Android Programming (R15)
UNIT - 5

CONTENTS

- **Part I: Creating Interface Menus and Action Bars:** Menus and Their Types, Creating Menus Through XML, Creating Menus Through Coding, Applying a Context Menu to a List View, Using the Action Bar, Replacing a Menu with the Action Bar, Creating a Tabbed Action Bar, Creating a Drop-Down List Action Bar
- **Part II: Using Databases:** Using the SQLiteOpenHelperclasss, Accessing Databases with the ADB, Creating a Data Entry Form
- **Part III: Communicating with SMS and Emails:** Understanding Broadcast Receivers, Using the Notification System, Sending SMS Messages with Java Code, Receiving SMS Messages, Sending Email, Working With Telephony Manager.

MENUS AND THEIR TYPES

- Display options in the form of menu items.
- Choosing a menu item results in the initiation of the desired task. Menus are therefore the preferred way of indicating the possible actions in an application.
- Android SDK supports three types of menus:
  1. Options,
  2. Submenu, and
  3. Context

Options Menu

- **Options Menu**—Also known as the Activity menu, this menu is displayed when a MENU button is clicked. In an Options Menu, the menu items are displayed in the form of text, check box, or radio buttons. It can display shortcut keys but does not display icons. In older Android API levels (Levels 10 and below), there were two types of Options Menus:
  - **Icon Menu**—The Icon Menu appears when the user presses the MENU button on the device. The Icon Menu shows the first six menu items of the menu in the form of large, finger-friendly buttons arranged in a grid at the bottom of the screen. The menu items in the Icon Menu can display icons as well as text. The Icon Menu does not display check boxes, radio buttons, or the shortcut keys for menu items. It is not mandatory to have icons for Icon Menu items; text will also do for the Icon Menu.
  - **Expanded Menu**—If the menu has more than six menu items, the first five items are displayed as an Icon Menu and the sixth option appears as a More button. Clicking the More button displays the Expanded Menu that shows the scrollable list of the rest of the menu items that could not be accommodated in the Icon Menu. The Expanded Menu can display menu items in the form of text, check boxes, or radio buttons. Also, it can display shortcut keys but does not display icons. Pressing the Back button from the Expanded Menu takes you back to the Activity the menu was launched from.

  The Icon Menu and Expanded Menu are not supported in Android API levels 11 and higher.

Submenu

- Submenu refers to the menu that displays more detailed or specific menu options when a menu item is selected.
- A submenu is displayed as a floating window showing all of its menu options.
- The name of the submenu is shown in the header bar, and each menu option is displayed with its full text, check box, radio button (if any), and shortcut key.
- Icons are not displayed in the submenu options.
- We cannot add a submenu to any menu option of a submenu as Android does not support nested submenus.
- Pressing the Back button closes the floating window without navigating back to the Expanded or Icon menus.

Context Menu

- The Context Menu is displayed when we tap-and-hold on the concerned View or when a user holds the middle D-pad button or presses the trackball.
- Context Menus support submenus, check boxes, radio buttons, and shortcuts, but we cannot display icons in a Context Menu.
- In the Context Menu's header bar, we can display a title and icon.

CREATING MENUS THROUGH XML

- Besides through Java code, in Android, menus can be created through Resources too.
- We can define a menu in the form of an XML file, which is then loaded by the Android SDK.
- As usual, Android generates resource IDs for each of the loaded menu items.
- The XML file for the menu has to be kept in a specific, designated folder that does not exist by default; we need to create it manually.

Creating an Options Menu

- An Options Menu is the menu displayed when the MENU button on the device is clicked.
- It shows menu items in the form of text, check boxes, and radio buttons. Also, the shortcut keys can be assigned to the menu items.
- An Options Menu is defined through an XML file in the res/menu folder of the application. That is, the menu items of the Options Menu are defined in that XML file.
- Each <item> node represents a menu item in the menu file.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/menu_settings"
      android:title="@string/menu_settings"
      android:orderInCategory="100"
      android:showAsAction="never" />
</menu>
```

Example

Listing 7.1. The Menu Items Defined in the File `activity_menu_app.xml`

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/create_datab"
        android:title="Create Database"
        android:icon="@drawable/ic_launcher" />
    <item android:id="@+id/insert_rows"
        android:title="Insert Rows"
        android:icon="@drawable/ic_launcher" />
    <item android:id="@+id/list_rows"
        android:title="List Rows" />
    <item android:id="@+id/search_row"
        android:title="Search"
        android:icon="@drawable/ic_launcher"/>
    <item android:id="@+id/delete_row"
        android:title="Delete" />
    <item android:id="@+id/update_row"
        android:title="Update"
        android:icon="@drawable/ic_launcher" />
</menu>
```

Listing 7.2. The Layout File `activity_menu_app.xml` on Adding the `TextView` Control

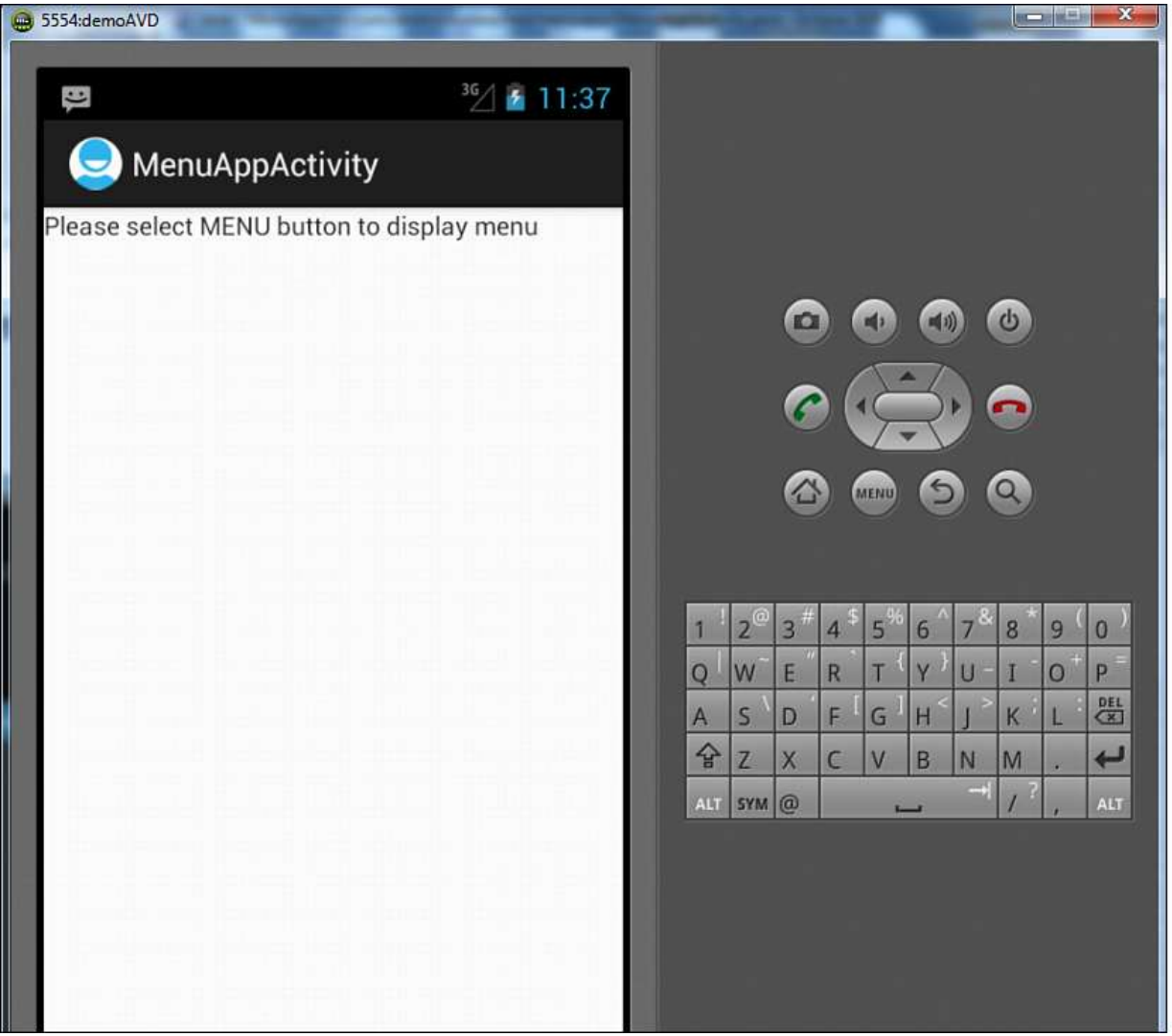
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/selectedopt" />
</LinearLayout>
```

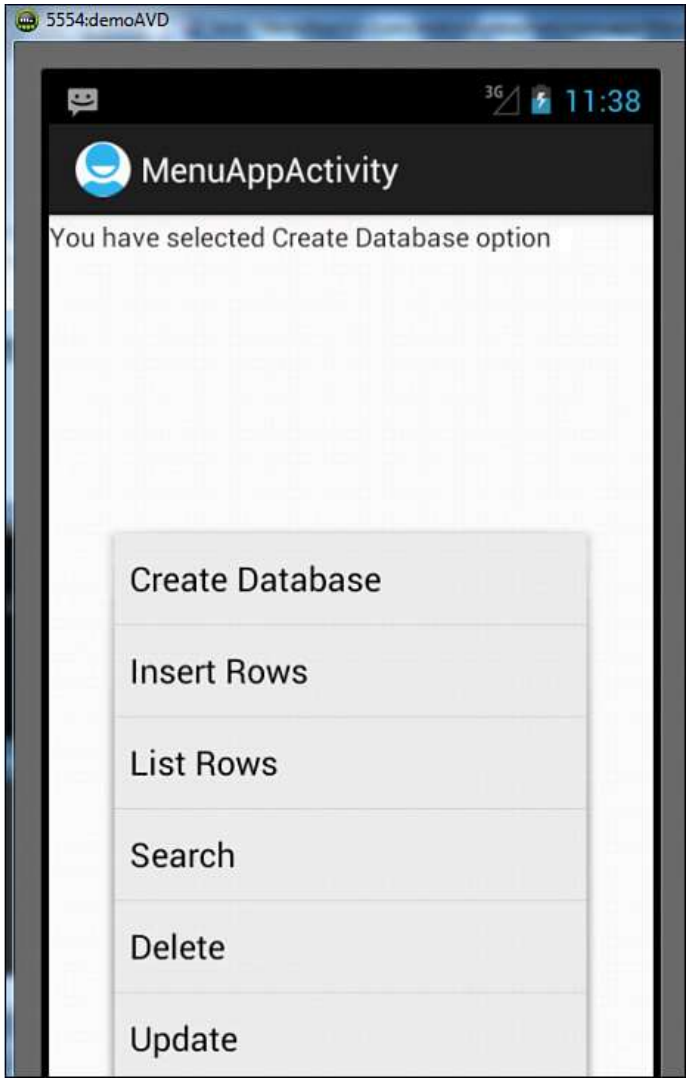
Listing 7.3. Code Written in the Java Activity File `MenuAppActivity.java`

```
package com.androidunleashed.menuapp;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.MenuInflater;
import android.widget.TextView;

public class MenuAppActivity extends Activity {
    private TextView selectedOpt;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_menu_app);
        selectedOpt=(TextView)findViewById(R.id.selectedopt);
        selectedOpt.setText("Please select MENU button to display menu");
    }
```

```
@Override
 public boolean onCreateOptionsMenu(Menu menu) {
  MenuInflater inflater = getMenuInflater();
  inflater.inflate(R.menu.activity_menu_app, menu);
  return true;
 }
 @Override
 public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
   case R.id.create_datab:
    selectedOpt.setText("You have selected Create Database option");
    break;
   case R.id.insert_rows:
    selectedOpt.setText("You have selected Insert Rows option");
    break;
   case R.id.list_rows:
    selectedOpt.setText("You have selected List Rows option");
    break;
   case R.id.search_row:
    selectedOpt.setText("You have selected Search Row option");
    break;
   case R.id.delete_row:
    selectedOpt.setText("You have selected Delete Row option");
    break;
   case R.id.update_row:
    selectedOpt.setText("You have selected Update Row option");
    break;
  }
  return true;
 } }
```

Figure 7.1. A `TextView` directing the user to select the `MENU` button to invoke a menu (left), and the `TextView` displays the option selected from the `Options menu` (right).

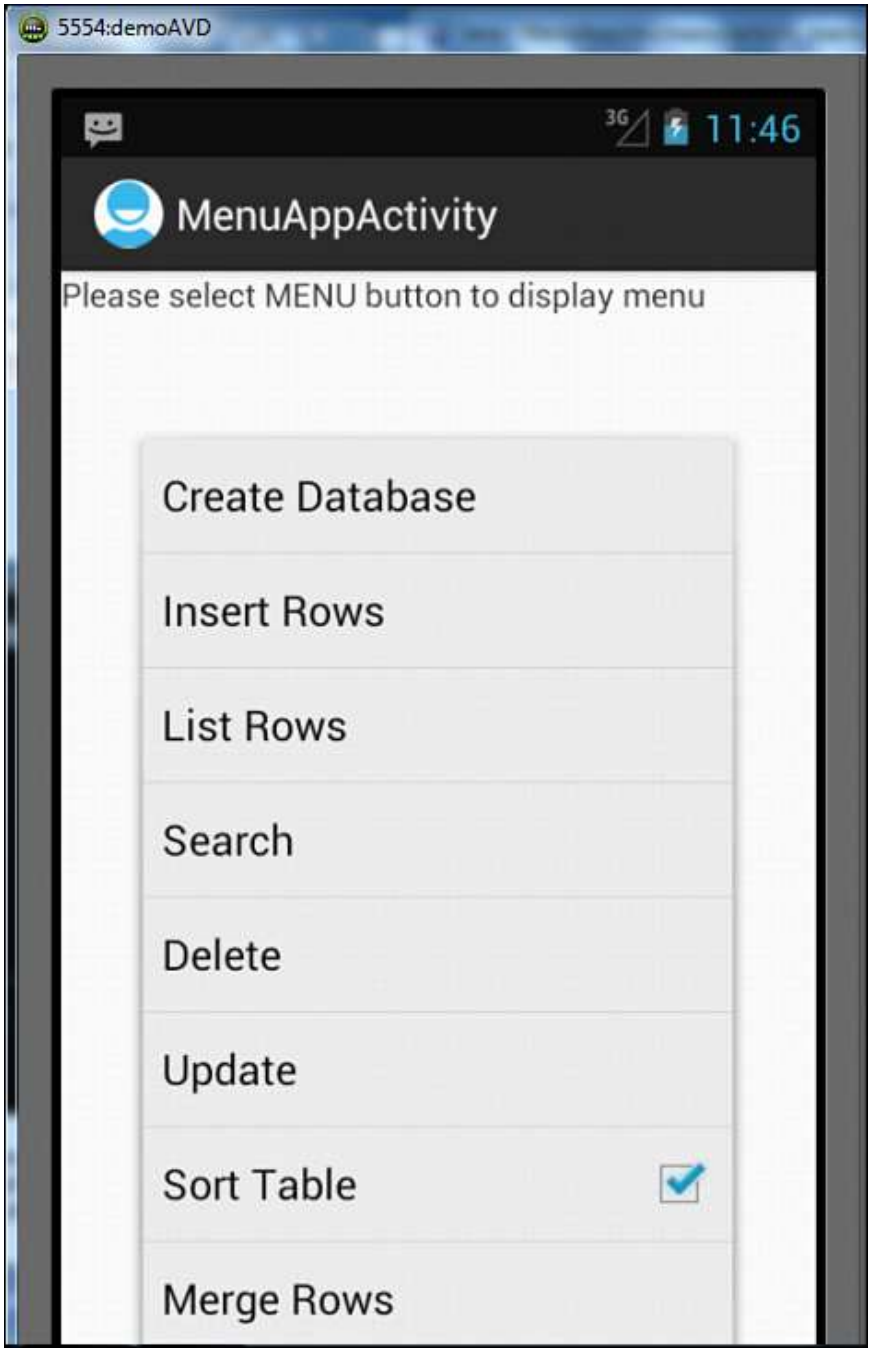




Defining Check Boxes and Shortcuts

- to define checkable menu items and shortcuts for them.
- We add more menu items to the menu in our application MenuApp.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/create_datab"
    android:title="Create Database"
    android:icon="@drawable/ic_launcher" />
  <item android:id="@+id/insert_rows"
    android:title="Insert Rows"
    android:icon="@drawable/ic_launcher" />
  <item android:id="@+id/list_rows"
    android:title="List Rows" />
  <item android:id="@+id/search_row"
    android:title="Search"
    android:icon="@drawable/ic_launcher"/>
  <item android:id="@+id/delete_row"
    android:title="Delete" />
  <item android:id="@+id/update_row"
    android:title="Update"
    android:icon="@drawable/ic_launcher" />
  <item android:id="@+id/sort_rows"
    android:title="Sort Table"
    android:checkable="true"
    android:checked="true"/>
  <item android:id="@+id/merge_row"
    android:title="Merge Rows"
    android:alphabeticShortcut="m"
    android:numericShortcut="4"/>
</menu>
```

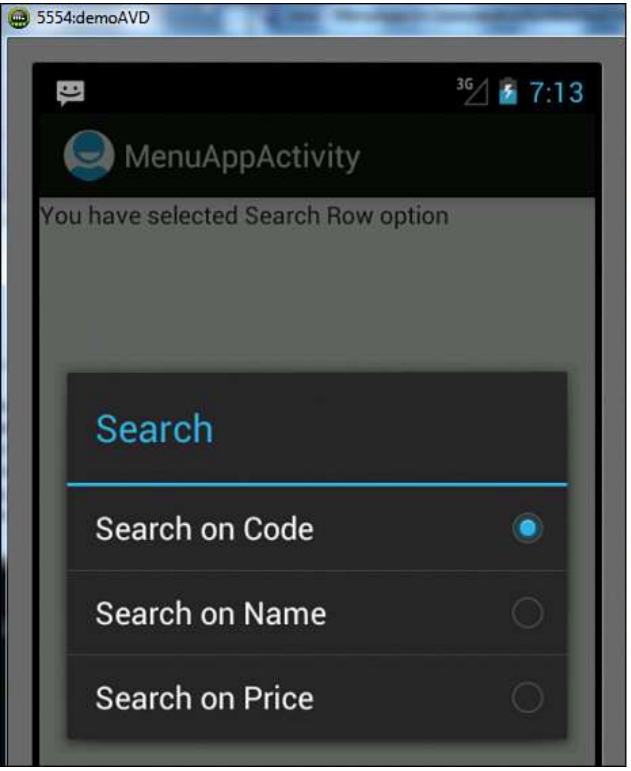

5554:demoAVD
11:46
MenuAppActivity
Please select MENU button to display menu
Create Database
Insert Rows
List Rows
Search
Delete
Update
Sort Table
Merge Rows

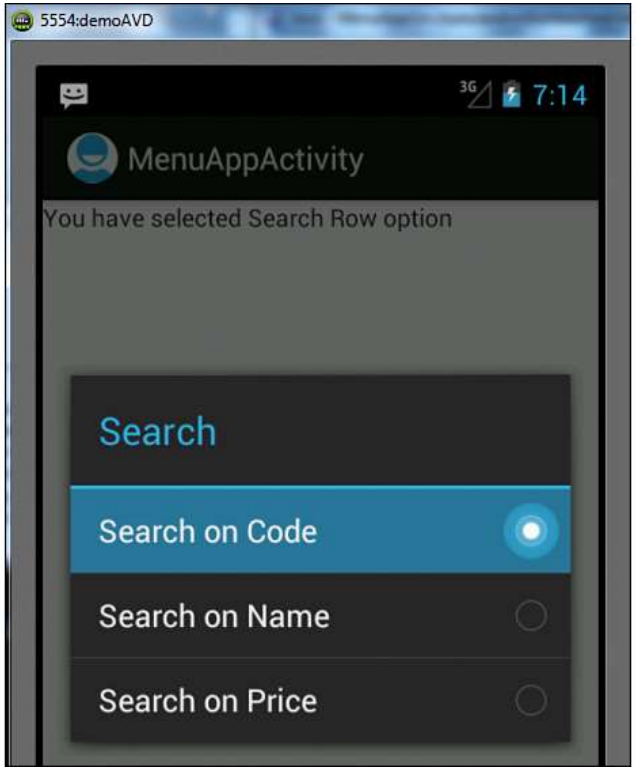
Adding Submenus

- A submenu is a collection of menu items invoked when a regular menu item is selected.
- Usually, a submenu represents more detailed options for the selected menu item. The submenu can be associated with any menu item.
- A submenu is created by enclosing <item>nodes within the <menu> node, where <item> nodes represent the submenu options and the <menu> node acts as the root node of the submenu.
- To associate a submenu with any menu item, just put the <menu> node of the submenu along with its nested <item> nodes inside the <item> node of the menu item to which we want to assign the submenu.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/create_datab"
    android:title="Create Database"
    android:icon="@drawable/ic_launcher" />
  <item android:id="@+id/insert_rows"
    android:title="Insert Rows"
    android:icon="@drawable/ic_launcher" />
  <item android:id="@+id/list_rows"
    android:title="List Rows" />
  <item android:id="@+id/search_row"
    android:title="Search"
    android:icon="@drawable/ic_launcher">
    <menu>
      <group android:checkableBehavior="single">
        <item android:id="@+id/search_code"
          android:title="Search on Code"
          android:checked="true" />
        <item android:id="@+id/search_name"
          android:title="Search on Name"
          android:alphabeticShortcut="n"
          android:numericShortcut="6" />
        <item android:id="@+id/search_price"
          android:title="Search on Price" />
      </group>
    </menu>
  </item>
  <item android:id="@+id/delete_row"
    android:title="Delete" />

</menu>
```

```
public boolean onOptionsItemSelected(MenuItem item) {
   switch (item.getItemId()) {
    case R.id.create_datab:
     selectedOpt.setText("You have selected Create Database option");
     break;
    case R.id.insert_rows:
     selectedOpt.setText("You have selected Insert Rows option");
     break;
    case R.id.list_rows:
     selectedOpt.setText("You have selected List Rows option");
     break;
    case R.id.search_row:
     selectedOpt.setText("You have selected Search Row option");
     break;
    case R.id.delete_row:
     selectedOpt.setText("You have selected Delete Row option");
     break;
    case R.id.update_row:
     selectedOpt.setText("You have selected Update Row option");
     break;
    case R.id.sort_rows:
     selectedOpt.setText("You have selected Sort Table option");
     item.setChecked(!item.isChecked());
     break;
    case R.id.merge_row:
     selectedOpt.setText("You have selected Merge Rows option");
     break;
    case R.id.search_code:
     selectedOpt.setText("You have selected Search on Code option");
     break;
    case R.id.search_name:
     selectedOpt.setText("You have selected Search on Name option");
     break;
    case R.id.search_price:
     selectedOpt.setText("You have selected Search on Price option");
     break;
   }
   return true;
  }
}
```


5554:demoAVD
7:13
MenuAppActivity
You have selected Search Row option
Search
Search on Code
Search on Name
Search on Price


5554:demoAVD
7:14
MenuAppActivity
You have selected Search Row option
Search
Search on Code
Search on Name
Search on Price


5554:demoAVD
7:15
MenuAppActivity
You have selected Search on Code option

Creating a Context Menu

- A context menu appears as a floating window and is displayed when the user taps and holds on a widget, holds the middle D-pad button, or presses the trackball.
- The difference between an options menu and a context menu is that the options menu is invoked when the MENU button is pressed. A context menu appears when we press and hold a View.
- An Activity can have multiple views, and each view can have its own context menu.
- An Activity can have only a single options menu but many context menus.
- Context menus are removed from memory when closed.

Let’s define two context menus

Listing 7.7. Code Written into the Context Menu mycontext_menu1.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <group android:checkableBehavior="all">
    <item android:id="@+id/cut_item"
      android:title="Cut" />
    <item android:id="@+id/copy_item"
      android:title="Copy" />
  </group>
  <item android:id="@+id/find_item"
    android:title="Find">
    <menu>
      <item android:id="@+id/find_next"
        android:title="Find Next" />
    </menu>
  </item>
</menu>
```

Listing 7.8. Code Written into the Context Menu `mycontext_menu2.xml file`

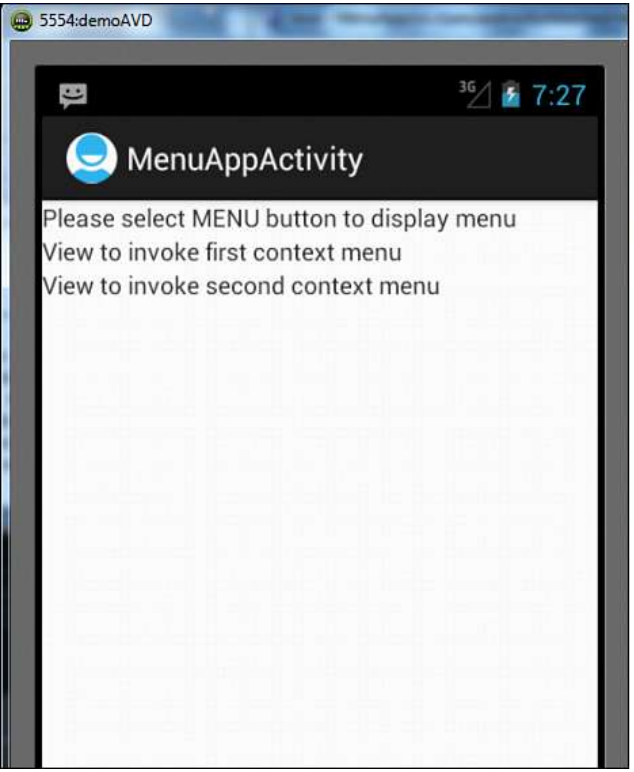
```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/open_item"
    android:title="Open"
    android:alphabeticShortcut="o"
    android:numericShortcut="5" />
  <item android:id="@+id/close_item"
    android:title="Close"
    android:checkable="true" />
</menu>
```

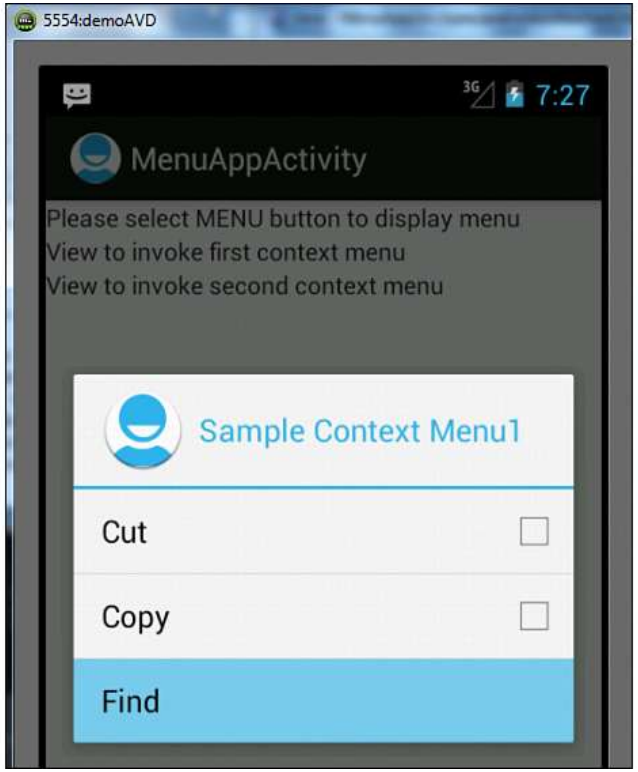
Listing 7.9. The Layout File `activity_menu_app.xml` After Adding Two `TextView` Controls

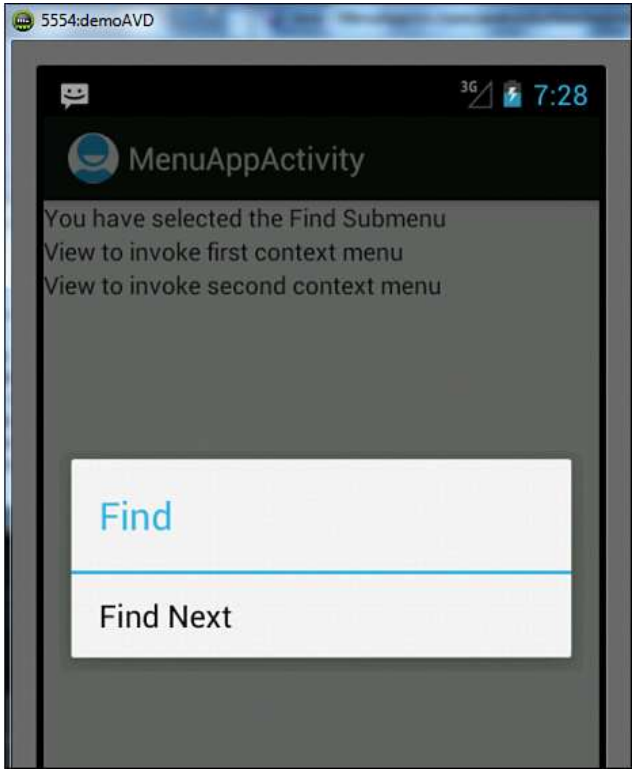
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >
  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/selectedopt" />
  <TextView
    android:text="View to invoke first context menu"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/contxt1_view" />
  <TextView
    android:text="View to invoke second context menu"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/contxt2_view"/>
</LinearLayout>
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);
   selectedOpt=(TextView)findViewById(R.id.selectedopt);
   selectedOpt.setText("Please select MENU button to display menu");
   TextView contxt1View=(TextView)findViewById(R.id.contxt1_view);
   TextView contxt2View=(TextView)findViewById(R.id.contxt2_view);
   registerForContextMenu(contxt1View);
   registerForContextMenu(contxt2View);
}
```

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenu-
Info menuInfo) {
   super.onCreateContextMenu(menu, v, menuInfo);
   if (v.getId()==R.id.contxt1_view) {
      MenuInflater inflater = getMenuInflater();
      inflater.inflate(R.menu.mycontext_menu1, menu);
      menu.setHeaderTitle("Sample Context Menu1");
      menu.setHeaderIcon(R.drawable.ic_launcher);
   }
   if (v.getId()==R.id.contxt2_view) {
      MenuInflater inflater = getMenuInflater();
      inflater.inflate(R.menu.mycontext_menu2, menu);
      menu.setHeaderTitle("Sample Context Menu2");
      menu.setHeaderIcon(R.drawable.ic_launcher);
   }
}
```


5554:demoAVD
7:27
MenuAppActivity
Please select MENU button to display menu
View to invoke first context menu
View to invoke second context menu


5554:demoAVD
7:27
MenuAppActivity
Please select MENU button to display menu
View to invoke first context menu
View to invoke second context menu
Sample Context Menu1
Cut
Copy
Find


5554:demoAVD
7:28
MenuAppActivity
You have selected the Find Submenu
View to invoke first context menu
View to invoke second context menu
Find
Find Next

CREATING MENUS THROUGH CODING

- We can create the menus through Java code too.

Defining Options Menus

- To define an Options or Activity menu, we override the onCreateOptionsMenu method in the Java Activity file.
- The method receives a menu object as a parameter, which is used to add menu items to it.
- To add menu items to the menu object, the add() method is used.

  add(Group ID, Menu Item ID, Sort order ID, Menu text)
- **Group ID—**An integer used for grouping a collection of menu items.
- **Menu Item ID—**A unique identifier assigned to each menu item to identify it.
- **Sort order ID—**Defines the order in which the menu items are to be displayed.
- **Menu text**—Text to appear in the menu item.
- use Menu.NONE in place of them if we don't want to specify any of them.

The following code adds a single menu item, `Create Database`, to the `Options Menu`:

```
private static final int CREATE_DATAB = Menu.FIRST;
menu.add(0,CREATE_DATAB,0,"Create Database").setIcon(R.drawable.ic_launcher);
```

We can avoid the setIcon() method if we want only a menu item text without an icon.

Assigning Icons

We can assign icons to the menu items in the `Icon Menu` using the `setIcon()` method.

syntax:

```
menuItem.setIcon(R.drawable.icon_filename);
```

Creating Submenus

- Submenus appear as single floating windows displaying all of their menu items.
- A submenu is attached to a regular menu item that, when selected, invokes the submenu, hence displaying all the menu items in it.
- A submenu is added through the addSubMenu() method.
- Android does not support nested submenus.

```
private static final int SEARCH_ROW = Menu.FIRST + 3;

SubMenu searchSub = menu.addSubMenu(0, SEARCH_ROW, 3, "Search");

searchSub.setHeaderIcon(R.drawable.ic_launcher);

searchSub.setIcon(R.drawable.ic_launcher);

searchSub.add(1, SEARCH_CODE, Menu.NONE, "Search on Code");
```

Using Check Boxes/Radio Buttons in Menus

- Android supports check boxes and radio buttons in menu items. To set a menu item as a check box, we use the setCheckable() method.

syntax:

setCheckable(boolean);

Android Programming (R15)
UNIT - 5

CONTENTS

- **Part I: Creating Interface Menus and Action Bars:** Menus and Their Types, Creating Menus Through XML, Creating Menus Through Coding, Applying a Context Menu to a List View, Using the Action Bar, Replacing a Menu with the Action Bar, Creating a Tabbed Action Bar, Creating a Drop-Down List Action Bar
- **Part II: Using Databases:** Using the SQLiteOpenHelperclasss, Accessing Databases with the ADB, Creating a Data Entry Form
- **Part III: Communicating with SMS and Emails:** Understanding Broadcast Receivers, Using the Notification System, Sending SMS Messages with Java Code, Receiving SMS Messages, Sending Email, Working With Telephony Manager.

Using Databases

- User-entered data needs to be saved somewhere so that it can be accessed at a later time.
- Android offers the SQLite relational database library for persisting data.
- It's an open-source, lightweight, and powerful database available in the form of a C library.
- It uses SQL (Structured Query Language) for storing and retrieving information and performing database maintenance.
- SQLite is extremely reliable and is popularly used in devices with restricted computing power.
- Through SQLite, we can create individual databases for each application and store and manage application-related data.

SQLiteOpenHelper class

- Android databases are stored in the /data/data/<package_name>/databases folder on devices or emulators.
- To create an Android application that stores, accesses, and manipulates user information in a SQLite-relational database, we use the SQLiteOpenHelper class.
- The SQLiteOpenHelperclass is an abstract class that provides a way to get read or write access to a database.
- The SQLiteOpenHelper class thus makes our task of manipulating the data stored in the database easy.
- The SQLiteOpenHelper class is an abstract class used to create, open, and upgrade databases.
- It provides several methods including getWritableDatabase(), getReadableDatabase(), and close() that returns an SQLiteDatabase object that lets the calling code do the reading, writing, and closing of the database, respectively.

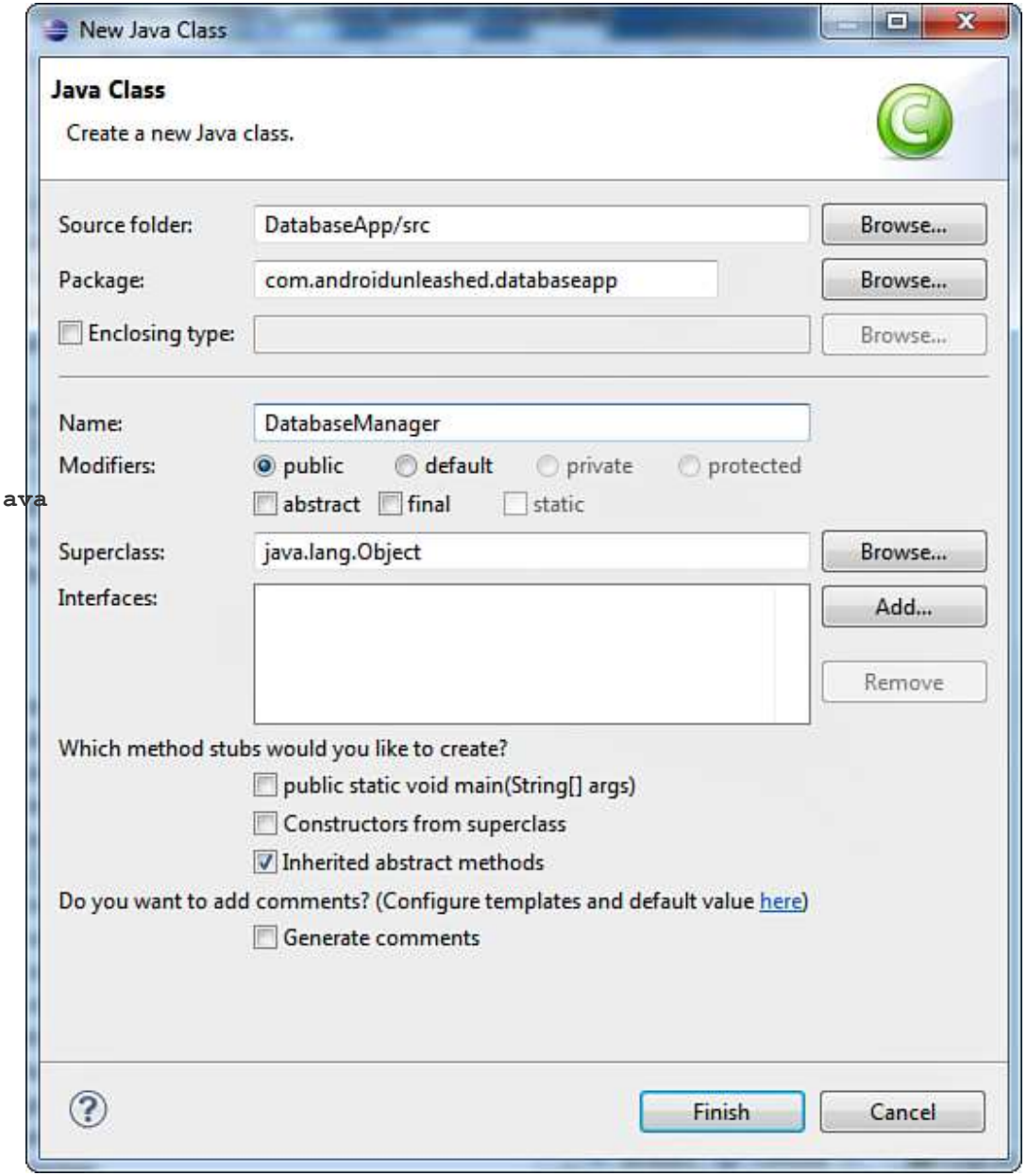
SQLiteOpenHelper class

- The SQLiteDatabase class also provides methods including insert() and query(), which are used to insert rows in the database table and execute different SQLite queries on the database table.
- The query() method accepts several parameters such as the database table to query, columns, and criteria, and returns a Cursor representing the rows that satisfy the supplied criteria.
- The Cursor class provides several methods to traverse the result set and move to the desired row and column position in the result set.

Example

```
package com.androidunleashed.databaseapp;

public class DatabaseManager {

}
```

Figure 8.1. Creating a new Java file called `DatabaseManager.java`



Write code in DatabaseManager.java to perform the following tasks:

- Inherit from the SQLiteOpenHelper class to access and take advantage of the methods defined in it.
- Open and return the writable and readable database file instance to perform writing and reading operations on the database, respectively.
- Create a database table, products, consisting of three columns: code, product_name, and price.
- Add rows to the products table. The information for the new product is supplied by the activity file DatabaseAppActivity.java.
- Fetch rows from the products table and return them to the activity file DatabaseAppActivity.java for displaying on the screen.

Listing 8.1. Code Written into the Java Class `DatabaseManager.java`

```
package com.androidunleashed.databaseapp;

import android.database.sqlite.SQLiteDatabase;
import android.content.Context;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

public class DatabaseManager {
    public static final String DB_NAME = "shopping";
    public static final String DB_TABLE = "products";
    public static final int DB_VERSION = 1;
    private static final String CREATE_TABLE = "CREATE TABLE " + DB_TABLE + " (code
        INTEGER PRIMARY KEY, product_name TEXT, price FLOAT);";
    private SQLHelper helper;
    private SQLiteDatabase db;
    private Context context;

    public DatabaseManager(Context c){
        this.context = c;
        helper=new SQLHelper(c);
        this.db = helper.getWritableDatabase();
    }
       public DatabaseManager openReadable() throws android.database.SQLException {
        helper=new SQLHelper(context);
        db = helper.getReadableDatabase();
        return this;
    }
```

```
public void close(){
        helper.close();
    }

    public void addRow(Integer c, String n, Float p){
        ContentValues newProduct = new ContentValues();
        newProduct.put("code", c);
        newProduct.put("product_name", n);
        newProduct.put("price", p);
        try{db.insertOrThrow(DB_TABLE, null, newProduct);}
        catch(Exception e)
        {
            Log.e("Error in inserting rows ", e.toString());
            e.printStackTrace();
        }
    }
    public String retrieveRows(){
        String[] columns = new String[]{"code", "product_name", "price"};
        Cursor cursor = db.query(DB_TABLE, columns, null, null, null, null, null);
        String tablerows = "";
        cursor.moveToFirst();
        while (cursor.isAfterLast() == false) {
          tablerows = tablerows + cursor.getInt(0) + ", "+cursor.getString(1)+", "
+ cursor.getFloat(2)+ "\n";
            cursor.moveToNext();
        }
        if (cursor != null && !cursor.isClosed()) {
            cursor.close();
        }
        return tablerows;
    }
```

```
public class SQLHelper extends SQLiteOpenHelper {
   public SQLHelper(Context c){
     super(c, DB_NAME, null, DB_VERSION);
   }

   @Override
   public void onCreate(SQLiteDatabase db) {
     db.execSQL(CREATE_TABLE);
   }

   @Override
   public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
     Log.w("Products table","Upgrading database i.e. dropping table and rec-
reating it");
     db.execSQL("DROP TABLE IF EXISTS " + DB_TABLE);
     onCreate(db);
   }
  }
}
```

Fetching the Desired Rows from Tables

- To fetch the desired rows from a database table, we execute queries containing a criterion against the given database.
- The queries are executed through the query() method provided by the SQLiteDatabase class.

```
db.query(bool_uniq, db_table, array_columns, where_clause, select_arg, group_by, having_clause, order);
```

The returned rows from the query() method are in the form of Cursor objects.

A Cursorobject does not create a separate result set of the extracted rows, but points at the result set that is part of a database table.

moveToFirst() - the cursor is moved to the first row

moveTo-Next() - fetched one by one

The Cursor is closed when all the rows pointed at in its result set are traversed.

Table 8.1. Parameters Used in the `query()` Method

Parameter	Usage
`bool_uniq`	Optional Boolean value to determine whether the result set should contain only unique values. A `True` value confirms the unique values in the result set.
`db_table`	The database table to be queried.
`array_columns`	A string array containing the list of columns we want in the result set.
`where_clause`	Defines the criteria for the rows to be returned. For specifying values in the clause, ? wildcard(s) are used, which are then replaced by the values stored in the `select_arg` parameter.
`select_arg`	An array of strings that provide values for the ? wildcards used in the `where_clause` parameter.
`group_by`	Defines the condition for grouping the returned result set.
`having_clause`	Defines the conditions to be applied to the groups defined in the `group_by` clause.
`order`	Defines the order of the returned rows.

Using Cursors

- Cursors are pointers pointing at the result set of the underlying data and can be set to traverse the rows and retrieve column data of the result set.
- The Cursor class provides several methods that can be used to set the pointer at the desired row and column position of the result set.

Table 8.2. The `Cursor` Class Methods Used to Set the Pointer at the Desired Location

Method	Usage
`moveToFirst`	Moves the cursor to the first row in the result set
`moveToNext`	Moves the cursor to the next row
`moveToPrevious`	Moves the cursor to the previous row
`getCount`	Returns the count of the rows in the result set
`getColumn-IndexOrThrow`	Returns the index for the column with the specified name. It throws an exception if no column exists with that name
`getColumnName`	Returns the column name with the specified column index
`getColumnNames`	Returns a string array that contains all the column names in the current cursor
`moveToPosition`	Moves the cursor to the specified row
`getPosition`	Returns the current cursor position

ACCESSING DATABASES WITH THE ADB

- Recall from Chapter 1
- the ADB (Android Debug Bridge) is a client/server program that is part of the Android SDK and is used to communicate with, control, and manage Android devices and emulators.
- We can perform several tasks via adb commands, including viewing applications, deleting them, installing new applications, and executing shell commands.

```
# cd /data/data/com.androidunleashed.databaseapp/databases
cd /data/data/com.androidunleashed.databaseapp/databases

# ls
ls
shopping

# sqlite3 shopping
sqlite3 shopping
SQLite version 3.5.9
Enter ".help" for instructions

sqlite> .tables
.tables
android_metadata  products
```

The .schema command displays the structure (field names, their types, width, and so on) of all the tables that exist in the currently active database:

```
sqlite> .schema
.schema
CREATE TABLE android_metadata (locale TEXT);
CREATE TABLE products (code INTEGER PRIMARY KEY, product_name TEXT, price FLOAT);
sqlite> select * from products;
select * from products;
101|Camera|15.0
102|Laptop|1005.98999023438
```

```
sqlite> delete from products where code=102;
delete from products where code=102;

sqlite> update products set product_name='Handy Cam' where code=101;
update products set product_name='Handy Cam' where code=101;
```

sqlite>.exit

Accessing the Database Through Menus

- To make our DatabaseApp application accessible through menus, we need to define a few menu items in the menu file activity_database_app available in the res/menu folder.
- In the menu file, we define a menu consisting of two menu items called Insert Rows and List Rows.
- These are used to initiate the tasks of inserting and accessing rows from the products table.

Listing 8.4. Code Written into the Menu File activity_database_app.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/insert_rows"
    android:title="Insert Rows"
    android:icon="@drawable/ic_launcher" />
  <item android:id="@+id/list_rows"
    android:title="List Rows"
    android:icon="@drawable/ic_launcher" />
</menu>
```

Listing 8.5. Code in the `DatabaseAppActivity.java` Activity File

```
package com.androidunleashed.databaseapp;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;

public class DatabaseAppActivity extends Activity {
    private DatabaseManager mydManager;
    private TextView response;
    private TextView productRec;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_database_app);
        response=(TextView)findViewById(R.id.response);
        productRec=(TextView)findViewById(R.id.prodrec);
        response.setText("Press MENU button to display menu");
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.activity_database_app, menu);
        return true;
    }
```

```
@Override
   public boolean onOptionsItemSelected(MenuItem item) {
      switch (item.getItemId()) {
         case R.id.insert_rows: insertRec();
            break;
         case R.id.list_rows: showRec();
            break;
      }
      return true;
   }

   public boolean insertRec(){
      mydManager = new DatabaseManager(this);
      mydManager.addRow(101, "Camera", Float.parseFloat("15"));
      mydManager.addRow(102, "Laptop",  Float.parseFloat("1005.99"));
      response.setText("The rows in the products table are inserted");
      productRec.setText("");
      mydManager.close();
      return true;
   }
   public boolean showRec(){
      mydManager = new DatabaseManager(this);
      mydManager.openReadable();
      String tableContent = mydManager.retrieveRows();
      response.setText("The rows in the products table are:");
      productRec.setText(tableContent);
      mydManager.close();
      return true;
   }
}
```


5554:demoAVD
12:25
DatabaseAppActivity
Press MENU button to display menu

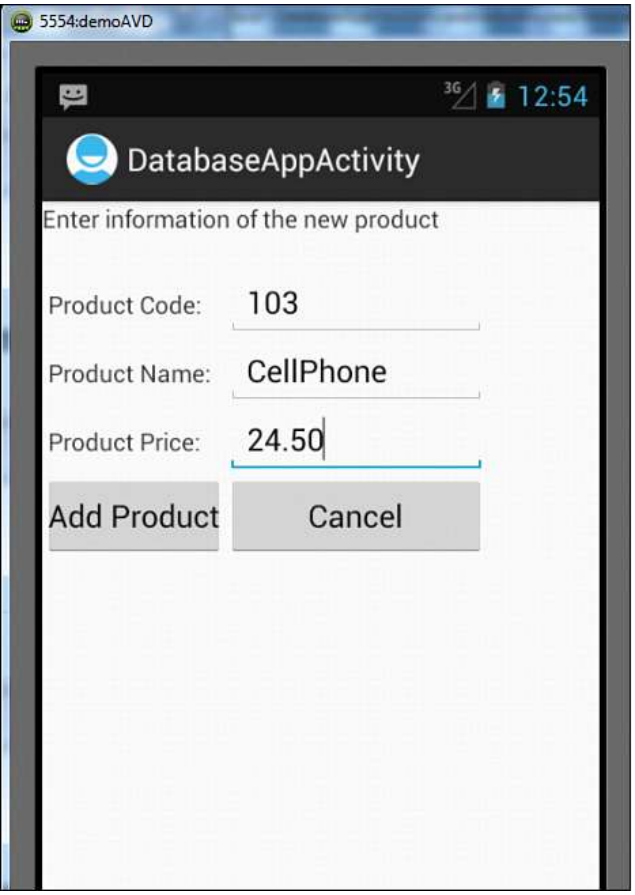

5554:demoAVD
12:26
DatabaseAppActivity
The rows in the products table are inserted
Insert Rows
List Rows

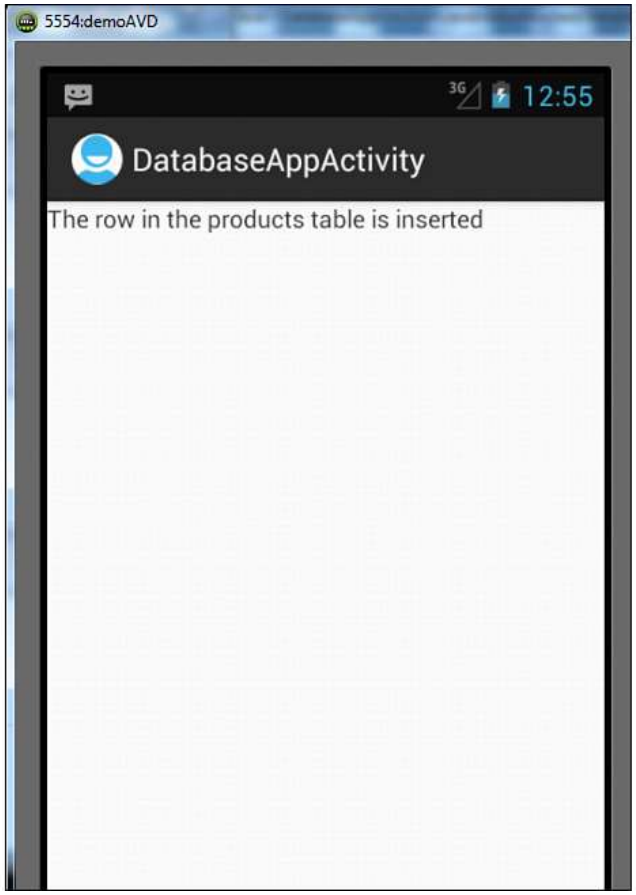

5554:demoAVD
12:27
DatabaseAppActivity
The rows in the products table are:
101, Camera, 15.0
102, Laptop, 1005.99
Insert Rows
List Rows

CREATING A DATA ENTRY FORM

- To add more user interaction
- To create a form for entering new product information, three TextViews, three EditText controls, and two Button controls are arranged in tabular format in the TableLayout.

EXAMPLE


5554:demoAVD
12:54
DatabaseAppActivity
Enter information of the new product
Product Code:
103
Product Name:
CellPhone
Product Price:
24.50
Add Product
Cancel


5554:demoAVD
12:55
DatabaseAppActivity
The row in the products table is inserted

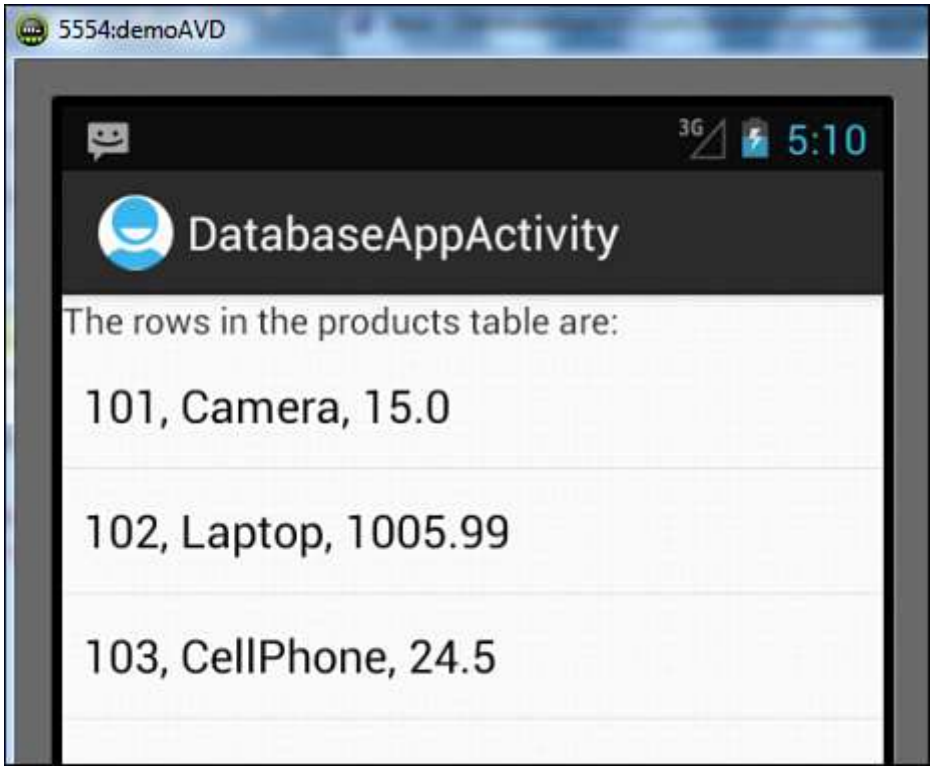

5554:demoAVD
12:55
DatabaseAppActivity
The rows in the products table are:
101, Camera, 15.0
102, Laptop, 1005.99
103, CellPhone, 24.5
Insert Rows
List Rows

Displaying Table Rows Via ListView

- To display the information fetched from the database table with the ListView control instead of the TextView control.

```
<ListView
    android:id="@+id/prodrec"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:drawSelectorOnTop="false"/>
```

```
import java.util.ArrayList;

public ArrayList<String> retrieveRows(){
      ArrayList<String> productRows=new ArrayList<String>();
      String[] columns = new String[]{"code", "product_name", "price"};
      Cursor cursor = db.query(DB_TABLE, columns, null, null, null, null, null);
      cursor.moveToFirst();
      while (cursor.isAfterLast() == false) {
         productRows.add(Integer.toString(cursor.getInt(0)) + ", "+cursor.get-
String(1)+“,"+Float.toString(cursor.getFloat(2)));
         cursor.moveToNext();
      }
      if (cursor != null && !cursor.isClosed()) {
         cursor.close();
      }
      return productRows;
   }
```


5554:demoAVD
3G
5:10
DatabaseAppActivity
The rows in the products table are:
101, Camera, 15.0
102, Laptop, 1005.99
103, CellPhone, 24.5

Android Programming (R15)
UNIT - 5

CONTENTS

- **Part I: Creating Interface Menus and Action Bars:** Menus and Their Types, Creating Menus Through XML, Creating Menus Through Coding, Applying a Context Menu to a List View, Using the Action Bar, Replacing a Menu with the Action Bar, Creating a Tabbed Action Bar, Creating a Drop-Down List Action Bar
- **Part II: Using Databases:** Using the SQLiteOpenHelperclasss, Accessing Databases with the ADB, Creating a Data Entry Form
- **Part III: Communicating with SMS and Emails:** Understanding Broadcast Receivers, Using the Notification System, Sending SMS Messages with Java Code, Receiving SMS Messages, Sending Email, Working With Telephony Manager.

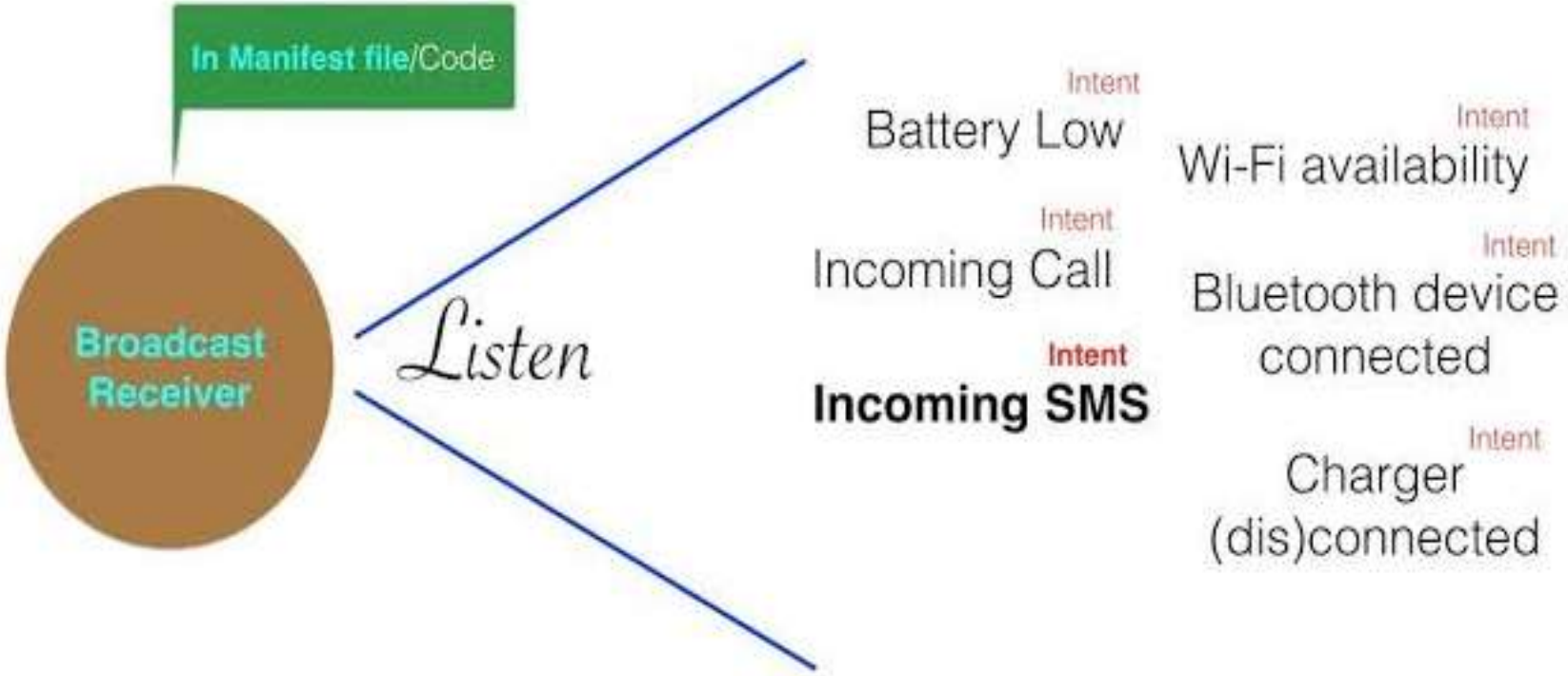
Communicating with SMS and Emails

- Communication with text messages is a popular way to send and receive information via cellular phones.
- Communicating through text messages not only consumes fewer network resources but also reduces network congestion, making it an inexpensive mode of communication.
- Moreover, users can respond to such messages at their leisure.
- We can make our application send and receive text messages via SMS (Short Message Service) automatically at periodic intervals or when a specific event occurs, such as when a button is clicked or when a task is complete.
- All, three communication mediums—SMS, email, and voice calls—use the concept of broadcast receivers and notifications.

UNDERSTANDING BROADCAST RECEIVERS

- A broadcast receiver is a component that responds to different messages that are broadcast by the system or other applications.
- Messages such as new mail, low battery, captured photo, or completed download require attention. Such messages are broadcast as an Intent object that is waiting for broadcast receivers to respond.
- The broadcast receiver responds to such broadcasted Intents and takes the necessary actions.
- The broadcast receivers can, for example, create a status bar notification to alert the user when a broadcast occurs.
- A broadcast Intent can invoke more than one receiver.

Broadcast Receiver - An Introduction



Typically occurring events in Android

- Two separate aspects of broadcast receivers:
  - Broadcasting an Intent
  - Receiving the broadcast Intent

Broadcasting an Intent

- To broadcast an Intent, we first create an Intent object, assign a specific action to it, attach data or a message to the broadcast receiver, and finally broadcast it.
- We can optionally put an extra message or data on the Intent.
- Table 11.1 lists the methods involved in broadcasting an Intent.

Table 11.1. Methods Involved in Broadcasting an `Intent`

Method	Description
`putExtra()`	Used to add data or a message to the `Intent` that we want to send to the broadcast receiver. Syntax: `putExtra(String name, String value)` Where `name` is the key or name of the `value` that we want to pass along with the `Intent`. The `name` is used to identify the `value`.
`setAction()`	Used to set the action to perform on the data or message being sent with the `Intent`. Syntax: `setAction(String action)` The broadcast receiver uses the `getAction()` method to retrieve the action to be performed on the received data.
`sendBroadcast()`	Available on the `Context` class, this method is used to send the broadcast `Intent` to all the registered `Intent` receivers. Syntax: `void sendBroadcast(intent_to_broadcast)` Where the `intent_to_broadcast` parameter represents the `Intent` that we want to broadcast.

The following code broadcasts an `Intent`:

```
public static String BROADCAST_STRING = "com.androidunleashed.testingbroadcast";

Intent broadcastIntent = new Intent();

broadcastIntent.putExtra("message", "New Email arrived");

broadcastIntent.setAction(BROADCAST_STRING);

sendBroadcast(broadcastIntent);
```

Receiving the Broadcast Intent

- A broadcast receiver is a class that extends the BroadcastReceiver.
- It also needs to be registered as a receiver in an Android application via the AndroidManifest.xml file or through code at runtime.
- The broadcast receiver class needs to implement the onReceive()method.

```
public void onReceive(Context context, Intent intent) {
   String actionName = intent.getAction();
   if(actionName != null && actionName.equals(
         "com.androidunleashed.testingbroadcast")) {
      String msg = intent.getStringExtra("message");
      Log.d("Received Message: ",msg);
   }
}
```

- **`getAction()`**—Retrieves the action to be performed from the `Intent` object.

It is the action that indicates what task has to be performed on the data passed along with the `Intent`.

Syntax:

```
getAction()
```

- **`getStringExtra()`**—Retrieves the extended data from the `Inte`

Syntax:

```
getStringExtra(String name)
```

Listing 11.1. Code Written into `activity_broadcast_app.xml`

```
<LinearLayout xmlns:android="http://schemas.android.com/
apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >
  <Button
    android:id="@+id/broadcast_button"
    android:text="Send Broadcast"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center" />
</LinearLayout>
```

Listing 11.2. Code Written into `BroadcastAppActivity.java`

```
package com.androidunleashed.broadcastapp;

import android.app.Activity;
import android.os.Bundle;
import android.content.Intent;
import android.widget.Button;
import android.view.View;

public class BroadcastAppActivity extends Activity {
  public static String BROADCAST_STRING = "com.androidunleashed.testingbroadcast";
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_broadcast_app);
    Button broadcastButton = (Button) this.findViewById(R.id.broadcast_button);
    broadcastButton.setOnClickListener(new Button.OnClickListener(){
      public void onClick(View v) {
        Intent broadcastIntent = new Intent();
        broadcastIntent.putExtra("message", "New Email arrived");
        broadcastIntent.setAction(BROADCAST_STRING);
        sendBroadcast(broadcastIntent);
      }
    });
  }
}
```

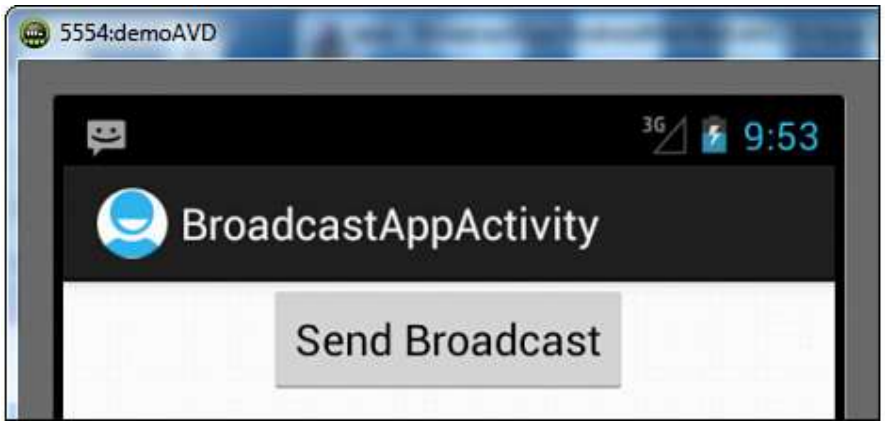
Listing 11.3. Code Written into `ReceiveBroadcastActivity.java`

```
package com.androidunleashed.broadcastapp;
import android.content.BroadcastReceiver;
import android.content.Intent;
import android.content.Context;
import android.util.Log;

public class ReceiveBroadcastActivity extends BroadcastReceiver
{
   @Override
   public void onReceive(Context context, Intent intent) {
     String actionName = intent.getAction();

     if(actionName != null && actionName.equals(“
       com.androidunleashed.testingbroadcast"))
       {
              String msg = intent.getStringExtra("message");
              Log.d("Received Message: ",msg);
       }
  }
}
```

AndroidManifest.xml

The code for registering the activity is as follows:

```
<receiver android:name=".ReceiveBroadcastActivity">
   <intent-filter>
      <action android:name="com.androidunleashed.testingbroadcast"></action>
   </intent-filter>
</receiver>
```



USING THE NOTIFICATION SYSTEM

- The Android notification system provides us with several ways of alerting users.
- For example, the user can be notified with text, vibration, blinking lights, and sound indicators.
- Notifications are usually displayed on the status bar at the top of the screen.
- Not all notifications are supported by all devices.

Notification via the Status Bar

- The simplest type of notification is status, which appears in the status bar as an icon along with some optional ticker text.
- Users can pull down the status bar to see the notification list and clear the notification by clicking the Clear button.
- After tapping the notification, the user navigates to the Intent defined by the notification.
- The status notification never launches an activity automatically, but simply notifies the user and launches the activity only when the notification is selected.
- Besides an icon and ticker text, the notification can have a title and body text displayed when the full notification is being displayed.
- The ticker text is briefly displayed in the status bar when the notification fires.

Notification via the Status Bar

- For creating notifications, the following two classes are used:
- **Notification**—The object that defines the information to be displayed, which can be text to be displayed on the status/expanded status bar, an icon displayed with the text, the number of times the notification is triggered, and so on.
- **NotificationManager**—The base object with which notifications are handled. It displays the information encapsulated in the Notification object, which is displayed via the notify() method.

Creating Notifications

The first step is to create a `Notification` object and configure it by defining notification properties.

```
Notification notification = new Notification();
notification.icon = R.drawable.ic_launcher;
notification.tickerText = "There is a new notification";
notification.when = System.currentTimeMillis();
notification.flags |= Notification.FLAG_AUTO_CANCEL;
```

We can also assign a notification icon, ticker text, and time of occurrence through the `Notification` object constructor,

```
Notification notification  = new Notification(R.drawable.ic_launcher,
 "There is a new notification", System.currentTimeMillis());
```

Creating PendingIntent

- After receiving the notification, we may choose to take a necessary action.
- We use the PendingIntent class to switch to the desired Intent when the notification is tapped.
- The PendingIntent class enables us to create Intents that can be triggered by our application when an event occurs.

```
Intent intent = new Intent(getBaseContext(), TargetActivity.class);
PendingIntent pendIntent =
PendingIntent.getActivity(getBaseContext(), 0, intent, 0);
```

- To display text after expanding the notification, and to specify the pending Intent that we want to launch, the setLatestEventInfo() method of the Notification class is used.
- The method is now deprecated. The task of configuring a notification is done through Notification.Builder.
- The syntax of the method is

  setLatestEventInfo(application_context, title, text, pending_intent);

Using Notification.Builder

- Notification.Builder is the Builder class for Notification objects and provides several methods to configure notification,

Method	Description
`setSmallIcon()`	Used to supply the small icon resource that is displayed to represent the notification in the status bar. Syntax: `setSmallIcon(int icon)` where the `icon` parameter represents the resource ID of the drawable to be used as the icon of the notification.
`setAutoCancel()`	Used to determine whether we want to make the notification invisible when it is tapped. The Boolean value `true` is supplied to this method to make the notification invisible. Syntax: `setAutoCancel(boolean autoCancel)`
`setTicker()`	Used to supply the ticker text that is displayed in the status bar when the notification arrives. Syntax: `setTicker(CharSequence textMessage)`
`setWhen()`	Used to supply the time of occurrence of the notification. Syntax:`setWhen(long timeOfOccurrence)`
`setContentTitle()`	Used to supply the title of the notification when the status bar is expanded. Syntax: `setContentTitle(CharSequence title)`
`setContentText()`	Used to supply the text of the notification. Syntax: `setContentText(CharSequence text)`
`setContentIntent()`	Used to supply a `PendingIntent` to be sent when the notification is tapped. Syntax: `setContentIntent(PendingIntent intent)`

Listing 11.5. Code Written into `activity_notification_app.xml`

```
<LinearLayout xmlns:android="http://schemas.android.com/
      apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >
  <Button
    android:id="@+id/createbutton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Create Notification"
    android:layout_gravity="center" />
</LinearLayout>
```

Listing 11.6. Code Written into `target.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/
      apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >
  <TextView
    android:id="@+id/messageview"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="This is target activity"
    android:layout_gravity="center" />
</LinearLayout>
```

Listing 11.7. Code Written into `TargetActivity.java`

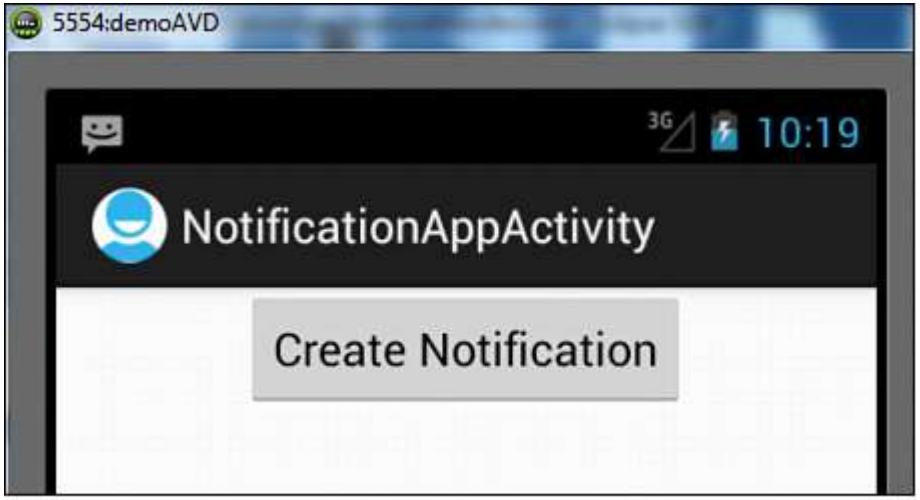
```
package com.androidunleashed.notificationapp;
import android.app.Activity;
import android.os.Bundle;

public class TargetActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.target);
  }
}
```

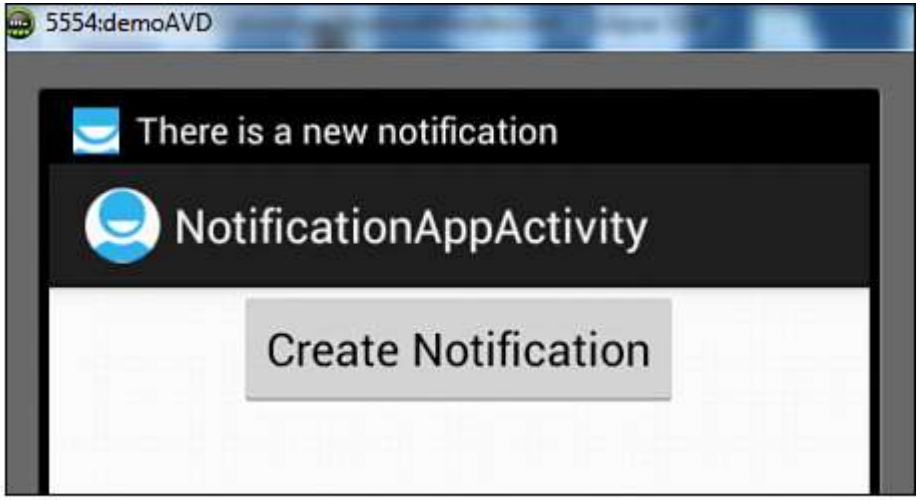
Listing 11.8. Code Written into `NotificationAppActivity.java`

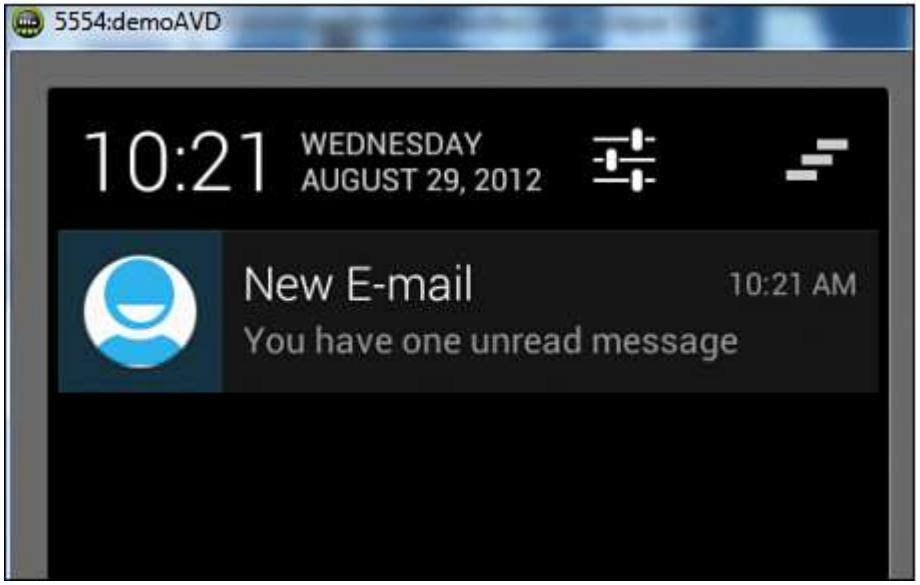
```
package com.androidunleashed.notificationapp;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.app.Notification;
import android.widget.Button;
import android.view.View.OnClickListener;

public class NotificationAppActivity extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_notification_app);
    Button createButton = (Button) findViewById(R.id.createbutton);
    createButton.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View arg0) {
        Intent intent = new Intent(getBaseContext(), TargetActivity.class);
        PendingIntent pendIntent = PendingIntent.
getActivity(getBaseContext(), 0, intent, 0);
        NotificationManager notificationManager = (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);
        Notification notification = new Notification();
        Notification.Builder builder = new Notification.
Builder(getBaseContext())
        .setSmallIcon(R.drawable.ic_launcher)
        .setAutoCancel(true)
        .setTicker("There is a new notification")
        .setWhen(System.currentTimeMillis())
        .setContentTitle("New E-mail")
        .setContentText("You have one unread message")
        .setContentIntent(pendIntent);
```

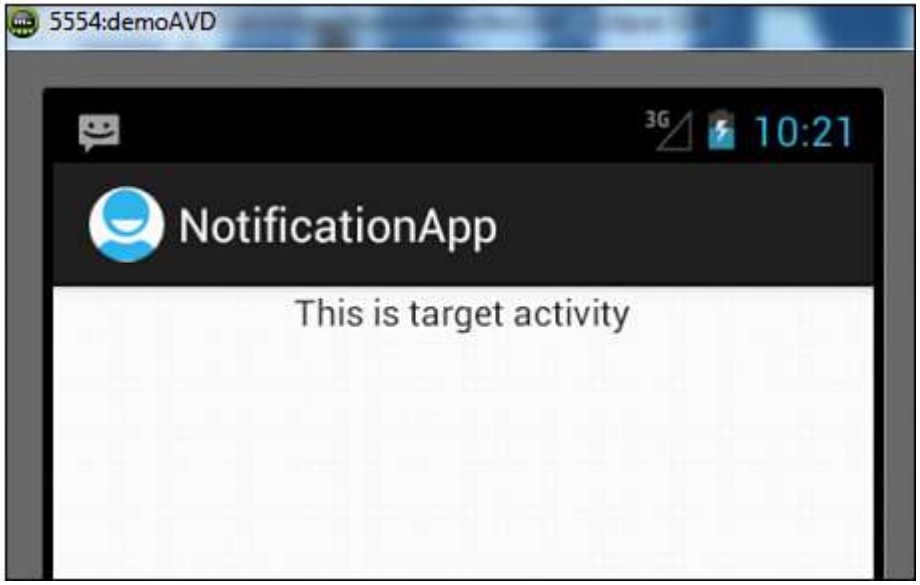
Listing 11.9. Code Written into `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.androidunleashed.notificationapp"
  android:versionCode="1"
  android:versionName="1.0" >
  <uses-sdk android:minSdkVersion="8"
    android:targetSdkVersion="15" />
  <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name=".NotificationAppActivity"
      android:label="@string/title_activity_notification_app" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name=".TargetActivity" android:label="@string/app_na
  </application>
</manifest>
```


5554:demoAVD
10:19
NotificationAppActivity
Create Notification


5554:demoAVD
There is a new notification
NotificationAppActivity
Create Notification


5554:demoAVD
10:21
WEDNESDAY
AUGUST 29, 2012
New E-mail
10:21 AM
You have one unread message


5554:demoAVD
10:21
NotificationApp
This is target activity

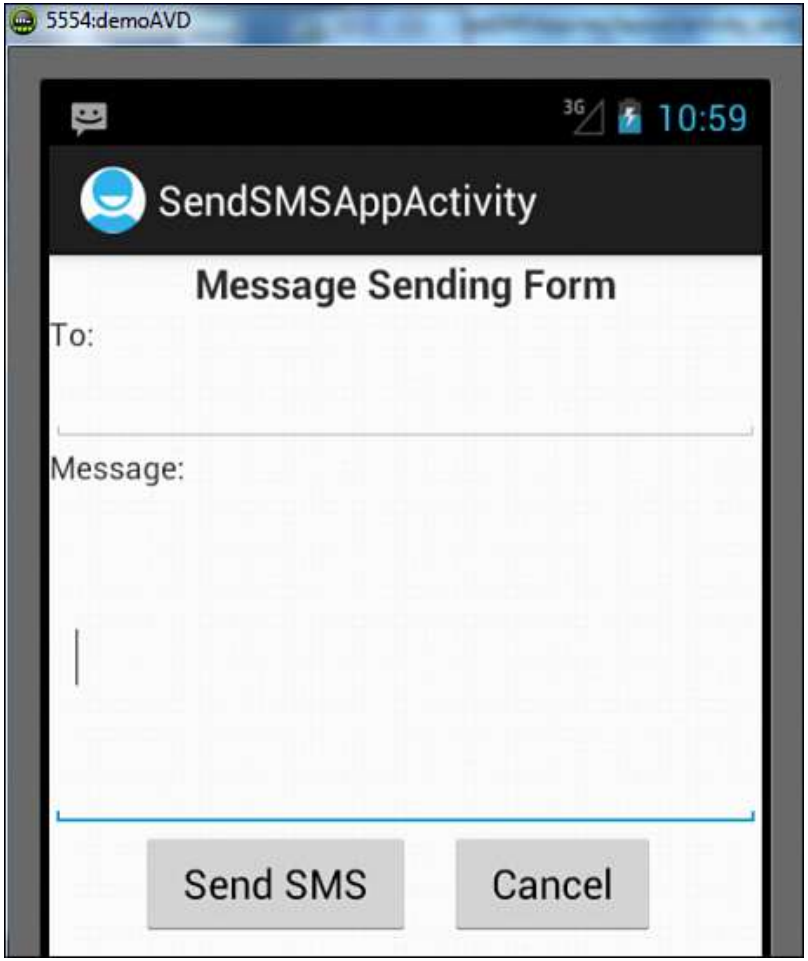
SENDING SMS MESSAGES WITH JAVA CODE

- Sending and receiving SMS messages is considered as one of the most economical and popular modes of communication.
- To implement the SMS Messaging facility in our application, Android provides a built-in class known as SmsManager, which we can use to send and receive SMS messages.
- For testing SMS messaging, we don't need a real device but can easily test it on the Android emulator.

Listing 11.10. Code Written into `activity_send_smsapp.xml`

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_height="wrap_content"
        android:text="Message Sending Form"
        android:textStyle="bold"
        android:textSize="18sp"
        android:layout_width="match_parent"
        android:gravity="center_horizontal"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="To:"  />
    <EditText
        android:id="@+id/recvr_no"
        android:layout_height="wrap_content"
        android:layout_width="match_parent" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Message:" />
    <EditText
        android:id="@+id/txt_msg"
        android:layout_width="match_parent"
        android:layout_height="150dp" />
```

```
<RelativeLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="horizontal">
   <Button
     android:id="@+id/send_button"
     android:text="Send SMS"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_marginLeft="40dip"
     android:paddingLeft="20dip"
     android:paddingRight="20dip" />
   <Button
     android:id="@+id/cancel_button"
     android:text="Cancel"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_toRightOf="@id/send_button"
     android:layout_marginLeft="15dip"
     android:paddingLeft="20dip"
     android:paddingRight="20dip" />
  </RelativeLayout>
</LinearLayout>
```



Getting Permission to Send SMS Messages

- To send and receive SMS messages in an application, we need to use permissions.
- All the permissions our application uses are defined in the AndroidManifest.xml file, so that when the application is installed on a device, the user is informed of the access permissions that the application uses.

```
<uses-permission
  android:name="android.permission.SEND_SMS"/>
```

Writing Java Code

- To add an action to the send_button and cancel_button, we need to write Java code into the SendSMSAppActivity.java activity file.
- Validate the content of the recvr_no and txt_msg EditText controls. Recall that the two EditText controls are meant for entering the phone number of the recipient and text message of the SMS. If either of the two EditText controls is empty, we want to suspend the process and prompt the user to enter data into both the controls.
- Send an SMS message to the recipient.
- Inform the user whether the SMS message was successfully sent and delivered.

- The difference between SMS Sent and SMS Delivered status is that the former means the SMS message was received by the server or SMSC (Short Message Service Center). The latter means that the SMS message was received by the recipient from the server.
- Remember that if the recipient is out of range or offline the SMS message still can be Sent. When the recipient actually receives the message, it is then successfully delivered.

Listing 11.12. Code Written into `SendSMSAppActivity.java`

```
package com.androidunleashed.sendsmsapp;
import android.app.Activity;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.widget.Button;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Intent;
import android.content.Context;
import android.content.IntentFilter;

public class SendSMSAppActivity extends Activity {
    EditText phoneNumber, message;
    BroadcastReceiver sentReceiver, deliveredReceiver;
    String SENT = "SMS_SENT";
    String DELIVERED = "SMS_DELIVERED";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_send_smsapp);
        final PendingIntent sentPendIntent = PendingIntent.getBroadcast(this, 0, new
Intent(SENT), 0);
        final PendingIntent delivered_pendintnet = PendingIntent.getBroadcast(this,
0,  new Intent(DELIVERED), 0);
sentReceiver = new BroadcastReceiver(){
            public void onReceive(Context arg0, Intent arg1) {
                switch (getResultCode()) {
                    case Activity.RESULT_OK:
                       Toast.makeText(getBaseContext(), "SMS sent", Toast.LENGTH_SHORT).show();
                       break;
```

```
case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
               Toast.makeText(getBaseContext(), "Generic failure", Toast.LENGTH_SHORT).show();
                        break;
                    case SmsManager.RESULT_ERROR_NO_SERVICE:
               Toast.makeText(getBaseContext(), "No service",  Toast.LENGTH_SHORT).show();
                        break;
                    case SmsManager.RESULT_ERROR_NULL_PDU:
               Toast.makeText(getBaseContext(), "Null PDU", Toast.LENGTH_SHORT).show();
                        break;
                    case SmsManager.RESULT_ERROR_RADIO_OFF:
               Toast.makeText(getBaseContext(), "Radio off",  Toast.LENGTH_SHORT).show();
                        break;
                 }
            }
        };
        deliveredReceiver = new BroadcastReceiver(){
            @Override
            public void onReceive(Context arg0, Intent arg1) {
                switch (getResultCode()) {
                    case Activity.RESULT_OK:
                        Toast.makeText(getBaseContext(), "SMS successfully deliv-
ered",  Toast.LENGTH_SHORT).show();
                        break;
                    case Activity.RESULT_CANCELED:
                        Toast.makeText(getBaseContext(), "Failure—SMS not deliv-
ered",  Toast.LENGTH_SHORT).show();
                        break;
                }
            }
        };
```

```
registerReceiver(sentReceiver, new IntentFilter(SENT));
   registerReceiver(deliveredReceiver, new IntentFilter(DELIVERED));
   Button sendBtn = (Button) this.findViewById(R.id.send_button);
   sendBtn.setOnClickListener(new Button.OnClickListener(){
    public void onClick(View v) {
      phoneNumber = (EditText) findViewById(R.id.recvr_no);
      message = (EditText) findViewById(R.id.txt_msg);
      if(phoneNumber.getText().toString().trim().length() >0 && message.
getText().toString().trim().length() >0) {
        SmsManager sms = SmsManager.getDefault();
        sms.sendTextMessage(phoneNumber.getText().toString(), null,
        message.getText().toString(), sentPendIntent, delivered_pendint-
net);
      }
      else {
        Toast.makeText(SendSMSAppActivity.this, "Either phone number or
text is missing", Toast.LENGTH_SHORT).show();
      }
    }
   });
   Button cancelBtn = (Button) this.findViewById(R.id.cancel_button);
   cancelBtn.setOnClickListener(new Button.OnClickListener(){
    public void onClick(View v) {
      phoneNumber.setText("");
      message.setText("");
    }
   });
  }
}
```

To run the application and watch its output, we need to have two AVDs (Android Virtual Devices) running.

RECEIVING SMS MESSAGES

- When a device receives a new SMS message, a new broadcast Intent is fired with the android.provider.Telephony.SMS_RECEIVED action.
- To listen for this action, application must register a BroadcastReceiver. That is, we need to create a receiver—a class that extends the BroadcastReceiver and then registers it in the manifest file.
- After register the receiver, our application is notified whenever an SMS message is received.

Listing 11.13. Code Written into `ReceiverSMS.java`

```
package com.androidunleashed.receivesmsapp;

import android.content.BroadcastReceiver;
import android.content.Intent;
import android.content.Context;
import android.os.Bundle;
import android.telephony.SmsMessage;
import android.widget.Toast;

public class ReceiverSMS extends BroadcastReceiver {

  @Override
  public void onReceive(Context context, Intent intent) {
    Bundle bundle = intent.getExtras();
    SmsMessage[] msg = null;
    String str = "";
    if (bundle != null) {
      Object[] pdus = (Object[]) bundle.get("pdus");
      msg = new SmsMessage[pdus.length];
      for (int i=0; i<msg.length; i++){
        msg[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
        str += "SMS Received from: " + msg[i].getOriginatingAddress();
        str += ":";
        str += msg[i].getMessageBody().toString();
        str += "\n";
      }
      Toast.makeText(context, str, Toast.LENGTH_SHORT).show();
    }
  }
}
```

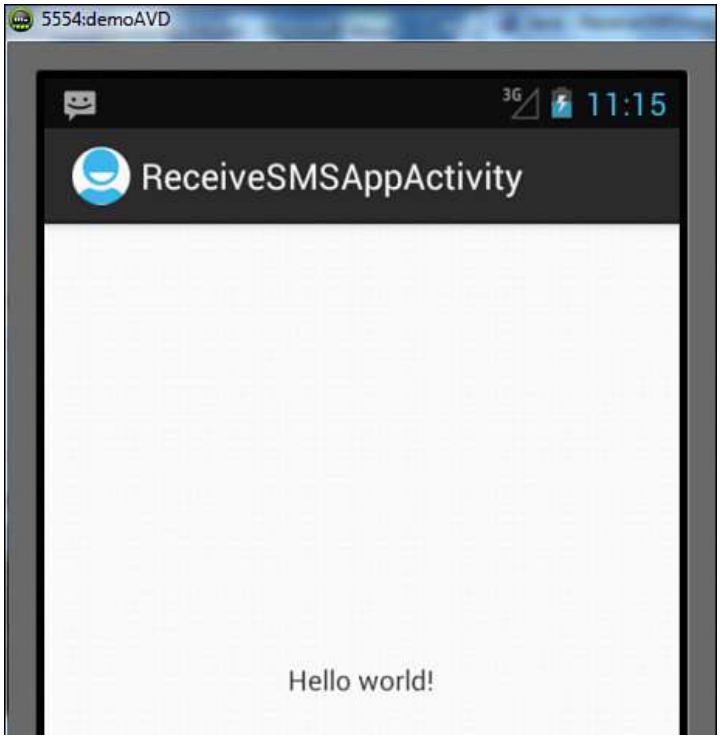
- The information of all received messages is gathered in the SmsMessage array msg.
- Each array element represents the complete information of a single received SMS that includes the originating address—the phone number, timestamp, and the message body.
- The following methods are used to fetch the SMS information from the SmsMessage array elements:
- **getOriginatingAddress()**—Fetches the phone number of the SMS recipient
- **getTimestampMillis()**—Fetches the time at which the SMS is received
- **getMessageBody()**—Fetches the message body

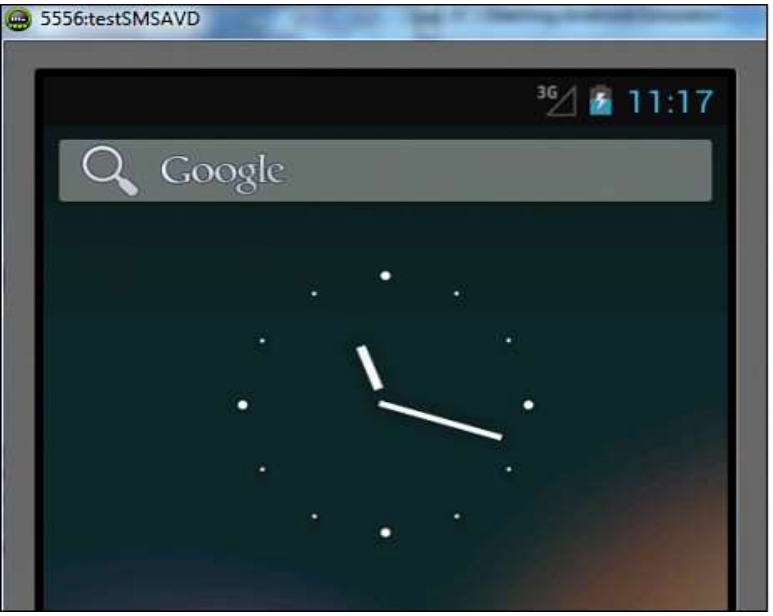
the `AndroidManifest.xml` file:

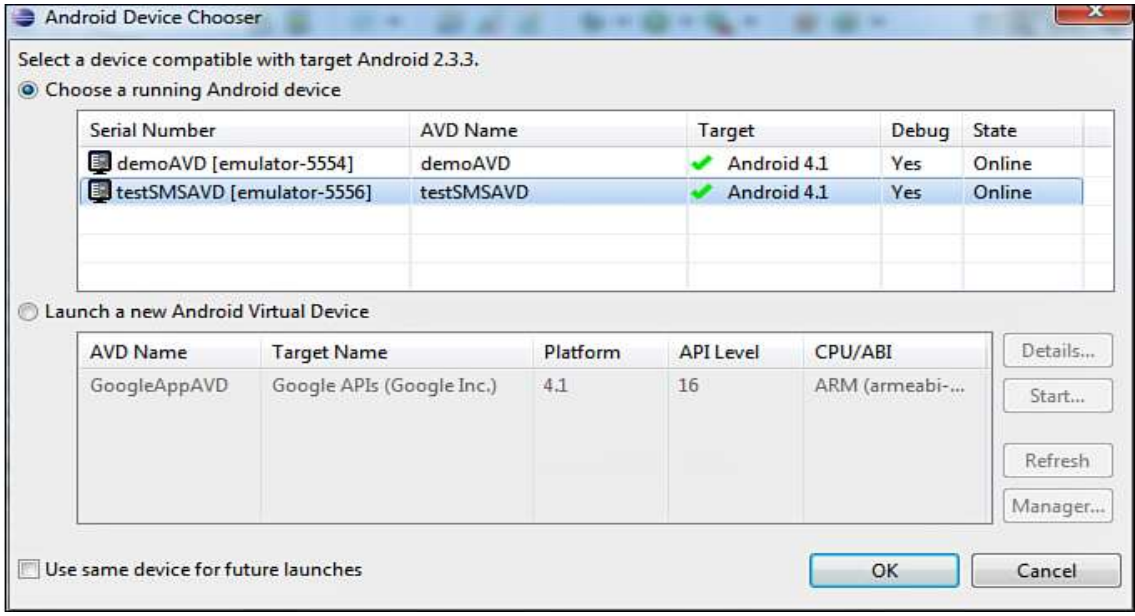
```
<receiver android:name=".ReceiverSMS">
   <intent-filter>
      <action android:name="android.provider.Telephony.SMS_RECEIVED" />
   </intent-filter>
</receiver>
```

Add the android.permission.RECEIVE_SMS permission to the manifest file:

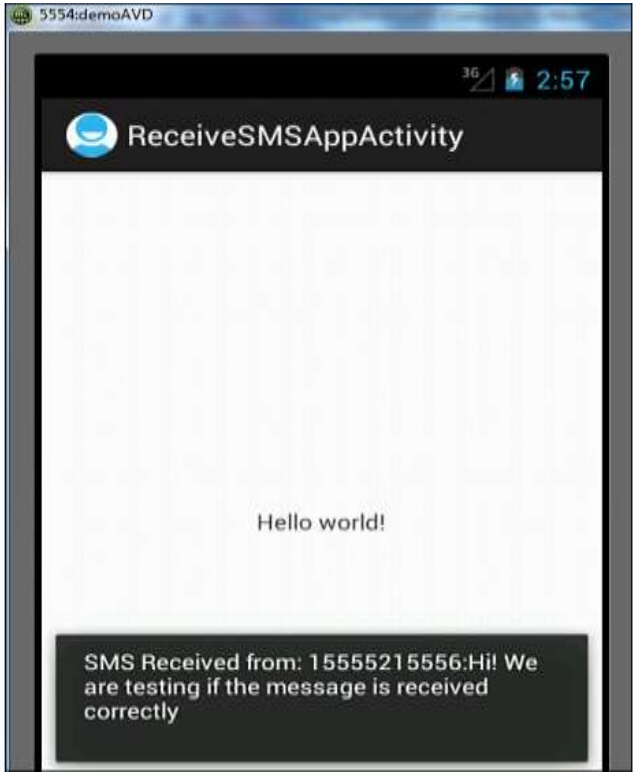
```
<uses-permission android:name="android.permission.RECEIVE_SMS" />
```

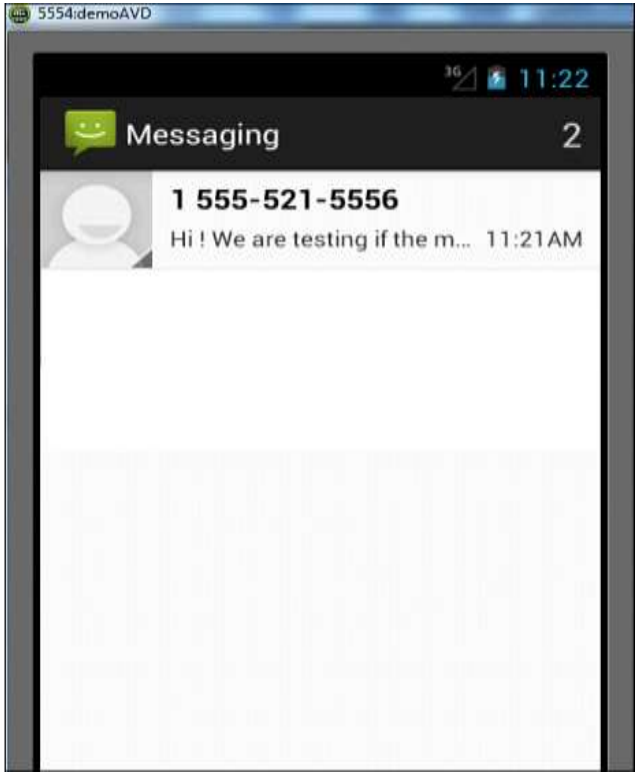



5556:testSMSAVD
11:17
Google


Android Device Chooser
Select a device compatible with target Android 2.3.3.
Choose a running Android device
Serial Number
AVD Name
Target
Debug
State
demoAVD [emulator-5554]
demoAVD
Android 4.1
Yes
Online
testSMSAVD [emulator-5556]
testSMSAVD
Android 4.1
Yes
Online
Launch a new Android Virtual Device
AVD Name
Target Name
Platform
API Level
CPU/ABI
GoogleAppAVD
Google APIs (Google Inc.)
4.1
16
ARM (armeabi-...
Details...
Start...
Refresh
Manager...
Use same device for future launches
OK
Cancel


5556:testSMSAVD
11:21
SendSMSAppActivity
Message Sending Form
To:
5554
Message:
Hi ! We are testing if the message is received correctly
Send SMS
Cancel


5554:demoAVD
2:57
ReceiveSMSAppActivity
Hello world!
SMS Received from: 15555215556:Hi! We are testing if the message is received correctly


5554:demoAVD
11:22
Messaging
2
1 555-521-5556
Hi ! We are testing if the m... 11:21AM

Android Programming (R15)
UNIT - 5

CONTENTS

- **Part I: Creating Interface Menus and Action Bars:** Menus and Their Types, Creating Menus Through XML, Creating Menus Through Coding, Applying a Context Menu to a List View, Using the Action Bar, Replacing a Menu with the Action Bar, Creating a Tabbed Action Bar, Creating a Drop-Down List Action Bar
- **Part II: Using Databases:** Using the SQLiteOpenHelperclasss, Accessing Databases with the ADB, Creating a Data Entry Form
- **Part III: Communicating with SMS and Emails:** Understanding Broadcast Receivers, Using the Notification System, Sending SMS Messages with Java Code, Receiving SMS Messages, Sending Email, Working With Telephony Manager.

SENDING EMAIL

- To send an email with Android, we use the following Intent:

  Intent.ACTION_SEND

- The Intent.ACTION_SEND calls an existing email client to send an email. So, for sending email through the Android emulator, we need to first configure the email client. If the email client is not configured, the emulator does not respond to the Intent.
- Through the ACTION_SEND Intent, messages of different types, such as text or image, can be sent. The only thing we need to do is to set the type of the Intent through the setType() method. For example, the following statement declares that the text data type will be sent through the Intent:

  emailIntent.setType("plain/text");

The emailIntent is the Intent.ACTION_SEND object.

SENDING EMAIL

- To let a user choose the email client to handle the Intent, we call startActivity() with the createChooser() method.
- As the name suggests, the createChooser() method prompts the user to choose the application to handle the Intent.

```
startActivity(Intent.createChooser(emailIntent, "Sending Email"));
```

- This statement displays all the applications that are eligible to handle the Intent, allowing the user to choose the application to launch.
- If there is only one application able to handle the Intent, it is launched automatically without prompting the user.

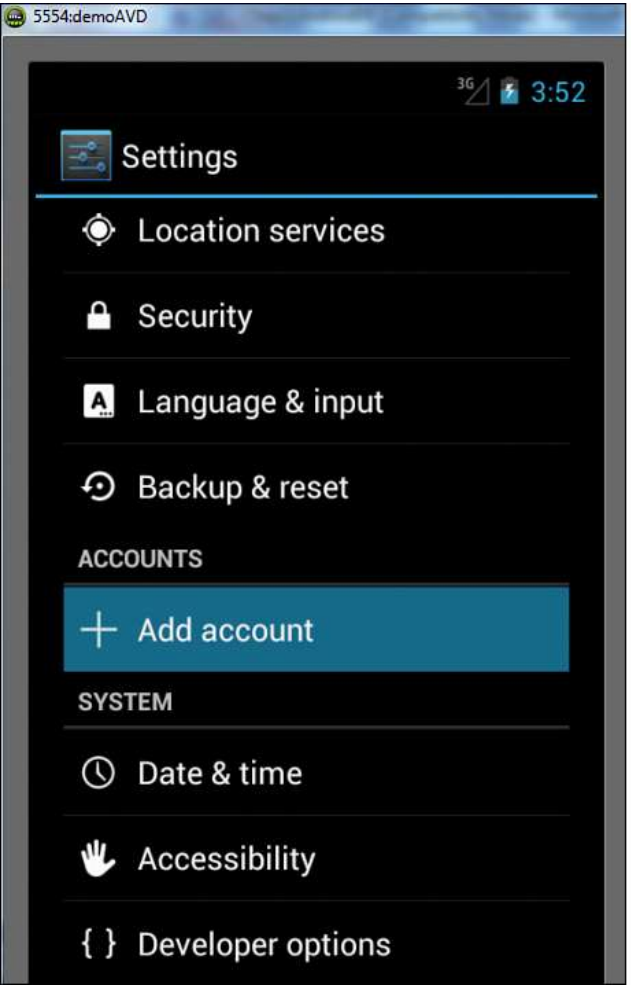
SENDING EMAIL

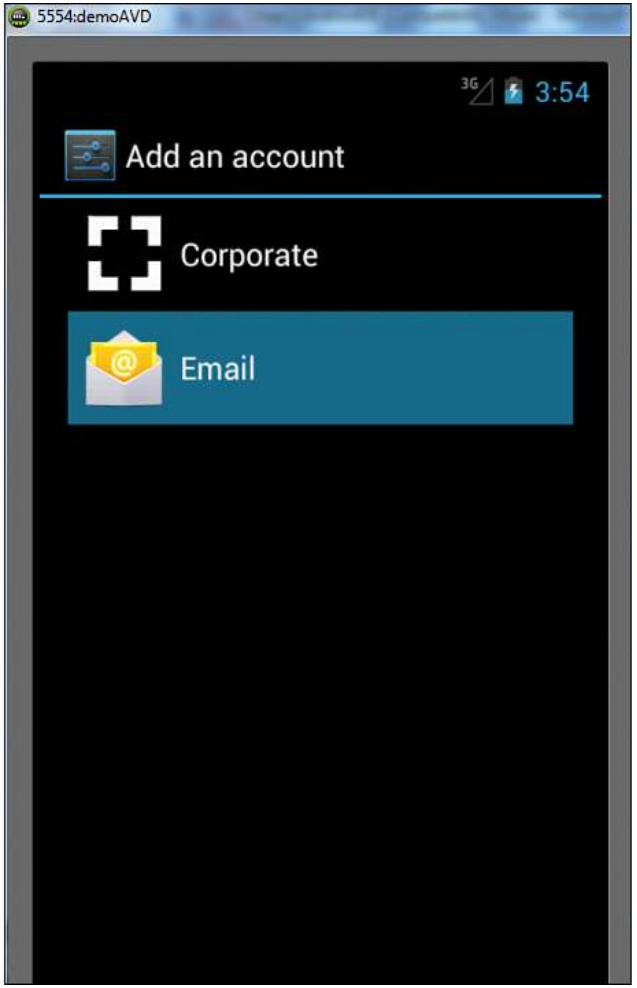
- To supply data for the email message fields, we set certain standard extras for the Intent.
- **EXTRA_EMAIL**—Sets the To: address (email address of the receiver)
- **EXTRA_CC**—Sets the Cc: address (email address of the carbon copy receiver)
- **EXTRA_BCC**—Sets the Bcc: address (email address of the blind carbon copy receiver)
- **EXTRA_SUBJECT**—Sets the Subject of the email
- **EXTRA_TEXT**—Sets the body of the email
- After setting the desired Intent's extras, launch the activity to initiate the sending email task.

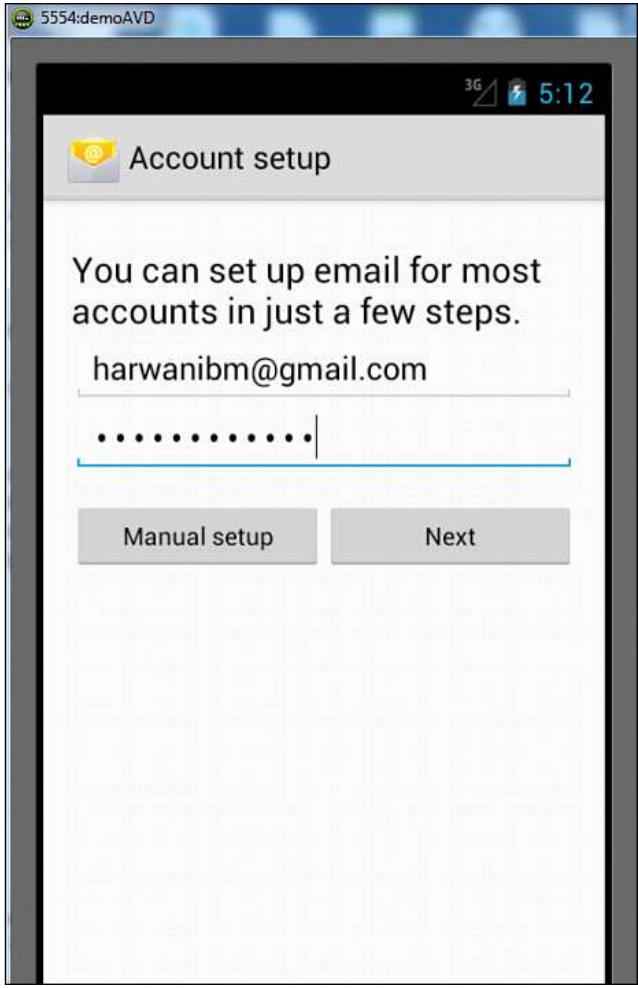
Configure the email client of the Android emulator

- Start the emulator and then click on the Menu button.
- Click on the System settings option.
- From the Accounts section, click on the Add account button.
- From the two options, Corporate and Email (see Figure 11.11—middle), click the Emailo ption. We get the Account setup form where we need to enter our existing Email ID and password (see Figure 11.11—right) followed by clicking the Next button.

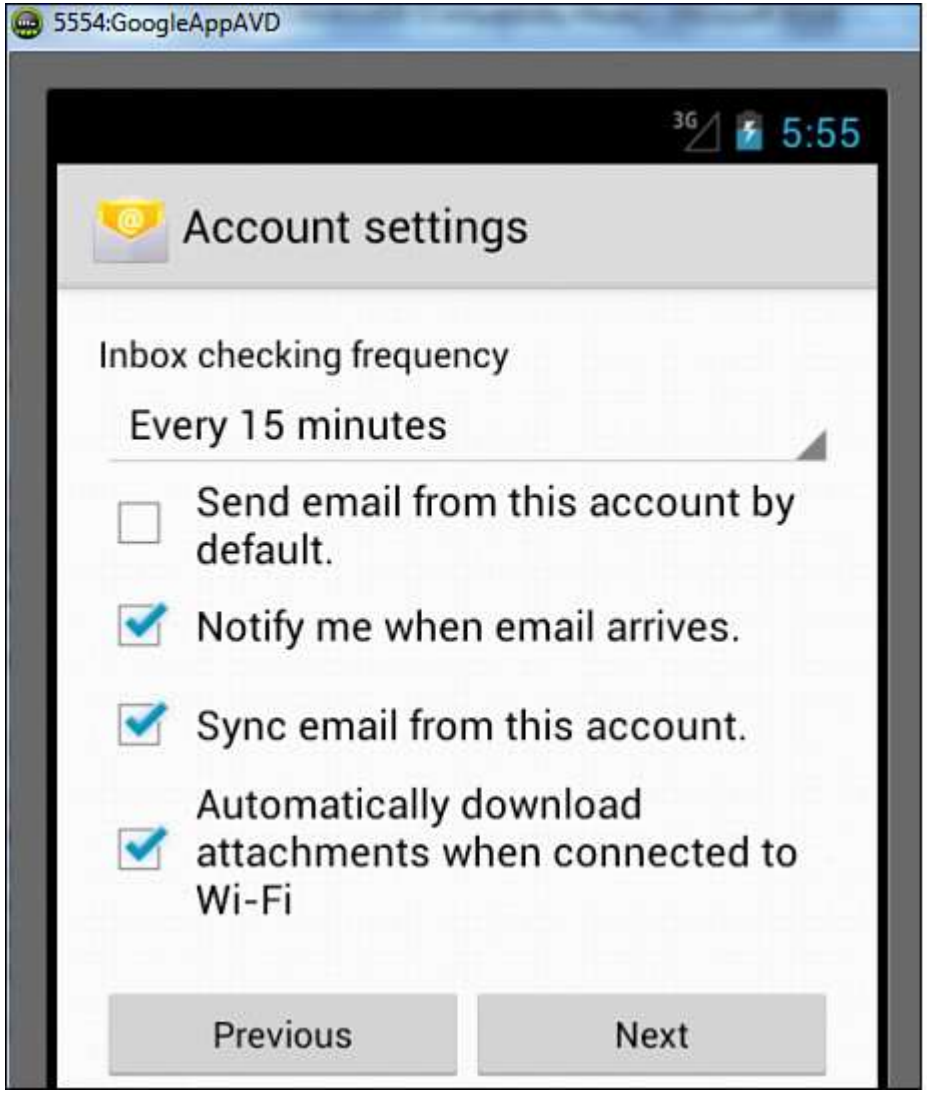
Figure 11.11. (left) Settings options in the Android emulator, (middle) two options of adding an account, and (right) account setup form to enter email ID and password

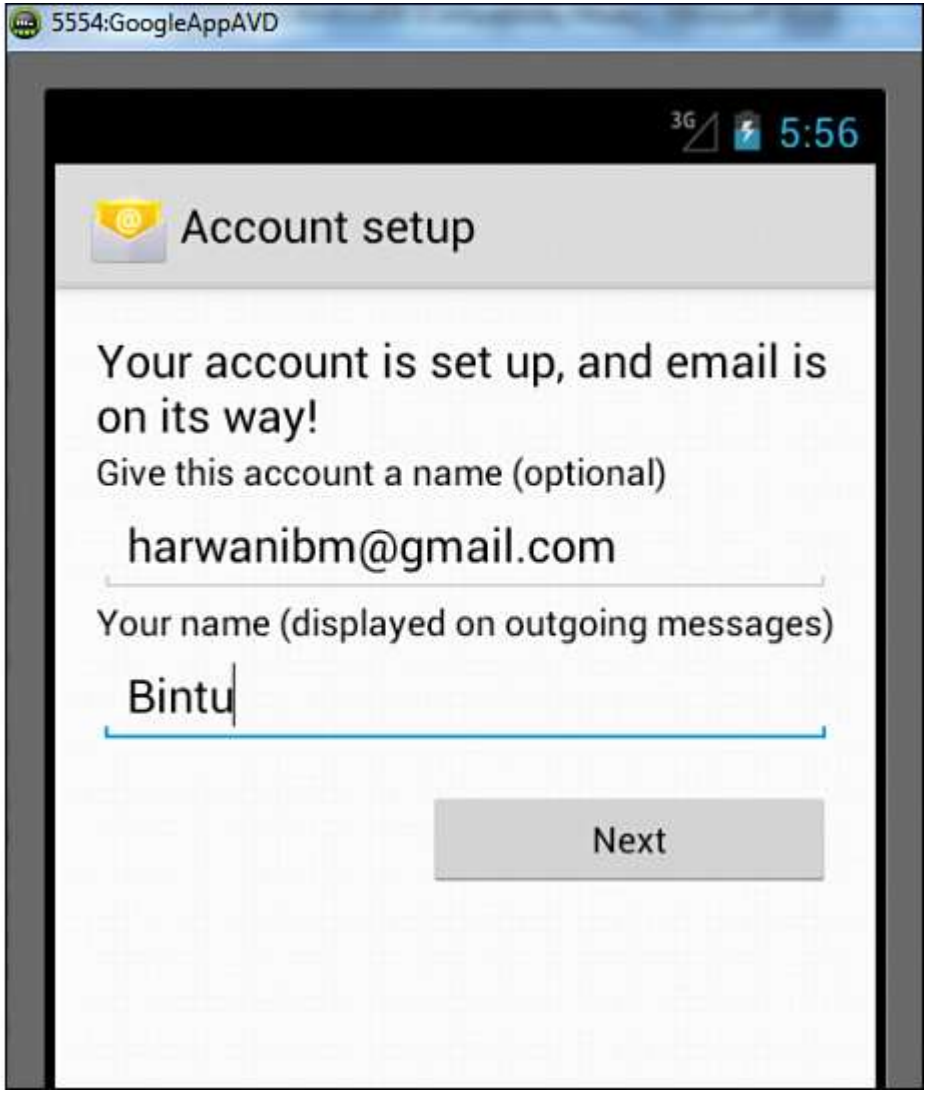






• The emulator checks for the incoming and outgoing servers and on finding them displays `Account settings` as shown in Figure 11.12 (left). Select the required check boxes and then click the `Next` button.
Our email client is set up, and we are prompted to enter the name to be displayed on the outgoing messages(see Figure 11.12—right).
Click the `Next` button to finish configuring the email client.





Now the email client is successfully configured in the emulator, and we can go ahead and send mail.
Let's create an application to send an email. Create a new Android project called SendEmailApp.

Listing 11.15. Code Written into `activity_send_email_app.xml`

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:id="@+id/email_form"
        android:text = "Email Form"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:typeface="serif"
        android:textSize="18sp"
        android:textStyle="bold"
        android:padding="10dip"
        android:layout_centerHorizontal="true"/>
    <TextView
        android:id="@+id/to_addressview"
        android:text = "To:"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="10dip"
        android:layout_below="@id/email_form" />
```

```
<EditText
    android:id="@+id/toaddresses"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:layout_below="@id/email_form"
        android:layout_toRightOf="@id/to_addressview"
        android:singleLine="true" />
    <TextView
        android:id="@+id/cc_addressview"
        android:text = "Cc:"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/to_addressview"
        android:layout_margin="10dip" />
    <EditText
        android:id="@+id/ccaddresses"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:singleLine="true"
        android:layout_below="@id/toaddresses"
        android:layout_toRightOf="@id/cc_addressview" />
    <TextView
        android:id="@+id/bcc_addressview"
        android:text = "Bcc:"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/cc_addressview"
        android:layout_margin="10dip" />
```

```
<TextView
        android:id="@+id/subjectview"
        android:text = "Subject:"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/bcc_addressview"
        android:layout_margin="10dip"
        android:paddingTop="10dip"/>
    <EditText
        android:id="@+id/emailsubject"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:singleLine="true"
        android:layout_below="@id/bccaddresses"
        android:layout_toRightOf="@id/subjectview"
        android:layout_marginTop="10dip" />
    <TextView
        android:id="@+id/emailtextview"
        android:text = "Message:"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/subjectview"
        android:layout_margin="10dip" />
    <EditText
        android:id="@+id/emailtext"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:lines="5"
        android:layout_below="@id/emailsubject"
        android:layout_toRightOf="@id/emailtextview" />
```

```
<Button
        android:id="@+id/send_button"
        android:text="Send"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:paddingLeft="25dip"
        android:paddingRight="25dip"
        android:layout_marginTop="10dip"
        android:layout_below="@id/emailtext" />
</RelativeLayout>
```

Listing 11.16. Code Written into `SendEmailAppActivity.java`

```
package com.androidunleashed.sendemailapp;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.content.Intent;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class SendEmailAppActivity extends Activity {
    Button sendbutton;
    EditText toAddress, ccAddress, bccAddress, subject, emailMessage;
    String  toAdds, ccAdds, bccAdds;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_send_email_app);
        sendbutton=(Button) findViewById(R.id.send_button);
        toAddress=(EditText) findViewById(R.id.toaddresses);
        ccAddress=(EditText) findViewById(R.id.ccaddresses);
        bccAddress=(EditText) findViewById(R.id.bccaddresses);
        subject=(EditText) findViewById(R.id.emailsubject);
        emailMessage=(EditText) findViewById(R.id.emailtext);
        sendbutton.setOnClickListener(new OnClickListener(){
            @Override
            public void onClick(View v) {
                final Intent emailIntent = new Intent(Intent.ACTION_SEND);
                if(toAddress.getText().length() >0) {
                    toAdds = '"'+toAddress.getText().toString()+'"';
                    emailIntent.putExtra(Intent.EXTRA_EMAIL, new String[]{ toAdds});
                }
```

```
if(ccAddress.getText().length() >0) {
                    ccAdds = '"'+ccAddress.getText().toString()+'"';
                    emailIntent.putExtra(Intent.EXTRA_CC, new String[]{ ccAdds});
                }
                if(bccAddress.getText().length() >0) {
                    bccAdds = '"'+bccAddress.getText().toString()+'"';
                    emailIntent.putExtra(Intent.EXTRA_BCC, new String[]{ bccAdds});
                }
                emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject.getText().
toString());
                emailIntent.putExtra(Intent.EXTRA_TEXT, emailMessage.getText());
                emailIntent.setType("plain/text");
                startActivity(Intent.createChooser(emailIntent, "Sending Email"));
            }
        });
    }
}
```

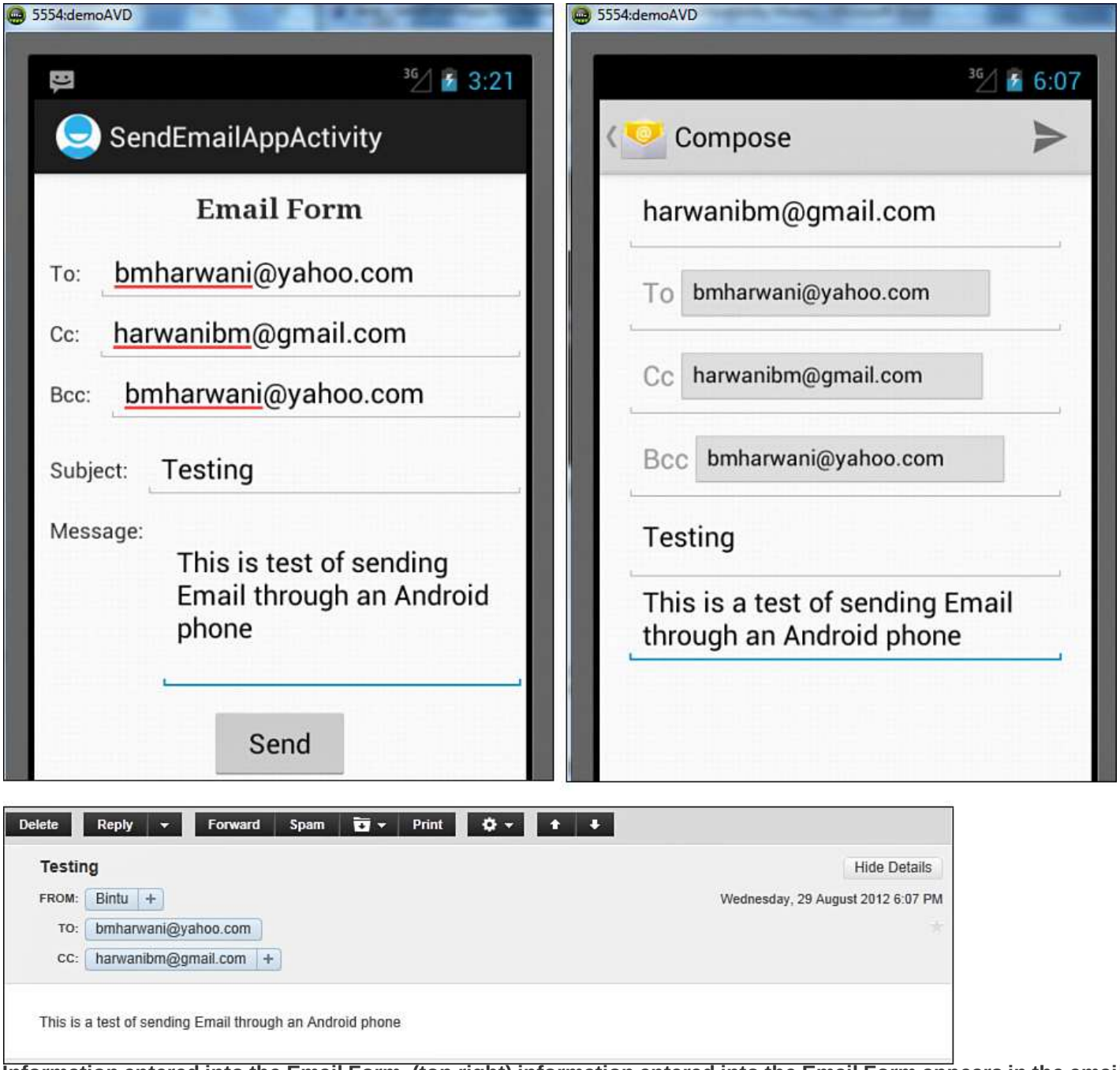


Figure 11.13. (top left) Information entered into the Email Form, (top right) information entered into the Email Form appears in the email client fields, and (bottom) email received by the recipient

WORKING WITH THE TELEPHONY MANAGER

- The Android telephony APIs include the Telephony Manager that accesses the telephony services on the device and enables us to
  - Provide a user interface for entering or modifying the phone number to dial.
  - Implement call handling in the application.
  - Register and monitor telephony state changes.
  - Get subscriber information.

Making the Outgoing Call

- The simplest way to make an outgoing call is to invoke the Dialer application by using the Intent.ACTION_CALL action.
- The sample code for doing so follows:

```
Intent callIntent = new Intent(Intent.ACTION_CALL);
callIntent.setData(Uri.parse("tel:1111122222"));
startActivity(callIntent);
```

- The preceding code initiates a phone call to 1111122222 using the system in-call Activity.
- For making the phone call, the application must have the permission to invoke the Dialerapplication. That is, to use this action, the application must request the CALL_PHONE uses-permission by adding the following statement to the manifest file:

```
<uses-permission android:name="android.permission.CALL_PHONE"/>
```

Listening for Phone State Changes

- To listen for phone state changes, that is, to see when the phone state is changed from idle to ringing, off-hook, and so on, we have to implement a broadcast receiver on android.intent.action.PHONE_STATE.
- That is, we need to implement a PhoneStateListener and call the listen() method of the TelephonyManager to receive notification whenever there is a change in the phone state.
- When a phone state changes, the onCallStateChanged() method of PhoneStateListener is called with the new phone state.
- The phone state is represented by the constants shown here:
  - **CALL_STATE_IDLE**—The phone is in an idle state.
  - **CALL_STATE_RINGING**—A phone call has arrived.
  - **CALL_STATE_OFFHOOK**—The phone is off-hook.
- To access the phone state information, the application must have permission for doing so. Add to the manifest file:

```
<uses-permission
   android:name="android.permission.READ_PHONE_STATE" />
```

Listing 11.17. Code Written into `activity_phone_call_app.xml`

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >
  <Button
    android:id="@+id/callbutton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Make Phone Call"
    android:layout_gravity="center" />
  <TextView
    android:id="@+id/messageview"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_gravity="center" />
</LinearLayout>
```

Listing 11.18. Code Written into `PhoneCallAppActivity.java`

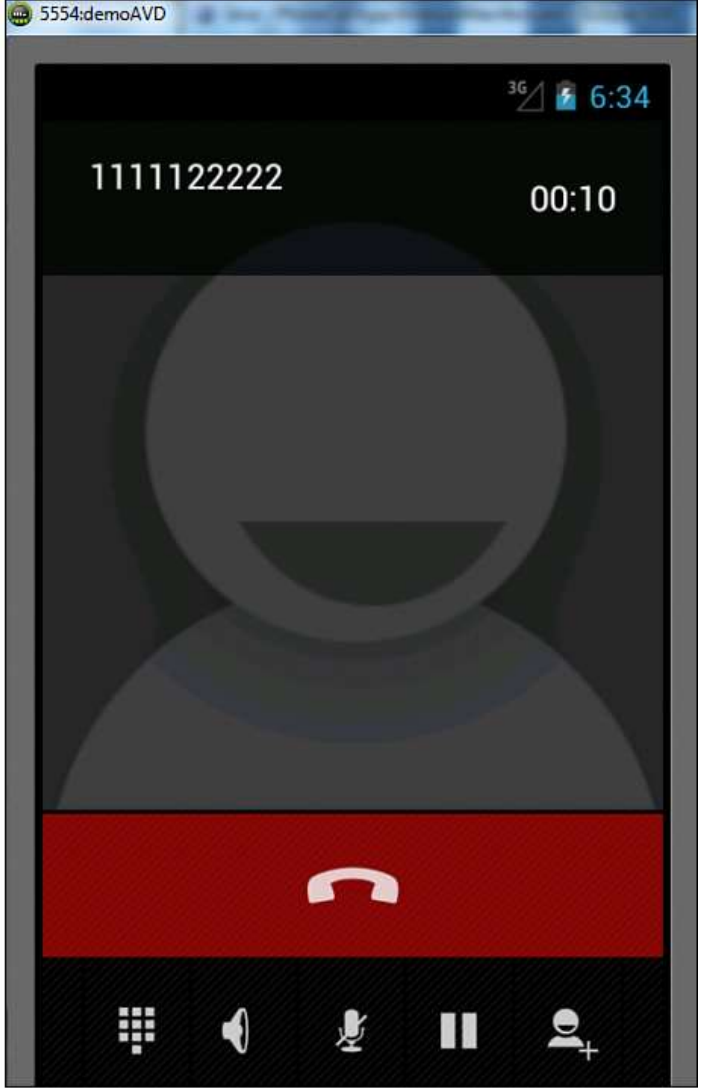
```
package com.androidunleashed.phonecallapp;
import android.app.Activity;
import android.os.Bundle;
import android.content.Context;
import android.content.Intent;
import android.net.Uri;
import android.telephony.PhoneStateListener;
import android.telephony.TelephonyManager;
import android.view.View;
import android.widget.TextView;
import android.widget.Button;
import android.view.View.OnClickListener;
import android.util.Log;

public class PhoneCallAppActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_phone_call_app);

        MyPhoneCallListener phoneListener = new MyPhoneCallListener();
         TelephonyManager telephonyManager = (TelephonyManager)
getSystemService(Context.TELEPHONY_SERVICE);
         telephonyManager.listen(phoneListener,PhoneStateListener.LISTEN_CALL_STATE
);
        Button callButton = (Button) findViewById(R.id.callbutton);
        callButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View arg0) {
                Intent callIntent = new Intent(Intent.ACTION_CALL);
                  callIntent.setData(Uri.parse("tel:1111122222"));
                  startActivity(callIntent);
            }
        });
    }
```

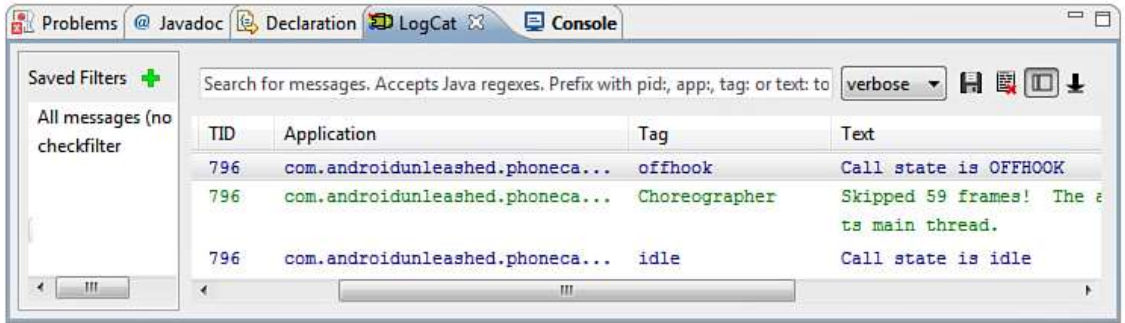
```
class MyPhoneCallListener extends PhoneStateListener {
   TextView messageview = (TextView)findViewById(R.id.messageview);
   String msg;

   @Override
   public void onCallStateChanged(int state, String incomingNumber) {
     super.onCallStateChanged(state, incomingNumber);
     switch(state){
       case TelephonyManager.CALL_STATE_IDLE:
         msg= "Call state is idle";
         Log.d("idle", msg);
         break;
       case TelephonyManager.CALL_STATE_RINGING:
         msg = "Call state is Ringing. Number is "+ incomingNumber;
         Log.d("ringing", msg);
         break;
       case TelephonyManager.CALL_STATE_OFFHOOK:
         msg = "Call state is OFFHOOK";
         Log.d("offhook", msg);
         break;
       default:
         msg = "Call state is" + state + ". Number is " + incomingNumber;
         Log.d("state", msg);
         break;
     }
     messageview.setText(msg);
   }
  }
}
```

Listing 11.19. Code in `AndroidManifest.xml`

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.androidunleashed.phonecallapp"
  android:versionCode="1"
  android:versionName="1.0" >
  <uses-sdk android:minSdkVersion="8"
    android:targetSdkVersion="15" />
  <uses-permission android:name="android.permission.CALL_PHONE" />
  <uses-permission android:name="android.permission.READ_PHONE_STATE" />
  <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name=".PhoneCallAppActivity"
      android:label="@string/title_activity_phone_call_app" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```


5554:demoAVD
6:33
PhoneCallAppActivity
Make Phone Call
Call state is idle


5554:demoAVD
6:34
1111122222
00:10


Problems
Javadoc
Declaration
LogCat
Console
Saved Filters
All messages (no checkfilter
Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to
verbose
TID
Application
Tag
Text
796
com.androidunleashed.phoneca...
offhook
Call state is OFFHOOK
796
com.androidunleashed.phoneca...
Choreographer
Skipped 59 frames! The a
ts main thread.
796
com.androidunleashed.phoneca...
idle
Call state is idle