Android Application Development

Session: 16

Material Design

Objectives

- Explain material design
- Describe the material design environment elements
- Describe various material design elements

Material Design

- Combines design principles with advanced tools
- Is a visual language developed by Google
- Shows the relation between visual, motion, and interaction design
- Uses various grid based layouts, animations, lighting effects, and shadows

Introduction to Material and Material Environment

- Material design
  - Has a 3-D environment
  - Contains light, material, and cast shadows
- Material design environment elements include:

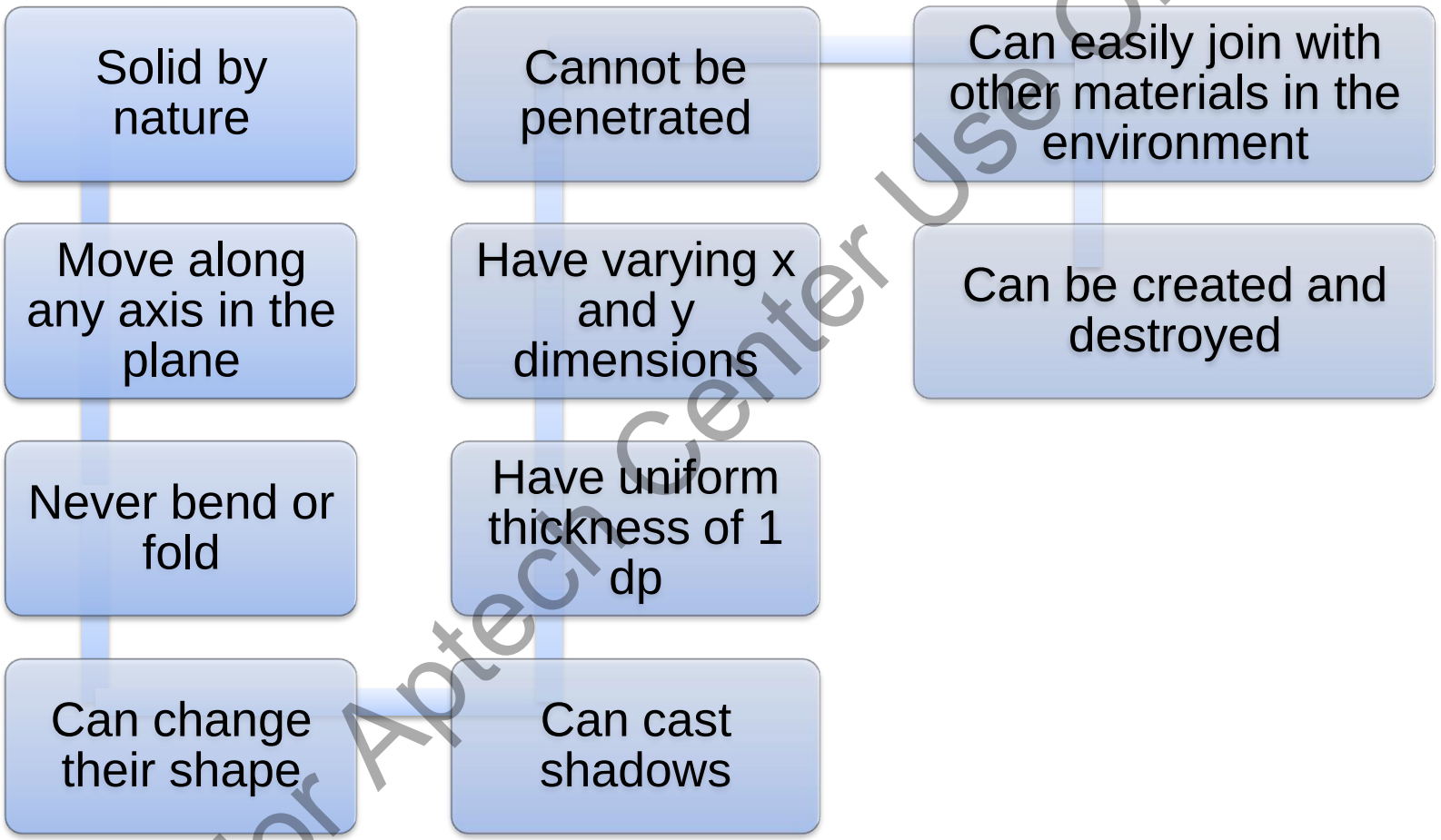
Objects

- Have three dimensions - x,y, and z.
- Material sheets within the objects have a standard thickness of 1dp.

Shadows

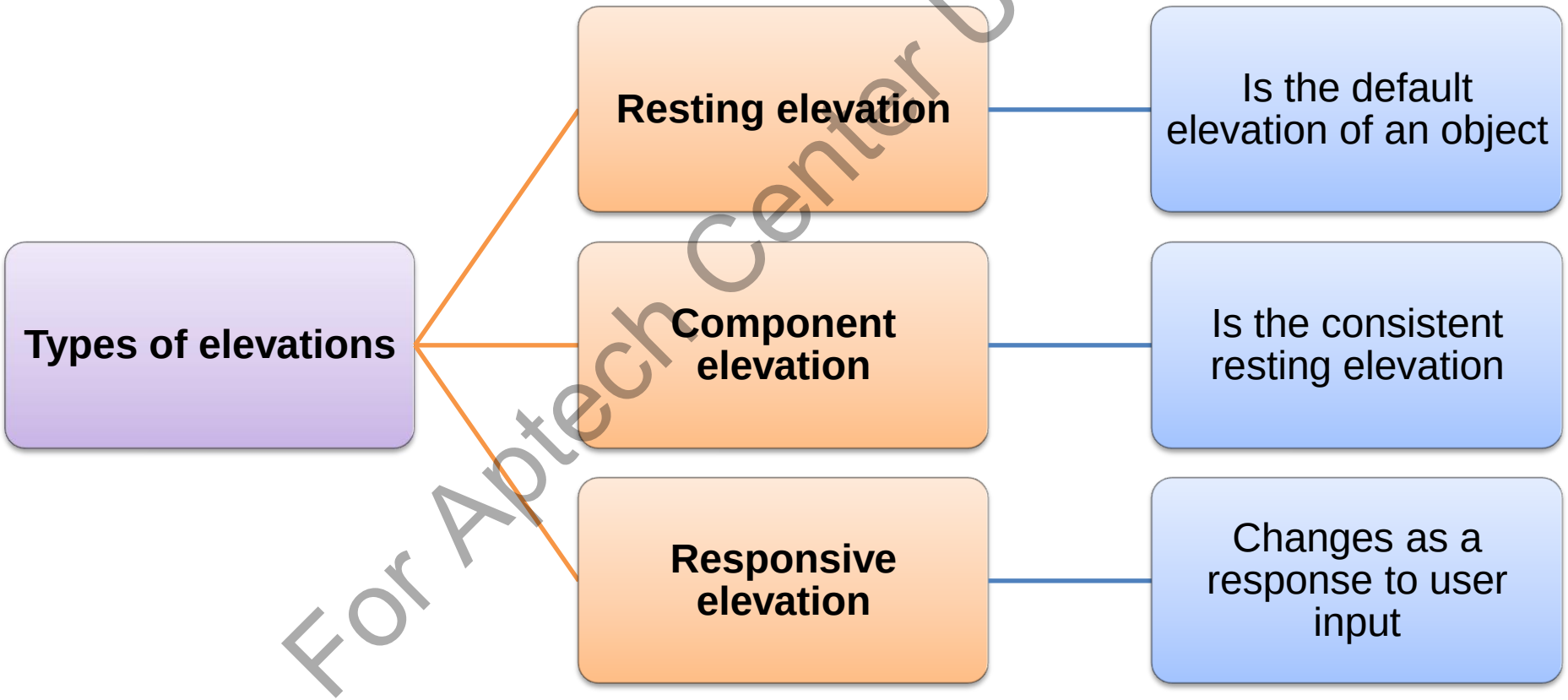
- Virtual lights illuminate the material environment.
- Difference in elevations between overlapping material objects creates shadows.
- The key lights create directional shadows; ambient lights create soft shadows.

Built-in Characteristics of Materials



Elevations

Elevations indicate the distance between surfaces and shadow's depth.



Material Motion

Material Motion → Hierarchical and spatial relationships between objects

Characteristics of material motion

- Describes the movement of materials
- Provides good transition to the user
- Organizes the visual appearance of apps
- Guides the user to have the right focus
- Is natural and responsive

Style 1-4

Following table lists the style elements that can be used in material design to customize the apps:

Element	Content	Description
Color	• Color schemes	• Primary color • Secondary color • Accent color • Text and background colors
	• Themes	• Light theme • Dark theme

Style 2-4

Element	Content	Description	
Icons	• Product icons	• Design approach • Product icon grid • Keyline shapes • DP unit grid • Geometry	• Product icon anatomy • Product icon metrics • Product icon patterns • Human iconography
	• System icons	• Design principles • Grid, proportion, and size • Icon grid • Content area • Dense layouts • Keyline shapes	• Geometry • System icon anatomy • Corners • Strokes • Optical corrections • Clearance area

Style 3-4

Element	Content	Description
Imagery	• Principles	• Personal relevance • Information • Delight • Dynamic and context-relevant
	• Best Practices	• Multiple mediums • No stock photos or clipart • Iconic point of focus in imagery • Immersive story
	• UI Integration	• Resolution • Scale • Text protection • Avatars and thumbnails • Hero Images • Gallery Images

Style 4-4

Element	Content	Description
Typography	• Roboto • Nato	• App bar • Buttons • Subheading • Body 1 • Text contrast ratios
Writing	• Clear • Easy to understand • Concise • Accurate text	• Tone • Capitalization • Punctuation • UI button text • Guidelines

Layout 1-2

Following table lists the layout elements that can be used in material design:

Element	Description
Seam	Two sheets of material sharing a common edge is called a seam.
Step	Two overlapping sheets of material with different depths is called a step.
Floating action button	Circular sheet that is separate from a toolbar is a floating action button.
Pixel density	Number of pixels that can accommodate in an inch is pixel density.
Density-independent pixels(dp)	Uniform display of user interface components on screen having different densities is termed as density-independent pixels.

Layout 2-2

Element	Description
Metrics and keylines	Includes the baseline grids, ratio keylines, sizing by increments, and touch target size for the screen elements.
Structure	Includes the UI regions, toolbars, app bar, system bars, slide nav, and white frames.
Responsive UI	Determines the consistency, adaptability, and scalability of apps on different screen sizes. Includes the breakpoints, grid, surface behaviors, and patterns.
Split screen	Allows the user to view two tasks or activities on the screen simultaneously.

Components 1-2

Following table lists the specific components:

Component	Description
Bottom navigation	Provides quick navigation between top-level views in an app
Bottom sheet	Reveals more screen content by sliding up from the bottom of the screen
Card	Displays information composed of different elements such as photo or text
Chip	Contains photos, text, or icons
Divider	Helps to organize the page into different tiles and thus separate the content

Components 2-2

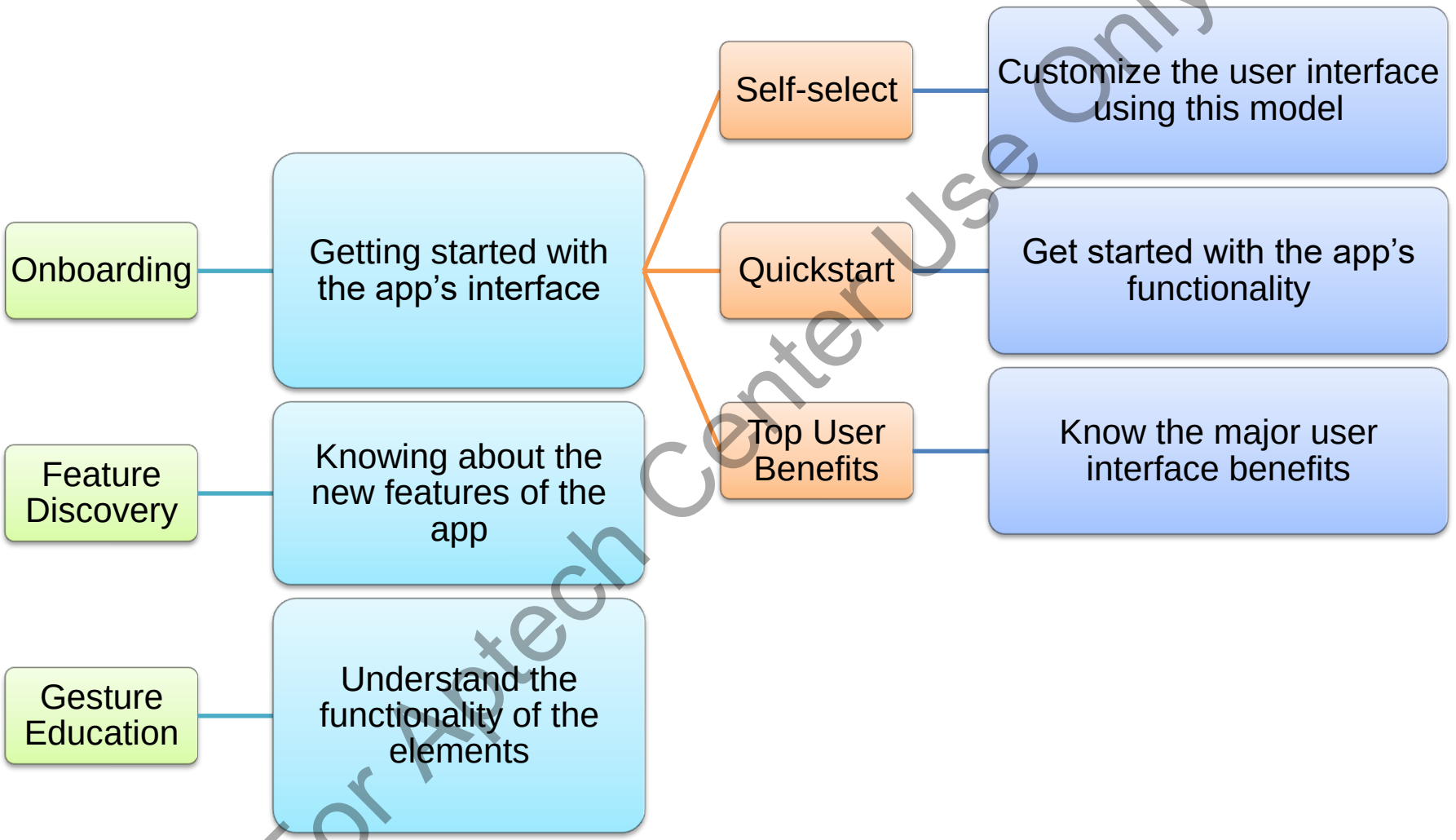
Component	Description
Picker	Helps to pick a single value from a pre-determined set of values
Snackbar	Provides a feedback on the performed operation at the bottom of the screen
Toast	Provides system messages at the bottom of the screen
Stepper	Displays the progress of an operation with the help of logical and numbered steps

Patterns

Following table lists some of the patterns:

Pattern	Description
Empty states	Occurs when an action or operation does not display any content
Fingerprint	Unlocks the device or sign into apps
Gestures	Includes touch mechanics and touch activities
Launch screens	Indicates the initial screen that is launched in an app
Permissions	Explains the need for each permission request
Swipe to refresh	Refreshes the screen content based on user's action

Growth and Communications



Usability

Users with accessibility issues can access the app's user interface

Bidirectional scripts such as Arabic, Hebrew, and Persian can be used

Resources

Following table lists the resources:

Resource	Description
Color palette	Includes the Adobe Photoshop and Adobe Illustrator color swatches
Devices	Include the device metrics
Layout templates	Include the mobile layout, tablet layout, desktop layout, and whiteframes
Roboto	Is designed for mobile and Web usage
Noto	Is the standard typeface that covers the languages not dealt by Roboto
Sticker sheets	Are used by components and include elements that make up layouts
Icons	Indicate system and product icons

Themes

What are Themes?

- The themes provide new styles for the apps.
- They offer uniform or logical styling of apps through surface shades, shadow depth, and ink opacity.

User can set:

- Various themes for the apps
- The color palette and touch animations for the system widgets using material themes
- Themes for the action bars and status bars

Material theme is defined as:

- @android:style/Theme.Material (dark version)
- @android:style/Theme.Material.Light (light version)
- @android:style/Theme.Material.Light.DarkActionBar

Applying Material Themes 1-2

Following code snippet demonstrates how to customize the theme's base colors or define custom colors:

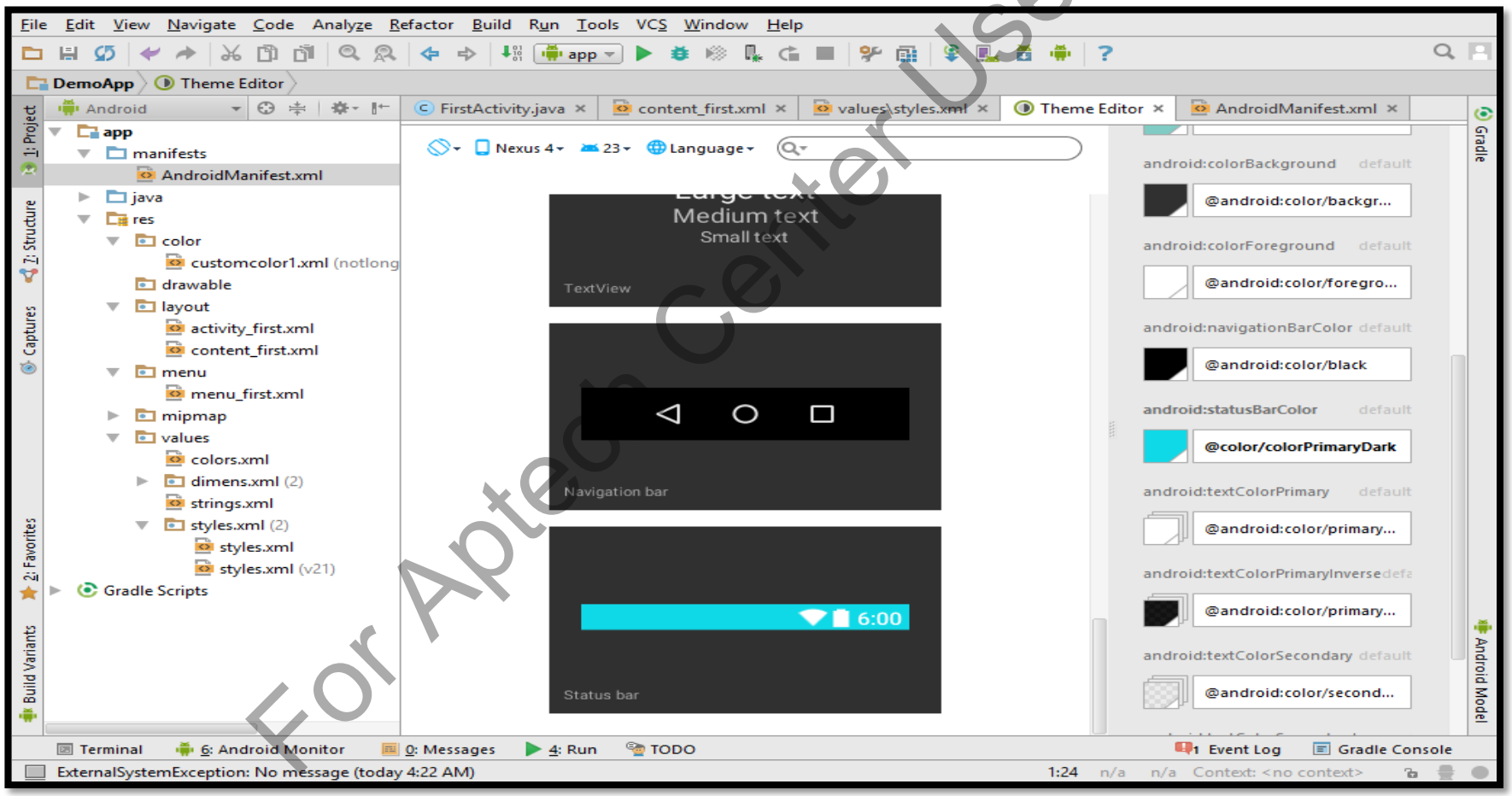
```
<resources>
<!-- Base application theme. -->
    <style name="AppTheme"
parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="AppTheme.NoActionBar">
        <item name="windowActionBar">false</item>
        <item name="windowNoTitle">true</item>
    </style>
```

Applying Material Themes 2-2

```
    <style name="AppTheme.AppBarOverlay"
parent="ThemeOverlay.AppCompat.Dark.ActionBar" />

    <style name="AppTheme.PopupOverlay"
parent="ThemeOverlay.AppCompat.Light" />

    <style name="CustomMaterialTheme"
parent="android:Theme.Material" >
        <item
name="android:colorPrimaryDark">@color/customcolor1</item>
    </style>

</resources>
```

Using Theme Editor

Material themes can also be applied using the Theme Editor: Tools → Android → Theme Editor



Customizing Status Bar

Following code snippet demonstrates how to customize the status bar:

```
<item name="android:statusBarColor">@color/colorPrimaryDark
</item>
```

Lists and Cards

- Lists display multiple line items in a vertical manner as a single continuous element

- Cards are material sheets that function as entry points to display detailed information

RecyclerView widget

- Is a pluggable version of ListView
- Supports different types of layouts
- Improves and enhances the app's performance

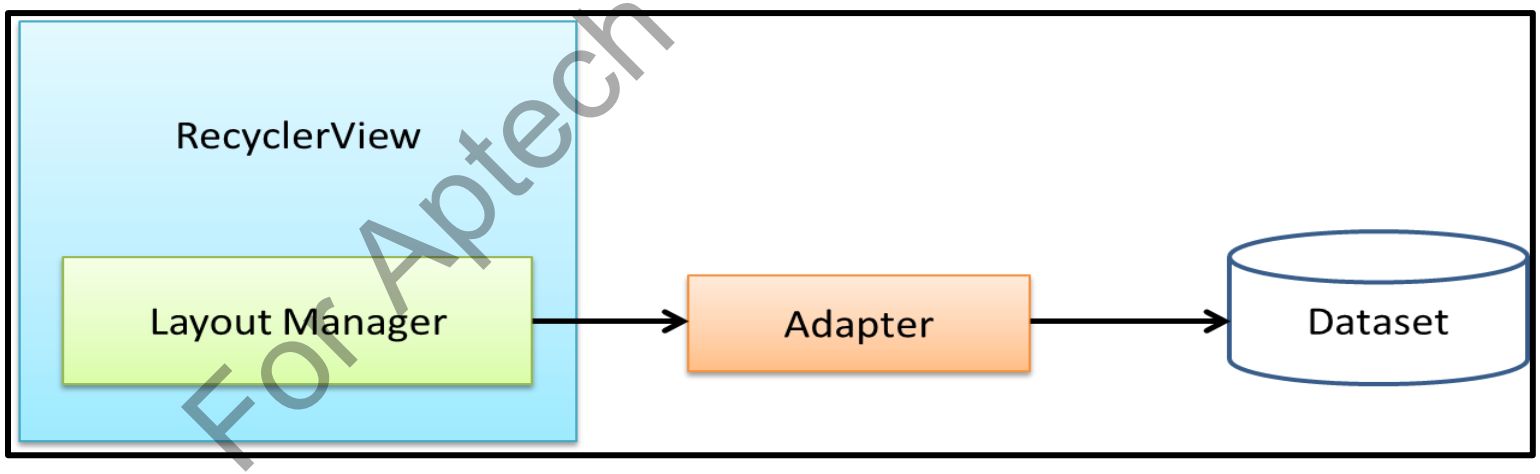
CardView widget

- Allows the user to display important information inside cards

Create a List

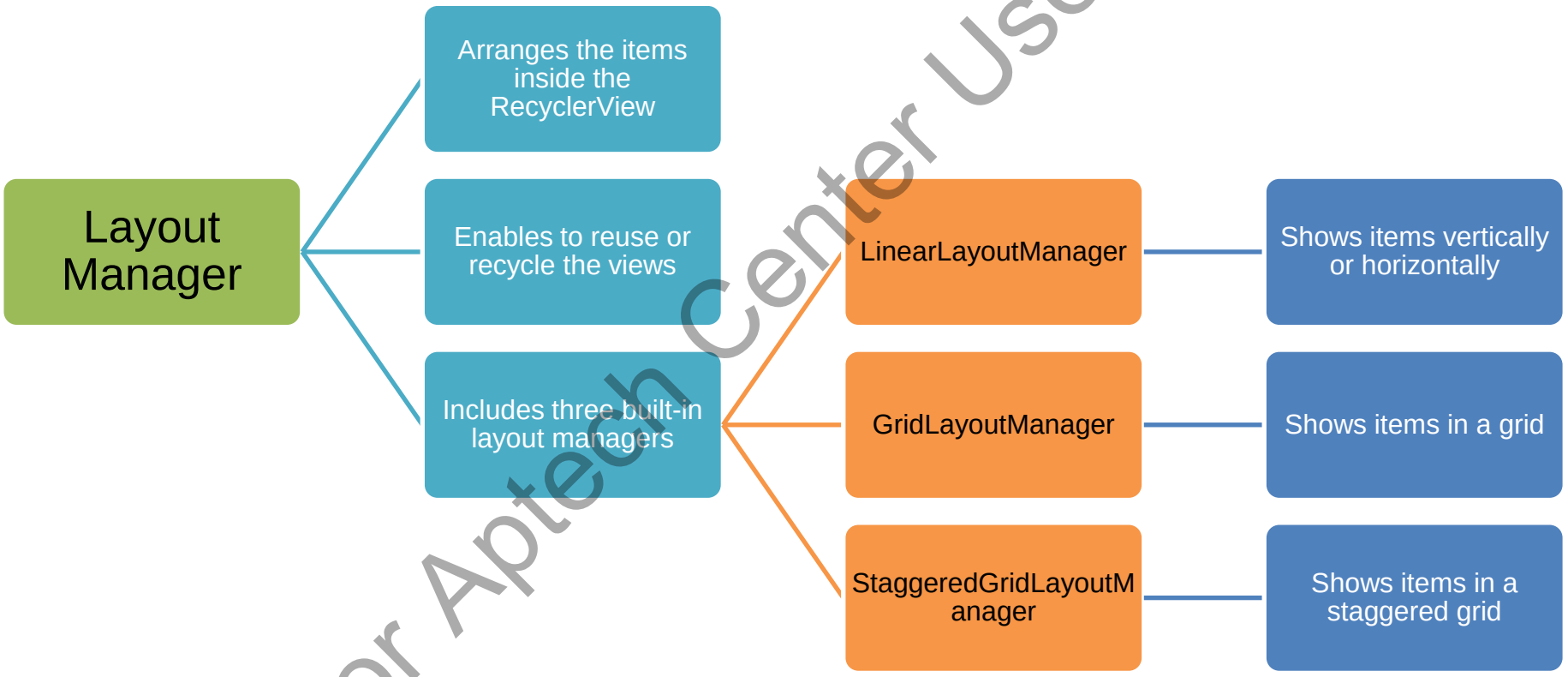
RecyclerView Widget

- Displays and handles huge amount of data with the help of scroll bars
- Displays data that changes at runtime based on user inputs or actions
- Provides layout managers for data items
- Provides default animations for items



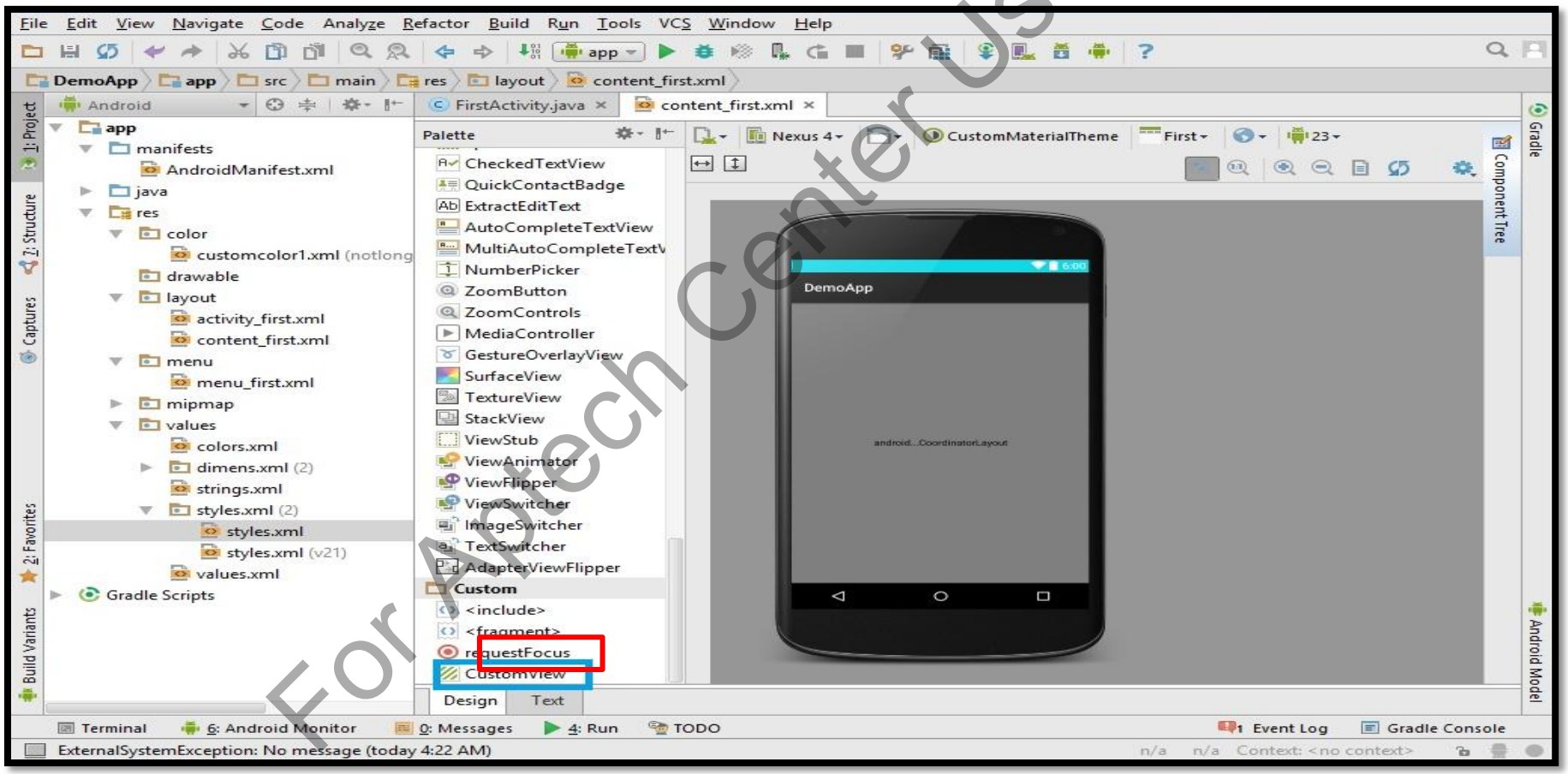
Layout Manager

Create a List



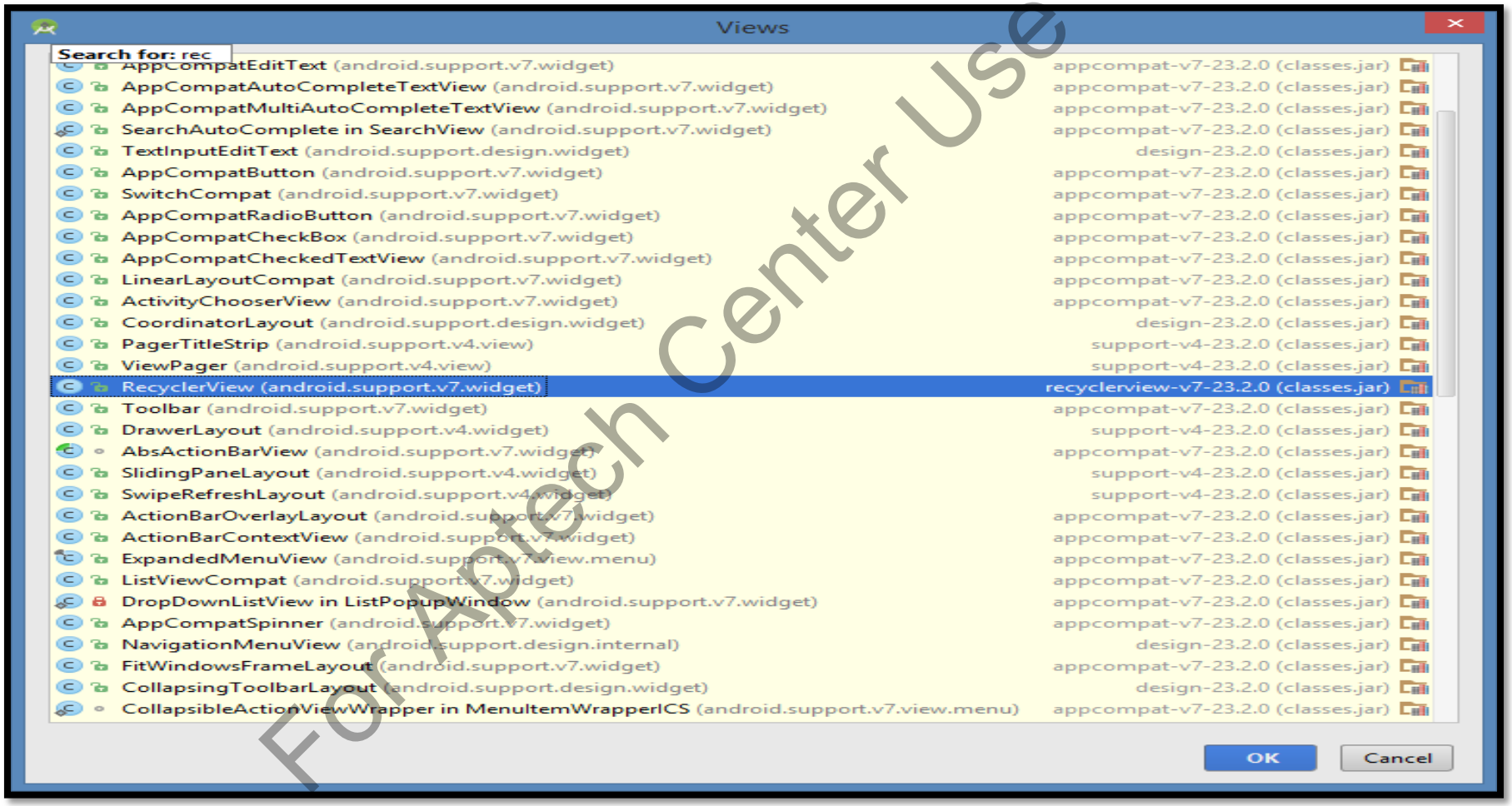
Custom View

To add a RecyclerView widget to a layout, a custom view needs to be created.



List of Custom Views

Following figure shows the list of views that can be added:



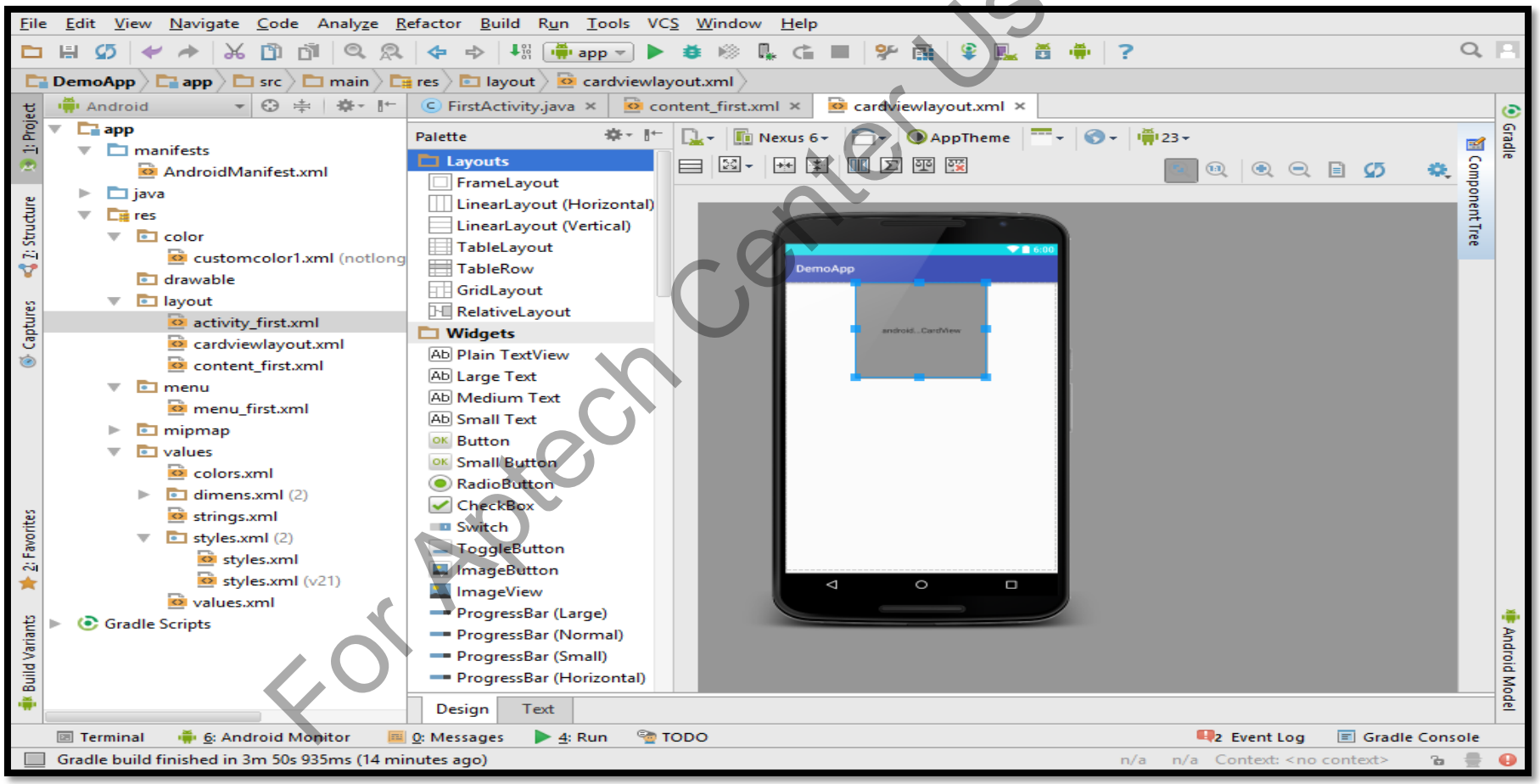
Adding RecyclerView to Layout

Following code snippet demonstrates how to add the RecyclerView widget to a layout:

```
<!-- A RecyclerView -->
<android.support.v7.widget.RecyclerView
    android:id="@+id/view"
    android:scrollbars="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

CardView Widget

To add a CardView widget, drag the widget from the list of widgets to the work area in the Design mode.



Adding CardView to Layout 1-2

Following code snippet demonstrates how to add the CardView widget to a layout:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    ... >
    <!-- A CardView with a TextView -->
    <android.support.v7.widget.CardView
        xmlns:card_view="http://schemas.android.com/apk/res-auto"
        android:id="@+id/mycardview"
        android:layout_gravity="center"
        android:layout_width="200dp"
        android:layout_height="200dp"
        mycardview:cardCornerRadius="4dp">
```

Adding CardView to Layout 2-2

Following code snippet demonstrates how to add the CardView widget to a layout:

```
<TextView
            android:id="@+id/cardtext"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:text="This is a Card" />
    </android.support.v7.widget.CardView>
</LinearLayout>
```

Z Property

The Z Property:

- Is an addition to the existing X and Y properties
- Represents the view's elevation
- Determines the shadow's size, appearance, and the drawing order of the views
- Is measured in dp

The two components for the Z value of the view:

- **Elevation**: It is the static component.
- **Translation**: It is the dynamic component used for animations.

Customizing View Shadows 1-2

Following code snippet demonstrates how to customize view shadows:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context="com.syskan.demoapp.FirstActivity"
    tools:showIn="@layout/activity_first">
```

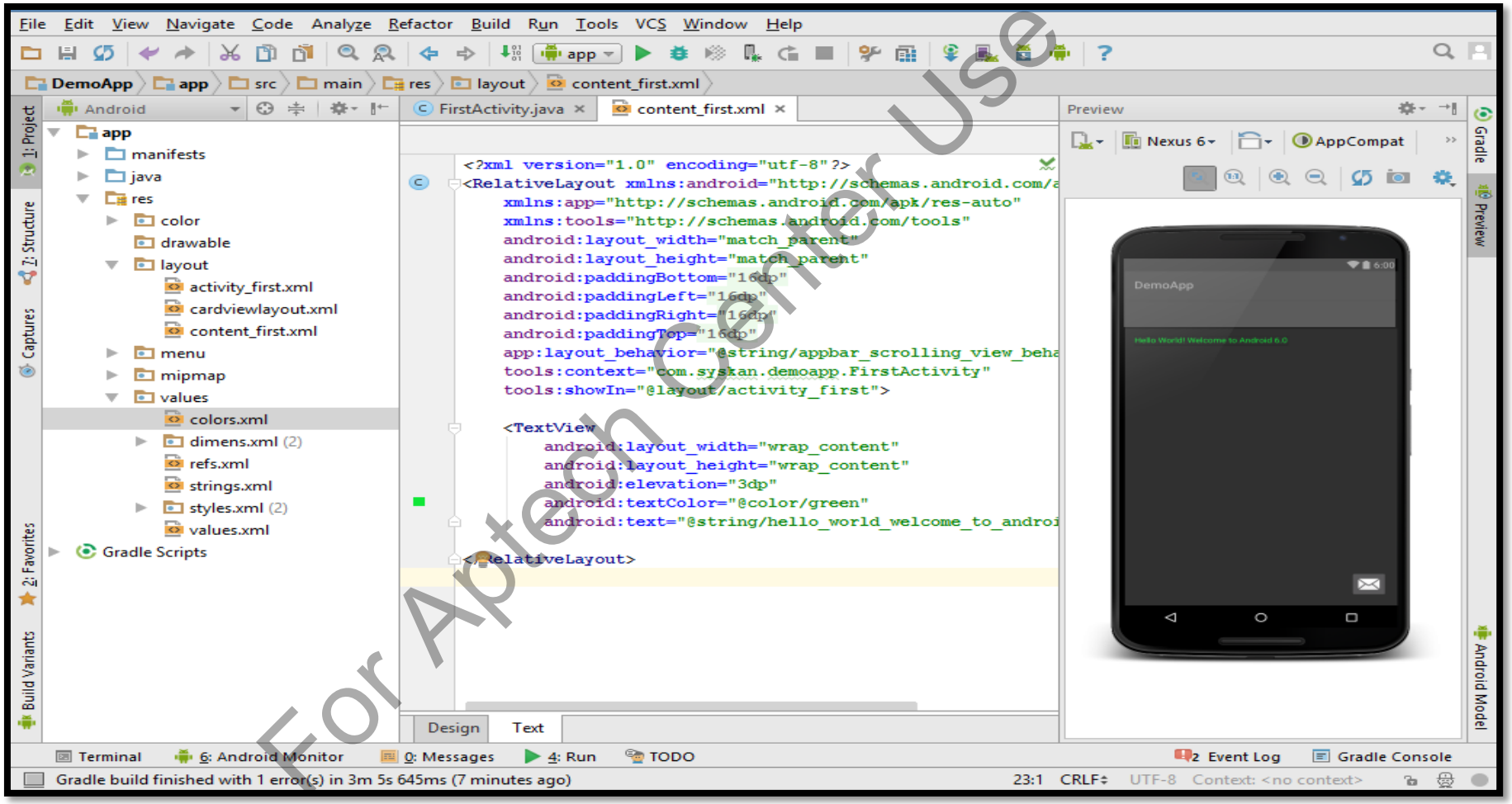
Customizing View Shadows 2-2

Following code snippet demonstrates how to customize view shadows:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:elevation="3dp"
    android:textColor="@color/green"
    android:text="@string/hello_world_welcome_to_android_6_0" />
</RelativeLayout>
```

View Shadows

The View Shadow appears.



Changing Shape of the View

Clipping Views

- Change the shape of the views
- Are applied to outline area
- Can be applied to rectangle, circle, and round rectangle

View.setClipToOutline() method

android:clipToOutline attribute

To clip view to the outline area

Creating Touch-based Custom Animations for UI Components

Touch feedback	• Provides instant feedback to the users
Circular Reveal	• Provides visual continuity to users during showing or hiding of elements
Activity transitions	• Provide visual connections between different states of views
Curved motion	• Enables to customize the timing curves and curve motion patterns
View state change	• Enables to animate changes to view state in Android

Use Hide And Reveal Animation Effect On A View 1-2

Following code snippet demonstrates how to use hide and reveal animation effect on a view:

```
// previously invisible view
View myView = findViewById(R.id.mycardview);
// get the center for the clipping circle
int width = myView.getWidth() / 2;
int height = myView.getHeight() / 2;
float finalRadius = (float) Math.hypot(width, height);
Animator anim =
   ViewAnimationUtils.createCircularReveal(myView, width, height, 0,
finalRadius);
myView.setVisibility(View.VISIBLE);
anim.start();
// previously visible view
final View myView = findViewById(R.id.mycardview);
```

Use Hide And Reveal Animation Effect On A View 2-2

Following code snippet demonstrates how to use hide and reveal animation effect on a view:

```
int width = myView.getWidth() / 2;
int height = myView.getHeight() / 2;
float initialRadius = (float) Math.hypot(width, height);
// create the animation (the final radius is zero)
Animator anim =
ViewAnimationUtils.createCircularReveal(myView, width, height, initialRadius, 0);
// make the view invisible when the animation is done
anim.addListener(new AnimatorListenerAdapter() {
    @Override
    public void onAnimationEnd(Animator animation) {
        super.onAnimationEnd(animation);
        myView.setVisibility(View.INVISIBLE);
    }});
// start the animation
anim.start();
```

Custom Transitions

Following code snippet demonstrates custom transitions:

```
<style name="BaseAppTheme" parent="android:Theme.Material">
    <!-- enable window content transitions -->
    <item name="android:windowActivityTransitions">true</item>
    <item
name="android:windowEnterTransition">@transition/burst</item>
    <item
name="android:windowExitTransition">@transition/burst</item>
    <!-- specify shared element transitions -->
    <item name="android:windowSharedElementEnterTransition">
        @transition/transform_image</item>
    <item name="android:windowSharedElementExitTransition">
        @transition/transform_image</item>
</style>
```

Customizing Timing Curves and Curve Motion Patterns

Following code snippet demonstrates the usage of curved motion.

```
<pathInterpolator
xmlns:android="http://schemas.android.com/apk/res/android"
    android:controlX1="0.4"
    android:controlY1="0"
    android:controlX2="1"
    android:controlY2="1"/>
ObjectAnimator myAnimator;
myAnimator = ObjectAnimator.ofFloat(view, View.X, View.Y, path);
...
myAnimator.start();
```

Animating Changes to View State 1-2

Following code snippet demonstrates the usage of the StateListAnimator class:

```
<!-- animate the translationZ property of a view when pressed -->
<selector
xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:state_pressed="true">
    <set>
      <objectAnimator android:propertyName="translationZ"
        android:duration="@android:integer/config_shortAnimTime"
        android:valueTo="3dp"
        android:valueType="floatType"/>
    </set>
  </item>
```

Animating Changes to View State 2-2

Following code snippet demonstrates the usage of the StateListAnimator class:

```
<item android:state_enabled="true"
    android:state_pressed="false"
    android:state_focused="true">
    <set>
      <objectAnimator android:propertyName="translationZ"
        android:duration="150"
        android:valueTo="0"
        android:valueType="floatType"/>
    </set>
  </item>
</selector>
```

Summary

- Material design combines good design principles with advanced tools to create a single underlying system across various platforms and mobile devices.
- The difference in the elevations between the overlapping material objects creates shadows in the environment.
- Material motion describes hierarchical and spatial relationships between objects.
- The layout simplifies the process of creating scalable apps by using elements from print-based design.
- The growth and communications guidelines define the best practices that can be adopted for developing the apps.
- RecyclerView widget is a pluggable version of ListView and supports different types of layouts.
- Activity transitions provide visual connections between different states of views.
- The shared elements transition determines the sharing of views between activities transitions.