SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A++' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

19ITE310 - MOBILE APPLICATION DEVELOPMENT

III YEAR - VI SEM

UNIT 4 - SPRUCING UP MOBILE APPS SPRUCING UP MOBILE APPS

UNIT – 4
SPRUCING UP MOBILE APPS SPRUCING UP MOBILE APPS

Graphics and animation – Custom views – canvas - animation APIs - multimedia – audio/video - playback and record - location awareness, and native hardware access (sensors such as accelerometer and gyroscope)

<u>Lab Experiments:</u>

1. Develop a native application that uses GPS location information.
2. To develop a Simple Android Application that draws basic Graphical Primitives on the screen.(Line,
Square, Rectangle and Circle)

Graphics and animation

- Animation in android is possible from many ways. In this chapter we will discuss one easy and widely used way of making animation called tweened animation.
- Tween Animation
- Tween Animation takes some parameters such as start value , end value, size , time duration , rotation angle e.t.c and perform the required animation on that object. It can be applied to any type of object. So in order to use this , android has provided us a class called Animation.
- In order to perform animation in android , we are going to call a static function loadAnimation() of the class AnimationUtils. We are going to receive the result in an instance of Animation Object. Its syntax is as follows −
- Animation animation = AnimationUtils.loadAnimation(getApplicationContext(), R.anim.myanimation);
- This animation class has many useful functions which are listed below package com.example.sairamkrishna.myapplication;−

Graphics and animation

Sr.No	Method & Description
1	start() -This method starts the animation.
2	setDuration(long duration)- This method sets the duration of an animation.
3	getDuration() -This method gets the duration which is set by above method
4	end()- This method ends the animation.
5	cancel() -This method cancels the animation.

Graphics and animation

- import android.app.Activity;
- import android.os.Bundle;
- import android.view.View;
- import android.view.animation.Animation;
- import android.view.animation.AnimationUtils;
- import android.widget.ImageView;
- import android.widget.Toast;

Graphics and animation

```
•  public class MainActivity extends Activity {
•     @Override
•     protected void onCreate(Bundle savedInstanceState) {
•        super.onCreate(savedInstanceState);
•        setContentView(R.layout.activity_main);
•     }
•
•     public void clockwise(View view){
•        ImageView image = (ImageView)findViewById(R.id.imageView);
•        Animation animation = AnimationUtils.loadAnimation(getApplicationContext(),
•           R.anim.myanimation);
•        image.startAnimation(animation);
•     }
```

Graphics and animation

- public void fade(View view){
-     ImageView image = (ImageView)findViewById(R.id.imageView);
-     Animation animation1 =
-        AnimationUtils.loadAnimation(getApplicationContext(),
-           R.anim.fade);
-     image.startAnimation(animation1);
- }
-     public void blink(View view){
-     ImageView image = (ImageView)findViewById(R.id.imageView);
-     Animation animation1 =
-        AnimationUtils.loadAnimation(getApplicationContext(),
-           R.anim.blink);
-     image.startAnimation(animation1)

Graphics and animation

```
•     public void move(View view){
•       ImageView image = (ImageView)findViewById(R.id.imageView);
•       Animation animation1 =
•          AnimationUtils.loadAnimation(getApplicationContext(), R.anim.move);
•       image.startAnimation(animation1);
•     }
•

•     public void slide(View view){
•       ImageView image = (ImageView)findViewById(R.id.imageView);
•       Animation animation1 =
•          AnimationUtils.loadAnimation(getApplicationContext(), R.anim.slide);
•       image.startAnimation(animation1);
•     }
```

Custom views, canvas

- Android offers a great list of pre-built widgets like Button, TextView, EditText, ListView, CheckBox, RadioButton, Gallery, Spinner, AutoCompleteTextView etc. which you can use directly in your Android application development, but there may be a situation when you are not satisfied with existing functionality of any of the available widgets. Android provides you with means of creating your own custom components which you can customized to suit your needs.

Creating a Simple Custom Component

Step	Description
1	You will use Android studio IDE to create an Android application and name it as myapplication under a package com.example.tutorialspoint7.myapplication as explained in the Hello World Example chapter.
2	Create an XML res/values/attrs.xml file to define new attributes along with their data type.
3	Create src/mainactivity.java file and add the code to define your custom component
4	Modify res/layout/activity_main.xml file and add the code to create Colour compound view instance along with few default attributes and new attributes.
5	Run the application to launch Android emulator and verify the result of the changes done in the application.

Custom views, canvas

- <?xml version="1.0" encoding="utf-8"?>
- <resources>
-    <declare-styleable name="TimeView">
-       <declare-styleable name="TimeView">
-          <attr name="title" format="string" />
-          <attr name="setColor" format="boolean"/>
-       </declare-styleable>
-    </declare-styleable>
- </resources>
- Change the layout file used by the activity to the following.
- <?xml version="1.0" encoding="utf-8"?>
- <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
-    xmlns:tools="http://schemas.android.com/tools"
-    xmlns:custom="http://schemas.android.com/apk/res-auto"
-    android:layout_width="match_parent"
-    android:layout_height="match_parent"
-    tools:context=".MainActivity" >
- <com.example.tutorialspoint7.myapplication.TimeView
-       android:id="@+id/timeView"
-       android:layout_width="match_parent"
-       android:layout_height="wrap_content"
-       android:textColor="#fff"
-       android:textSize="40sp"
-       custom:title="my time view"
-       custom:setColor="true" />
-
-    <TextView
-       android:layout_width="match_parent"
-       android:layout_height="wrap_content"
-       android:id="@+id/simple"
-       android:layout_below="@id/timeView"
-       android:layout_marginTop="10dp"

Animation APIs

- The API allows to define for arbitrary object properties a start and end value and apply a time-based change to this attribute. This API can be applied on any Java object not only on Views.
- **Animator and AnimatorListener**
- The superclass of the animation API is the Animator class. Typically the ObjectAnimator class is used to modify the attributes of an object.
- The ViewPropertyAnimator class introduced in Android 3.1 provides a simpler access to typical animations which are performed on Views.
- The animate() method on a View will return the ViewPropertyAnimator object. This object allows to perform simultaneous animations. It has a fluent API and allows to set the duration of the animation.
- The target of ViewPropertyAnimator is to provide a very simple API for typical animations.
- he following code shows an example of the usage of this method.

Animation APIs

- // Using hardware layer
- myView.animate().translationX(400).withLayer();

- For performance optimization you can also let ViewPropertyAnimator use a hardware layout.
- // Using hardware layer
- myView.animate().translationX(400).withLayer();

- You can also directly define a Runnable to be executed at the start and the end of the animation.

Animation APIs

- // StartAction
- myView.animate().translationX(100).withStartAction(new Runnable(){
-   public void run(){
-     viewer.setTranslationX(100-myView.getWidth());
-     // Do something
-   }
- });

- // EndAction
- myView.animate().alpha(0).withStartAction(new Runnable(){
-   public void run(){
-     // Remove the view from the layout called parent
-     parent.removeView(myView);
-   }
- });

Layout animations

- The LayoutTransition class allows to set animations on a layout container and a changes on the Views of this container will be animated.
- package com.example.android.layoutanimation;

- import android.animation.LayoutTransition;
- import android.app.Activity;
- import android.os.Bundle;
- import android.view.Menu;
- import android.view.View;
- import android.view.ViewGroup;
- import android.widget.Button;

- public class MainActivity extends Activity {

-   private ViewGroup viewGroup;

Layout animations

- public void onCreate(Bundle savedInstanceState) {
-     super.onCreate(savedInstanceState);
-     setContentView(R.layout.activity_main);
-     LayoutTransition l = new LayoutTransition();
-     l.enableTransitionType(LayoutTransition.CHANGING);
-     viewGroup = (ViewGroup) findViewById(R.id.container);
-     viewGroup.setLayoutTransition(l);
- }
- public void onClick(View view) {
-     viewGroup.addView(new Button(this));
- }
- @Override
- public boolean onCreateOptionsMenu(Menu menu) {
-     getMenuInflater().inflate(R.menu.activity_main, menu);
-     return true;
- }

Animations for Activity transition PAGE TOP

- Animations can be applied to Views but it is also possible to apply them on the transition between activities.
- The ActivityOptions class allows to define default or customer animations.
- public void onClick(View view) {
-   Intent intent = new Intent(this, SecondActivity.class);
-   ActivityOptions options = ActivityOptions.makeScaleUpAnimation(view, 0,
-       0, view.getWidth(), view.getHeight());
-   startActivity(intent, options.toBundle());
- }

Multimedia

- Android Multimedia
- Android multimedia api deals in playing and controlling the audio and video.
- Playing Audio in android Example
- Playing Video in android Example

- Play and control the audio files in android by the help of MediaPlayer class.
- MediaPlayer class
- The android.media.MediaPlayer class is used to control the audio or video files.
- Methods of MediaPlayer class

Multimedia

- There are many methods of MediaPlayer class. Some of them are as follows:

Method	Description
public void setDataSource(String path)	sets the data source (file path or http url) to use.
public void prepare()	prepares the player for playback synchronously.
public void start()	it starts or resumes the playback.
public void stop()	it stops the playback.
public void pause()	it pauses the playback.
public boolean isPlaying()	checks if media player is playing.
public void seekTo(int millis)	seeks to specified time in miliseconds.
public void setLooping(boolean looping)	sets the player for looping or non-looping.
public boolean isLooping()	checks if the player is looping or non-looping.
public void selectTrack(int index)	it selects a track for the specified index.
public int getCurrentPosition()	returns the current playback position.
public int getDuration()	returns duration of the file.
public void setVolume(float leftVolume,float rightVolume)	sets the volume on this player.

Multimedia

```
•
•    package com.example.audiomediaplayer1;
•
•    import android.media.MediaPlayer;
•    import android.net.Uri;
•    import android.os.Bundle;
•    import android.app.Activity;
•    import android.view.Menu;
•    import android.widget.MediaController;
•    import android.widget.VideoView;
•
•    public class MainActivity extends Activity {
•
•        @Override
•        protected void onCreate(Bundle savedInstanceState) {
•            super.onCreate(savedInstanceState);
•            setContentView(R.layout.activity_main);
•
•            MediaPlayer mp=new MediaPlayer();
•            try{
•                mp.setDataSource("/sdcard/Music/maine.mp3");
```

Multimedia

```
• //Write your location here
•              mp.prepare();
•              mp.start();
•
•          }catch(Exception e){e.printStackTrace();}
•
•      }
•
•      @Override
•      public boolean onCreateOptionsMenu(Menu menu) {
•          // Inflate the menu; this adds items to the action bar if it is present.
•          getMenuInflater().inflate(R.menu.activity_main, menu);
•          return true;
•      }
• }
```

Audio/video and images

- MediaController and VideoView classes, we can play the video files in android
- MediaController class
- The android.widget.MediaController is a view that contains media controls like play/pause, previous, next, fast-forward, rewind etc.
- VideoView class
- The android.widget.VideoView class provides methods to play and control the video player. The commonly used methods of VideoView class are as follows:

Audio/video and images

Method	Description
public void setMediaController(MediaController controller)	sets the media controller to the video view.
public void setVideoURI (Uri uri)	sets the URI of the video file.
public void start()	starts the video view.
public void stopPlayback()	stops the playback.
public void pause()	pauses the playback.
public void suspend()	suspends the playback.
public void resume()	resumes the playback.
public void seekTo(int millis)	seeks to specified time in miliseconds.

Playback and record

- Android provides many ways to control playback of audio/video files and streams. One of this way is through a class called MediaPlayer.
- Android is providing MediaPlayer class to access built-in mediaplayer services like playing audio,video e.t.c. In order to use MediaPlayer, we have to call a static Method create() of this class. This method returns an instance of MediaPlayer class. Its syntax is as follows
- MediaPlayer mediaPlayer = MediaPlayer.create(this, R.raw.song);

Playback and record

- The second parameter is the name of the song that you want to play. You have to make a new folder under your project with name raw and place the music file into it.
- Once you have created the Mediaplayer object you can call some methods to start or stop the music. These methods are listed below.
- mediaPlayer.start();
- mediaPlayer.pause();
- On call to start() method, the music will start playing from the beginning. If this method is called again after the pause() method, the music would start playing from where it is left and not from the beginning.

Playback and record

- In order to start music from the beginning, you have to call reset() method. Its syntax is given below.
- mediaPlayer.reset();
- Apart from the start and pause method, there are other methods provided by this class for better dealing with audio/video files. These methods are listed below

Playback and record

1	isPlaying() This method just returns true/false indicating the song is playing or not
2	seekTo(position) This method takes an integer, and move song to that particular second
3	getCurrentDuration() This method returns the current position of song in milliseconds
4	getDuration() This method returns the total time duration of song in milliseconds
5	reset() This method resets the media player

Playback and record

6	release() This method releases any resource attached with MediaPlayer object
7	setVolume(float leftVolume, float rightVolume) This method sets the up down volume for this player
8	setDataSource(FileDescriptor fd) This method sets the data source of audio/video file
9	selectTrack(int index) This method takes an integer, and select the track from the list on that particular index
10	getTrackInfo() This method returns an array of track information

MediaRecorder

- 1. Requesting permission to record audio
- 2. Creating and running a MediaRecorder
- 3. Using MediaMuxer to record multiple channels
- 4. Adding metadata
- 5. Sample code

MediaRecorder

- The Android multimedia framework includes support for capturing and encoding a variety of common audio and video formats. You can use the MediaRecorder APIs if supported by the device hardware.
- This document shows you how to use MediaRecorder to write an application that captures audio from a device microphone, save the audio, and play it back

Creating and running a MediaRecorder

- Requesting permission to record audio
- To be able to record, app must tell the user that it will access the device's audio input. <uses-permission android:name="android.permission.RECORD_AUDIO" />

Creating and running a MediaRecorder

- Initialize a new instance of MediaRecorder with the following calls:
- Set the audio source using setAudioSource().
- Note: Most of the audio sources (including DEFAULT) apply processing to the audio signal. To record raw audio select UNPROCESSED. Some devices do not support unprocessed input. Call AudioManager.getProperty("PROPERTY_SUPPORT_AUDIO_SOURCE_UNPROCESSED") first to verify it's available. If it is not, try using

LOCATION AWARENESS

- Android location APIs make it easy for you to build location-aware applications, without needing to focus on the details of the underlying location technology.
- This becomes possible with the help of Google Play services, which facilitates adding location awareness to your app with automated location tracking, geofencing, and activity recognition.
- This tutorial shows you how to use Location Services in your APP to get the current location, get periodic location updates, look up addresses etc.

The Location Object

- The Location object represents a geographic location which can consist of a latitude, longitude, time stamp, and other information such as bearing, altitude and velocity. There are following important methods which you can use with Location object to get location specific information −

The Location Object

Sr.No.	Method & Description
1	float distanceTo(Location dest) Returns the approximate distance in meters between this location and the given location.
2	float getAccuracy() Get the estimated accuracy of this location, in meters.
3	double getAltitude() Get the altitude if available, in meters above sea level.
4	float getBearing() Get the bearing, in degrees.
5	double getLatitude() Get the latitude, in degrees.

The Location Object

Sr.No.	Method & Description
6	double getLongitude() Get the longitude, in degrees.
7	float getSpeed() Get the speed if it is available, in meters/second over ground.
8	boolean hasAccuracy() True if this location has an accuracy.
9	boolean hasAltitude() True if this location has an altitude.
10	boolean hasBearing() True if this location has a bearing.

The Location Object

Sr.No.	Method & Description
11	boolean hasSpeed() True if this location has a speed.
12	void reset() Clears the contents of the location.
13	void setAccuracy(float accuracy) Set the estimated accuracy of this location, meters.
14	void setAltitude(double altitude) Set the altitude, in meters above sea level.
15	void setBearing(float bearing) Set the bearing, in degrees.

The Location Object

Sr.No.	Method & Description
16	void setLatitude(double latitude) Set the latitude, in degrees.
17	void setLongitude(double longitude) Set the longitude, in degrees.
18	void setSpeed(float speed) Set the speed, in meters/second over ground.
19	String toString() Returns a string containing a concise, human-readable description of this object.

Get the Current Location

- To get the current location, create a location client which is LocationClientobject, connect it to Location Services using connect() method, and then call its getLastLocation() method. This method returns the most recent location in the form of Location object that contains latitude and longitude coordinates and other information as explained above. To have location based functionality in your activity, you will have to implement two interfaces –
- • GooglePlayServicesClient.ConnectionCallbacks
- • GooglePlayServicesClient.OnConnectionFailedListener
- These interfaces provide following important callback methods, which you need to implement in your activity class –

Get the Current Location

Sr.No.	Callback Methods & Description
1	abstract void onConnected(Bundle connectionHint) This callback method is called when location service is connected to the location client successfully. You will use connect() method to connect to the location client.
2	abstract void onDisconnected() This callback method is called when the client is disconnected. You will use disconnect() method to disconnect from the location client.
3	abstract void onConnectionFailed(ConnectionResult result) This callback method is called when there was an error connecting the client to the service.

Motion sensors

- The Android platform provides several sensors that let you monitor the motion of a device.
- The sensors' possible architectures vary by sensor type:
- • The gravity, linear acceleration, rotation vector, significant motion, step counter, and step detector sensors are either hardware-based or software-based.
- • The accelerometer and gyroscope sensors are always hardware-based.
- Most Android-powered devices have an accelerometer, and many now include a gyroscope. The availability of the software-based sensors is more variable because they often rely on one or more hardware sensors to derive their data. Depending on the device, these software-based sensors can derive their data either from the accelerometer and magnetometer or from the gyroscope.

Motion sensors

- Motion sensors are useful for monitoring device movement, such as tilt, shake, rotation, or swing. The movement is usually a reflection of direct user input (for example, a user steering a car in a game or a user controlling a ball in a game), but it can also be a reflection of the physical environment in which the device is sitting (for example, moving with you while you drive your car). In the first case, you are monitoring motion relative to the device's frame of reference or your application's frame of reference; in the second case you are monitoring motion relative to the world's frame of reference. Motion sensors by themselves are not typically used to monitor device position, but they can be used with other sensors, such as the geomagnetic field sensor, to determine a device's position relative to the world's frame of reference (see Position Sensors for more information).

Motion sensors

- All of the motion sensors return multi-dimensional arrays of sensor values for each SensorEvent. For example, during a single sensor event the accelerometer returns acceleration force data for the three coordinate axes, and the gyroscope returns rate of rotation data for the three coordinate axes. These data values are returned in a float array (values) along with other SensorEvent parameters. Table 1 summarizes the motion sensors that are available on the Android platform.
- The rotation vector sensor and the gravity sensor are the most frequently used sensors for motion detection and monitoring. The rotational vector sensor is particularly versatile and can be used for a wide range of motion-related tasks, such as detecting gestures, monitoring angular change, and monitoring relative orientation changes. For example, the rotational vector sensor is ideal if you

Android Open Source Project sensors

- All of the motion sensors return multi-dimensional arrays of sensor values for each SensorEvent. For example, during a single sensor event the accelerometer returns acceleration force data for the three coordinate axes, and the gyroscope returns rate of rotation data for the three coordinate axes. These data values are returned in a float array (values) along with other SensorEvent parameters. Table 1 summarizes the motion sensors that are available on the Android platform.
- The rotation vector sensor and the gravity sensor are the most frequently used sensors for motion detection and monitoring. The rotational vector sensor is particularly versatile and can be used for a wide range of motion-related tasks, such as detecting gestures, monitoring angular change, and monitoring relative orientation changes. For example, the rotational vector sensor is ideal if you

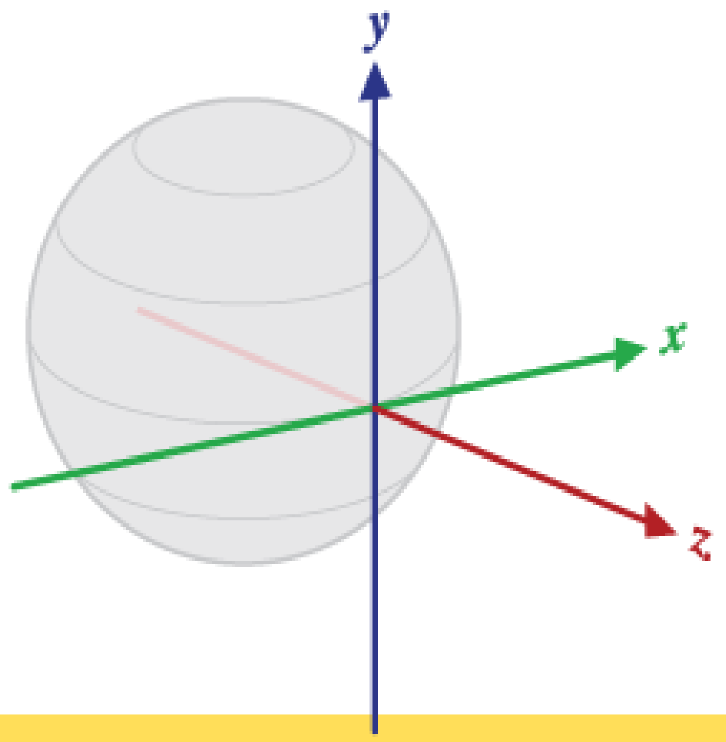
Android Open Source Project sensors

- **Use the gravity sensor**
- The gravity sensor provides a three dimensional vector indicating the direction and magnitude of gravity. Typically, this sensor is used to determine the device's relative orientation in space. The following code shows you how to get an instance of the default gravity sensor:

Use the rotation vector sensor

- The rotation vector represents the orientation of the device as a combination of an angle and an axis, in which the device has rotated through an angle θ around an axis (x, y, or z).
- The following code shows you how to get an instance of the default rotation vector sensor:
- The three elements of the rotation vector are expressed as follows:
-

$$x \cdot \sin(\frac{\theta}{2})$$
$$y \cdot \sin(\frac{\theta}{2})$$
$$z \cdot \sin(\frac{\theta}{2})$$



Use the rotation vector sensor

- Use the step counter sensor
- The step counter sensor provides the number of steps taken by the user since the last reboot while the sensor was activated. The step counter has more latency (up to 10 seconds) but more accuracy than the step detector sensor.

- Use the step detector sensor
- The step detector sensor triggers an event each time the user takes a step. The latency is expected to be below 2 seconds. The following code shows you how to get an instance of the default step detector sensor:

Use the rotation vector sensor

- val sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
- val sensor: Sensor? = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER)

- val sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
- val sensor: Sensor? = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR)