PDP ONLINE

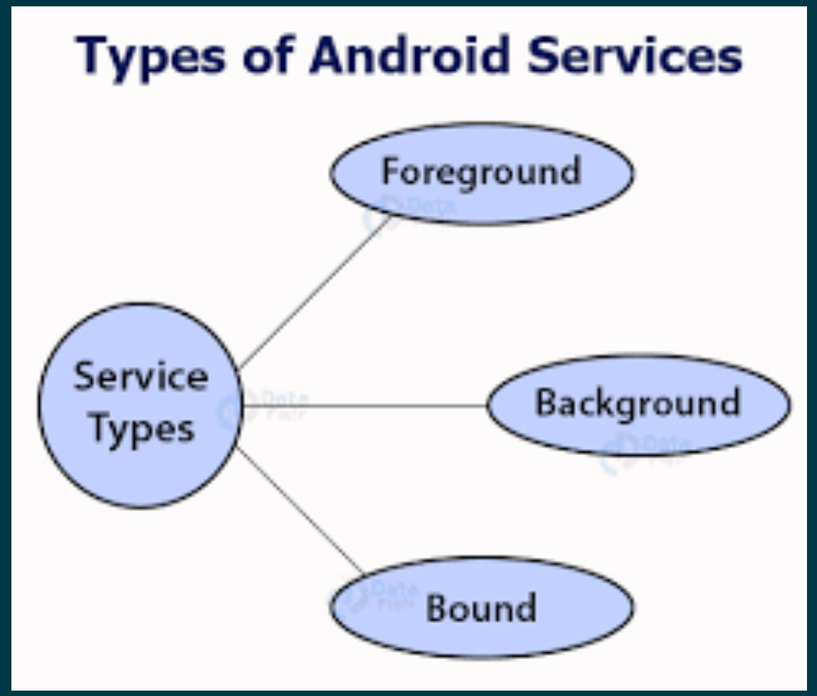
Android – Android Intermediate

Service & Broadcast Receiver

Android Service

In android, **Service** is a component which keep an app running in the background to perform long-running operations based on our requirements

Types of Android Services



Foreground

Type of services that perform operations in the background that is **noticeable for the users.**

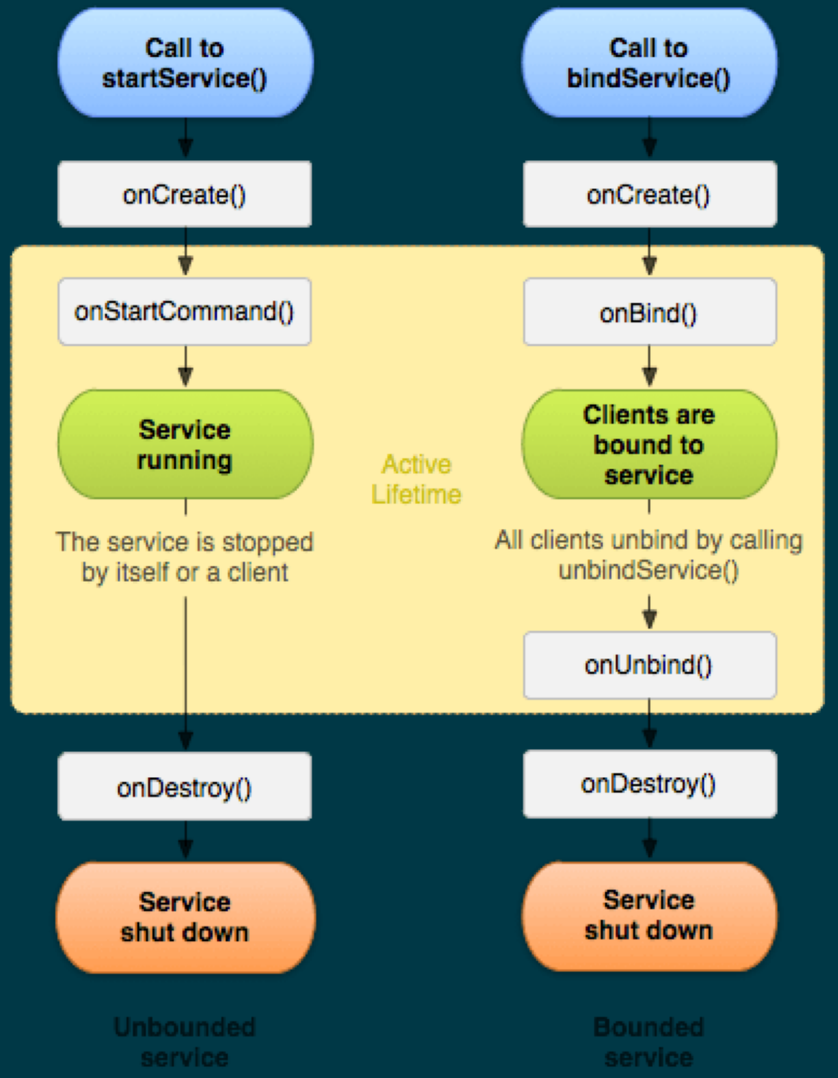
Background

Type of services that perform operations in the background that is not giving any information to user via notification.

Bound

A **bound service** is one that’s bound to another application component, such as an activity.

Lifecycle of Service



In android, the life cycle of service will follow two different paths Started or Bound.

Sample Service (Java)

```java
public class SampleService extends Service {
    int startMode;       // indicates how to behave if the service is killed
    IBinder binder;      // interface for clients that bind
    boolean allowRebind; // indicates whether onRebind should be used

    @Override
    public void onCreate() {
        // The service is being created
    }

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

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

    @Override
    public boolean onUnbind(Intent intent) {
        // All clients have unbound with unbindService()
        return allowRebind;
    }
```

Sample Service (Kotlin)

```kotlin
class SampleService : Service() {
    var startMode // indicates how to behave if the service is killed
            = 0
    var binder // interface for clients that bind
            : IBinder? = null
    var allowRebind // indicates whether onRebind should be used
            = false

    override fun onCreate() {
        // The service is being created
    }

    override fun onStartCommand(intent: Intent, flags: Int, startId: Int): Int {
        // The service is starting, due to a call to startService()
        return startMode
    }

    override fun onBind(intent: Intent): IBinder? {
        // A client is binding to the service with bindService()
        return binder
    }

    override fun onUnbind(intent: Intent): Boolean {
        // All clients have unbound with unbindService()
        return allowRebind
    }
```

Started Service (Java)

```java
public class StartedService extends Service {
    private MediaPlayer player;

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

    @Override
    public void onCreate() {
        Toast.makeText( context: this,  text: "Started Service Created", Toast.LENGTH_LONG).show();
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        player = MediaPlayer.create( context: this, Settings.System.DEFAULT_RINGTONE_URI);
        // This will play the ringtone continuously until we stop the service.
        player.setLooping(true);
        // It will start the player
        player.start();
        Toast.makeText( context: this,  text: "Service Started", Toast.LENGTH_LONG).show();
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        // Stopping the player when service is destroyed
        player.stop();
        Toast.makeText( context: this,  text: "Started Service Stopped", Toast.LENGTH_LONG).show();
    }
}
```

Started Service (Kotlin)

```kotlin
class StartedService : Service() {
    private var player: MediaPlayer? = null

    override fun onBind(intent: Intent): IBinder? {
        return null
    }

    override fun onCreate() {
        Toast.makeText( context: this,  text: "Started Service Created", Toast.LENGTH_LONG).show()
    }

    override fun onStartCommand(intent: Intent, flags: Int, startId: Int): Int {
        player = MediaPlayer.create( context: this, Settings.System.DEFAULT_RINGTONE_URI)
        // This will play the ringtone continuously until we stop the service.
        player!!.setLooping(true)
        // It will start the player
        player!!.start()
        Toast.makeText( context: this,  text: "Service Started", Toast.LENGTH_LONG).show()
        return START_STICKY
    }

    override fun onDestroy() {
        super.onDestroy()
        // Stopping the player when service is destroyed
        player!!.stop()
        Toast.makeText( context: this,  text: "Started Service Stopped", Toast.LENGTH_LONG).show()
    }
}
```

Bound Service (Java)

```java
public class BoundService extends Service {
    private IBinder mBinder = new MyBinder();
    private Chronometer mChronometer;

    @Override
    public void onCreate() {
        super.onCreate();
        Toast.makeText( context: this,  text: "Bound Service Created", Toast.LENGTH_LONG).show();
        mChronometer = new Chronometer( context: this);
        mChronometer.setBase(SystemClock.elapsedRealtime());
        mChronometer.setOnChronometerTickListener(new Chronometer.OnChronometerTickListener() {
            @Override
            public void onChronometerTick(Chronometer chronometer) {
                Toast.makeText(getApplication(), String.valueOf(chronometer.getBase()), Toast.LENGTH_LONG).show();
            }
        });
        mChronometer.start();
    }

    @Override
    public IBinder onBind(Intent intent) {
        Toast.makeText( context: this,  text: "Bound Service onBind", Toast.LENGTH_LONG).show();
        return mBinder;
    }

    @Override
    public void onRebind(Intent intent) {
        Toast.makeText( context: this,  text: "Bound Service onRebind", Toast.LENGTH_LONG).show();
        super.onRebind(intent);
    }
```

Bound Service (Kotlin)

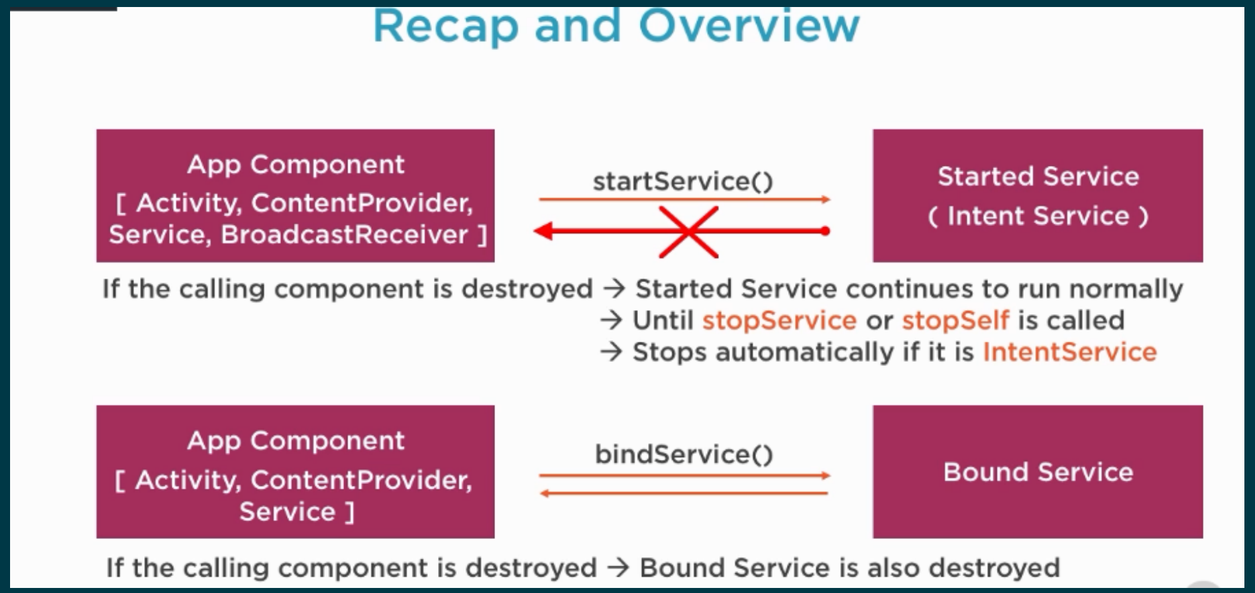
```kotlin
class BoundService : Service() {
    private val mBinder: IBinder = MyBinder()
    private var mChronometer: Chronometer? = null

    override fun onCreate() {
        super.onCreate()
        Toast.makeText( context: this,  text: "Bound Service Created", Toast.LENGTH_LONG).show()
        mChronometer = Chronometer( context: this)
        mChronometer!!.base = SystemClock.elapsedRealtime()
        mChronometer!!.onChronometerTickListener =
            OnChronometerTickListener { chronometer ->
                Toast.makeText(
                    application,
                    chronometer.base.toString(),
                    Toast.LENGTH_LONG
                ).show()
            }
        mChronometer!!.start()
    }

    override fun onBind(intent: Intent): IBinder? {
        Toast.makeText( context: this,  text: "Bound Service onBind", Toast.LENGTH_LONG).show()
        return mBinder
    }

    override fun onRebind(intent: Intent) {
        Toast.makeText( context: this,  text: "Bound Service onRebind", Toast.LENGTH_LONG).show()
        super.onRebind(intent)
    }
```

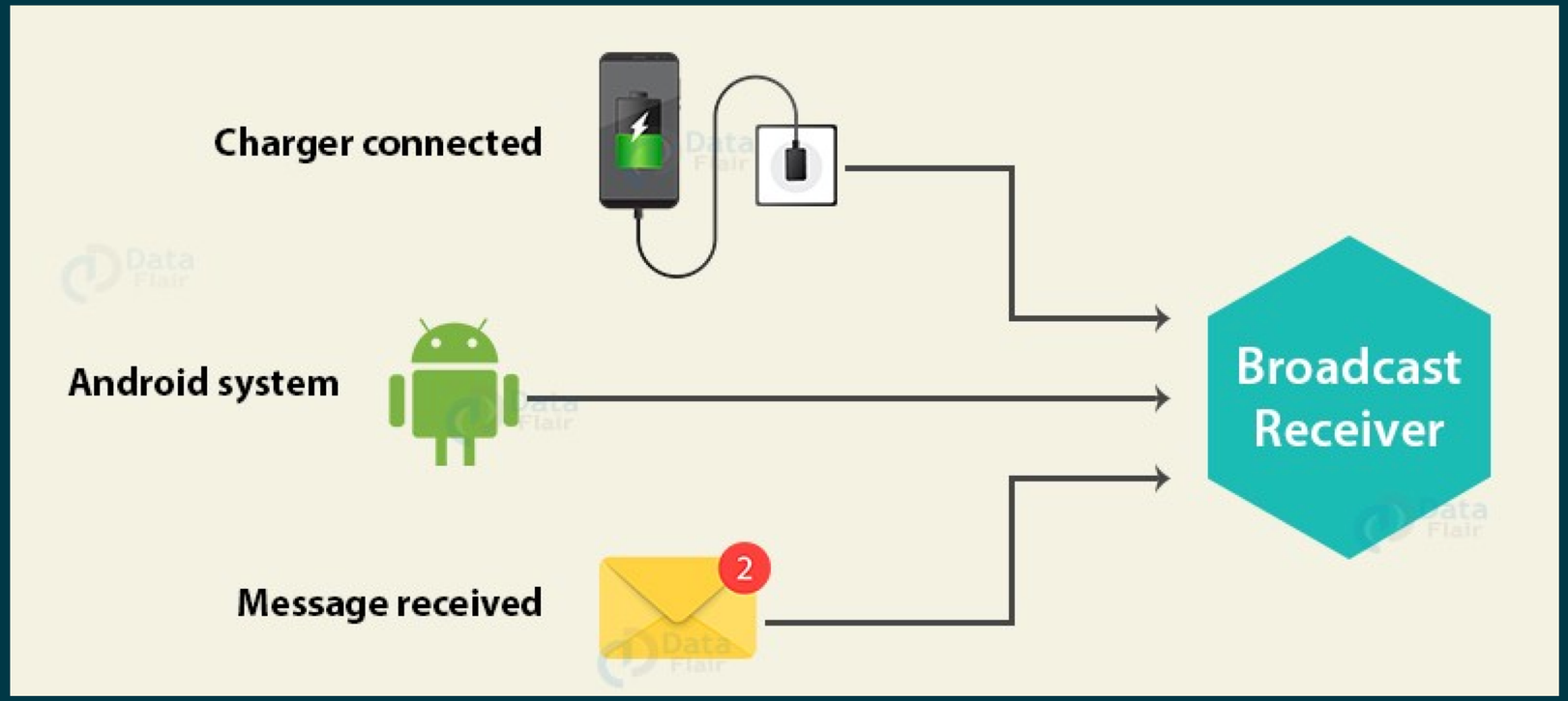
Started Service & Bound Service



Started Service vs Bound Service

Started Service / IntentService	Bound Service
Just to accomplish task [May be Long Task]	For long-standing connection
Invoked by startService()	Invoked by bindService()
onBind() returns null	onBind() returns IBinder
Continue to run even if the calling component is destroyed	If the calling component is destroyed then Bound Services too gets destroyed

Why Broadcast Receivers?



Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself.

Static Receiver (Network) – (Java, Kotlin)

```java
public class NetworkBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {

        if (ConnectivityManager.CONNECTIVITY_ACTION.equals(intent.getAction())) {
            Toast.makeText(context, text: "Connectivity changed", Toast.LENGTH_SHORT).show();
        }
    }
}
```

```xml
<receiver
    android:name=".activity.receivers.NetworkBroadcastReceiver"
    android:exported="false">
    <intent-filter>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
    </intent-filter>
</receiver>
```

```kotlin
class NetworkBroadcastReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        if (ConnectivityManager.CONNECTIVITY_ACTION == intent.action) {
            Toast.makeText(context, text: "Connectivity changed", Toast.LENGTH_SHORT).show()
        }
    }
}
```

API level 26 or higher

From the android documentation :

https://developer.android.com/guide/components/broadcast-exceptions

> As part of the Android 8.0 (API level 26) Background Execution Limits, apps that target the API level 26 or higher can no longer register broadcast receivers for implicit broadcasts in their manifest. However, several broadcasts are currently exempted from these limitations. Apps can continue to register listeners for the following broadcasts, no matter what API level the apps target.

and

https://developer.android.com/distribute/best-practices/develop/target-sdk

> Google Play will require that new apps target at least Android 8.0 (API level 26) from August 1, 2018, and that app updates target Android 8.0 from November 1, 2018.

Dynamic Receiver (Network) – (Java, Kotlin)

```java
@Override
protected void onStart() {
    super.onStart();
    IntentFilter filter = new IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION);
    registerReceiver(receiver, filter);
}

@Override
protected void onDestroy() {
    super.onDestroy();
    unregisterReceiver(receiver);
}
```

```kotlin
override fun onStart() {
    super.onStart()
    val filter = IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION)
    registerReceiver(receiver, filter)
}

override fun onDestroy() {
    super.onDestroy()
    unregisterReceiver(receiver)
}
```

Task 1:
Intent Service ni mustaqil o`rganib tushuntirib bering.
Java, Kotlin

Task 2:
android.intent.action.CALL uchun Broadcast Receiver yaratib ko`rsating.
Java, Kotlin

Task 3:
android.provider.Telephony.SMS_RECEIVED uchun Broadcast Receiver yaratib ko`rsating.
Java, Kotlin

Task 4:
android.intent.action.BOOT_COMPLETED uchun Broadcast Receiver yaratib ko`rsating.
Java, Kotlin