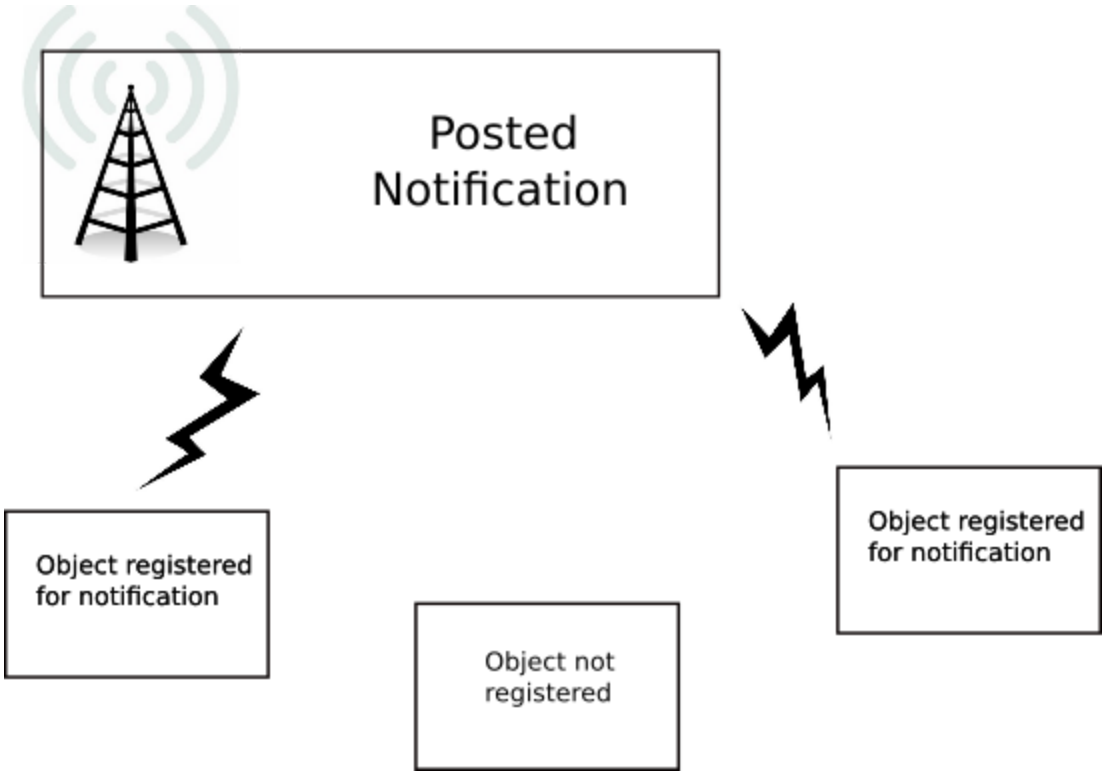
Stage 5.1 - Notifications

Scene 1: On-Set

Notifications are a mechanism for broadcasting a simple message throughout an application. A notification can be posted from anywhere, and any object instance can register for a notification that the message has been posted.

<MOTION>



Notifications work something like a radio station broadcasting on a specific frequency. Objects registered for a notification are like radios tuned to that frequency.

As the analogy suggests, coupling between notification posters and listeners is minimal. Unlike KVO, where observing objects must register with the object whose value they wish to observe, in the notifications system, no direct relationship is required.

Adding observers and posting notifications are both done through a singleton instance of NSNotificationCenter available through the NSNotification class method, +defaultCenter.

For a given notification, a poster has no idea if anyone is listening for their notification, and listeners have no idea how many if any posters may exist.

As with KVO, registering is done using a call to an -addObserver method, but the message is sent to the notification center, not a targeted instance.

</MOTION>
An object can stop receiving notifications at any time by sending a -removeObserver message to the notificationCenter.

NSNotifications allow a small amount of data to be piggybacked along with the notification message.
<KEYNOTE SLIDE bold type on slide>
One form of the post method:

- (void)postNotificationName:(NSString *)notificationName
object:(id)notificationSender
userInfo:(NSDictionary *)userInfo

allows the poster to append a user defined dictionary to the posted notification. This information is available to an observer through the userInfo property of the NSNotification parameter passed with the notifying callback.

</KEYNOTE SLIDE>.

<KEYNOTE SLIDE>
As I mentioned, notification registration is done with an -addObserver method call. There are two forms, one which specifies a selector callback, the other provides a block.

-addObserver:selector:name:object
-addObserverForName:object:queue:usingBlock:

The first form takes the following arguments:
observing object,
a method selector on that object
a notification name, which is an NSString identifying the type of notification
And a sender object which indicates that only notifications posted by a unique instance will be observed

An observing object and selector must be provided, but the notification name and/or the sender object may be *nil*.

If *notification name* is nil, the selector will be called when any notification is posted. If the sender object is nil, the selector will be called regardless of which object posts the notification. Setting both to *nil* indicates an interest in observing all notifications.

The second form takes the

- name of the notification
- the posting object
- an operation queue
- and a block to execute

Rather than a selector, a block to execute in response to a notification is provided.
The queue allows you to specify an nsoperation queue to which the block will be added.

All parameters except the block may be nil, which would specify that the block should be run on the posting thread in response to any object posting any notification.
</KEYNOTE>

As an experiment, let's try that.

Scene 2: Screencast

We'll create a project using the iOS single-view template, and call it Notif_Test.

We'll add an -addObserver:usingBlock call to -didfinishLaunching in the application delegate. For every parameter, except the block, will be set to nil.

The block will contain only a log statement which will display the name and userInfo properties of the NSNotification input parameter to the block.

1. Running the project, we see 20 or so notifications that our application uses internally. Most are UIWindow state and UIDevice orientation notifications that most applications will have little interest in, but the experiment makes it clear that Cocoa-Touch makes extensive use of NSNofications internally. When using the notification system, you should not assume that your notifications are the only ones being posted.

Code challenge

1. Write a call to the default nsnotification center to post a notification named "Swizzle". The notification sender should be "self".

Ans:

```
[[NSNotificationCenter defaultCenter] postNotificationName:@"Swizzle" object:self];
```

2. Using the same sender and notification center, post a notification named "Tralfaz" accompanied by a user dictionary with the value "dog" stored for the key "pet"

Ans:

```
[[NSNotificationCenter defaultCenter] postNotificationName:@"Tralfaz" object:self
userInfo:@{@"pet": @"dog"}];
```

3. Using the block form of the call, add an observer for the "Tralfaz" notification which will display the accompanying "pet" information.

Ans:

```
[[NSNotificationCenter defaultCenter] addObserverForName:@"Tralfaz" object:nil queue:nil
usingBlock:^(NSNotification *notification){
NSLog(@"notification.userInfo.pet");
}
```