Windows

a window object has several responsibilities:
=> It contains your application's visible content.
=> It plays a key role in the delivery of touch events to your views and other application objects.
=> It works with your application's view controllers to facilitate orientation changes.
A window is just a blank container for one or more views.

Tasks That Involve Windows
You can use your application's window object to perform a few application-related tasks.
=> Use the window object to convert points and rectangles to or from the window's local coordinate system.
=> Use window notifications to track window-related changes.

Creating and Configuring a Window

> Creating Windows in Interface Builder
1. Creating a window object using interface builder is simple, and you need to do the following:
=> To access the window at runtime, you should connect the window to an outlet, typically one defined in your application delegate or the File's Owner of the nib file.
=> If your retrofit plans include making your new nib file the main nib file of your application, you must also set the NSMainNibFile key in your application's Info.plist file to the name of your nib file. Changing the value of this key ensures that the nib file is loaded and available for use by the time the application:didFinishLaunchingWithOptions: method of your application delegate is called.

> Creating a Window Programmatically
1. If you prefer to create a window object programmatically, you should include code similar to the following in the application:didFinishLaunching:withOptions: method of your application delegate:

```
self.window = [[[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]] autorelease];
```

2. When creating windows, you should always set the size of the window

to the full bounds of the screen.

> Adding Content to Your Window
1. Each window object has a root view object that contains all of the other views representing your content. To install a new view to a window, use the addSubview: method.
[window addSubview:viewController.view];

2. You can use any view you want for a window's root view.

3. When configuring the root view of the window, you are responsible for setting and its initial size and position within the window.

> Changing the Window Level
Each window object has a configurable windowLevel property.that determines how that window is positioned relative to another windows.

Monitoring Window Changes
1. If you want to track appearance or disappearance of windows in your application, you can do so using these window-related notifications:
=> UIWindowDidBecomeVisibleNotification
=> UIWindowDidBecomeHiddenNotification
=> UIWindowDidBecomeKeyNotification
=> UIWindowDidResignKeyNotification

> Displaying Content on an External Display
1. To display content on an external display, you must create an additional window for your application and associate it with the screen object representing the external window.

2. The UIScreen class maintains a list of screen objects representing the available hardware display.

3. The process for displaying content on an external display is described in the following sections:
=> At application startup, register for the screen connection and disconnection notifications.
=> When it is time to display content on the external display, create and configure a window.
=> Show the window and update it normally.

> Handling Screen Connection and Disconnection Notifications

1. The important thing to remember about the connection and disconnection notifications is that they can come at any time, even when your application s suspended in the background.

2. Registering for screen connect and disconnect notifications

```
- (void)setupScreenConnectionNotificationHandlers
{
    NSNotificationCenter* center = [NSNotificationCenter defaultCenter];

    [center addObserver:self
selector:@selector(handleScreenConnectNotification:)
            name:UIScreenDidConnectNotification object:nil];
    [center addObserver:self
selector:@selector(handleScreenDisconnectNotification:)
            name:UIScreenDidDisconnectNotification object:nil];
}
```

3. Listing below shows how to create a secondary window and fill it with some content.

```
- (void)handleScreenConnectNotification:(NSNotification*)aNotification
{
    UIScreen*    newScreen = [aNotification object];
    CGRect       screenBounds = newScreen.bounds;

    if (!_secondWindow)
    {
        _secondWindow = [[UIWindow alloc] initWithFrame:screenBounds];
        _secondWindow.screen = newScreen;

        // Set the initial UI for the window.
        [viewController
displaySelectionInSecondaryWindow:_secondWindow];
    }
}

- (void)handleScreenDisconnectNotification:
(NSNotification*)aNotification
{
    if (_secondWindow)
    {
```

```
        // Hide and then delete the window.
        _secondWindow.hidden = YES;
        [_secondWindow release];
        _secondWindow = nil;

        // Update the main screen based on what is showing here.
        [viewController displaySelectionOnMainScreen];
    }

}
```

> Configuring a Window for an External Display
1. To display a window on an external screen, you must associate it with the correct screen object. This process involves locating the proper UIScreen object and assuaging it to the window's screen property.

2. Configuring a window for an external display

> Configuring the Screen Mode of an External Display
If you plan to use a screen mode other than the default one, you should apply that mode to the UIScreen object before associating the screen with a window.