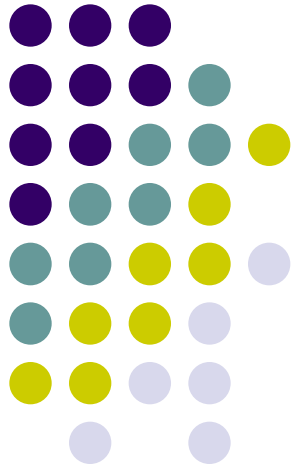
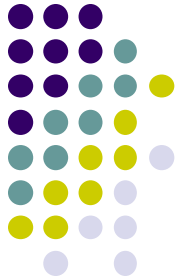
Activity and Multimedia with Databases

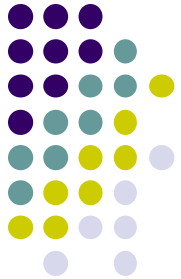


Prof. Prasad Koyande
Vidyalankar Polytechnic

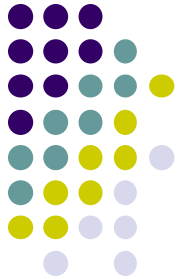
Introduction to Activity in Android



- The activity represents a single screen or interfaces that allows the user to interact with an application.
- Depending on the application on the application types, applications have single to multiple activities.
- Activities could be independent of one another, thereby enabling a different application to use those activities



- Each activity is independent of the others.
- Each one is implemented as a subclass of the Activity base class.
- Moving from one activity to another is proficient by having the current activity which start the next one

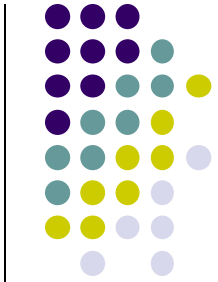


- There are two methods to implement activity:
  - **onCreate(Bundle)** is a method in which we initialize our activity. Here we usually call setCotentView(int)with a layout resource defining the UI, and using findViewById(int) to retrieve the widgets in that UI which we need to interact with programmatically.
  - **onPause()** is another method in which we deal with the user leaving the activity. Any changes made by the user should at this point be committed

INTENT

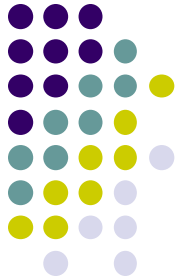
- An Intent is a messaging object that you can use to request an action from an app component.
- An Intent is basically an intention to do an action.
- It is a way to communicate between Android components to request an action from a component, by different components.
- It's like a message that Android listens for and then react accordingly by identifying and invoking the app's appropriate component (like an Activity, Service, Content Provider, etc.).

Uses of Intent in Android



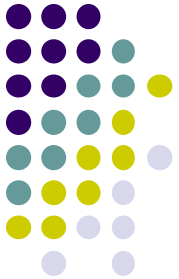
- **To Start an Activity**
  - We can start a new instance of an Activity by passing an Intent to startActivity().
  - The Intent describes the activity to start and carries any necessary data along.
- **To Start a Service**
  - We can start a service to perform a one-time operation (such as downloading a file) by passing an Intent to startService().
  - The Intent describes which service to start and carries any necessary data.

To Deliver a Broadcast



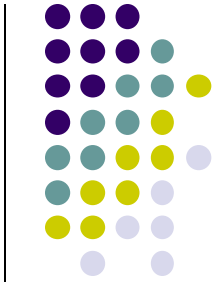
- **To Deliver a Broadcast**
  - The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging
  - We can deliver a broadcast to other apps by passing an Intent to sendBroadcast() or sendOrderedBroadcast().

Types of Intents



- There are two types of Intents
  - Explicit Intents
  - Implicit Intents

Explicit Intents

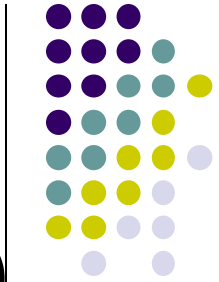


- When we explicitly define which Android component should be opened on some user action, then you use explicit intents.
- We generally use an explicit intent to start a new component in our own app, because we know which exact activity or service you want to start.
- For example, you can start a new activity in response to a user action or start a service to download a file in the background.

Create an Explicit Intent

- We need to make an Intent object. The constructor of the Explicit Intent's object needs two parameters as follows:
  - **Context c:** This represents the object of the Activity from where you are calling the intent.
  - **Java file name:** This represents the name of the java file of the Activity you want to open.
- Call **startActivity()** method and pass the intent's object as the parameter.
- To pass some information or data to the new Activity we are calling, you can do this by calling putExtra() method before the startActivity() method.
- This method accepts key-value pair as its parameter.

```
i.putExtra(“key1”,”I am value1”);
i.putExtra(“key2”,”I am value2”);
startActivity(i);
```



- To receive the data in the new Activity and use it accordingly, you need to call the getIntent() method and then getStringExtra() method in the java class of the Activity you want to open through explicit intent.

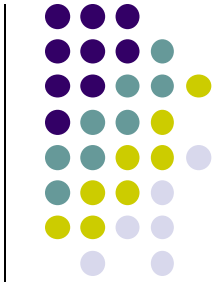
  String a=getIntent().getStringExtra(“key1”);

- Example

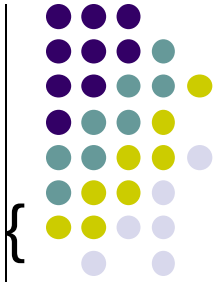
```
Intent intent = new Intent(getApplicationContext(), Second.class);
intent.putExtra("message", str);
startActivity(intent);
```

In Second Activity

```
Intent intent = getIntent();
String str = intent.getStringExtra("message");
```



```
import android.content.Intent;
    public class FirstActivity extends AppCompatActivity {
     @Override
     protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_first);
     }
     public void callSecondActivity(View view){
     Intent i = new Intent(getApplicationContext(),  SecondActivity.class);
     i.putExtra("Value1", “MAD");
     i.putExtra("Value2", “Prof. Prasad Koyande");
     startActivity(i);
    }
}
```

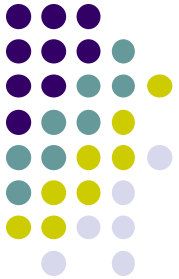


```
import android.content.Intent;
public class SecondActivity extends AppCompatActivity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_second);
 Bundle extras = getIntent().getExtras();
 String value1 = extras.getString("Value1");
 String value2 = extras.getString("Value2");
 Toast.makeText(getApplicationContext(),"Values are:\n
First value: "+value1+ "\n Second Value: "+value2,
Toast.LENGTH_LONG).show();
 }
}
```

Implicit Intents

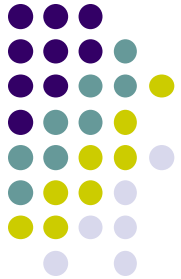
- When we just have to tell what action we want to perform without worrying which component will perform it.
- Implicit intents do not name a specific component to perform a particular action, but instead it declares a general action to be performed, which allows any component, even from another app to handle it.
- For example, if we want to show a specific location of the user on a map, we can use an implicit intent to pass the coordinates through the intent and then any other app, which is capable of showing the coordinates on a map will accept that intent.

Create an Implicit Intent



- We need to make an Intent object.
- The constructor of the Implicit Intent's object needs a type of action we want to perform.
- An action is a string that specifies the generic action to be performed.
  - **ACTION_VIEW:** This action is used when you have some information that an activity can show to the user, such as a photo to view in a Gallery app, or an address to view in a Map app.
  - **ACTION_SEND:** This action is used when you have some data that the user can share through another app, such as an Email app or some Social Networking app.

```
Intent i = new Intent(Intent.ACTION VIEW);
```

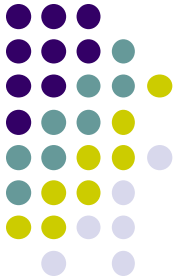


- We need to provide some data for the action to be performed.
- Data is typically expressed as a URI (Uniform Resource Identifier) which provides data to the other app so that any other app which is capable of handling the URI data can perform the desired action.

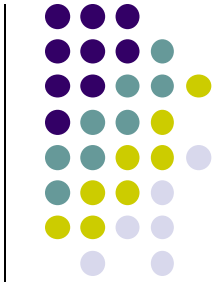
```
i.setData(Uri.parse(http://www.google.co.in));
```

- Call startActivity() method in the end with the intent object as the parameter.

```
import android.content.Intent;
import android.net.Uri;
public class MainActivity extends AppCompatActivity {
 Button button;
 EditText editText;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 button = findViewById(R.id.button);
 editText = findViewById(R.id.editText);
 button.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View view) {
 String url=editText.getText().toString();
 Intent intent=new Intent(Intent.ACTION_VIEW, Uri.parse(url));
 startActivity(intent);
 }
 });
 }}
```

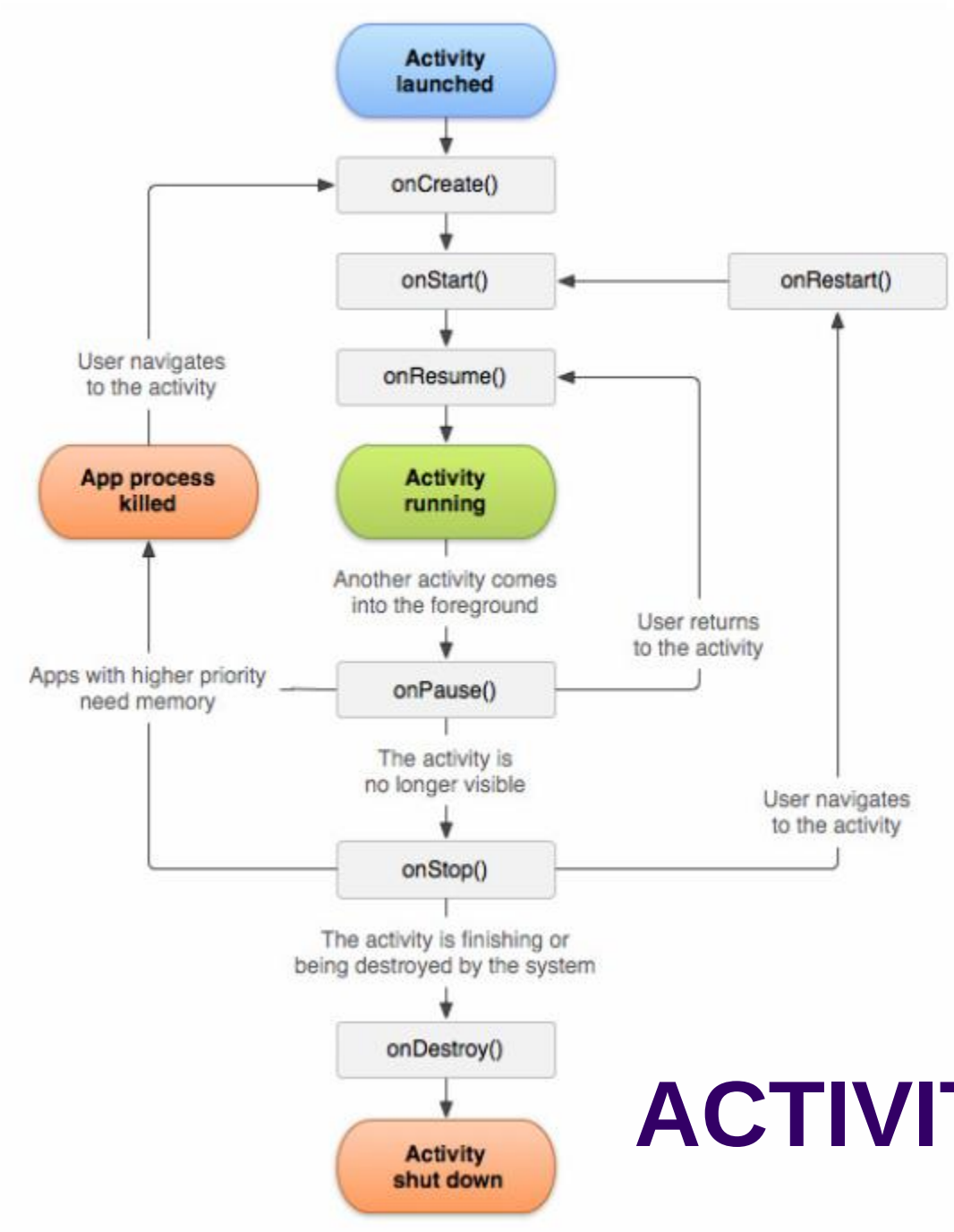


Intent Filter



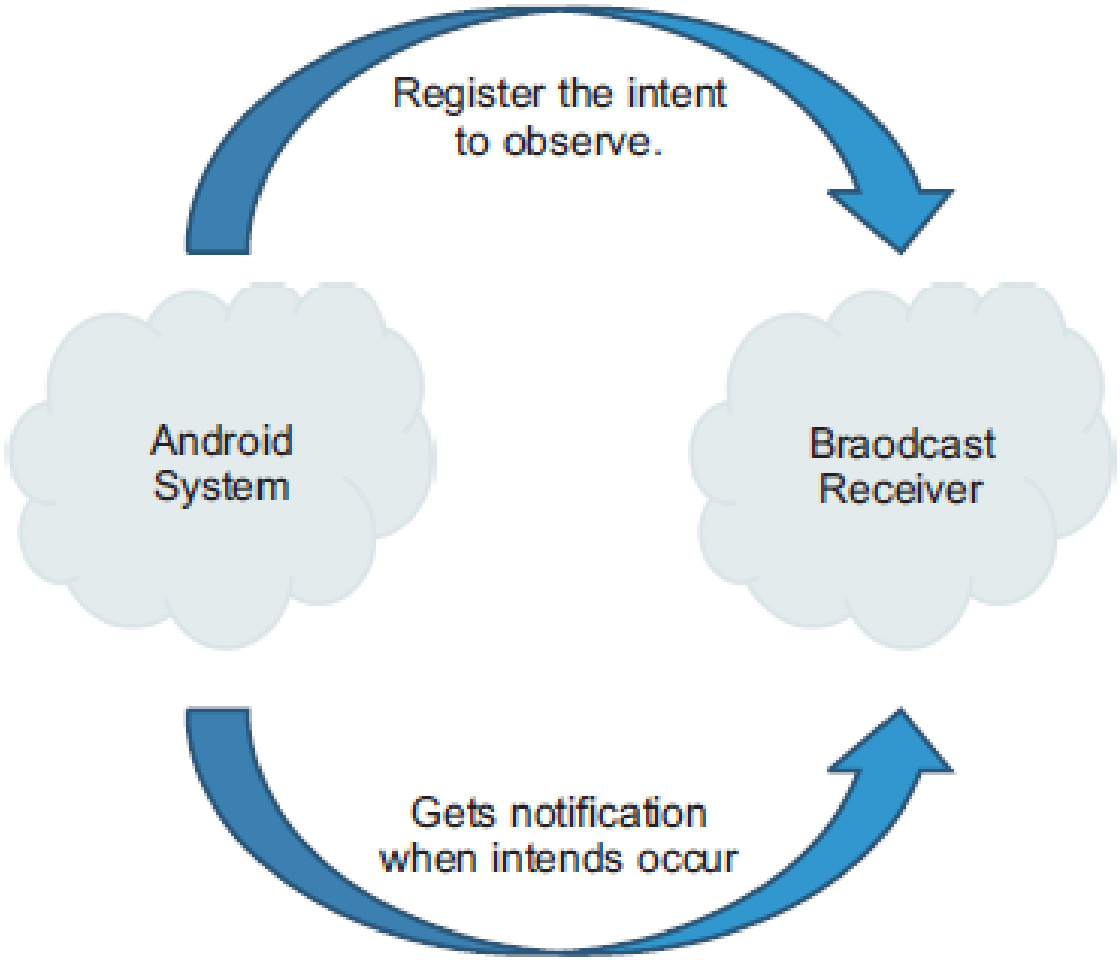
- Intent_filters is a very powerful way to connect different applications together hence allowing better user experience.
- They take care of intent resolution to match activities, services and broadcast receiver.
- Intent filter are declared in the AndroidManifest.xml. an intent filters is an instance of the IntentFilter class.

```
<intent-filter android:icon=”drawable resource”
 android:label=”string resource”
 android:priority=”integer”>
. . .
</intent-filter>
```

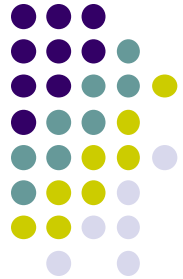


ACTIVITY LIFE CYCLE

Broadcast Life Cycle

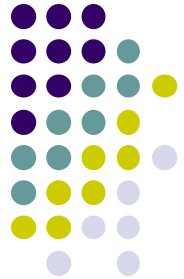


CONTENT PROVIDER



- A Content Provider component supplies data from one application to others on request.
- Such requests are handled by the methods of the ContentResolver class.
- A content provider can use different ways to store its data and the data can be stored in files, in a database or even over a network.
- Content Providers support the four basic operations, normally called CRUD-operations.
- With content providers those objects simply represent data as most often a record of a database, but they could also be a photo on our SD-card or a video on the web.

Content URIs

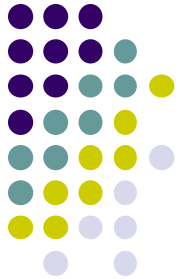


- Whenever we want to access data from a content provider we have to specify a URI. URIs for content providers look like this:

  content://authority/optionalPath/optionalId

- Two types of URI
  - directory-based URIs
  - id-based URIs

Available standard Content Providers



1. CalendarContract SDK 14: Manages the calendars on the user's device.

2. Browser SDK 1: Manages our web-searches, bookmarks and browsing-history.

3. CallLog SDK 1: Keeps track of our call history.

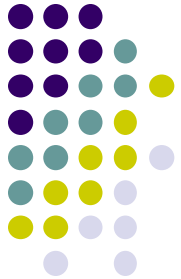
4. MediaStore SDK 1: The content provider responsible for all our media files like music, video and

pictures.

5. Settings SDK 1: Manages all global settings of our device.

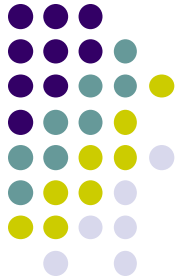
6. UserDictionary SDK 3: Keeps track of words we add to the default dictionary.

Create Content Provider



1. Create a Content Provider class that extends the **ContentProviderbaseclass**
2. Define our content provider URI address which will be used to access the content.
3. Create our own database to keep the content
4. Content Provider queries to perform different database specific operations
5. Register our Content Provider in our activity file using tag

List of methods we need to override in Content Provider class



1**. onCreate():** This method is called when the provider is started.

2. **query():** This method receives a request from a client. The result is returned as a Cursor object.

3. **insert():** This method inserts a new record into the content provider.

4. **delete():** This method deletes an existing record from the content provider.

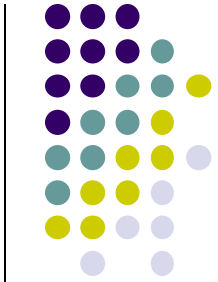
5. **update():** This method updates an existing record from the content provider.

6. **getType():** This method returns the MIME type of the data at the given URI.

FRAGMENTS

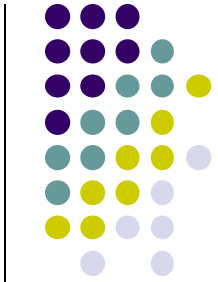
- A fragment is a self-contained, modular section of an application's User Interface (UI) and corresponding behavior that can be embedded within an activity.
- Fragments can be assembled to create an activity during the application design phase, and added to or removed from an activity during application runtime.
- Fragments may only be used as part of an activity and cannot be instantiated as standalone application elements.
- Fragments are stored in the form of XML layout files and may be added to an activity either by placing appropriate elements in the activity's layout file, or directly through code

Creating a Fragment



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:background="@color/red" >

 <TextView
 android:id="@+id/textView1"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_centerHorizontal="true"
 android:layout_centerVertical="true"
 android:text="@string/fragone_label_text"
 android:textAppearance="?android:attr/textAppearanceLarge" />
</RelativeLayout>
```



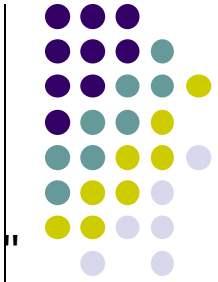
```
package com.example.myfragmentdemo;

import android.support.v4.app.Fragment;

public class FragmentOne extends Fragment {

 @Override
 public View onCreateView(LayoutInflater inflater,
 ViewGroup container,
 Bundle savedInstanceState) {
 // Inflate the layout for this fragment
 return inflater.inflate(R.layout.fragment_one_layout,
 container, false);
 }
}
```

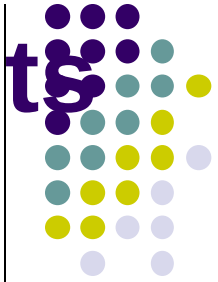
Adding a Fragment to an Activity using the Layout XML File



```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 tools:context=".FragmentDemoActivity" >

 <fragment
 android:id="@+id/fragment_one"
 android:name="com.example.myfragmentdemo.myfragmentdemo.FragmentOne"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:layout_alignParentLeft="true"
 android:layout_centerVertical="true"
 tools:layout="@layout/fragment_one_layout" />
</RelativeLayout>
```

Adding and Managing Fragments in Code



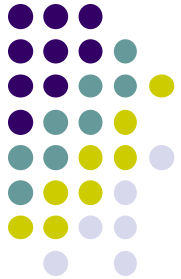
```
FragmentOne firstFragment = new FragmentOne();
firstFragment.setArguments(getIntent().getExtras());

FragmentManager fragManager =
getSupportFragmentManager();
FragmentTransaction transaction =
fragManager.beginTransaction();

transaction.add(R.id.LinearLayout1, firstFragment);
transaction.commit();
```

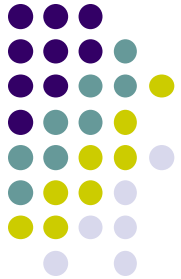
1. Create an instance of the fragment's class.
2. Pass any additional intent arguments through to the class instance.
3. Obtain a reference to the fragment manager instance.
4. Call the beginTransaction() method on the fragment manager instance.
5. Call the add() method of the fragment transaction instance, passing through as arguments the resource ID of the view that is to contain the fragment and the fragment class instance.
6. Call the commit() method of the fragment transaction.

SERVICE



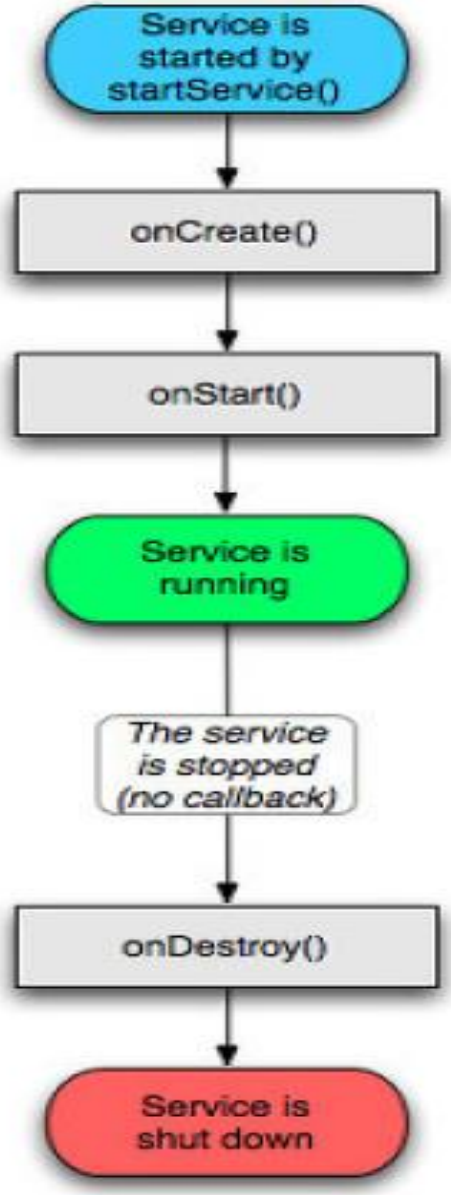
- A service is an application component which runs without direct interaction with the user in the background.
- Services are used for repetitive and potentially long running operations, i.e., Internet downloads, checking for new data, data processing, updating content providers.
- Services run with a higher priority than inactive or invisible activities
- Services can also be configured to be restarted if they get terminated by the Android system once sufficient system resources are available again.
- Services are started with two methods namely, Context.startService(), Context.bindService().

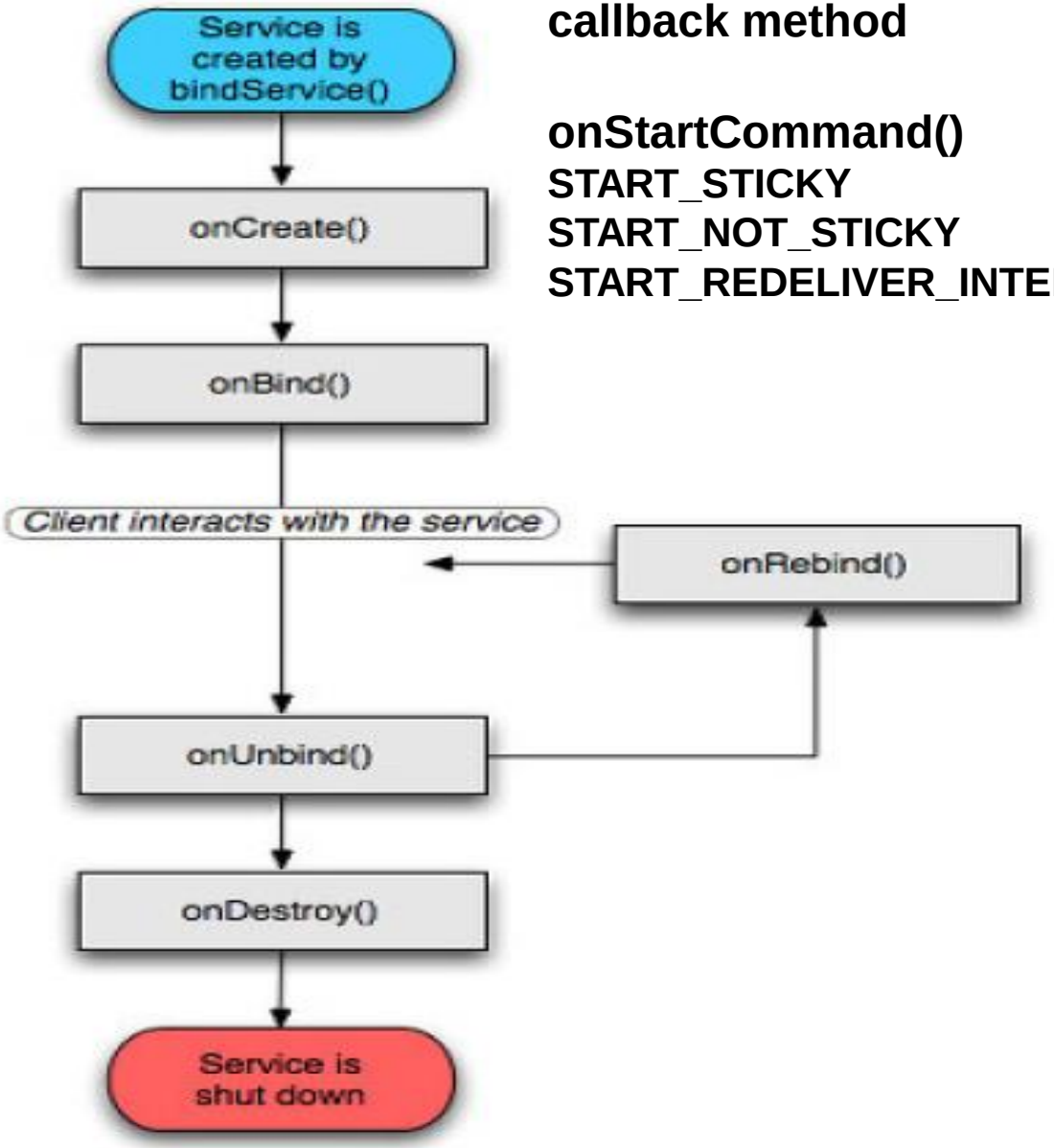
SERVICE (cont…)



- Services are the faceless components of Android as they have their individual interfaces.
- Services are communicate with other Android components and use the Android's notification framework to notify the users.
- Services are job-specific and they are unaffected by the switching activity.
- They will continue to run in the background even if you switch to the interface of a different application.

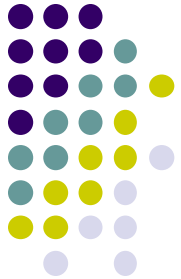
Service Life Cycle



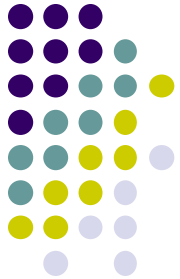


callback method

onStartCommand()
START_STICKY
START_NOT_STICKY
START_REDELIVER_INTENT



```
public class MyService extends Service {
//Declaring the handler
private Handler handler;
//Declaring our implementation of Runnable
private Runner runner;
/*
Regardless of whether you want our service to be binded
or not you should always implement onBind. You should return null if you
dont want it to bind
*/
@Nullable
@Override
public IBinder onBind(Intent intent) {
return null;
}
/*
```

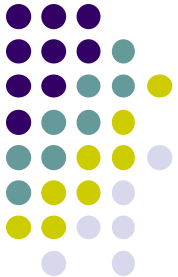


```
Initialization of Handler and Runner
*/
public void onCreate() {
super.onCreate();
Toast.makeText(this, "Service Started",
Toast.LENGTH_LONG).show();
handler = newHandler();
runner = newRunner();
}
// Starting the Runnable with handler
public int onStartCommand(Intent intent,
int id, int startID) {
handler.post(runner);
return START_STICKY;
}
```

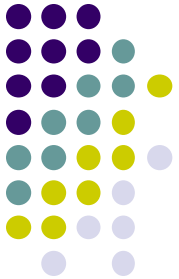
```
@Override
public void onDestroy() {
super.onDestroy();
handler.removeCallbacks(runner);
Toast.makeText(this, "Service Destroyed",
Toast.LENGTH_LONG).show();
}

public class Runner implements Runnable
 {
@Override
public void run() {
Log.d("AndroidClarified", "Running");
handler.postDelayed(this, 1000 * 5);
}
}
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.androidclarified.serviceapp">
<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">
<activityandroid:name=".MainActivity">
<intent-filter>
<actionandroid:name="android.intent.action.MAIN"/>
<categoryandroid:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
<serviceandroid:name=".MyService"/>
</application>
</manifest>
```




9:54
ServiceApp
START SERVICE
STOP SERVICE



```
public class MainActivity extends AppCompatActivity implements
View.OnClickListener{ private Button startButton, stopButton; @Override
protectedvoidonCreate(Bundle savedInstanceState){
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
startButton = (Button)findViewById(R.id.start_button);
stopButton = (Button)findViewById(R.id.stop_button);
startButton.setOnClickListener(this);
stopButton.setOnClickListener(this);
}
@Override
public void onClick(View v){
Intent intent = newIntent(this, MyService.class);
switch(v.getId()){
case R.id.start_button:
startService(intent);
break;
case R.id.stop_button:
stopService(intent);
break;
}}}
```

MULTIMEDIA FRAMEWORK

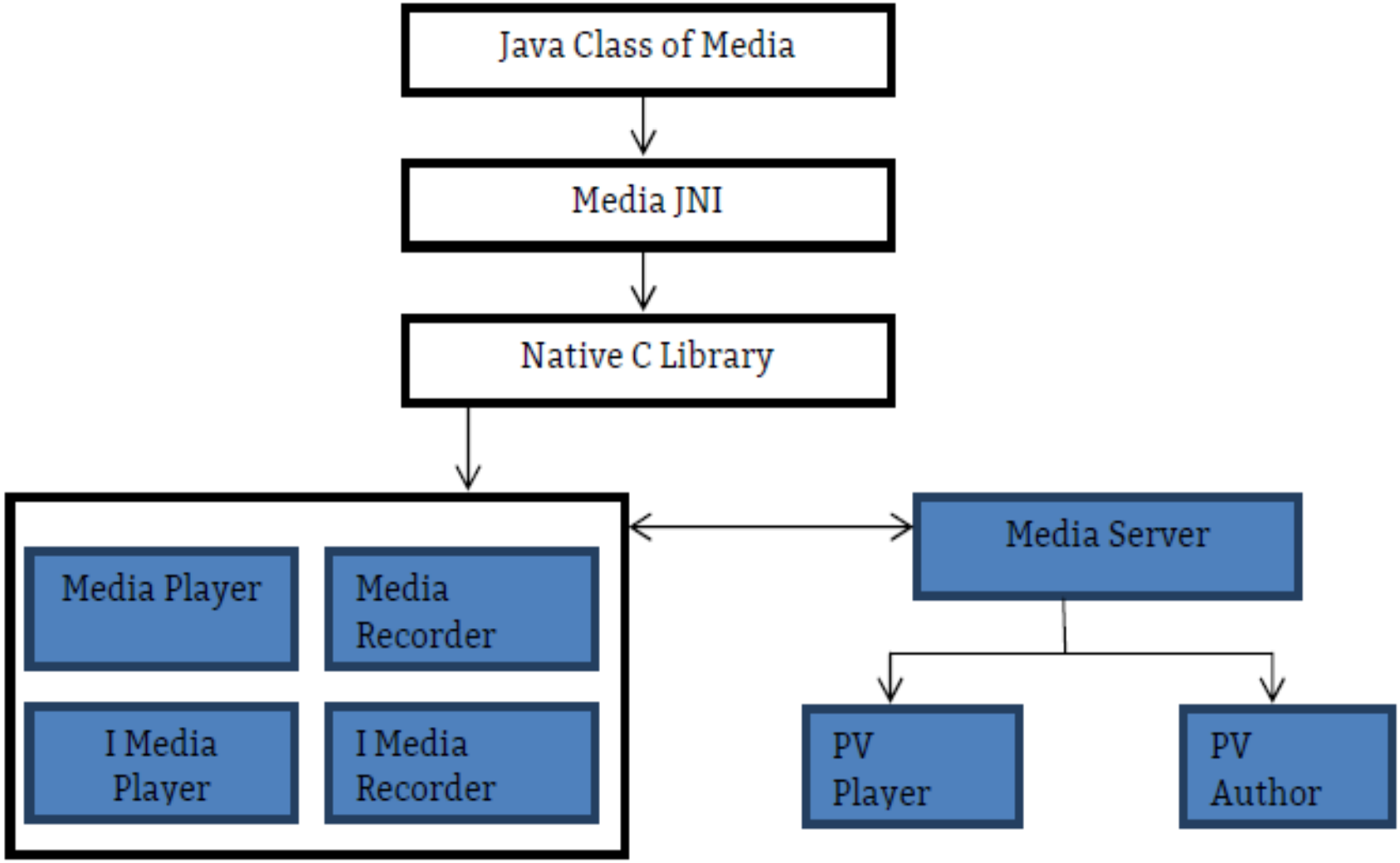


Fig. 5.6: Android Multimedia Framework Architecture

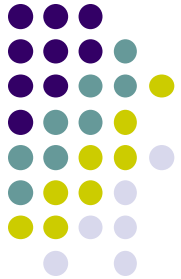
PLAY AUDIO AND VIDEO

- The following classes are used to play sound and video in the Android framework:
  - **MediaPlayer:** This class is the primary API for playing sound and video.
  - **AudioManager:** This class manages audio sources and audio output on a device.
- **Manifest Declarations:**
  - **Internet Permission:** If you are using MediaPlayer to stream network-based content

    <uses-permissionandroid:name="android.permission.INTERNET"/>
  - **Wake Lock Permission :**If our player application needs to keep the screen from dimming or the processor from sleeping

    <uses-permissionandroid:name="android.permission.WAKE_LOCK"/>

Basic MediaPlayer Tasks



1. Load a media file for playback. This is done with the methods setDataSource(), prepare(), and prepareAsync().

2. Start playback / Play audio. This is handled by start().

3. Pause playback (once playback has started). This is handled by pause().

4. Stop playback and reset the MediaPlayer, so that you can load another media file into it. This is handled by reset().

5. Find the length of a song (in ms). This is handled by getDuration().

6. Find what part of the song is playing. This is handled by getCurrentPosition().

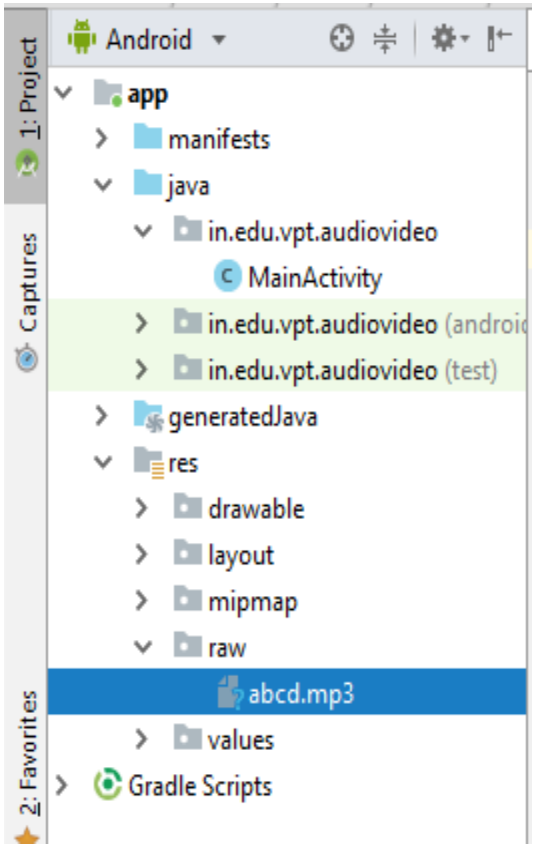
7. Jump to a specific time position (in ms) in the song and play from there. This is handled by seekTo(position).

8. Check to see if audio is being played back right now. This is handled by isPlaying().

9. Find out when a song is done playing. This is handled by attaching a MediaPlayer.OnCompletionListener. Our code will get an onCompletion() callback from the listener.

10. Deallocate resources used by the player. This is handled by release(), which releases all the resources attached to the player. After being released the player is no longer usable.

Audio / Video Examples



Audio Example

Design File

Java File

Video Example

Design File

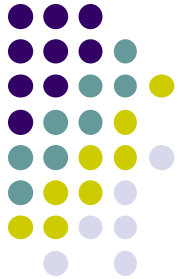
Java File

TEXT TO SPEECH

- Text to speech (TTS) makes an android device read the text and convert it to audio out via the speaker.
- Android TTS supports multiple languages.
- Effectively used in
  - Mobile APPs dedicated to visually impaired people
  - Educational app for kids
  - Pronunciation learning app
- TextToSpeech enhances interaction between the user and the mobile application
- Android TTS was available from version 1.6
- It has features to control speed, pitch of speech as well as type of language.

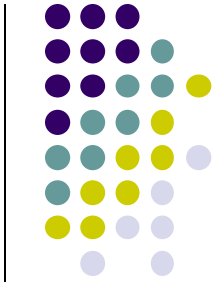
TEXT TO SPEECH EXAMPLE

SENSOR



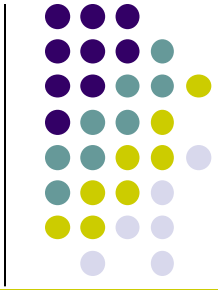
- Android devices have built-in sensors that measure motion, orientation, and various environmental conditions.
- Sensors are capable of providing raw data with high precision and accuracy
- Useful to monitor three-dimensional device movement or positioning, or changes in the ambient environment near a device.

Categories of sensors



- **Motion Sensors:** Measure acceleration forces and rotational forces along three axes. **(accelerometers, gravity sensors, gyroscopes, and rotational vector sensors)**
- **Environmental Sensors:** Measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. **(barometers, photometers, and thermometers)**
- **Position Sensors:** Measure the physical position of a device. **(Orientation sensors and magnetometers)**

Sensor types supported by the Android Platform

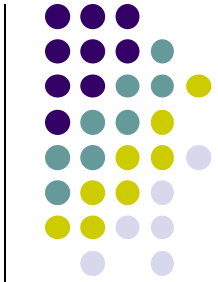


Sensor	Type	Description	Common Use
TYPE_ACCELEROMETER	Hardware	Measures the acceleration force in m/s2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity	Motion detection (shake, tilt, etc.)
TYPE_AMBIENT_TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius (°C)	Monitoring air temperatures
TYPE_GRAVITY	Software or Hardware	Measures the force of gravity in m/s2 that is applied to a device on all three physical axes (x, y, z)	Motion detection (shake, tilt, etc.).
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z)	Rotation detection (spin, turn, etc.)
TYPE_LIGHT	Hardware	Measures the ambient light level (illumination) in lx	Controlling screen brightness
TYPE_LINEAR_ACCELERATION	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in µT	Creating a compass
TYPE_ORIENTATION	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z).	Determining device position

Sensor types supported by the Android Platform (cont..)

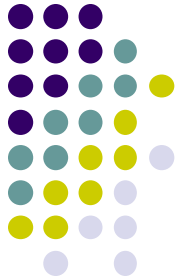
Sensor	Type	Description	Common Use
TYPE_PRESSURE	Hardware	Measures the ambient air pressure in hPa or mbar	Monitoring air pressure changes
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
TYPE_RELATIVE_HUMIDITY	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dew point, absolute, and relative humidity
TYPE_ROTATION_VECTOR	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector	Motion detection and rotation detection
TYPE_TEMPERATURE	Hardware	Measures the temperature of the device in degrees Celsius (°C). This sensor implementation varies across devices and this sensor was replaced with the TYPE_AMBIENT_TEMPERATURE sensor in API Level 14	Monitoring temperature

Sensor Framework



- You can access these sensors and acquire raw sensor data by using the Android sensor framework
- The sensor framework includes the following classes and interfaces:
  - SensorManager
  - Sensor
  - SensorEvent
  - SensorEventListener

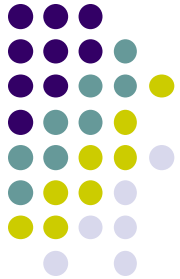
Sensor Examples



Program to display the list of sensors supported by the mobile device

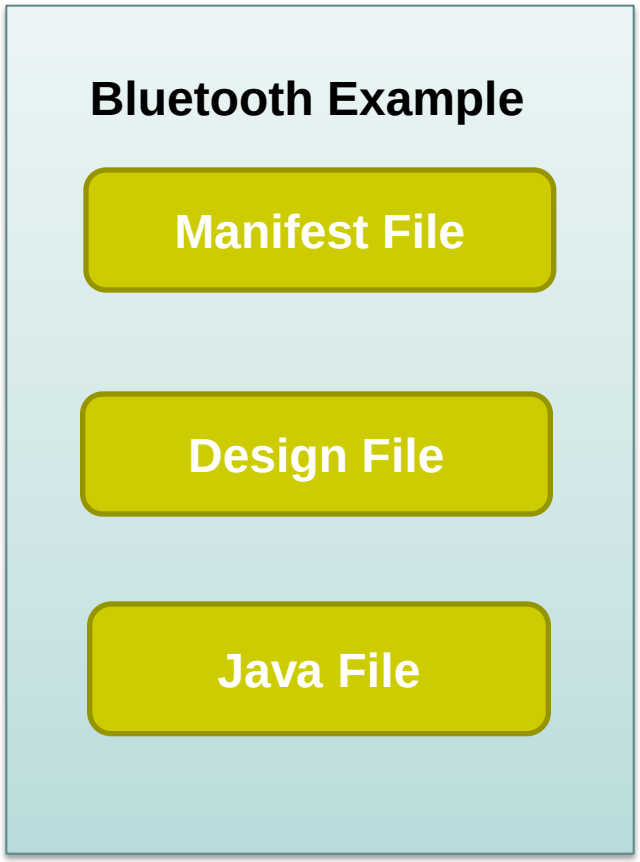
Program to change the background color when device is shuffled

BLUETOOTH



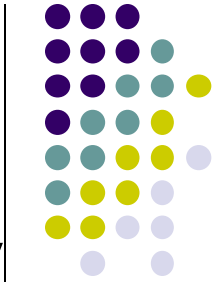
- Bluetooth is a communication network protocol, which allow a devices to connect wirelessly to exchange the data with other Bluetooth devices
- Bluetooth API's provides following functionalities
  1. Scan for the available Bluetooth devices within the range
  2. Use local Bluetooth adapter for paired Bluetooth devices
  3. Connect to other devices through service discovery
  4. Transfer data to and from other devices
  5. Manage multiple connections

Bluetooth Example



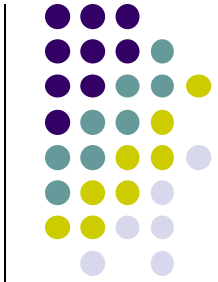
Program to turn on, get visible, list devices and turn off Bluetooth with help of following GUI

ANDROID ASYNCTASK



- Android AsyncTask is an abstract category provided by android which provides the freedom to perform significant tasks within the background and keep the UI thread lightweight thus making the application more responsive.
- Android AsyncTask to perform the significant tasks in background on a dedicated thread and passing the results back to the UI thread.
- Use of AsyncTask in android application keeps the UI thread responsive at all times.

Methods of AsyncTask class



- doInBackground() :
- onPreExecute() :
- onPostExecute() :
- onProgressUpdate() :

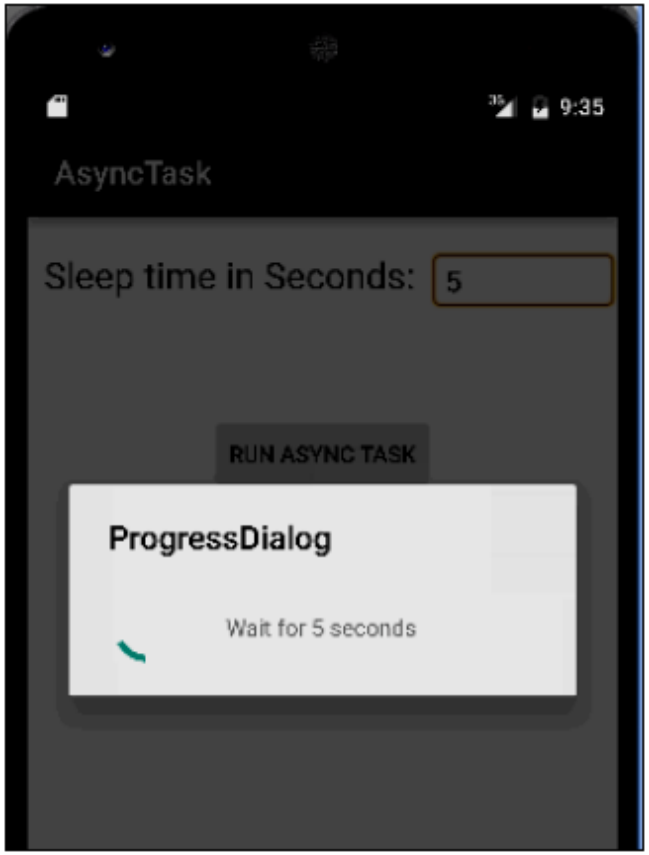
Generic Types

1. Params : The type of the parameters sent to the task upon execution

2. Progress : The type of the progress units published during the background computation

3. Result : The type of the result of the background computation

ASYNCTASK Example

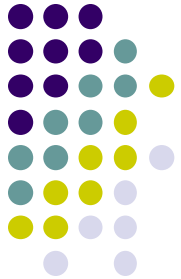


AsyncTask Example

Design File

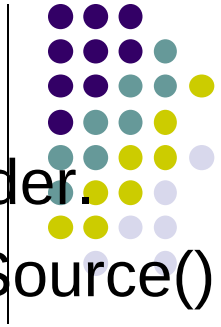
Java File

AUDIO CAPTURE



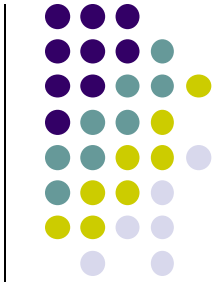
- The Android multimedia system framework includes support for capturing and encoding variety of common audio formats, so we will simply integrate audio into our applications.
- We can record audio using the MediaRecorder APIs if supported by the device hardware.

Performing Audio Capture:



1. Create a new instance of android.media.MediaRecorder.
2. Set the audio source using MediaRecorder.setAudioSource(). We will probably want to use MediaRecorder.AudioSource.MIC.
3. Set output file format using MediaRecorder.setOutputFormat().
4. Set output file name using MediaRecorder.setOutputFile().
5. Set the audio encoder using MediaRecorder.setAudioEncoder().
6. Call MediaRecorder.prepare() on the MediaRecorder instance.
7. To start audio capture, call MediaRecorder.start().
8. To stop audio capture, call MediaRecorder.stop().
9. When we are done with the MediaRecorder instance, call MediaRecorder.release() on it. Calling MediaRecorder.release() is always recommended to free the resource immediately.

CAMERA



There are two ways to integrate the camera module

1. Using In-built Camera App
2. Writing Custom Camera App

Basics for Camera:

1. Camera
2. SurfaceView
3. MediaRecorder
4. Intent

1. Camera Permission: Our application must request permission to use a device camera.

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

2. Camera Features: Our application must also declare use of camera features, for example:

<uses-feature android:name="android.hardware.camera" />

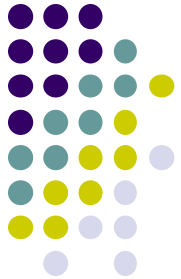
<uses-feature android:name="android.hardware.camera" android:required="false" />

3. Storage Permission: If our application saves images or videos to the device's external storage (SD Card), we must also specify this in the manifest.

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

4. Audio Recording Permission: For recording audio with video capture, our application must request the audio capture permission.

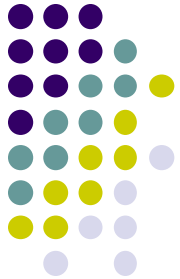
<uses-permission android:name="android.permission.RECORD_AUDIO" />



5. Location Permission: If our application tags images with GPS location information, you mustrequest location permission:

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

Using Existing Camera Apps:



1. Compose a Camera Intent

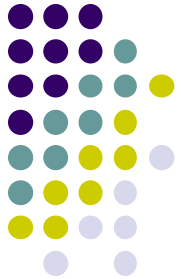
(i) MediaStore.ACTION_IMAGE_CAPTURE

(ii) MediaStore.ACTION_VIDEO_CAPTURE

2. Start the Camera Intent

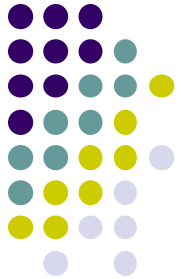
3. Receive the Intent Result

ANIMATION



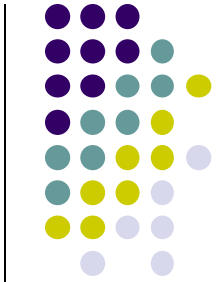
- Animation is the process of creating motion and shape change.
- Animations are useful when the screen changes state, i.e when the content loads or new actions become available

Android Defines Three Types of Animations



1. View Animation:
   - It define the properties of our Views that should be animated using a technique called Tween Animation.It take the following parameters i.e. size, time duration , rotation angle, start value , end value, and perform the required animation on that object.
   - You can execute the animation by specifying transformations on our View.
   - Android View animation can make animation on any View objects, such as ImageView, TextView or Button objects.
   - View animation can only animate simple properties like position, size, rotation, and the alpha property that allows you animate the transparency of a View.

2. Property Animation:



- Property animations are highly customizable, we can specify the duration, the number of repeats, the type of interpolation, and the frame rate of the animation.
- The Property Animation system is always preferred for more complex animations.

Sr. No.	Property	Description
1.	alpha	Fade in or out
2.	rotation, rotationX, rotationY	Spin or flip
3.	scaleX ,scaleY	Grow or shrink
4.	x,y,z	Position
5.	translationX, translationY, translationZ (API 21+)	Offset from Position

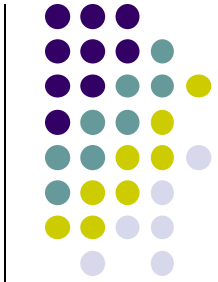
3. Drawable Animation:

- This animation allows the user to load drawable resources and display them one frame after another.
- This method of animation is useful when user wants to animate things that are easier to represent with Drawable resources.
- Android has provided us a class called Animation
- The Animation class has many methods given below:

1. start(): This method will start the animation.
2. setDuration(long duration): This method sets the duration of an animation.
3. getDuration(): This method gets the duration.
4. end(): This method ends the animation.
5. cancel(): This method cancels the animation.

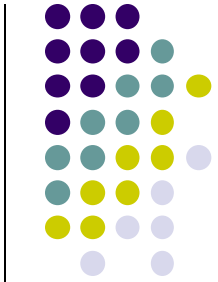
Animation Example

SQLITE DATABASE



- SQLite is an embedded, relational database management system (RDBMS).
- SQLite is referred to as embedded because it is provided in the form of a library that is linked into applications.
- As such, there is no standalone database server running in the background.
- All database operations are handled internally within the application through calls to functions contained in the SQLite library.
- SQLite is written in the C programming language and as such, the Android SDK provides a Java based “wrapper” around the underlying database interface.

Why SQLite/Necessity of SQLite



1. Serverless: SQLite is serverless which does not need a detach server process or system to operate.

2. Zero Configurations: SQLite does not require any setup or administration.

3. Cross-platform: a complete SQLite database is stored in a single cross-platform disk file.

4. Less Memory: SQLite is very small and light weight, less than 400 KiB completely configured or less than 250 KiB with optional featues omitted.

5. Self-Contained: SQLite has no external dependencies.

6. Transaction: SQLite transactions are supported ACID properties to allow safe access from multiple processes or threads.

7. Languages and operating system: SQLite supports most of the query language featueres found in SQL92(SQL2) standared.

Database Example