iOS Interview Questions and Answers

BY: SANDEEP REDDY CHALLA

1-How would you create your own custom view?

By Subclassing the UIView class.

2-Whats fast enumeration?

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

3-Whats a struct?

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

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

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

5-Explain retain counts.

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

6-Whats the difference between frame and bounds?

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

7-Is a delegate retained?

No, the delegate is never retained! Ever!

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

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

9- What is dynamic?

You use the @dynamic keyword to tell the compiler that you will fulfill the API contract implied by a property either by providing method

implementations directly or at runtime using other mechanisms such as dynamic loading of code or dynamic method resolution. It suppresses the warnings that the compiler would otherwise generate if it can't find suitable implementations. You should use it only if you know that the methods will be available at runtime

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

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

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

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

12-Whats the NSCoder class used for?

NSCoder is an abstractClass which represents a stream of data. They are used in Archiving and Unarchiving objects. NSCoder objects are usually used in a method

that is being implemented so that the class conforms to the protocol. (which has something like encodeObject and decodeObject methods in them).

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

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

14-Explain the correct way to manage Outlets memory

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

15-Is the delegate for a CAAnimation retained?

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

16-What happens when the following code executes?

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

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

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

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

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

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

```
- (NSString*) title // Getter method
{

return title;
}
- (void) setTitle: (NSString*) newTitle //Setter method
```

```
{
if (newTitle != title)
{
[title release];
title = [newTitle retain]; // Or copy, depending on your needs. }
}
```

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

```
-(id)retain
{
NSIncrementExtraRefCount(self);
return self;
}
-(void)release
{ if(NSDecrementExtraRefCountWasZero(self)) {
NSDeallocateObject(self);
}
}
-(id)autorelease
{ // Add the object to the autorelease pool [NSAutoreleasePool addObject:self];
return self
```

20-What are the App states. Explain them?

- Not running State: The app has not been launched or was running but was terminated by the system.

- Inactive state: The app is running in the foreground but is currently not receiving events. (It may be executing other code though.) An app usually stays in this state only briefly as it transitions to a different state. The only time it stays inactive for any period of time is when the user locks the screen or the system prompts the user to respond to some event, such as an incoming phone call or SMS message.

- Active state: The app is running in the foreground and is receiving events. This is the normal mode for foreground apps.

- Background state: The app is in the background and executing code. Most apps enter this state briefly on their way to being suspended. However, an app that requests extra execution time may remain in this state for a

period of time. In addition, an app being launched directly into the background enters this state instead of the inactive state. For information about how to execute code while in the background, see “Background Execution and Multitasking.”

- Suspended state:The app is in the background but is not executing code. The system moves apps to this state automatically and does not notify

them before doing so. While suspended, an app remains in memory but does not execute any code. When a low-memory condition occurs, the system may purge suspended apps without notice to make more space for the foreground app.

21-What is Automatic Reference Counting (ARC) ?

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

22-Multitasking support is available from which version?

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

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

256bytes.

24-What is the difference between retain & assign?

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

```
if (_variable != object)
{

[_variable release];
_variable = nil;
_variable = object;
}
```

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

```
if (_variable != object)
{ [_variable release];
```

_variable = nil;
_variable = [object retain];
}

25-Why do we need to use @Synthesize?

We can use generated code like nonatomic, atmoic, retain without writing any lines of code. We also have getter and setter methods. To use this, you have 2 other ways: @synthesize or @dynamic: @synthesize, compiler will generate the getter and setter automatically for you, @dynamic: you have to write them

yourself.@property is really good for memory management, for example: retain.How can you do retain without @property?
if (_variable != object)
{

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

26-What is categories in iOS?

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

27-What is Delegation in iOS?

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 thewindowWillResize:toSize: 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 theapplicationDidFinishLaunching: message. The delegate is then able to perform additional tasks if it wants.

28-How can we achieve singleton pattern in iOS?

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 includeNSFileManager, NSWorkspace, NSApplication, and, in UIKit,UIApplication. 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.

29-What is delegate pattern in iOS?

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 NSXMLParserobject) that is seeking to accomplish something, but can only do so in a generic fashion. The delegate, which is almost always an instance of a custom class, acts in coordination with the host object, supplying program-specific behavior at certain points in the task (see Figure 4-3). Thus delegation makes it possible to modify or extend the behavior of another object without the need for subclassing.Refer: delegate pattern

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

Category is a feature of the Objective-C language that enables you to add methods (interface and implementation) to a class without having to make a subclass. There is no runtime difference—within the scope of your program—

between the original methods of the class and the methods added by the category. The methods in the category become part of the class type and are inherited by all the class's subclasses.As with delegation, categories are not a strict adaptation of the Decorator pattern, fulfilling the intent but taking a different path to implementing that intent. The behavior added by categories is a compile-time artifact, and is not something dynamically acquired. Moreover, categories do not encapsulate an instance of the class being extended.The Cocoa frameworks define numerous categories, most of them informal protocols . Often they use categories to group related methods. You may implement categories in your code to extend classes without subclassing or to group related methods. However, you should be aware of these caveats:

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

unpredictably.

31-What is notification in iOS?

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

32-What is the difference between delegates and notifications?

We can use notifications for a variety of reasons. For example, you could broadcast a notification to change how user-interface elements display information based on a certain event elsewhere in the program. Or you could

use notifications as a way to ensure that objects in a document save their state before the document window is closed. The general purpose of notifications is to inform other objects of program events so they can respond appropriately.But objects receiving notifications can react only after the event has occurred. This is a significant difference from delegation. The delegate is given a chance to reject or modify the operation proposed by the delegating object. Observing objects, on the other hand, cannot directly affect an impending operation.

33-What is posing in iOS?

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

• A class may only pose as one of its direct or indirect superclasses
• The posing class must not define any new instance variables which are

absent from the target class (though it may define or override methods). • No messages must have been sent to the target class prior to the posing.

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

- A posing class can call overridden methods through super, thus incorporating the implementation of the target class.

- A posing class can override methods defined in categories.

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

You can use this attribute to specify that accessor methods are not atomic. (There is no keyword to denote atomic.)
nonatomic
Specifies that accessors are nonatomic. By default, accessors are atomic.
Properties are atomic by default so that synthesized accessors provide robust access to properties in a multithreaded environment—that is, the value returned from the getter or set via the setter is always fully retrieved or set regardless of what other threads are executing concurrently.
If you specify strong, copy, or retain and do not specifynonatomic, then in a reference-counted environment, a synthesized get accessor for an object

property uses a lock and retains and autoreleases the returned value—the implementation will be similar to the following:

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

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

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

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

36-Which JSON framework is supported by iOS?

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Background state.

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

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

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

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

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

On state transitions can be responded to state changes in an appropriate way by calling corresponding methods on app's delegate object.
For example: applicationDidBecomeActive method can be used to prepare to run as the foreground app.

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

applicationWillTerminate method is called when your app is being terminated.

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

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

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

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

55-What is the use of controller object UIApplication?

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

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

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

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

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

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

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

59- Which app specific objects store the app's content?

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

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

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

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

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

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

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

63-What is the purpose of UIWindow object?

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

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

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

65-Define view object.

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

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

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

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

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

68-What is App Bundle?

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

69-Define property?

It is used to access instance variables outside of class.

70-Why synthesized is used?

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

71-What is retaining?

It is reference count for an object.

72- What is webservice?

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

73-What is parsing?

To get data from web service we use parsing.

74-which xml parser we use on iphone?

“NSXML” Parser.

75-Which type of parse does iphone support?

“SAX” parser.

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

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

77-Tell the difference between DOM and SAX Parser?

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

78-Name three method of NSXML parser.

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

79-Tell methods used in NSURLConnection

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

80-.What is json-parser?

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

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

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

82-Tell me about tab bar controller?

It is used to display the data on the view.

83-Which are the protocols used in table view?

Table view contain two delegate protocols
(1) Uitable view data source
(2).Uitable view delegate.
ui view table view data source three method namely (1)No of sections.

(2)No of rows in sections.
(3)Cell for row index path row.
In ui table view delegate contain (1)Did select row at index path row

84-Name data base used in iphone?

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

85-Tell four frameworks used in iphone?

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

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

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

87-Tell me about the MVC architecture?

M-model, V-view, C-controller
Main advantage of MVC architecture is to provide “reusability and security” by separating the layer by using MVC architecture.
Model: it is a class model is interact with database.
Controller: controller is used for by getting the data from model and controls the views.
Display the information in views. : View

88-What is the instance methods?

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

Declaration of instance method :
– (void)click me: (id)sender;
Void is return type which does not giving any thing here. Click me is method name.
Id is data type which returns any type of object.

89-What is the class method?

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

{
}
+(class name *)new alloc: -(int)total open

90-What is data encapsulation?

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

91-What is accessor methods?

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

92-What is synthesized accessor methods?

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

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

(a)Data encapsulation encourages the use of methods to +get and set the values of instance variables in a class.
(b)But the developer to want to directly access an instance variable without having to go through an accessor method.

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

94-What is dot notation?

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

95-Difference between shallow copy and deep copy?

Shallow copy is also known as address copy. In this process you only copy address not actual data while in deep copy you copy data.
Suppose there are two objects A and B. A is pointing to a different array while B is pointing to different array. Now what I will do is following to do shallow copy.
Char *A = {'a','b','c'};

Char *B = {'x','y','z'};

B = A;

Now B is pointing is at same location where A pointer is pointing.Both A and B in this case sharing same data. if change is made both will get altered value of data.Advantage is that coping process is very fast and is independent of size of array.

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

96-Difference between categories and extensions?

Class extensions are similar to categories. The main difference is that with an extension, the compiler will expect you to implement the methods within your main @implementation, whereas with a category you have a separate

@implementation block. So you should pretty much only use an extension at the top of your main .m file (the only place you should care about ivars, incidentally) — it's meant to be just that, an extension.

97-What are KVO and KVC?

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

NSString age = emp.age;

setting property value.

emp.age = @"20″;

Now how KVC works is like this

[emp valueForKey:@"age"];

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

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

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

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

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

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

int x=10;

int y=5;

x=x+y;

NSLog(@"x==> %d",x); y=x-y;

NSLog(@"Y Value==> %d",y);

x=x-y;

NSLog(@"x Value==> %d",x);

100-What is push notification?

Imagine, you are looking for a job. You go to software company daily and ask sir "is there any job for me" and they keep on saying no. Your time and money is wasted on each trip.(Pull Request mechanism)
So, one day owner says, if there is any suitable job for you, I will let you know. In this mechanism, your time and money is not wasted. (Push Mechanism)

How it works?

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

Flow of push notification

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

101-What is polymorphism?

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

Ability of base class pointer to call function from derived class at runtime is called polymorphism.
For example, there is super class human and there are two subclasses software engineer and hardware engineer. Now super class human can hold reference to any of subclass because software engineer is kind of human. Suppose there is speak function in super class and every subclass has also speak function. So at

runtime, super class reference is pointing to whatever subclass, speak function will be called of that class. I hope I am able to make you understand.

101-What is responder chain?

Suppose you have a hierarchy of views such like there is superview A which have subview B and B has a subview C. Now you touch on inner most view C. The system will send touch event to subview C for handling this event. If C View does not want to handle this event, this event will be passed to its superview B (next

responder). If B also does not want to handle this touch event it will pass on to superview A. All the view which can respond to touch events are called responder chain. A view can also pass its events to uiviewcontroller. If view controller also does not want to respond to touch event, it is passed to application object which discards this event.

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

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

103-What is a protocol?

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

• Ad hoc protocols called informal protocol

• Compiler protocols called formal protocols

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

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

1. If you are writing a program that is not based on a UI framework, such as a command-line tool.

2. If you write a loop that creates many temporary objects.You may create an autorelease pool inside the loop to dispose of those objects before the next iteration. Using an autorelease pool in the loop helps to reduce the maximum memory footprint of the application.

3. If you spawn a secondary thread.

105- 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.

That's it, pretty fast and easy, but there are a lot of caveats :
• The most important problem is that the thread which called this method will be blocked until the connection finish or timeout, so we surely don't want to start the connection on the main thread to avoid freezing the UI. That means we need to create a new thread to handle the connection, and all programmers

know that threading is hard.
• Cancellation, it’s not possible to cancel a synchronous connection, which is bad because users like to have the choice to cancel an operation if they think it takes too much time to execute.
• Authentication, there is no way to deal with authentication challenges.
• It’s impossible to parse data on the fly.
So let’s put it up straight, avoid using synchronousNSURLConnection, there is absolutely no benefit of using it.
It’s clear that asynchronous connections give us more control :
• You don’t have to create a new thread for the connection because your main thread will not be blocked.
• You can easily cancel the connection just by calling the cancelmethod.
• If you need authentication just implement the required delegate methods.
• Parsing data on the fly is easy.
So clearly we have a lot of more control with this, and the code is really not difficult.
Even better, we don’t have to handle the creation of a new thread, which is a good thing, because you know, threading is hard.
Well, if you read me until here, you should be convinced to use asynchronous connections, and forget about synchronous ones. They clearly give us more control and possibilities and, in some case can spare us to create new thread.
So I encourage you to move away from synchronous connections, just think of them as evil.

107-What is the navigation controller?

Navigation controller contains the stack of controllers every navigation controller

must be having root view controller by default these controllers contain 2 method
(a) push view (b) pop view
By default navigation controller contain “table view”.

108- What is the split view controller?

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

109-Cocoa.

Cocoa is an application environment for both the Mac OS X operating system and iOS. It consists of a suite of object-oriented software libraries, a runtime

system, and an integrated development environment. Carbon is an alternative environment in Mac OS X, but it is a compatibility framework with procedural programmatic interfaces intended to support existing Mac OS X code bases.

110- Frameworks that make Cocoa.

Appkit (Application Kit) Foundation

111- Objective-C.

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

Dynamic typing—determining the class of an object at runtime Dynamic binding—determining the method to invoke at runtime Dynamic loading—adding new modules to a program at runtime

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

· The Objective-C class allows a method and a variable with the exact same name. In C++, they must be different.
· Objective-C does not have a constructor or destructor. Instead it has init and dealloc methods, which must be called explicitly.
· Objective-C uses + and – to differentiate between factory and instance methods, C++ uses static to specify a factory method.
· Multiple inheritance is not allowed in Obj-C, however we can use protocol to some extent.
· Obj-C has runtime binding leading to dynamic linking.

· Obj-C has got categories.
· Objective-C has a work-around for method overloading, but none for operator overloading.
· Objective-C also does not allow stack based objects. Each object must be a pointer to a block of memory.
· In Objective-C the message overloading is faked by naming the parameters. C ++ actually does the same thing but the compiler does the name mangling for us. In Objective-C, we have to mangle the names manually.
· One of C++'s advantages and disadvantages is automatic type coercion.
· Another feature C++ has that is missing in Objective-C is references. Because pointers can be used wherever a reference is used, there isn't much need for references in general.
· Templates are another feature that C++ has that Objective-C doesn't.

Templates are needed because C++ has strong typing and static binding that prevent generic classes, such as List and Array.

113-Appilcation Kit/App kit.

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

You also have the choice at which level you use the Application Kit:
· Use Interface Builder to create connections from user interface objects to your application objects.
· Control the user interface programmatically, which requires more familiarity with AppKit classes and protocols.
· Implement your own objects by subclassing NSView or other classes.

114-Foundation Kit.

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

· Make software development easier by introducing consistent conventions for things such as deallocation.
· Support Unicode strings, object persistence, and object distribution.
· Provide a level of OS independence, to enhance portability.

115-Dynamic and Static Typing.

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

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

116-Selectors

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

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

117-Class Introspection

· Determine whether an objective-C object is an instance of a class
[obj isMemberOfClass:someClass];
· Determine whether an objective-C object is an instance of a class or its descendants
[obj isKindOfClass:someClass];
· The version of a class
[MyString version]
· Find the class of an Objective-C object
Class c = [obj1 class]; Class c = [NSString class];
· Verify 2 Objective-C objects are of the same class
[obj1 class] == [obj2 class]

118- Proxy

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

119- Why category is better than inheritance?

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

120-Formal Protocols

Formal Protocols allow us to define the interface for a set of methods, but implementation is not done. Formal Protocols are useful when you are using DistributedObjects, because they allow you to define a protocol for

communication between objects, so that the DO system doesn't have to constantly check whether or not a certain method is implemented by the distant object.

121- Formal vs informal protocol.

In addition to formal protocols, you can also define an informal protocol by grouping the methods in a category declaration:
@interface NSObject (MyProtocol)
//someMethod();

@end
Informal protocols are typically declared as categories of the NSObject class, because that broadly associates the method names with any class that inherits from NSObject. Because all classes inherit from the root class, the methods aren't restricted to any part of the inheritance hierarchy. (It is also possible to declare an informal protocol as a category of another class to limit it to a certain branch of the inheritance hierarchy, but there is little reason to do so.) When used to declare a protocol, a category interface doesn't have a corresponding implementation. Instead, classes that implement the protocol declare the methods again in their own interface files and define them along with other methods in their implementation files.
An informal protocol bends the rules of category declarations to list a group of methods but not associate them with any particular class or implementation. Being informal, protocols declared in categories don't receive much language support. There's no type checking at compile time nor a check at runtime to see whether an object conforms to the protocol. To get these benefits, you must use a formal protocol. An informal protocol may be useful when all the methods are optional, such as for a delegate, but (in Mac OS X v10.5 and later) it is typically better to use a formal protocol with optional methods.

122- Optional vs required

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

@protocol MyProtocol @optional
-(void) optionalMethod; @required

-(void) requiredMethod; @end

123- Memory Management

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

124-Copy vs assign vs retain

· Assign is for primitive values like BOOL, NSInteger or double. For objects use retain or copy, depending on if you want to keep a reference to the original object or make a copy of it.
· assign: In your setter method for the property, there is a simple assignment of your instance variable to the new value, eg: (void)setString: (NSString*)newString{

string = newString;

}
This can cause problems since Objective-C objects use reference counting, and therefore by not retaining the object, there is a chance that the string could be deallocated whilst you are still using it.
· retain: this retains the new value in your setter method. For example:
This is safer, since you explicitly state that you want to maintain a reference of the object, and you must release it before it will be deallocated. (void)setString: (NSString*)newString{

[newString retain];
 [string release];
 string = newString;

}
· copy: this makes a copy of the string in your setter method:
This is often used with strings, since making a copy of the original object ensures that it is not changed whilst you are using it. (void)setString: (NSString*)newString{

if(string!=newString){
 [string release];

string = [newString copy];

}

}

125- alloc vs new

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

126- release vs pool drain

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

127- NSAutoReleasePool : release vs drain

Strictly speaking, from the big picture perspective drain is not equivalent to release:
In a reference-counted environment, drain does perform the same operations as release, so the two are in that sense equivalent. To emphasise, this means you do not leak a pool if you use drain rather than release.
In a garbage-collected environment, release is a no-op. Thus it has no effect. drain, on the other hand, contains a hint to the collector that it should “collect if needed”. Thus in a garbage-collected environment, using drain helps the system balance collection sweeps.

128-autorelease vs release

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

Release: retain count is decremented at this point.

129- Autorelease Pool

Autorelease pools provide a mechanism whereby you can send an object a “deferred” release message. This is useful in situations where you want to relinquish ownership of an object, but want to avoid the possibility of it being deallocated immediately (such as when you return an object from a method).

Typically, you don’t need to create your own autorelease pools, but there are some situations in which either you must or it is beneficial to do so.

130- How autorelease pool is managed.

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

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

131-Memory Leak

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

132- Event Loop

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

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

133-Differnce between boxName and self.boxName.

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

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

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

135-Collection

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

136-Threads and how to use

Use this class when you want to have an Objective-C method run in its own thread of execution. Threads are especially useful when you need to perform a lengthy task, but don't want it to block the execution of the rest of the application. In particular, you can use threads to avoid blocking the main thread

of the application, which handles user interface and event-related actions. Threads can also be used to divide a large job into several smaller jobs, which can lead to performance increases on multi-core computers.
Two ways to create threads...

· detachNewThreadSelector:toTarget:withObject:
· Create instances of NSThread and start them at a later time using the "start" method.
NSThread is not as capable as Java's Thread class, it lacks
· Built-in communication system.
· An equivalent of "join()"

137-Threadsafe

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

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

The first rule for implementing signal handlers in applications is to avoid assumptions about which thread is handling the signal. If a specific thread

wants to handle a given signal, you need to work out some way of notifying that thread when the signal arrives. You cannot just assume that installation of a signal handler from that thread will result in the signal being delivered to the same thread.

138-Notification and Observers

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

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

139-Delegate vs Notification

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

· The other difference between notifications and delegates is that there is no possibility for the receiver of a notification to return a value to the sender.
· Typical uses of notifications might be to allow different objects with an

application to be informed of an event such as a file download completing or a user changing an application preference. The receiver of the notification might then perform additional actions such as processing the downloaded file or updating the display.

140-Plist

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

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

141-Helper Objects

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

142-Cluster Class

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

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

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

144-Difference between coreData and Database

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

Leaves data constraints to the business logic side of the program

145- Core data vs sqlite.

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

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

146-Retain cycle or Retain loop.

When object A retains object B, and object B retains A. Then Retain cycle happens. To overcome this use “close” method.
Objective-C’s garbage collector (when enabled) can also delete retain-loop groups but this is not relevant on the iPhone, where Objective-C garbage collection is not supported.

147-What is unnamed category.

A named category — @interface Foo(FooCategory) — is generally used to: i. Extend an existing class by adding functionality.
ii. Declare a set of methods that might or might not be implemented by a delegate.

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

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

@interface Foo:NSObject @property(readonly, copy) NSString *bar;

-(void) publicSaucing;
@end
Foo.m
@interface Foo()
@property(readwrite, copy) NSString *bar; – (void)
superSecretInternalSaucing;

@end
@implementation Foo
@synthesize bar;
.... must implement the two methods or compiler will warn .... @end

148-Copy vs mutableCopy.

copy always creates an immutable copy. mutableCopy always creates a mutable copy.

149- Strong vs Weak

The strong and weak are new ARC types replacing retain and assign respectively. Delegates and outlets should be weak.
A strong reference is a reference to an object that stops it from being deallocated. In other words it creates a owner relationship.

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

150-__strong, __weak, __unsafe_unretained, __autoreleasing.

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

__strong – is the default so you don't need to type it. This means any object created using alloc/init is retained for the lifetime of its current scope. The "current scope" usually means the braces in which the variable is declared
__weak – means the object can be destroyed at anytime. This is only useful if the object is somehow strongly referenced somewhere else. When destroyed, a variable with __weak is set to nil.

__unsafe_unretained – is just like __weak but the pointer is not set to nil when the object is deallocated. Instead the pointer is left dangling.
__autoreleasing, not to be confused with calling autorelease on an object before returning it from a method, this is used for passing objects by reference, for example when passing NSError objects by reference such as [myObject performOperationWithError:&tmp];

151-Types of NSTableView

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

152-Abstract class in cocoa.

Cocoa doesn't provide anything called abstract. We can create a class abstract which gets check only at runtime, compile time this is not checked. @interface AbstractClass : NSObject
@end

@implementation AbstractClass
+ (id)alloc{
if (self == [AbstractClass class]) { NSLog(@"Abstract Class cant be used"); }

return [super alloc]; @end

153- Difference between HTTP and HTTPS.

· HTTP stands for HyperText Transfer Protocol, whereas, HTTPS is HyperText Transfer Protocol Secure.
· 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 andSSL (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 80 by default, while HTTPS URLs stars with "https:// " and use port 443.
· HTTP is unsafe from attacks like man-in-the-middle and eavesdropping, but HTTPS is secure from these sorts of attacks.

154-GCD

Grand Central Dispatch is not just a new abstraction around what we've already been using, it's an entire new underlying mechanism that makes multithreading easier and makes it easy to be as concurrent as your code can be without worrying about the variables like how much work your CPU cores are doing, how many CPU cores you have and how much threads you should spawn in response. You just use the Grand Central Dispatch API's and it handles the work of doing the appropriate amount of work. This is also not just in Cocoa, anything running on Mac OS X 10.6 Snow Leopard can take advantage of Grand Central Dispatch

(libdispatch) because it's included in libSystem.dylib and all you need to do is include #import <dispatch/dispatch.h> in your app and you'll be able to take advantage of Grand Central Dispatch.

155-How you attain the backward compatibility?

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.

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.

157.What is the difference between setting object = nil and [object release] ?

object = nil;
[object release]
Don't do that. You are sending a release message on a nil object that will just do nothing. But the object that was referenced by your object is still in memory because it has never received a release message.
[object release];
object = nil;

Here you release the object, and for convenience and security, you set nil to its reference. So you can call (by mistake of course :-)) any method on that object and the app won't crash.
But if you use a retained property @property(nonatomic, retain), calling :
self.object = nil;
equals to call :
[object release];
object = nil;

158.Subclass, Category and Extensions in Objective C

Today lets see what is subclassing, categories and extensions in Objective C, and where, when and how to use these concepts.

Note: A complete working Xcode project copy is available for download, which includes all the required examples to understand these concepts practically at the end of this post.

1) Subclass in Objective C

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

Suppose for example consider a UITextField class, by default the placeholder text of UITextField will be of light gray color with default system font. If we want to change this style just subclass UITextField and
override drawPlaceholderInRect method.

Example:
Create a class of type UITextField and name it some thing
like CustomUITextFieldPlaceholderAppearance

CustomUITextFieldPlaceholderAppearance.h
```
#import <UIKit/UIKit.h>
@interface CustomUITextFieldPlaceholderAppearance : UITextField
@end
```

CustomUITextFieldPlaceholderAppearance.m
```
#import "CustomUITextFieldPlaceholderAppearance.h"

@implementation CustomUITextFieldPlaceholderAppearance
```

```
// override drawPlaceholderInRect method
- (void)drawPlaceholderInRect:(CGRect)rect {
    // Set color and font size and style of placeholder text
    [[UIColor redColor] setFill]; //set placeholder text color to red
    [[self placeholder] drawInRect:rect withFont:
[UIFont fontWithName:@"verdana" size:14.0]]; //set custom font style and size
to placeholder text
}
@end
```

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

2) Categories in Objective C

An Objective C category allows you add your own methods to an existing class.

Categories are also called as "informal protocols".

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

Consider NSString Class and if suppose we want to add a reverse string method to NSString class, so that in our application at any point of time any NSString object can call this category method and get a reversed string as a result. We can do this as below,

Note: Usually naming convention for category file is like OriginalClassName +CategoryName

Example:
Lets create a category class with a name something like NSString +NSString_ReverseString

NSString+NSString_ReverseString.h

```
#import <Foundation/Foundation.h>

@interface NSString (NSString_ReverseString)
```

```
- (NSString *)reverseString:(NSString *)yourString;
@end
```

NSString+NSString_ReverseString.m

```
#import "NSString+NSString_ReverseString.h"

@implementation NSString (NSString_ReverseString)

- (NSString *)reverseString:(NSString *)yourString
{
    NSMutableString *reversedStr = [NSMutableString stringWithCapacity:
[yourString length]];

    [yourString enumerateSubstringsInRange:NSMakeRange(0,
[yourString length])
                               options:(NSStringEnumerationReverse |
 NSStringEnumerationByComposedCharacterSequences)
                            usingBlock:^(NSString *substring, NSRange substringRange
, NSRange enclosingRange, BOOL *stop) {
                                [reversedStr appendString:substring];
                            }];
    return reversedStr;
}
@end
```

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

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

Note that in a category you can't add an instance variable, since methods within a category are added to a class at runtime.

3) Extensions in Objective C

Extensions are similar to categories but the need of extension is different.
- Class extensions are often used to extend the public interface with additional private methods or properties for use within the implementation of the class.

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

The syntax to declare class extension looks like,

```
@interface ClassName()

@end
```

since no name is given in the parentheses, class extensions are often referred to as anonymous categories.

Note Extensions can add instance variables.

Example:

```
@interface ABCExtension()
@property NSObject *yourProperty;
@end
```

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

Example:

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

```
@interface ViewController ()
-(void)printName:(NSString *)name;
@end

@implementation ViewController
-(void)printName:(NSString *)name
{
   NSLog(@"%@",name);
}
```

In the above extension example printName: method is private to class ViewController, and cannot be accessed from outside the ViewController class. (even if you inherit since printName: is a private method their will not be any access to this method outside the class)

You can call this extension method only inside ViewController class, as below

```
[self printName:@"MyName"];
```

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

159.Is a delegate retained?

No, the delegate is never retained! Ever!

160.Outline the class hierarchy for a UIButton until NSObject.

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

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

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

162.Whats the NSCoder class used for?

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

163.Whats an NSOperationQueue and how/would you use it?

The NSOperationQueue class regulates the execution of a set of NSOperation objects. An operation queue is generally used to perform some

asynchronous operations on a background thread so as not to block the main thread.

164.Explain the correct way to manage Outlets memory

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

165.What happens when the following code executes?

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

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

166.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.

167.What is Automatic Reference Counting (ARC) ?

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

168.How many bytes we can send to apple push notification server.

256bytes.

169.How can we achieve singleton pattern in iOS?

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 includeNSFileManager, NSWorkspace, NSApplication, and, in UIKit,UIApplication. 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.

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

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

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

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

172.What is the purpose of UIWindow object?

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

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

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

174.Define view object.

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

175.What is App Bundle?

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

176.Name those classes used to establish connection b/w application to webserver?

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

177.Tell methods used in NSURLConnection

(1)Connection did receive Response

(2)Connection did recevice Data

(3)Connection fail with error

(4)Connection did finish loading.

178.By default which things are in the application?

iPhone applications by default have 3 things

1.main: entry point of application.

2.Appdelegate: perform basic application and functionality.

3.Window: provide uiinterface.

179.Name data base used in iphone?

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

180.What is the instance methods?

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

Declaration of instance method :

– (void)click me: (id)sender;

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

Click me is method name.

Id is data type which returns any type of object.

181.What is the class method?

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

@interface class name :ns object

{

```
}
```

+(class name *)new alloc:

-(int)total open

182.What is data encapsulation?

Encapsulation means that your data is "self-contained" and than only your objects can do changes to it.

Encapsulation prevents your data from being changed accidentally and therefore avoid the introduction of nasty bugs in your programs, specially in big projects.

Getters and setters are what make encapsulation elegant in most languages. The point of getters and setters is to explicitly write self-documented code that prevents accidental changes in your code. myObject.getSomeVar() and myObject.setSomeVar("foo") explicitly tell whoever is maintaining your code (which can and most of the time will include yourself) that your intentions are clear.

```
@interface Fraction{
  int _numerator;
  int _denominator;
}

-(int) numerator;
-(int) denominator;

@end


@implementation Fraction

–(int) numerator
{
return _numerator;
}

–(int) denominator
{
return _denominator;
```

```
}

//setters

@end
```

Then we can use this class in other classes to get the numerator / denominator of a fraction object:

```
//some other class
Fraction* fraction = [[Fraction alloc]init];
//set numerator / denominator
int fractionNumerator = [fraction numerator];
```

What we have done above is created a Fraction object and then called it's getNumerator method which returns an int. We capture this return value by assigning it to fractionNumerator.

183.What is Polymorphism?

The word polymorphism means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance. Objective-C polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

Consider the example, we have a class Shape that provides the basic interface for all the shapes. Square and Rectangle are derived from the base class Shape. We have the method printArea that is going to show about the OOP feature polymorphism.

```
#import <Foundation/Foundation.h>

@interface Shape : NSObject

{
   CGFloat area;
}

- (void)printArea;
- (void)calculateArea;
@end

@implementation Shape

- (void)printArea{
   NSLog(@"The area is %f", area);
```

```
}

- (void)calculateArea{

}

@end


@interface Square : Shape
{
    CGFloat length;
}

- (id)initWithSide:(CGFloat)side;

- (void)calculateArea;

@end

@implementation Square

- (id)initWithSide:(CGFloat)side{
    length = side;
    return self;
}

- (void)calculateArea{
    area = length * length;
}

- (void)printArea{
    NSLog(@"The area of square is %f", area);
}

@end

@interface Rectangle : Shape
{
    CGFloat length;
    CGFloat breadth;
}
```

```
- (id)initWithLength:(CGFloat)rLength andBreadth:(CGFloat)rBreadth;


@end

@implementation Rectangle

- (id)initWithLength:(CGFloat)rLength andBreadth:(CGFloat)rBreadth{
   length = rLength;
   breadth = rBreadth;
   return self;
}

- (void)calculateArea{
   area = length * breadth;
}

@end


int main(int argc, const char * argv[])
{
   NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
   Shape *square = [[Square alloc]initWithSide:10.0];
   [square calculateArea];
   [square printArea];
   Shape *rect = [[Rectangle alloc]
   initWithLength:10.0 andBreadth:5.0];
   [rect calculateArea];
   [rect printArea];
   [pool drain];
   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
2013-09-22 21:21:50.785 Polymorphism[358:303] The area of square is
100.000000
2013-09-22 21:21:50.786 Polymorphism[358:303] The area is 50.000000
```

In the above example based on the availability of the method calculateArea and printArea, either the method in the base class or the derived class executed.

Polymorphism handles the switching of methods between the base class and derived class based on the method implementation of the two classes.

184.Abstract class for Objective-C?

There are no abstract classes but you can produce something similar using a combination of a class and a protocol (which is similar to Java's interface). First divide up your abstract class into those methods you wish to provide default implementations for and those you require sub-classes to implement. Now declare the default methods in an @interface and implement them in an @implementation, and declare the required methods in an @protocol. Finally derive your sub-classes from class<protocol> - a class which implements the protocol. For example:

```
@interface MyAbstract

- (void) methodWithDefaultImplementation;

@end

@protocol MyAbstract

- (void) methodSubclassMustImplement;

@end

@implementation MyAbstract

- (void) methodWithDefaultImplementation { ... }

@end

@interface MyConcreteClass: MyAbstract<MyAbstract>
  ...
@end

@implementation MyConcreteClass

// must implement abstract methods in protocol
- (void) methodSubclassMustImplement { ... }

@end
```

If you are concerned over using the same name for a class and a protocol look at Cocoa where NSObject follows this pattern…

185.What are KVO and KVC?

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

You access property like

NSString age = emp.age;

setting property value.

emp.age = @"20″;

Now how KVC works is like this

[emp valueForKey:@"age"];

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

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

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

186.Can we use two tableview controllers on one view controller?

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

187.Swap the two variable values without taking third variable?

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

y=x-y;
        NSLog(@"Y Value==> %d",y);

x=x-y;
        NSLog(@"x Value==> %d",x);
```

188.What is responder chain?

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

189.the advantages and disadvantages about synchronous versus asynchronous connections.

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

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

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

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

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

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

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

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

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

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

• Parsing data on the fly is easy.

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

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

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

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

190.What is the navigation controller?

Navigation controller contains the stack of controllers every navigation controller

must be having root view controller by default these controllers contain 2 method

(a) push view (b) pop view

By default navigation controller contain “table view”.

191.What is the split view controller?

This control is used for ipad application and it contain proper controllers by default

split view controller contain root view controller and detail view controller.

192.Cocoa.

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

193.Frameworks that make Cocoa.

Appkit (Application Kit)

Foundation

194.Frameworks that make CocoaTouch.

UIkit (UserInterface Kit)

Foundation

195.Objective-C.

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

Dynamic typing—determining the class of an object at runtime

Dynamic binding—determining the method to invoke at runtime

Dynamic loading—adding new modules to a program at runtime

196.Dynamic and Static Typing.

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

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

197.Selectors

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

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

is equivalent to:

[friend gossipAbout:aNeighbor];

198.Class Introspection

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

[obj isMemberOfClass:someClass];

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

[obj isKindOfClass:someClass];

· The version of a class

[MyString version]

· Find the class of an Objective-C object

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

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

[obj1 class] == [obj2 class]

199.Formal vs informal protocol.

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

@interface NSObject (MyProtocol)

//someMethod();

@end

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

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

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

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

200.Optional vs required

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

@protocol MyProtocol

@optional

-(void) optionalMethod;

@required

-(void) requiredMethod;

@end

201.Memory Management

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

202.Copy vs assign vs retain

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

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

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


string = newString;


}
```

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

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

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

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

[newString retain];

[string release];

string = newString;

}
```

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

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

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

if(string!=newString){

[string release];
```

```
string = [newString copy];

}
        }
```

203.alloc vs new

"alloc" creates a new memory location but doesn't initializes it as compared to "new".

204.release vs pool drain

"release" frees a memory. "drain" releases the NSAutoreleasePool itself.

205.NSAutoReleasePool : release vs drain

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

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

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

206.autorelease vs release

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

Release: retain count is decremented at this point.

207.Autorelease Pool

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

208.Differnce between boxName and self.boxName.

boxName: Accessing directly.

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

209.Threads and how to use?

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

Two ways to create threads…

· detachNewThreadSelector:toTarget:withObject:

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

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

· Built-in communication system.

· An equivalent of "join()"

210.Threadsafe

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

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

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

211.Notification and Observers

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

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

212.Delegate vs Notification

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

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

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

213.Plist

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

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

214.Helper Objects

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

215.Differentiate Foundation vs Core Foundation

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

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

216.What is unnamed category.

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

i. Extend an existing class by adding functionality.

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

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

A class extension — @interface Foo() — is designed to allow you to declare additional private API — SPI or System Programming Interface — that is

used to implement the class innards. This typically appears at the top of the .m file. Any methods / properties declared in the class extension must be implemented in the @implementation, just like the methods/properties found in the public @interface.

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

Example:

Foo.h

@interface Foo:NSObject

@property(readonly, copy) NSString *bar;

-(void) publicSaucing;

@end

Foo.m

@interface Foo()

@property(readwrite, copy) NSString *bar;

– (void) superSecretInternalSaucing;

@end

@implementation Foo

@synthesize bar;

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

@end

217.Strong vs Weak

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

Delegates and outlets should be weak.

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

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

218.difference between udid and device token?

The device ID and UDID are different names for the same thing. The device token is used for a server to send notifications to that device using Apple's Push Notification service - it lets the APN server know which device you're talking about, but doesn't have any meaning outside that context.

219.how to check crash reports client in ios app?

Using Testflight

220.How to sort an array of strings alphabetically in objective c.

localizedCaseInsensitiveCompare

221.What is the difference between NSString and NSMutableString?

```
NSString *s = [[NSString alloc] initWithString:@"Hello"];

s = [s stringByAppendingString:@"World"];
```
and other code like this
```
NSMutableString *ms = [[NSMutableString alloc] initWithString:@"Hello"];

[ms appendString:@"World"];
```
Both of these, functionally, do the same thing except for one difference - the top code block leaks. -[NSString stringByAppendingString:] generates a new immutable NSString object which you then tell the pointer s to point to. In the process, however, you orphan the NSString object that s originally pointed to. Replacing the line as follows would get rid of the leak:
```
s = [[s autorelease] stringByAppendingString:@“World"];
```

222.How to check whether a substring exists in a string?

```
NSString *originalString;
NSString *compareString;
```

Let your string in stored in originalString and the substring that you want to compare is strored in compareString.

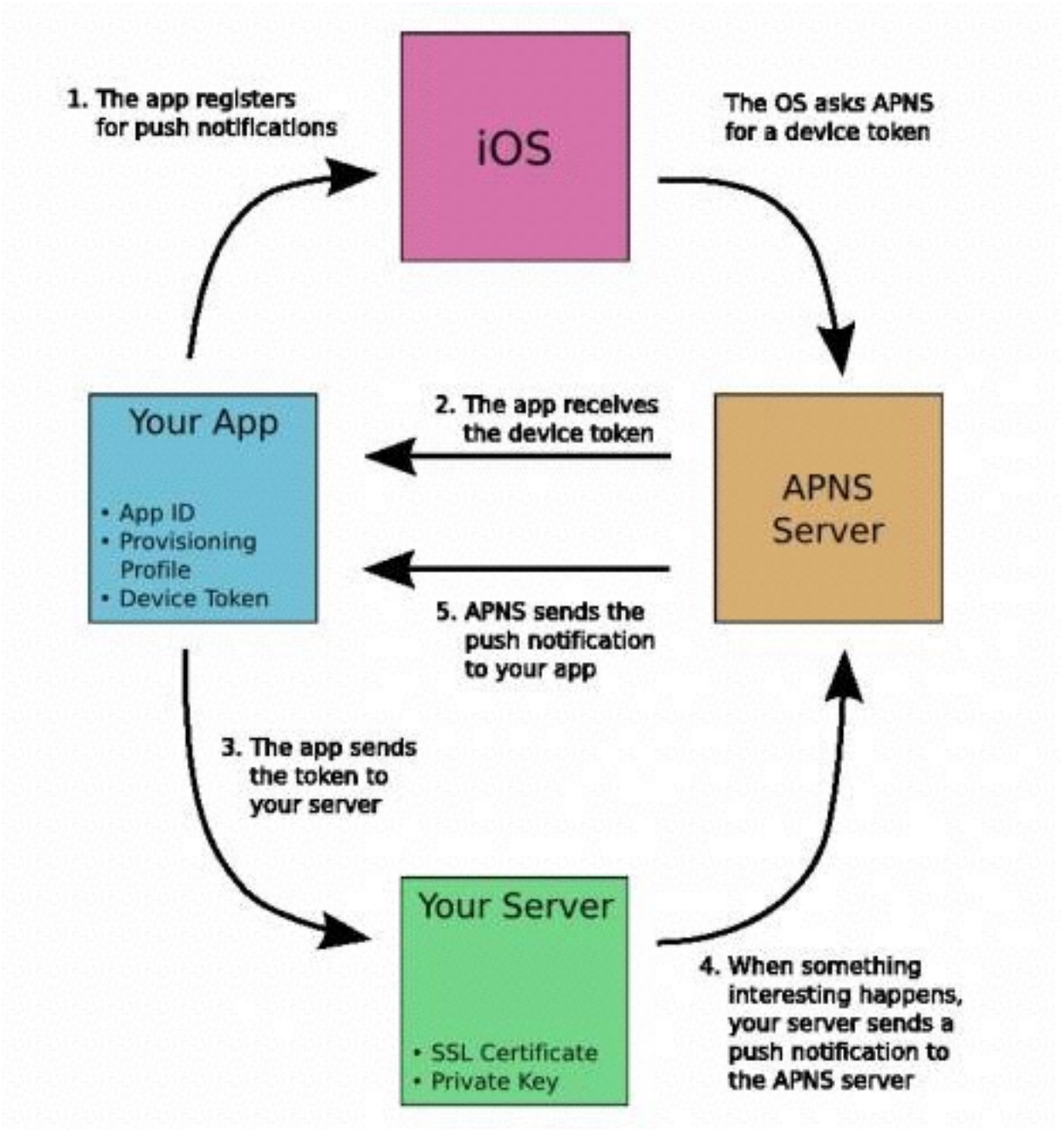
```
if ([originalString rangeOfString:compareString].location==NSNotFound)
 {
NSLog(@"Substring Not Found");

}
else
{
NSLog(@"Substring Found Successfully");
}
```

223.Apple Push Notification Services?

1 An app enables push notifications. The user has to confirm that he wishes to receive these notifications.
2 The app receives a “device token”. You can think of the device token as the address that push notifications will be sent to.
3 The app sends the device token to your server.
4 When something of interest to your app happens, the server sends a push notification to the Apple Push Notification Service, or APNS for short.
5 APNS sends the push notification to the user’s device.

When the user’s device receives the push notification, it shows an alert, plays a sound and/or updates the app’s icon. The user can launch the app from the alert. The app is given the contents of the push notification and can handle it as it sees fit.


1. The app registers for push notifications
iOS
The OS asks APNS for a device token
Your App
• App ID
• Provisioning Profile
• Device Token
2. The app receives the device token
APNS Server
5. APNS sends the push notification to your app
3. The app sends the token to your server
Your Server
• SSL Certificate
• Private Key
4. When something interesting happens, your server sends a push notification to the APNS server