

# Recording in Progress

This class is being recorded

Please turn off your video and/or video if you do not wish to be recorded

# CMSC436: Programming Handheld Systems

# Alarms

# Today's Topics

Alarms

AlarmManager APIs

Alarm Types

Example Application

# Alarms

Mechanism for sending Intents at some point in the future

Allows one application to make code execute, even when that application is no longer running

# Alarms

Once registered, Alarms remain active even if the device is asleep

Can configure Alarms to wake a sleeping device

Alarms are canceled on device shutdown/restart

# Alarm Examples

MMS - Retry Scheduler

Settings - Bluetooth Discoverable timeout

Phone - User Info Cache

# AlarmManager

You create and manage Alarms by interacting with the AlarmManager

Get a reference to the AlarmManager by calling the Context class'

```
getSystemService(Context.ALARM_SERVICE)
```



# Creating Alarms

```
// One-shot Alarm with inexact timing. If there is already an alarm  
// scheduled for the same IntentSender, that previous alarm will first  
// be canceled
```

```
open fun set(type: Int, triggerAtMillis: Long,  
             operation: PendingIntent?): Unit
```

# Creating Alarms

// One-shot Alarm with exact timing

```
open fun setExact(type: Int, triggerAtMillis: Long,  
                 operation: PendingIntent?): Unit
```

# Creating Alarms

// Repeating alarm with inexact timing

```
open fun setRepeating(type: Int, triggerAtMillis: Long,  
                    intervalMillis: Long, operation: PendingIntent?): Unit
```

# Alarm Types

Two degrees of configurability

How to interpret time

What to do if the device is sleeping when the Alarm fires

# Interpreting Time

Realtime - relative to system clock

Elapsed - relative to time since last boot up

# Sleeping Devices

When Alarm fires and device is asleep, can either

Wake up device now & deliver Intent

Wait to deliver Intent until device wakes up

# Alarm Type Constants

RTC\_WAKEUP

RTC

ELAPSED\_REALTIME

ELAPSED\_REALTIME\_WAKEUP

# PendingIntent

A description of an Intent and a target action to perform with it

Can be handed to other applications so that they can perform actions on your behalf at a later time

Key concern - proxy applications shouldn't perform operations that originating application can't



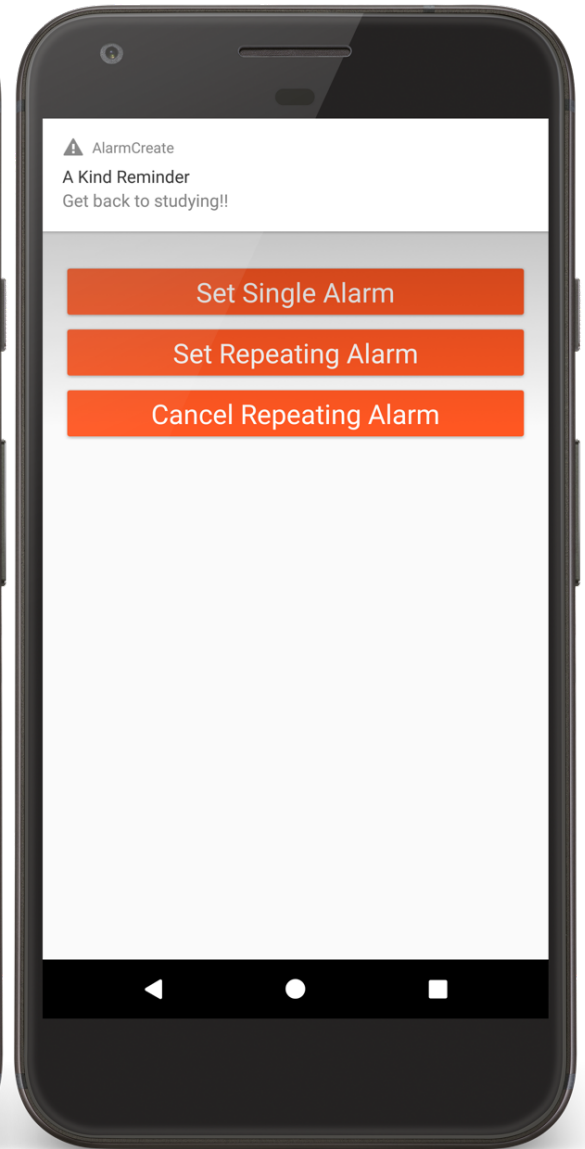
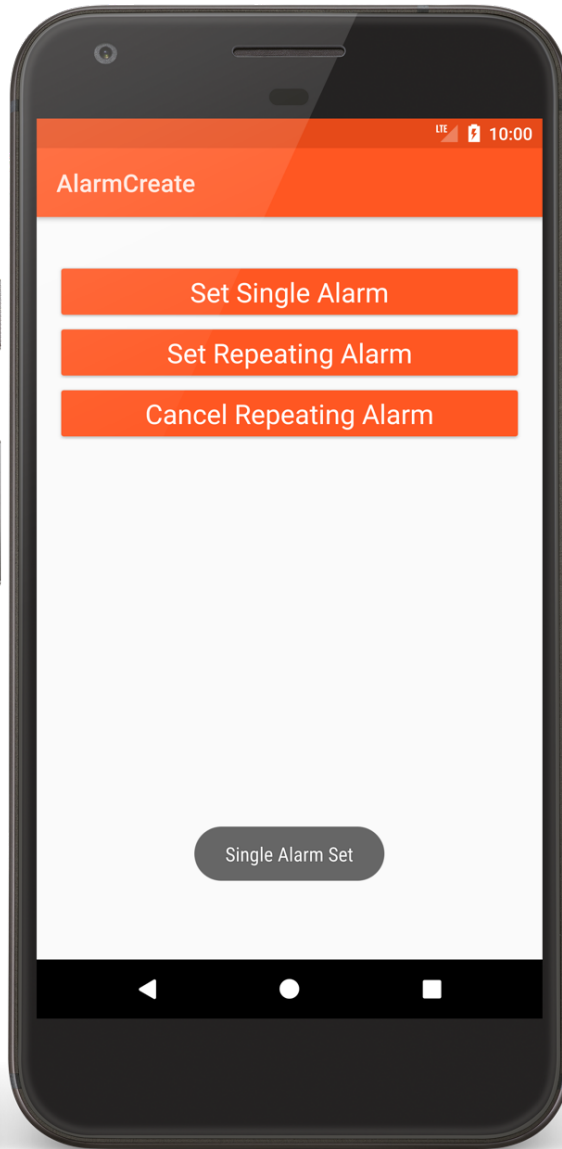
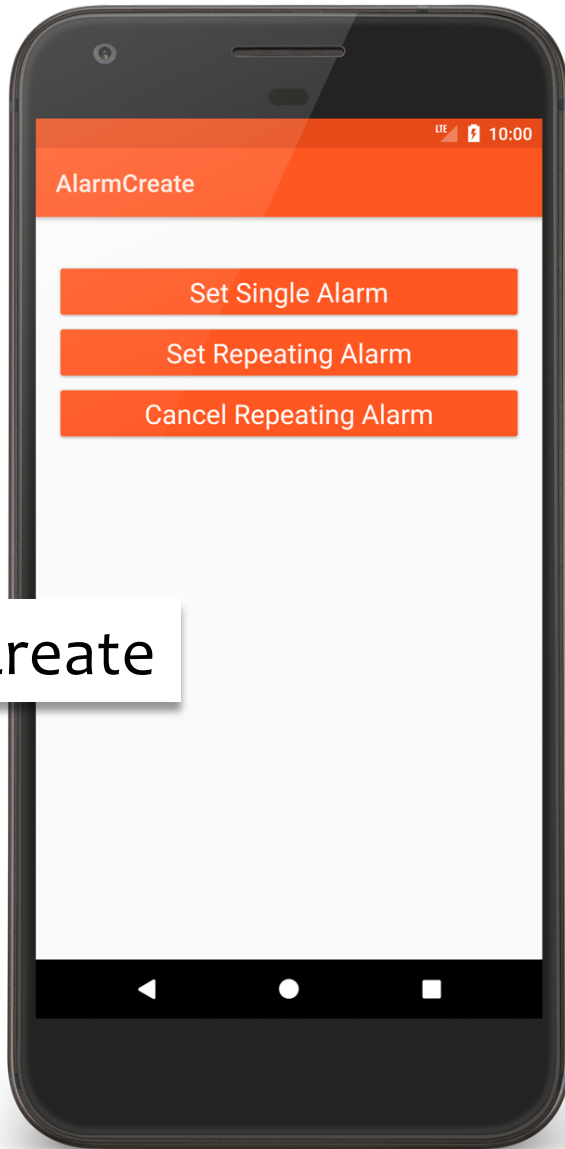
# PendingIntent

```
static fun getActivity(context: Context!, requestCode: Int,  
                      intent: Intent!, flags: Int): PendingIntent!
```

```
static fun getBroadcast(context: Context!, requestCode: Int,  
                       intent: Intent!, flags: Int): PendingIntent!
```

```
static fun getService(context: Context!, requestCode: Int,  
                     intent: Intent, flags: Int): PendingIntent!
```

# AlarmCreate



# AlarmCreateActivity.kt

```
public override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.main)

    // Get the AlarmManager Service
    mAlarmManager =
        getSystemService(Context.ALARM_SERVICE) as AlarmManager

    // Create an Intent to broadcast to the AlarmNotificationReceiver
    val mNotificationReceiverIntent = Intent(this@AlarmCreateActivity,
        AlarmNotificationReceiver::class.java)

    // Create an PendingIntent that holds theNotificationReceiverIntent
    mNotificationReceiverPendingIntent = PendingIntent.getBroadcast(
        this@AlarmCreateActivity, 0, mNotificationReceiverIntent, 0)
```

# AlarmCreateActivity.kt

```
// Create an Intent to broadcast to the AlarmLoggerReceiver
val mLoggerReceiverIntent = Intent(this@AlarmCreateActivity,
                                   AlarmLoggerReceiver::class.java)

// Create PendingIntent that holds the mLoggerReceiverPendingIntent
mLoggerReceiverPendingIntent = PendingIntent.getBroadcast(
    this@AlarmCreateActivity, 0, mLoggerReceiverIntent, 0)
}
```

# AlarmCreateActivity.kt

```
fun onClickCancelRepAlarmButton(v: View) {  
  
    // Cancel all alarms using mNotificationReceiverPendingIntent  
    mAlarmManager.cancel(mNotificationReceiverPendingIntent)  
  
    // Cancel all alarms using mLoggerReceiverPendingIntent  
    mAlarmManager.cancel(mLoggerReceiverPendingIntent)  
  
    // Show Toast message  
    Toast.makeText(applicationContext, "Repeating Alarms Cancelled",  
                  Toast.LENGTH_LONG).show()  
}
```

# AlarmLoggerReceiver.kt

```
class AlarmLoggerReceiver : BroadcastReceiver() {
    companion object {
        private const val TAG = "AlarmLoggerReceiver"
    }

    override fun onReceive(context: Context, intent: Intent) {
        // Log receipt of the Intent with timestamp
        Log.i(TAG, context.getString(R.string.logging_at_string) +
            DateFormat.getDateTimeInstance().format(Date()))
    }
}
```

# AlarmCreateActivity.kt

```
fun onClickSetSingleAlarmButton(v: View) {  
  
    // Set single alarm  
    mAlarmManager.set( AlarmManager.RTC_WAKEUP,  
        System.currentTimeMillis(), mNotificationReceiverPendingIntent)  
  
    // Set single alarm to fire shortly after previous alarm  
    mAlarmManager.set(AlarmManager.RTC_WAKEUP,  
        System.currentTimeMillis() + JITTER, mLoggerReceiverPendingIntent)  
  
    // Show Toast message  
    Toast.makeText(applicationContext, "Single Alarm Set",  
        Toast.LENGTH_LONG).show()  
}
```

# AlarmCreateActivity.kt

```
fun onClickSetRepAlarmButton(v: View) {  
  
    // Set repeating alarm  
    mAlarmManager.setRepeating(AlarmManager.ELAPSED_REALTIME,  
        SystemClock.elapsedRealtime(), REPEAT_INTERVAL,  
        mNotificationReceiverPendingIntent)  
  
    // Set repeating alarm to fire shortly after previous alarm  
    mAlarmManager.setRepeating(AlarmManager.ELAPSED_REALTIME,  
        SystemClock.elapsedRealtime() + JITTER, REPEAT_INTERVAL,  
        mLoggerReceiverPendingIntent)  
  
    // Show Toast message  
    Toast.makeText(applicationContext, "Repeating Alarm Set",  
        Toast.LENGTH_LONG).show()  
}
```



# Next Time

Threads, Messages and Handlers

# Example Applications

AlarmCreate