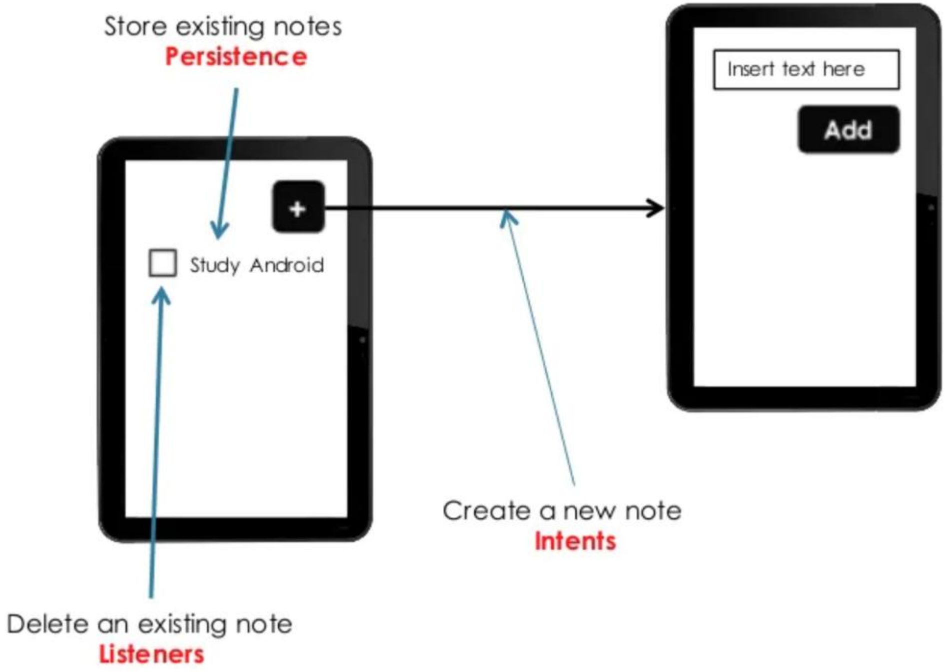
# Broadcast

Customized by Rasel

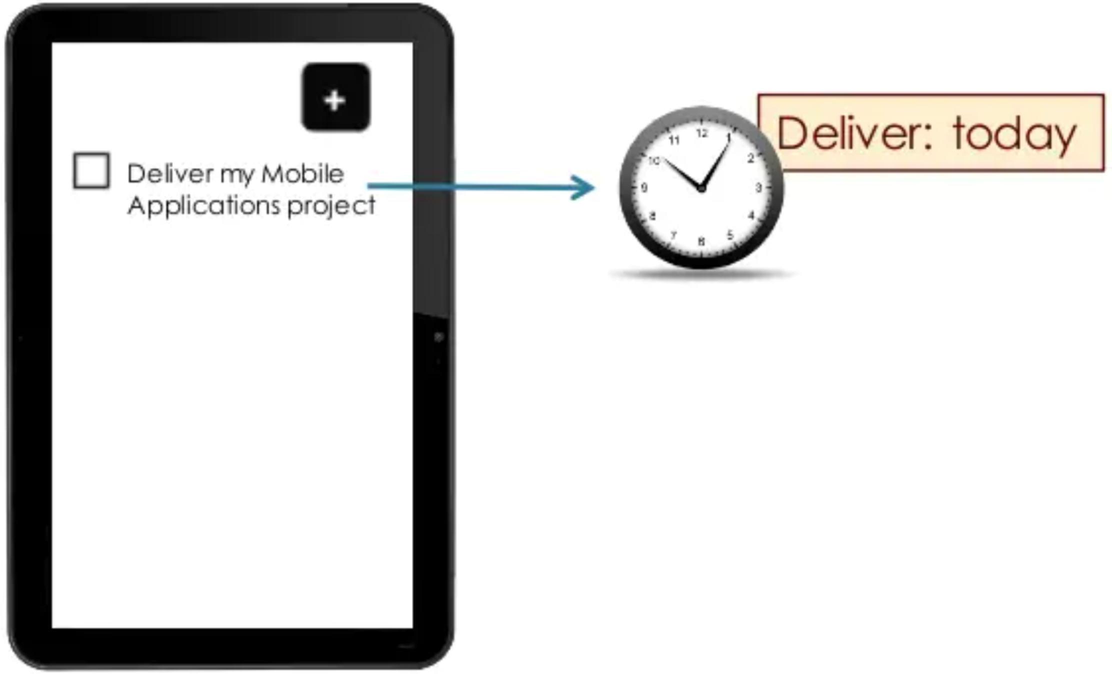
# Contents

- Broadcast intents
- Broadcast receivers
- Implementing broadcast receivers
- Custom broadcasts
- Security
- Local broadcasts

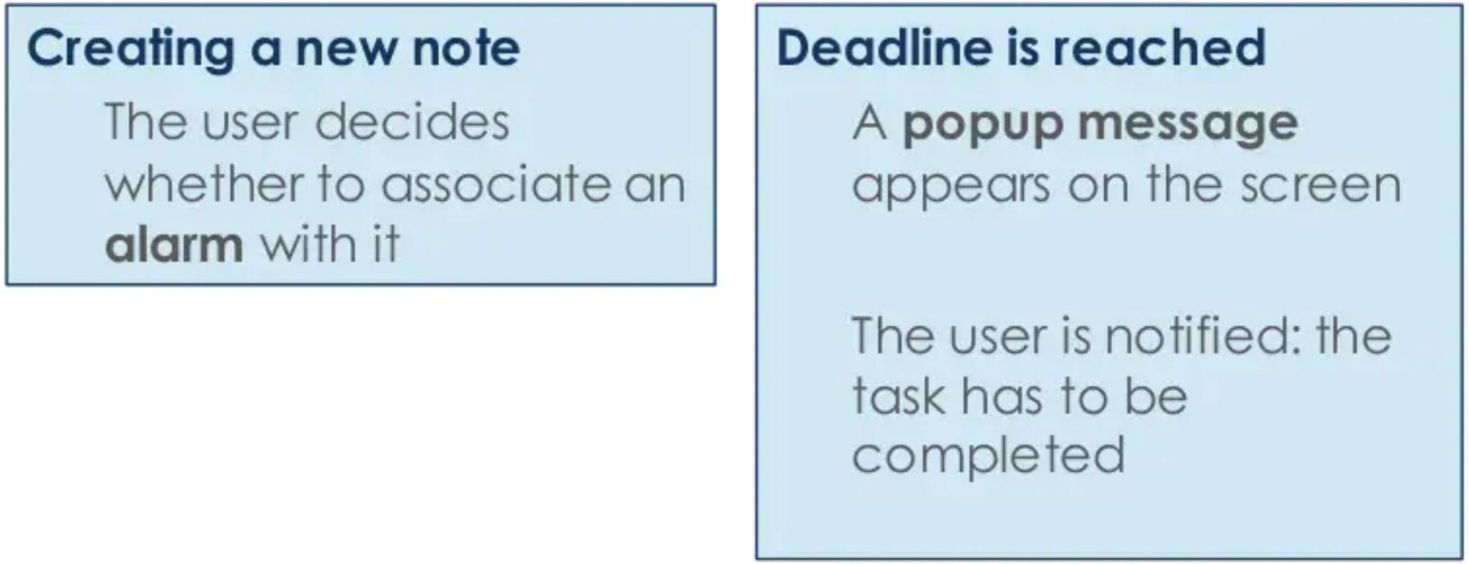
## App Feature: Take Note



## App Feature: Remind something in a specific time



## Merging both features: Reminder for a task



**Creating a new note**

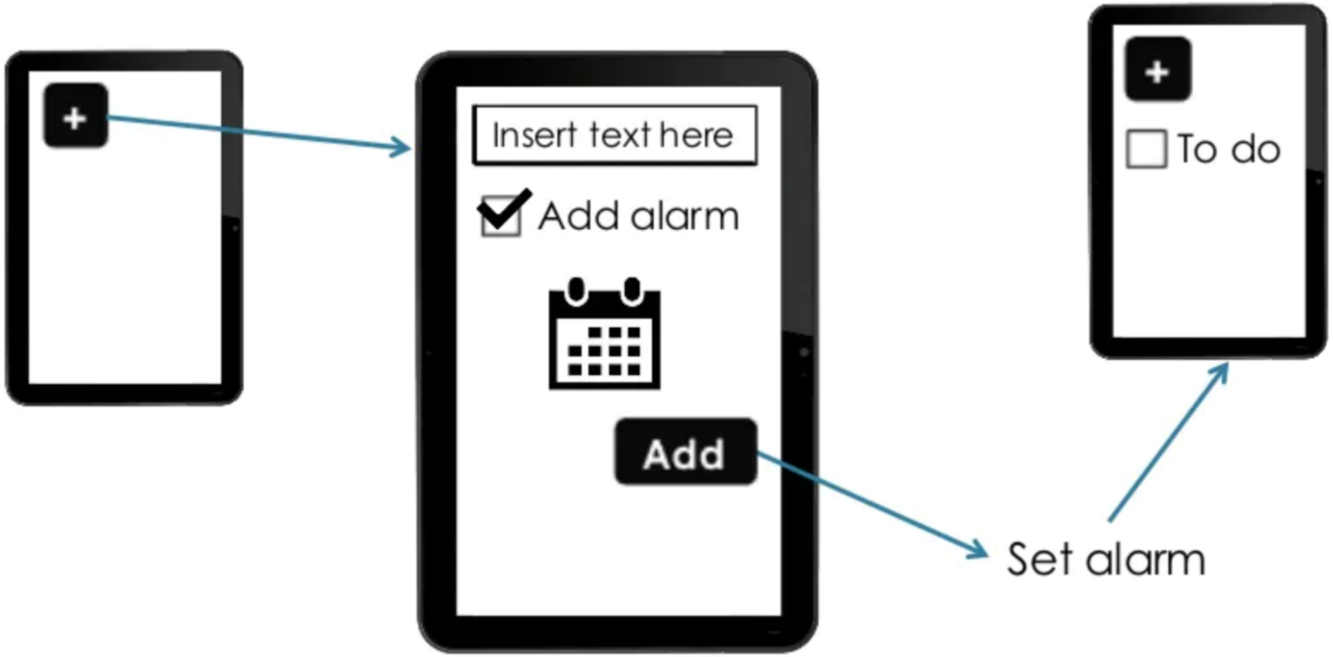
The user decides whether to associate an **alarm** with it

**Deadline is reached**

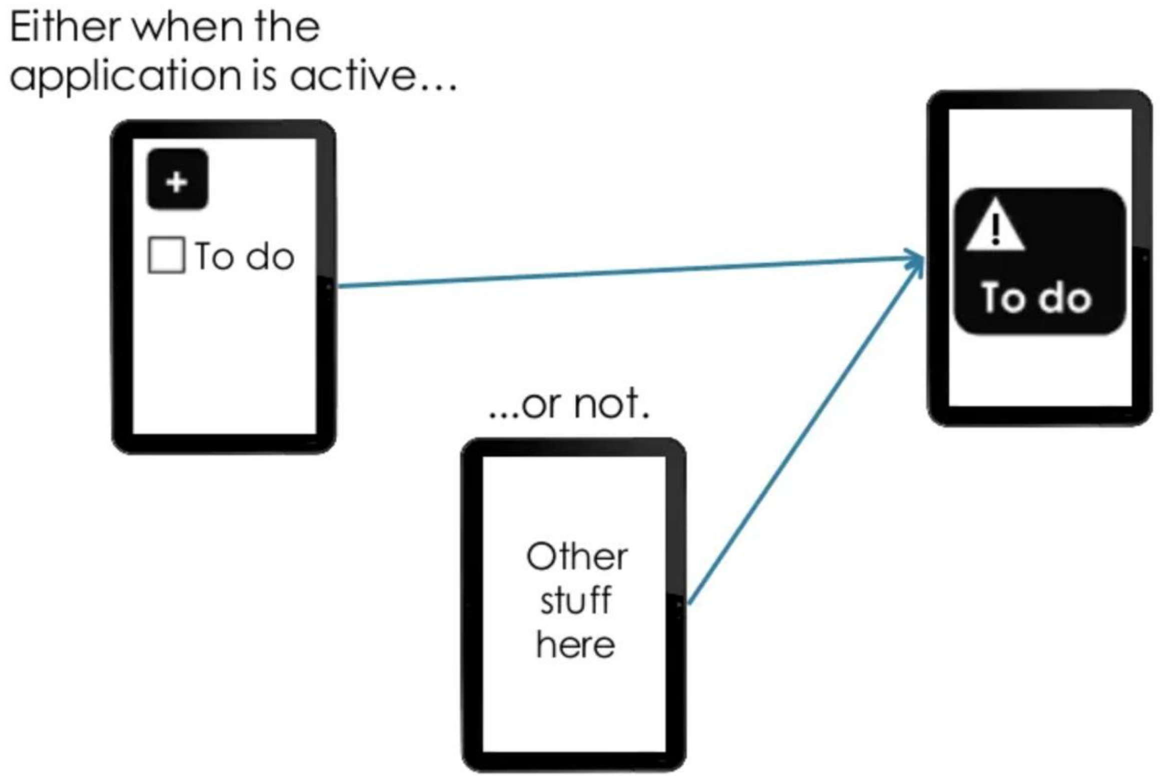
A **popup message** appears on the screen

The user is notified: the task has to be completed

## Add both taking note and reminder in same app

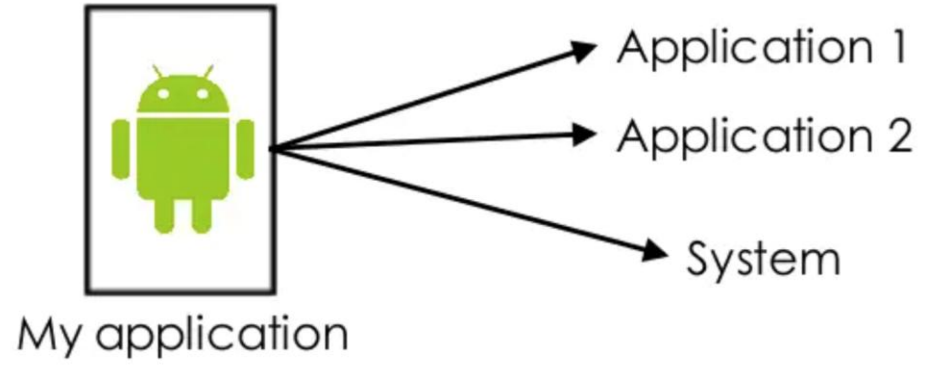


## Notify the User



## Using Intents to Broadcast Events

Intents are able to send messages **across process boundaries**



You can implement a **Broadcast Receiver** to listen for (and respond to) these broadcast messages

## Broadcast Intents

# Broadcast vs. Activity

Use implicit intents (with ACTION) to send broadcasts or start activities

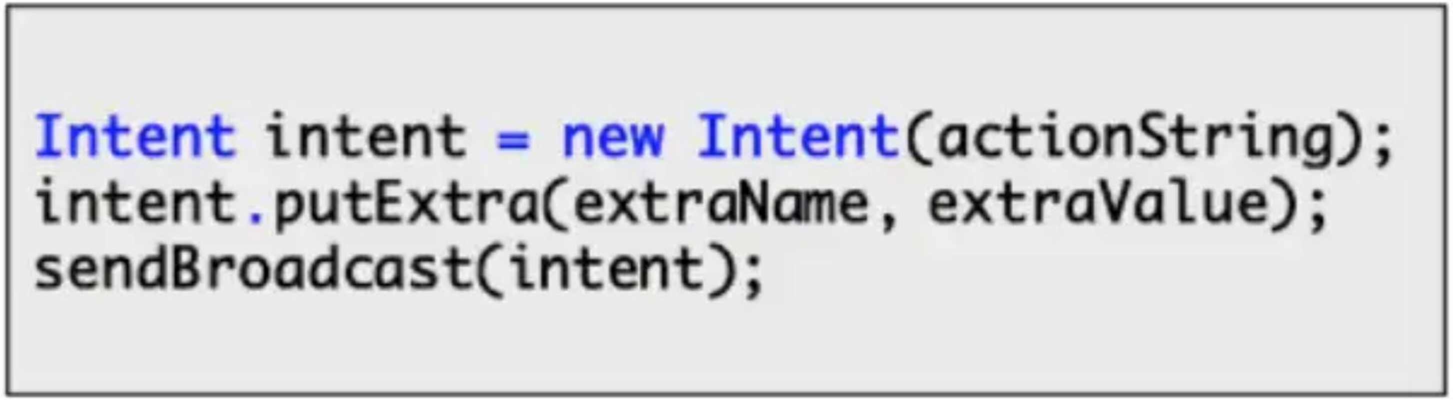
**Sending broadcasts**

- Use sendBroadcast()
- Can be received by any application registered for the intent
- Used to notify **all** apps about an event
    - Example?

**Starting activities**

- Use startActivity()
- Find a single activity to accomplish a task
- Accomplish a specific action

## Implicit Intent Declaration



```
Intent intent = new Intent(actionString);
intent.putExtra(extraName, extraValue);
sendBroadcast(intent);
```

# Broadcast Receivers

## What is a broadcast receiver?

- Listens for incoming intents initiated by system or using sendBroadcast()
    - Everything happens in the background
- Intents is sent
    - By the system, when an event occurs that might change the  behavior of an app
        - Changes in network activity (WiFi/Data connections)
        - Incoming Calls (Skype/Messenger/WhatsApp)
    - By another application, including your own

## Broadcast receiver always responds

- Responds even when your app is closed
- Independent from any activity
- When a broadcast intent is received and delivered to onReceive(), it has 5 seconds to execute, and then the receiver is destroyed
    - Execute?: Store something in DB, play ringtone, show notification, …

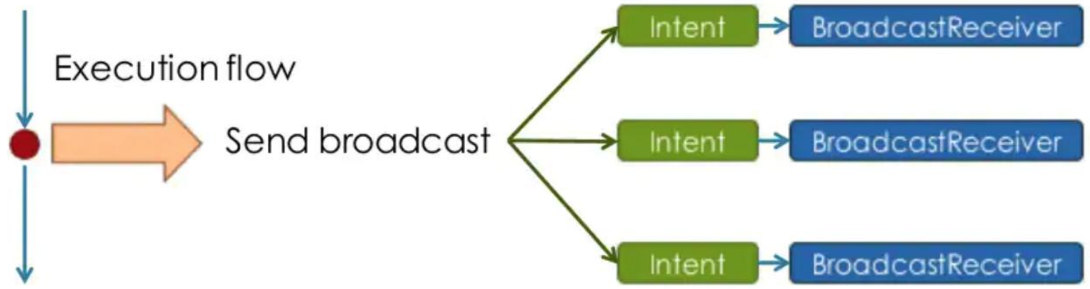
## System broadcasts

- Automatically delivered by the system when certain events occur - Examples
  - After the system completes a boot
    - android.intent.action.BOOT_COMPLETED
  - When the wifi state changes
    - android.net.wifi.WIFI_STATE_CHANGED

## Custom Broadcasts

- Deliver any custom intent from the app as a broadcast
  - sendBroadcast() method—asynchronous
  - sendOrderedBroadcast()—synchronously
  - Use custom action name. i.e. android.example.com.CUSTOM_ACTION

# sendBroadcast()

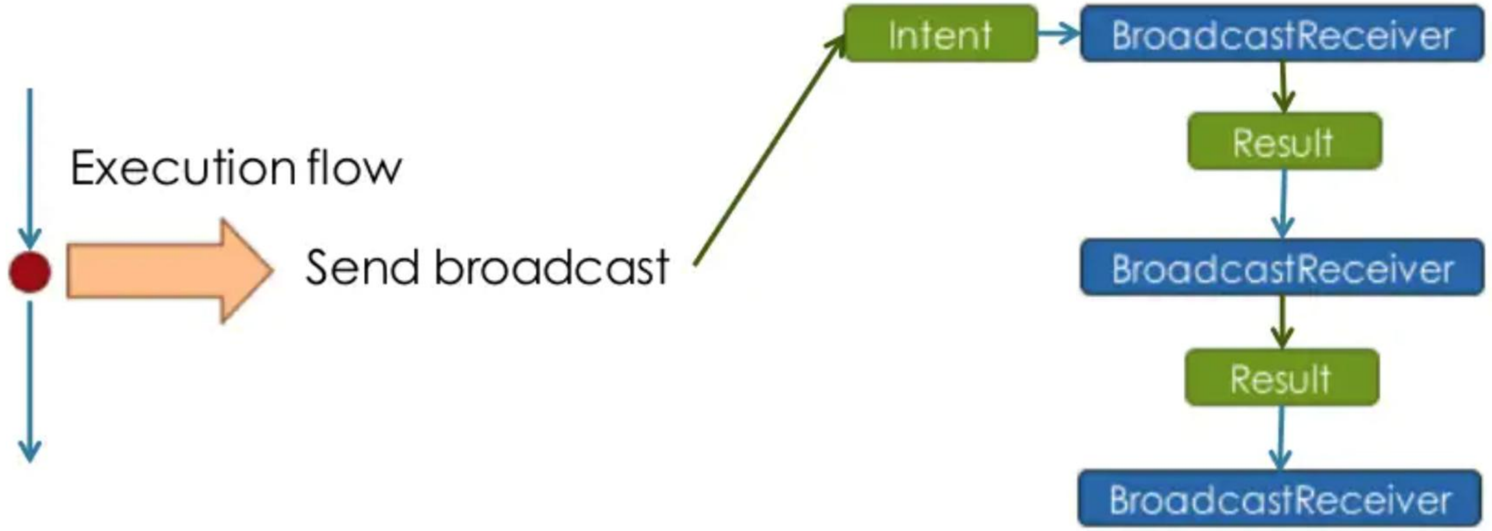


- All receivers of the broadcast are run in an undefined order
- Can be at the same time
- Efficient
- Use to send custom broadcasts

# sendOrderedBroadcast()

- Delivered to one receiver at a time
- Receiver can propagate result to the next receiver or abort the broadcast
- Control order with android:priority of matching intent filter
- Receivers with same priority run in arbitrary order

# sendOrderedBroadcast()



# Implementing Broadcast Receivers

# Steps for creating a broadcast receiver

1. Inherits BroadcastReceiver
2. Implement onReceive() method
3. Register to receive broadcast
    - Statically, in AndroidManifest
    - Dynamically, with registerReceiver()

## BroadcastReceiver

```
public class CustomReceiver extends BroadcastReceiver {
    public CustomReceiver() {
    }
    @Override
    public void onReceive(Context context, Intent intent) {
        // TODO: This method is called when the BroadcastReceiver
        // is receiving an Intent broadcast.
        //  Write code to perform some task here.
            // i.e. show notification, start service or ...
    }
}
```

## Register in Android Manifest

- <receiver> element inside </receiver>
- <intent-filter> registers receiver for specific intents

```
<receiver
    android:name=".CustomReceiver"
    android:enabled="true"
    android:exported="true">
        <intent-filter>
          <action android:name="android.intent.action.BOOT_COMPLETED" />
        </intent-filter>
</receiver>
```

# Available system intents/actions

- ACTION_TIME_TICK
- ACTION_TIME_CHANGED
- ACTION_TIMEZONE_CHANGED
- ACTION_BOOT_COMPLETED
- ACTION_PACKAGE_ADDED
- ACTION_PACKAGE_CHANGED
- ACTION_PACKAGE_REMOVED
- ACTION_PACKAGE_RESTARTED
- ACTION_PACKAGE_DATA_CLEARED
- ACTION_PACKAGES_SUSPENDED
- ACTION_PACKAGES_UNSUSPENDED
- ACTION_UID_REMOVED
- ACTION_BATTERY_CHANGED
- ACTION_POWER_CONNECTED
- ACTION_POWER_DISCONNECTED
- ACTION_SHUTDOWN

# Implement onReceive()

```
@Override
public void onReceive(Context context, Intent intent) {
    String intentAction = intent.getAction();
    switch (intentAction){
        case Intent.ACTION_POWER_CONNECTED:
            break;
        case Intent.ACTION_POWER_DISCONNECTED:
            break;
    }
}
```

# Custom Broadcasts

# Custom broadcasts

- Sender and receiver must agree on unique name for intent (action name)
- Define in activity and broadcast receiver

```
private static final String ACTION_CUSTOM_BROADCAST =
          "com.example.android.powerreceiver.ACTION_CUSTOM_BROADCAST";
```

## Send custom broadcasts

```
Intent customBroadcastIntent = new Intent(ACTION_CUSTOM_BROADCAST);

sendBroadcast(customBroadcastIntent);
```

# Register dynamically

- In **onStart()**
- Use registerReceiver() and pass in the intent filter
- Must unregister in onStop()

```
registerReceiver(mReceiver, mIntentFilter)
unregisterReceiver(mReceiver)
```

```
IntentFilter mIntentFilter = new IntentFilter(ACTION_CUSTOM_BROADCAST);
```

## Destroy!

```
@Override
protected void onStop() {
    super.onStop();
    LocalBroadcastManager.getInstance(this)
            .unregisterReceiver(mReceiver);
    }
```

## Local Broadcast Manager

# Local Broadcast Manager

- For broadcasts only in your app
- No security issues since no cross-app communication

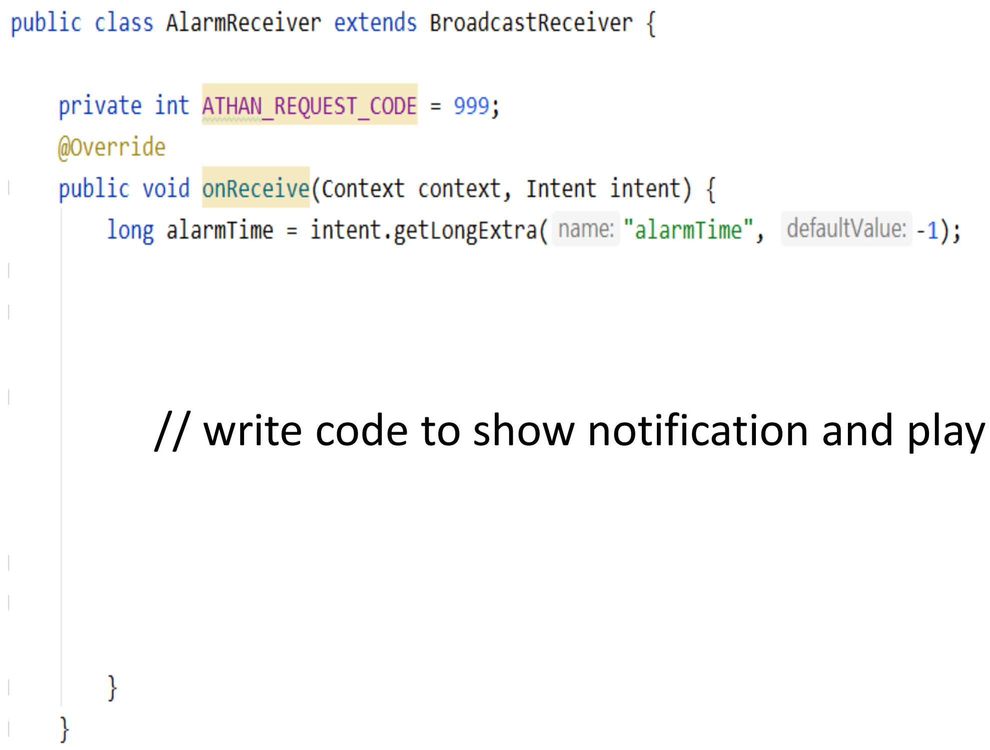
```
LocalBroadcastManager.sendBroadcast()

LocalBroadcastManager.registerReceiver()
```

## Register local broadcast manager

```
IntentFilter inFilter = new IntentFilter(ACTION_CUSTOM_BROADCAST);

LocalBroadcastManager.getInstance(this)
    .registerReceiver(mReceiver, inFilter);
```

# Pending Broadcast Intent

```
long aTime= System.currentTimeMillis() + 60*1000;

// BroadcastReceiver

Intent intent = new Intent(this,
AlarmReceiver.class);

Intent.putExtra(“alarmTime”, aTime);


// call broadcast using pendingIntent

pendingIntent = PendingIntent.getBroadcast(this, 0,
intent, 0);




alarmManager.set(AlarmManager.RTC_WAKEUP, aTime,
pendingIntent);
```



```
public class AlarmReceiver extends BroadcastReceiver {

    private int ATHAN_REQUEST_CODE = 999;
    @Override
    public void onReceive(Context context, Intent intent) {
        long alarmTime = intent.getLongExtra( name: "alarmTime",  defaultValue: -1);



                // write code to show notification and play music



        }
    }
```

## Security

## Security

- Receivers cross app boundaries
- Make sure namespace for intent is unique and you own it
- Other apps can send broadcasts to your receiver
    - use  permissions to control this
- Other apps can respond to broadcast your app sends
- Access permissions can be enforced by sender or receiver

## Controlling permission sender

- void sendBroadcast (Intent intent,
  String receiverPermission)

- Receivers must request permission with <uses-permission> in AndroidManifest.xml

# Controlling permission receiver

- registerReceiver(BroadcastReceiver, IntentFilter, **String**, android.os.Handler)
- or in <receiver> tag

- Users must request permission with <uses-permission> in AndroidManifest.xml for sending or receiving system broadcast

# END