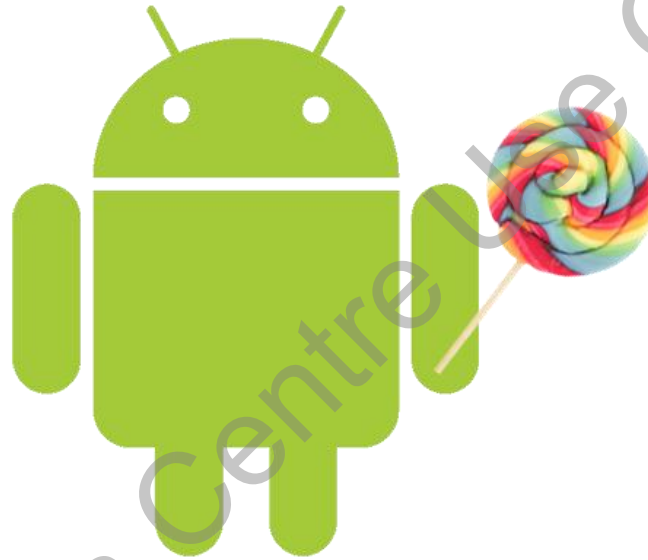


Programming in Android



Session: 8

**Services, Broadcast Receivers, and
Intent Filters**

Objectives

- ◆ Explain services
- ◆ Explain service lifecycle
- ◆ Describe broadcast receiver and its working
- ◆ Explain filters
- ◆ Explain intent matching and its rules
- ◆ Explain filters in manifest file and broadcast receivers



Introduction

- ◆ The main components that play an important role are:
 - ◆ Activities
 - ◆ Services
 - ◆ Content Providers
 - ◆ Broadcast receivers
- ◆ Services are executed in the background to perform long-running operations
- ◆ Broadcast receivers respond to announcements
- ◆ Intents are used for activating components



- ◆ Services can perform background tasks without providing a user interface
- ◆ Service class creates application components that are specifically suited to handle functions that should run at the background
- ◆ Even if the user switches to another application, a service will continue to run in the background
- ◆ Service has higher priority when compared with an inactive activity



Implementing Services

- ◆ Services are declared in the application's manifest file
- ◆ To declare a service in the AndroidManifest.xml file, add a `<service>` element as a child of the `<application>` element as shown in the following Code Snippet:

```
<manifest ... >
...
<application ... >
<service android:name=".SampleService"/>
...
</application>
</manifest>
```

Creating a Service

- ◆ A component starts a service by invoking the `startService()` method
- ◆ It results in a call to the Service class `onStartCommand()` method
- ◆ To create a started service, following are the two classes from which the service class can extend from:
 - ◆ `IntentService`
 - ◆ `Service`
- ◆ `IntentService` class is implemented to avoid multi-threading
- ◆ It is started as a normal service, performs the tasks within a worker thread, and terminates when the task is performed

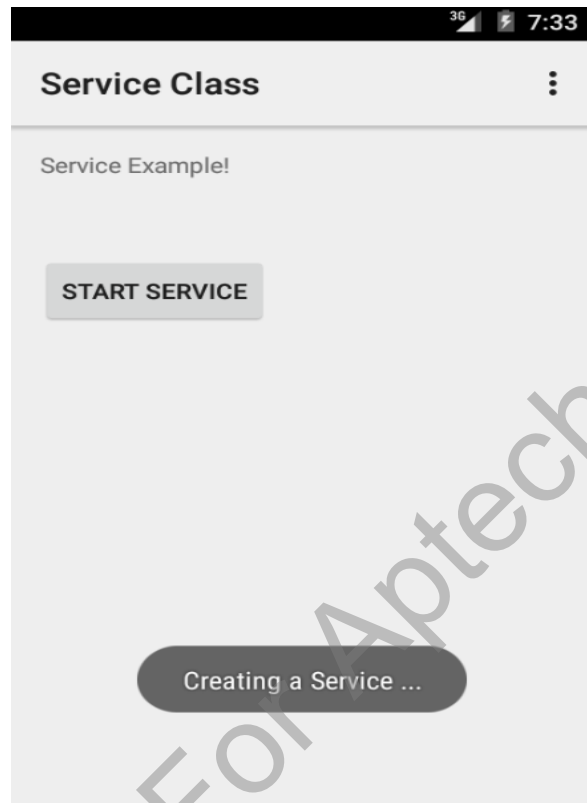
Extending IntentService

- ◆ Following Code Snippet demonstrates an example creating services by extending the IntentService class:

```
public class MyIntentService extends IntentService {  
    public MyIntentService() {  
        super("MyIntentService");  
    }  
  
    @Override  
    protected void onHandleIntent(Intent Intent) {  
        ...  
    }  
    @Override  
    public void onCreate() {  
        super.onCreate();  
        ...  
    }  
}
```

Services Example Application

- Using the code, an application for demonstrating Services is created as shown in the following figure:



- The application logic is as follows:
 - ◆ Developer creates a Service by extending the Intent Service class
 - ◆ The Service is started when the START SERVICE button is clicked
 - ◆ The Service creates a Toast once it is started
 - ◆ The Service creates another toast when an Intent is handled

Extending Service Class

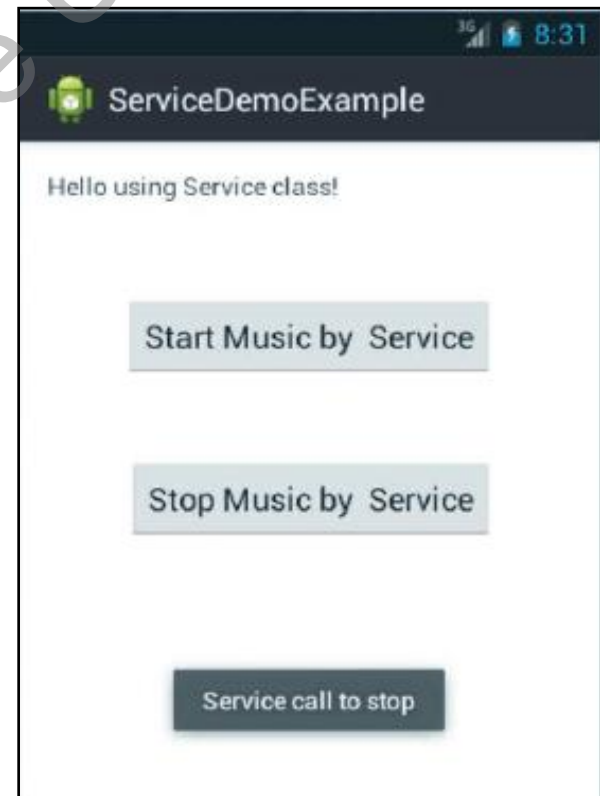
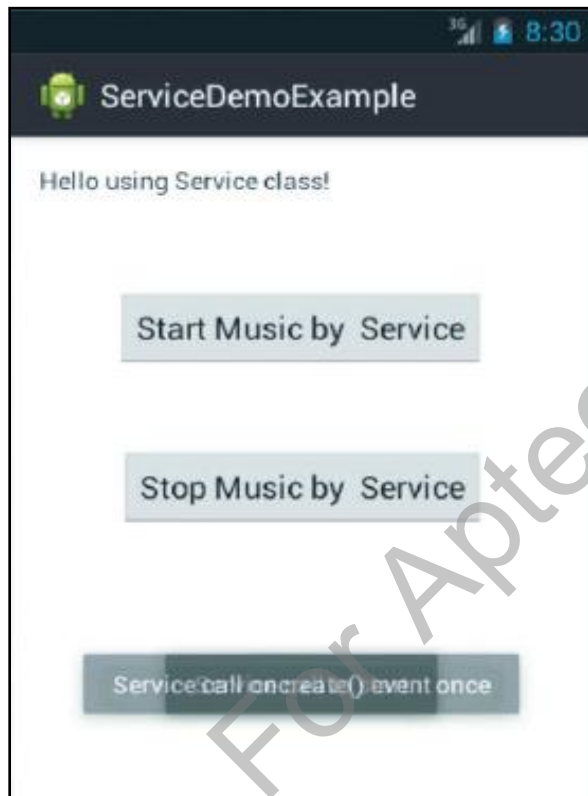
- ◆ Following Code Snippet demonstrates an example for creating a Service by extending the Service class:

```
public class ServiceExample extends Service {  
  
    @Override  
    public IBinder onBind(Intent Intent) { ... }  
  
    @Override  
    public void onCreate() {  
        ...  
    }  
  
    @Override  
    public void onDestroy() {  
        ...  
    }  
  
    @Override  
    public void onStart(Intent Intent, int startid) {  
        ...  
    }  
}
```

Example Application Extending Service 1-2

- Using the code, an application for demonstrating Services using the Service class is created as shown in the following figure:

- By clicking the 'Star Music by Service', the music starts to play as shown in the following figure:



Example Application Extending Service 2-2

- The application logic is as follows:

- ◆ Developer creates a Service by extending the Service class
- ◆ The Service is started when the Start Music by Service button is clicked
- ◆ Once started, the Service creates a MediaPlayer object to play an audio file
- ◆ The Service is stopped when the Stop Music by Service is clicked

For Aptech Centre Use Only

Bound Services

- ◆ When components of an application bind to a service by calling `bindService()`, it is called a bound service
- ◆ Binding to a service helps in creating long-standing connection
- ◆ A service can be bound to:
 - ◆ Activity
 - ◆ Services
 - ◆ Content Providers
- ◆ To use a bound service, the `onBind()` callback method must be implemented
- ◆ It returns an `IBinder` that specifies the interface for communication with the service



Bound Service Example

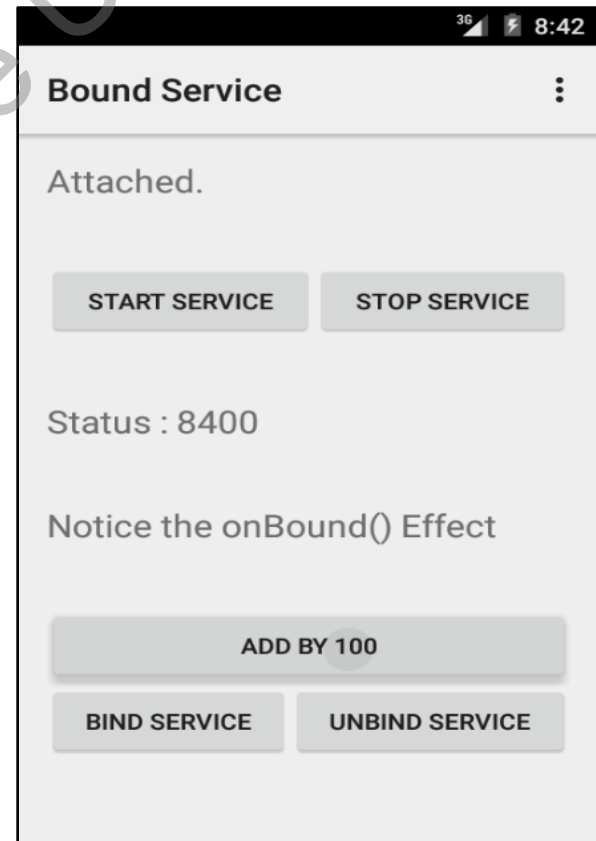
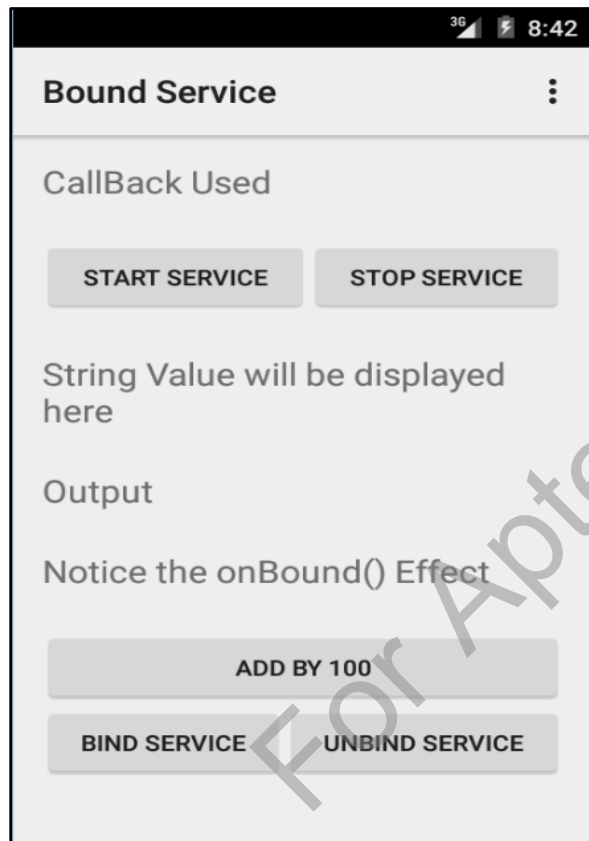
- ◆ Following Code Snippet demonstrates how to create a Bound Service:

```
public class MyBoundService extends Service {  
    ...  
    final Messenger mMessenger = new Messenger (new IncomingHandler());  
    ...  
    @Override  
    public IBinder onBind(Intent Intent) {  
        return mMessenger.getBinder();  
    }  
    ...  
}
```

Bound Services Example Application 1-2

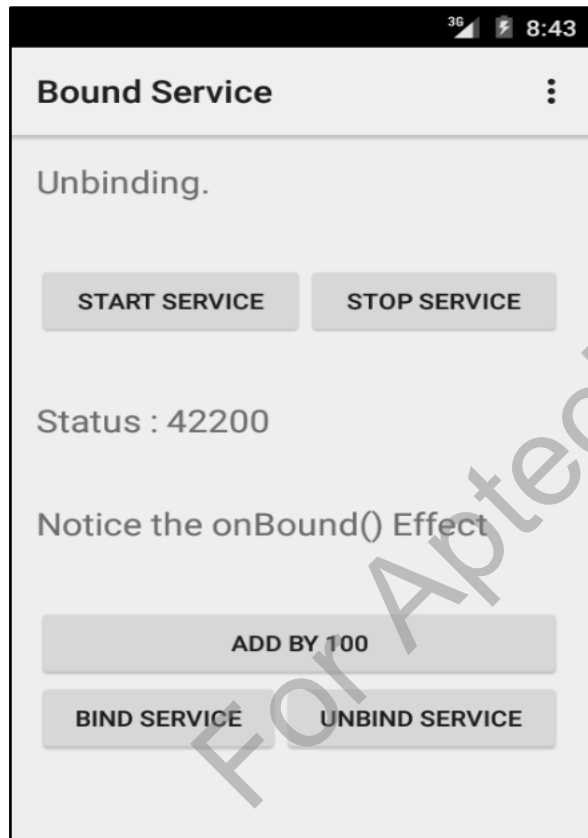
- Using the explained code, an application for demonstrating Bound Services as shown in the following figure:

- Click START SERVICE, then click ADD BY 100 and finally, click BIND SERVICE to get the output as shown in the following figure:



Bound Services Example Application 2-2

- The STOP SERVICE button will stop the service and unbind the service. The output of stopping the service will be as shown in the following figure:



- The application logic is as follows:

- ◆ Developer creates a Service by extending the Service class
- ◆ The Service is started and stopped by clicking the Start and Stop buttons respectively
- ◆ Once the Bind Service button is clicked, the `bindService()` method is called
- ◆ When the UNBIND SERVICE button is clicked, the service is unbound
- ◆ Clicking on the ADD BY 100 button will send a message to the service which will increment the count by a value of 100

Broadcast Receivers and Intents

- ◆ Responsible for responding to system-wide broadcast announcements
- ◆ Intents are responsible for binding the different individual components to each other at runtime
- ◆ A broadcast receiver is an Android component which allows the application to register for system or application events
- ◆ Broadcast receivers do not display a user interface



Implementing Broadcast Receivers

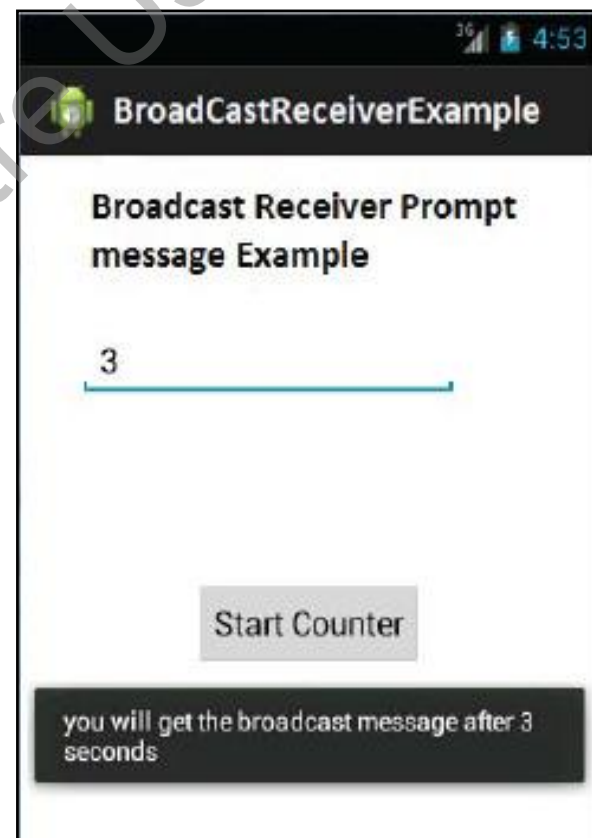
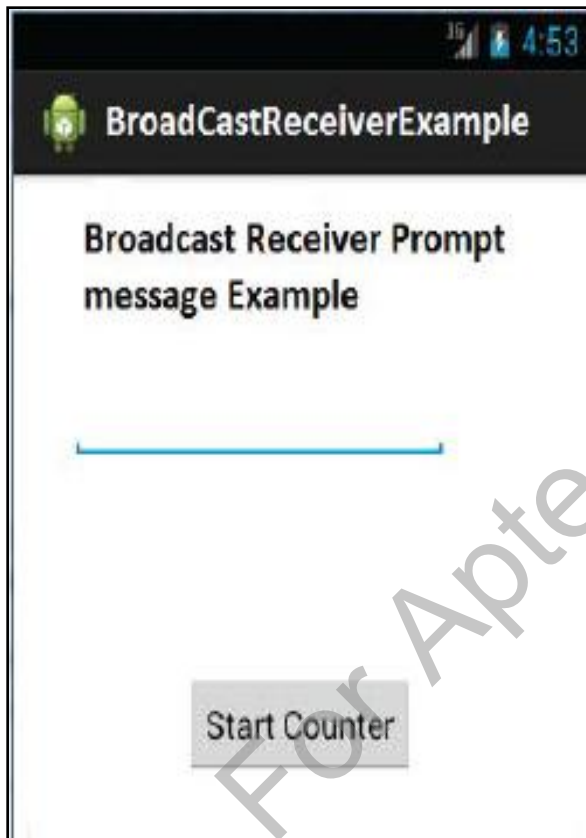
- ◆ The steps to create a Broadcast Receiver are:
 - ◆ You have to create a subclass of Android's Broadcast Receiver
 - ◆ You have to implement the onReceive() method
- ◆ Android calls the onReceive() method on all registered broadcast receivers, when a matching broadcast Intent is identified
- ◆ The onReceive() method accepts following two arguments:
 - ◆ Context
 - ◆ Intent
- ◆ Following Code Snippet demonstrates how to create a Broadcast Receiver:

```
public class MyBroadcast extends BroadcastReceiver{  
    @Override  
    public void onReceive(Context context, Intent Intent) {  
        ...  
    }  
}
```

Broadcast Receiver Example Application 1-2

- Using the code, an application for demonstrating Broadcast Receiver is as shown in the following figure:

- Enter 3 in the Edit box and the output will be as shown in the following figure:



Broadcast Receiver Example Application 2-2

- After three seconds the message will be displayed as shown in the following figure:



- The application logic is as follows:

- ◆ Developer creates an Activity with an EditText and a Button
- ◆ The user enters an interval to send the Broadcast Message
- ◆ When the timeout occurs, a Broadcast Message is sent
- ◆ A toast is displayed signifying the same

System Broadcast 1-2

- In the API, there are many classes that have specific broadcast events
- Following table lists some of the available events:

Event	Usage
Intent.ACTION_BATTERY_LOW	The battery level has dropped below a threshold
Intent.ACTION_BATTERY_OKAY	The battery level has increased again
Intent.ACTION_BOOT_COMPLETED	Android is up and running
Intent.ACTION_DEVICE_STORAGE_LOW	Storage space on the device is becoming less
Intent.ACTION_DEVICE_STORAGE_OK	The storage situation has improved again
Intent.ACTION_HEADSET_PLUG	A headset was plugged in or a previously plugged headset was removed
Intent.ACTION_LOCALE_CHANGED	The language of the device has been changed by the user

System Broadcast 2-2

Event	Usage
Intent.ACTION_MY_PACKAGE_REPLACED	The application has been updated
Intent.ACTION_PACKAGE_ADDED	A new application has been installed
Intent.ACTION_POWER_CONNECTED	The device has been plugged in
Intent.ACTION_POWER_DISCONNECTED	The device has been disconnected again
KeyChain.ACTION_STORAGE_CHANGED	The key store has changed
BluetoothDevice.ACTION_ACL_CONNECTED	A Bluetooth ACL connection has been established
AudioManager.ACTION_AUDIO_BECOMING_NOISY	The internal audio speaker is about to be used instead of other output means (like a headset)

Role of Filters

- ◆ When Intent is sent to the Android system, it determines suitable applications for this Intent
- ◆ If several components have been registered for this type of Intent, Android offers the user the choice to open one of them
- ◆ This decision is based on IntentFilters
- ◆ An Intent filter is an instance of the IntentFilter class



Intent Filter Example

- ◆ Following Code Snippet demonstrates how to create an Intent Filter to trigger an activity:

```
<activity android:name=".BrowserActivitiy"  
android:label="@string/app_name">  
<Intent-filter>  
<action android:name="android.Intent.action.VIEW" />  
<category android:name="android.Intent.category.DEFAULT" />  
<data android:scheme="http"/>  
</Intent-filter>  
</activity>
```

Intent Matching

- ◆ Matching of intents with intent filters is performed to discover a target component to activate
- ◆ Also used to get information about the set of components available on the device
- ◆ Android system populates the application launcher, by finding all the activities with intent filters that specify the `android.Intent.action.MAIN` action and `android.Intent.category.LAUNCHER` category

For Aptech Centre Use Only

Intent Matching Rules

- ◆ To match an IntentFilter with an Intent, three conditions must be fulfilled - the action, the category, and the data (both the data type and data scheme+authority+path, if specified) must match
- ◆ The Data characteristic is divided into four attributes:
 - ◆ Types
 - ◆ Schemes
 - ◆ Authority
 - ◆ Path
- ◆ Data Scheme matches when any of the given values match the Intent data's scheme
- ◆ Data Authority matches the given values with Intent data's authority
- ◆ Data Path matches any given values with the Intent's data path and when both a scheme

Filters in Manifest

- ◆ By using intent filters, components specify their capability that is, the kinds of Intents they can respond to
- ◆ Intent filters are specified in the manifest as <Intent-filter> elements
- ◆ A component can have any number of filters, every filter describing a diverse capability
- ◆ When an intent explicitly names a target component, it activates that component; the filter does not play any role
- ◆ When Intent does not specify a target by name, it activates a component only if it can pass through one of the component's filters
- ◆ Following Code Snippet demonstrates how to create a Intent Filter in the AndroidManifest File:

```
<intent-filterandroid:icon="resource path from drawables"  
android:label="resource path of string" android:priority="integer">  
  
</intent-filter>
```

Filters in Broadcast Receivers

- ◆ When the user registers a BroadcastReceiver to be executed in the main activity thread, the receiver is called in the main application thread
- ◆ The system can broadcast Intents that are 'sticky'
- ◆ There might be times when multiple sticky Intents may match the filter
- ◆ Only one of these Intents can be returned directly by the function and that is randomly decided by the system
- ◆ When the user knows that the registered Intent is sticky, then null can be supplied for the receiver
- ◆ Following Code Snippet demonstrates how to create an Intent Filter in Broadcast Receivers:

```
...  
<receiverandroid:name=".BCastReceiver">  
  <intent-filter>  
    <action android:name="android.Intent.action.MAIN"/>  
    <category android:name="android.Intent.category.LAUNCHER"/>  
    <action android:name="android.Intent.action.ACTION_POWER_CONNECTED"/>  
  </intent-filter>  
</receiver>  
...
```

Summary

- ◆ A service is an application component that is able to carry out long-running operations in the background. It does not provide a user interface
- ◆ In order to interact with the components, a service can let other components bind to it and perform Inter Process Communication (IPC). A service runs in the main thread of the application that hosts it
- ◆ A Broadcast Receiver is a component that responds to system-wide broadcast announcements
- ◆ Activities, Services, and Broadcast Receivers have intent filters to inform the system which intents they can handle
- ◆ Intent filters can handle two types of intents: Implicit and Explicit
- ◆ Matching of intents with intent filters is done to discover a target component to be activated, and also to get information about the set of components available on the device
- ◆ To match an intent filter with an Intent, three conditions must be fulfilled: the action, category, and the data