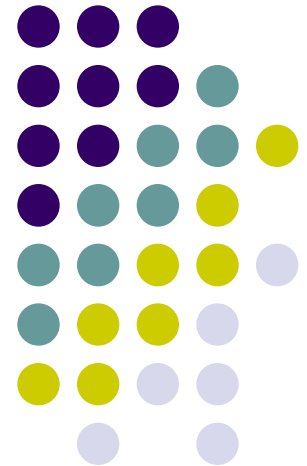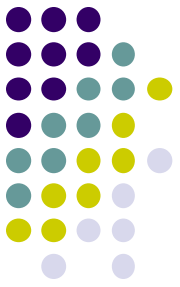# Mobile Application Develpment

Background Tasks in Android

Service

MOBILE APPLICATION
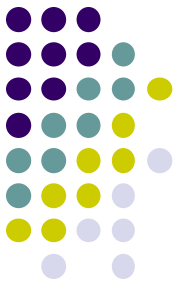DEVELOPMENT

# Background Tasks in Android

- Background Tasks in Android

  - Service Life Cycle

  - Unbound Service

  - Bound Service

  - Intent & Intent Filter

  - Broadcast Receiver

A service can essentially take two forms:

- **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**.

  - **Usually**, a started service performs a single operation and does not return a result to the caller.

  - **For example**,

    - it might **playing music**. When the operation is done, the service should stop itself.

A service can essentially take two forms: (Cont.)
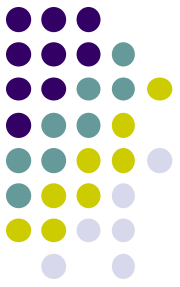
- **Bound:** A service is "**bound**"

  - 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 bound service** runs only as long as another application component is bound to it.

  - Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

Service - Life Cycle

a service has lifecycle callback methods
that you can implement to monitor changes
in the service's state and perform work
at the appropriate times.

- **The entire lifetime of a service:**
  happens between the time *onCreate()*
  is called and the time *OnDestroy()* returns
- **The active lifetime of a service**
  begins with a call to either
  onStartCommand() or *onBind()* ends
  the same time that the entire lifetime
  Ends. If the service is bound, the active
 lifetime ends when *onUnbind()* returns.
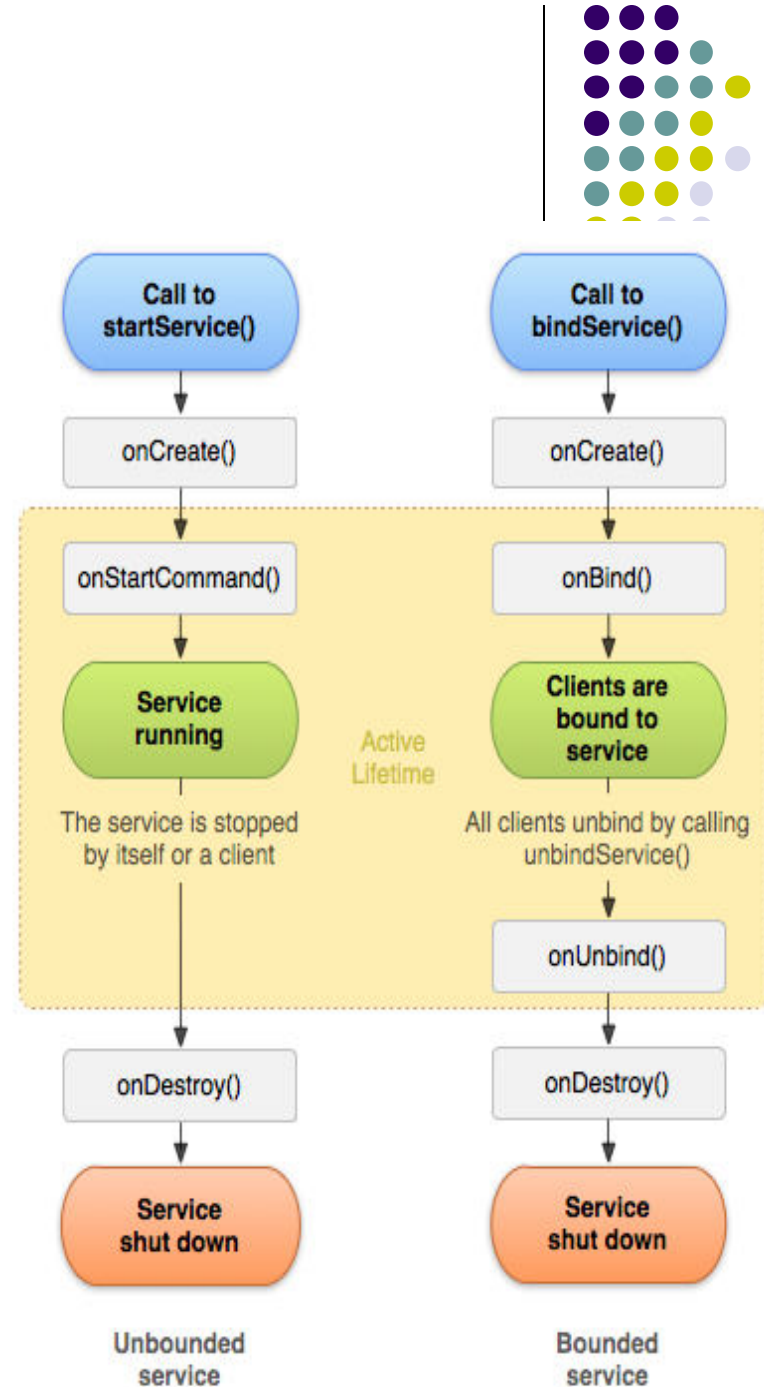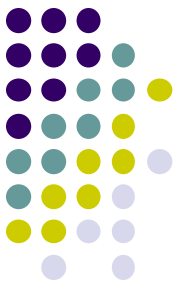
Life Cycle methods are:

- **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.

- **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*.
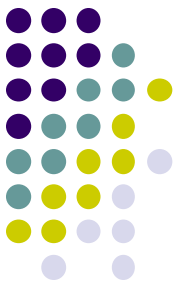
Life Cycle methods (Cont.)

- **onUnbind()** The system calls this method when **all clients have disconnected** from a particular interface published by the service.

- **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)*.

- **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.

- **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.
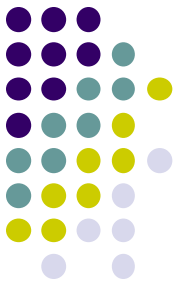
Example:

- **Start a Service that explicitly performs the following:**
  - Continuously playing a ringtone.
  - Stops playing a ringtone.

Creating User Interface

- Once the project is loaded come inside **activity_main.xml** and create the following layout.

<?xml version="1.0" encoding="utf-8"?>

<RelativeLayoutxmlns:android="http://schemas.android.com/apk/res/android"

....

```
<Button
    android:id="@+id/buttonStart"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Start Service" />
<Button
    android:id="@+id/buttonStop"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Stop Service" />
</LinearLayout>
</RelativeLayout>
```

My ApplicationService

START SERVICE

STOP SERVICE

MainActivity.java

package com.example.hp1000**.myapplicationservice***;*

import …;

public class **MainActivity** extends **AppCompatActivity** implements **View.OnClickListener** {

  private Button **buttonStart**;
  private Button **buttonStop**;

  @Override
  protected void **onCreate**(Bundle savedInstanceState) **{**
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
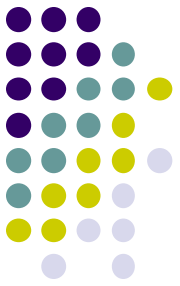   **buttonStart** = (Button) findViewById(R.id.buttonStart);
    buttonStop = (Button) findViewById(R.id.buttonStop);
    buttonStart.setOnClickListener(this); //attaching onclicklistener to buttons
    buttonStop.setOnClickListener(this);
  }

MainActivity.java

```java
@Override
public void onClick(View view) {
    if (view == buttonStart) {
        startService(new Intent(this, MyService.class));    //starting service
    } else if (view == buttonStop) {
        stopService(new Intent(this, MyService.class));    //stopping service
    }
}
}
```

Creating Service

```java
package com.example.hp1000.myapplicationservice;
import …;
 public class MyService extends Service {
        private MediaPlayer player;  //creating a mediaplayer object
        @Override
        public Ibinder onBind(Intent intent) {
                return null;
        }
        @Override
        public int onStartCommand(Intent intent, int flags, int startId) {
            player = MediaPlayer.create(this,Settings.System.DEFAULT_RINGTONE_URI  ) ;
                                                //getting systems default ringtone
            player.setLooping(true);   //this will make the ringtone continuously playing
            player.start(); //staring the player
            return START_STICKY;  //start sticky means service explicitly started and stopped
        }
        @Override
        public void onDestroy() {
                super.onDestroy();
                player.stop(); //stopping the player when service is destroyed
        }
}
```

Defining Service in Manifest
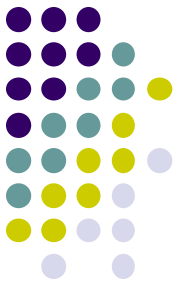
```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="net.simplifiedcoding.androidserviceexample">
    <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:supportsRtl="true"
      android:theme="@style/AppTheme">
      <activity android:name=".MainActivity">
         <intent-filter>
            <action android:name="android.intent.action.MAIN"  />

            <category android:name="android.intent.category.LAUNCHER"  />
         </intent-filter>
      </activity>
      <!-- defining the service class here -->
      <service android:name=".MyService" />
   </application>
</manifest>
```
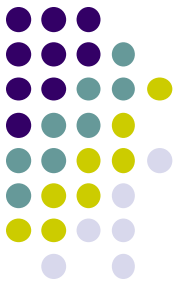
Reference

- **Services | Android Developers**
  - *https://developer.android.com/guide/components/services.html*

- **Implementing an Android Started Service in Android Studio**
  - *http://www.techotopia.com/index.php/Implementing_an_Android_Started_Service_in_Android_Studio*

- **Android - Services**
  - *https://www.tutorialspoint.com/android/android_services.htm*