Name : Mujahid Ullah

Subject: Android Development

Submitted To Sir Junaid

Date : 28-Dec-20

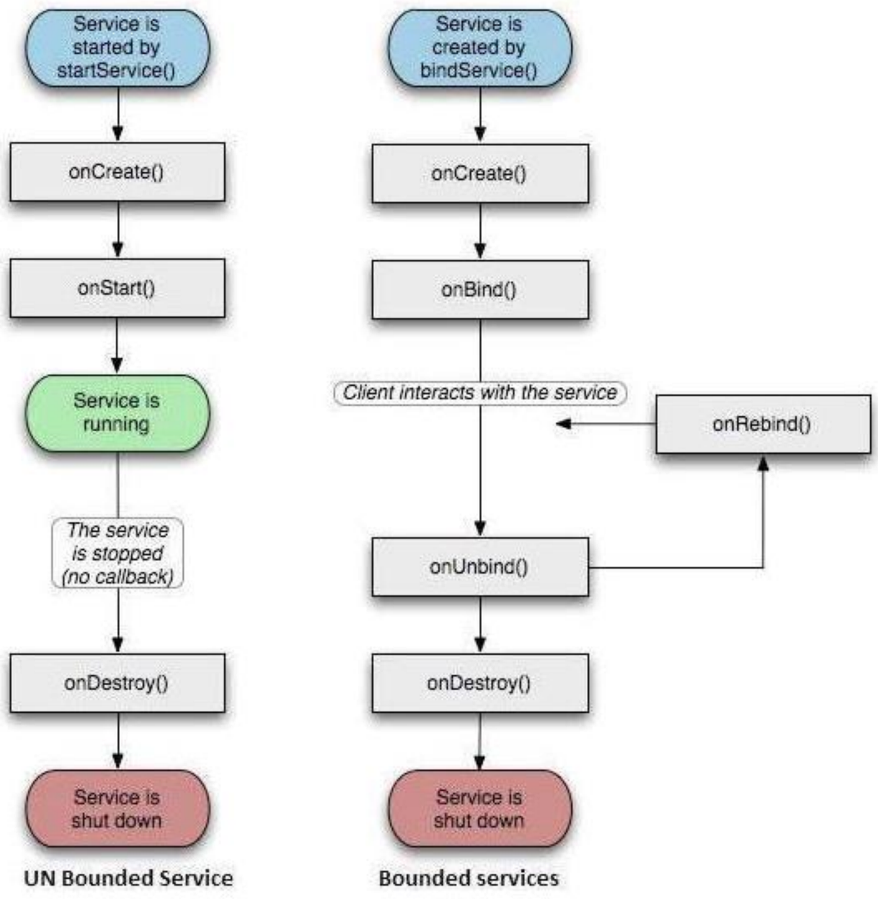
Assignment No 6

1) What is Service:

A service is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed. A service can essentially take two states –

Sr. no	State and Description
1	**Started:** A service is started when an application component, such as an activity, starts it by calling startService(). Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
2	**Bound:** A service is bound when an application component binds to it by calling bindService(). A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).

A service has life cycle callback methods that you can implement to monitor changes in the service's state and you can perform work at the appropriate stage. The following diagram on the left shows the life cycle when the service is created with startService() and the diagram on the right shows the life cycle when the service is created with bindService(): (image courtesy : android.com)



To create an service, you create a Java class that extends the Service base class or one of its existing subclasses. The Service base class defines various callback methods and the most important are given below. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

Sr.No	Callback & Description
1	**onStartCommand():** The system calls this method when another component, such as an activity, requests that the service be started, by calling startService(). If you implement this method, it is your responsibility to stop the service when its work is done, by calling stopSelf() or stopService() methods**.**
2	**onBind():** The system calls this method when another component wants to bind with the service by calling bindService(). If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an IBinder object. You must always implement this method, but if you don't want to allow binding, then you should return null.

3	**onUnbind():** The system calls this method when all clients have disconnected from a particular interface published by the service.
4	**onRebind():** The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its onUnbind(Intent).
5	**onCreate():** The system calls this method when the service is first created using onStartCommand() or onBind(). This call is required to perform one-time set-up.
6	**onDestroy():** The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc.

The following skeleton service demonstrate each life cycle method:-

```java
package com.skeleton.services;

        import android.app.Service;
        import android.os.IBinder;
        import android.content.Intent;
        import android.os.Bundle;

public class HelloService extends Service {

    /** indicates how to behave if the service is killed */
    int mStartMode;

    /** interface for clients that bind */
    IBinder mBinder;

    /** indicates whether onRebind should be used */
    boolean mAllowRebind;

    /** Called when the service is being created. */
    @Override
    public void onCreate() {

    }

    /** The service is starting, due to a call to startService() */
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        return mStartMode;
    }

    /** A client is binding to the service with bindService() */
    @Override
    public IBinder onBind(Intent intent) {
        return mBinder;
    }

    /** Called when all clients have unbound with unbindService() */
    @Override
    public boolean onUnbind(Intent intent) {
        return mAllowRebind;
    }

    /** Called when a client is binding to the service with bindService()*/
    @Override
    public void onRebind(Intent intent) {

    }

    /** Called when The service is no longer used and is being destroyed */
    @Override
    public void onDestroy() {

    }
}
```

Example

This example will take you through simple steps to show how to create your own Android Service. Follow the following steps to modify the Android application we created in Hello World Example chapter –

Step	Description
1	You will use Android StudioIDE to create an Android application and name it as *My Application* under a package *com.example.tutorialspoint7.myapplication* as explained in the *Hello World Example* chapter.
2	Modify main activity file *MainActivity.java* to add *startService()* and *stopService()* methods.
3	Create a new java file *MyService.java* under the package *com.example.My Application*. This file will have implementation of Android service related methods.
4	Define your service in *AndroidManifest.xml* file using <service.../> tag. An application can have one or more services without any restrictions.
5	Modify the default content of *res/layout/activity_main.xml* file to include two buttons in linear layout.
6	No need to change any constants in *res/values/strings.xml* file. Android studio take care of string values
7	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **MainActivity**.java. This file can include each of the fundamental life cycle methods. We have added startService() and stopService() methods to start and stop the service.

```
package com.cal.services;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    String msg="Android";

    /** Called when the activity is first created. */

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


    }
public void startService(View view){

        startService(new Intent(getBaseContext(),Services.class));

}

    // Method to stop the service
public void stopService(View view){
        stopService(new Intent(getBaseContext(),Services.class));


}

}
```

Following is the content of MyService.java. This file can have implementation of one or more methods associated with Service based on requirements. For now we are going to implement only two methods onStartCommand() and onDestroy() –

```
package com.cal.services;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;

import androidx.annotation.Nullable;
```

```java
import java.security.Provider;

public class Services extends Service {
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }



    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // Let it continue running until it is stopped.
        Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        Toast.makeText(this, "Service Destroyed", Toast.LENGTH_LONG).show();

    }
}
```

Following will the modified content of AndroidManifest.xml file. Here we have added <service.../> tag to include our service –

```xml
<service android:name=".Services"/>
```

Following will be the content of res/layout/activity_main.xml file to include two buttons –

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Example of services"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:textSize="30dp" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Tutorials point "
        android:textColor="#ff87ff09"
        android:textSize="30dp"
        android:layout_above="@+id/imageButton"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="40dp" />

    <ImageButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageButton"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button2"
        android:text="Start Services"
        android:onClick="startService"
        android:layout_below="@+id/imageButton"
        android:layout_centerHorizontal="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Stop Services"
        android:id="@+id/button"
        android:onClick="stopService"
        android:layout_below="@+id/button2"
        android:layout_alignLeft="@+id/button2"
        android:layout_alignStart="@+id/button2"
        android:layout_alignRight="@+id/button2"
        android:layout_alignEnd="@+id/button2" />

</LinearLayout>
```

Play Music On Background :

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    // Let it continue running until it is stopped.
    myPlayer = MediaPlayer.create(this, Settings.System.DEFAULT_RINGTONE_URI);
    myPlayer.setLooping(true); // Set looping
    Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
    myPlayer.start();
    return START_STICKY;
}

@Override
public void onDestroy() {
    Toast.makeText(this, "Service Destroyed", Toast.LENGTH_LONG).show();

    myPlayer.stop();
}
```

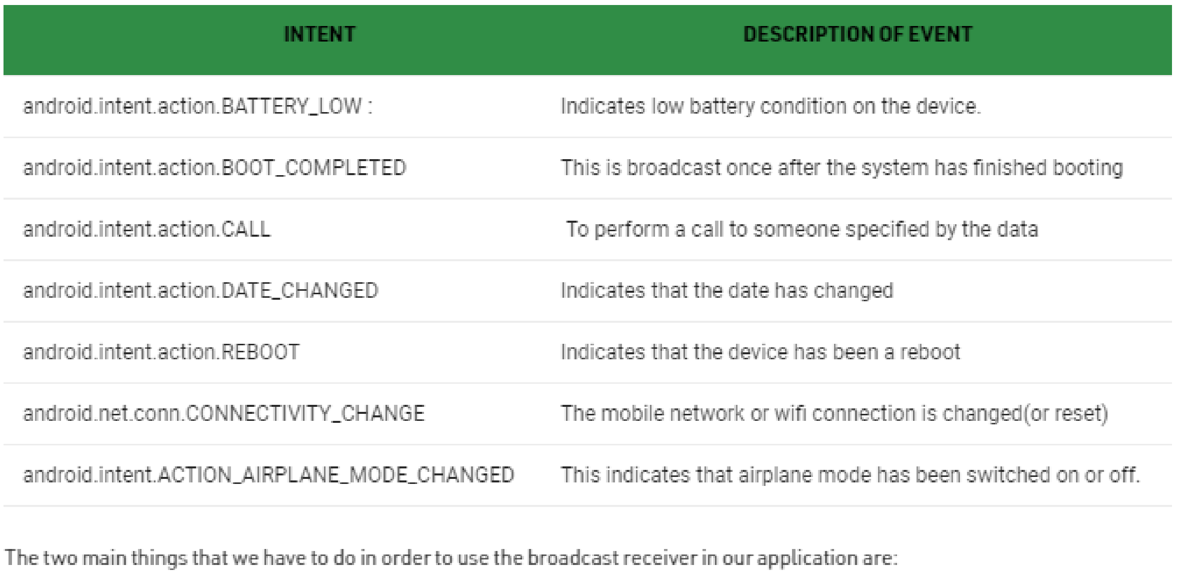
2. What is brodacaster explain with example

Broadcast Receiver : Broadcast in android is the system-wide events that can occur when the device starts, when a message is received on the device or when incoming calls are received, or when a device goes to an airplane mode, etc. Broadcast Receivers are used to respond to these system-wide events. Broadcast Receivers allow us to register for the system and application events, and when that event happens, then the register receivers get notified. There are mainly two types of Broadcast Receivers:

Static Broadcast Receivers: These types of Receivers are declared in the manifest file and works even if the app is closed.

Dynamic Broadcast Receivers: These types of receivers work only if the app is active or minimized.

Since from API Level 26, most of the broadcast can only be caught by the dynamic receiver, so we have implemented dynamic receivers in our sample project given below. There are some static fields defined in the Intent class which can be used to broadcast different events. We have taken a change of airplane mode as a broadcast event, but there are many events for which broadcast register can be used. Following are some of the important system-wide generated intents:-



INTENT	DESCRIPTION OF EVENT
android.intent.action.BATTERY_LOW :	Indicates low battery condition on the device.
android.intent.action.BOOT_COMPLETED	This is broadcast once after the system has finished booting
android.intent.action.CALL	To perform a call to someone specified by the data
android.intent.action.DATE_CHANGED	Indicates that the date has changed
android.intent.action.REBOOT	Indicates that the device has been a reboot
android.net.conn.CONNECTIVITY_CHANGE	The mobile network or wifi connection is changed(or reset)
android.intent.ACTION_AIRPLANE_MODE_CHANGED	This indicates that airplane mode has been switched on or off.

The two main things that we have to do in order to use the broadcast receiver in our application are:

Creating the Broadcast Receiver:

```
class AirplaneModeChangeReceiver:BroadcastReceiver() {
```

```
override fun onReceive(context: Context?, intent: Intent?) {

// logic of the code needs to be written here

}

}
```

Registering a BroadcastReceiver:

```
IntentFilter(Intent.ACTION_AIRPLANE_MODE_CHANGED).also {


        // receiver is the broadcast receiver that we have registered


        // and it is the intent filter that we have created

        registerReceiver(receiver,it)


  }
```

Example:

Below is the sample project showing how to create the broadcast Receiver and how to register them for a particular event and how to use them in the application.

A BroadcastReceiver can be registered in two ways.

1 : By defining it in the AndroidManifest.xml file as shown below.

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

2 By defining it programmatically:

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

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

//Create ConnectionReceiver.java

```java
public class ConnectionReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d("API123",""+intent.getAction());


if(intent.getAction().equals("com.journaldev.broadcastreceiver.SOME_ACTION"))
            Toast.makeText(context, "SOME_ACTION is received",
Toast.LENGTH_LONG).show();

        else {
            ConnectivityManager cm =
                    (ConnectivityManager)
context.getSystemService(Context.CONNECTIVITY_SERVICE);

            NetworkInfo activeNetwork = cm.getActiveNetworkInfo();
            boolean isConnected = activeNetwork != null &&
                    activeNetwork.isConnectedOrConnecting();
            if (isConnected) {
                try {
                    Toast.makeText(context, "Network is connected",
Toast.LENGTH_LONG).show();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            } else {
                Toast.makeText(context, "Network is changed or reconnected",
Toast.LENGTH_LONG).show();
            }
        }
    }
    }
```

To unregister a broadcast receiver in onStop() or onPause() of the activity the following snippet can be used.

```java
@Override
protected void onPause() {
    unregisterReceiver(myReceiver);
    super.onPause();
}
```

Sending Broadcast intents from the Activity The following snippet is used to send an intent to all the related BroadcastReceivers.

```java
Intent intent = new Intent("com.journaldev.broadcastreceiver.SOME_ACTION");
sendBroadcast(intent);
```

Don’t forget to add the above action in the intent filter tag of the manifest or programmatically.

Let’s develop an application that listens to network change events and also to a custom intent and handles the data accordingly.

ALL Methods In Main Activity:

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

    myReceiver = new ConnectionReceiver();
    intentFilter = new
IntentFilter("com.journaldev.broadcastreceiver.SOME_ACTION");


}

@Override
protected void onResume() {
    super.onResume();
    registerReceiver(myReceiver, intentFilter);

}

@Override
protected void onDestroy() {
    super.onDestroy();

    unregisterReceiver(myReceiver);
}

@Override
protected void onPause() {
    unregisterReceiver(myReceiver);
    super.onPause();
}

public void btn_braod(View view) {

    Intent intent = new Intent("com.journaldev.broadcastreceiver.SOME_ACTION");
    sendBroadcast(intent);
}
```

Question No 3: How To Get Image From Gallery And Camera:

First you must have a ImageView in your layout implemented to capture the image you upload either through the camera or Gallery.

Following is my ImageView implementation for the above purpose.

```
<ImageView
    android:layout_width="match_parent"
    android:id="@+id/my_avatar"
    android:layout_height="wrap_content"
    android:adjustViewBounds="true"
    app:layout_collapseMode="parallax"
    app:layout_collapseParallaxMultiplier="0.7" />
```

Now in the activity method, you have to declare the ImageView first.

```
private ImageView imageView;
```

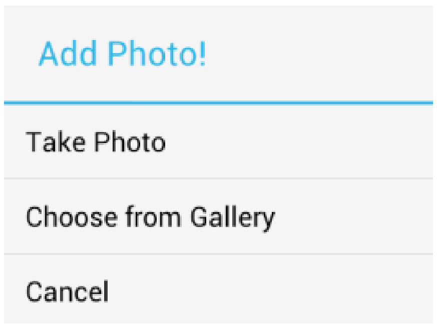
In the onCreate override method, you have to initialize the ImageView element by adding the following,

```
imageView = (ImageView) findViewById(R.id.image);
```

Now you have to include the following method, which let's the user to choose how he/she want to upload the picture, either through camera or Gallery/Photos.

```
private void selectImage(Context context) {
    final CharSequence[] options = { "Take Photo", "Choose from Gallery","Cancel"
};

    AlertDialog.Builder builder = new AlertDialog.Builder(context);
    builder.setTitle("Choose your profile picture");

    builder.setItems(options, new DialogInterface.OnClickListener() {

        @Override
        public void onClick(DialogInterface dialog, int item) {

            if (options[item].equals("Take Photo")) {
                Intent takePicture = new
Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
                startActivityForResult(takePicture, 0);

            } else if (options[item].equals("Choose from Gallery")) {
                Intent pickPhoto = new Intent(Intent.ACTION_PICK,
android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
                startActivityForResult(pickPhoto , 1);

            } else if (options[item].equals("Cancel")) {
                dialog.dismiss();
            }
        }
    });
    builder.show();
}
```

This allows to show the following dialog box,



Now you have to implement the onActivityResult override method as follows,

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != RESULT_CANCELED) {
        switch (requestCode) {
            case 0:
                if (resultCode == RESULT_OK && data != null) {
                    Bitmap selectedImage = (Bitmap) data.getExtras().get("data");
                    imageView.setImageBitmap(selectedImage);
                }

                break;
            case 1:
                if (resultCode == RESULT_OK && data != null) {
                    Uri selectedImage = data.getData();
                    String[] filePathColumn = {MediaStore.Images.Media.DATA};
                    if (selectedImage != null) {
                        Cursor cursor = getContentResolver().query(selectedImage,
                                filePathColumn, null, null, null);
                        if (cursor != null) {
                            cursor.moveToFirst();

                            int columnIndex =
cursor.getColumnIndex(filePathColumn[0]);
                            String picturePath = cursor.getString(columnIndex);

imageView.setImageBitmap(BitmapFactory.decodeFile(picturePath));
                            cursor.close();
                        }
                    }

                }
                break;
        }
    }
}
```

Now you have to call the selectImage method from the onCreate method.