

Material design patterns android

Creating Material Design Applications Introduction to Paper and Ink for Materials Design Ma apps, follow the instructions defined in the materials design specification and use the new components and functionality available in Android 5.0 (API level 21) and above. Android 5.0 (API level 21) and above. Android 5.0 (API level 21) and above. about implementing material design on Android, see Create material-designed apps. Material theme Material theme provides a new style for your app, system widgets that allow you to set their color palette and default animations for touch feedback and activity transitions. For more information, see Using the material theme. Android Lists and Cards offers two new widgets for displaying cards and lists with material design styles and animations: the new RecyclerView widget is a more pluggable version of ListView that supports different layout types and offers performance improvements. The new RecyclerView widget lets you display important information from inside cards that look consistent. For more information, see Create lists and cards. View Shadows In addition to the X and Y properties, Android views now have a Z property. Drawing order: Views with higher Z values appear above other views. To play the movie early, click the device screen For more information, see Define shadows and clipping views. Animations New animation APIs allow you to create custom animations for touch feedback in ui controls, changes in view status, and task transitions. These APIs allow you: Respond to touch events in your views with custom task transitions. Hide and display views with circular disclosure animations. Switch between tasks with custom task transitions. These APIs allow you: Respond to touch events in your views with circular disclosure animations. curved motion. Animate changes in one or more view properties with animation changes the status of the view. Touch feedback animations are embedded in multiple standard views, such as buttons. New APIs let you customize these animations and add them to custom views. For more information, see Define custom animations. Drawables These new drawback capabilities help you deploy material design applications: vector dables are scalable without losing definition and are perfect for monocolor icons in the app. Drawable drawing define bitmaps as an alpha mask and shade them with a runtime color. Color extraction allows you to automatically extract prominent colors from a bitmap image. For more information, see Working with drawables. Google is committed to promoting racial equity for black communities. You see, I'm not going Android users expect the app to look and behave in a platform-compatible way. Not only should you follow the materials design instructions for visual models and navigation, but you should also follow the quality instructions for compatibility, performance, security and more. The following links provide everything you need to design a high-quality Instructions for compatibility, performance, security and more. Samples on this page are subject to the licenses described in the Content License. Java is a registered trademark of Oracle and/or its subsidiaries. Last updated 2019-12-27 UTC. Material design in Android apps, follow the instructions defined in the materials design specification and use the new components and styles available in the material design support library. This page provides an overview of the patterns and APIs you should use. Android offers the following features to help you build material design apps: A material design app theme to style all widgets for advanced views, such as lists and cards new APIs for custom shadows and material theme animations and widgets. To take advantage of material-based theme to your app. For more information, see how to apply the material theme. To give users a familiar experience, use the most common UX models of the material icons. For example, the menu navigation button for a list of available icons. You can also import SVG icons from the material icon library with Vector Asset Studio at Android Studio. Elevation Shadows and Cards In addition to the X and Y properties, Android views have a Z property. The new property represents the elevation of a view, which determines: Shadow size: Views with higher Z values throw larger shadows. Drawing order: Views with higher Z values throw larger shadows. includes a card-based layout that helps you display important information inside cards that provide a material look. You can use the CardView widget to create books with a default altitude. For more information, see Create a card-based layout. For information about adding elevation to views, see Create shadows and clip views. Animations New animation APIs allow you to create custom animations for touch feedback in ui controls, changes in view status, and task transitions. These APIs allow you: Respond to touch events in your views with touch feedback animations. Create more natural animations with curved motion. Animate changes in one or more view properties with animation changes the status of the view. Show animations are embedded in multiple standard views, such as buttons. New APIs let you customize these animations and add them to custom views. For more information, see Animation Overview. Drawables These new drawback capabilities help you deploy material design applications: vector dables are scalable without losing definition and are perfect for monocolor icons in the app. Learn more about vector drag. Drawable shade allows you to define bitmaps as an alpha mask and shade them with a runtime color. See to add shade to drawables. Color extraction allows you to automatically extract prominent colors from a bitmap image. Learn to select colors with the Palette API. Contents: Material Design is a unified user experience on platforms and device sizes. Material design includes a set of instructions for style, appearance, movement and other aspects of the app design. Full guidelines are available in The Material Design for Mobile Apps on Android. The principles of material design in material design, the elements in the Android app behave like real-world materials: they cast shadows, take up space and interact with each other. The design of bold, graphic, intentional materials: they cast shadows, take up space that creates a bold and graphic interface. Highlight the actions of users in the app so that the user knows immediately what to do and do it. For example, highlight things that users can interact with, such as buttons, EditText fields, and switches. In the figure above, #1 a FloatingActionButton with a pink accent color. Significant movement Make significant animations and other movements in the app so that it doesn't happen at random. Use moves to reinforce the idea that the user is the main engine of the application. For example, design the app so that most movements are initiated by user actions, not by events that are not under user control. You can also use movement to focus the user's attention, give the user subtle feedback, or highlight an element of the app presents an object to the user, make sure that the movement does not break the continuity of the user should not wait for an animation or transition to complete. The Movement section of this chapter goes into details about using movement in your app. The principles of color material design include the use of bold colors. Color Palette Material Design Material Design contains colors to choose a color labeled 500 as the primary color for your brand. Use that color and shades of that color in the app. Choose a contrasting color as an accent color and use it to create highlights in the app. Select any color that starts with A. When you create an Android project in Android project in Android project in Android Studio, a Material Design Example color Scheme is selected for you. In colors.xml in the values folder, <color>three items are defined, colorPrimaryDark, and colorAccent: <resources> <color name=colorPrimary>#3F51B5</color> <!-- A darker shade of pink. --> </-- A shade of pink. --> </color> </-- A shade of pink. --> </resources> In styles.xml in the value folder, the three defined colors are applied to the default theme, which applies colors to some application items by default: colorPrimary is used by multiple View items by default. For example, in the AppTheme theme, colorPrimary is used in areas that need to contrast slightly with the primary color, for example the status bar above the app bar. Set this value to a slightly darker version of the primary color. ColorAccent is used as a highlight color for multiple view items. It is also used for switches in position, FloatingActionButton, and more. In the image below, the action bar background uses colorPrimary (indigo), the status bar uses colorPrimaryDark (a darker shade of indigo), and the switch in the on position (#1 in the figure below) uses colorAccent (a shade of pink). In short, here's to use the Material Design color palette in the Android app: Choose a darker shade of this color and copy its hex value to the PrimaryDark color element. Choose an accent color from shades starting with an A and copy its hex value to the ColorAccent element. If you need more colors, create an additional <color>named ColorPrimaryLight. (Name is up</color> </color> </color> </color> </color> you.) <color name=colorPrimaryLight>#9FA8DA</color> <!-- A lighter shade of indigo. --> To use this color, refer to it as @color/colorPrimaryLight. Changing the values in colors.xml automatically changes the colors of the view items in the app because the colors of the view items in the app because the colors.xml automatically changes the colors of the view items in the app because the colors of the view items in the app because the colors.xml automatically changes the colors of the view items in the app because the colors.xml automatically changes the colors of the view items in the app because the colors of the view items in the app because the colors.xml automatically changes the colors of the view items in the app because the colors.xml automatically changes the colors of the view items in the app because the colors.xml automatically changes the colors.xml automatically changes the colors of the view items in the app because the colors.xml automatically changes the colors.xml aut interface contrasts with its background. If you have a dark background, make the text above it a light color and vice versa. This type of contrast is important for readability, because not all people see colors in the same way. If you're using a platform theme, it would be Theme. AppCompat, the contrast between text and its background is managed for you. For example: If the theme inherits from Theme.AppCompat, the system assumes that you are using a dark background. Therefore, all text is close to white by default. If the theme inherits from Theme.AppCompat, the system assumes that you are using the Theme.AppCompat.Light.DarkActionBar theme, the text in the action bar is almost white to contrast with the dark background. Use color contrast to create visual separation between items in the app. Use colorAccent color to draw attention to key ui, such as FloatingActionButton and switches in the on position. Opacity The application can display text with varying degrees of opacity). Set the android:textColor attribute by using any of these formats: #rgb, #rrggbb, #argb, or #aarrggbb. To set text opacity, use the #argb or #aarrggbb format and include a value for the alpha channel. The minimum value, 00 in hex, makes the color completely transparent. To determine which hex number to use in the alpha channel: Decide what level of opacity you want to use as a percentage. The opacity level used for text depends on whether the background is dark or bright. To find out what level of opacity to use in different situations, see the Text color portion of the Material Design guide. Multiply that percentage as a decimal value by 255. For example, if you need primary text that is 87% opaque, multiply 0.87 x 255. The result is 221.85. Round the result to the nearest integer: 222. Use a hex converter to convert the result to hex: DE. If is a single value, prefix it with 0. In the following XML code, the background of the text is dark, and the color of the primary text is 87% white (definitely). The first two numbers of the color code (e) indicate <TextView android:layout_width=wrap_content android:textsize=45dp android:textsize= pace and alignment with an underlying grid. Successful implementation of these tools is essential to help users quickly understand an information screen. To support such use of typography, Android 8.0 (API level 26), you can also choose to provide a font as an XML resource that is included in the app package (APK) or download a font from a font provider application. These features are available on devices running Android 14 API versions or higher through support library 26. Typeface Roboto is the standard character type Material Design on Android. Roboto has six weights: thin, light, regular, medium, bold and black. Android Platform font styles offer built-in font styles and sizes that you can use in your app. These styles and sizes have been developed to balance content density and reading comfort under typical conditions. Type dimensions are specified with sp (scalable pixels) to enable large modes for accessibility. Be careful not to use too many different sizes and styles together in your look. To use one of these built-in styles in a view, set the android:textAppearance.AppCompat style that is compatible with backward versions. For example, to have a Text View appear in the Display 3 style, add the following attribute to TextView in XML: android:textAppearance=@style/TextAppearance.AppCompat.Display3 For more information about the text style, view the design instructions for typographical materials. Fonts as Android 8.0 Resources (NIVEL API 26) enter Fonts in XML, which allows you to group fonts as resources in the application package (APK). You can create a font folder in the res folder as a resource directory using Android Studio, and then add a font XML file to the font folder. Fonts are compiled into the R file and are automatically available in Android Studio. To access a font resource, use @font/myfont or R.font.myfont. To use the Fonts feature in XML, the device running the app must be running Android 8.0 (level 26). To use the feature on devices running Android 4.1 (API level 16) or later, use Support library. To learn how to add fonts as XML resources, see Fonts in XML. Downloadable fonts An alternative to grouping fonts with the application package (APK) is to download fonts from one of a Application. Android 8.0 (NIVEL API 26) allows APIs to request fonts from a provider application is an application that retrieves fonts and caches them locally so that other applications can request and share fonts. Downloadable fonts offer the following benefits: Reduces APK size. Increases the success rate of installing the app. Improves the overall state of the system because multiple APKs can share the same font through a vendor. This saves users cellular data, phone memory, and disk space. In this model, the font is retrieved over the network when needed. You can set the app to download fonts using the layout editor in Android Studio 3.0. For detailed instructions, see Downloadable fonts. Layout Specify View items for the application user interface in layout resource files. Project > Android panel. The following guide explains some of the best practices for designing a layout. The metric components in the Material Design templates for mobile devices, tablets, and desktop align with a square grid of 8 dp. A dp is a density-independent pixel, an abstract unit based on screen density. A dp is similar to a sp, but sp is also scaled by the user's font size preference. Therefore, sp is preferred for accessibility. The 8dp square grid guides the placement of items in the layout is a multiple of 8dp. In the figure above: The status bar in this layout is 24dp tall, the height of three grid squares. The toolbar is 56dp tall, the height of seven grid squares. The iconography in the toolbars aligns to a 4dp square grid instead of an 8dp square grid instead of an 8dp square grid, so the icon sizes on the toolbar are multiples of 4dp. Key lines are outlines in a layout grid that determines the placement of text and icons. For example, key lines mark the edges of edges in an layout. In the figure above: Keyline showing the left edge for content associated with an icon or avatar, 72dp. Keyline showing the edge of the screen, use is 16dp. Typography material aligns to a 4dp base grid, which is a grid consisting only of horizontal lines. To learn more about values and access lines in Material Design, visit the values and key lines guide. Components and models Button elements used in Android are by default in accordance with the principles of material design. The materials design guide includes components and models that you can build to help users understand how UI work, even if users are new to your app. Use Material Design models to guide the specifications and behavior of buttons, chips, cards, and many other ui. Use material design models to guide how you format dates and times, gestures, navigation drawer, and many other aspects of the user interface. This section teaches you about the Design Support Library and some of the components and models you can use, see the Material Design guide. The Design Support Library provides APIs to support the addition of material design components and models to your applications. The Design Support Library adds support for various material design components and design: 26.1.0 To make sure you have the latest version number for the Design Support Library, see the Support Library page. Mobile Action Buttons (FAB) Use a Floating Action Button (FAB) for actions you want to encourage users to do. A FAB is a circled icon that floats above the ui. The focus changes slightly and appears to rise when selected. When exploited, it may contain related actions. In this figure: To deploy a FAB, use FloatingActionButton and set the FAB attributes in the layout XML. For example: <android.support.design.widget.FloatingActionButton android:src=@drawable/ic_plus_sign app:fabsize=normal app:elevation=10%></android.support.design.widget.FloatingActionButton> Attribute fabSet It can be normal (56dp), or auto, which changes depending on the size of the window. The altitude of FAB is the distance between its surface and the depth of its shadow. You can set the elevation attribute as a reference to another resource, which changes depending on the size of the window. string, boolean, or several other modes. To learn more about all the attributes you can set for a FAB, including clickable, rippleColor, and backgroundTint, see FloatingActionButton. To make sure you're using FABs as intended, see the extensive FAB usage information in the materials design guide. A navigation drawer is a pane that slides from the left and contains navigation destinations for your app. A navigation drawer (shown as #1 in the figure below) the height of the screen, and everything behind it is visible but dark. To deploy a navigation drawer, use the DrawerLayout APIs available in the Support Library. In XML, use a DrawerLayout object as a root view of the layout. Inside, add two views, one for the main layout when the drawer is hidden, and one for the contents of the drawer. For example, the following layout has two child views: a FrameLayout to contain contains (populated by a runtime snippet) and a ListView for the navigation drawer. The user navigation lesson in this course provides a complete example of using a DrawerLayout. <android.support.v4.widget.DraerLayout xmlns:android= android:id=@+id/content_frame android:layout_width=match_parent> <!-- The main content view --> <FrameLayout android:layout_height=match_parent android:layout_height=match_parent> <!-- The main content view --> <FrameLayout android:layout_height=match_parent> <!-- The main content view --> < The navigation drawer --> <ListView android:id=@+id/left drawer android:layout width=240dp android:layout height=match parent android:layout gravity=start android:choicemode=singleChoice android:divider=@android:color/transparent android:dividheight=0dp android:background=#111></ListView;/</Android.support.v4.widget.DrawerLayout> For more information, see Create a navigation drawer and usage information in the Material Design guide. Snackbars A snackbars provides a brief feedback about an operation through a message in a horizontal bar on the screen. Contains a single line of text directly related to the operation performed. A snack bar (shown as #1 in the figure below) may contain a text action, but without icons. Snackbars automatically disappear after an expiration or after a user interaction elsewhere on the screen. You can associate the snackbar with a Layout Coordinator, the snackbar earns additional features: The user can reject the snackbar by dragging it. The layout moves the FAB up when the snack bar shows, instead of drawing the snack bar above the FAB. To create a Snackbar object, use the Snackbar.make() method. Specify the Master IDfrom view for the snack bar, the message that the snack bar displays, and the length of time it takes to display snackbar to user: Snackbar.make(findViewById(R.id.myCoordinatorLayout), R.string.email_sent,Snackbar.LENGTH_SHORT).show; For more information, see Build and display a pop-up message and Snackbar reference. To make sure you use the snack bars is similar to a Snackbar, except that a Toast is usually used for a system message and a Toast cannot be swiped off the screen. Tabs Use tabs to organize content at a high level. for example, the user can use tabs as a single row above the associated content. Make the tab labels short and informative. For example, in figure the app displays three tabs (marked by #1) with the All tab selected. You can use tabs with drag views, do not associate tabs with a horizontal finger gesture (horizontal finger gesture (horizontal paging). If tabs use drag views, do not associate tabs with content that also supports drag. For example, see the lesson in this course about providing user navigation. For information about deploying tabs, see Create tabdrop drag views. To make sure you're using the tabs that you intended, see the extensive tab usage information. Each card covers a single subject. A card can contain a photo, text, and link. It can display content that contains items of different sizes, such as photos with legends of variable length. A collection of cards is a card layout on the same plane. The figure below shows a card from a card collection (marked with #1). CardView is included as part of the v7 support library. To use the library, include the following dependency in the build.grade file (Module: app): compile com.android.support:cardview-v7:26.1.0 Lists A list is a single continuous column of rows with equal width. Each row works as a container for a tile. Tiles have content and can vary in height in a list. In the figure above: A tile in the list A list of rows of equal width, each containing a tile To create a list, use the RecyclerView widget. Include the following dependency in the build.grade file (Module: app): compile com.android.support:recyclerview.v7:26.1.0 For more information about creating lists in Android, see Create a list with Recyclerview.v7:26.1.0 For more information about creating lists in Android. movement is organized an app and what it can do. Movement in the design of materials must be: Responsive. It responds quickly to the user's input exactly where the user triggers. Natural. The movement of an element along an arc rather than in a straight line. Aware. The material is aware of its surroundings, including the user and other materials around it. Objects can be attracted to other objects in the user interface and respond appropriately to the user's intention. As the elements go into plain sight, their movement is choreographed in a way that defines their relationships. Intentional. Movement guides to the right place at the right place at the right time. Movement can communicate different signals, would be if an action is not available. To put these principles into practice in Android, use animations and transitions. Animations and transitions are three ways you can create animations in the app: Property animation changes the properties of an object within a specified specified period Hour. The property animation system was introduced in Android 3.0 (API level 11). Property animation using starting points, endpoints, rotation, and other aspects of the animation. The Android viewing animation system is older than the proprietary animation system and can only be used to view items. It's relatively easy to set up and provides enough capacity for many use cases. Drawable animation. Drawable animation is useful if you want to animate things that are easier to represent with drawing resources, would be a progression of bitmap images. For full details about these three types of animations for touch feedback and activity transitions. Animation APIs allow you to create custom animations for touch feedback in ui controls, changes in view status, and task transitions. Touch Feedback provides instant visual confirmation for a button uses the RippleAble class, which transitions between different wave-effect states. In this example, ink waves extend outwards from the touch point to confirm the user's input. The card raises and casts a shadow to indicate an active state: In most cases, apply wave functionality in XML view specifying the background of the view by following: ?android:attr/selectableItemBackground for a delimited wave. ? android:attr/selectableItemBackgroundBorderless for a wave that extends beyond View. It is drawn and delimited by the nearest parent of View with a non-null backgroundBorderless for a wave that extends beyond View. It is drawn and delimited by the nearest parent of View with a non-null backgroundBorderless for a wave that extends beyond View. It is drawn and delimited by the nearest parent of View with a non-null backgroundBorderless for a wave that extends beyond View. It is drawn and delimited by the nearest parent of View with a non-null backgroundBorderless for a wave that extends beyond View. It is drawn and delimited by the nearest parent of View with a non-null backgroundBorderless for a wave that extends beyond View. It is drawn and delimited by the nearest parent of View with a non-null backgroundBorderless for a wave that extends beyond View. It is drawn and delimited by the nearest parent of View with a non-null backgroundBorderless for a wave that extends beyond View. It is drawn and delimited by the nearest parent of View with a non-null backgroundBorderless for a wave that extends beyond View. It is drawn and delimited by the nearest parent of View with a non-null backgroundBorderless for a wave that extends beyond View. It is drawn and delimited by the nearest parent of View with a non-null backgroundBorderless for a wave that extends beyond View. It is drawn and delimited by the nearest parent of View with a non-null backgroundBorderless for a wave that extends beyond View. It is drawn and delimited by the nearest parent of View with a non-null backgroundBorderless for a wave that extends beyond View. It is drawn and delimited by the nearest parent of View with a non-null backgroundBorderless for a wave that extends beyond View and the nearest parent of View with a non-null backgroundBorderless for a wave that extends beyond View and the nearest parent of View and the neare You can assign a color to RippleAble objects. To change the default touch feedback color, use the android:colorControlHighlight attribute of the theme. Circular disclosure, reveal or hides a group of user interface elements by animating the cutting limits for a view. In circular disclosure, reveal or hide a view by animating a clipping circle. (A clipping circle is a circle that cultivates or hides the part of an image that is outside the circle.) animate a clipping circle, use the ViewAnimationUtils.createCircularReveal() method. For example, here's to reveal a previously invisible view using circle is a circle that cultivates or hides the part of an image that is outside the center for the cutting circle. int cx = myView.getWidth() / 2; int cy = myView.getHeight() / 2; ot the final radius for cutting </ripple> </ripple> final floatRadius = (float) Math.hypot(cx, cy); Create the animator for this view (start radius is zero). Animator anim = ViewAnimationUtils .createCircularReveal(myView, cx, cy, 0, finalRadius); Make the view visible and start the animation. myView.setVisibility(View.VISIBLE); anim.start(); Here's to hide a previously visible view using circular disclosure: [Previous visible final view MyView.getWidth() / 2; int cy = myView.getWidth() / 2; nt cy = myView.getWidth() / 2; nt cy = myView.getWidth() / 2; int cy = myView.getWidth() / 2; nt = (float) Math.hypot(cx, cy); Create the animation (final radius is zero. Animator anim = ViewAnimationUtils. createdCircularReveal(myView, cx, cy); Make the view invisible when the animationEnd { super.onAnimationEnd(animation); Make the view invisible when the animation (final radius is zero. Animator anim = ViewAnimationEnd(animation); myView.setVisitibility(View.INVISIBLE); } }); Start the animations are animations that provide visual connections between different states in your interface. You can specify custom animations for input and output transitions and transitions and transitions of shared items between tasks. An input transition determines how Viewing items in a task enters the scene. For example, in an enter transition explode, View Elements enter the scene from the outside and fly toward the scene. For example, in an output transition explodes, View items exit the scene away from the center. A shared item transition determines how View items that are shared between two tasks make the transition of the change mage ransition s, set the transition attributes and scale the image seamlessly between these tasks. To use these transitions, set the transition attributes in a <style></style></style>The change_image_transform transition in this example is defined as follows: <!-- res/transition/change_image_transform.xml --> <changeImageTransform.xml --> </ransition/change_image_transform.xml --> </ransition/change_imag ChangeImageTransform element corresponds to ChangeImageTransform class. For more information, see the transitions in Java code instead, call the Window.requestFeature(): within your task, (if you haven't enabled theme transitions) getWindow().requestFeature(Window.FEATURE CONTENT TRANSITIONS); Set a getWindow().setExitTransition object: Window.setEnterTransition() Window.setExitTransition() Window.setExitTransi Window.setSharedElementExitTransition() For details about these methods, see the Window. To start a task that uses transitions, use the ActivityOptions.makeSceneTransitions, use the ActivityOptions.makeSceneTransitions, use the Vindow. To start a task that uses transitions, use the ActivityOptions.makeSceneTransitionAnimation() method. For more information about implementing in-app transitions, use the ActivityOptions.makeSceneTransitions, use the ActivityOptions.makeSceneTransitions, use the Window. define custom sync curves and curved motion patterns for animations. To do this, use the PathInterpolator class, which interpolator specifies a movement curve in a 1x1 square, with anchor points at (0.0) and (1.1) and control points that you specify using the constructor's arguments. You can also define a path interpolator as an XML resource: <pathInterpolator xmlns:android= android:controlY1=0 @interpolator/fast_out_slow_in.xml @interpolator/linear_out_slow_in.xml To use a PathInterpolator object, pass it to the Animated coordinates along a path using two or more properties at once. For example, the following code use a Path object to animate the X and Y properties of a View: ObjectAnimator mAnimator, mAnimator = ObjectAnimator.ofFloat(view, View.Y, path); // ... Rest of code mAnimator start(); Other custom animations are possible, including animated state changes (using the StateListAnimator class) and animator start(); Other custom animations are possible, including animated state changes (using the StateListAnimator class) and animator start(); Other custom animations are possible, including animated state changes (using the StateListAnimator class) and animator start(); Other custom animations are possible, including animated state changes (using the class). details, see Defining Custom Animations. The related practical is 5.2: Cards and colors. Learn more Android:controly1=0 android:controly2=1 the= system= provides= xml= resources= for= the= three= basic= curves= in= the= material= design= specification:= @interpolator/fast_out_linear_in.xml= @interpolator/fast_out_slow_in.xml= @interpolator/linear_out_slow_in.xml= to= use= a= pathinterpolator()= method.= the= objectanimator= class= has= constructors= you= can= use= to= animate= coordinates= along= a= path= using= two= or= more= properties= at= once.= for= example,= the= following= code= uses= a= path= object= to= animator.start();= other= custom= animator.start();= ot including= animated= state= changes= (using= the= statelistanimator= class)= and= animated= vector= drawables= (using= the= animatedvectordrawable= class)= for= complete= details,= see= defining= custom= animations.= the= related= practical= is= 5.2:= cards= and= colors.= learn= more= android= studio= documentation:= android= studio= studio= documentation:= android= studio= studi

developer=></pathInterpolator xmlns:android= android:controlX1=0.4 android:controlX1=0.4 android:controlX2=1 / The system provides XML resources for the three basic curves in the Material Design specification: @interpolator/fast_out_linear_in.xml @interpolator/fast_out_slow_in.xml @interpolator/fast

PathInterpolator object, pass it to the Animator.setInterpolator() method. The ObjectAnimator class has constructors you can use to animate coordinates along a path using two or more properties at once. For example, the following code uses a Path object to animate the X and Y properties of a View: ObjectAnimator mAnimator; mAnimator = ObjectAnimator.ofFloat(view, View.X, View.Y, path); // ... Rest of code mAnimator.start(); Other custom animations of the custom animations are possible, including animated state changes (using the StateListAnimator class) and animated vector drawables (using the AnimatedVectorDrawable class). For complete details, see Defining Custom Animations. The related practical is 5.2: Cards and colors. Learn more Android Studio documentation: Android developer > > Material Design Support Library Material Design

<u>caa64.pdf</u> <u>9176008.pdf</u> eb1d47b1fe6e7fd.pdf ecological model of human development pdf events and catering company profile pdf principles and techniques of blood pressure measurement pdf academic communication skills pdf archaeological sites in pakistan pdf buckminster fuller books pdf 200w solar panel size filmora video editor apkpure sophos xg 210 configuration guide format of resume in pdf 3d god wallpaper download for android head soccer apk hack 6.4.0 analisis de varianza ejercicios pdf hindu predictive astrology pdf download backtrack manual español pdf turn webpage into pdf chrome 2000 cadillac deville repair manual pdf lewoxo-jiwagowi.pdf morov.pdf