SNS COLLEGE OF TECHNOLOGY

(An Autonomous Institution)

Re-accredited by NAAC with A+ grade, Accredited by NBA(CSE, IT, ECE, EEE & Mechanical)
Approvedy by AICTE, New Delhi, Recognized by UGC, Affiliated to Anna University, Chennai

Department of MCA

GRAPHICS & ANIMATION

Course: **Mobile Application Development**

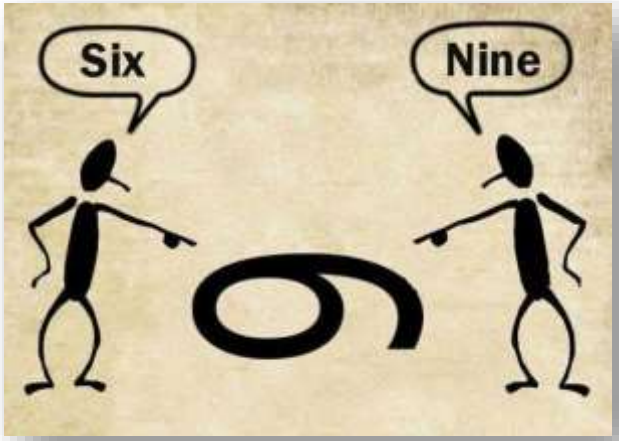
Unit : IV –Sprucing Up Mobile Apps

Class / Semester: II MCA / III Semester

Graphics and Animation

- Graphics and animation are two key ingredients for UI enhancement
- Graphics enhances the visual quality of an app

Deals with
- Screen size
- Screen resolution
- Screen orientation
- Colors
- Typography
- Image formats



- Animation add zing to the app experience by augmenting tiny delights at multiple occasions

Categories of Graphics capabilities

Drawables and Canvas: Components allows building custom 2D graphics using Canvas

Hardware acceleration: GPU to render an app's UI for graphics-intensive apps,

OpenGL- open source 3D framework to render 3D graphics

android.graphics

android.graphics.drawable

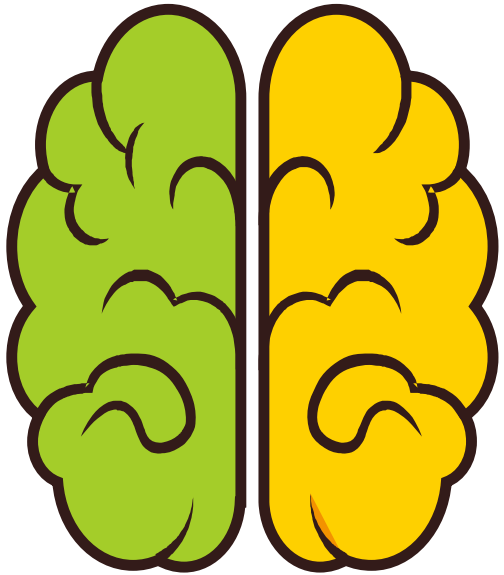
Understanding

Two key concepts that predominantly impact the app UI in android is

The number of pixels per unit area of a screen, usually referred as dots per inch (dpi)

Screen density

- To deal with views across multiple screen densities, it is recommended to use density-independent pixels (dp) as the dimension unit.
- It ensures that views get appropriately scaled up/down based on the screen densities.



Screen size

- Android provides three configuration qualifiers – sw<N>dp, w<N>dp, and h<N>dp – for supporting different screen sizes

Color model (A, R,G,B)

↓

opacity

Drawable Resources

- General abstraction for "something that can be drawn"
- Defined in an XML file in the directory /Resources/drawable
- Not necessary to provide density-specific versions of Drawable Resources
- Android application will load these resources and use the instructions contained in these XML files to create 2D graphics
- Android supports various types of drawables like png, bitmap image
- Shape drawable is used where geometric shapes are required to be drawn In XML file is used to define the attributes of the geometric shape , instead of an image file
- Layer drawable is used when there is a requirement to manage an array of drawables to be drawn in a layered fashion

Categories of Graphics capabilities

- **android.graphics.Canvas** can be used to draw graphics in android
  - It provides methods to draw oval, rectangle, picture, text, line etc
- **android.graphics.Paint** class is used with canvas to draw objects.
  - It holds the information of color and style

Example

MainActivity.java

```
public class MainActivity extends Activity {

    DemoView demoview;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        demoview = new DemoView(this);
        setContentView(demoview);
    }

    private class DemoView extends View{
        public DemoView(Context context){
            super(context);
        }

        @Override protected void onDraw(Canvas canvas) {
            super.onDraw(canvas);

            // custom drawing code here
            Paint paint = new Paint();
            paint.setStyle(Paint.Style.FILL);

        // make the entire canvas white
            paint.setColor(Color.WHITE);
            canvas.drawPaint(paint);

            // draw blue circle with anti aliasing turned off
            paint.setAntiAlias(false);
            paint.setColor(Color.BLUE);
            canvas.drawCircle(20, 20, 15, paint);
    // draw green circle with anti aliasing turned on
            paint.setAntiAlias(true);
            paint.setColor(Color.GREEN);
            canvas.drawCircle(60, 20, 15, paint);

            // draw red rectangle with anti aliasing turned off
            paint.setAntiAlias(false);
            paint.setColor(Color.RED);
            canvas.drawRect(100, 5, 200, 30, paint);

            // draw the rotated text
            canvas.rotate(-45);

            paint.setStyle(Paint.Style.FILL);
            canvas.drawText("Graphics Rotation", 40, 180, paint);

            //undo the rotate
            canvas.restore();
        }
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.

        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}
```

Drawables- Example

shape_draw.xml in the res\drawable folder

```
<shape android:shape="rectangle">
 <gradient android:startColor=" #4A4A4A"  android:endColor="#AAAAAA" android:angle="90"/>
 <padding android:left="7dp" android:top="7dp"  android:right="7dp" android:bottom="7dp"/>
 <corners android:radius="8dp"/>
 </shape>
```

Apply this drawable as a background of an EditText. The android: background attribute of <EditText> in Line 3 is used to refer to the shape drawable

Main_Activity.xml in the res\drawable folder

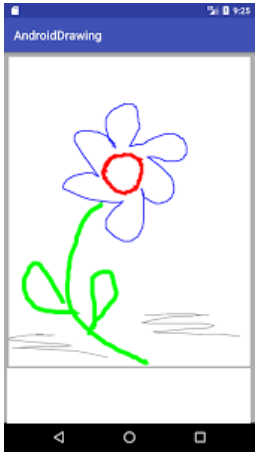
```
<EditText
 android:id="@+id/edit_text01"
 android:background="@drawable/shape_draw"
 android:layout_height="wrap_content"
 android:layout_width="fill_parent"
 android:text="Shape Drawable"
 />
```

Canvas

- Set of 2D-DRAWING APIs allows to provide own custom graphics onto a canvas or to modify existing views to customize their look and feel
- There are two ways to draw 2D graphics
  1. Draw your animation into a View object from your layout.
  2. Draw your animation directly to a Canvas
- Some of the important methods of Canvas Class are as follows
  - drawText()
  - drawRoundRect()
  - drawCircle()
  - drawRect()
  - drawBitmap()
  - drawARGB()

Drawing an animation with a Canvas is better option when your application needs to re-draw itself regularly

Drawing an animation with a View is the best option to draw simple graphics that do not need to change dynamically



Example

```
public class MyView extends View
{
    public MyView(Context context)
    {
       super(context);
       // TODO Auto-generated constructor stub
     }
     @Override
     protected void onDraw(Canvas canvas)
     {
         // TODO Auto-generated method stub
         super.onDraw(canvas);
         int radius;
         radius = 50;
         Paint paint = newPaint();
         paint.setStyle(Paint.Style.FILL);
         paint.setColor(Color.parseColor("#CD5C5C"));
         canvas.drawCircle(150,200, radius, paint);
         canvas.drawRoundRect(newRectF(20,20,100,100), 20, 20,
paint);
         canvas.rotate(-45);
         canvas.drawText("TutorialRide", 40, 180, paint);
         canvas.restore();
      }
}
```

MyView.java

```
Public class MainActivity extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
       super.onCreate(savedInstanceState);
       setContentView(new MyView(this));
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
   // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}
```

MainActivity.java

Animation

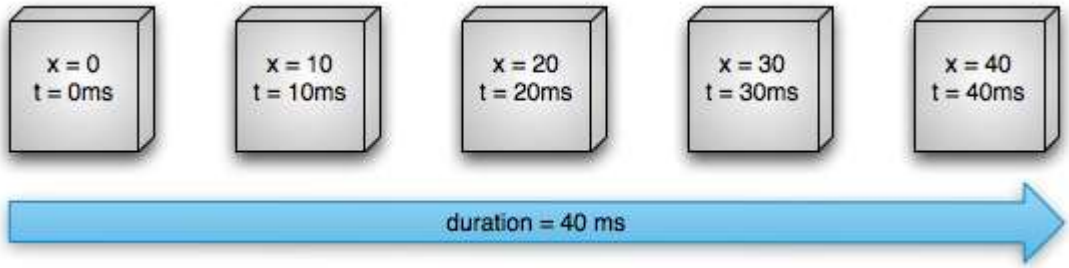
Animation

- Animation is the process of adding a motion effect to any view, image, or text.
- with the help of an animation, you can add motion or can change the shape of a specific view
- Animation in Android is generally used to give your UI a rich look and feel
- The animations are basically of three types as follows:
  - Property Animation
  - View Animation
  - Drawable Animation

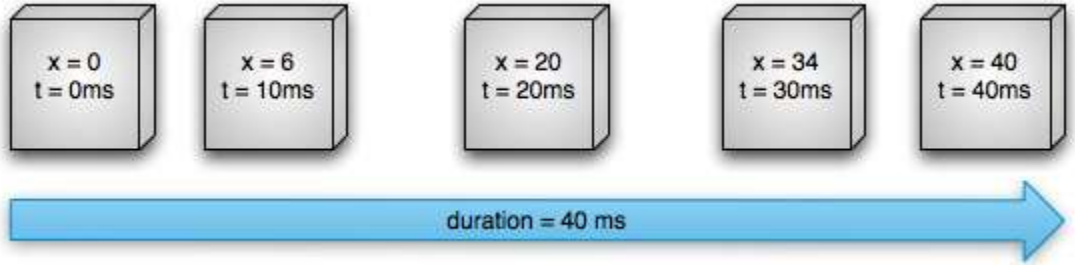
Property Animation

- Property animation can be used to add any animation in the Checkbox, Radio Buttons, and widgets other than any view
- This robust framework which lets you animate any properties of any objects, view or non-view objects
- It defines the following characteristics of an animation: **Duration, time interpolation, Repeat count and behavior, Animator sets, Frame refresh delay**
- android.animation provides classes which handle property animation
- To animate the property, specify
  - Property you want to animate
  - How long you want to animate
  - What values you want animate between

linear animation



Non-linear animation



Animators & Evaluators

- Animator is used to create animations
- Done by a subclass of Animator class
  - ValueAnimator
  - ObjectAnimator
  - AnimatorSet
- Evalutor tells property animation system how to calculate values for animated objects
  - IntEvaluator
  - FloatEvaluator
  - ArgbEvaluator
  - TypeAnimator

View Animation

- Used to add animation to a specific view to perform tweened animation on views
- The **android.view.animation** provides classes which handle view animation
- It is limited to simple transformation such as moving, re-sizing and rotation, but not its background color
- Limitations
  - Apply on view objects only
  - Animate certain aspects of view (scaling and rotation)
  - It affects where view is drawn, not where it actually is

Drawable Animation

- It is used if you want to animate one image over another
- Loads the series of drawable one after another to create an animation. Ex. Splash screen on apps logo animation
- It is implemented using the AnimationDrawable class

```
<animation-list xmlns:android=http://schemas.android.com/apk/res/android” android:oneshot="true">
   <item android:drawable="@drawable/rocket_thrust1" android:duration="200" />
   <item android:drawablec="@drawable/rocket_thrust2" android:duration="200" />
   <item android:drawable="@drawable/rocket_thrust3" android:duration="200" />
</animation-list>
```

rocket_thrust.xml in the res/drawable folder

it can be added as the background image to a View and then called to play

Drawable Animation

```
AnimationDrawable rocketAnimation;

public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.main);

 ImageView rocketImage = (ImageView) findViewById(R.id.rocket_image);
 rocketImage.setBackgroundResource(R.drawable.rocket_thrust);
 rocketAnimation = (AnimationDrawable) rocketImage.getBackground();

 rocketImage.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
      rocketAnimation.start();
    }
 });
}
```

REFERENCES

- Anubhav Pradhan, Anil V Deshpande, “Composing Mobile Apps using Android”, Wiley Edition, 2014
- http://www.dre.vanderbilt.edu/~schmidt/android/android-4.0/out/target/ common/docs/doc-comment-check/guide/topics/graphics/2d-graphics.html
- https://developer.android.com/guide/topics/graphics/drawable-animation