Programming in Android

Session: 4

Android User Interface (UI)

Objectives

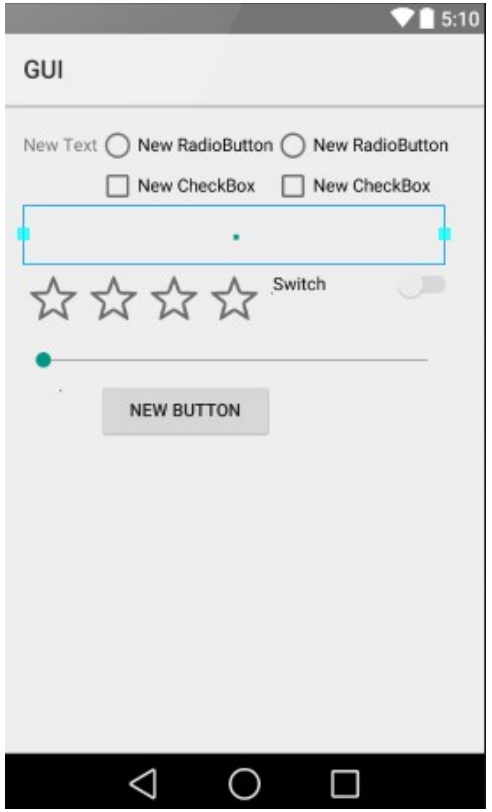
- Explain the process to create the UI for Android applications
- Describe the views, layouts, UI components, styles, and themes
- Explain the procedure for handling user events

Introduction

- UI is a space that enables interaction between the user and the machine
- UI Enables the following actions:
  - Input
  - Output
- Touch screen, Graphical User Interface (GUI) and Text-based User Interface (TUI)

Android User Interface

- User Interface is what the user will see and interact with to perform operations
- Android comes with many friendly UI elements and layouts which help to build interactive applications
- Basic concepts of UI are:
  - Views
  - Layouts
  - UI Components

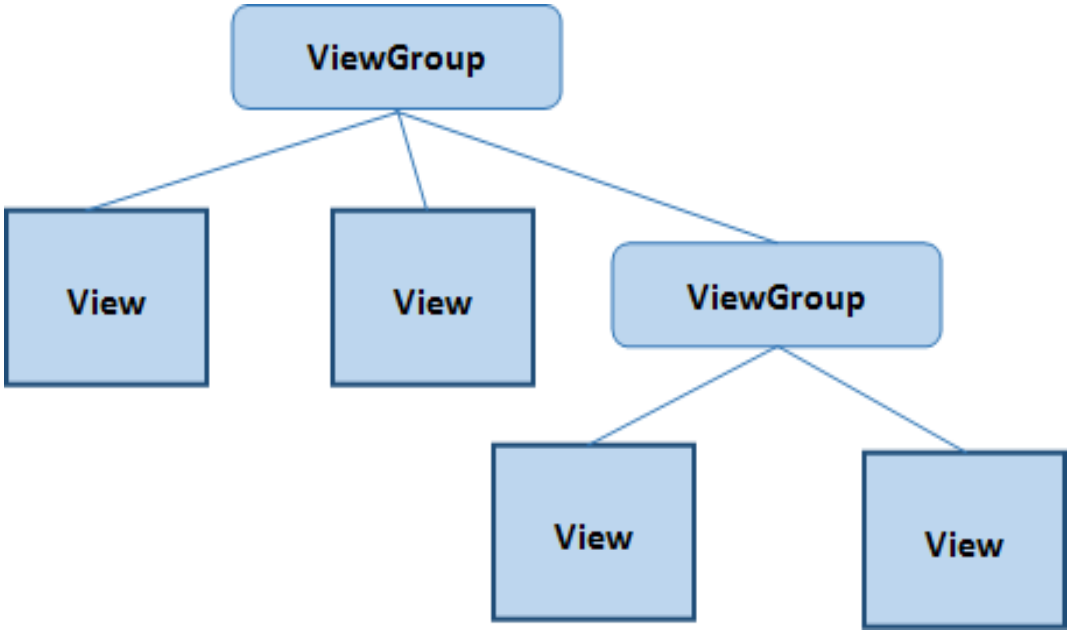


Android UI Components

- Input Controls
- Input Events
- Menus
- Action Bar
- Settings
- Dialogs
- Additional Views
- Status Notifications
- Toasts
- Search
- Drag and Drop

Views

- All UI components in Android are built using View and ViewGroup objects
- Each item in the Android UI belongs to the View class
- Views are also known as widgets
- A set of several views make a ViewGroup
- Each ViewGroup can have other ViewGroup and View within. These are called child Views



Concepts of Views

- Focus
- Listeners
- Attributes
  - ID
  - Setting the Padding
  - Size of a View
  - Setting View Location

Overview of Layouts

- A layout is an extension of ViewGroup class
- It defines the visual structure of the UI
- It mainly comprises interconnected child views
- It helps to define the architecture of the UI present in an Activity
- The developer can either write code for specifying a layout or create an XML file
- Every element in the XML file is either a View or a ViewGroup object
- The Android framework gives the developer flexibility to use either or both methods to declare and control the UI layout

Advantages of Declaring UI in XML

- It enables the developer to separate the presentation of your application from the application logic that controls its behavior
- The developer's UI descriptions are external to the application code, so that the developer can make changes without having to modify the source code and recompile
- Declaring the layout in XML also helps in easy visualization of the structure of the UI
- XML declarations are reflected in the GUI editor of the IDE
- UI declared in XML can easily be reused in other projects. As a result, it is easier to debug problems

Rules for writing Layout in XML

- Declaring UI elements in XML follows the same structure that one follows while naming classes and methods
- The direct relation between the class name and methods with that of UI element name and its attributes helps the developer to understand easily
- Vocabulary is same except for some instances where there are naming differences
- Each layout file must contain exactly one root element, which must be a View or ViewGroup object
- The developer needs to define the root element and then add additional layout objects or widgets as child elements
- Once the developer declares the default layout in a XML file, it is necessary to save it with the .xml extension in Android projects res/layout/ directory to compile it properly

Working with XML UI

- **Load the XML Resource**
  - The XML file in an application is compiled into a View resource
  - The developer must load the View resource in the application code using the onCreate() callback method of the Activity class
  - A reference to the View resource must be provided in the setContentView() call back feature as well

- **Reference the Widget Programmatically**
  - Define a View or widget in the layout file, activity_main.xml and assign a unique ID
  - Create an instance for the View object and obtain its reference programmatically from the layout using the findViewById() method

Loading an XML Layout/Referencing an XML View

- The process for loading an XML layout is shown in the following Code Snippet:

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
```

- The process for referencing an XML view is shown in the following Code Snippet:

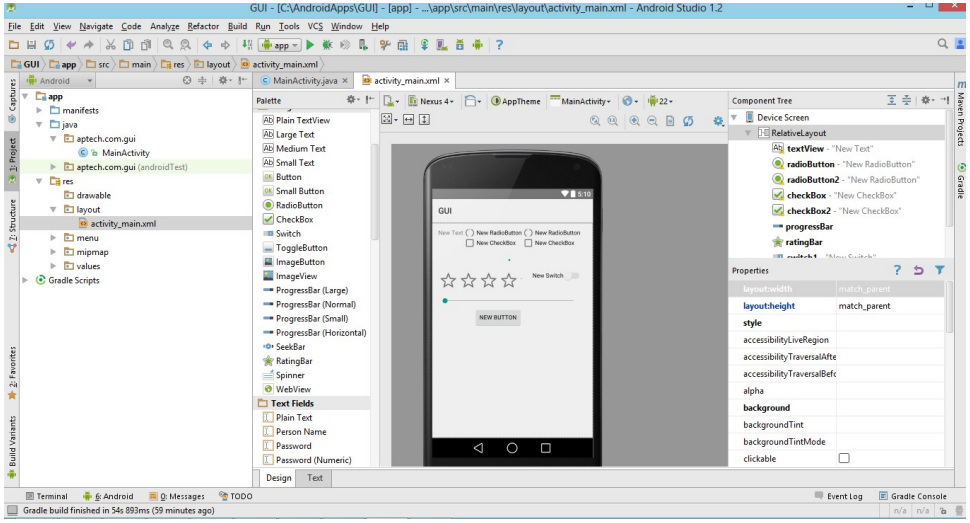
```
<Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Button"
        android:id="@+id/button"
        android:layout_below="@+id/seekBar"
        android:layout_alignEnd="@+id/ratingBar" />
…


Button buttonName = (Button) findViewById(R.id.button);
```

Working with Layouts 1-2

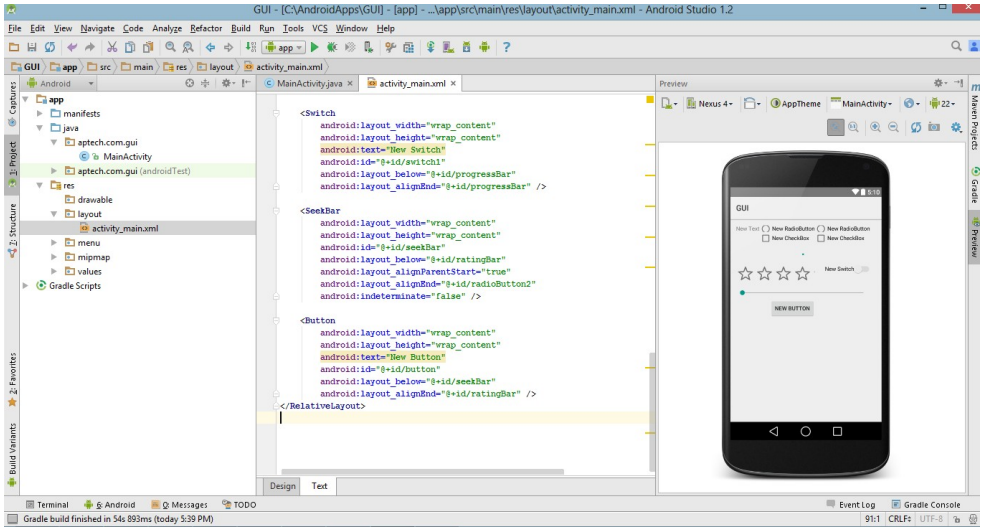
- It is preferable that the developers work using XML from external resources
- The XML layout will comprise a root node
- This root node can have multiple nested layouts and views for designing the UI
- Layouts can be defined in two views:
  - Graphical View
  - Text View

Working with Layouts 2-2

- The Views for editing XML Layouts is shown in the following figures:



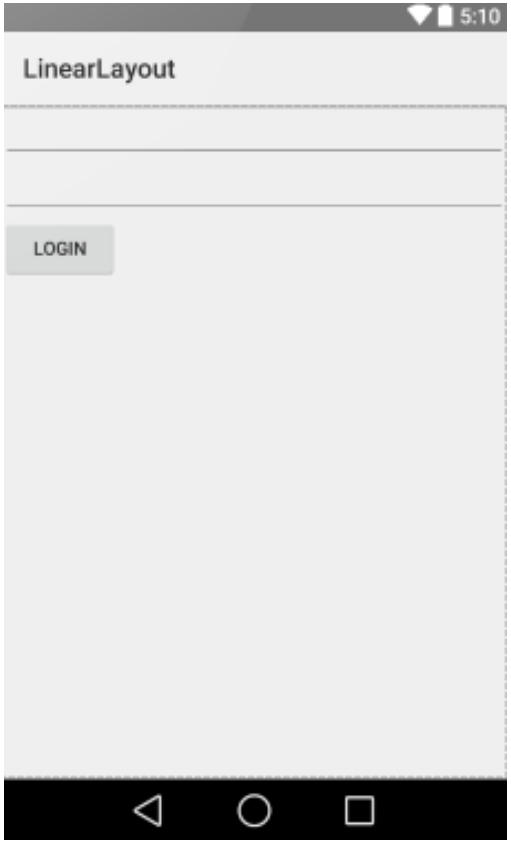
Graphical View



Text View

Linear Layout

- It is the simplest type of layout
- Linear Layout aligns all its child nodes in a single direction, either vertically or horizontally
- The android:orientation attribute helps the developer to specify the layout direction
- A scrollbar appears if the window length exceeds the length of the screen



Linear Layout Example

- The process creating a Linear layout is shown in the following Code Snippet:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" >

    <EditText
        android:id="@+id/userName" />

    <EditText
        android:id="@+id/passWord" />

    <Button
        android:text="Login"
       android:id="@+id/button" />

</LinearLayout>
```

Relative Layout

- The developer can specify the location of child objects relative to each other or to the parent
- It can remove the need for nested view groups keeping the layout tree simple
- If a developer has several nested linear layout groups, the groups could be replaced with a single relative layout instead

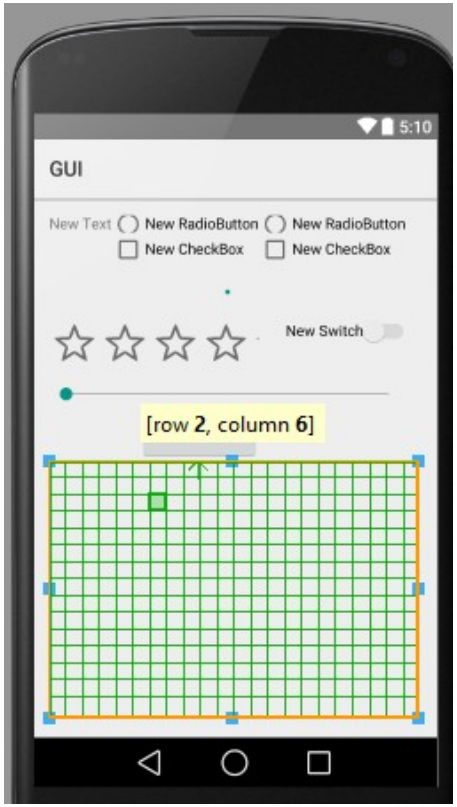
Relative Layout Example

- The process creating a Relative layout is shown in the following Code Snippet:

```
<RelativeLayout android:layout_width="match_parent"
android:layout_height="match_parent"    android:id="@+id/relativeLayout">

    <TextView
        android:layout_alignParentStart="true" />

    <TextView
        android:layout_toStartOf="@+id/editText" />

    <EditText
        android:layout_toEndOf="@+id/userName" />

    <EditText
        android:layout_toEndOf="@+id/userName" />

    <Button
        android:layout_alignStart="@+id/editText2" />
</RelativeLayout>
```

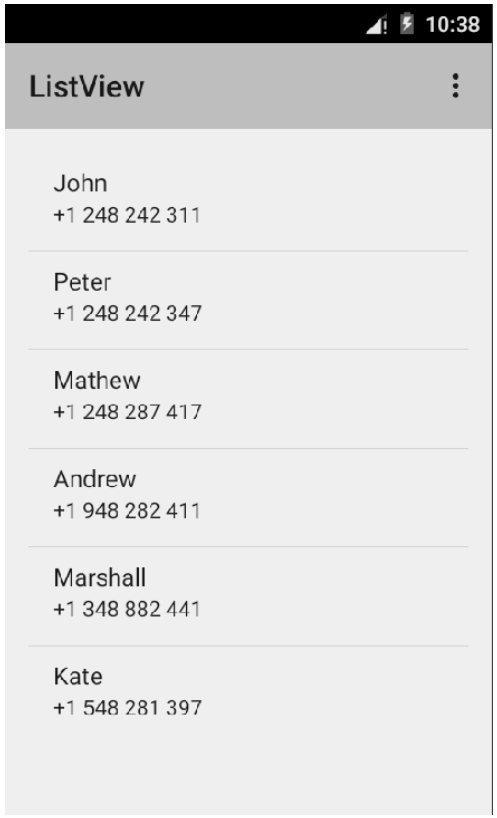
Table Layout

- The screen area is divided in a tabular format
- The developer can specify the row and column position for each view inside the layout
- It is one of the most commonly used layouts in UI design



List View

- ListView is a view group that displays a list of scrollable items
- An Adapter that pulls content from a source, such as a query or an array, helps to insert the list items automatically
- Each item result is converted into a View and added to the list by the Adapter



List View Example 1-2

- The process of creating a List View is shown in the following Code Snippet:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
    android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".ListActivity">

    <ListView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/listView"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true" />
</RelativeLayout>
```

List View Example 2-2

- The process of adding elements to a List View is shown in the following Code Snippet:

```
ArrayList contactsList = new ArrayList<Map<String, Object>>();

Map<String, Object> contact = new HashMap<String, Object>();
        contact.put("name","John");
        contact.put("number","+1 248 242 311");
        contactsList.add(contact);
…


SimpleAdapter adapter = new
SimpleAdapter(this,contactsList,android.R.layout.simple_list_item_2,new
String[]{"name", "number"}, new int[] {android.R.id.text1,
android.R.id.text2});
        listView.setAdapter(adapter);
    }
…
```

Grid View

- This layout displays a scrolling grid consisting of rows and columns
- GridView is a ViewGroup that displays items in a scrollable grid
- The grid items are automatically inserted to the layout using a ListAdapter

Grid View Example 1-2

- The process creating a Grid View is shown in the following Code Snippet:

```
<?xmlversion="1.0"encoding="utf-8"?>
<GridViewxmlns:android="http://schemas.android.com/apk/res/
android"
android:id="@+id/gridView1"
android:numColumns="auto_fit"
android:gravity="center"
android:columnWidth="100dp"
android:stretchMode="columnWidth"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
</GridView>
```

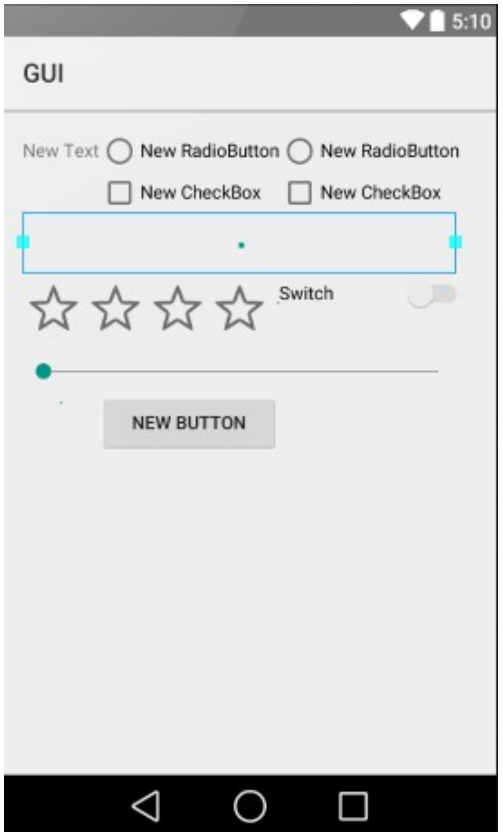
Grid View Example 2-2

- The process of adding elements to a Grid View is shown in the following Code Snippet:

```
staticfinal String[] IMAGES = new String[] { “Image 1”,
“Image 2”, “Image 3”, “Image 4” };
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
gridView = (GridView) findViewById(R.id.gridView1);
gridView.setAdapter(new ImageAdapter(this, IMAGES));
gridView.setOnItemClickListener(new
OnItemClickListener() {
public void onItemClick(AdapterView<?> parent,
View vw,
int position, long id) {
Toast.makeText(getApplicationContext(),
“Click ListItem Number ”+ position,Toast.LENGTH_ LONG)
                                                    .show();
}
});
```

UI Components

- UI components are interactive controls in an app's UI that enables the user to perform a wide range of actions
- Through UI components, the user can interact with the application
- Some of the basic UI components are Button, TextView, DatePicker, ProgressBar, and so on



Input Controls

- Android provides a number of input controls

Following table lists the different input controls:

Input Control	Syntax
Button	<Button … />
Check box	<CheckBox … />
Radio button	<RadioButton … />
Spinner	<Spinner … />
Picker	<DatePicker … />
Switch	<Switch … />
SeekBar	<SeekBar …/>
Toggle Button	<ToggleButton …/>
Text Input	<EditText … />

Display Views

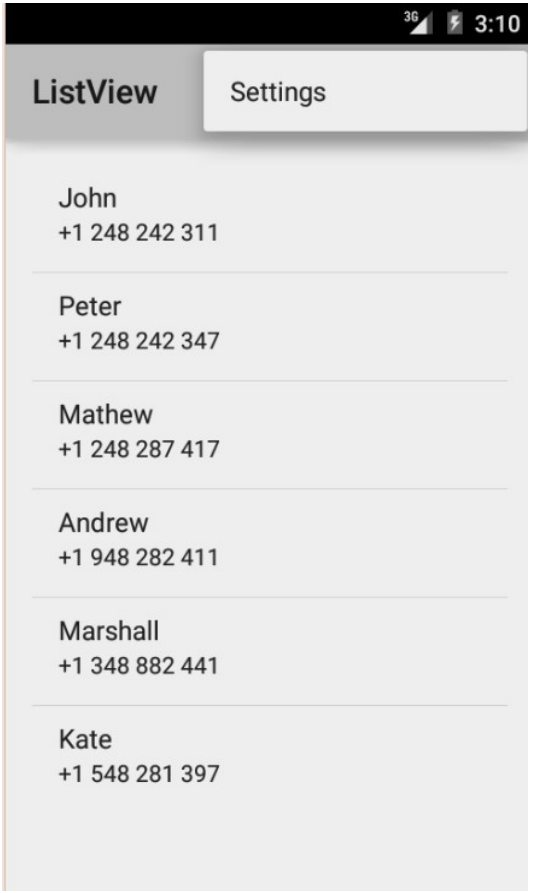
- Android provides a number of Display Views

Following table lists the different Display Views:

Display View	Syntax
Text View	<TextView … />
Progress Bar	<ProgressBar … />
Image View	<ImageView … />
Video View	<VideoView … />

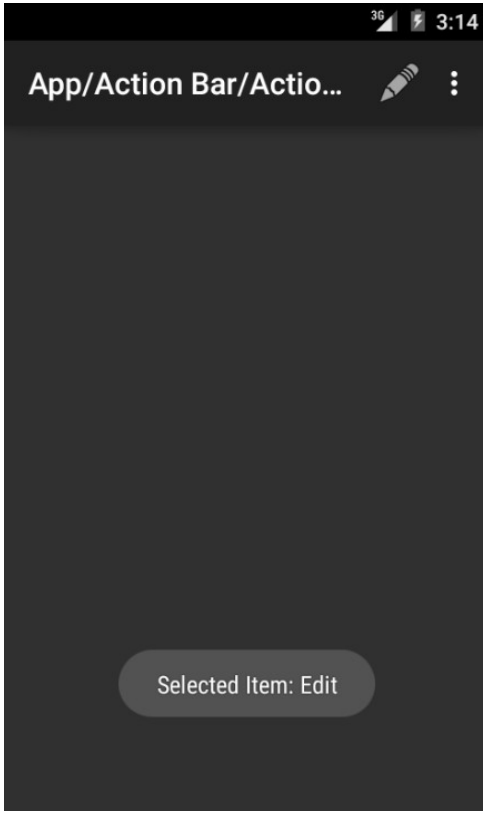
Menus

- Menu is an UI component which holds several items that provide navigation or settings or more functionality to an application
- It is a common interface component seen in Android phones and appears when menu buttons on the device are clicked
- The menu displays all available options
- Android 4.0 onwards, the standard menu button on device has been replaced with the Action bar menu button



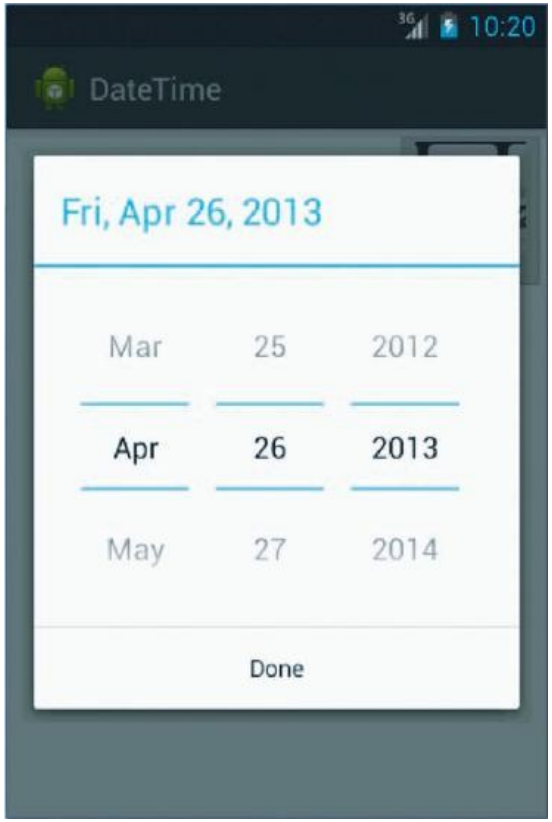
Action Bar

- Action bar is always present at the top of the Android screen
- Using it, the user can navigate or perform an action
- It provides user actions and navigation modes
- Commonly used actions can directly be taken from options menu and placed in the action bar
- Other options are available in ‘overflow menu’ button in the action bar



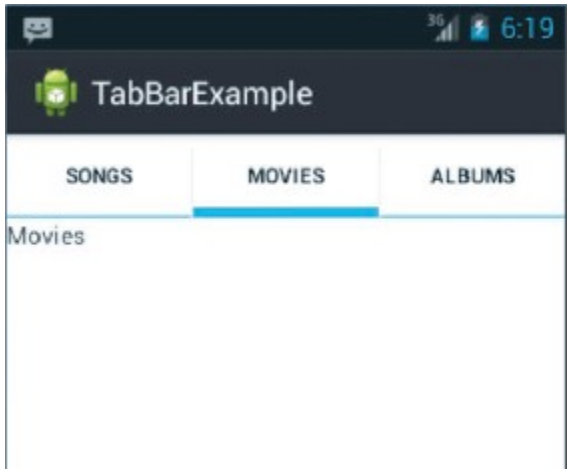
Dialogs

- Dialogs are prompt or alert displayed to the user to take a decision or to input any information
- The dialogs are also used to notify user when a task has been completed
- It does not fill the entire screen and usually appears when a user has to take a particular action before proceeding
- The different types of Dialogs are:
  - AlertDialog
  - Toast
  - TimePicker
  - DatePicker



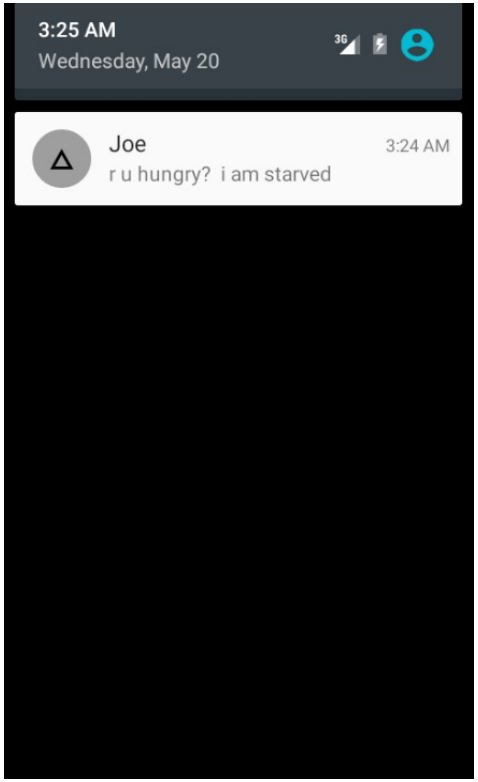
Additional Views

- Android contains some important Views such as:
  - **TabHost** – Is used to maintain tabs in an application such as in default contacts application. Each tab contains child layout to navigate within
  - **WebView** – Is used to load URL. WebView is like a browser which will display the Web content within
  - **SearchView** – Is used to provide search capability with a search provider within the application



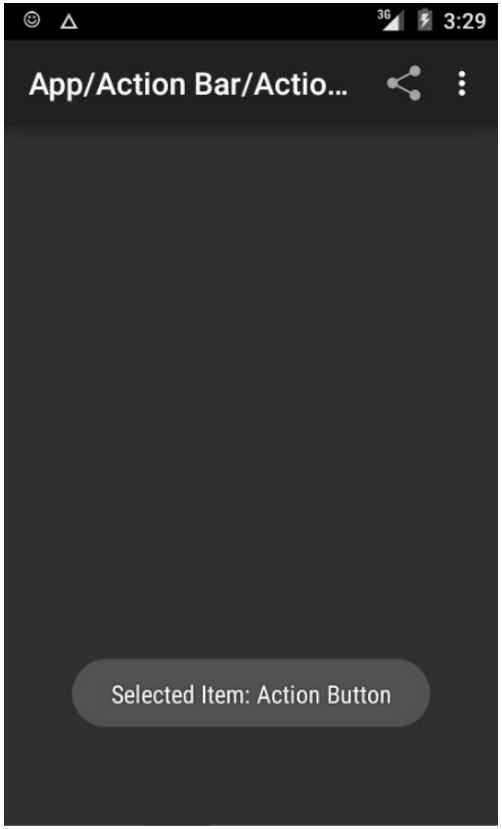
Notifications

- Notifications are used to notify or provide alerts to the user when a message or notification arrives
- Notification contains icon, title, body, and the notification arrival time
- A status bar notification displays an icon on the status bar along with a message
- When the notification is chosen, an Intent is sent by Android to launch the Activity
- The status bar notification can be initiated by an Activity or Service class
- The user can view the details by opening the 'Notifications drawer'



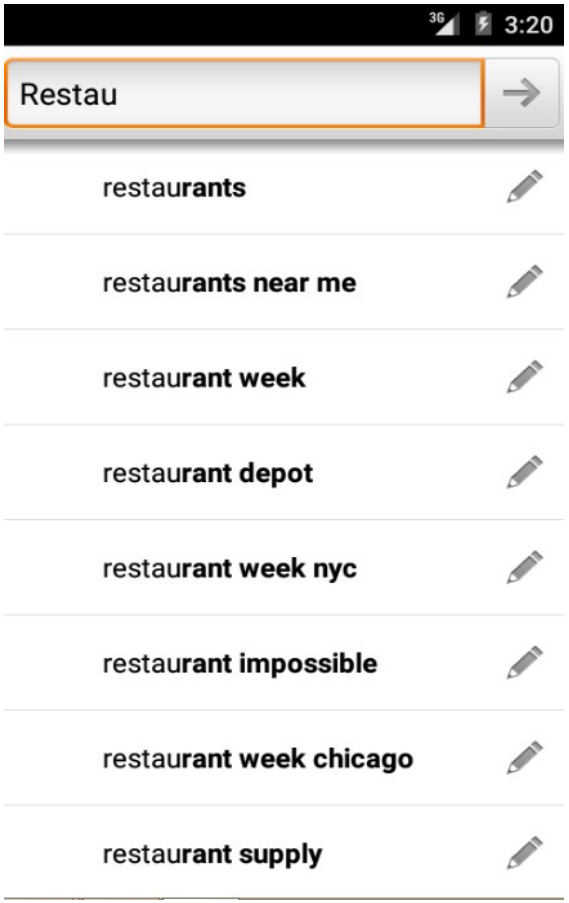
Toasts

- These are simple messages that provide feedback about a user's action in a popup window
- It is displayed in a small window and does not interfere with the user's ongoing activity



Search, Drag and Drop, and Accessibility

- **Search**
  - It is an important feature in Android and enables user to search for an item in the gadget or the Internet by entering a keyword

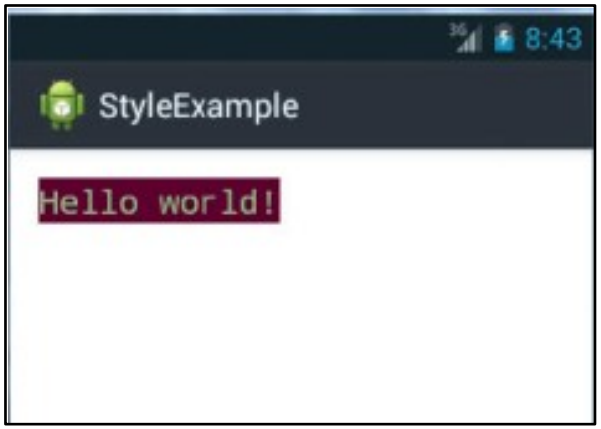


- **Drag and Drop**
  - This UI component allows user to move data from one View to another using a graphical drag and drop gesture
  - The framework consists of drag event class, drag listeners, helper methods, and classes

- **Accessibility**
  - Android has accessibility features to cater to the needs of users with special challenges like visual impairment or hearing difficulty
  - This includes 'text-to-speech' converter, audio prompting, gesture navigation, trackball, and directional pad navigation

Styles and Themes

- Presentation and formatting of an application can be improved using Styles and Themes
- Android does possess default styles and themes
- A style can be referred to as a collection of properties that specify the look and format for a View or window in UI
- All of the attributes related to style can be removed from the XML layout file and incorporated into a style definition file
- A theme is a style applied to an entire Activity or application, rather than an individual View



Style Properties and Themes Application

- **Style Properties**
  - Some style properties are not supported by any View element and can only be applied as a theme
  - These style properties apply to the entire window and not to any type of View
  - This kind of style properties do not belong to any View object
  - To find out theme-only style properties, the user can look at the 'R.attr' reference for attributes that begin with window
- **Application of Themes**
  - Adding the 'style' attribute to a View element in the XML for the layout. This is added to an individual View
  - Adding the android:theme attribute to the <activity> or <application> in the Android manifest file that sets the style to the entire application

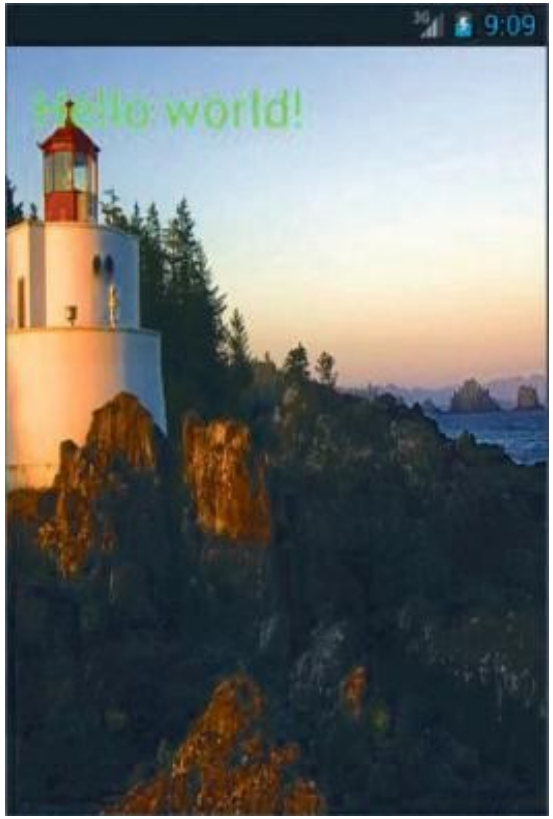
Styles and Themes Example 1-2

- The process of defining and applying a style is shown in the following Code Snippet:

```
<resources>
<style name="NewStyle"parent="@android:style/TextAppearance. Medium">
<item name="android:layout_width">fill_parent
</item>
<item name="android:layout_height">wrap_content
</item>
<item name="android:background">#551033
</item>
<item name="android:textColor">#86C67C
</item> </style>
</resources>
…
<TextView
        style="@style/NewStyle"
        android:text="@string/hello_world"/>
…
<application
        android:theme="@style/NewStyle" >
```

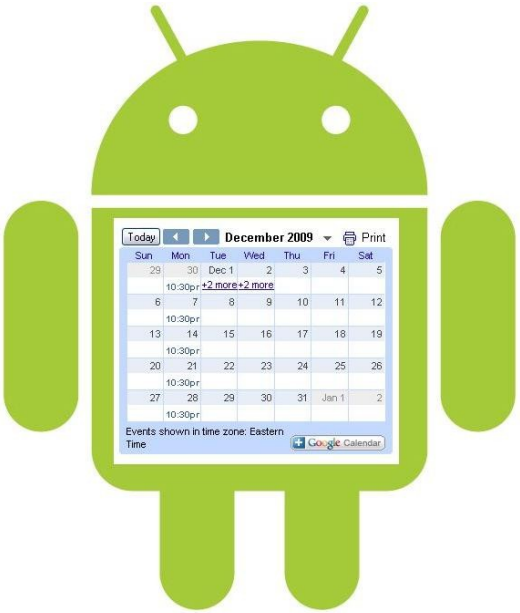
Styles and Themes Example 2-2

- Using the code, an application for demonstrating Styles and Themes is created as shown in the following figure:



Handling User Events

- An event refers to a response generated for an external action
- The Android framework maintains an Event Queue in which the events are arranged as they occur
- The events are removed on First-In-First-Out (FIFO) basis
- The two actions that need to be performed by the developer to inform the user about the user input events are as follows:
  - Defining an event listener and registering it with the view
  - Overriding an existing callback method for the view



Event Listeners

- Each of the event listeners consist of a single callback method which are invoked when the view to which the listener is registered generates an event due to user interaction
- Following table lists the different Event Listeners:

Event Listener	Method
View.OnClickListener	onClick ()
View.OnLongClickListener	onLongClick()
View.OnFocusChange Listener	onFocusChange()
View.OnKeyListener	onKey()
View.OnTouchListener	onTouch()
View.OnCreateContext MenuListener	onCreateContext Menu()

Event Listener Callback Methods

- **onKeyDown(int, KeyEvent)** - Invoked when a key has been pressed and was not handled by the Views within an Activity
- **onKeyUp(int, KeyEvent)** - Invoked when a key was released and was not handled by the Views within an Activity
- **onTrackballEvent(MotionEvent)** - Invoked when a trackball motion event occurs
- **onTouchEvent(MotionEvent)** - Invoked when a touch screen motion event occurs
- **onFocusChanged(boolean, int, Rect)** - Invoked when the View gains or loses focus

Event Listener Example 1-6

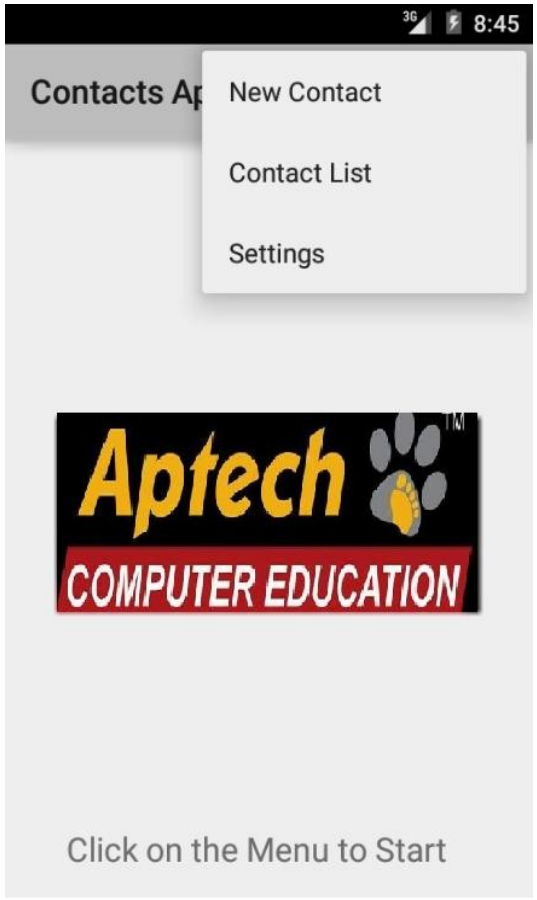
- The process for creating and registering and Event Listener is shown in the following Code Snippet:

```
private View.OnClickListener addButtonListener = new View.OnClickListener()
{
        public void onClick(View v) {

                …
        }
    };
```

Event Listener Example 2-6

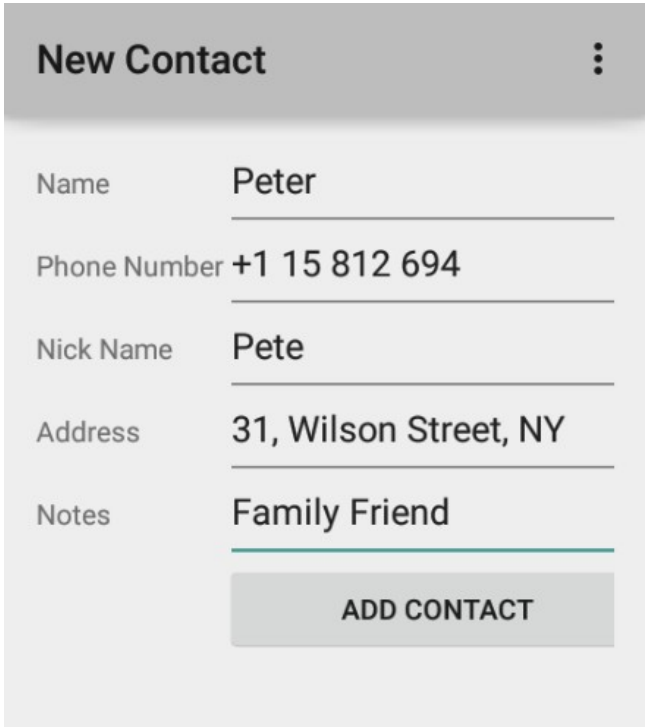
- Using the code, an application for demonstrating Event Listeners is created as shown in the following figure:

- Open the Menu and select New Contact as shown in the following figure:

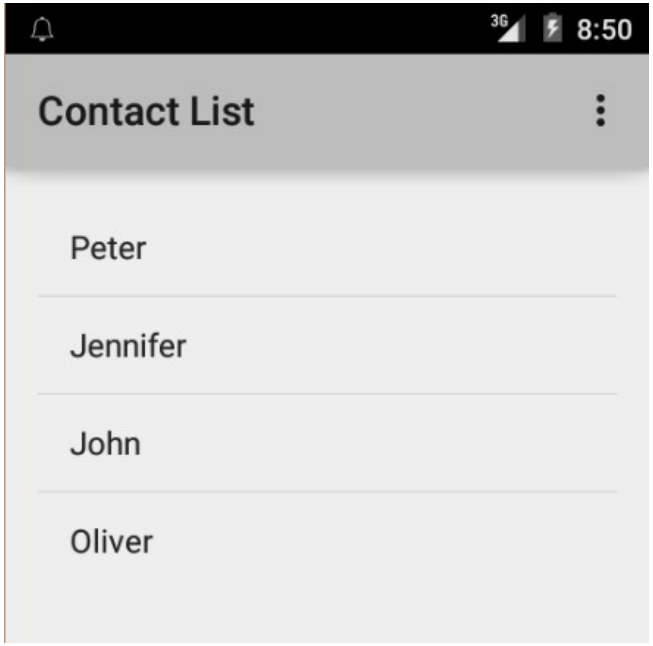


Event Listener Example 3-6

- The New Contact Screen is displayed. Enter the details as shown in the following figure:

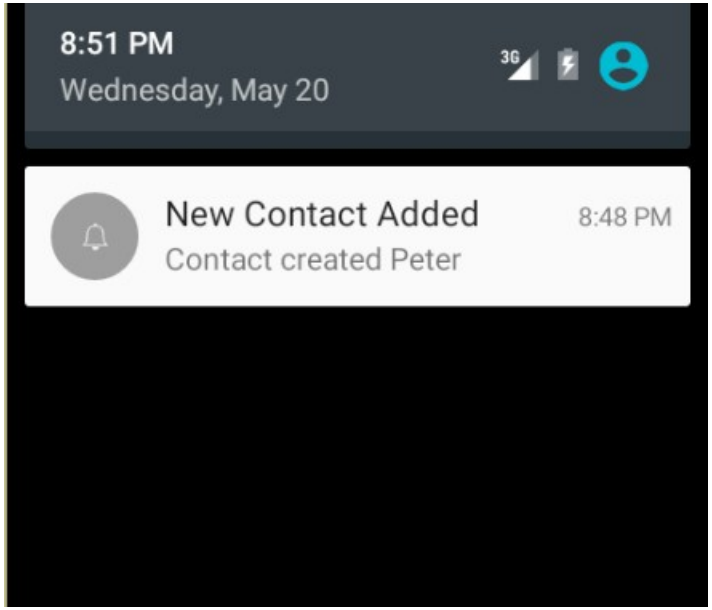


- Repeat these steps to add multiple contacts as shown in the following figure:

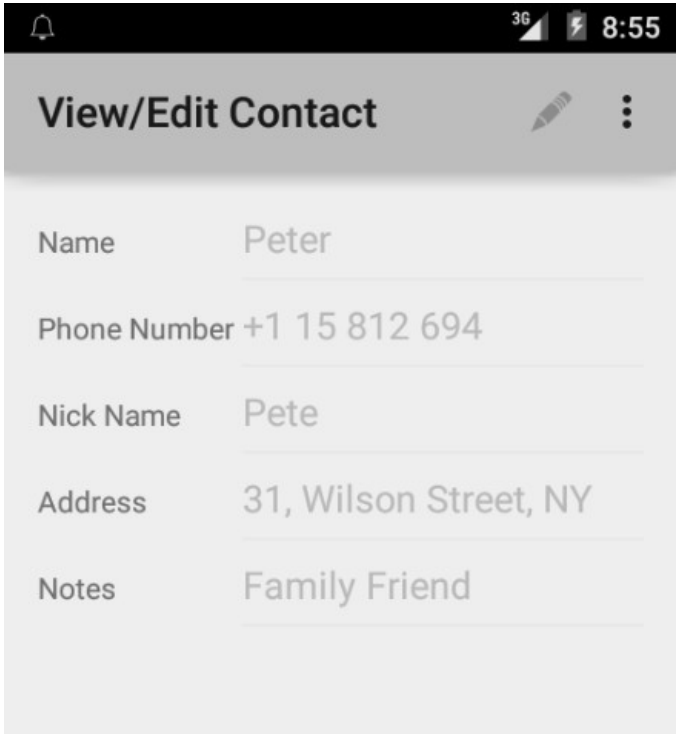


Event Listener Example 4-6

- A notification is displayed when a contact is added. This can be viewed within the notification drawer as shown in the following figure:

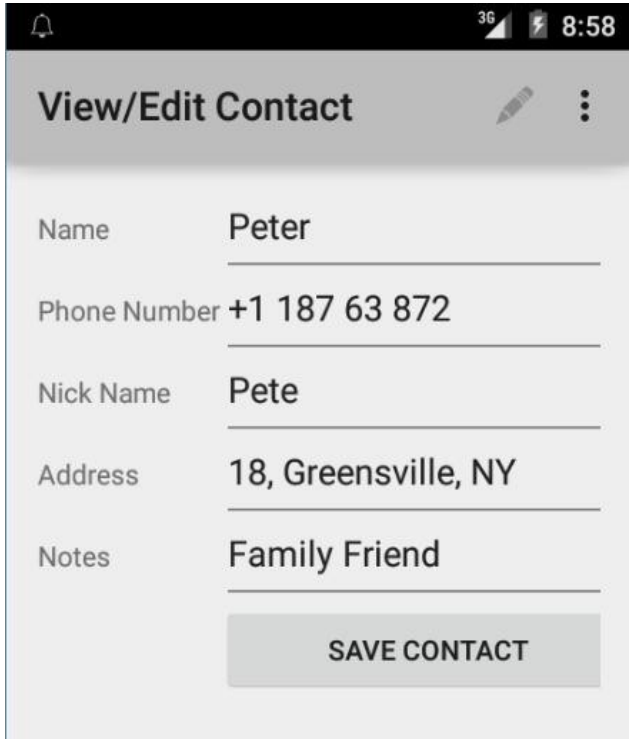


- Click any of the Contacts to view contact details as shown in the following figure:

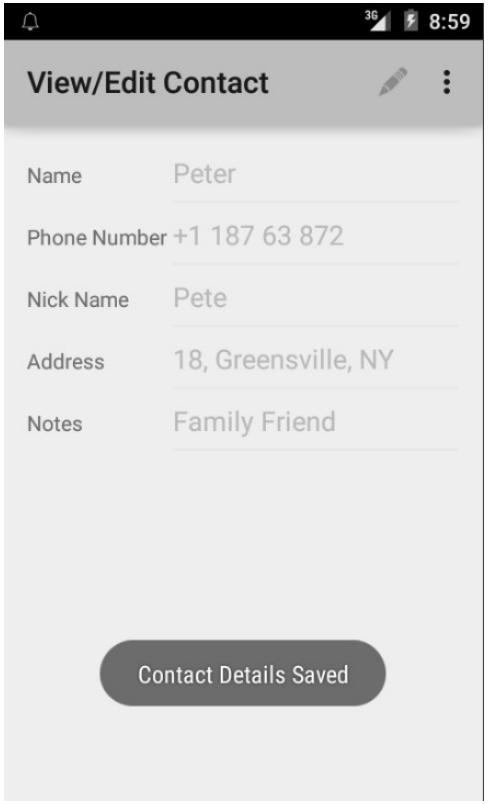


Event Listener Example 5-6

- Click the Edit Action button to enable editing the contact. Edit the details as shown in the following figure:

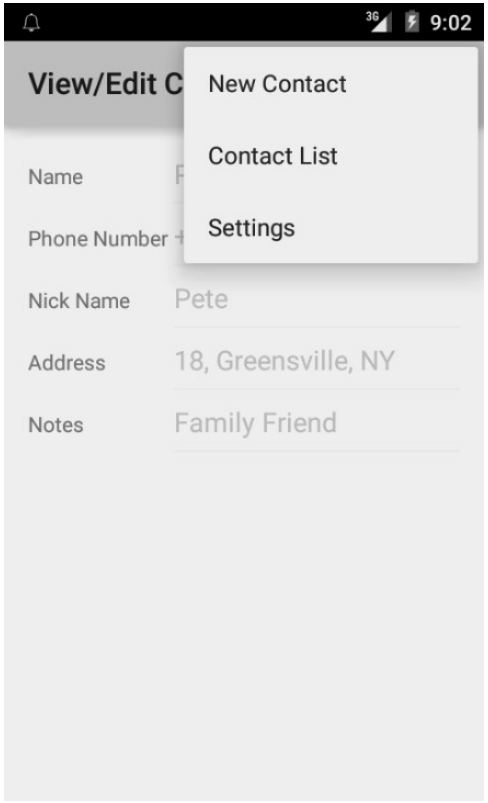


- Click Save Contact. The contact details are saved. A toast is displayed to notify this information as shown in the following figure:



Event Listener Example 6-6

- The user may return to the New Contact screen or the Contact List using the menu as shown in the following figure:



Summary

- All UI components in Android are built using View and ViewGroup objects. View draws something on the screen that the user can interact with
- Layout defines the visual structure of UI as in the case of an activity or an app widget. There are different types of layouts such as Linear layout, Relative layout, and Grid layout to name a few
- In Android, UI components are interactive controls in an app's UI that enables the user to perform a wide range of actions. Some common GUI components are buttons, picker, and so on
- A style is a collection of properties that specify the look and format for a View or window
- A theme is a style applied to an entire Activity or application, rather than an individual View
- In order to handle a particular event, it is necessary that the Event Listener implements the corresponding callback or Event Handler in response