Tema 8

Multitasking

Introducción

- Muchas de las aplicaciones que pasan al estado *background* se suspenden y no ejecutan ningún código.

- Con *iOS6* hay distintas opciones disponibles para ejecutar código en *background* definiendo cada una de ellas distintos comportamientos.

Preparando las aplicaciones

- Las apps compiladas con iOS4 o superior incluyen automáticamente el soporte multitarea, incluyendo los métodos que definen los comportamientos entre estados.

- De todas maneras, se requiere la implementación de cierto código para asegurar la correcta transición entre estados.

Comprobando disponibilidad

- La multitarea no está disponible en todos los sistemas iPhone OS.
- Si el dispositivo no es >iOS4 o el hardware no es capaz de ejecutar aplicaciones en *background*, el sistema vuelve al comportamiento anterior.

Comprobando disponibilidad

- Las aplicaciones deben estar preparadas para entornos no multitarea.

- Uso de la propiedad *multitaskingSupported*

```
UIDevice* device = [UIDevice currentDevice];
BOOL backgroundSupported = NO;
if ([device respondsToSelector:@selector(isMultitaskingSupported)])
   backgroundSupported = device.multitaskingSupported;
```

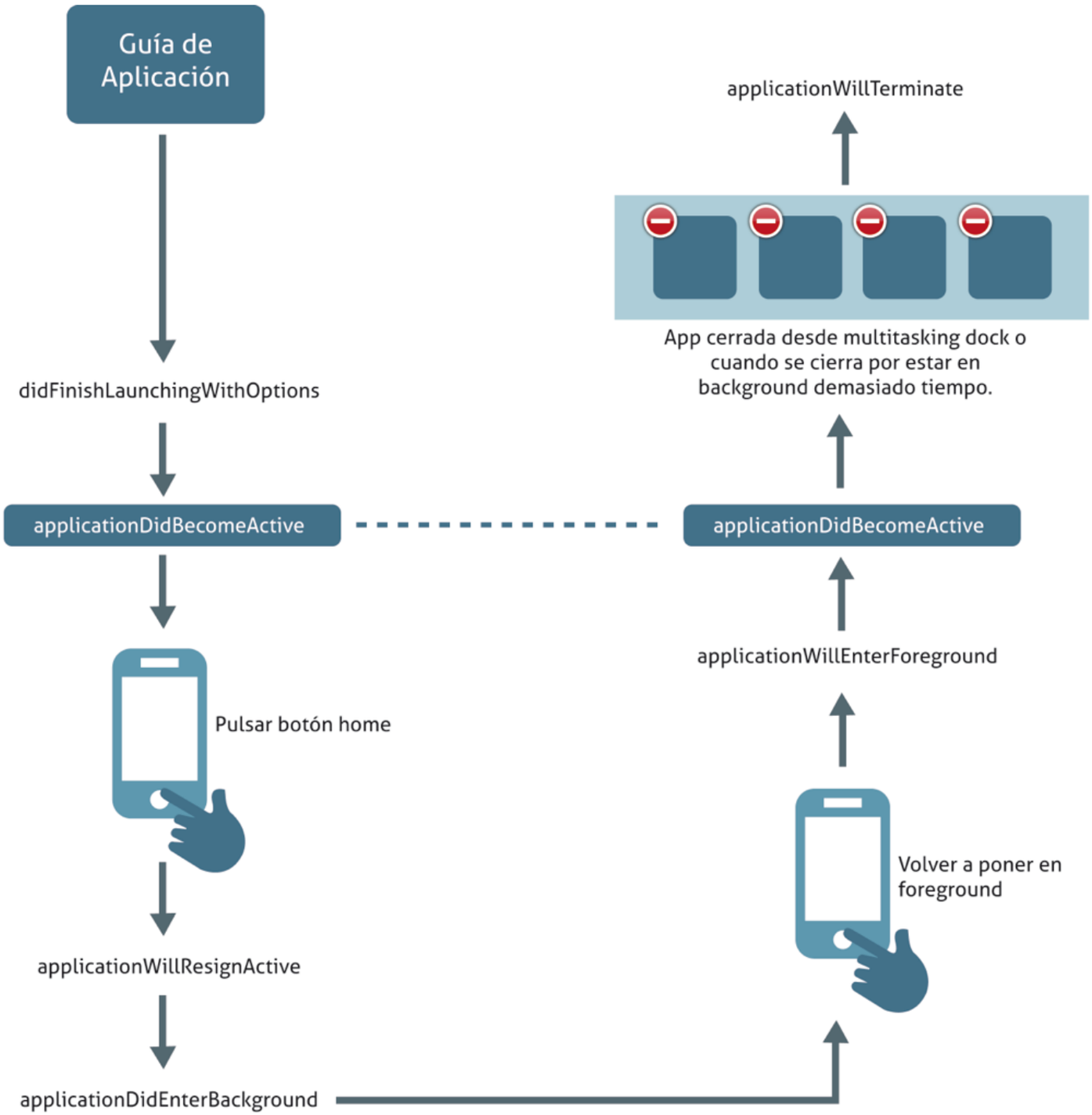
Declarando las tareas en background

- Incluir *UIBackgroundModes* en *Info.plist*
- Puede tomar por separado o juntos, los siguientes valores:
  - *audio*: reproducir sonido en *background*.
  - *location*: mantener al usuario al tanto de su posición.
  - *voip*: llamadas usando la conexión a Internet

Transición entre estados

- El Application Delegate debe implementar los siguientes métodos para describir los comportamientos ante cambios de estado:
  - *applicationDidFinishLaunchingWithOptions*
  - *applicationDidBecomeActive*
  - *applicationWillResignActive*
  - *applicationDidEnterBackground*
  - *applicationWillEnterForeground*
  - *applicationWillTerminate*

Transición entre estados



applicationDidFinishLaunchingWithOptions

- Este método se ejecuta sólo en una ocasión, cuando la aplicación se ejecuta y termina de hacer las cargas iniciales.

- Aquí debemos incluir el código de carga de los objetos necesarios durante la aplicación.

applicationDidBecomeActive

- Este método se ejecuta cada vez que la aplicación pasa a estar activa. Ocurrirá la primera vez que se ejecute y cada vez que vayamos de background a foreground.

- Aquí se suele poner el código relativo al retorno de background (como actualizar una interfaz con los datos que hemos recogido cuando no estaba activa).

applicationWillResignActive

- Se invoca a este método cuando pasamos de *foreground* a *background* y la aplicación deja de estar activa. Es el primer paso, justo antes de que se pase totalmente a *background*.

- Aquí podemos descargar las vistas o los objetos que ocupen mucha memoria, para asegurarnos que nuestra app sigue viva en *background*.

applicationDidEnterBackground

- Este método se ejecuta cuando la aplicación pasa totalmente a *background*.
- Ejecutaremos los métodos que deseemos que sigan funcionando en segundo plano.

applicationWillEnterForeground

- Este método se ejecuta cuando la aplicación pasa de *background* a *foreground*.

- Debemos preparar nuestra aplicación para que esté activa (recuperando los objetos que hubiéramos liberado).

applicationWillTerminate

- Se puede invocar a este método de dos formas. Si la aplicación no puede pasar a modo background o cuando cerramos la aplicación desde la barra de aplicaciones.

- Debemos almacenar el estado de la aplicación si deseamos que se restaure al volver a iniciar o mandar los datos a un servidor para que se sincronicen.

A tener en cuenta...

- El código que se ejecuta en *background* está más limitado que el que se ejecuta en *foreground*.

- Hay ciertas reglas que hay que seguir cuando programamos aplicaciones multitarea.

- Si no se siguen la aplicación termina su ejecución inmediatamente.

Guía de programación (1/3)

- No hacer llamadas a OpenGL ES.

- Cancelar todos los servicios Bonjour antes de suspender la aplicación.

- Preparar la aplicación para manejar los problemas de conexión en *background*.

- Salvar el estado de la aplicación antes de en enviar la aplicación a *background*.

Guía de programación (2/3)

- Liberar toda la memoria que no se necesite antes de enviar la aplicación a *background.*

- Dejar de usar los recursos compartidos antes de suspender.

- Evitar actualizar las ventanas y vistas de la aplicación en *background.*

Guía de programación (3/3)

- Contestar a las notificaciones de conexión y desconexión.

- Liberar los recursos de las alertas activas cuando se mueva la aplicación a *background.*

- Eliminar información “sensible” de las vistas antes de mover la aplicación a *background.*

- Mínima cantidad de código en *background.*

Iniciando tareas en background

- Los pasos a seguir dependen completamente de la tarea en cuestión.
- Unas se inician explícitamente y otras de forma automática.
- A continuación explicaremos algunas de ellas.

Tareas intensivas en tiempo (1/3)

- *beginBackgroundTaskWithExpirationHandler:* este método pide al sistema tiempo extra para completar una tarea larga. Para finalizar la tarea: *endBackgroundTask.*

- Para conocer el tiempo restante para que una aplicación acabe, disponemos de la propiedad *backgroundTimeRemaining* del objeto *UIAplication.*

Tareas intensivas en tiempo (2/3)

- Estas técnicas se usan sobre todo cuando se cierra abruptamente una aplicación que necesita más tiempo para acabar su ejecución:
  - Descarga de un archivo importante.
  - Tareas de configuración.
  - Salvar información, etc.

Tareas intensivas en tiempo (3/3)

```
-(void)applicationDidEnterBackground:(UIApplication *) application {
    UIApplication* app = [UIApplication sharedApplication];

//Pedir permiso para ejecutar en background. Proveer de un manejador por
//si la tarea requiere más tiempo



    NSAssert (bgTask == UIBackgroundTaskInvalid, nil);



    bgTask = [app beginBackgroundTaskWithExpirationHandler:^{
        dispactch_async(dispatch_get_main_queue(), ^{
                if (bgTask != UIBackgroundTaskInvalid) {
                        [app endBackgroundTask:bgTask];
                        bgTask = UIBackgroundTaskInvalid;
                }
        })
    }];

    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
        dispatch_async(dispatch_get_main_queue(), ^{
                if (bgTask != UIBackgroundTaskInvalid) {
                        [app endBackgroundTask:bgTask];
                        bgTask = UIBackgroundTaskInvalid;
                }
        });
    });
}
```

Programando la entrega de notificaciones locales (1/3)

- El procedimiento sería el siguiente:
  - Crear una instancia de *UILocalNotification*
  - Configurar dicha instancia y programarla usando los métodos de *UIApplication.*
- Los tipo de notificación a entregar son:
  - *sound*, *alert*, *badge*.

Programando la entrega de notificaciones locales (2/3)

- Las aplicaciones pueden programar hasta 128 notificaciones simultáneas, cada una de las cuales pueden ser configuradas para repetirse en un intervalo especificado.

- El siguiente ejemplo muestra una alarma con un archivo de sonido y una alerta que se ejecuta tanto si la aplicación no está ejecutándose como si está en *background*.

Programando la entrega de notificaciones locales (3/3)

```
- (void) scheduleAlarmForDate: (NSDate *) theDate {
   UIApplication* app = [UIApplication sharedApplication];
   NSArray* oldNotifications = [app scheduledLocalNotifications];

   if ([oldNotifications count] > 0)
        [app cancelAllLocalNotifications];

   UILocalNotification* alarm = [[UILocalNotification alloc] init];

   if (alarm) {
        alarm.fireDate = theDate;
        alarm.timeZone = [NSTimeZone defaultTimeZone];
        alarm.repeatInterval = 0;
        alarm.soundName = @"alarmsound.caf";
        alarm.alertBody = @"Time to wake up!";

        [app scheduleLocalNotification:alarm];
   }
}
```

Recibiendo eventos de localización en background (1/3)

- Nuevas opciones a la hora de recibir eventos de localización:
  - Las aplicaciones pueden registrarse solo a cambios significantes de localización.
  - Pueden seguir utilizando los servicios de localización estándar.
  - Autodeclaración como aplicación de localización continuamente en *background*.

Recibiendo eventos de localización en background (2/3)

- Si la aplicación no necesita una localización exacta, se recomienda utilizar la primera de las opciones anteriores. Ahorro significante de batería.

- Se activa de la siguiente manera:

```
CLLocationManager *locManager = [[[CLLocationManager] alloc] init ];
[locManager startMonitoringSignificantLocationChanges];
```

Recibiendo eventos de localización en background (3/3)

- Otra característica importante es la posibilidad de relanzar y despertar aplicaciones.

- Para las aplicaciones de uso intensivo de la localización (sistemas navegación), tenemos la tercera opción. Penaliza el consumo de energía. Conviene evitar esta opción en la medida de lo posible.

Reproduciendo audio en background

- Se indica en *Info.plist* introduciendo la cadena "*audio*" en la variable *UIBackgroundModes.*

- Cuando la aplicación este en *background* hay que limitar que la aplicación haga más trabajo del necesario.

  - Ej: Una aplicación de streaming de audio.

Implementando una aplicación VoIP

- Se indica en *Info.plist* introduciendo la cadena "*voip*" en la variable *UIBackgroundModes.*

- Configuración del *socket* para uso VoIP.

- Invocar el método *setKeepAliveTimeout: handler:* para especificar la frecuencia con la que despertar la aplicación para el correcto funcionamiento del servicio.

Notificaciones

Una notificación encapsula información sobre un evento en un objeto de la clase **NSNotification.**

- Cuando el evento sucede, el **notification center,** o centro de notificaciones, avisa inmediatamente a los objetos que tiene registrados, llamados observadores.

- Para la comunicación entre objetos se suele utilizar el paso de mensajes, mediante la invocación de métodos. En estos casos, el objeto que envía el mensaje debe conocer al otro objeto, pero en ocasiones debemos comunicar dos objetos independientes. Para solucionar estos casos se hace uso de las notificaciones.

Notificaciones

Cualquier objeto puede enviar notificaciones o registrarse como observador en el centro de notificaciones para recibir notificaciones.

- El centro de notificaciones se encarga de emitir las notificaciones a los objetos registrados.
- El emisor, el objeto incluido en la notificación y el observador pueden ser el mismo objeto o completamente diferentes.
- Los objetos que envían notificaciones no necesitan conocer nada sobre los objetos que las reciben.
- Los objetos que reciben las notificaciones deben saber como mínimo el nombre de la notificación.

Notification Center

El notification center consiste en un objeto de la clase **NSNotificationCenter**.

- Para acceder al centro de notificaciones se emplea el método **defaultCenter** de la clase NSNotificationCenter.

- Las notificaciones se envían a los objetos registrados como observadores de forma síncrona. Para enviar notificaciones asíncronamente se necesita hacer uso de las colas de notificaciones o notification queues.

Observador

El observador es el objeto que recibe la notificación.

- Para registrarse en el notification center como observador se usa el método **addObserver:selector:name:object:**.
- Es posible registrar un mismo objeto para recibir más de una notificación.
- Para eliminar un objeto del notification center y no recibir más notificaciones se emplean los métodos **removeObserver:** o **removeObserver:name:object:**.

Envío de notificaciones

Para enviar notificaciones primero se deben crear mediante **notificationWithName:object:** o **notificationWithName:object:userInfo:**

- **Son inmutables** por lo que una vez creadas no pueden ser modificadas

- Para enviar una notificación al notification center se emplea el método **postNotification**:

Envío de notificaciones : Utilidad

El envío de notificaciones puede ser interesante cuando queremos notificar un evento pero no sabemos quién ni cómo lo va a recibir.

- Por ejemplo, se puede enviar una notificación al volver de background y que algún ViewController haga de observador de dicha notificación para realizar una acción cuando se lance.