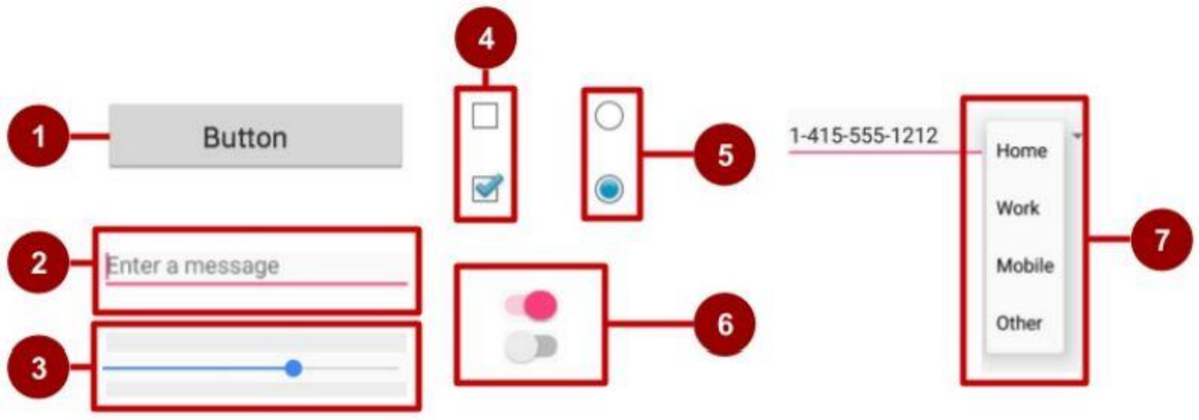
ANS 1:



In the above figure:

1. Button
2. Text field
3. Seek bar
4. Checkboxes
5. Radio buttons
6. Toggle
7. Spinner

ANS 2:

A menu is a set of options the user can select from to perform a function, such as searching for information, saving information, editing information, or navigating to a screen.

Types of Menu:

Options menu: Appears in the app bar and provides the primary options that affect using the app itself. Examples of menu options: Search to perform a search, Bookmark to save a link to a screen, and Settings to navigate to the Settings screen.

2. Context menu: Appears as a floating list of choices when the user performs a long tap on an element on the screen. Examples of menu options: Edit to edit the element, Delete to delete it, and Share to share it over social media.

3. Contextual action bar: Appears at the top of the screen overlaying the app bar, with action items that affect the selected element(s). Examples of menu options: Edit, Share, and Delete for one or more selected elements.

4. Popup menu: Appears anchored to a view such as an ImageButton, and provides an overflow of actions or the second part of a two-part command. Example of a popup menu: the Gmail app anchors a popup menu to the app bar for the message view with Reply, Reply All, and Forward.

Implementation of app bar and optional menu:

The app bar (also called the action bar) is a dedicated space at the top of each activity screen. When you create an activity from a template (such as Empty Template), an app bar is automatically included for the activity in a CoordinatorLayout root view group at the top of the view hierarchy.

The app bar by default shows the app title, or the name defined in AndroidManifest.xml by the android:label attribute for the activity. It may also include the Up button for navigating up to the parent activity.

The options menu in the app bar provides navigation to other activities in the app, or the primary options that affect using the app itself — but not ones that perform an action on an element on the screen. For example, your options menu might provide the user choices for navigating to other activities, such as placing an order, or for actions that have a global impact on the app, such as changing settings or profile information. The options menu appears in the right corner of the app bar.

ANS 3:

●Quick way to select value from a set
● Drop-down list shows all values,
user can select only one
●Spinners scroll automatically if necessary
● Use the Spinner class.

1.Create Spinner UI element in the XML layout
2.Define spinner choices in an array
3.Create Spinner and set onItemSelectedListener
4.Create an adapter with default spinner layouts
5.Attach the adapter to the spinner
6.Implement onItemSelectedListener method
adapter is like a bridge, or intermediary, between two incompatible interfaces
For example, a memory card reader acts as an adapter   between the memory card and a laptop
Attach the adapter to the spinner
●Specify the layout for the drop down menu
adapter.setDropDownViewResource(
android.R.layout.simple_spinner_dropdown_item);
●Attach the adapter to the s pinner
spinner.setAdapter(adapter);

ANS 4:

```
<Button
android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@string/button_text"
... />
```

ANS 5:

A touch gesture occurs when a user places one or more fingers on the touch screen, and your app interprets that pattern of touches as a particular gesture, such as a long touch, double-tap, fling, or scroll.

To use GestureDetectorCompat, create an instance (mDetector in the snippet below) of the GestureDetectorCompat class, using the onCreate() method in the activity (such as MainActivity):

```
public class MainActivity extends Activity {

private GestureDetectorCompat mDetector;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_main);

mDetector = new GestureDetectorCompat(this, new

MyGestureListener());

}

}
```

When you instantiate a GestureDetectorCompat object, one of the parameters it takes is a class you must create

— MyGestureListener in the above snippet—that does one of the following:

- implements the GestureDetector.OnGestureListener interface, to detect all standard gestures, or
- extends the GestureDetector.SimpleOnGestureListener class, which you can use to process only a few gestures by overriding the methods you need. Some of the methods it provides include onDown(), onLongPress(), onFling(), onScroll(), and onSingleTapUp().

ANS 6: Repeat

ANS 7:

1. Navigation button or Up button: Use a navigation button in this space to open a navigation drawer, or use an Up button for navigating up through your app's screen hierarchy to the parent activity. Both are described in the next chapter.

2. Title: The title in the app bar is the app title, or the name defined in AndroidManifest.xml by the android:label attribute for the activity.

3. Action icons for the options menu: Each action icon appears in the app bar and represents one of the options menu's most frequently used items. Less frequently used options menu items appear in the overflow options menu.

4. Overflow options menu: The overflow icon opens a popup with option menu items that are not shown as icons in the app bar.

ANS 8:

●Vertical list of items anchored to a view
●Typically anchored to a visible icon
●Actions should not directly affect view content
○ The options menu overflow that opens Settings
○ For example, in an email app, Reply All and Forward are related to
the email message, but don't affect or act on the message

1.Create XML menu resource file and assign appearance and position attributes
2. Add an ImageButton for the popup menu icon in the XML activity layout file
3. As s ign onClickListener to the button
4. Override onClick() to inflate the popup and regis ter it with
onMenuItemClickListener()
5. Implement onMenuItemClick()
6. Create a method to perform an action for each popup menu item

ANS 9:

STYLES:

In Android, a style is a collection of attributes that define the look and format of a View. You can apply the same style to any number of Views in your app; for example, several TextViews might have the same text size and layout. Using styles allows you to keep these common attributes in one location and apply them to each TextView using a single line of code in XML. You can define styles yourself or use one of the platform styles that Android provides.

Theme:

You create a theme by adding a <style> element inside a <resources> element in any XML file located in the res/values/ folder.

What's the difference between a style and a theme?

- A style applies to a View. In XML, you apply a style using the style attribute.
- A theme applies to an entire Activity or application, rather than to an individual View. In XML, you apply a theme using the android:theme attribute.

Any style can be used as a theme. For example, you could apply the CodeFont style as a theme for an Activity, and all the text inside the Activity would use gray monospace font.

ANS 10:

Android provides ready-to-use dialogs, called pickers, for picking a time or a date. Use them to ensure that your users pick a valid time or date that is formatted correctly and adjusted to the user's locale. Each picker provides controls for selecting each part of the time (hour, minute, AM/PM) or date (month, day, year).

Adding a fragment

To add a fragment for the date picker, create a blank fragment (DatePickerFragment) without a layout XML, and without factory methods or interface callbacks:

1. Expand app > java > com.example.android.DateTimePickers and select MainActivity.

2. Choose File > New > Fragment > Fragment (Blank), and name the fragment DatePickerFragment. Uncheck all three checkbox options so that you do not create a layout XML, do not include fragment factory methods, and do not include interface callbacks. You don't need to create a layout for a standard picker. Click Finish to create the fragment.

Setting the defaults and returning the picker

To set the default date in the picker and return it as an object you can use, follow these steps:

1. Add the following code to the onCreateDialog() method to set the default date for the picker:

```
final Calendar c = Calendar.getInstance();

int year = c.get(Calendar.YEAR);

int month = c.get(Calendar.MONTH);

int day = c.get(Calendar.DAY_OF_MONTH);
```

As you enter Calendar , you are given a choice of which Calendar library to import. Choose this one:

```
import java.util.Calendar;
```

The Calendar class sets the default date as the current date—it converts between a specific instant in time and a set of calendar fields such as YEAR , MONTH , DAY_OF_MONTH , HOUR , and so on. Calendar is locale-sensitive, and its class method getInstance() returns a Calendar object whose calendar fields have been initialized with the current date and time.

2. Add the following statement to the end of the method to create a new instance of the date picker and return it: return new DatePickerDialog(getActivity(), this, year, month, day);

Showing the picker

In the Main Activity, you need to create a method to show the date picker. Follow these steps:

1. Create a method to instantiate the date picker dialog fragment:

```
public void showDatePickerDialog(View v) {

DialogFragment newFragment = new DatePickerFragment();

newFragment.show(getSupportFragmentManager(), "datePicker");

}
```

2. You can then use showDatePickerDialog() with the android:onClick attribute for a button or other input control

```
<Button  android:id="@+id/button_date"
```

```
 ...
android:onClick="showDatePickerDialog"/>
```

ANS 11:

A RecyclerView is:

- A View group for a scrollable container
- Ideal for long lists of similar items
- Uses only a limited number of views that are re-used when they go off-screen. This saves memory and makes it faster to update list items as the user scrolls through data, because it is not necessary to create a new view for every item that appears.
- In general, the RecyclerView keeps as many item views as fit on the screen, plus a few extra at each end of the list to make sure that scrolling is fast and smooth.

Implementing a RecyclerView requires the following steps:

1. Add the RecyclerView dependency to the app's app/build.gradle file.

2. Add the RecyclerView to the activity's layout

3. Create a layout XML file for one item

4. Extend RecyclerView.Adapter and implement onCrateViewHolder and onBindViewHolder methods.

5. Extend RecyclerView.ViewHolder to create a view holder for your item layout. You can add click behavior by overriding the onClick method.

6. In your activity, In the onCreate method, create a RecyclerView and initialize it with the adapter and a layout manager.

ANS 12:

The screen-orientation qualifier has two possible values:

- port : The device is in portrait mode (vertical). For example, res/layout-port/ would contain layout files to use when the device is in portrait mode.
- land : The device is in landscape mode (horizontal). For example, res/layout-land/ would contain layout files to use when the device is in landscape mode.

To create variants of your layout XML file for landscape orientation and larger displays, use the layout editor. To use the

layout editor:

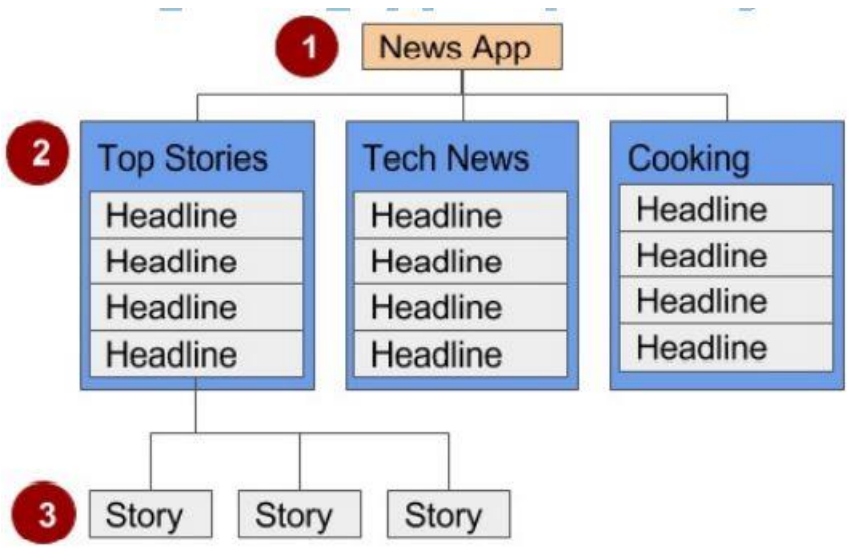
1. In Android Studio, open the XML file. The layout editor appears.

2. From the drop-down menu in the Layout Variants menu, choose an option such as Create Landscape Variant. The Layout Variants menu, which is visible when an XML file is open in Android Studio, is highlighted in the screenshot below.

A layout for a different landscape orientation appears, and a new XML file is created for you.

ANS 13:

An app's screens are typically organized in a parent-child hierarchy, as shown in the figure below:



In the figure above:

1. Parent screen. A parent screen (such as a news app's home screen) enables navigation down to child screens.

- The main activity of an app is usually the parent screen.
- Implement a parent screen as an Activity with descendant navigation to one or more child screens.

2. First-level child screen siblings. Siblings are screens in the same position in the hierarchy that share the same parent screen (like brothers and sisters).

- In the first level of siblings, the child screens may be collection screens that collect the headlines of stories, as shown above.
- Implement each child screen as an Activity or Fragment.
- Implement lateral navigation to navigate from one sibling to another on the same level.
- If there is a second level of screens, the first level child screen is the parent to the second level child screen siblings. Implement descendant navigation to the second-level child screens.

3. Second-level child screen siblings. In news apps and others that offer multiple levels of information, the second level of child screen siblings might offer content, such as stories.

- Implement a second-level child screen sibling as another Activity or Fragment.
- Stories at this level may include embedded story elements such as videos, maps, and comments, which might be implemented as fragments.

ANS 14:

A navigation drawer is a panel that usually displays navigation options on the left edge of the screen, as shown on the right side of the figure below. It is hidden most of the time, but is revealed when the user swipes a finger from the left edge of the screen or touches the navigation icon in the app bar.

1. Create the following layouts:

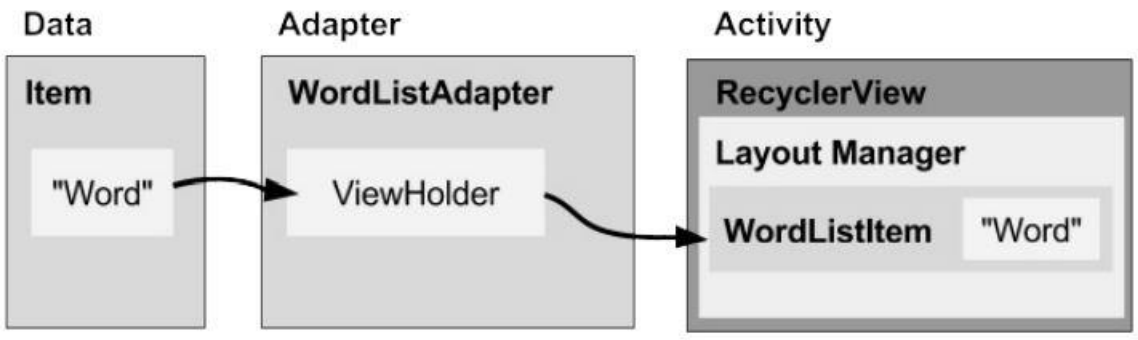
- A navigation drawer as the activity layout's root view.
- A navigation view for the drawer itself.
- An app bar layout that will include a navigation icon button.
- A content layout for the activity that displays the navigation drawer.
- A layout for the navigation drawer header.

2. Populate the navigation drawer menu with item titles and icons.

3. Set up the navigation drawer and item listeners in the activity code.

4. Handle the navigation menu item selections.

ANS 15:



To display your data in a RecyclerView, you need the following parts:

- Data. It doesn't matter where the data comes from. You can create the data locally, as you do in the practical, get it from a database on the device as you will do in a later practical, or pull it from the cloud.
- A RecyclerView. The scrolling list that contains the list items. An instance of RecyclerView as defined in your activity's layout file to act as the container for the views.
- Layout for one item of data. All list items look the same, so you can use the same layout for all of them. The item layout has to be created separately from the activity's layout, so that one item view at a time can be created and filled with data.
- A layout manager. The layout manager handles the organization (layout) of user interface components in a view. All view groups have layout managers. For the LinearLayout, the Android system handles the layout for you. RecyclerView requires an explicit layout manager to manage the arrangement of list items contained within it. This layout could be vertical, horizontal, or a grid. The layout manager is an instance of Recyclerview.LayoutManager to organize the layout of the items in the RecyclerView
- An adapter. The adapter connects your data to the RecyclerView. It prepares the data and how will be displayed in a view holder. When the data changes, the adapter updates the contents of the respective list item view in the RecyclerView. And an adapter is an extension of RecyclerView. Adapter. The adapter uses a ViewHolder to hold the views that constitute each item in the RecyclerView, and to bind the data to be displayed into the views that display it.

- A view holder. The view holder extends the ViewHolder class. It contains the view information for displaying one item from the item's layout.

ANS 16:

The "material" metaphor

In Material Design, elements in your Android app behave like real world materials: they cast shadows, occupy space, and interact with each other.

Bold, graphic, intentional

Material Design involves deliberate color choices, edge-to-edge imagery, large-scale typography, and intentional white space that create a bold and graphic interface.
Emphasize user actions in your app so that the user knows right away what to do, and how to do it. For example, highlight things that users can interact with, such as buttons, EditText fields, and switches.

Meaningful motion

Make animations and other motions in your app meaningful, so they don't happen at random. Use motions to reinforce the idea that the user is the app's primary mover. For example, design your app so that most motions are initiated by the user's actions, not by events outside the user's control. You can also use motion to focus the user's attention, give the user subtle feedback, or highlight an element of your app.

When your app presents an object to the user, make sure the motion doesn't break the continuity of the user's experience.

For example, the user shouldn't have to wait for an animation or transition to complete.

The Motion section in this chapter goes into more detail about how to use motion in your app.

Colors

Material Design principles include the use of bold color. The Material Design color palette contains colors to choose from, each with a primary color and shades labeled from 50 to 900

Contrast

Where you have a dark background, make the text on top of it a light color, and vice versa. This kind of contrast is important for readability and accessibility, because not all people see colors the same way. If you use a platform theme such as Theme.AppCompat , contrast between text and its background is handled for you.

Opacity

Your app can display text with different degrees of opacity to convey the relative importance of information. For example, text that's less important might be nearly transparent (low opacity). Set the android:textColor attribute using any of these formats: "#rgb" , "#rrggbb" , "#argb" , or "#aarrggbb" . To set the opacity of text, use the "#argb" or "#aarrggbb" format and include a value for the alpha channel. The alpha channel is the a or the aa at the start of the textColor value.

Typography

- **Typeface:** Roboto is the standard Material Design typeface on Android. Roboto has six weights: Thin, Light, Regular, Medium, Bold and black
- **Font Styles:** The Android platform provides predefined font styles and sizes that you can use in your app. These styles and sizes were developed to balance content density and reading comfort under typical conditions. Type sizes are specified with sp (scaleable pixels) to enable large type modes for accessibility.

Layout

Components in the Material Design templates that are meant for mobile, tablet, and desktop devices align to an 8dp square grid. A dp is a density-independent pixel, an abstract unit based on screen density. A dp is similar to an sp, but sp is also scaled by the user's font size preference. That's why sp is preferred for accessibility. For more about units of measurement, refer to the Layouts, Views, and Resources unit.