MOBILE APPLICATION DEVELOPMENT (MAD) – LAB 8

Objectives:

- Types of processes and its lifecycle
- To understand foreground service
- To understand Alarm Manager
- Practice Activities

OBJECTIVE 1: Understanding Types of processes and its lifecycle

1. A **foreground process** is one that is required for what the user is currently doing. Various application components can cause its containing process to be considered foreground in different ways. A process is considered to be in the foreground if any of the following conditions hold:
   - It is running an Activity at the top of the screen that the user is interacting with (its **onResume**() method has been called).
   - It has a **BroadcastReceiver** that is currently running (its **BroadcastReceiver.onReceive**() method is executing).
   - It has a Service that is currently executing code in one of its callbacks (**Service.onCreate**(), **Service.onStart**(), or **Service.onDestroy**()).

2. A **visible process** is doing work that the user is currently aware of, so killing it would have a noticeable negative impact on the user experience. A process is considered visible in the following conditions:
   - It is running an Activity that is visible to the user on-screen but not in the foreground (its **onPause**() method has been called). This may occur, for example, if the foreground Activity is displayed as a dialog that allows the previous Activity to be seen behind it.
   - It has a Service that is running as a foreground service, through **Service.startForeground**() (which is asking the system to treat the service as something the user is aware of, or essentially visible to them).
   - It is hosting a service that the system is using for a particular feature that the user is aware, such as a live wallpaper, input method service, etc.

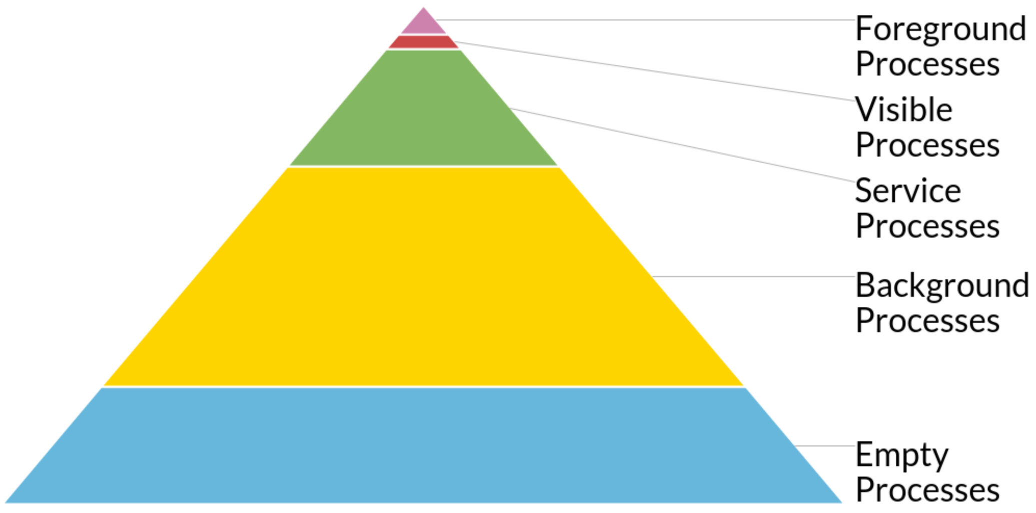
3. A **service process** is one holding a Service that has been started with the **startService**() method. Though these processes are not directly visible to the user, they are generally doing things that the user cares about (such as background network data upload or download), so the system will always keep such processes running unless there is not enough memory to retain all foreground and visible processes.
   - Services that have been running for a long time (such as 30 minutes or more) may be demoted in importance to allow their process to drop to the cached list described next. This helps avoid situations where long running services that use excessive resources (for example, by leaking memory) prevent the system from delivering a good user experience.

4. A cached process is one that is not currently needed, so the system is free to kill it as desired when resources like memory are needed elsewhere. In a normally behaving system, these are the only processes involved in resource management: a well running system will have multiple cached processes always available (for more efficient switching between applications) and regularly kill

- ANDROID BACKGROUND MEDIAPLAYER

the oldest ones as needed. Only in very critical (and *undesireable*) situations will the system get to a point where all cached processes are killed and it must start killing service processes.

- These processes often hold one or more Activity instances that are not currently visible to the user (the **onStop**() method has been called and returned). Provided they implement their Activity life-cycle correctly (see Activity for more details), when the system kills such processes it will not impact the user's experience when returning to that app: it can restore the previously saved state when the associated activity is recreated in a new process.
- These processes are kept in a list. The exact policy of ordering on this list is an implementation detail of the platform, but generally it will try to keep more useful processes (one hosting the user's home application, the last activity they saw, etc) before other types of processes. Other policies for killing processes may also be applied: hard limits on the number of processes allowed, limits on the amount of time a process can stay continually cached, etc



- [ANDROID BACKGROUND MEDIAPLAYER]

OBJECTIVE 2: Understanding foreground Process

Foreground Service

For more clarity let's takes example Gmail, You are just using Gmail app and listening to music that is being played by Music Player application. So Music Player is basically using the foreground service to play the music. The foreground service always uses the notification to notify the user and using the notification you can actually interact with the service or the ongoing operation such as pause the music or play the next music.

So whenever in your app you see a notification that is performing some long running tasks that service is basically the foreground service and Foreground Service is always noticeable to the user that is the user is aware of this ongoing process.

- ANDROID BACKGROUND MEDIAPLAYER

Steps:

1. **Create a subclass of Service – ForgroundService.java**

```
public class ForegroundService extends Service {
    public static final String CHANNEL_ID = "ForegroundServiceChannel";

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

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        String input = intent.getStringExtra("inputExtra");
        createNotificationChannel();
        Intent notificationIntent = new Intent(this, MainActivity.class);
        PendingIntent pendingIntent = PendingIntent.getActivity(this,
                0, notificationIntent, 0);

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

        startForeground(1, notification);

        //do heavy work on a background thread


        //stopSelf();

        return START_NOT_STICKY;
    }

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

    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
```

- ANDROID BACKGROUND MEDIAPLAYER

2. **Create Notification Channel, and add following method in ForegroundService class**

```java
private void createNotificationChannel() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            NotificationChannel serviceChannel = new NotificationChannel(
                    CHANNEL_ID,
                    "Foreground Service Channel",
                    NotificationManager.IMPORTANCE_DEFAULT
            );

            NotificationManager manager = getSystemService(NotificationManager.class);
            manager.createNotificationChannel(serviceChannel);
        }
    }
```

3. **Modify the onstartCommand Method of ForegroundService Class**

```java
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    String input = intent.getStringExtra("inputExtra");
    createNotificationChannel();
    Intent notificationIntent = new Intent(this, MainActivity.class);
    PendingIntent pendingIntent = PendingIntent.getActivity(this,
            0, notificationIntent, 0);

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

    startForeground(1, notification);

    //do heavy work on a background thread


    //stopSelf();

    return START_NOT_STICKY;
}
```

4. **Declare your service and add uses permission in Manifest.xml**

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

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <service
        android:name=".ForegroundService"
        android:enabled="true"
        android:exported="true"></service>

    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

- ANDROID BACKGROUND MEDIAPLAYER

5. Open MainActivity and add onClickListener on button

```java
public class MainActivity extends AppCompatActivity {
    Button btnStartService, btnStopService;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btnStartService = findViewById(R.id.buttonStartService);
        btnStopService = findViewById(R.id.buttonStopService);

        btnStartService.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                startService();
            }
        });

        btnStopService.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                stopService();
            }
        });
    }

    public void startService() {
        Intent serviceIntent = new Intent(this, ForegroundService.class);
        serviceIntent.putExtra("inputExtra", "Foreground Service Example in Android");

        ContextCompat.startForegroundService(this, serviceIntent);
    }

    public void stopService() {
        Intent serviceIntent = new Intent(this, ForegroundService.class);
        stopService(serviceIntent);
    }
}
```

- ANDROID BACKGROUND MEDIAPLAYER

OBJECTIVE 3 – Understanding Alarm Manager

`AlarmManager` provides access to the system alarm services. These allow you to schedule your application to be run at some point in the future. When an alarm goes off, the Intent that had been registered for it is broadcast by the system that you can receive inside your activity. Registered alarms are retained while the device is asleep (and can optionally wake the device up if they go off during that time), but will be cleared if it is turned off and rebooted. Alarm has been set automatically on boot completed.

How to use

Step 1. Add the `JitPack` repository to your build file. Add it in your root `build.gradle` at the end of repositories:

```
allprojects {
    repositories {
    ...
    maven { url 'https://jitpack.io' }
    }
}
```

Step 2. Add the dependency

```
dependencies {
     implementation 'com.github.zubairehman:AlarmManager:v1.0.0-alpha01'
}
```

Using `AlarmManager` is really simple and easy, all you need to do is to pass in your configuration and start getting notifications in your activity/fragment. This could further be illustrated in the example below:

```
AlarmBuilder().with(context)
               .setTimeInMilliSeconds(TimeUnit.SECONDS.toMillis(10))
               .setId("UPDATE_INFO_SYSTEM_SERVICE")
               .setAlarmType(AlarmType.REPEAT)
               .setAlarm()
```

Note that you can get a builder object for later use, as shown in the code below:

- ANDROID BACKGROUND MEDIAPLAYER

Register listener

In-order to get notified, you need to register a listener in `onResume` as shown below:

```
override fun onResume() {
    super.onResume()
    builder?.addListener(this)
}
```

Un-Register listener

In-order to prevent your activity listening to un-attended events you need to un-register your listener in `onPause` as shown below:

```
override fun onPause() {
    super.onPause()
    builder?.removeListener(this)
}
```

Cancel alarm

In-order to cancel your alarm just call `cancelAlarm()` as shown below:

```
builder?.cancelAlarm()
```

Callback

The `perform()` method will be called once the desired time has been reached, you can write your logic in this method as shown below:

```
override fun perform(context: Context, intent: Intent) {
    Timber.i("Do your work here")
}
```

- ANDROID BACKGROUND MEDIAPLAYER

Complete Example:

Here is the complete code that demonstrate the use of AlarmManager:

```
class MainActivity : AppCompatActivity(), AlarmListener {

    //Alarm builder
    var builder: AlarmBuilder? = null;

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        //creating alarm builder
        builder = AlarmBuilder().with(this)
                .setTimeInMilliSeconds(TimeUnit.SECONDS.toMillis(10))
                .setId("UPDATE_INFO_SYSTEM_SERVICE")
                .setAlarmType(AlarmType.REPEAT)

        //setting click listeners
        btnSetAlarm.setOnClickListener {
            builder?.setAlarm()
        }

        btnCancelAlarm.setOnClickListener {
            builder?.cancelAlarm()
        }
    }

    override fun onResume() {
        super.onResume()
        builder?.addListener(this)
    }

    override fun onPause() {
        super.onPause()
        builder?.removeListener(this)
    }

    override fun perform(context: Context, intent: Intent) {
        Log.i("Alarm", "Do your work here")
    }
}
```

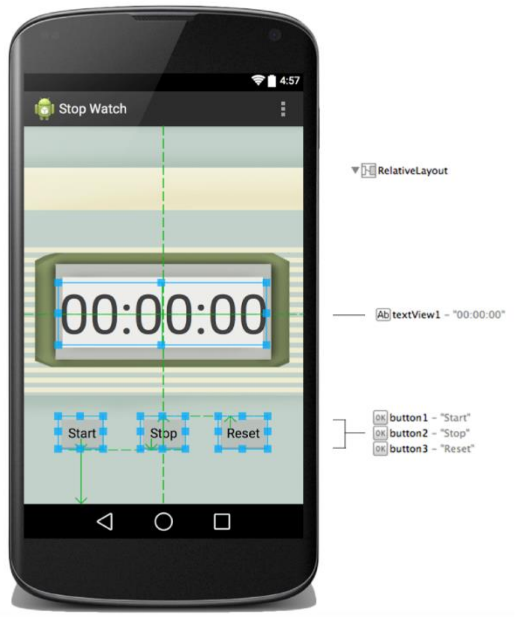
- ANDROID BACKGROUND MEDIAPLAYER

MOBILE APPLICATION DEVELOPMENT (MAD) – LAB 8

OBJECTIVE 4: ACTIVITIES.

Activity 1: Create a Stop Watch.

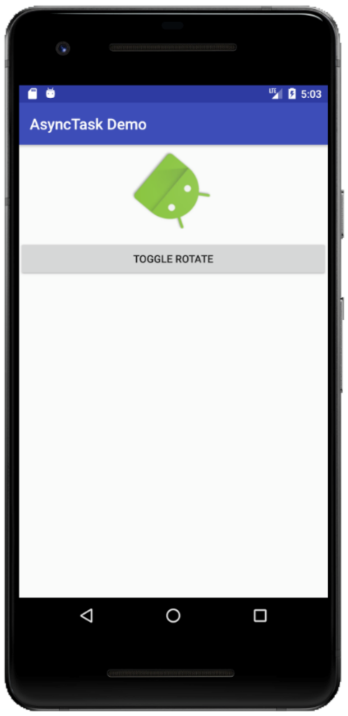
- Create the following application using Thread / Services etc for all background tasks.



- ANDROID BACKGROUND MEDIAPLAYER

Activity 2: Create a Stop Watch.

Create the following application using Thread / Service. When the user hits TOGGLE ROTATE button, the image begin a rotation animation. This button works as a toggle mechanism, which means the image stops a rotation animation by hitting the button again. The rotation will begin where it leaves off in the previous animation.



- [ANDROID BACKGROUND MEDIAPLAYER]

Activity 3: Create a Media Player using Foreground Service

Create a mediaplayer using foreground service, which will constantly show the song being currently played in Notification bar.

- ANDROID BACKGROUND MEDIAPLAYER

Activity 4: Create a simple alaram application, which will notifiy user at specific time

Use the alaram manager to create an alarm service which will notify user on specific times.

- ANDROID BACKGROUND MEDIAPLAYER