Lec. #7: Android Services (Part 2)

Mobile Applications Development 2

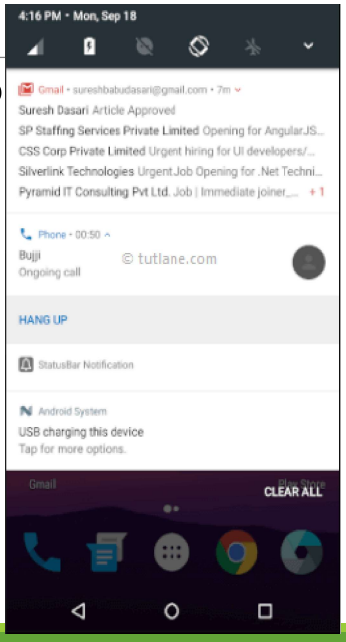
SECOND SEMESTER OF THE ACADEMIC YEAR 2020/2021

Outlines

- Notification
- Application Class
- Started Service (Foreground)
- Bound Service
- JobService

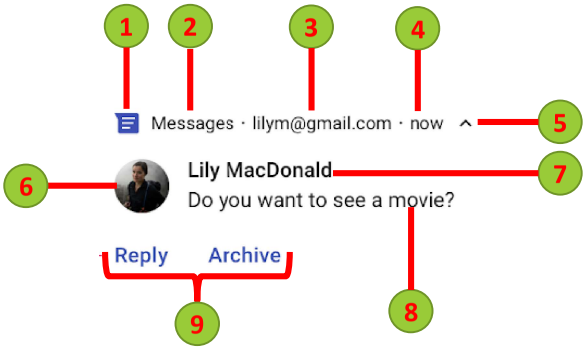
Notifications

- **Notification** is a short message/alert which is used outside of the app (on Notifications App in the system) to alert the users about some events that happening in app, without interrupting current activities.
- **How notifications may be noticed?**
  - a flash/blinking the device's LED
  - Playing a sound or vibrating
  - Showing a status bar icon
  - Appearing on the lock screen
- To see the details of android app notification, we need to open the notification drawer

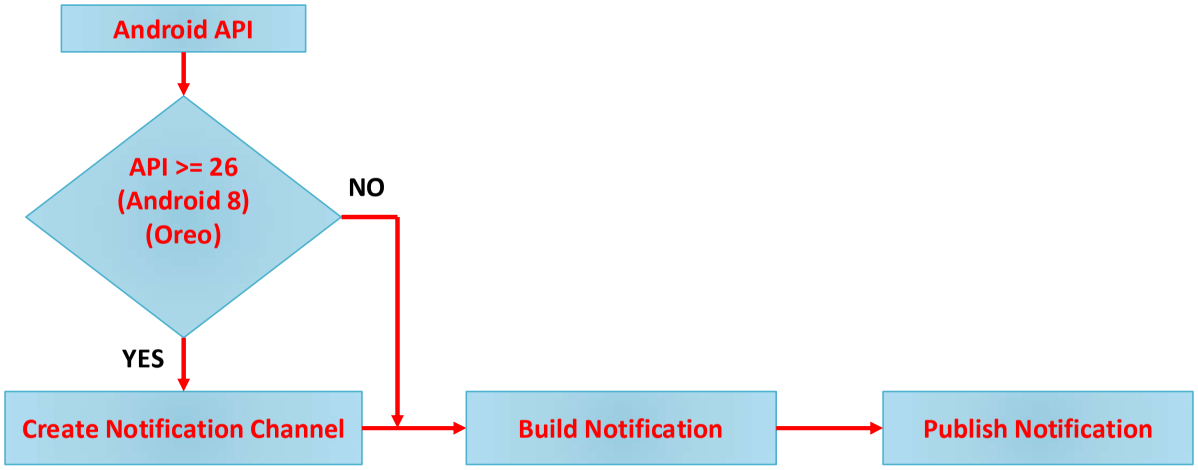


Notification Anatomy

Item		Meaning
Header area	1	The app icon: small two-dimensional representation of app
	2	The app name: automatically
	3	Header text (optional): if an app sends notifications from multiple sources, such as the account name for users with multiple accounts.
	4	Timestamp (optional): By the system
	5	Expand indicator:
Content area	6	A content title:
	7	Content text:
	8	Large icon (optional):
Action area	9	a notification may display up to three actions at the bottom.



Notification Creation



Application Class

- Application class is primarily used for initialization of global state before the first Activity is displayed
- Application class, or your subclass of the Application class, is instantiated before any other class when the process for your application/package is created.
- It is base class for maintaining global application state.
- You can provide your own implementation by creating a subclass
- Specifying the fully-qualified name of this subclass as the "android:name" attribute in your AndroidManifest.xml's <application> tag
- **onCreate():** Called when the application is starting, before any activity, service, or receiver objects (excluding content providers) have been created.

Notification Example (CreateNotification)

```
public class MyApp extends Application {
    final static String CHANNEL_ID = "MyChannelID";
    @Override
    public void onCreate() {
        super.onCreate();
        //Create Notification Channel
        createNotificationChannel();
    }
    private void createNotificationChannel() {
        //create channel for only API 26 and Higher
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            NotificationChannel myChannel = new NotificationChannel(CHANNEL_ID, "My Channel",
                    NotificationManager.IMPORTANCE_DEFAULT);
            myChannel.setDescription("This is my  channel");

            //get the service of the Notifications App
            NotificationManager manager = getSystemService(NotificationManager.class);

            //create the notification channel in the Notifications App
            manager.createNotificationChannel(myChannel);
        }
    }
}
```

Java Class

```
<application
    android:name=".MyApp"……>
.
.
/<application>
```

Manifest file

Notification Example (CreateNotification)

```
private void showNotification() {

    //create pending intent to open activity outside notification app
    //it will allow users to go directly from the notification to an activity of  app
    Intent notificationIntent = new Intent(getApplicationContext(), MainActivity.class);
    PendingIntent pendingIntent = PendingIntent.getActivity(getApplicationContext(), 0, notificationIntent, 0);

    //Build the Notification
    NotificationCompat.Builder nBuilder=new NotificationCompat.Builder(getApplicationContext(),MyApp.CHANNEL_ID);
    nBuilder.setContentTitle("Notification Title")
                .setContentText("content of notification")
                .setSmallIcon(R.drawable.ic_alert)
                .setPriority(Notification.PRIORITY_HIGH)
                .setContentIntent(pendingIntent)
                .addAction(R.drawable.ic_replay, "Relay", pendingIntent);

    //Publish the Notification
    NotificationManagerCompat nm = NotificationManagerCompat.from(getBaseContext());
    nm.notify(123, nBuilder.build());
}
```

Started Service(Foreground): Steps

1- The same as background service, with only a creation a link between the service and UI through Notifications. **(Notification closes only, if the service is stopped)**

2- Create the Notification.

3- on onStartCommand() , call startForeground() to link the notification with service.

4- You must use permission for foreground service

5- **Start Servicw:**

- **startService():** start the service while app is still opened
- **startForegroundService() [after API 26]:** from support library (ContextCompat). Start the service app while the app itself in background**.** After 5 sec, it calls startForeground(), if not (commit startForeground), service is closed.

```
public static void startForegroundService
                                    (@NonNull Context context, @NonNull Intent intent) {
    if (Build.VERSION.SDK_INT >= 26)
        context.startForegroundService(intent);
    else
        context.startService(intent);
}
```

[Ctrl + B]

Started Service(Foreground): Steps

- ✓ **It has higher priority over normal background service.**
- ✓ **In Android 8 and Higher:** Without showing UI to the user (clicking back button), Background Service is stopped after one minute to save resources; and when activity is closed, the service is stopped. This is why forground service is important.
- ✓ **The system does not kill Foreground Services even though it is running on low memory or battery, or activity is closed because notification will be still shown.**
- ✓ **Notification is gone only when the service is stopped.**
- ✓ **For each calling for startService , onStartCommand() is called BUT the onCreate() is called once.**
- ✓ **Unlike any other services, the user can interact with foreground service through notifications**
- ✓ **It is a started service, codes in onStartCommand() works in UI thread. So, heavy operations without a separate worker thread or in IntentService will cause with the app will not response**

Foreground Service Example
(ForegroundService)

```
public class MyApp extends Application {
    final static String CHANNEL_ID = "MyChannelID";
    @Override
    public void onCreate() {
        super.onCreate();
        //Create Notification Channel
        createNotificationChannel();
    }
    private void createNotificationChannel() {
        //create channel for only API 26 and Higher
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            NotificationChannel myChannel = new NotificationChannel(CHANNEL_ID, "My Channel",
                    NotificationManager.IMPORTANCE_DEFAULT);
            myChannel.setDescription("This is my  channel");

            //get the service of the Notifications App
            NotificationManager manager = getSystemService(NotificationManager.class);

            //create the notification channel in the Notifications App
            manager.createNotificationChannel(myChannel);
        }
    }
}
```

Java Class

```
<application
    android:name=".MyApp"……>
.
.
/<application>
```

Manifest file

Foreground Service Example
(ForegroundService)

```
// Start the service
public void startService(View view) {
    Intent intent = new Intent(getApplicationContext(), MyService.class);

    intent.putExtra("id", 10);
    intent.putExtra("username","admin");
    intent.putExtra("body",tvBody.getText().toString());

    ContextCompat.startForegroundService(getBaseContext(), intent);

}
// Stop the service
public void stopService(View view) {
    stopService(new Intent(this, MyService.class));
}
```

MainActivity Class

Foreground Service Example
(ForegroundService)

```
public int onStartCommand(Intent intent, int flags, int startId) {
    Log.e("you", "onStartCommand: Service is Started");
    body = intent.getStringExtra("body");
    //Link the Notification with Service
     startForeground(1, getNotificationObject());
}
```

```
private Notification getNotificationObject() {
    Intent notificationIntent = new Intent(getApplicationContext(), MainActivity.class);
    PendingIntent pendingIntent = PendingIntent.getActivity(getApplicationContext(), 0, notificationIntent, 0);
    //Build the Notification
    NotificationCompat.Builder nBuilder = new NotificationCompat.Builder(getApplicationContext(), MyApp.CHANNEL_ID);
    nBuilder.setContentTitle("Notification Title")
            .setContentText(body)
            .setSmallIcon(R.drawable.ic_replay)
            .setPriority(Notification.PRIORITY_HIGH)
            .setContentIntent(pendingIntent)
            .addAction(R.drawable.ic_replay, "Relay", pendingIntent);
    return nBuilder.build();
}
```


Application Class

- A bound service can be also a started service. But a start service can not be a bound service.
- Started Service is started by calling:
  - startService (for Background)
  - startForegroundService (for Foreground)
- Bound Service is started by calling:
  - When other component binds to it
  - startService or startForegroundService and THEN bind to it
-

Application Class

- Bound service: Client-Server Interaction
  - Start a Server and bind to it with some kind of client
  - The service acts as the Server
  - Other component acts as a Client
  - Client can be Activity, Fragment, a smart watch or other application
- Bound service is used when consistent or frequent communication between some client and the service such as playing progress of video/audio using the seek bar which is a data coming from a service and sending to the activity or fragment.

Bound Service: Steps

1- Extend **Service** class

2- add to the manifest file

3- Extending the Binder class inside the service

a) In your service, **create an instance of Binder** that does one of the following:
- Contains public methods that the client can call.
- Returns the current Service instance, which has public methods the client can call.
- Returns an instance of another class hosted by the service with public methods the client can call.

b) Return this **instance of Binder** from the **onBind()** callback method.

c) In the **client(activity**), receive the Binder from the **onServiceConnected()** callback method that inside the private **ServiceConnection** object and make calls to the bound service using the methods provided.

4- Where to Bound/Unbound:
- **onCreat()/onDestroy():** bound/unbound even if the Activity is not in foreground (background)
- **onStart()/onStop():** bound/unbound when only activity is active to user (foreground)

Bound Service Example: (BoundService)

```
//create log tag for debugging (logt)
  private static String LOG_TAG = "BoundService";

  //create object of the class that return the instance of BoundService
  private IBinder mBinder = new MyBinder();

  //return the object that return the instance of BoundService
  @Override
  public IBinder onBind(Intent intent) {
      Log.v(LOG_TAG, "in onBind");
      return mBinder;
  }
  //Return instance of Service (Singleton) to Client.
  public class MyBinder extends Binder {
      public BoundService getService() {
          return BoundService.this;
      }
  }
```

BoundService Class

Bound Service Example: (BoundService)

```
//Chronometer implements a simple timer
private Chronometer mChronometer;

@Override
public void onCreate() {
    super.onCreate();
    Log.v(LOG_TAG, "in onCreate");
    mChronometer = new Chronometer(this);

   //set the Chronometer to start from zero (when service is created). By default, it starts from app running time
    mChronometer.setBase(SystemClock.elapsedRealtime());
    mChronometer.start();
}
public String getTimestamp() {
    //get the time of running service (in ms)
    long elapsedMillis = SystemClock.elapsedRealtime() - mChronometer.getBase();
    //split this time to hours, minutes, seconds, milli seconds
    int hours = (int) (elapsedMillis / 3600000);
    int minutes = (int) (elapsedMillis - hours * 3600000) / 60000;
    int seconds = (int) (elapsedMillis - hours * 3600000 - minutes * 60000) / 1000;
    int millis = (int) (elapsedMillis - hours * 3600000 - minutes * 60000 - seconds * 1000);

    return hours + ":" + minutes + ":" + seconds + ":" + millis;
}
```

BoundService Class

Bound Service Example: (BoundService)

```
//call when activity is in onStop()
@Override
public void onRebind(Intent intent) {
    Log.v(LOG_TAG, "in onRebind");
    super.onRebind(intent);
}
//call when activity move from onStop() to onStart()
@Override
public boolean onUnbind(Intent intent) {
    Log.v(LOG_TAG, "in onUnbind");
    return true;
}
@Override
public void onDestroy() {
    super.onDestroy();
    Log.v(LOG_TAG, "in onDestroy");
    mChronometer.stop();
}
```

BoundService Class

Bound Service Example: (BoundService)

```
//variable of BoundService to store instance of running bound service
BoundService mBoundService;

//flag to indicate bound/unbound
boolean isBound = false;

//Call when client bound to/unbound from the service
private ServiceConnection mServiceConnection = new ServiceConnection() {
  @Override
  public void onServiceConnected(ComponentName name, IBinder service) {
    //IBinder is the Link/interface between service and client
    BoundService.MyBinder myBinder = (BoundService.MyBinder) service;
    //get instance of running bound service
    mBoundService = myBinder.getService();
    isBound = true;
  }
  @Override
  public void onServiceDisconnected(ComponentName name) {
    isBound = false;
  }
};
```

MainActivity Class

Bound Service Example: (BoundService)

```
//Bound/Unbound in onStart()/onStop()
@Override
protected void onStart() {
  super.onStart();
  Intent intent = new Intent(this, BoundService.class);
//start service
  startService(intent);
//bound the service
  bindService(intent, mServiceConnection, Context.BIND_AUTO_CREATE);
}
@Override
protected void onStop() {
  super.onStop();
  if (isBound && mBoundService != null) {
    unbindService(mServiceConnection);
    isBound = false;
  }
}
```

MainActivity Class

Bound Service Example: (BoundService)

```
protected void onCreate(Bundle savedInstanceState) {
  btnPrintTimestamp.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
      if (isBound) {
        tvTimestampText.setText(mBoundService.getTimestamp());
      }
    }
  });
  btnStopService.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
      //unbound
      if (isBound && mBoundService != null) {
        unbindService(mServiceConnection);
        isBound = false;
      }
      //stop service
      Intent intent = new Intent(MainActivity.this, BoundService.class);
      stopService(intent);
    }
  });
}
```

MainActivity Class

Job Service

- **AlarmManager (Deprecated):**
  - Execute operation during predefined time/period.
  - e.g. repeat notification every 4 hour. AlarmManager exhausts resources.
  - So, JobScheduler is developed.
- **JobScheduler:**
  - A class to schedule triggered events or operations in certain times for certain period or both.
  - You can put conditions to be executed such as when power is charging/Internet is connected.
  - e.g backup every 24 hours when WiFi is conncted.
  - JobScheduler is done using a JobService/JobIntentService.

Job Service: Steps

1- Create JobService Class:

- extends JobService
- must be declared in Manifest

```
<service
    android:name=".MyJobService"
    android:permission="android.permission.BIND_JOB_SERVICE" />
```

- must take BIND JOB SERVICE permission
- Three main methods
  - **onStartJob:** called **by the system** when service is started. **Return False:** if the code inside it is short and is execute inside the main (UI) thread. **Return true:** if the code inside it is long time and is executed inside worker thread so that the system waits and does not finishes the service.
  - **jobFinished:** must be **called manually** when the service is finished so that the system finishes the service
  - **onStopJob:** called **by the system** when service is cancelled (**Not Finished**), for example when any conditions are not met during execution. **Return true:** restart the job. **Return False:** do not restart the job.

Job Service: Steps

2- Define the JobService inside a component

3- Create JobInfo to set the conditions and times

setPersisted (): *keep the job even device is rebooted*. It needs:

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

4- Make JobScheduler to schedule the job

Job Service: (JobService)

1- Create JobService Class:

```
public class MyJobService extends JobService {
    private static final String TAG = "JobService";
    private boolean jobCancelled = false;
    @Override
    public boolean onStartJob(JobParameters params) {
        Log.e(TAG, "Job started");
        doBackgroundWork(params);
        //Long background operation. Keep the service runnin
          finishes it using jobFinished
        return true;
    }
    public boolean onStopJob(JobParameters params) {
        Log.e(TAG, "Job cancelled before completion (Stopped
        jobCancelled = true;
        return false;
    }
    private void doBackgroundWork(JobParameters params) {
        new Thread(new Runnable() {
            public void run() {
                for (int i = 0; i < 10; i++) {
                    if(jobCancelled){
                        return;
                    }
                    Log.e(TAG, "Run: " + i);
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                Log.e(TAG, "Job Finished");
                //finish job because code execution is finished.
                jobFinished(params, false);
            }
        }).start();
    }
}
```

Job Service: (JobService)

```
//define the JobService inside a component
ComponentName cn = new ComponentName(getBaseContext(), MyJobService.class);

//determine all conditions and times
JobInfo info;
if (Build.VERSION.SDK_INT <= Build.VERSION_CODES.N) {
    info = new JobInfo.Builder(10, cn)
            .setPeriodic(5000)
            .setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED)
            .setRequiresCharging(true)
            .setPersisted(true)  //keep the job even device is rebooted
            .build();
} else {
    info = new JobInfo.Builder(10, cn)
            .setMinimumLatency(10000)
            .setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED)
            .setRequiresCharging(true)
            .build();
}
```

2- Define the JobService inside a component

3- Create JobInfo to set the conditions and times

Job Service: (JobService)

4- Make JobScheduler to schedule the job

```
//Schedule the Job
JobScheduler scheduler = (JobScheduler) getSystemService(JOB_SCHEDULER_SERVICE);
int resultCode = scheduler.schedule(info);
if (resultCode == JobScheduler.RESULT_SUCCESS) {
    Log.e(TAG, "Job scheduled");
} else {
    Log.e(TAG, "Job scheduled failed");
}
```

5- Cancel the scheduling for the job

```
JobScheduler scheduler = (JobScheduler)
getSystemService(JOB_SCHEDULER_SERVICE);
scheduler.cancel(10);
Log.e(TAG, "Scheduler cancelled");
```