponding implementation. Instead, classes that implement the protocol declare the methods again in their own interface files and define them along with other methods in their implementation files.

An informal protocol bends the rules of category declarations to list a group of methods but not associate them with any particular class or implementation.

Being informal, protocols declared in categories don’t receive much language support. There’s no type checking at compile time nor a check at runtime to see whether an object conforms to the protocol. To get these benefits, you must use a formal protocol. An informal protocol may be useful when all the methods are optional, such as for a delegate, but (in Mac OS X v10.5 and later) it is typically better to use a formal protocol with optional methods.

19. Protocol's two types : @optional vs @required

Protocol methods can be marked as optional using the @optional keyword. Corresponding to the @optional modal keyword, there is a @required keyword to formally denote the semantics of the default behavior. You can use @optional and @required to partition your protocol into sections as you see fit. If you do not specify any keyword, the default is @required.

                @protocol MyProtocol

                                @optional

                                                -(void) optionalMethod;

                                @required

                                                -(void) requiredMethod;

                @end

20. Memory Management  
      In MRC or for Foundation variables like CG... , CF...:

If you alloc, retain, or copy/mutablecopy it, it's your job to release/autorelease it. Otherwise it isn't.  
In ARC, for Foundation varialbes you need to release/autoreleast it. For others(NS... or subclass from NS...) no need to release/autorelease.

21. Retain Counting

Every object has a RetainCount that goes up by one when the object gets a retain message. It goes down by one when the object gets a release message. When the RetainCount reaches 0, the object will call [self dealloc], thereby releasing the object's memory.

22. Copy vs assign vs retain

∙         Assign is for primitive values like BOOL, NSInteger or double. For objects use retain or copy, depending on if you want to keep a reference to the original object or make a copy of it.

∙         **assign**: In your setter method for the property, there is a simple assignment of your instance variable to the new value, eg:

(void)setString:(NSString\*)newString{

        string = newString;

 }

This can cause problems since Objective-C objects use reference counting, and therefore by not retaining the object, there is a chance that the string could be deallocated whilst you are still using it.

∙         **retain**: this *retains* the new value in your setter method. For example:

This is safer, since you explicitly state that you want to maintain a reference of the object, and you must release it before it will be deallocated.

(void)setString:(NSString\*)newString{

        [newString retain];

          [string release];

 string = newString;

}

∙         **copy**: this makes a copy of the string in your setter method:

This is often used with strings, since making a copy of the original object ensures that it is not changed whilst you are using it.

(void)setString:(NSString\*)newString{

         if(string!=newString){

                        [string release];

                        string = [newString copy];

         }

 }

23. alloc vs new

“alloc” creates a new memory location but doesn’t initializes it as compared to “new”.

24. release vs pool drain

“release” frees a memory. “drain” releases the NSAutoreleasePool itself.

25. NSAutoReleasePool : release vs drain

Strictly speaking, from the big picture perspective drain is not equivalent to release:

In a reference-counted environment, drain does perform the same operations as release, so the two are in that sense equivalent. To emphasise, this means you do not leak a pool if you use drain rather than release.

In a garbage-collected environment, release is a no-op. Thus it has no effect. drain, on the other hand, contains a hint to the collector that it should "collect if needed". Thus in a garbage-collected environment, using drain helps the system balance collection sweeps.

26. autorelease vs release

Autorelase: By sending an object an autorelease message, it is added to the local AutoReleasePool, and you no longer have to worry about it, because when the AutoReleasePool is destroyed (as happens in the course of event processing by the system) the object will receive a release message, its RetainCount will be decremented, and the GarbageCollection system will destroy the object if the RetainCount is zero.

Release: retain count is decremented at this point.

27. Autorelease Pool

Autorelease pools provide a mechanism whereby you can send an object a “deferred” release message. This is useful in situations where you want to relinquish ownership of an object, but want to avoid the possibility of it being deallocated immediately (such as when you return an object from a method). Typically, you don’t need to create your own autorelease pools, but there are some situations in which either you must or it is beneficial to do so.

28. How autorelease pool is managed.

Every time -autorelease is sent to an object, it is added to the inner-most autorelease pool. When the pool is drained, it simply sends -release to all the objects in the pool.

Autorelease pools are simply a convenience that allows you to defer sending -release until "later". That "later" can happen in several places, but the most common in Cocoa GUI apps is at the end of the current run loop cycle.

29.  Memory Leak

If RetainingAndReleasing are not properly used then RetainCount for AnObject doesn’t reach 0. It doesn’t crash the application.

30. Event Loop

In a Cocoa application, user activities result in events. These might be mouse clicks or drags, typing on the keyboard, choosing a menu item, and so on. Other events can be generated automatically, for example a timer firing periodically, or something coming in over the network. For each event, Cocoa expects there to be an object or group of objects ready to handle that event appropriately. The event loop is where such events are detected and routed off to the appropriate place. Whenever Cocoa is not doing anything else, it is sitting in the event loop waiting for an event to arrive. (In fact, Cocoa doesn't poll for events as suggested, but instead its main thread goes to sleep. When an event arrives, the OS wakes up the thread and event processing resumes. This is much more efficient than polling and allows other applications to run more smoothly).

Each event is handled as an individual thing, then the event loop gets the next event, and so on. If an event causes an update to be required, this is checked at the end of the event and if needed, and window refreshes are carried out.

31. @property  @synthesize @dyanamic

∙       Properties are a feature in Objective-C that allow us to automatically generate accessors

∙       The @synthesize directive automatically generates the setters and getters for us, so all we have to implement for this class is the dealloc method.

∙       @synthesize will generate getter and setter methods for your property. @dynamic just tells the compiler that the getter and setter methods are implemented not by the class itself but somewhere else (like the superclass)

∙       Uses for @dynamic are e.g. with subclasses of NSManagedObject (CoreData) or when you want to create an outlet for a property defined by a superclass that was not defined as an outlet:

32. @property vs @synthesize

@property - declares a property.

@synthesize - creates getter and setter methods for a property

Example:

       @property float value;

 is equivalent to:

        -(float)value;

        - (void)setValue:(float)newValue;

@synthesize defines the properties.

33. Relation between iVar and @property.

iVar are just instance variables. It cant be accessed unless we create accessors, which are generated by @property. iVar and its counterpart @property can be of different names.

@interface Box : NSObject{

    NSString \*boxName;

}

@property (strong) NSString \*boxDescription;//this will become another ivar

-(void)aMethod;

@end

@implementation Box

@synthesize boxDescription=boxName;//now boxDescription is accessor for name

-(void)aMethod {

    NSLog(@"name=%@", boxName);

     NSLog(@"boxDescription=%@",self.boxDescription);

    NSLog(@"boxDescription=%@",boxDescription); //throw an error

}

@end

34. Differnce between boxName and self.boxName.

boxName: Accessing directly.

self. boxName: Accessing boxName through accessors. If property/synthesize is not there it will throw error.

35. What it does “@synthesize boxDescription=boxName;”  ?

Here you can use boxName or self.boxName. We cant use boxDescription.

36. atomic vs nonatomic

@property(atomic, retain)…….

@property(retain)…….

@property(nonatomic, retain)…….

atomic is the default behavior, so first and second are same.

Assuming that you are @synthesizing the method implementations, atomic vs. non-atomic changes the generated code. If you are writing your own setter/getters, atomic/nonatomic/retain/assign/copy are merely advisory.

With atomic, the synthesized setter/getter will ensure that a *whole* value is always returned from the getter or set by the setter, regardless of setter activity on any other thread. That is, if thread A is in the middle of the getter while thread B calls the setter, an actual viable value -- an autoreleased object, most likely -- will be returned to the caller in A.

In nonatomic, no such guarantees are made. Thus, nonatomic is considerably faster than atomic.

What atomic does **not** do is make any guarantees about thread safety. If thread A is calling the getter simultaneously with thread B and C calling the setter with different values, thread A may get any one of the three values returned -- the one prior to any setters being called or either of the values passed into the setters in B and C. Likewise, the object may end up with the value from B or C, no way to tell.

Ensuring data integrity -- one of the primary challenges of multi-threaded programming -- is achieved by other means.

37. Collection

In Cocoa and Cocoa Touch, a collection is a Foundation framework class used for storing and managing groups of objects. Its primary role is to store objects in the form of either an array, a dictionary, or a set.

38. Threads and how to use

Use this class when you want to have an Objective-C method run in its own thread of execution. Threads are especially useful when you need to perform a lengthy task, but don’t want it to block the execution of the rest of the application. In particular, you can use threads to avoid blocking the main thread of the application, which handles user interface and event-related actions. Threads can also be used to divide a large job into several smaller jobs, which can lead to performance increases on multi-core computers.

Two ways to create threads…

∙         detachNewThreadSelector:toTarget:withObject:

∙         Create instances of NSThread and start them at a later time using the “start” method.

NSThread is not as capable as Java’s Thread class, it lacks

∙         Built-in communication system.

∙         An equivalent of “join()”

39.Threadsafe

When it comes to threaded applications, nothing causes more fear or confusion than the issue of handling signals. Signals are a low-level BSD mechanism that can be used to deliver information to a process or manipulate it in some way. Some programs use signals to detect certain events, such as the death of a child process. The system uses signals to terminate runaway processes and communicate other types of information.

The problem with signals is not what they do, but their behavior when your application has multiple threads. In a single-threaded application, all signal handlers run on the main thread. In a multithreaded application, signals that are not tied to a specific hardware error (such as an illegal instruction) are delivered to whichever thread happens to be running at the time. If multiple threads are running simultaneously, the signal is delivered to whichever one the system happens to pick. In other words, signals can be delivered to any thread of your application.

The first rule for implementing signal handlers in applications is to avoid assumptions about which thread is handling the signal. If a specific thread wants to handle a given signal, you need to work out some way of notifying that thread when the signal arrives. You cannot just assume that installation of a signal handler from that thread will result in the signal being delivered to the same thread.

40. Which one is thread-safe-atomic or non-atomic?

Immutable objects are generally threadsafe. E.g, NSString

None are threadsafe.

atomic guarantees atomic access to the variable but it DOESN'T make your code thread safe. Neither does non-atomic.

With "atomic", the synthesized setter/getter methods will ensure that a whole value is always returned from the getter or set by the setter, regardless of setter activity on any other thread. So if thread A is in the middle of the getter while thread B calls the setter, an actual viable value will be returned to the caller in A. For nonatomic, you have no such guarantees.

41. @synchronized

Objective-C supports multithreading in applications. Therefore, two threads can try to modify the same object at the same time, a situation that can cause serious problems in a program. To protect sections of code from being executed by more than one thread at a time, Objective-C provides the @synchronized() directive.

The @synchronized()directive locks a section of code for use by a single thread. Other threads are blocked until the thread exits the protected code—that is, when execution continues past the last statement in the @synchronized() block.

The @synchronized() directive takes as its only argument any Objective-C object, including self. This object is known as a mutual exclusion semaphore or mutex. It allows a thread to lock a section of code to prevent its use by other threads. You should use separate semaphores to protect different critical sections of a program. It’s safest to create all the mutual exclusion objects before the application becomes multithreaded, to avoid race conditions.

 (void)criticalMethod{

                          @synchronized(self) {

                                              // Critical code.

         }

}

42. MVC

It is useful to divide the complex task of computer application design into domains in order to simplify the process. Object oriented approaches are modular in philosophy, so the Model View Controller design pattern is a popular way to make logical divisions among class responsibilities.

Each object oriented programming environment/language has a slightly different definition/convention of MVC.

The main advantage of adopting a design pattern like MVC is that it allows the code in each unit to be decoupled from the others, making it more robust and immune to changes in other code.

Within the scope of Cocoa, MVC is an extremely important pattern. The major enhancements that Apple have made to Cocoa, namely Bindings and Core Data, are manifestly based on the MVC pattern.

Model: A Model object:

∙         is usually a simple subclass of NSObject (or an instance of NSManagedObject for CoreData)

∙         has a set of instance variables in which to store its data

∙         has a series of accessor methods for those ivars

∙         has one or more init: methods to return new instances to other classes

∙         has a dealloc method

∙         may have custom methods to manipulate the model objects internal data

∙         is frequently reusable

View: A View object:

∙         is some subclass of NSView

∙         contains a drawRect: method which is the basis of all drawing

∙         is rarely subclassed or modified

∙         makes extensive use of delegates for customisation

∙         is generally reusable

Controller: A Controller object:

∙         mediates between model and view

∙         is usually a subclass of NSObject

∙         contains the outlets and actions for IB

∙         contains ivars and collections to own and hold model objects

∙         has methods for manipulating and composing model objects

∙         contains the main awakeFromNib method

∙         is instantiated in the nib file/s

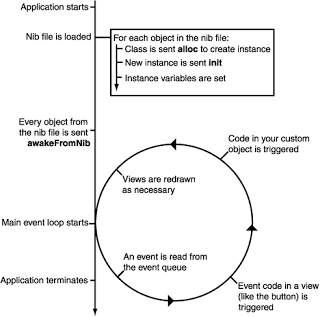
∙         contains the business logic of the program

∙         is rarely reusable

43. Application lifecycle

Chronology of an application: When the process is started, it runs the NSApplicationMain function, which creates an instance of NSApplication. The application object reads the main nib file and unarchives the objects inside. The objects are all sent the message awakeFromNib. Then the application object checks for events.

This lifecylce of a typical cocoa application is depicted in the below diagram.



44. NSBundle

An NSBundle object represents a location in the file system that groups code and resources that can be used in a program. NSBundle objects locate program resources, dynamically load and unload executable code, and assist in localization. You build a bundle in Xcode using one of these project types: Application, Framework, plug-ins.

45. Delegate

The delegation is a commonly used pattern in object-oriented programming. It is a situation where an object, instead of performing a tasks itself, delegates that task to another, helper object. The helper object is called the delegate.

A delegate allows one object to send messages to another object when an event happens.

A delegate is just an object that another object sends messages to when certain things happen, so that the delegate can handle app-specific details the original object wasn't designed for. It's a way of customizing behavior without subclassing.

They are never retained.

46. Notification and Observers

A notification is a message sent to one or more observing objects to inform them of an event in a program. The notification mechanism of Cocoa follows a **broadcast** model. It is a way for an object that initiates or handles a program event to communicate with any number of objects that want to know about that event. These recipients of the notification, known as observers, can adjust their own appearance, behavior, and state in response to the event. The object sending (or posting) the notification doesn’t have to know what those observers are. Notification is thus a powerful mechanism for attaining coordination and cohesion in a program. It reduces the need for strong dependencies between objects in a program (such dependencies would reduce the reusability of those objects). Many classes of the Foundation, AppKit, and other Objective-C frameworks define notifications that your program can register to observe.

The centerpiece of the notification mechanism is a per-process singleton object known as the notification center (**NSNotificationCenter**). When an object posts a notification, it goes to the notification center, which acts as a kind of clearing house and broadcast center for notifications. Objects that need to know about an event elsewhere in the application register with the notification center to let it know they want to be notified when that event happens. Although the notification center delivers a notification to its observers synchronously, you can post notifications asynchronously using a notification queue (NSNotificationQueue).

47. Delegate vs Notification

∙       The concept of notification differs from delegation in that it allows a message to be sent to more than one object. It is more like a broadcast rather than a straight communication between two objects. It removes dependencies between the sending and receiving object(s) by using a notification center to manage the sending and receiving of notifications. The sender does not need to know if there are any receivers registered with the notification center. There can be one, many or even no receivers of the notification registered with the notification center. Simply, Delegate is 1-to-1 object and Notification can be \*-to-\* objects.

∙       The other difference between notifications and delegates is that there is no possibility for the receiver of a notification to return a value to the sender.

∙       Typical uses of notifications might be to allow different objects with an application to be informed of an event such as a file download completing or a user changing an application preference. The receiver of the notification might then perform additional actions such as processing the downloaded file or updating the display.

48. Plist

Property lists organize data into named values and lists of values using several object types. These types give you the means to produce data that is meaningfully structured, transportable, storable, and accessible, but still as efficient as possible. Property lists are frequently used by applications running on both Mac OS X and iOS. The property-list programming interfaces for Cocoa and Core Foundation allow you to convert hierarchically structured combinations of these basic types of objects to and from standard XML. You can save the XML data to disk and later use it to reconstruct the original objects.

The user defaults system, which you programmatically access through the NSUserDefaults class, uses property lists to store objects representing user preferences. This limitation would seem to exclude many kinds of objects, such as NSColor and NSFont objects, from the user default system. But if objects conform to the NSCoding protocol they can be archived to NSData objects, which are property list–compatible objects

49. Responder Chain and First Responder

A ResponderChain is a hierarchy of objects that have the opportunity to respond to events received.

The first object in the ResponderChain is called the FirstResponder.

50. Helper Objects

Helper Objects are used throughout Cocoa and CocoaTouch, and usually take the form of a delegate or dataSource. They are commonly used to add functionality to an existing class without having to subclass it.

51. TableView’s delegate, datasource

“delegate” methods control tableview behavior.

“datasource methods focus on providing and effecting data.

**Delegate** of NSTableView

∙         (void)tableViewSelectionDidChange:(NSNotification \*)aNotification

∙         NSTableViewSelectionDidChangeNotification

To be a **data source** for an NSTableView, you just have to implement three methods:

∙         (int)numberOfRowsInTableView:(NSTableView \*)aTableView                 *- Returns the number of rows to display*

∙         (id)tableView:(NSTableView\*)aTableView

 objectValueForTableColumn:(NSTableColumn \*)aTableColumn

 row:(int)rowIndex - *Returns data for a given row in a given column. This data is fed into the NSCell set up for that column.*

∙         (void)tableView:(NSTableView\*)aTableView

 setObjectValue:(id)anObject

forTableColumn:(NSTableColumn \*)aTableColumn

row:(int)rowIndex - *Called when the user changes the value of a given data cell. Don't implement this if you want a read only NSTableView*.

52. KVC

Key-value coding is a mechanism for accessing an object’s properties indirectly, using strings to identify properties, rather than through invocation of an accessor method or accessing them directly through instance variables. E.g. NSDictionary and NSMutableDictionary

 "Keys" are just strings, and "values" can be any type of object.

53. Key value path

Cocoa makes a distinction between "keys" and "key paths". A "key" allows you to get a value on an object. A "key path" allows you to chain multiple keys together, separated by dots.

For example :

      [p valueForKeyPath:@"spouse.name"];

*… is exactly the same as this…*

      [[p valueForKey:@"spouse"] valueForKey:@"name"];

54. KVO

Key-value observing is a mechanism that allows objects to be notified of changes to specified properties of other objects. Key-value observing is a mechanism that allows objects to be notified of changes to specified properties of other objects.

55. KVB or Binding

In simple terms KVB=KVC + KVO

A binding is an attribute of one object that may be bound to a property in another such that a change in either one is reflected in the other. For example, the “value” binding of a text field might be bound to the temperature attribute of a particular model object. More typically, one binding might specify that a controller object “presents” a model object and another binding might specify that the value of a text field be tied to the temperature property of the object presented by the controller.

 Cocoa bindings reduces the code dependencies between models, views and controllers, supports multiple ways of viewing your data, and automatically synchronizes views when models change. Cocoa bindings provides extensible controllers, protocols for models and views to adopt, and additions to classes in Foundation and the Application Kit. You can eliminate most of your glue code by using bindings available in Interface Builder to connect controllers with models and views.

56. #import and @class filename

#import brings the entire header file in question into the current file; any files that THAT file #imports are also included. @class, on the other hand (when used on a line by itself with some class names), just tells the compiler "Hey, you're going to see a new token soon; it's a class, so treat it that way).

57. Singleton class

Only one instance of that class is created in the application.

@interface SomeManager : NSObject

             + (id)singleton;

 @end

 @implementation SomeManager

            + (id)singleton {

                                 static id sharedMyManager = nil;

                                 @synchronized([MyObject class]){

                                                     if (sharedMyManager == nil) {

                                                                         sharedMyManager = [[self alloc] init];

                                                      }

                                 }

                                return sharedMyManager;

            }

 @end  
  
//using block

+ (id) singleton {

    static SomeManager \*sharedMyManager = nil;

    static dispatch\_once\_t  onceToken;

    dispatch\_once(&onceToken, ^{

        sharedMyManager = [[self alloc] init];

    });

    return sharedMyManager;

}

58. File’s owner

The File's Owner of your primary nib file is, by default, the NSApplication class and this comes ready-made for you when you create your Cocoa application. If your application has just the one nib file, you really don't need to worry about File's Owner. But if that's true, your application is probably really trivial or not well written.

The File's Owner of the nib is the object that makes communication possible between this new nib and other parts of the application.

59. Whats the difference between frame and bounds?

The frame of a view is the rectangle, expressed as a location (x,y) and size (width,height) relative to the superview it is contained within. The bounds of a view is the rectangle, expressed as a location (x,y) and size (width,height) relative to its own coordinate system (0,0).

60. What’s the NSCoder class used for?

NSCoder is an abstractClass which represents a stream of data. They are used in Archiving and Unarchiving objects. NSCoder objects are usually used in a method that is being implemented so that the class conforms to the protocol. (which has something like encodeObject and decodeObject methods in them).

61. Implement the following methods: retain, release, autorelease.

-(id) retain{

                NSIncrementExtraRefCount(self);

                return self;

}

-(id) release{

                if(NSDecrementExtraRefCountWasZero(self)){

                                NSDeallocObject(self);

                }

}

-(id) autorelease{

                //add the object to the autorelease pool

                [NSAutoreleasePool addObject:self];

                return self;

}

62. Implement your own synthesized methods for the property NSString \*title.

-(NSString\*) title {

                return title;

}

-(void) setTitle: (NSString \*) newTitle{

                if(newTitle != title){

                                [title release];

                                title=[newTitle retain]; // or copy as per your need

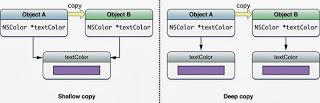
                }

}

64. Shallow copying vs deep copy.

Copies of objects can be shallow or deep. Both shallow- and deep-copy approaches directly duplicate scalar properties but differ on how they handle pointer references, particularly references to objects (for example, NSString \*str). A deep copy duplicates the objects referenced while a shallow copy duplicates only the references to those objects. So if object A is shallow-copied to object B, object B refers to the same instance variable (or property) that object A refers to. Deep-copying objects is preferred to shallow-copying, especially with value objects.

Shallow is “by Reference”, Deep is “by Value”



65. Differentiate Foundation vs Core Foundation

CoreFoundation is a general-purpose C framework whereas Foundation is a general-purpose Objective-C framework. Both provide collection classes, run loops, etc, and many of the Foundation classes are wrappers around the CF equivalents. CF is mostly open-source , and Foundation is closed-source.

**Core Foundation** is the C-level API, which provides CFString, CFDictionary and the like. **Foundation** is Objective-C, which provides NSString, NSDictionary, etc. CoreFoundation is written in C while Foundation is written in Objective-C. Foundation has *a lot* more classes CoreFoundation is the common base of Foundation and Carbon.

66. Difference between coreData and Database

|  |  |
| --- | --- |
| **Database** | **Core Data** |
| Primary function is storing and fetching data | Primary function is graph management (although reading and writing to disk is an important supporting feature) |
| Operates on data stored on disk (or minimally and incrementally loaded) | Operates on objects stored in memory (although they can be lazily loaded from disk) |
| Stores "dumb" data | Works with fully-fledged objects that self-manage a lot of their behavior and can be subclassed and customized for further behaviors |
| Can be transactional, thread-safe, multi-user | Non-transactional, single threaded, single user (unless you create an entire abstraction around Core Data which provides these things) |
| Can drop tables and edit data without loading into memory | Only operates in memory |
| Perpetually saved to disk (and often crash resilient) | Requires a save process |
| Can be slow to create millions of new rows | Can create millions of new objects in-memory very quickly (although saving these objects will be slow) |
| Offers data constraints like "unique" keys | Leaves data constraints to the business logic side of the program |

67. Core data vs sqlite.

Core data is an object graph management framework. It manages a potentially very large graph of object instances, allowing an app to work with a graph that would not entirely fit into memory by faulting objects in and out of memory as necessary. Core Data also manages constraints on properties and relationships and maintains reference integrity (e.g. keeping forward and backwards links consistent when objects are added/removed to/from a relationship). Core Data is thus an ideal framework for building the "model" component of an MVC architecture.

To implement its graph management, Core Data *happens* to use sqlite as a disk store. It *could* have been implemented using a different relational database or even a non-relational database such as CouchDB. As others have pointed out, Core Data can also use XML or a binary format or a user-written atomic format as a backend (though these options require that the entire object graph fit into memory).

68. Retain cycle or Retain loop.

When object A retains object B, and object B retains A. Then Retain cycle happens. To overcome this use “close” method.

Objective-C's garbage collector (when enabled) can also delete retain-loop groups  but this is not relevant on the iPhone, where Objective-C garbage collection is not supported.

69. What is unnamed category.

A named category -- **@interface Foo(FooCategory)** -- is generally used to:

    i. Extend an existing class by adding functionality.

  ii. Declare a set of methods that might or might not be implemented by a delegate.

Unnamed Categories has fallen out of favor now that @protocol has been extended to support @optional methods.

A class extension -- **@interface Foo()** -- is designed to allow you to declare additional private API -- SPI or System Programming Interface -- that is used to implement the class innards. This typically appears at the top of the .m file. Any methods / properties declared in the class extension must be implemented in the @implementation, just like the methods/properties found in the public @interface.

Class extensions can also be used to redeclare a publicly readonly @property as readwrite prior to @synthesize'ing the accessors.

Example:

**Foo.h**

@interface Foo:NSObject

     @property(readonly, copy) NSString \*bar;

    -(void) publicSaucing;

@end

**Foo.m**

@interface Foo()

     @property(readwrite, copy) NSString \*bar;

     - (void) superSecretInternalSaucing;

@end

@implementation Foo

     @synthesize bar;

.... must implement the two methods or compiler will warn ....

@end

70. Copy vs mutableCopy.

copy always creates an immutable copy.

mutableCopy always creates a mutable copy.

71. Strong vs Weak

The strong and weak are new ARC types replacing retain and assign respectively.

Delegates and outlets should be weak.

A **strong reference** is a reference to an object that stops it from being deallocated. In other words it creates a owner relationship.

A **weak reference** is a reference to an object that does not stop it from being deallocated. In other words, it does not create an owner relationship.

72. \_\_strong, \_\_weak, \_\_unsafe\_unretained, \_\_autoreleasing.

Generally speaking, these extra qualifiers don’t need to be used very often. You might first encounter these qualifiers and others when using the migration tool. For new projects however, you generally you won’t need them and will mostly use strong/weak with your declared properties.

**\_\_strong** – is the default so you don’t need to type it. This means any object created using alloc/init is retained for the lifetime of its current scope. The “current scope” usually means the braces in which the variable is declared

**\_\_weak** – means the object can be destroyed at anytime. This is only useful if the object is somehow strongly referenced somewhere else. When destroyed, a variable with \_\_weak is set to nil.

**\_\_unsafe\_unretained** – is just like \_\_weak but the pointer is not set to nil when the object is deallocated. Instead the pointer is left dangling.

**\_\_autoreleasing**, not to be confused with calling autorelease on an object before returning it from a method, this is used for passing objects by reference, for example when passing NSError objects by reference such as [myObject performOperationWithError:&tmp];

73. Types of NSTableView

      Cell based and View based. In view based we can put multiple objects.

74. When to use Bindings.

75.

76. Mac OS Releases.

Version 10.0: "Cheetah"

Version 10.1: "Puma"

Version 10.2: "Jaguar"

Version 10.3: "Panther"

Version 10.4: "Tiger"

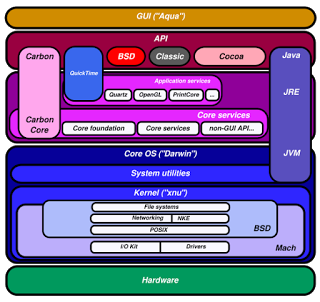
Version 10.5: "Leopard"

Version 10.6: "Snow Leopard"

Version 10.7: "Lion"

Version 10.8: "Mountain Lion"  
Version 10.9: "Mavericks"  
Version 10.10: "Yosemite".

77. Structure of MacOS.



78. Web Services

**What are Web Services?**

∙         Web services are application components

∙         Web services communicate using open protocols

∙         Web services are self-contained and self-describing

∙         Web services can be discovered using UDDI

∙         Web services can be used by other applications

∙         XML is the basis for Web services

**How Does it Work?**

The basic Web services platform is XML + HTTP.

XML provides a language which can be used between different platforms and programming languages and still express complex messages and functions.

The HTTP protocol is the most used Internet protocol.

Web services platform elements:

∙         SOAP (Simple Object Access Protocol)

∙         UDDI (Universal Description, Discovery and Integration)

∙         WSDL (Web Services Description Language)

In cocoa, NSURLConnection and NSURLRequest are used for this.

79. Abstract class in cocoa.

Cocoa doesn’t provide anything called abstract.  We can create a class abstract which gets check only at runtime, compile time this is not checked.

@interface AbstractClass : NSObject

@end

@implementation AbstractClass

+ (id)alloc{

    if (self == [AbstractClass class]) {

        NSLog(@"Abstract Class cant be used");

    }

    return [super alloc];

@end

80. xml parsers.

NSXML is SAX parser.

SAX parser is one where your code is notified as the parser walks through the XML tree, and you are responsible for keeping track of state and constructing any objects you might want to keep track of the data as the parser marches through.

A DOM parser reads the entire document and builds up an in-memory representation that you can query for different elements. Often, you can even construct XPath queries to pull out particular pieces.  
  
**SAX parser is advantageous than using DOM parser.**  
-The input document is too big for available memory (actually in this case SAX is your only choice)  
-You can process the document in small contiguous chunks of input. You do not need the entire document before you can do useful work  
-You just want to use the parser to extract the information of interest, and all your computation will be completely based on the data structures created by yourself. Actually in most of our applications, we create data structures of our own which are usually not as complicated as the DOM tree. From this sense, I think, the chance of using a DOM parser is less than that of using a SAX parser.  
  
**DOM parser is advantageous than using SAX parser.**  
-Your application needs to access widely separately parts of the document at the same time.  
-Your application may probably use a internal data structure which is almost as complicated as the document itself.  
-Your application has to modify the document repeatedly.  
  
-Your application has to store the document for a significant amount of time through many method calls.

81. What is syncronous and asynchronous call?  
A synchronous process is invoked by a request/response operation, and the result of the process is returned to the caller immediately via this operation.

An asynchronous process is invoked by a one-way operation and the result and any faults are returned by invoking other one-way operations. The process result is returned to the caller via a callback operation.

For example, you can think of a synchronous process as a telephone, and an asynchronous process as the postal system. When you are having a conversation on the phone, you send and receive messages instantaneously using the same connection. If you were to send the same message in a letter via the postal service, it would be delivered in one manner, and its response returned in another.

82.  How you send notification to other applications / observe them?  
        [[NSWorkspace shardWorkspace] notificationCenter].  
        Distributed Objects.  
        NSInvocation.

*~~83. How u decide background or foreground application?~~*

*~~84. How u do unit testing for your code and gui.~~*

85. Where are objects and variables stored in Objective-C (Heap or Stack)?

In Objective-C, objects are usually created on the heap:

NSObject \*obj = [[NSObject alloc] init];

The storage for the obj variable itself is on the stack, but the object it points to is in the heap. The [NSObject alloc] call allocates a chunk of heap memory, and fills it out to match the layout needed for an NSObject.

A stack object is just an object where the memory for that object is allocated on the stack. Objective-C doesn't have any support for this directly

Objective-C supports stack objects only for Blocks.

*~~86. Sandboxing~~*

87. NSURLConnection class, types and how to use.

There are two ways of using the NSURLConnection class. One is asynchronous, and the other is synchronous. An asynchronous connection will create a new thread and does its downloading process on the new thread. A synchronous connection will block the *calling thread* while downloading content and doing its communication.   
Many developers think that a synchronous connection blocks the *main thread,* but that is incorrect. A synchronous connection will always block the thread from which it is fired. If you fire a synchronous connection from the main thread, yes, the main thread will be blocked. But if you fire a synchronous connection from a thread other than the main thread, it will be like an asynchronous connection in that it won’t block your main thread. In fact, the only difference between a synchronous and an asynchronous con‐ nection is that the runtime will create a thread for the asynchronous connection, while it won’t do the same for a synchronous connection.   
In order to create an asynchronous connection, we need to do the following:

* Have our URL in an instance of NSString.
* Convert our string to an instance of NSURL.
* Place our URL in a URL Request of type NSURLRequest, or in the case of mutable URLs, in an instance of NSMutableURLRequest.
* Create an instance of NSURLConnection and pass the URL request to it.

88. Cocoa class naming conventions.

89. Difference between HTTP and HTTPS.

∙         HTTP stands for [HyperText](http://www.blogger.com/blogger.g?blogID=2378569646178591916) Transfer [Protocol](http://www.blogger.com/blogger.g?blogID=2378569646178591916), whereas, HTTPS is HyperText Transfer Protocol [Secure](http://www.blogger.com/blogger.g?blogID=2378569646178591916).

∙         HTTP transmits everything as plan text, while HTTPS provides encrypted communication, so that only the recipient can decrypt and read the information.  Basically, HTTPS is a combination of HTTP and [SSL](http://www.blogger.com/blogger.g?blogID=2378569646178591916) (Secure Sockets Layer). This SSL is that protocol which encrypts the data.

∙         HTTP is fast and cheap, where HTTPS is slow and expensive.

As, HTTPS is safe it’s widely used during payment transactions or any sensitive transactions over the internet. On the other hand, HTTP is used most of the sites over the net, even this blogspot sites also use HTTP.

∙         HTTP URLs starts with “http:// “ and use [port](http://www.blogger.com/blogger.g?blogID=2378569646178591916) 80 by default, while HTTPS URLs stars with “https:// “ and use port 443.

∙         HTTP is unsafe from attacks like [man-in-the-middle](http://adf.ly/1974254/http:/en.wikipedia.org/wiki/Man-in-the-middle_attack) and eavesdropping, but HTTPS is secure from these sorts of attacks.

90.GCD

Grand Central Dispatch is not just a new abstraction around what we've already been using, it's an entire new underlying mechanism that makes multithreading easier and makes it easy to be as concurrent as your code can be without worrying about the variables like how much work your CPU cores are doing, how many CPU cores you have and how much threads you should spawn in response. You just use the Grand Central Dispatch API's and it handles the work of doing the appropriate amount of work. This is also not just in Cocoa, anything running on Mac OS X 10.6 Snow Leopard can take advantage of Grand Central Dispatch ( libdispatch ) because it's included in libSystem.dylib and all you need to do is include #import <dispatch/dispatch.h> in your app and you'll be able to take advantage of Grand Central Dispatch.

91.  How you attain the backward compatibility?

* Set the Base SDK to Current version of Mac (ex. 10.8)
* Set the Deployment SDK to older version (ex.1.6)

92. Call Back.

Synchronous operations are ones that happen in step with your calling code. Most of Cocoa works this way: you send a message to an object, say to format a string, etc, and by the time that line of code is "done", the operation is complete.

But in the real world, some operations take longer than "instantaneous" (some intensive graphics work, but mainly high or variably latency things like disk I/O or worse, network connectivity). These operations are unpredictable, and if the code were to block until finish, it might block indefinitely or forever, and that's no good.

So the way we handle this is to set up "callbacks"-- you say "go off and do this operation, and when you're done, call this other function". Then inside that "callback" function, you start the second operation that depends on the first. In this way, you're not spinning in circles waiting, you just get called "asynchronously" when each task is done.  
  
93. When to use Blocks?  
Blocks are first-class functions, which is a fancy way of saying that Blocks are regular Objective-C objects. Since they’re objects, they can be passed as parameters, returned from methods and functions, and assigned to variables. Blocks are called closures in other languages such as Python, Ruby and Lisp, because they encapsulate state when they are declared. A block creates a const copy of any local variable that is referenced inside of its scope. Before blocks, whenever you wanted to call some code and have it call you back later, you would typically use delegates or NSNotificationCenter. That worked fine, except it spreads your code all over – you start a task in one spot, and handle the result in another.

94.    Does Objective-C contain private methods?  
NO.

There is nothing called a private method in Obj-C. If a method is defined in .m only then it becomes protected. If in .h it is public.

If you really want a private method then you need to add a local category/ unnamed category/ class extension on the class and add the method in the category and define the method in the class.m  
  
95. What is new with XCode4.4+  
The compiler LLVM/clang 3.2  
literals for arrays, dictionary.  
auto-synthesize for @property. @synthisize prop=\_prop;

96. NSThread vs GCD

Migrating Away from Threads

The idea is that you eliminate work on your part, since the paradigm fits MOST code more easily.

* It reduces the memory penalty your application pays for storing thread stacks in the application’s memory space.
* It eliminates the code needed to create and configure your threads.
* It eliminates the code needed to manage and schedule work on threads.
* It simplifies the code you have to write.

Empirically, using GCD-type locking instead of @synchronized is about 80% faster or more, though micro-benchmarks may be deceiving. Read more [here](http://www.fieryrobot.com/blog/2010/09/01/synchronization-using-grand-central-dispatch/comment-page-1/#comment-221262), though I think the advice to go async with writes does not apply in many cases, and it's slower (but it's asynchronous).

**Advantages of Threads**

Why would you continue to use Threads? From the same document:

It is important to remember that queues are not a panacea for replacing threads. The asynchronous programming model offered by queues is appropriate in situations where latency is not an issue. Even though queues offer ways to configure the execution priority of tasks in the queue, higher execution priorities do not guarantee the execution of tasks at specific times. Therefore, threads are still a more appropriate choice in cases where you need minimal latency, such as in audio and video playback.

Another place where I haven't personally found an ideal solution using queues is daemon processes that need to be constantly rescheduled. Not that you cannot reschedule them, but looping within a NSThread method is simpler (I think). **Edit:** Now I'm convinced that even in this context, GCD-style locking would be faster, and you could also do a loop within a GCD-dispatched operation.

97. What is Bundle Identifiers.

A **bundle ID** precisely identifies a single app. A bundle ID is used during the development process to provision devices and by the operating system when the app is distributed to customers. For example, Game Center and In-App Purchase use a bundle ID to identify your app when using these services. The preferences system uses this string to identify the app for which a given preference applies. Similarly, Launch Services uses the bundle ID to locate an app capable of opening a particular file, using the first app it finds with the given identifier. The bundle ID is also used to validate an app’s signature.

The bundle ID string must be a uniform type identifier (UTI) that contains only alphanumeric characters (A-Z,a-z,0-9), hyphen (-), and period (.). The string should be in reverse-DNS format. For example, if your company’s domain is Ajax.com and you create an app named Hello, you could assign the string com.Ajax.Hello as your app’s bundle ID.

98. What are differences between Library and Frameworks.  
A static library is actually compiled as part of your app, whereas a framework is distributed with your app.

Basically, frameworks ARE libraries and provide a handy mechanism for working with them. If you look "inside" a framework, it's just a directory containing a static library and header files (in some folder structure with metadata).

If you want to create your own framework, you have to create a "static library" and pack it in a specific way.

---

**Static library** - a unit of code linked at compile time, which does not change.

However, iOS static libraries are **not** allowed to contain images/assets (only code). You can get around this challenge by using a **media bundle** though.

**Dynamic library** - a unit of code and/or assets linked at runtime that *may* change.

However, only Apple is allowed to create dynamic libraries for iOS , for OSX you can create your own.

**Software Framework** - a compiled set of code that accomplishes a task... hence, you can actually have a **static framework** or a **dynamic framework**, which are typically just the compiled versions of the above.

99. Difference between JSON and XML.  
JSON is so much more lightweight and less verbose*.*

|  |  |
| --- | --- |
| Q. [Compile Java in Xcode 4](http://stackoverflow.com/questions/15946163/compile-java-in-xcode-4) | [Compile Java in Xcode 4](http://stackoverflow.com/questions/15946163/compile-java-in-xcode-4) |

|  |  |
| --- | --- |
|  | Is it possible to do that?  **YES this is possible.**  1) File > New > New Project > Other > External Build System 2) Give it some name and save it somewhere. 3) File > New > New File > Other > Empty 4) Give it a Java-friendly name. In my example, use "HelloWorld.java" 5) Copy the contents below into the file and save. 6) File > New > New File > Other > Empty 7) Save as "Makefile". 8) Copy the contents below into the file and save. 9) The "Run" (>) button should at least compile your Java now.  Now it gets tricky. You don't have to do the next part. You could just open a Terminal to your project directory and run "java HelloWorld" if you want. I strongly suggest that. You can also just type "make" and use Xcode purely as a text editor.  9) Project > Edit Scheme > Debug > Info 10) Executable > Other > type ^⌘g > type "/bin" > choose "sh" 11) Arguments 12) Set "Base Expansion on" whatever you named your target 13) Add an argument '-c "/usr/bin/java -classpath $(PROJECT\_DIR) HelloWorld"' for this example. 14) Cross your fingers 15) Click the "Run" (>) button.  *An Extract from* [Apple Support Community](https://discussions.apple.com/thread/3651347?start=0&tstart=0) |

1.What is latest iOS version?

IOS - 6.1.3 (updated on 5/15/13 3:15 AM  
Pacific Daylight Time)

2.What is latest Xcode version?

Xcode- 4.6.2 (updated on 5/15/13 3:15 AM

Pacific Daylight Time)

3.What is latest mac os version?

 Mac- Mountain Lion (updated on 5/15/13 3:15 AM

Pacific Daylight Time)

4.What is iPad screen size?

1024X768

5.what is iPhone screen size?

320X480

6.What are the features is IOS 6?

1.Map :beautifully designed from the ground up (and the sky down)

2.Integration of Facebook with iOS

3.shared photo streams.

4.Passbook - boarding passes, loyalty cards, retail coupons, cinema tickets and more all in one place

5.Facetime - on mobile network as wifi

6.changed Phone app - \*remind me later,\*reply with message.

7.Mail - redesigned more streamline interface.

8.Camera with panorama .

7.Who invented Objective c?

Broad cox and Tom Love

8.What is Cococa and cocoa touch?

Cocoa is for Mac App development  and cocoa touch is for apples touch devices - that provide all development environment

9.What is Objective c?

\*Objective-C is a reflective, object-oriented programming language which adds Smalltalk-style messaging to the C programming language. strictly superset of c.

10. how declare methods in Objective c? and how to call them?

 - (return\_type)methodName:(data\_type)parameter\_name : (data\_type)parameter\_name

11. What is property in Objective c?

Property allow declared variables with specification like atomic/nonatmic, or retain/assign

12.What is meaning of "copy" keyword?

copy object during assignment and increases retain count by 1

13.What is meaning of "readOnly" keyword?

 Declare read only object / declare only getter method

14.What is meaning of "retain" keyword?

Specifies that retain should be invoked on the object upon assignment. takes ownership of an object

15.What is meaning of "assign" keyword?

Specifies that the setter uses simple assignment. Uses on attribute of scalar type like float,int.

16.What is meaning of "atomic" keyword?

"atomic", the synthesized setter/getter will ensure that a whole value is always returned from the getter or set by the setter, only single thread can access variable to get or set value at a time

17.What is meaning of "nonatomic" keyword?

In non atomic no such guaranty that value is returned from variable is same that setter sets. at same time

18.What is difference between "assign" and "retain" keyword?

Retain -Specifies that retain should be invoked on the object upon assignment. takes ownership of an object

Assign - Specifies that the setter uses simple assignment. Uses on attribute of scalar type like float,int.

19.What is meaning of "synthesize" keyword ?

ask the compiler to generate the setter and getter  methods according to the specification in the declaration

20.What is "Protocol" on objective c?

A protocol declares methods that can be implemented by any class. Protocols are not classes themselves. They simply define an interface that other objects are responsible for implementing. Protocols have many advantages. The idea is to provide a way for classes to share the same method and property declarations without inheriting them from a common ancestor

21.What is use of UIApplication class?

The UIApplication class implements the required behavior of an application.

22.What compilers apple using ?

The Apple compilers are based on the compilers of the GNU Compiler Collection.

23.What is synchronized() block in objective c? what is the use of that?

The @synchronized()directive locks a section of code for use by a single thread. Other threads are blocked until the thread exits the protected code.

24. What is the "interface" and "implementation"?

interface declares the behavior of class and implementation defines the behavior of class.

25.What is "private", "Protected" and "Public" ?

private - limits the scope class variable to the class that declares it.

protected - Limits instance variable scope to declaring and inheriting classes.

public - Removes restrictions on the scope of instance variables

26. What is the use of "dynamic" keyword?

Instructs the compiler not to generate a warning if it cannot find implementations of accessor methods associated with the properties whose names follow.

27.What is "Delegate" ?

A delegate is an object that will respond to pre-chosen selectors (function calls) at some point in the future., need to implement the protocol method by the delegate object.

28.What is "notification"?

provides a mechanism for broadcasting information within a program, using notification we can send message to other object by adding observer .

29.What is difference between "protocol" and "delegate"?

protocol is used the declare a set of methods that a class that "adopts" (declares that it will use this protocol) will implement.

Delegates are a use of the language feature of protocols. The delegation design pattern is a way of designing your code to use protocols where necessary.

30.What is "Push Notification"?

to get the any update /alert from server .

31.How to deal with SQLite database?

Dealing with sqlite database in iOS:

1. Create database : sqlite3 AnimalDatabase.sql

2.Create table and insert data in to  table :

CREATE TABLE animals ( id INTEGER PRIMARY KEY, name VARCHAR(50), description TEXT, image VARCHAR(255) );

INSERT INTO animals (name, description, image) VALUES ('Elephant', 'The elephant is a very large animal that lives in Africa and Asia', 'http://dblog.com.au/wp-content/elephant.jpg');

3. Create new app --> Add SQLite framework and database file to project

4. Read the database and close it once work done with database :

// Setup the database object

sqlite3 \*database;

// Init the animals Array

animals = [[NSMutableArray alloc] init];

// Open the database from the users filessytem

if(sqlite3\_open([databasePath UTF8String], &database) == SQLITE\_OK) {

// Setup the SQL Statement and compile it for faster access

const char \*sqlStatement = "select \* from animals";

sqlite3\_stmt \*compiledStatement;

if(sqlite3\_prepare\_v2(database, sqlStatement, -1, &compiledStatement, NULL) == SQLITE\_OK) {

// Loop through the results and add them to the feeds array

while(sqlite3\_step(compiledStatement) == SQLITE\_ROW) {

// Read the data from the result row

NSString \*aName = [NSString stringWithUTF8String:(char \*)sqlite3\_column\_text(compiledStatement, 1)];

NSString \*aDescription = [NSString stringWithUTF8String:(char \*)sqlite3\_column\_text(compiledStatement, 2)];

NSString \*aImageUrl = [NSString stringWithUTF8String:(char \*)sqlite3\_column\_text(compiledStatement, 3)];

// Create a new animal object with the data from the database

Animal \*animal = [[Animal alloc] initWithName:aName description:aDescription url:aImageUrl];

// Add the animal object to the animals Array

[animals addObject:animal];

[animal release];

}

}

// Release the compiled statement from memory

sqlite3\_finalize(compiledStatement);

}

sqlite3\_close(database);

32.What is storyboard?

With Storyboards, all screens are stored in a single file. This gives you a conceptual overview of the visual representation for the app and shows you how the screens are connected. Xcode provides a built-in editor to layout the Storyboards.

* .storyboard is essentially one single file for all your screens in the app and it shows the flow of the screens. You can add segues/transitions between screens, this way. So, this minimizes the boilerplate  code required to manage multiple screens.
* 2.   Minimizes the overall no. of files in an app.

33.What is Category in Objective c?

A category allows you to add methods to an existing class—even to one for which you do not have the source.

34.What is block in objective c?

Blocks are a language-level feature added to C, Objective-C and C++, which allow you to create distinct segments of code that can be passed around to methods or functions as if they were values. Blocks are Objective-C objects, which means they can be added to collections like NSArray or NSDictionary. They also have the ability to capture values from the enclosing scope, making them similar to closures or lambdas in other programming languages.

35. How to parse xml? explain in deep.

Using NSXMLParser.

Create xml parser object with xml data, set its delegate , and call the parse method with parserObject.

Delegate methods getting called :

[– parserDidStartDocument:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parserDidStartDocument:)

[– parserDidEndDocument:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parserDidEndDocument:)

[– parser:didStartElement:namespaceURI:qualifiedName:attributes:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:didStartElement:namespaceURI:qualifiedName:attributes:)

[– parser:didEndElement:namespaceURI:qualifiedName:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:didEndElement:namespaceURI:qualifiedName:)

[– parser:didStartMappingPrefix:toURI:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:didStartMappingPrefix:toURI:)

[– parser:didEndMappingPrefix:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:didEndMappingPrefix:)

[– parser:resolveExternalEntityName:systemID:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:resolveExternalEntityName:systemID:)

[– parser:parseErrorOccurred:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:parseErrorOccurred:)

[– parser:validationErrorOccurred:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:validationErrorOccurred:)

[– parser:foundCharacters:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:foundCharacters:)

[– parser:foundIgnorableWhitespace:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:foundIgnorableWhitespace:)

[– parser:foundProcessingInstructionWithTarget:data:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:foundProcessingInstructionWithTarget:data:)

[– parser:foundComment:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:foundComment:)

[– parser:foundCDATA:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:foundCDATA:)

36.How to parse JSON? explain in deep.

By using NSJSONSerialization.

For example : NSArray \*jsonArray = [NSJSONSerialization JSONObjectWithData: data options: NSJSONReadingMutableContainers error: &e];

37.How to use reusable cell in UITableview?

By using dequeReusableCellWithIdentifier

38.What is the meaning of "strong"keyword?

\*strong -o "own" the object you are referencing with this property/variable. The compiler will take care that any object that you assign to this property will not be destroyed as long as you (or any other object) points to it with a strong reference.

39.What is the meaning of "weak" keyword?

\*Weak - weak reference you signify that you don't want to have control over the object's lifetime. The object you are referencing weakly only lives on because at least one other object holds a strong reference to it. Once that is no longer the case, the object gets destroyed and your weak property will automatically get set to nil.

40.What is difference strong and  weak reference ? explain.

complier with be responsible for lifetime of object which is declared as strong. for weak object - compiler will destroy object once strong reference that hold weak object get destroyed.

41.What is ARC ? How it works? explain in deep.

Automatic reference counting (ARC)  If the compiler can recognize where you should be retaining and releasing objects, and put the retain and release statement in code.

42. What manual memory management ?  how it work?

In Manual memory management  developers is responsible for life cycle of object. developer has to retain /alloc and release the object wherever needed.

43. How to find the memory leaks in MRC?

By using -

1.  Static analyzer.

2. Instrument

44.what is use of NSOperation? how NSOperationque works?

An operation object is a single-shot object—that is, it executes its task once and cannot be used to execute it again. You typically execute operations by adding them to an operation queueAn NSOperationQueue object is a queue that handles objects of the NSOperation class type. An NSOperation object, simply phrased, represents a single task, including both the data and the code related to the task. The NSOperationQueue handles and manages the execution of all the NSOperation objects (the tasks) that have been added to it.

45.How to send crash report from device?

46.What is autorealease pool?

Every time -autorelease is sent to an object, it is added to the inner-most autorelease pool. When the pool is drained, it simply sends -release to all the objects in the pool.

Autorelease pools are simply a convenience that allows you to defer sending -release until "later". That "later" can happen in several places, but the most common in Cocoa GUI apps is at the end of the current run loop cycle.

47.What happens when we invoke a method on a nil pointer?

48.Difference between nil and Nil.

Nil is meant for class pointers, and nil is meant for object pointers

49.What is fast enumeration?

for(id object in objets){

}

50. How to start a thread?

- (void)performSelectorInBackground:(SEL)aSelector withObject:(id)arg on NSObject

  NSThread\* evtThread = [ [NSThread alloc] initWithTarget:self

                            selector:@selector( saySomething )

                          object:nil ];

    [ evtThread start ];

51.How to download something from the internet?

By Using NSURLConnection , by starting connection or sending synchronous request.

52.what is synchronous web request and asynchronous ?

In  synchronous request main thread gets block and control will not get back to user till that request gets execute.

In Asynchronous control gets back to user even if request is getting execute.

53. Difference between sax parser and dom parser ?

SAX (Simple API for XML)

* Parses node by node
* Doesn't store the XML in memory
* We can not insert or delete a node
* Top to bottom traversing

DOM (Document Object Model)

* Stores the entire XML document into memory before processing
* Occupies more memory
* We can insert or delete nodes
* Traverse in any direction

54.Explain stack and heap?

55.What are the ViewController  lifecycle in ios?

loadView - viewDidLoad-viewWillAppear-viewDidAppear - viewDisappear  - viewDidUnload

56.Difference between coredata & sqlite?

There is a huge difference between these two. SQLLite is a database itself like we have MS SQL Server. But CoreData is an ORM (Object Relational Model) which creates a layer between the database and the UI. It speeds-up the process of interaction as we dont have to write queries, just work with the ORM and let ORM handles the backend. For save or retrieval of large data, I recommend to use Core Data because of its abilities to handle the less processing speed of IPhone.

57.Steps for using coredata?

NSFetchedResultsController - It is designed primarily to function as a data source for a UITableView

58.Procedure to push the app in AppStore?

59.What are the Application lifecycle in ios?

ApplicationDidFinishLaunchingWithOption -ApplicationWillResignActive- ApplicationDidBecomeActive-ApplicationWillTerminate

60.Difference between release and autorelease ?

release - destroy the object from memory,

autorelease - destroy the object from memory in future when it is not in use.

61.How to start a selector on a background thread

- (void)performSelectorInBackground:(SEL)aSelector withObject:(id)arg on NSObject

62.What happens if the methods doesn’t exist

App will crash with exception unrecognized selector sent to instance.

63. How Push notification works?

Server - Apple server - device by using APNs

Delegate methods :

UITableView:

DataSource -

Configuring a Table View

[– tableView:cellForRowAtIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDataSource_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDataSource/tableView:cellForRowAtIndexPath:)  required method

[– numberOfSectionsInTableView:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDataSource_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDataSource/numberOfSectionsInTableView:)

[– tableView:numberOfRowsInSection:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDataSource_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDataSource/tableView:numberOfRowsInSection:)  required method

[– sectionIndexTitlesForTableView:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDataSource_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDataSource/sectionIndexTitlesForTableView:)

[– tableView:sectionForSectionIndexTitle:atIndex:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDataSource_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDataSource/tableView:sectionForSectionIndexTitle:atIndex:)

[– tableView:titleForHeaderInSection:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDataSource_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDataSource/tableView:titleForHeaderInSection:)

[– tableView:titleForFooterInSection:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDataSource_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDataSource/tableView:titleForFooterInSection:)

Inserting or Deleting Table Rows

[– tableView:commitEditingStyle:forRowAtIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDataSource_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDataSource/tableView:commitEditingStyle:forRowAtIndexPath:)

[– tableView:canEditRowAtIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDataSource_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDataSource/tableView:canEditRowAtIndexPath:)

Reordering Table Rows

[– tableView:canMoveRowAtIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDataSource_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDataSource/tableView:canMoveRowAtIndexPath:)

[– tableView:moveRowAtIndexPath:toIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDataSource_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDataSource/tableView:moveRowAtIndexPath:toIndexPath:)

  Delegate -

Configuring Rows for the Table View

[– tableView:heightForRowAtIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:heightForRowAtIndexPath:)

[– tableView:indentationLevelForRowAtIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:indentationLevelForRowAtIndexPath:)

[– tableView:willDisplayCell:forRowAtIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:willDisplayCell:forRowAtIndexPath:)

Managing Accessory Views

[– tableView:accessoryButtonTappedForRowWithIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:accessoryButtonTappedForRowWithIndexPath:)

Managing Selections

[– tableView:willSelectRowAtIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:willSelectRowAtIndexPath:)

[– tableView:didSelectRowAtIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:didSelectRowAtIndexPath:)

[– tableView:willDeselectRowAtIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:willDeselectRowAtIndexPath:)

[– tableView:didDeselectRowAtIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:didDeselectRowAtIndexPath:)

Modifying the Header and Footer of Sections

[– tableView:viewForHeaderInSection:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:viewForHeaderInSection:)

[– tableView:viewForFooterInSection:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:viewForFooterInSection:)

[– tableView:heightForHeaderInSection:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:heightForHeaderInSection:)

[– tableView:heightForFooterInSection:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:heightForFooterInSection:)

Editing Table Rows

[– tableView:willBeginEditingRowAtIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:willBeginEditingRowAtIndexPath:)

[– tableView:didEndEditingRowAtIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:didEndEditingRowAtIndexPath:)

[– tableView:editingStyleForRowAtIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:editingStyleForRowAtIndexPath:)

[– tableView:titleForDeleteConfirmationButtonForRowAtIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:titleForDeleteConfirmationButtonForRowAtIndexPath:)

[– tableView:shouldIndentWhileEditingRowAtIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:shouldIndentWhileEditingRowAtIndexPath:)

Reordering Table Rows

[– tableView:targetIndexPathForMoveFromRowAtIndexPath:toProposedIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:targetIndexPathForMoveFromRowAtIndexPath:toProposedIndexPath:)

Copying and Pasting Row Content

[– tableView:shouldShowMenuForRowAtIndexPath:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:shouldShowMenuForRowAtIndexPath:)

[– tableView:canPerformAction:forRowAtIndexPath:withSender:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:canPerformAction:forRowAtIndexPath:withSender:)

[– tableView:performAction:forRowAtIndexPath:withSender:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UITableViewDelegate/tableView:performAction:forRowAtIndexPath:withSender:)

 UIPickerView-

 DataSource -

Providing Counts for the Picker View

[– numberOfComponentsInPickerView:](http://developer.apple.com/library/ios/documentation/iPhone/Reference/UIPickerViewDataSource_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UIPickerViewDataSource/numberOfComponentsInPickerView:)

[– pickerView:numberOfRowsInComponent:](http://developer.apple.com/library/ios/documentation/iPhone/Reference/UIPickerViewDataSource_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/UIPickerViewDataSource/pickerView:numberOfRowsInComponent:)

  Delegate -

Setting the Dimensions of the Picker View

[– pickerView:rowHeightForComponent:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UIPickerViewDelegate_Protocol/Reference/UIPickerViewDelegate.html#//apple_ref/occ/intfm/UIPickerViewDelegate/pickerView:rowHeightForComponent:)

[– pickerView:widthForComponent:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UIPickerViewDelegate_Protocol/Reference/UIPickerViewDelegate.html#//apple_ref/occ/intfm/UIPickerViewDelegate/pickerView:widthForComponent:)

Setting the Content of Component Rows

The methods in this group are marked @optional. However, to use a picker view, you must implement either the[pickerView:titleForRow:forComponent:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UIPickerViewDelegate_Protocol/Reference/UIPickerViewDelegate.html#//apple_ref/occ/intfm/UIPickerViewDelegate/pickerView:titleForRow:forComponent:) or the [pickerView:viewForRow:forComponent:reusingView:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UIPickerViewDelegate_Protocol/Reference/UIPickerViewDelegate.html#//apple_ref/occ/intfm/UIPickerViewDelegate/pickerView:viewForRow:forComponent:reusingView:) method to provide the content of component rows.

[– pickerView:titleForRow:forComponent:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UIPickerViewDelegate_Protocol/Reference/UIPickerViewDelegate.html#//apple_ref/occ/intfm/UIPickerViewDelegate/pickerView:titleForRow:forComponent:)

[– pickerView:viewForRow:forComponent:reusingView:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UIPickerViewDelegate_Protocol/Reference/UIPickerViewDelegate.html#//apple_ref/occ/intfm/UIPickerViewDelegate/pickerView:viewForRow:forComponent:reusingView:)

Responding to Row Selection

[– pickerView:didSelectRow:inComponent:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UIPickerViewDelegate_Protocol/Reference/UIPickerViewDelegate.html#//apple_ref/occ/intfm/UIPickerViewDelegate/pickerView:didSelectRow:inComponent:)

UITextFeild-

Delegate -

Managing Editing

[– textFieldShouldBeginEditing:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextFieldDelegate_Protocol/UITextFieldDelegate/UITextFieldDelegate.html#//apple_ref/occ/intfm/UITextFieldDelegate/textFieldShouldBeginEditing:)

[– textFieldDidBeginEditing:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextFieldDelegate_Protocol/UITextFieldDelegate/UITextFieldDelegate.html#//apple_ref/occ/intfm/UITextFieldDelegate/textFieldDidBeginEditing:)

[– textFieldShouldEndEditing:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextFieldDelegate_Protocol/UITextFieldDelegate/UITextFieldDelegate.html#//apple_ref/occ/intfm/UITextFieldDelegate/textFieldShouldEndEditing:)

[– textFieldDidEndEditing:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextFieldDelegate_Protocol/UITextFieldDelegate/UITextFieldDelegate.html#//apple_ref/occ/intfm/UITextFieldDelegate/textFieldDidEndEditing:)

Editing the Text Field’s Text

[– textField:shouldChangeCharactersInRange:replacementString:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextFieldDelegate_Protocol/UITextFieldDelegate/UITextFieldDelegate.html#//apple_ref/occ/intfm/UITextFieldDelegate/textField:shouldChangeCharactersInRange:replacementString:)

[– textFieldShouldClear:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextFieldDelegate_Protocol/UITextFieldDelegate/UITextFieldDelegate.html#//apple_ref/occ/intfm/UITextFieldDelegate/textFieldShouldClear:)

[– textFieldShouldReturn:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextFieldDelegate_Protocol/UITextFieldDelegate/UITextFieldDelegate.html#//apple_ref/occ/intfm/UITextFieldDelegate/textFieldShouldReturn:)

 UItextView-

 Delegate - Responding to Editing Notifications

[– textViewShouldBeginEditing:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextViewDelegate_Protocol/Reference/UITextViewDelegate.html#//apple_ref/occ/intfm/UITextViewDelegate/textViewShouldBeginEditing:)

[– textViewDidBeginEditing:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextViewDelegate_Protocol/Reference/UITextViewDelegate.html#//apple_ref/occ/intfm/UITextViewDelegate/textViewDidBeginEditing:)

[– textViewShouldEndEditing:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextViewDelegate_Protocol/Reference/UITextViewDelegate.html#//apple_ref/occ/intfm/UITextViewDelegate/textViewShouldEndEditing:)

[– textViewDidEndEditing:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextViewDelegate_Protocol/Reference/UITextViewDelegate.html#//apple_ref/occ/intfm/UITextViewDelegate/textViewDidEndEditing:)

Responding to Text Changes

[– textView:shouldChangeTextInRange:replacementText:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextViewDelegate_Protocol/Reference/UITextViewDelegate.html#//apple_ref/occ/intfm/UITextViewDelegate/textView:shouldChangeTextInRange:replacementText:)

[– textViewDidChange:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextViewDelegate_Protocol/Reference/UITextViewDelegate.html#//apple_ref/occ/intfm/UITextViewDelegate/textViewDidChange:)

Responding to Selection Changes

[– textViewDidChangeSelection:](http://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextViewDelegate_Protocol/Reference/UITextViewDelegate.html#//apple_ref/occ/intfm/UITextViewDelegate/textViewDidChangeSelection:)

 MKMapView-

Delegate -

Responding to Map Position Changes

[– mapView:regionWillChangeAnimated:](http://developer.apple.com/library/ios/documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html#//apple_ref/occ/intfm/MKMapViewDelegate/mapView:regionWillChangeAnimated:)

[– mapView:regionDidChangeAnimated:](http://developer.apple.com/library/ios/documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html#//apple_ref/occ/intfm/MKMapViewDelegate/mapView:regionDidChangeAnimated:)

Loading the Map Data

[– mapViewWillStartLoadingMap:](http://developer.apple.com/library/ios/documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html#//apple_ref/occ/intfm/MKMapViewDelegate/mapViewWillStartLoadingMap:)

[– mapViewDidFinishLoadingMap:](http://developer.apple.com/library/ios/documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html#//apple_ref/occ/intfm/MKMapViewDelegate/mapViewDidFinishLoadingMap:)

[– mapViewDidFailLoadingMap:withError:](http://developer.apple.com/library/ios/documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html#//apple_ref/occ/intfm/MKMapViewDelegate/mapViewDidFailLoadingMap:withError:)

Tracking the User Location

[– mapViewWillStartLocatingUser:](http://developer.apple.com/library/ios/documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html#//apple_ref/occ/intfm/MKMapViewDelegate/mapViewWillStartLocatingUser:)

[– mapViewDidStopLocatingUser:](http://developer.apple.com/library/ios/documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html#//apple_ref/occ/intfm/MKMapViewDelegate/mapViewDidStopLocatingUser:)

[– mapView:didUpdateUserLocation:](http://developer.apple.com/library/ios/documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html#//apple_ref/occ/intfm/MKMapViewDelegate/mapView:didUpdateUserLocation:)

[– mapView:didFailToLocateUserWithError:](http://developer.apple.com/library/ios/documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html#//apple_ref/occ/intfm/MKMapViewDelegate/mapView:didFailToLocateUserWithError:)

[– mapView:didChangeUserTrackingMode:animated:](http://developer.apple.com/library/ios/documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html#//apple_ref/occ/intfm/MKMapViewDelegate/mapView:didChangeUserTrackingMode:animated:)  required method

Managing Annotation Views

[– mapView:viewForAnnotation:](http://developer.apple.com/library/ios/documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html#//apple_ref/occ/intfm/MKMapViewDelegate/mapView:viewForAnnotation:)

[– mapView:didAddAnnotationViews:](http://developer.apple.com/library/ios/documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html#//apple_ref/occ/intfm/MKMapViewDelegate/mapView:didAddAnnotationViews:)

[– mapView:annotationView:calloutAccessoryControlTapped:](http://developer.apple.com/library/ios/documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html#//apple_ref/occ/intfm/MKMapViewDelegate/mapView:annotationView:calloutAccessoryControlTapped:)

Dragging an Annotation View

[– mapView:annotationView:didChangeDragState:fromOldState:](http://developer.apple.com/library/ios/documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html#//apple_ref/occ/intfm/MKMapViewDelegate/mapView:annotationView:didChangeDragState:fromOldState:)

Selecting Annotation Views

[– mapView:didSelectAnnotationView:](http://developer.apple.com/library/ios/documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html#//apple_ref/occ/intfm/MKMapViewDelegate/mapView:didSelectAnnotationView:)

[– mapView:didDeselectAnnotationView:](http://developer.apple.com/library/ios/documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html#//apple_ref/occ/intfm/MKMapViewDelegate/mapView:didDeselectAnnotationView:)

Managing Overlay Views

[– mapView:viewForOverlay:](http://developer.apple.com/library/ios/documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html#//apple_ref/occ/intfm/MKMapViewDelegate/mapView:viewForOverlay:)

[– mapView:didAddOverlayViews:](http://developer.apple.com/library/ios/documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html#//apple_ref/occ/intfm/MKMapViewDelegate/mapView:didAddOverlayViews:)

NSURLConnection-

Delegate -

Connection Authentication

[– connection:willSendRequestForAuthenticationChallenge:](http://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLConnectionDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSURLConnectionDelegate/connection:willSendRequestForAuthenticationChallenge:)

[– connection:canAuthenticateAgainstProtectionSpace:](http://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLConnectionDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSURLConnectionDelegate/connection:canAuthenticateAgainstProtectionSpace:)

[– connection:didCancelAuthenticationChallenge:](http://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLConnectionDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSURLConnectionDelegate/connection:didCancelAuthenticationChallenge:)

[– connection:didReceiveAuthenticationChallenge:](http://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLConnectionDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSURLConnectionDelegate/connection:didReceiveAuthenticationChallenge:)

[– connectionShouldUseCredentialStorage:](http://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLConnectionDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSURLConnectionDelegate/connectionShouldUseCredentialStorage:)

Connection Completion

[– connection:didFailWithError:](http://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLConnectionDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSURLConnectionDelegate/connection:didFailWithError:)

NSURLConnectionDownloadDelegate

[– connection:didWriteData:totalBytesWritten:expectedTotalBytes:](http://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLConnectionDownloadDelegate_Protocol/NSURLConnectionDownloadDelegate/NSURLConnectionDownloadDelegate.html#//apple_ref/occ/intfm/NSURLConnectionDownloadDelegate/connection:didWriteData:totalBytesWritten:expectedTotalBytes:)

[– connectionDidResumeDownloading:totalBytesWritten:expectedTotalBytes:](http://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLConnectionDownloadDelegate_Protocol/NSURLConnectionDownloadDelegate/NSURLConnectionDownloadDelegate.html#//apple_ref/occ/intfm/NSURLConnectionDownloadDelegate/connectionDidResumeDownloading:totalBytesWritten:expectedTotalBytes:)

[– connectionDidFinishDownloading:destinationURL:](http://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLConnectionDownloadDelegate_Protocol/NSURLConnectionDownloadDelegate/NSURLConnectionDownloadDelegate.html#//apple_ref/occ/intfm/NSURLConnectionDownloadDelegate/connectionDidFinishDownloading:destinationURL:)

 NSURLConnection

Preflighting a Request

[+ canHandleRequest:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSURLConnection_Class/Reference/Reference.html#//apple_ref/occ/clm/NSURLConnection/canHandleRequest:)

Loading Data Synchronously

[+ sendSynchronousRequest:returningResponse:error:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSURLConnection_Class/Reference/Reference.html#//apple_ref/occ/clm/NSURLConnection/sendSynchronousRequest:returningResponse:error:)

Loading Data Asynchronously

[+ connectionWithRequest:delegate:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSURLConnection_Class/Reference/Reference.html#//apple_ref/occ/clm/NSURLConnection/connectionWithRequest:delegate:)

[– initWithRequest:delegate:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSURLConnection_Class/Reference/Reference.html#//apple_ref/occ/instm/NSURLConnection/initWithRequest:delegate:)

[– initWithRequest:delegate:startImmediately:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSURLConnection_Class/Reference/Reference.html#//apple_ref/occ/instm/NSURLConnection/initWithRequest:delegate:startImmediately:)

[+ sendAsynchronousRequest:queue:completionHandler:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSURLConnection_Class/Reference/Reference.html#//apple_ref/occ/clm/NSURLConnection/sendAsynchronousRequest:queue:completionHandler:)

[– start](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSURLConnection_Class/Reference/Reference.html#//apple_ref/occ/instm/NSURLConnection/start)

Stopping a Connection

[– cancel](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSURLConnection_Class/Reference/Reference.html#//apple_ref/occ/instm/NSURLConnection/cancel)

Scheduling Delegate Messages

[– scheduleInRunLoop:forMode:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSURLConnection_Class/Reference/Reference.html#//apple_ref/occ/instm/NSURLConnection/scheduleInRunLoop:forMode:)

[– setDelegateQueue:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSURLConnection_Class/Reference/Reference.html#//apple_ref/occ/instm/NSURLConnection/setDelegateQueue:)

[– unscheduleFromRunLoop:forMode:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSURLConnection_Class/Reference/Reference.html#//apple_ref/occ/instm/NSURLConnection/unscheduleFromRunLoop:forMode:)

 NSXMLParser-

Handling XML

[– parserDidStartDocument:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parserDidStartDocument:)

[– parserDidEndDocument:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parserDidEndDocument:)

[– parser:didStartElement:namespaceURI:qualifiedName:attributes:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:didStartElement:namespaceURI:qualifiedName:attributes:)

[– parser:didEndElement:namespaceURI:qualifiedName:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:didEndElement:namespaceURI:qualifiedName:)

[– parser:didStartMappingPrefix:toURI:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:didStartMappingPrefix:toURI:)

[– parser:didEndMappingPrefix:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:didEndMappingPrefix:)

[– parser:resolveExternalEntityName:systemID:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:resolveExternalEntityName:systemID:)

[– parser:parseErrorOccurred:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:parseErrorOccurred:)

[– parser:validationErrorOccurred:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:validationErrorOccurred:)

[– parser:foundCharacters:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:foundCharacters:)

[– parser:foundIgnorableWhitespace:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:foundIgnorableWhitespace:)

[– parser:foundProcessingInstructionWithTarget:data:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:foundProcessingInstructionWithTarget:data:)

[– parser:foundComment:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:foundComment:)

[– parser:foundCDATA:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:foundCDATA:)

Handling the DTD

[– parser:foundAttributeDeclarationWithName:forElement:type:defaultValue:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:foundAttributeDeclarationWithName:forElement:type:defaultValue:)

[– parser:foundElementDeclarationWithName:model:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:foundElementDeclarationWithName:model:)

[– parser:foundExternalEntityDeclarationWithName:publicID:systemID:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:foundExternalEntityDeclarationWithName:publicID:systemID:)

[– parser:foundInternalEntityDeclarationWithName:value:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:foundInternalEntityDeclarationWithName:value:)

[– parser:foundUnparsedEntityDeclarationWithName:publicID:systemID:notationName:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:foundUnparsedEntityDeclarationWithName:publicID:systemID:notationName:)

[– parser:foundNotationDeclarationWithName:publicID:systemID:](http://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSXMLParserDelegate_Protocol/Reference/Reference.html#//apple_ref/occ/intfm/NSXMLParserDelegate/parser:foundNotationDeclarationWithName:publicID:systemID:)

7.NSURLConnection

Connection Authentication

* – connection:willSendRequestForAuthenticationChallenge:
* – connection:canAuthenticateAgainstProtectionSpace:
* – connection:didCancelAuthenticationChallenge:
* – connection:didReceiveAuthenticationChallenge:
* – connectionShouldUseCredentialStorage:

Connection Completion

* – connection:didFailWithError:

MethodGroup

* – connection:needNewBodyStream
* [– connection:didSendBodyData:totalBytesWritten:totalBytesExpectedToWrite:](https://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLConnectionDelegate_Protocol/DeprecationAppendix/AppendixADeprecatedAPI.html#//apple_ref/occ/intfm/NSURLConnectionDelegate/connection:didSendBodyData:totalBytesWritten:totalBytesExpectedToWrite:)  required method
* [– connection:didReceiveData:](https://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLConnectionDelegate_Protocol/DeprecationAppendix/AppendixADeprecatedAPI.html#//apple_ref/occ/instm/NSObject/connection:didReceiveData:)  required method
* [– connection:didReceiveResponse:](https://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLConnectionDelegate_Protocol/DeprecationAppendix/AppendixADeprecatedAPI.html#//apple_ref/occ/instm/NSObject/connection:didReceiveResponse:)  required method
* [– connection:willCacheResponse:](https://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLConnectionDelegate_Protocol/DeprecationAppendix/AppendixADeprecatedAPI.html#//apple_ref/occ/instm/NSObject/connection:willCacheResponse:)  required method
* [– connection:willSendRequest:redirectResponse:](https://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLConnectionDelegate_Protocol/DeprecationAppendix/AppendixADeprecatedAPI.html#//apple_ref/occ/instm/NSObject/connection:willSendRequest:redirectResponse:)  required method
* [– connectionDidFinishLoading:](https://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLConnectionDelegate_Protocol/DeprecationAppendix/AppendixADeprecatedAPI.html#//apple_ref/occ/instm/NSObject/connectionDidFinishLoading:)  required method