Mobile Computing

COMP5216/COMP4216

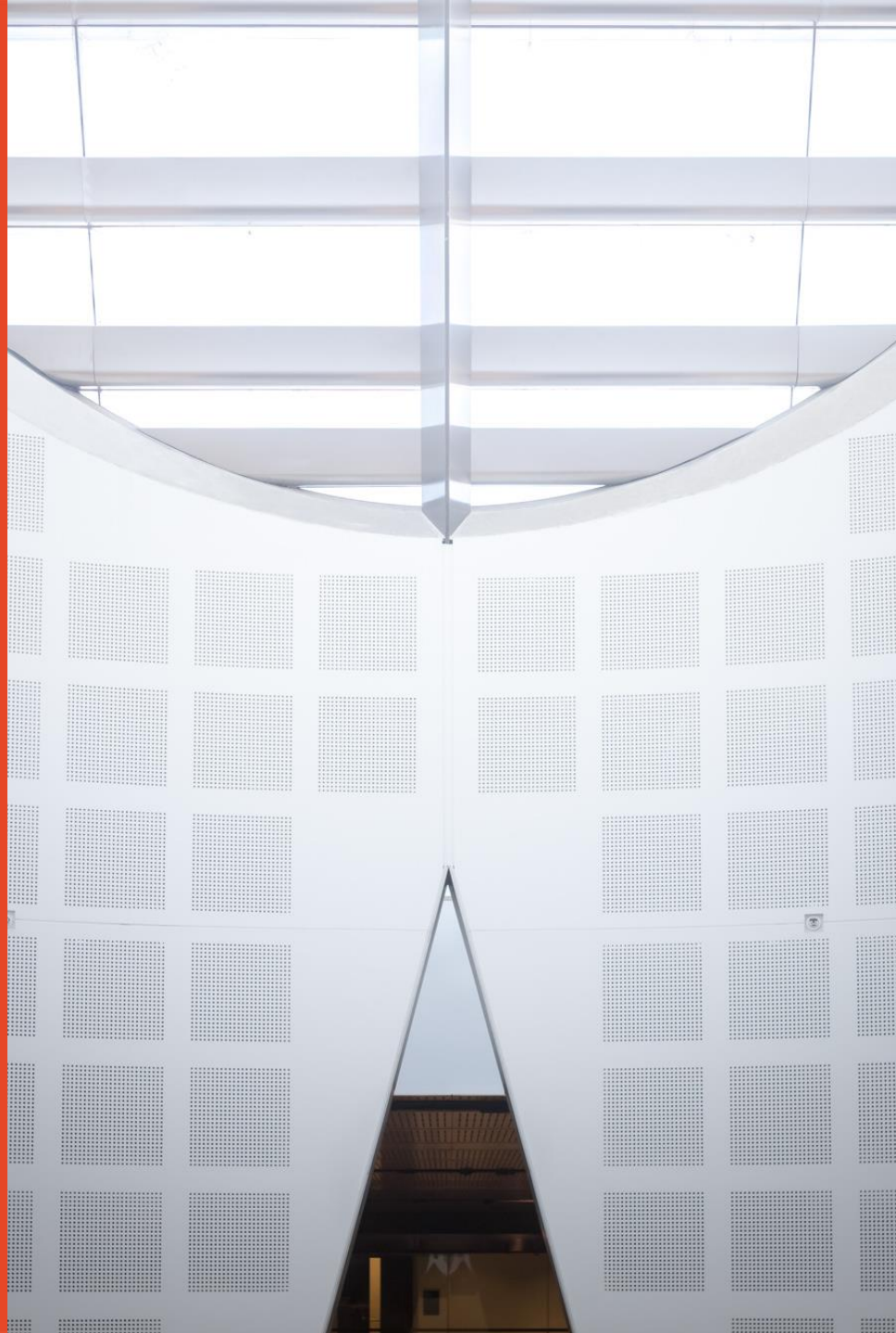
Week 04

Semester 2, 2023

Dr Thilina Halloluwa

School of Computer Science

THE UNIVERSITY OF SYDNEY



Muddy Cards- Week 3

THE UNIVERSITY OF SYDNEY

Inter-disciplinary Problems

- Learning about some other field, you'll probably see problems that software could solve.
  - (a) the inhabitants of that domain are not as likely as software people to have already solved their problems with software, and
  - (b) since you come into the new domain totally ignorant, you don't even know what the status quo is to take it for granted.

- Taking a class on, say, genetics; or better still, go work for a biotech company.
  - **One way to ensure you do a good job solving other people's problems is to make them your own.**

Problems

- Serve a small initial group
  - High demands → high competition
  - E.g. Apple, Google, Facebook, Microsoft, etc.
- Make the users who care about your product happy
  - First iPhone does not have copy/paste function
- Germ problems
  - Hard to tell, even experienced investors
  - Airbnb
    - Let hosts rent out space on their floors during conventions. They didn't foresee the expansion of this idea; it forced itself upon them gradually

Building blocks of Android - II

THE UNIVERSITY OF SYDNEY

Building blocks of Android

App components

- Activities
- **Broadcast Receivers**
- **Content Providers**
- **Services**

- Activating components – **Intent**

- **Android Developer Page**
  - https://developer.android.com

Recap Last Week

1. A single app can have multiple Activities. True or False ?
2. A single Activity can have multiple screens. True or False ?
3. An Intent can be used to start an Activity. True or False?

4. What type of Intent is used in the following code snippet?

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

5. An Intent must be declared in AndroidManifest.xml file. True or False?
6. What is the purpose of Intent Filters?

Broadcast Receiver

- System-wide events an app can consume and receive.
  - Can either be sent by the Android system itself or an app
- Many components need to know that some events have occurred.
  - New package installed.
  - Phone call received.
  - WiFi is connected.
  - Device is rebooted.
- Android uses a **Broadcast Intent** to tell everyone about it.
- All intents can be found at `BROADCAST_ACTIONS.TXT` file in the relevant SDK

- https://developer.android.com/guide/components/broadcasts

Broadcast Receiver

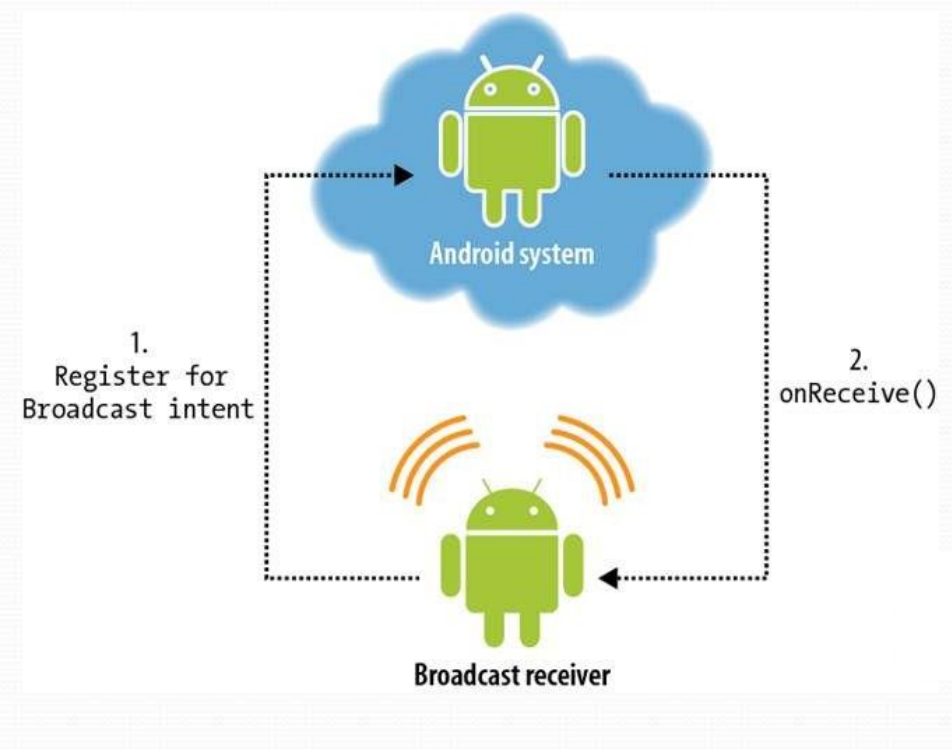
– Common broadcast intents

1. **android.intent.action.BATTERY_LOW** : Indicates low battery condition on the device.
2. **android.intent.action.BOOT_COMPLETED** : This is broadcast once, after the system has finished booting
3. **android.intent.action.CALL** : To perform a call to someone specified by the data
4. **android.intent.action.DATE_CHANGED** : The date has changed
5. **android.intent.action.REBOOT** : Have the device reboot
6. **android.net.conn.CONNECTIVITY_CHANGE** : The mobile network or wifi connection is changed(or reset)

Setting up a BroadcastReceiver

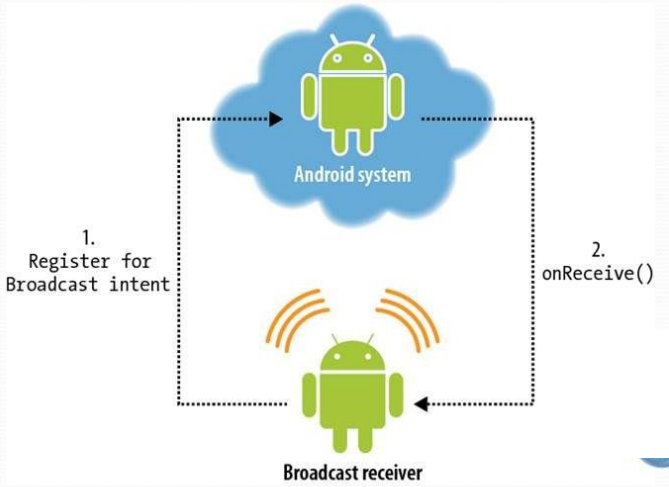
1. Creating a BroadcastReceiver

2. Registering a BroadcastReceiver



Setting up a BroadcastReceiver- Creating

```
public class MyBroadcastReciever extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, text: "Action: " + intent.getAction(), Toast.LENGTH_SHORT).show();
    }
}
```



Setting up a BroadcastReceiver- Registering

A BroadcastReceiver can be registered in two ways.

1. In the `AndroidManifest.xml` file

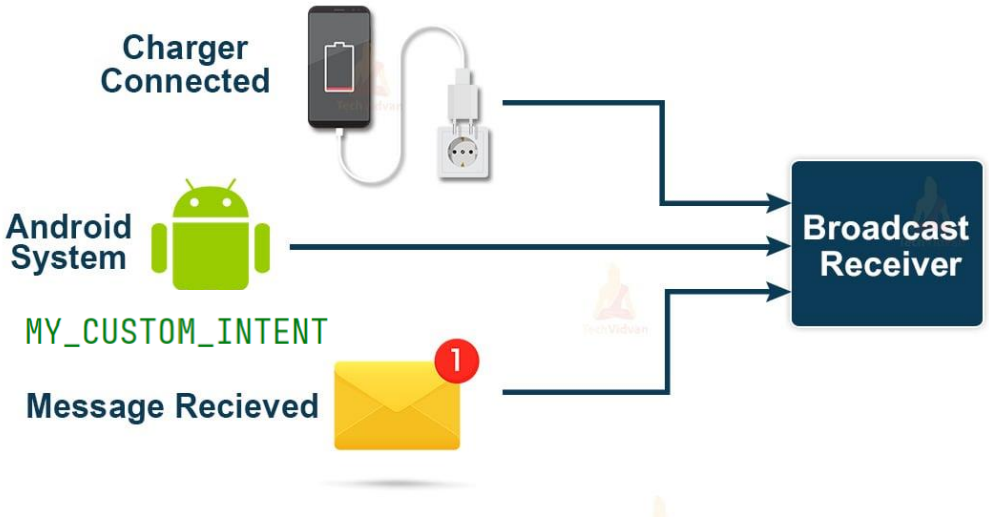
```
<receiver android:name=".MyBroadcastReciever"
    android:exported="true">
    <intent-filter>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
    </intent-filter>
</receiver>
```

2. Defining programmatically

```
IntentFilter filter = new IntentFilter();
filter.addAction(getPackageName() + "android.net.conn.CONNECTIVITY_CHANGE");

MyBroadcastReciever myReceiver = new MyBroadcastReciever();
registerReceiver(myReceiver, filter);
```

Sending a Broadcast intent

```
Intent intent = new Intent();
intent.setAction("com.example.broadcastactivity.MY_CUSTOM_INTENT");
sendBroadcast(intent);
```



Sending a Broadcast with permissions

- Only receivers who have requested that permission with the tag in their manifest (and subsequently been granted the permission if it is dangerous) can receive the broadcast.

```
sendBroadcast(new Intent(BluetoothDevice.ACTION_FOUND),
              Manifest.permission.BLUETOOTH_CONNECT)
```

- To receive the broadcast, the receiving app must request the permission

```
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT"/>
```

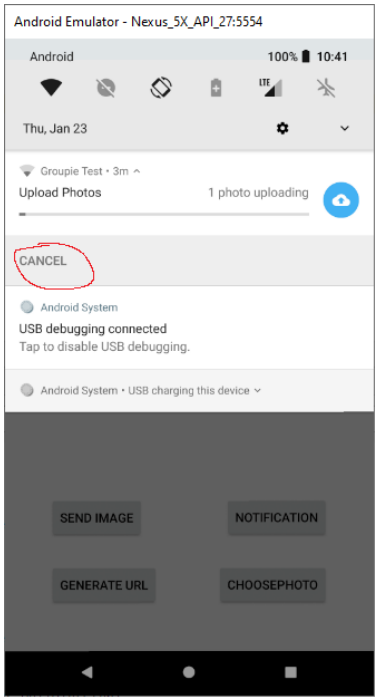
Services

- Does not involve a GUI component. Runs in the **background** and suitable for long running processes.
- Example functionalities achieved through services are network communications, play music, and software updates.
- Three types of services:
  - **Foreground**
  - **Background**
  - **Bound**

- For more info: https://developer.android.com/guide/components/services

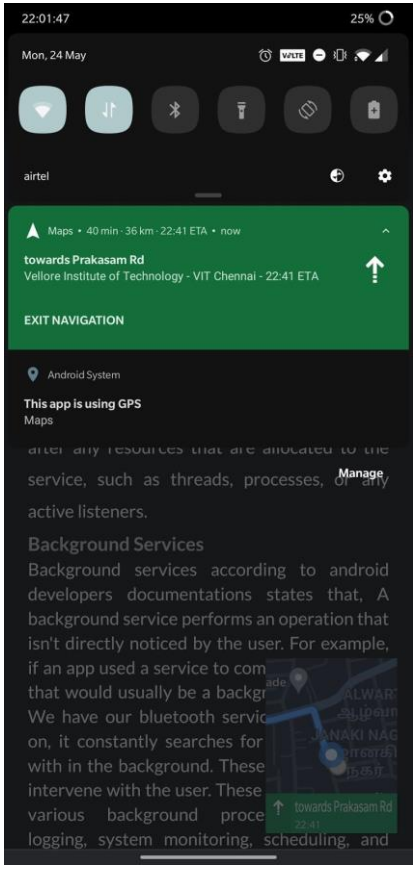
Services

- Background
  - Runs in the background
  - E.g. scheduled backup
- Foreground
  - Operation is noticeable to the user and must display a Notification
  - Does not require user interaction
  - E.g. Audio playback
- Bound
  - Useful to create a connection between application's components and a service to perform some background tasks, such as downloading data, playing music, or fetching information from a remote server.



Services

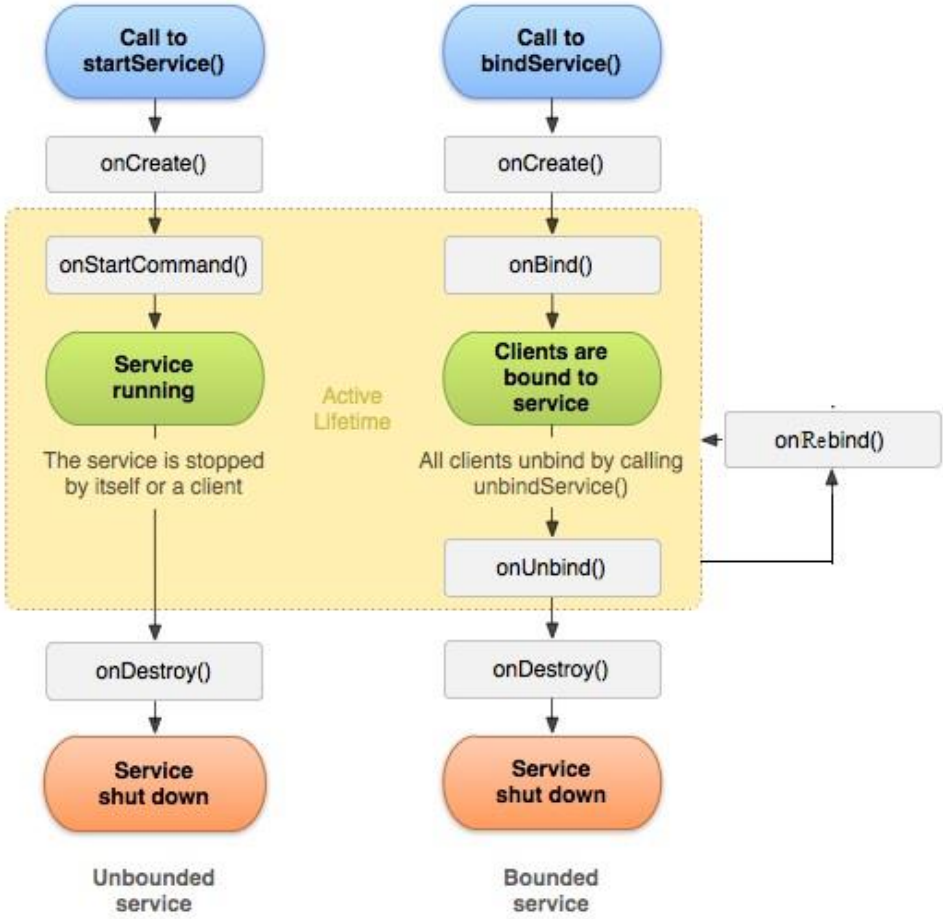
- Background
  - Runs in the background
  - E.g. scheduled backup
- Foreground
  - Operation is noticeable to the user and must display a Notification
  - Does not require user interaction
  - E.g. Audio playback
- Bound
  - Useful to create a connection between application's components and a service to perform some background tasks, such as downloading data, playing music, or fetching information from a remote server.



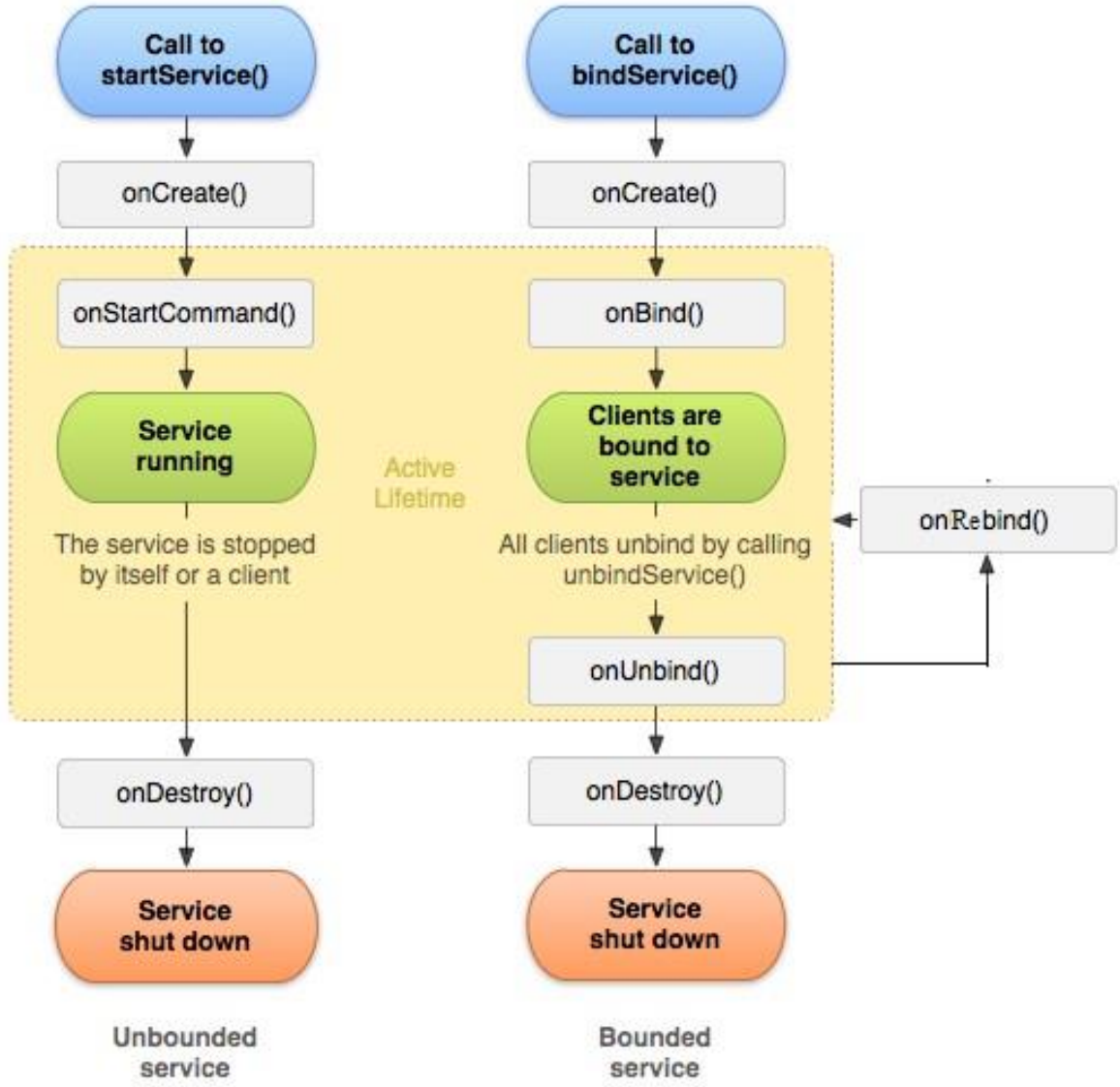
Service Lifecycle

There are two different types of service lifecycle in Android.

- Started Service: When an application component calls startService() then service will start.
- Bound service: Bound services are services that are bound with the application component with the help of the bindService() method



Service Lifecycle



Using a foreground service

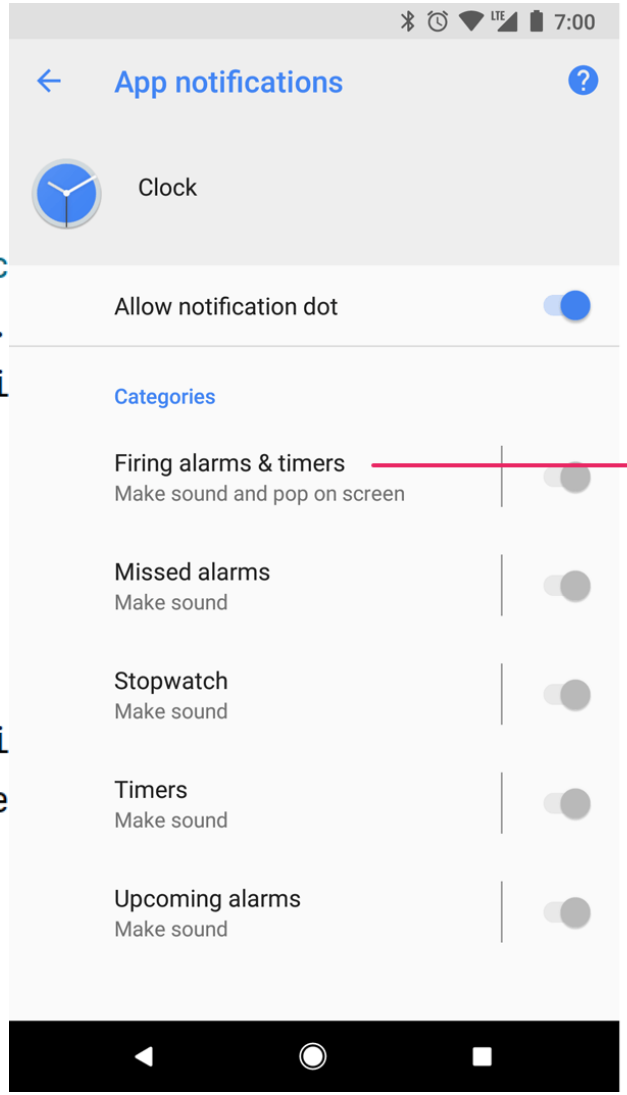
- Step 1 : Create a service class

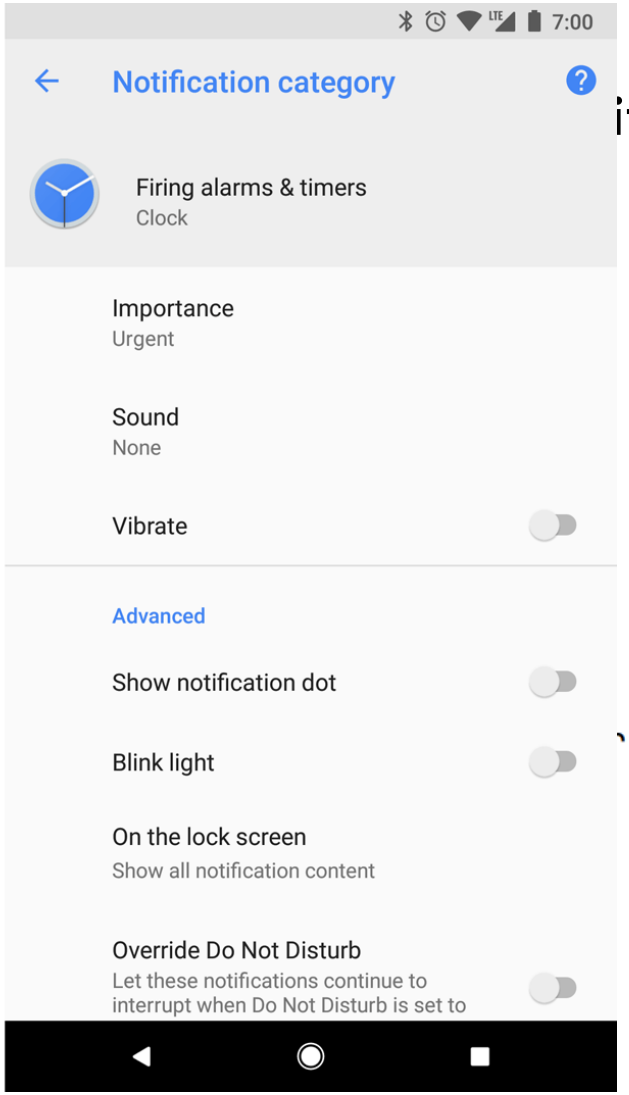
```
public class MyService extends Service {
    no usages
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        // Here we do not use a binder to interact with the foreground service.
        // Since its not a bound service
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();
    }

    5 usages
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        return super.onStartCommand(intent, flags, startId);
    }
}
```

Using a foreground service

- Step ification

```
private void c
    if (Build.
        Notifi
        );

        Notifi                    .class);
        manage
    }
}
```





```java
private void createNotificationChannel() {
    // Create the NotificationChannel, but only on API 26+ because
    // the NotificationChannel class is not in the Support Library.
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        CharSequence name = getString(R.string.channel_name);
        String description = getString(R.string.channel_description);
        int importance = NotificationManager.IMPORTANCE_DEFAULT;
        NotificationChannel channel = new NotificationChannel(CHANNEL_ID, name, importance);
        channel.setDescription(description);
        // Register the channel with the system. You can't change the importance
        // or other notification behaviors after this.
        NotificationManager notificationManager = getSystemService(NotificationManager.class);
        notificationManager.createNotificationChannel(channel);
    }
}
```

User-visible importance level	Importance (Android 8.0 and higher)
Urgent Makes a sound and appears as a heads-up notification.	`IMPORTANCE_HIGH`
High Makes a sound.	`IMPORTANCE_DEFAULT`
Medium Makes no sound.	`IMPORTANCE_LOW`
Low Makes no sound and doesn't appear in the status bar.	`IMPORTANCE_MIN`
None Makes no sound and doesn't appear in the status bar or shade.	`IMPORTANCE_NONE`

Using a foreground service

Step 03 : Call the notification create method from onStartCommand ()

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    int duration = Toast.LENGTH_SHORT;
    Toast toast = Toast.makeText( context: this, text: "onStartCommand",duration);
    toast.show();


    String input = intent.getStringExtra( name: "inputExtra");
    createNotificationChannel();
    Intent notificationIntent = new Intent( packageContext: this, MainActivity.class);
    PendingIntent pendingIntent = PendingIntent.getActivity( context: this,
            requestCode: 0, notificationIntent, PendingIntent.FLAG_IMMUTABLE);

    Notification notification = new NotificationCompat.Builder( context: this, CHANNEL_ID)
            .setContentTitle("Foreground Service")
            .setContentText(input)
            .setSmallIcon(R.drawable.ic_launcher_foreground)
            .setContentIntent(pendingIntent)
            .build();

    startForeground( id: 1, notification);

    // Here is a good place to handle the location consent.
    // You can already start the LocationEngine here.

    return START_NOT_STICKY;
}
```

Using a foreground service

- Step 04: Start the Foreground Service from Activity

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    no usages
    public void startService() {
        Intent serviceIntent = new Intent( packageContext: this, MyService.class);
        serviceIntent.putExtra( name: "inputExtra",  value: "Foreground Service Example in Android");
        ContextCompat.startForegroundService( context: this, serviceIntent);
    }

    no usages
    public void stopService() {
        Intent serviceIntent = new Intent( packageContext: this, MyService.class);
        stopService(serviceIntent);
    }

}
```

Using a foreground service

- Step 5: Declare the Service and Add Permissions

```
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
```

```
<service
    android:name=".MyService"
    android:enabled="true"
    android:exported="true"></service>
```

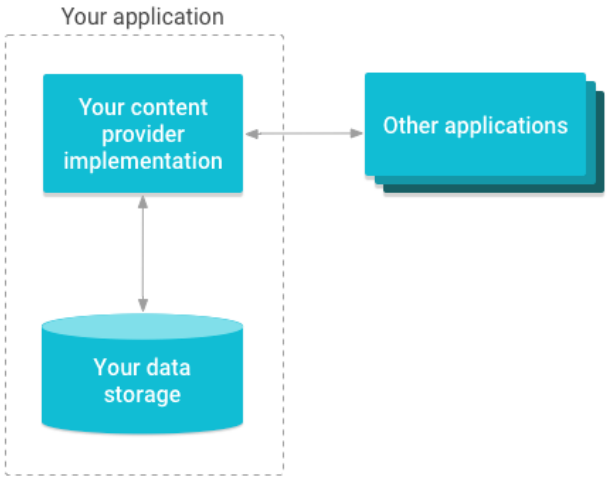
Question

- Use explicit intent when starting a Service – Why? (HW)

Content Provider

- Provides access to a central repository of structured data.
- Use to **securely** exchange data between applications.
- A standardized way to manage and share data between apps, while maintaining data security and integrity
- Android content providers
  - **Contacts, Audio, Video, Images, Calendar, User Dictionary**

- For more info: http://developer.android.com/guide/topics/providers/content-providers.html

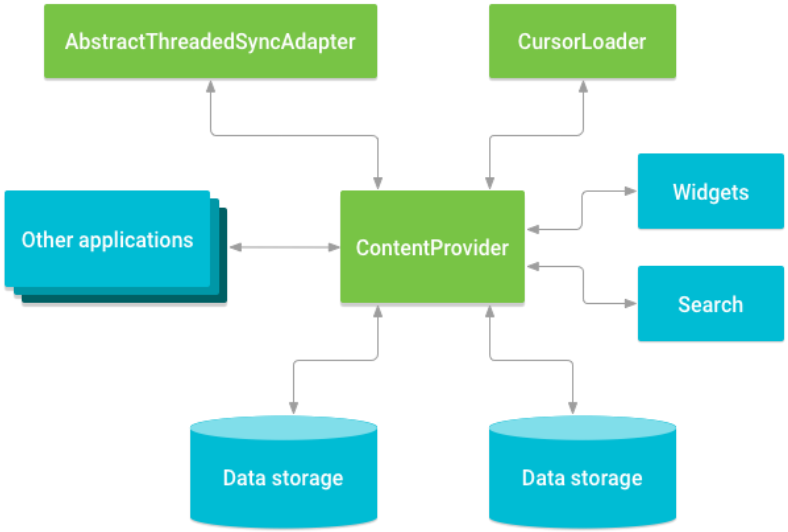


Content Provider- How it works

- A content provider presents data to external applications as one or more tables that are similar to the tables found in a relational database.
  - A row represents an instance of some type of data the provider collects, and each column in the row represents an individual piece of data collected for an instance.

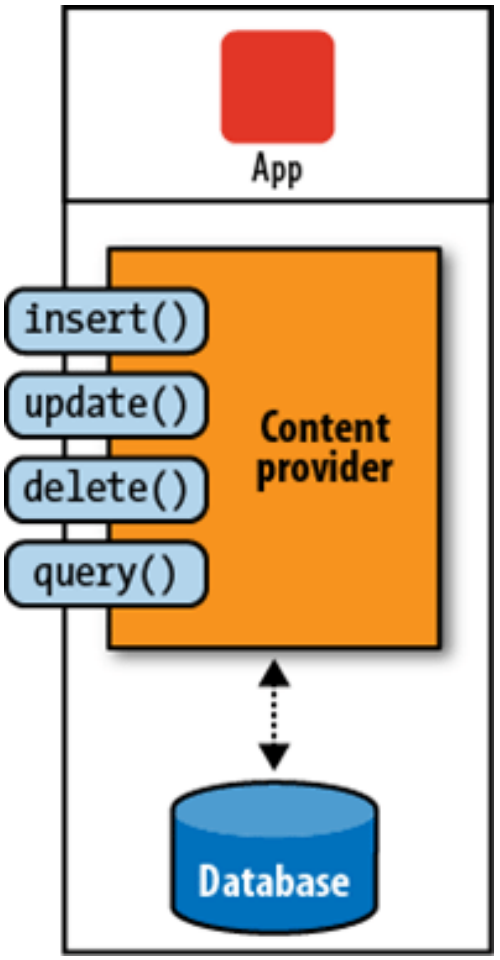
Content Provider- How it works

- A content provider coordinates access to the data storage layer in your application for a number of different APIs and components.
  - Sharing access to your application data with other applications
  - Sending data to a widget
  - Returning custom search suggestions for your application through the search framework using SearchRecentSuggestionsProvider
  - Synchronizing application data with your server using an implementation of AbstractThreadedSyncAdapter
  - Loading data in your UI using a CursorLoader



Content Provider

- Accessing content provider – API "CURD"
  - Create (insert)
  - Retrieve (query)
  - Update
  - Delete



What's Next ?

- Week 5: Mobile Phone Capabilities
- Reminder: Assignment 1 is due next week !

- Happy Learning !

Let's Discuss Group Project Ideas