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

## ***Ans :*** By Subclassing the UIView class.

## **2-What's fast enumeration?**

## **Ans :**Fast enumeration is a language feature that allows you to numerate 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-What's a struct?**

## **Ans :**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?**

## **Ans :**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.**

## ***Ans :*** 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?**

## **Ans :**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?**

## ***Ans :*** No, the delegate is never retained! Ever!

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

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

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

## **Ans :** 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?**

## **Ans :**Yes, the object is retained. It creates a timer that calls a selector on the current thread's 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?**

## **Ans :**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?**

## **Ans :**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?**

## **Ans :**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**

## **Ans :**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?**

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

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

## **Ans :** 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.**

## **Ans :**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.**

## **Ans :** 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 self20-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) ?**

## **Ans :** 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.

## **69-Define property?**

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

## 

## **70-Why synthesized is used?**

## **Ans :** 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?**

## **Ans :**It is reference count for an object.

## **72- What is webservice?**

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

## **73-What is parsing?**

## **Ans :**To get data from web service we use parsing.

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

## **Ans :** “NSXML” Parser.

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

## **Ans :**“SAX” parser.

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

***Ans :*** (a)Dom is “documents based parser”.

b)SAX is a event driven parser

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

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

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

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

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

**Ans :**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?**

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

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

***Ans :*** 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?**

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

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

**Ans :**(1)Ui kit framework

(2)Map kit framework

(3)ADI kit framework

(4)Core data framework

(5)core foundation framework

**05- InApp purchase product type**

1. **Consumable :**products must be purchased each time the user needs that item. For example, one-time services are commonly implemented as consumable products.
2. **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.
3. **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.
4. **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.**

**Ans :** 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?**

**Ans :** 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?**

**Ans :**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.**

**Ans :** 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.**

**Ans :***Appkit (Application Kit)*

*Foundation*

**111- Objective-C.**

**Ans :** 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++.**

**Ans :***·* 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.**

**Ans :**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.

· Implemkent your own objects by subclassing NSView or other classes.

**114-Foundation Kit.**

**Ans :** *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.**

**Ans :**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**

**Ans :**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**

**Ans : ·**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**

**Ans :**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?**

**Ans :** *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***

***Ans :*** *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.***

***Ans :*** *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**

**Ans :***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**

**Ans :** *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**

**Ans :** *· 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**

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

**126- release vs pool drain**

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

**127- NSAutoReleasePool : release vs drain**

**Ans :** *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**

**Ans :***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**

**Ans :***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.**

**Ans :***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**

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

**132- Event Loop**

**Ans :** *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.**

**Ans :** *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;” ?**

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

**135-Collection**

**Ans :***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**

**Ans :** *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**

**Ans :** *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**

**Ans :** *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**

**Ans :***· 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**

**Ans :** *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**

**Ans :***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.*

**142-Cluster Class**

**Ans :** *Class clusters are a design pattern that the Foundation framework makes extensive use of. Class clusters group a number of private concrete subclasses under a public abstract superclass. The grouping of classes in this way simplifies the publicly visible architecture of an object-oriented framework without reducing its functional richness.*

**143-Differentiate Foundation vs Core Foundation**

**Ans :** *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**

**Ans :**

|  |  |
| --- | --- |
| ***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.**

**Ans :** *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.**

**Ans:** *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.**

**Ans :** *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.**

**Ans :**copy always creates an immutable copy.

mutableCopy always creates a mutable copy.

**149- Strong vs Weak**

**Ans :***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.**

**Ans :** *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**

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

**152-Abstract class in cocoa.**

**Ans :** *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.**

**Ans :***· 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.*

**154-GCD**

**Ans :***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?**

**Ans :**

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

**156-Call Back.**

**Ans :** *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.*

## **1) Explain what objective-C program basically consists of?**

## **Ans :**Objective-C program basically consists of

## Preprocessor commands

## Interface

## Implementation

## Method

## Variables

## Statements & Expressions

## Comments

## **2) Explain what is OOP?**

## **Ans :**OOP means Object Oriented Programming; it is a type of programming technique that helps to manage a set of objects in a system. With the help of various programming languages, this method helps to develop several computer programs and applications.

## **3) Explain what is the protocol in Objective C?**

## **Ans :**In Objetive-C, a protocol is a language feature, that provides multiple inheritances in a single inheritance language. Objective C supports two types of protocol.

## Ad hoc protocols known as informal protocol

## Compiler protocols known as a formal protocol

## **4) Explain what is the difference between polymorphism and abstraction?**

## **Ans :**Abstraction in OOP is the process of reducing the unwanted data and maintaining only the relevant data for the users while polymorphism enables an object to execute their functions in two or more forms.

**6) Mention which class are used to establish a connection between applications to the web server?**

**Ans :**The class used to establish connection between applications to web server are

* NSURL
* NSURL REQUEST
* NSURL CONNECTION

**7) Explain what is an accessor methods?**

**Ans :**Accessor methods are methods belonging to a class that enables you to get and set the values of instance valuable contained within the class.

**8) Explain what is #import?**

**Ans :**#import is a C pre-processor construct to avoid multiple inclusions of the same file.

**9) Mention what is the class of a constant string?**

**Ans :**It’s and NXConstantString. NXConstantString \*myString = @ “my string”;

**10) List out the methods used in NSURL connection?**

**Ans :**Methods used in NSURL connection are

* Connection did receive response
* Connection did receive data
* Connection fail with error
* Connection did finish loading

**11) Explain class definition in Objective-C?**

**Ans :**A class definition begins with the keyword **@interface** followed by the interface (class) name, and the class body, closed by a pair of curly braces. In Objective-C, all classed are retrieved from the base class called **NSObject.** It gives basic methods like memory allocation and initialization.

**12) Mention what is the use of category in Objective-C?**

**Ans :**The use of category in Objective-C is to extend an existing class by appending behavior that is useful only in certain situations. In order to add such extension to existing classes, objective –C provides extensions and categories. The syntax used to define a category is **@interface** keyword.

**13) Explain what are the characteristics of the category?**

**Ans :**Characteristics of category includes

* Even if you don’t have the original source code for implementation, a category can be declared for any class
* Any methods that you define in a category will be available to all instances of the original class as well as any sub-classes for the original class
* At runtime, there is no variation between a method appended by a category and one that is implemented by the original class

**14) Explain what is single inheritance in Objective-C?**

**Ans :**Objective-c subclass can only be obtained from a single direct parent class this concept is known as “single inheritance” .

**15) Explain what is polymorphism in Objective-C?**

**Ans :**Polymorphism in Objective-C is referred to a capability of base class pointer to call the function.

**16) Explain when to use NSArray and NSMutableArray?**

## **Ans :**

* **NSArray:** You will use an NS array when data in the array don’t change. For example, the company name you will put in NS Array so that no one can manipulate it.
* **NSMutableArray:** This array will be used in an array, when data in an array will change. For instance, if you are passing an array to function and that function will append some elements in that array then you will choose NSMutable Array.

**17) Explain what is synthesized in Objective-C?**

**Ans :**Once you have declared the property in objective-C, you have to tell the compiler instantly by using synthesize directive. This will tell the compiler to generate a getter&setter message.

**18) Explain how string is represented in Objective-C?**

**Ans :**In Objective-C, the string is represented by using NSS string and its sub-class NSMutableString provides several ways for creating string objects.

**19) Explain what is data encapsulation in Objective-C?**

**Ans :**In Objective-C, data encapsulation is referred to the mechanism of connecting the data and the functions that use them.

**20) Explain how to call the function in Objective-C?**

**Ans :**To call the function in Objective-C, you have to do Account -> Object Name -> Display account information -> Method name

**21) Explain what are objective- C blocks?**

**Ans :**In Objective-C class, there is an object that combines data with related behavior. It enables you to form distinct segments of code that can be passed around to functions or methods as if they were values. Objective-C blocks can be added to collections like NSDictionary or NSArray.

**22) Explain what is the main difference between the function calls and messages?**

**Ans :**The main difference between function call and message is that a function and its arguments are linked together in the compiled code, but a message and a receiving object are not linked until the program is executing and the message is sent.

**23) Explain how messaging works in Objective-C?**

**Ans :**Messaging are not bound to method implementation until runtime in Objective-C. The compiler transforms a message expression, into a call on a messaging function, objc\_msgSend(). This function connects the receiver and the name of the method mentioned in the message.

**24) Explain how the class “IMPLEMENTATION” is represented in Objective-C?**

**Ans :**In Objective-C the class “ IMPLEMENTATION” is represented with @implementation directive and ends with @end.

**25) Explain what is dot notation?**

**Ans :**Dot notation involves assessing an instance variable by determining a class “instance” followed by a “dot” followed in turn by the name of instance variable or property to be accessed.

**26) Mention whether NS object is a parent class or derived class?**

**Ans :**NS object is the parent class and consists of a number of instance variables and instance methods.

**27) What is the reuseIdentifier used for?**

**Answer:**

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

The reuseIdentifier is used to indicate that a cell can be re-used in a UITableView. For example when the cell looks the same, but has different content. The UITableView will maintain an internal cache of UITableViewCell’s with the reuseIdentifier and allow them to be re-used when dequeueReusableCellWithIdentifier: is called. By re-using table cell’s the scroll performance of the tableview is better because new views do not need to be created.

**28)Explain the difference between atomic and nonatomic synthesized properties?**

**Answer:** Atomic and non-atomic refers to whether the setters/getters for a property will atomically read and write values to the property. When the atomic keyword is used on a property, any access to it will be “synchronized”. Therefore a call to the getter will be guaranteed to return a valid value, however this does come with a small performance penalty. Hence in some situations nonatomic is used to provide faster access to a property, but there is a chance of a race condition causing the property to be nil under rare circumstances (when a value is being set from another thread and the old value was released from memory but the new value hasn’t yet been fully assigned to the location in memory for the property).

**29)Explain the difference between copy and retain?**

**Answer:** Retaining an object means the retain count increases by one. This means the instance of the object will be kept in memory until it’s retain count drops to zero. The property will store a reference to this instance and will share the same instance with anyone else who retained it too. Copy means the object will be cloned with duplicate values. It is not shared with any one else.

**30)What is method swizzling in Objective C and why would you use it?**

**Answer:** Method swizzling allows the implementation of an existing selector to be switched at runtime for a different implementation in a classes dispatch table. Swizzling allows you to write code that can be executed before and/or after the original method. For example perhaps to track the time method execution took, or to insert log statements

#import "UIViewController+Log.h"  
@implementation UIViewController (Log)  
 + (void)load {  
 static dispatch\_once\_t once\_token;  
 dispatch\_once(&once\_token, ^{  
 SEL viewWillAppearSelector = @selector(viewDidAppear:);  
 SEL viewWillAppearLoggerSelector = @selector(log\_viewDidAppear:);  
 Method originalMethod = class\_getInstanceMethod(self, viewWillAppearSelector);  
 Method extendedMethod = class\_getInstanceMethod(self, viewWillAppearLoggerSelector);  
 method\_exchangeImplementations(originalMethod, extendedMethod);  
 });  
 }  
 - (void) log\_viewDidAppear:(BOOL)animated {  
 [self log\_viewDidAppear:animated];  
 NSLog(@"viewDidAppear executed for %@", [self class]);  
 }  
@end

**31)What’s the difference between not-running, inactive, active, background and suspended execution states?**

**Answer:**

* **Not running:** The app has not been launched or was running but was terminated by the system.
* **Inactive:** 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.
* **Active:** The app is running in the foreground and is receiving events. This is the normal mode for foreground apps.
* **Background:** 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.
* **Suspended:** 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.

**31)What is a category and when is it used?**

**Answer:** A category is a way of adding additional methods to a class without extending it. It is often used to add a collection of related methods. A common use case is to add additional methods to built in classes in the Cocoa frameworks. For example adding async download methods to the UIImage class.

**32)Can you spot the bug in the following code and suggest how to fix it:**

**Answer:**

@interface MyCustomController : UIViewController   
  
@property (strong, nonatomic) UILabel \*alert;   
  
@end   
  
@implementation MyCustomController   
  
- (void)viewDidLoad {  
 CGRect frame = CGRectMake(100, 100, 100, 50);  
 self.alert = [[UILabel alloc] initWithFrame:frame];  
 self.alert.text = @"Please wait...";  
 [self.view addSubview:self.alert];  
 dispatch\_async(  
 dispatch\_get\_global\_queue(DISPATCH\_QUEUE\_PRIORITY\_DEFAULT, 0),  
 ^{  
 sleep(10);  
 self.alert.text = @"Waiting over";  
 }  
 );   
}   
  
@end

All UI updates must be done on the main thread. In the code above the update to the alert text may or may not happen on the main thread, since the global dispatch queue makes no guarantees . Therefore the code should be modified to always run the UI update on the main thread

dispatch\_async(   
 dispatch\_get\_global\_queue(DISPATCH\_QUEUE\_PRIORITY\_DEFAULT, 0),  
 ^{  
 sleep(10);  
 dispatch\_async(dispatch\_get\_main\_queue(), ^{  
 self.alert.text = @"Waiting over";  
 });  
});

**32)What is the difference between viewDidLoad and viewDidAppear? Which should you use to load data from a remote server to display in the view?**

**Answer:** viewDidLoad is called when the view is loaded, whether from a Xib file, storyboard or programmatically created in loadView. viewDidAppear is called every time the view is presented on the device. Which to use depends on the use case for your data. If the data is fairly static and not likely to change then it can be loaded in viewDidLoad and cached. However if the data changes regularly then using viewDidAppear to load it is better. In both situations, the data should be loaded asynchronously on a background thread to avoid blocking the UI.

**33)What considerations do you need when writing a UITableViewController which shows images downloaded from a remote server?**

**Answer:** This is a very common task in iOS and a good answer here can cover a whole host of knowledge. The important piece of information in the question is that the images are hosted remotely and they may take time to download, therefore when it asks for “considerations”, you should be talking about:

* Only download the image when the cell is scrolled into view, i.e. whencellForRowAtIndexPath is called.
* Downloading the image asynchronously on a background thread so as not to block the UI so the user can keep scrolling.
* When the image has downloaded for a cell we need to check if that cell is still in the view or whether it has been re-used by another piece of data. If it’s been re-used then we should discard the image, otherwise we need to switch back to the main thread to change the image on the cell.

Other good answers will go on to talk about offline caching of the images, using placeholder images while the images are being downloaded.

**34)What is a protocol, how do you define your own and when is it used?**

**Answer:** A protocol is similar to an interface from Java. It defines a list of required and optional methods that a class must/can implement if it adopts the protocol. Any class can implement a protocol and other classes can then send messages to that class based on the protocol methods without it knowing the type of the class.

@protocol MyCustomDataSource  
- (NSUInteger)numberOfRecords;  
- (NSDictionary \*)recordAtIndex:(NSUInteger)index;  
@optional  
- (NSString \*)titleForRecordAtIndex:(NSUInteger)index;  
@end

A common use case is providing a DataSource for UITableView or UICollectionView.

**35)What is KVC and KVO? Give an example of using KVC to set a value.**

**Answer:** *KVC* stands for *Key-Value Coding*. It's a mechanism by which an object's properties can be accessed using string's at runtime rather than having to statically know the property names at development time. *KVO* stands for *Key-Value Observing* and allows a controller or class to observe changes to a property value.

Let's say there is a property name on a class:

@property (nonatomic, copy) NSString \*name;

We can access it using KVC:

NSString \*n = [object valueForKey:@"name"]

And we can modify it's value by sending it the message:

[object setValue:@"Mary" forKey:@"name"]

**36)What are blocks and how are they used?**

**Answer:** Blocks are a way of defining a single task or unit of behavior without having to write an entire Objective-C class. Under the covers Blocks are still Objective C objects. They are a language level feature that allow programming techniques like lambdas and closures to be supported in Objective-C. Creating a block is done using the ^ { } syntax:

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

It can be invoked like so:

myBlock();

It is essentially a function pointer which also has a signature that can be used to enforce type safety at compile and runtime. For example you can pass a block with a specific signature to a method like so:

- (void)callMyBlock:(void (^)(void))callbackBlock;

If you wanted the block to be given some data you can change the signature to include them:

- (void)callMyBlock:(void (^)(double, double))block {  
 ...  
 block(3.0, 2.0);  
}

**37)What mechanisms does iOS provide to support multi-threading?**

* NSThread creates a new low-level thread which can be started by calling the start method.

NSThread\* myThread = [[NSThread alloc] initWithTarget:self  
 selector:@selector(myThreadMainMethod:)  
 object:nil];  
[myThread start];

* NSOperationQueue allows a pool of threads to be created and used to execute NSOperations in parallel. NSOperations can also be run on the main thread by asking NSOperationQueue for the mainQueue.

NSOperationQueue\* myQueue = [[NSOperationQueue alloc] init];  
[myQueue addOperation:anOperation];   
[myQueue addOperationWithBlock:^{  
 /\* Do something. \*/  
}];

* *GCD* or *Grand Central Dispatch* is a modern feature of Objective-C that provides a rich set of methods and API's to use in order to support common multi-threading tasks. *GCD*provides a way to queue tasks for dispatch on either the main thread, a concurrent queue (tasks are run in parallel) or a serial queue (tasks are run in FIFO order).

dispatch\_queue\_t myQueue = dispatch\_get\_global\_queue(DISPATCH\_QUEUE\_PRIORITY\_DEFAULT, 0);  
dispatch\_async(myQueue, ^{  
 printf("Do some work here.\n");  
});

**38)What's the difference between using a *delegate* and *notification*?**

**Answer:** Both are used for sending values and messages to interested parties. A *delegate* is for one-to-one communication and is a pattern promoted by Apple. In *delegation* the class raising events will have a property for the *delegate* and will typically expect it to implement some protocol. The *delegating* class can then call the *delegate*s protocol methods.

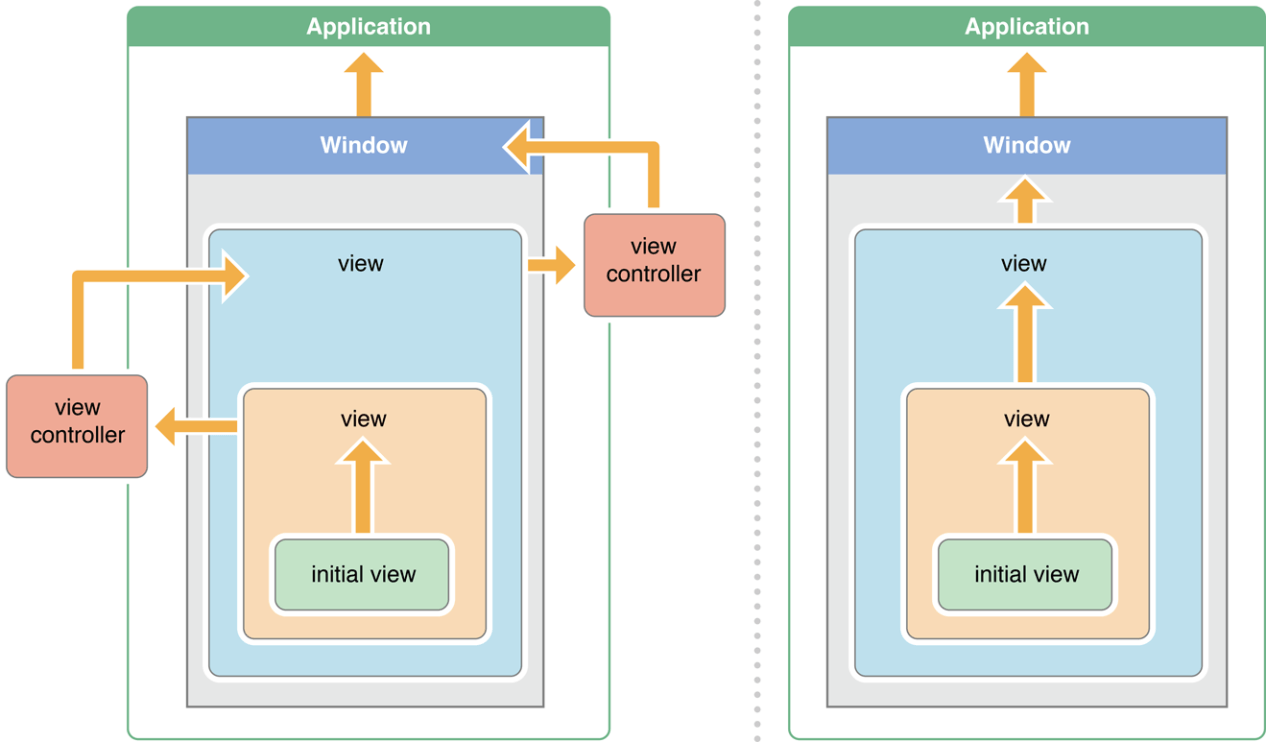
*Notification* allows a class to broadcast events across the entire application to any interested parties. The broadcasting class doesn't need to know anything about the listeners for this event, therefore *notification* is very useful in helping to decouple components in an application.

[NSNotificationCenter defaultCenter]   
 postNotificationName:@"TestNotification"   
 object:self];

**39)What is the Responder Chain?**

**Answer:** When an event happens in a view, for example a touch event, the view will fire the event to a chain of UIResponder objects associated with the UIView. The first UIResponder is the UIViewitself, if it does not handle the event then it continues up the chain to until UIResponder handles the event. The chain will include UIViewControllers, parent UIViews and their associatedUIViewControllers, if none of those handle the event then the UIWindow is asked if it can handle it and finally if that doesn't handle the event then the UIApplicationDelegate is asked.

If you get the opportunity to draw this one out, it's worth doing to impress the interviewer:



**40)What's your preference when writing UI's? Xib files, Storyboards or programmaticUIView?**

**Answer:** There's no right or wrong answer to this, but it's great way of seeing if you understand the benefits and challenges with each approach. Here's the common answers I hear:

* Storyboard's and Xib's are great for quickly producing UI's that match a design spec. They are also really easy for product managers to visually see how far along a screen is.
* Storyboard's are also great at representing a flow through an application and allowing a high-level visualization of an entire application.
* Storyboard's drawbacks are that in a team environment they are difficult to work on collaboratively because they're a single file and merge's become difficult to manage.
* Storyboards and Xib files can also suffer from duplication and become difficult to update. For example if all button's need to look identical and suddenly need a color change, then it can be a long/difficult process to do this across storyboards and xibs.
* Programmatically constructing UIView's can be verbose and tedious, but it can allow for greater control and also easier separation and sharing of code. They can also be more easily unit tested.

Most developers will propose a combination of all 3 where it makes sense to share code, then re-usable UIViews or Xib files.

**41)How would you securely store private user data offline on a device? What other security best practices should be taken?**

**Answer:** Again there is no right answer to this, but it's a great way to see how much a person has dug into iOS security. If you're interviewing with a bank I'd almost definitely expect someone to know something about it, but all companies need to take security seriously, so here's the ideal list of topics I'd expect to hear in an answer:

* If the data is extremely sensitive then it should never be stored offline on the device because all devices are crackable.
* The keychain is one option for storing data securely. However it's encryption is based on the pin code of the device. User's are not forced to set a pin, so in some situations the data may not even be encrypted. In addition the users pin code may be easily hacked.
* A better solution is to use something like [SQLCipher](https://www.zetetic.net/sqlcipher/) which is a fully encrypted SQLite database. The encryption key can be enforced by the application and separate from the user's pin code.

Other security best practices are:

* Only communicate with remote servers over SSL/HTTPS.
* If possible implement certificate pinning in the application to prevent man-in-the-middle attacks on public WiFi.
* Clear sensitive data out of memory by overwriting it.
* Ensure all validation of data being submitted is also run on the server side.

**41)What is MVC? How is it implemented in iOS? What are some pitfalls you've experienced with it? Are there any alternatives to MVC?**

**Answer:** *MVC* stands for *Model, View, Controller*. It is a design pattern that defines how to separate out logic when implementing user interfaces. In iOS, Apple provides UIView as a base class for all*View*s, UIViewController is provided to support the *Controller* which can listen to events in a*View* and update the *View* when data changes. The *Model* represents data in an application and can be implemented using any NSObject, including data collections like NSArray andNSDictionary.

Some of the pitfalls that people hit are bloated UIViewController and not separating out code into classes beyond the MVC format. I'd highly recommend reading up on some solutions to this:

In terms of alternatives, this is pretty open ended. The most common alternative is MVVM using ReactiveCocoa, but others include VIPER and using Functional Reactive code.

**42)A product manager in your company reports that the application is crashing. What do you do?**

**Answer:** This is a great question in any programming language and is really designed to see how you problem solve. You're not given much information, but some interviews will slip you more details of the issue as you go along. Start simple:

* get the exact steps to reproduce it.
* find out the device, iOS version.
* do they have the latest version?
* get device logs if possible.

Once you can reproduce it or have more information then start using tooling. Let's say it crashes because of a memory leak, I'd expect to see someone suggest using *Instruments*leak tool. A really impressive candidate would start talking about writing a unit test that reproduces the issue and debugging through it. Other variations of this question include slow UI or the application freezing. Again the idea is to see how you problem solve, what tools do you know about that would help and do you know how to use them correctly.

**43)What is AutoLayout? What does it mean when a constraint is "broken" by iOS?**

**Answer:** *AutoLayout* is way of laying out UIViews using a set of constraints that specify the location and size based relative to other views or based on explicit values. *AutoLayout* makes it easier to design screens that resize and layout out their components better based on the size and orientation of a screen. *Constraint*s include:

* setting the horizontal/vertical distance between 2 views
* setting the height/width to be a ratio relative to a different view
* a width/height/spacing can be an explicit static value

Sometimes constraints conflict with each other. For example imagine a UIView which has 2 height constraints: one says make the UIView 200px high, and the second says make the height twice the height of a button. If the iOS runtime can not satisfy both of these constraints then it has to pick only one. The other is then reported as being "broken" by iOS.

**44)What are different ways that you can specify the layout of elements in a UIView?**

**Answer:**

* 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.

**45)What’s the difference between an “app ID” and a “bundle ID” and what is each used for?**

**Answer:** An [App ID](https://developer.apple.com/library/mac/documentation/General/Conceptual/DevPedia-CocoaCore/AppID.html) is a two-part string used to identify one or more apps from a single development team. The string consists of a Team ID and a bundle ID search string, with a period (.) separating the two parts. The Team ID is supplied by Apple and is unique to a specific development team, while the bundle ID search string is supplied by teh developer to match either the bundle ID of a single app or a set of bundle IDs for a group of apps.

Because most people think of the App ID as a string, they think it is interchangeable with Bundle ID. It appears this way because once the App ID is created in the Member Center, you only ever use the App ID Prefix which matches the Bundle ID of the Application Bundle.

The bundle ID uniquely defines each App. It is specified in Xcode. A single Xcode project can have multiple Targets and therefore output multiple apps. A common use case for this is an app that has both lite/free and pro/full versions or is branded multiple ways.

**46)What are “strong” and “weak” references? Why are they important and how can they be used to help control memory management and avoid memory leaks?**

**Answer:** By default, any variable that points to another object does so with what is referred to as a “strong” reference. A retain cycle occurs when two or more objects have *reciprocal* strong references (i.e., strong references to each other). Any such objects will never be destroyed by [ARC](https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/AutomaticReferenceCounting.html) (iOS’ Automatic Reference Counting). Even if every other object in the application releases ownership of these objects, these objects (and, in turn, any objects that reference them) will continue to exist by virtue of those mutual strong references. This therefore results in a memory leak.

Reciprocal strong references between objects should therefore be avoided to the extent possible. However, when they are necessary, a way to avoid this type of memory leak is to employ weak references. Declaring one of the two references as weak will break the retain cycle and thereby avoid the memory leak.

To decide which of the two references should be weak, think of the objects in the retain cycle as being in a parent-child relationship. In this relationship, the parent should maintain a strong reference (i.e., ownership of) its child, but the child should not maintain maintain a strong reference (i.e., ownership of) its parent.

For example, if you have Employer and Employee objects, which reference one another, you would most likely want to maintain a strong reference from the Employer to the Employee object, but have a weak reference from the Employee to thr Employer.

**47)Describe managed object context and the functionality that it provides.**

**Answer:** A managed object context (represented by an instance of NSManagedObjectContext) is basically a temporary “scratch pad” in an application for a (presumably) related collection of objects. These objects collectively represent an internally consistent view of one or more persistent stores. A single managed object instance exists in one and only one context, but multiple copies of an object can exist in different contexts.

You can think of a managed object context as an intelligent scratch pad. When you fetch objects from a persistent store, you bring temporary copies onto the scratch pad where they form an object graph (or a collection of object graphs). You can then modify those objects however you like. Unless you actually save those changes, though, the persistent store remains unchanged.

Key functionality provided by a managed object context includes:

* *Life-cycle management.* The context provides validation, inverse relationship handling, and undo/redo. Through a context you can retrieve or “fetch” objects from a persistent store, make changes to those objects, and then either discard the changes or commit them back to the persistent store. The context is responsible for watching for changes in its objects and maintains an undo manager so you can have finer-grained control over undo and redo. You can insert new objects and delete ones you have fetched, and commit these modifications to the persistent store.
* *Notifications.* A context posts notifications at various points which can optionally be monitored elsewhere in your application.
* *Concurrency.* Core Data uses thread (or serialized queue) confinement to protect managed objects and managed object contexts. In OS X v10.7 and later and iOS v5.0 and later, when you create a context you can specify the concurrency pattern with which you will use it using initWithConcurrencyType:.

**48)Compare and contrast the different ways of achieving concurrency in OS X and iOS.**

**Answer:** There are basically three ways of achieving concurrency in iOS:

* threads
* dispatch queues
* operation queues

The disadvantage of threads is that they relegate the burden of creating a scalable solution to the developer. You have to decide how many threads to create and adjust that number dynamically as conditions change. Also, the application assumes most of the costs associated with creating and maintaining the threads it uses.

OS X and iOS therefore prefer to take an asynchronous design approach to solving the concurrency problem rather than relying on threads.

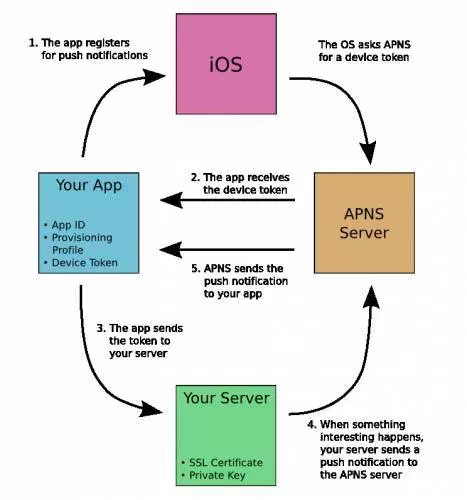
One of the technologies for starting tasks asynchronously is [Grand Central Dispatch (GCD)](https://developer.apple.com/library/prerelease/mac/documentation/Performance/Reference/GCD_libdispatch_Ref/index.html) that relegates thread management down to the system level. All the developer has to do is define the tasks to be executed and add them to the appropriate *dispatch queue*. GCD takes care of creating the needed threads and scheduling tasks to run on those threads.

All [dispatch queues](https://developer.apple.com/library/ios/documentation/General/Conceptual/ConcurrencyProgrammingGuide/OperationQueues/OperationQueues.html) are first-in, first-out (FIFO) data structures, so tasks are always started in the same order that they are added.

An [operation queue](https://developer.apple.com/library/ios/documentation/General/Conceptual/ConcurrencyProgrammingGuide/ConcurrencyandApplicationDesign/ConcurrencyandApplicationDesign.html#//apple_ref/doc/uid/TP40008091-CH100-SW9) is the Cocoa equivalent of a *concurrent dispatch queue* and is implemented by the [NSOperationQueue](https://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSOperation_class/index.html) class. Unlike dispatch queues, operation queues are not limited to executing tasks in FIFO order and support the creation of complex execution-order graphs for your tasks.

### **\*Q: Explain how the push notification works.**

**Ans:**



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

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.

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

**Ans:** 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?**

**Ans:** iOS 4.0.

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

**Ans:**  256bytes.

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

**Ans:** 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: What is Controller Object?**

**Ans:** A controller object acts as a coordinator or as an intermediary between one or more view objects and one or more model objects. In the Model-View-Controller design pattern, a controller object (or, simply, a*controller*) interprets user actions and intentions made in view objects—such as when the user taps or clicks a button or enters text in a text field—and communicates new or changed data to the model objects.

When model objects change—for example, the user opens a document stored in the file system—it communicates that new model data to the view objects so that they can display it. Controllers are thus the conduit through which view objects learn about changes in model objects and vice versa. Controller objects can also set up and coordinate tasks for an application and manage the life cycles of other objects. The Cocoa frameworks offer three main controller types: coordinating controllers, view controllers (on iOS), and mediating controllers (on OS X).

### **\*Q: What is Wildcard App IDs?**

**Ans:** A wildcard app ID allows you to use an app ID to match multiple apps; wildcard app IDs are useful when you first start developing new apps because you don’t need to create a separate app ID for each app. However, wildcard app IDs can’t be used to provision an app that uses APNS, In App Purchase, or Game Center.

A wildcard app ID omits some or all of the bundle ID in the search string and replaces that portion with an asterisk character (\*). The asterisk must always appear as the last character in the bundle search string.

When you use a wildcard app ID, characters preceding the asterisk (if any) must match the characters in the bundle ID, exactly as they would for an explicit app ID. The asterisk matches all remaining characters in the bundle ID. Further, the asterisk must match at least one character in the bundle ID. This table shows an example search string and how it matches some bundle IDs but not others.

if an app id uses an \* as the bundle ID, then the search string matches any bundle ID.

**\*Q: What is categories in iOS?**

**Ans:** 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.

Refer: [Categories and Extensions](https://developer.apple.com/library/ios/#documentation/cocoa/conceptual/objectivec/chapters/occategories.html)

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

**Ans:** Delegation is a design pattern in which one object sends messages to another object—specified as its delegate—to ask for input or to notify it that an event is occurring. Delegation is often used as an alternative to class inheritance to extend the functionality of reusable objects. For example, before a window changes size, it asks its delegate whether the new size is ok. The delegate replies to the window, telling it that the suggested size is acceptable or suggesting a better size. (For more details on window resizing, see the [block]2[/block] message.)

Delegate methods are typically grouped into a protocol. A protocol is basically just a list of methods. The delegate protocol specifies all the messages an object might send to its delegate. If a class conforms to (or *adopts*) a protocol, it guarantees that it implements the required methods of a protocol. (Protocols may also include optional methods).In this application, the application object tells its delegate that the main startup routines have finished by sending it the [block]3[/block] message. The delegate is then able to perform additional tasks if it wants.

### **\*Q: How can we achieve singleton pattern in iOS?**

**Ans:** The Singleton design pattern ensures a class only has one instance, and provides a global point of access to it. The class keeps track of its sole instance and ensures that no other instance can be created. Singleton classes are appropriate for situations where it makes sense for a single object to provide access to a global resource.

Several Cocoa framework classes are singletons.

They include [block]4[/block], NSWorkspace, NSApplication, and, in UIKit, [block]5[/block]. A process is limited to one instance of these classes. When a client asks the class for an instance, it gets a shared instance, which is lazily created upon the first request.Refer: [Singleton Pattren](https://developer.apple.com/library/mac/#documentation/General/Conceptual/DevPedia-CocoaCore/Singleton.html)

### **\*Q: What is delegate pattern in iOS?**

**Ans:** 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 [block]7[/block]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.

**\*Q: What are all the difference between categories and subclasses? Why should we go to subclasses?**

**Ans:** 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.

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

**Ans:** 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?**

**Ans:** 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?**

**Ans:** 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?**

**Ans:** 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 ?**

**Ans:** 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?**

**Ans:** 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?**

**Ans:** 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?**

**Ans:**  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?**

**Ans:** YES.

### What is Memory management in iOS?

Memory management is the programming discipline of managing the life cycles of objects and freeing them when they are no longer needed. Managing object memory is a matter of performance; if an application doesn’t free unneeded objects, its memory footprint grows and performance suffers. Memory management in a Cocoa application that doesn’t use garbage collection is based on a reference counting model. When you create or copy an object, its *retain count* is 1. Thereafter other objects may express an ownership interest in your object, which increments its retain count. The owners of an object may also relinquish their ownership interest in it, which decrements the retain count. When the retain count becomes zero, the object is deallocated (destroyed).

### **\*Q: What’s the correct way to release outlets? Why my outlets are not released in low-memory situation while views can be released automatically by iOS in low-memory situation as Apple said?**

***Ans:*** Make sure you release outlets in *UIViewController:*viewDidUnloadanddealloc methods.

* viewDidUnload is called ONLY in the case iOS tries to unload hidden views when the device getting low-memory situation, thus don’t depend on this event to free all your retained objects.
* dealloc is called when an object is going to be DE-allocated
* Are there any cases that the UIViewController never gets released or not being released immediately after the UIView is dismissed?

*Answer*:

Yes, this is often happened with multi-threaded implementations. And in your UIViewController, you create background processes using performSelectorInBackground or threadByDetachSelector, or blocks, the UIViewController will be released once background processes are completed (there is no way in iOS to TERMINATE a running thread as in Windows).

**\*Q:How to properly handle IBOutlet within a ViewController**

**You can do it by implementing property/synthesize pattern**

**(Example A) FooController.h: @interfaceFooController : UIViewController { UILabel \*fooLabel;} @property (nonatomic, retain) IBOutletUILabel \*fooLabel; @end FooController.m:@implementationFooController @synthesize fooLabel; @endBut you can also use direct instance variable (notice: no property and no synthesize):(Example B) FooController.h: @interfaceFooController : UIViewController { IBOutletUILabel\*fooLabel; } @end FooController.m: @implementationFooController @end**

**Q1. Where can you test Apple iPhone apps if you don’t have the device?**

**A.** 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.

**Q2. Does iOS support multitasking?**

**A.** 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.

**Q3. Which JSON framework is supported by iOS?**

**A.** 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.

**Q4. What are the tools required to develop iOS applications?**

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

**Q5. 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.

**Q6. Name the application thread from where UIKit classes should be used?**

**A.** 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.

**Q7. Which API is used to write test scripts that help in exercising the application's user interface elements?**

**A.** 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.

### **App States and Multitasking**

**Q8. Why an app on iOS device behaves differently when running in foreground than in background?**

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

**Q9. How can an operating system improve battery life while running an app?**

**A.** 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.

**Q10. Which framework delivers event to custom object when app is in foreground?**

**A.** 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.

### **App States**

**Q11. When an app is said to be in not running state?**

**A.** An app is said to be in 'not running' state when:

- it is not launched.

- it gets terminated by the system during running.

**Q12. Assume that your app is running in the foreground but is currently not receiving events. In which sate it would be in?**

**A.** 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.

**Q13. Give example scenarios when an application goes into InActive state?**

**A.** 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.

**Q14. When an app is said to be in active state?**

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

**Q15. Name the app sate which it reaches briefly on its way to being suspended.**

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

**Q16. Assume that an app is not in foreground but is still executing code. In which state will it be in?**

**A.** Background state.

**Q17. An app is loaded into memory but is not executing any code. In which state will it be in?**

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

**Q18. Assume that system is running low on memory. What can system do for suspended apps?**

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

**Q19. How can you respond to state transitions on your app?**

**A.** 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.

**Q20. List down app's state transitions when it gets launched.**

**A.** 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.

**Q21. Who calls the main function of you app during the app launch cycle?**

**A.** 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.

### **Core App Objects**

**Q22. What is the use of controller object UIApplication?**

**A.** 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.

**Q23. Which object is create by UIApplicationMain function at app launch time?**

**A.** 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.

**Q24. How is the app delegate is declared by Xcode project templates?**

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

**Q25. What happens if IApplication object does not handle an event?**

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

**Q26. Which app specific objects store the app's content?**

**A.** 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.

**Q27. Are document objects required for an application? What does they offer?**

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

**Q28. Which object manage the presentation of app's content on the screen?**

**A.** 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.

**Q29. Which is the super class of all view controller objects?**

**A.** 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.

**Q30. What is the purpose of UIWindow object?**

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

**Q31. How do you change the content of your app in order to change the views displayed in the corresponding window?**

**A.** 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.

**Q32. Define view object.**

**A.** 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.

**Q33. You wish to define your custom view. Which class will be subclassed?**

**A.** Custom views can be defined by subclassing UIView.

**Q34. Apart from incorporating views and controls, what else an app can incorporate?**

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

**Q35. What are layer objects and what do they represent?**

**A.** 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.

## **1. What was the latest version of iOS you worked with? What do you like about it and why?**

**Expected answer:** he/she tells you what the latest version of the system is and what he/she has worked with lately. And tells you about one of the new features of the system that he/she is excited about (i.e. I love new multitasking on iPad because it’s going to make user experience way better and opens up a lot of opportunities for us developers to build new cool things, etc.).

## **2. Have you worked with Swift? What do you like about it? What you don’t like?**

**Expected answer:** be cautious, use Swift along with Objective-C for now and move to pure Swift down the road in several years when it and the ecosystem of libraries around it matures enough.

## **3. How memory management is handled on iOS?**

**Expected answer:** it is great if interviewee knows MRC but it is not necessary. More important that he/she understands [`strong`, `weak`, `assign`](https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/EncapsulatingData/EncapsulatingData.html), etc. directives on properties and can confidently tell you what those directives imply and how it’s handled with blocks.

## **4. What do you know about singletons? Where would you use one and where you wouldn't?**

This is a typical objective oriented programming question. In case of iOS you can get a feel of how the interviewee is using it in his/her apps. You’ll be able to weed out those people who came from Java/PHP and such and use it as a “global” store or something similar.

Expected answer: singletons should be avoided and used only for objects that need to be shared throughout your application for same reason. It should be used carefully, instantiated lazily, if it’s possible, and passed into other objects as a dependency using Inversion of Control and Dependency Injection principles.Red flag: is when the interviewee talks about global variables and stores.

## **5. Could you explain what is the difference between Delegate and KVO?**

With this question you are assessing their knowledge of different types of messaging patterns used in iOS. Senior developer should’ve used both of those many times in his/her applications.

Expected answer: Both are ways to have relationships between objects.[Delegation](https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/Delegation.html) is a one to one relationship where one object implements a delegate protocol and another uses it and sends messages to assuming that those methods are implemented since the receiver promised to comply to the protocol. [KVO](http://nshipster.com/key-value-observing/) is many-to-many relationship where one object could broadcast a message and one or multiple other objects can listen to it and react.

## **6. How do you usually persist data on iOS?**

This will tell you if they just played with rudimentary ways to store data locally on iOS like `NSUserDefaults`, `Plists`, disk file storage, etc. or if they have used more advanced storages like `Core Data` and `Realm`. Ideally they should know when it is appropriate to use all of the above persistence tools.

Expected answer: A senior developer should at least be able to tell you when it is appropriate to use `NSUserDefaults`, `Core Data`, and disk file storage. `Core Data` is what you’re expecting them to explain you the most. Other possible options are `Realm` and similar third party (non-Apple) solutions but make sure they know why they would want to use them instead of `Core Data`.With `Core Data` they should be able to explain how it works and how it’s different from `SQLite` (i.e. it’s an object graph rather than a relational database, etc.).

## **7. How do you typically do networking?**

Networking is a vital part of almost any application these days. In fact the majority of them is useless without an internet connection. Any decent iOS developer should know how to initiate various networking requests (GET, POST, PUT, DELETE, etc.) using Apple’s frameworks or third party tools such as `[AFNetworking](https://github.com/AFNetworking/AFNetworking)`.

Expected answer: Interviewee should be able to explain how to do not just basic networking but more advanced things like HTTP headers and token handling. But in general look for developers who use either `[AFNetworking](https://github.com/AFNetworking/AFNetworking)` or `[Alamofire](https://github.com/Alamofire/Alamofire)` or similar. The reason is these libraries are used very extensively by iOS community and are utilized in many other tools. Senior developer should be able to utilize that instead of reinventing a wheel.

## **8. How do you serialize data on iOS (JSON from the web or local app data to Core Data and other storages) ?**

Data serialization is an important task for every iOS application. JSON is defacto standard of data serialization on the web and every developer should know how to effectively work with it without spending a lot of time writing boilerplate code. Same applies to data serialization for local storage. It can be handled in multiple ways but a good developer should at least be aware of the challenges with these tasks. Expected answer: This is a tricky question. A naive developer would say that he/she parses JSON data using Apple’s `[NSJSONSerialization](https://developer.apple.com/library/ios/documentation/Foundation/Reference/NSJSONSerialization_Class/)` and then takes the resulting dictionary and assigns the data from each key to respective properties on his/her model object. This is not what you expect a senior developer to tell you. Senior developer should be aware of the issues that could arise with dataserialization such as data validation and conversion. A good answer would be clear understanding that in general there are two types of data serialization: JSON to domain model, and domain model to persistence. Utilization of pods like[Mantle](https://github.com/Mantle/Mantle) or [ObjectMapper](https://github.com/Hearst-DD/ObjectMapper) or a similar library/pod is expected.

## **9. What design patterns do you know and use on iOS?**

This could be a very simple or a very complicated answer. Every iOS developer should know about MVC but if you’re looking for a senior developer than he/she should have a lot of patterns and best practices on how to organize code under his belt.

Expected answer: at least [MVVM](http://www.sprynthesis.com/2014/12/06/reactivecocoa-mvvm-introduction/). This is the holy grail that saves iOS developers from [Massive View Controllers](https://www.objc.io/issues/1-view-controllers/). Senior developer should be able to explain you what MVVM is, why it’s better than MVC, and should be able to tell you what other patterns he/she uses along with MVVM (Inversion of Control, Dependency Injection, etc.).Red flags: if interviewee tells you that he uses only MVC because Apple said so and he/she never tried MVVM than that person is definitely not a senior iOS developer.

## **10. What is Autolayout?**

Autolayout is one of the technologies on iOS that helps us build scalable UI that can be used on many devices of different shape and size. This is a must know for any iOS developer especially for a senior one.

Expected answer: do not expect proficiency with this technology but the interviewee should be able to tell you when and how he/she would use it and what benefits it gives them (i.e. scalable adjustable declarative UI).

## **11. How do you handle async tasks?**

Asynchronous programming is a vital part of any iOS application. Networking, local storage, and other heavy computation tasks should be handled in the background to avoid blocking UI and having users wait or system kill your application.

Expected answer: answers to this questions may vary from [NSOperations](http://nshipster.com/nsoperation/) and[GCD](https://developer.apple.com/library/ios/documentation/Performance/Reference/GCD_libdispatch_Ref/) to Promises and RAC. A good developer knows multiple ways to execute async operations and knows when they are necessary (i.e. with networking, local persistence, etc. see above). From a senior developer though we expect a more higher and broader level of tools they use such as [ReactiveCocoa](https://github.com/ReactiveCocoa/ReactiveCocoa/) and [PromiseKit](https://github.com/mxcl/PromiseKit).

## **12. How do you manage dependencies?**

Dependencies management is a vital but daunting task. It was very difficult and error prone to do before but these days we have several tools to help us out with it. Every iOS dev should know how to handle dependencies and collaborate with other teammates.

Expected answer: CocoaPods and/or Carthage.Red flags: if the say that they don’t use any dependency manager and just copy files to the project folder. Also a red flag if the use git submodules (it leads to project configuration issues and inconsistencies between local setups for different developers on the team).

## **13. How do you debug and profile things on iOS?**

No one writes perfect code and debugging and profiling is one of the tools that we use to figure out the right technical solution. On iOS we have all the typical “manual” debugging tools such as `NSLog`/`print` functions to output things in console. But Apple also provides us with more advanced variety of tools and instruments to help with identifying where problems lie.

Expected answer: every iOS developer should be able to explain how he/she would use breakpoints and logs to get the runtime data but from a senior developer you should expect to hear things about [Instruments](https://developer.apple.com/library/watchos/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/index.html) and [Crash Logs](https://developer.apple.com/library/ios/technotes/tn2151/_index.html).

## **14. Do you code review?**

Code reviews is one of the most effective development methodologies. It helps understand the problem better, share knowledge, share techniques, catch bugs, share ownership of the codebase, etc. This style of development is not for everyone but every development should be able to do that effectively.

Expected answer: senior developer should be more or less proficient in this type of code collaboration. Again, this is not for everyone (depends on personality, compatibility and other factors), but that is a skill that should show you if the candidate is able to work with other people and communicate his thoughts and ideas clearly to another teammate.

## **15. Do you test your code? What do you do to make your code testable?**

This is embarrassing but we admit that we don’t do testing as much as we should. We know we really really should do it more. We are talking about Unit Testing and Integration Testing specifically here.

Expected answer: there is only one right answer here: either they do it or they wish they would. In general iOS community isn’t as big on testing as say Ruby community. Partially it is due to inherited lack of testing culture and partially due to lack of good tools for testing but things seems to be improving on that front.