

Laboratory Manual

SDL-II Mobile Application Development (Android)

For

Third year Students (CSE)

FOREWORD

It is my great pleasure to present this laboratory manual for Third year engineering students for the subject of SD1-II Mobile Application Development (Android).

As a student, many of you may be wondering with some of the questions in your mind regarding the subject and exactly what has been tried is to answer through this manual.

As you may be aware that MGM has already been awarded with ISO 9001:2000 certification and it is our endure to technically equip our students taking the advantage of the procedural aspects of ISO 9001:2000 Certification.

Faculty members are also advised that covering these aspects in initial stage itself, will greatly relieve them in future as much of the load will be taken care by the enthusiasm energies of the students once they are conceptually clear.

LABORATORY MANUAL CONTENTS

This manual is intended for the third year students of Computer Science and Engineering in the subject of SDL-II Mobile Application Development (Android). This manual typically contains practical/lab sessions related Android implemented in java-eclipse covering various aspects related the subject to enhanced understanding.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Good Luck for your Enjoyable Laboratory Sessions

Vision of CSE Department:

To develop computer engineers with necessary analytical ability and human values who can creatively design, implement a wide spectrum of computer systems for welfare of the society.

Mission of the CSE Department:

- I. Preparing graduates to work on multidisciplinary platforms associated with their professional position both independently and in a team environment.
- II. Preparing graduates for higher education and research in computer science and engineering enabling them to develop systems for society development.

Programme Educational Objectives:

Graduates will be able to

- I. To analyze, design and provide optimal solution for Computer Science & Engineering and multidisciplinary problems.
- II. To pursue higher studies and research by applying knowledge of mathematics and fundamentals of computer science.
- III. To exhibit professionalism, communication skills and adapt to current trends by engaging in lifelong learning.

Programme Outcomes (POs):

Engineering Graduates will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

LAB INDEX

1. Introduction to android operating system and study of basic widgets.
2. Study of android lifecycle and demonstration of it.
3. Study of intents and types of intents
4. Study of list views and adapters
5. Study of dialog interfaces in android
6. Study of Sensors in android
7. Study of Services in android
8. Study of touch in android
9. Study of android database (SQLite)
10. Mini Project

DOs and DON'Ts in Laboratory:

1. Make entry in the Log Book as soon as you enter the Laboratory.
2. All the students should sit according to their roll numbers starting from their left to right.
3. All the students are supposed to enter the terminal number in the log book.
4. Do not change the terminal on which you are working.
5. All the students are expected to get at least the algorithm of the program/concept to be implement.
6. Strictly follow the instructions given by the teacher/Lab Instructor.

Instruction for Laboratory Teachers

1. Submission related to whatever lab work has been completed should be done during the next lab session.
2. The immediate arrangements for printouts related to submission on the day of practical assignments.

3. Students should be taught for taking the printouts under the observation of lab teacher.
4. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students.

Evaluation and marking system:

Basic honesty in the evaluation and marking system is absolutely essential and in the process impartial nature of the evaluator is required in the examination system to become popular amongst the students. It is a wrong approach or concept to award the students by way of easy marking to get cheap popularity among the students to which they do not deserve. It is a primary responsibility of the teacher that right students who are really putting up lot of hard work with right kind of intelligence are correctly awarded.

The marking patterns should be justifiable to the students without any ambiguity and teacher should see that students are faced with unjust circumstances.

The assessment is done according to the directives of the Principal/ Vice-Principal/ Dean Academics.

Assignment 1

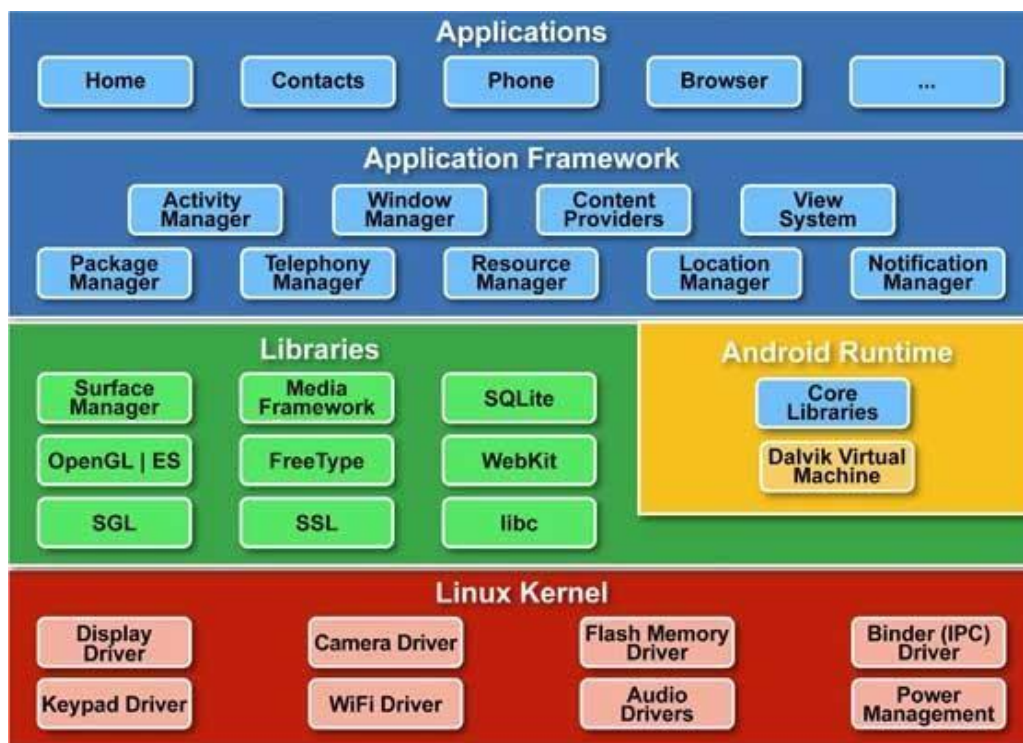
Aim: Introduction to android

Objective: Student should get the knowledge of android operating system background.

Outcome: Student will be aware of the android operating system.

Android Architecture

Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram.



Linux kernel

At the bottom of the layers is Linux - Linux 2.6 with approximately 115 patches. This provides basic system functionality like process management, memory management, device management like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at such as networking and

a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

Libraries

On top of Linux kernel there is a set of libraries including open -source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

Android Runtime

This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called **Dalvik Virtual Machine** which is a kind of Java Virtual Machine specially designed and optimized for Android.

The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine.

The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

Application Framework

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

Applications

You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games etc.

Android UI

An Android application user interface is everything that the user can see and interact with. You have learned about the various layouts that you can use to position your views in an activity. This chapter will give you detail on various views.

A **View** is an object that draws something on the screen that the user can interact with and a **ViewGroup** is an object that holds other View (and ViewGroup) objects in order to define the layout of the user interface.

You define your layout in an XML file which offers a human-readable structure for the layout, similar to HTML. For example, a simple vertical layout with a text view and a button looks like this:

```
<?xml version="1.0" encoding="utf-8"?>

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

    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am a TextView" />

    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am a Button" />
```

```
</LinearLayout>
```

Android UI Controls

There are number of UI controls provided by Android that allow you to build the graphical user interface for your app.

S.N.	UI Control & Description
1	TextView This control is used to display text to the user.
2	EditText EditText is a predefined subclass of TextView that includes rich editing capabilities.
3	AutoCompleteTextView The AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing.
4	Button A push-button that can be pressed, or clicked, by the user to perform an action.
5	ImageButton AbsoluteLayout enables you to specify the exact location of its children.
6	CheckBox An on/off switch that can be toggled by the user. You should use checkboxes when presenting users with a group of selectable options that are not mutually exclusive.
7	ToggleButton An on/off button with a light indicator.

8	RadioButton The RadioButton has two states: either checked or unchecked.
9	RadioGroup A RadioGroup is used to group together one or more RadioButtons.
10	ProgressBar The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.
11	Spinner A drop-down list that allows users to select one value from a set.
12	TimePicker The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode.
13	DatePicker The DatePicker view enables users to select a date of the day.

Create UI Controls

As explained in previous chapter, a view object may have a unique ID assigned to it which will identify the View uniquely within the tree. The syntax for an ID, inside an XML tag is:

```
android:id="@+id/text_id"
```

To create a UI Control/View/Widget you will have to define a view/widget in the layout file and assign it a unique ID as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical" >

<TextView android:id="@+id/text_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="I am a TextView" />
</LinearLayout>
```

Then finally create an instance of the Control object and capture it from the layout, use the following:

```
TextView myText = (TextView) findViewById(R.id.text_id);
```

Android Event Handling

Events are a useful way to collect data about a user's interaction with interactive components of your app, like button presses or screen touch etc. The Android framework maintains an event queue into which events are placed as they occur and then each event is removed from the queue on a first-in, first-out (FIFO) basis. You can capture these events in your program and take appropriate action as per requirements.

There are following three concepts related to Android Event Management:

- **Event Listeners:** The **View** class is mainly involved in building up a Android GUI, same View class provides a number of Event Listeners. The Event Listener is the object that receives notification when an event happens.

- **Event Listeners Registration:** Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.
- **Event Handlers:** When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

Event Listeners & Event Handlers

Event Handler	Event Listener & Description
onClick()	OnClickListener() This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. You will use onClick() event handler to handle such event.
onLongClick()	OnLongClickListener() This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event.
onFocusChange()	OnFocusChangeListener() This is called when the widget loses its focus i.e. user goes away from the view item. You will use onFocusChange() event handler to handle such event.
onKey()	OnFocusChangeListener() This is called when the user is focused on the item and presses or releases a hardware key on

	the device. You will use <code>onKey()</code> event handler to handle such event.
<code>onTouch()</code>	OnTouchListener() This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use <code>onTouch()</code> event handler to handle such event.
<code>onMenuItemClick()</code>	OnMenuItemClickListener() This is called when the user selects a menu item. You will use <code>onMenuItemClick()</code> event handler to handle such event.

There are many more event listeners available as a part of **View** class like `OnHoverListener`, `OnDragListener` etc which may be needed for your application. So I recommend to refer official documentation for Android application development in case you are going to develop a sophisticated apps.

Exercise:

Key points to study: History, Versions, Architecture, IDE (Eclipse), SDK of android.

Assignment 2

Aim: Study of UI in Android

Objective: Student should be able to design their own UI for android application using XML.

Outcome: Student will demonstrate the basic application using UI in android.

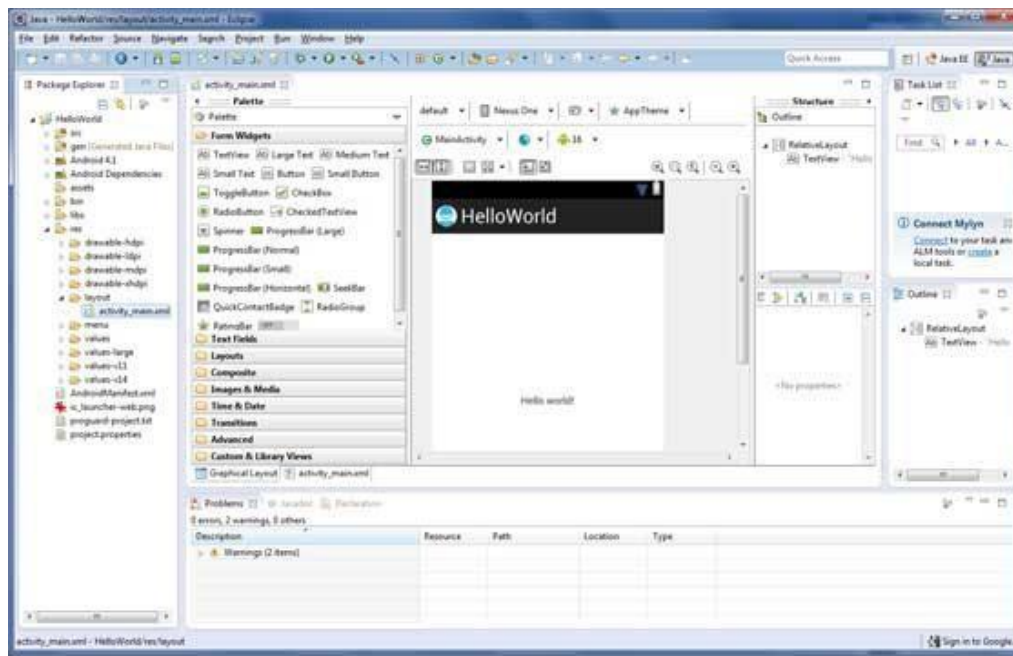
Create Android Application

The first step is to create a simple Android Application using Eclipse IDE. Follow the option

File → New → Project and finally select → Android New Application wizard from the wizard list. Now name your application as HelloWorld using the wizard window as follows:

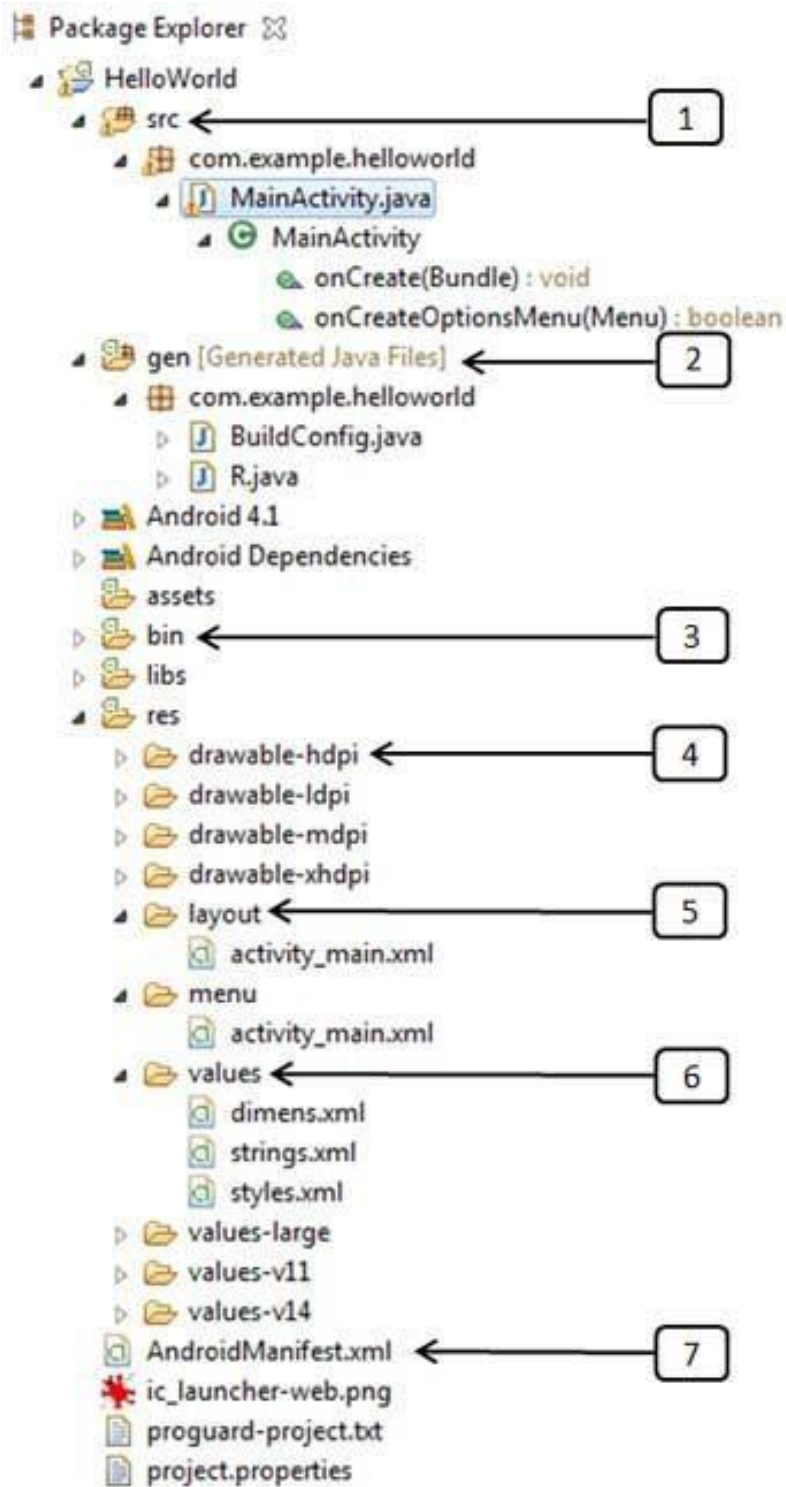
Next, follow the instructions provided and keep all other entries as default till the final step. Once your project is created successfully, you will have following project screen:





Anatomy of Android Application

Before you run your app, you should be aware of a few directories and files in the Android project:



S.N.	Folder, File & Description
1	src This contains the .java source files for your project. By default, it

	includes an <i>MainActivity.java</i> source file having an activity class that runs when your app is launched using the app icon.
2	gen This contains the .R file, a compiler-generated file that references all the resources found in your project. You should not modify this file.
3	bin This folder contains the Android package files .apk built by the ADT during the build process and everything else needed to run an Android application.
4	res/drawable-hdpi This is a directory for drawable objects that are designed for high-density screens.
5	res/layout This is a directory for files that define your app's user interface.
6	res/values This is a directory for other various XML files that contain a collection of resources, such as strings and colors definitions.
7	AndroidManifest.xml This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.

Following section will give a brief overview few of the important application files.

The Main Activity File

The main activity code is a Java file *MainActivity.java*. This is the actual application file which ultimately gets converted to a Dalvik executable and runs

your application. Following is the default code generated by the application wizard for *Hello World!* application:

```
package com.example.helloworld;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.MenuItem;
import android.support.v4.app.NavUtils;
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }
}
```

Here, *R.layout.activity_main* refers to the *activity_main.xml* file located in the *res/layout* folder. The *onCreate()* method is one of many methods that are fired when an activity is loaded.

The Manifest File

Whatever component you develop as a part of your application, you must declare all its components in a *manifest* file called *AndroidManifest.xml* which resides at the root of the application project directory. This file works as an interface between Android OS and your application, so if you do not declare your component in this file, then it will not be considered by the OS. For example, a default manifest file will look like as following file:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.helloworld"
    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=".MainActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>

```

Here `<application>...</application>` tags enclosed the components related to the application. Attribute `android:icon` will point to the application icon available under `res/drawable-hdpi`. The application uses the image named `ic_launcher.png` located in the drawable folders

The `<activity>` tag is used to specify an activity and `android:name` attribute specifies the fully qualified class name of the *Activity* subclass and the `android:label` attributes specifies a string to use as the label for the activity. You can specify multiple activities using `<activity>` tags.

The action for the intent filter is named `android.intent.action.MAIN` to indicate that this activity serves as the entry point for the application. The category for the intent-filter is named `android.intent.category.LAUNCHER` to indicate that the application can be launched from the device's launcher icon.

The `@string` refers to the `strings.xml` file explained below. Hence, `@string/app_name` refers to the `app_name` string defined in the `strings.xml` file, which is "HelloWorld". Similar way, other strings get populated in the application.

Following is the list of tags which you will use in your manifest file to specify different Android application components:

`<activity>` elements for activities

<service> elements for services

<receiver> elements for broadcast receivers

<provider> elements for content providers

The Strings File

The strings.xml file is located in the *res/values* folder and it contains all the text that your application uses. For example, the names of buttons, labels, default text, and similar types of strings go into this file. This file is responsible for their textual content. For example, a default strings file will look like as followingfile:

```
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_main">MainActivity</string>
</resources>
```

The R File

The gen/com.example.helloworld/R.java file is the glue between the activityJava files like *MainActivity.java* and the resources like *strings.xml*. It is an automatically generated file and you should not modify the content of the R.java file. Following is a sample of R.java file:

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found. It
 * should not be modified by hand.
 */

package com.example.helloworld;

public final class R {
    public static final class attr {
    }
    public static final class dimen {
        public static final int padding_large=0x7f040002;
        public static final int padding_medium=0x7f040001;
        public static final int padding_small=0x7f040000;
    }
}
```

```

public static final class drawable {
    public static final int ic_action_search=0x7f020000;
    public static final int ic_launcher=0x7f020001;
}
public static final class id {
    public static final int menu_settings=0x7f080000;
}
public static final class layout {
    public static final int activity_main=0x7f030000;
}
public static final class menu {
    public static final int activity_main=0x7f070000;
}
public static final class string {
    public static final int app_name=0x7f050000;
    public static final int hello_world=0x7f050001;
    public static final int menu_settings=0x7f050002;
    public static final int title_activity_main=0x7f050003;
}
public static final class style {
    public static final int AppTheme=0x7f060000;
}
}

```

The Layout File

The `activity_main.xml` is a layout file available in `res/layout` directory, that is referenced by your application when building its interface. You will modify this file very frequently to change the layout of your application. For your "Hello World!" application, this file will have following content related to default layout:

```

<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" >


<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:padding="@dimen/padding_medium"
    android:text="@string/hello_world"
    tools:context=".MainActivity" />

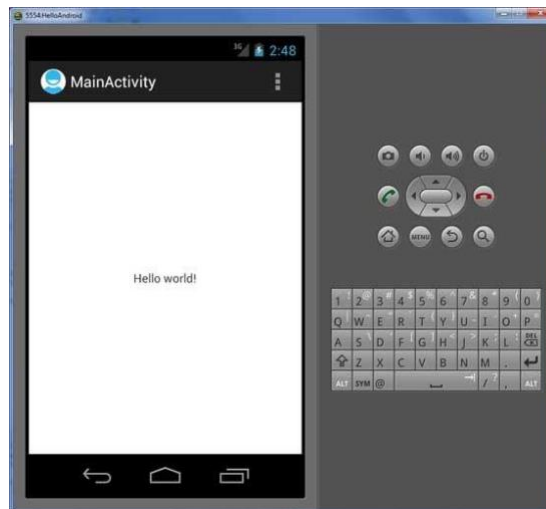
```

</RelativeLayout>

This is an example of simple *RelativeLayout* which we will study in a separate chapter. The *TextView* is an Android control used to build the GUI and it has various attributes like *android:layout_width*, *android:layout_height* etc which are being used to set its width and height etc. The *@string* refers to the *strings.xml* file located in the *res/values* folder. Hence, *@string/hello_world* refers to the hello string defined in the *strings.xml* file, which is "Hello World!".

Running the Application

Let's try to run our Hello World! application we just created. I assume you had created your AVD while doing environment setup. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Eclipse installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window:



Exercise:

Key points to study: Package explorer (Description about folders e.g. *src*, *gen*, *res*, etc.), Description about XML file, Layouts.

1. Develop an application Hello world program description about tags used in XML

2. Develop an application using basic controls like button, textview, edittext etc. e.g. application for calculator.

Exercise 3

Aim: Study of Activity LifeCycle

Objective: Student should get the knowledge of android activity lifecycle.

Outcome: Student will demonstrate the Activity lifecycle through the application.

Activity life cycle

States of an activity

An activity can be in different states depending how it is interacting with the user. These states are described by the following table.

Table 1. Activity state

State	Description
Running	Activity is visible and interacts with the user.
Paused	Activity is still visible but partially obscured, instance is running but might be killed by the system.
Stopped	Activity is not visible, instance is running but might be killed by the system.
Killed	Activity has been terminated by the system or by a call to its <code>finish()</code> method.

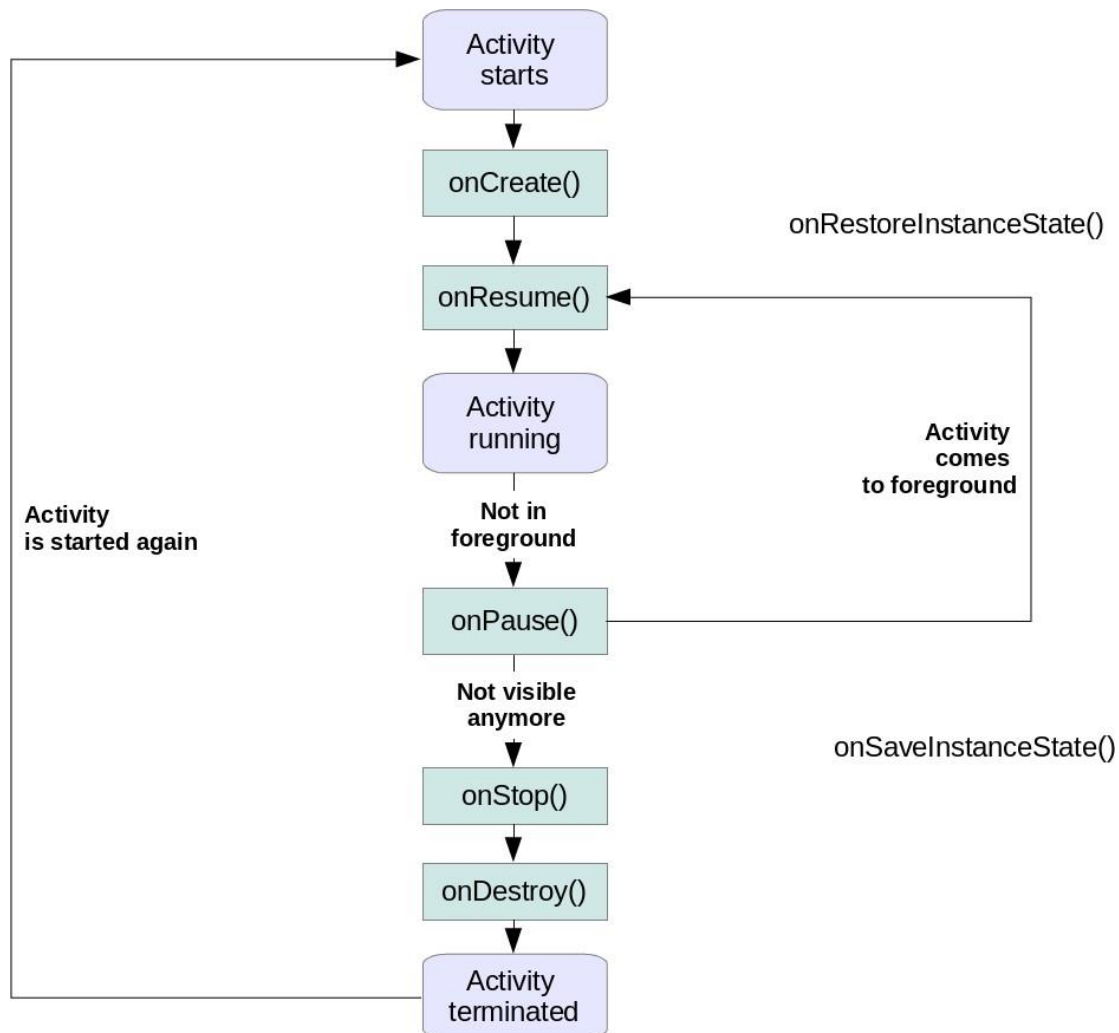
The live cycle methods

The Android system defines a life-cycle for activities via predefined life-cycle methods. The most important methods are:

Table 3. Important Activity lifecycle methods

Method	Purpose
onCreate()	Called then the activity is created. Used to initialize the activity, for example create the user interface.
onResume()	Called if the <i>activity</i> get visible again and the user starts interacting with the activity again. Used to initialize fields, register listeners, bind to services, etc.
onPause()	Called once another activity gets into the foreground. Always called before the <i>activity</i> is not visible anymore. Used to release resources or save application data. For example you unregister listeners, intent receivers, unbind from services or remove system service listeners.
onStop()	Called once the activity is no longer visible. Time or CPU intensive shut-down operations, such as writing information to a database should be down in the <code>onStop()</code> method. This method is guaranteed to be called as of API 11.

The life cycle of an activity with its most important methods is displayed in the following diagram.



Android has more life cycle methods but not all of these methods are guaranteed to be called. The `onDestroy()` method is not guaranteed to be called, hence you typically do not use it.

Activity instance state

Instance state of an activity which is required to restore the activity to the state in which the user left it. This is non-persistent application data that needs to be passed between activities restarts during a configuration change to restore user selections. The application is responsible for restoring its instance state.

Assume for example the user scrolled through a ListView with thousands of items and the activity is recreated. Losing the position in the list is annoying for the user, hence the position should be restored.

The `onSaveInstanceState()` can be used to store this instance state as a Bundle. A Bundle can contain primitive data types, arrays, String and objects which are of the Parcelable or Serializable type.

The persisted Bundle data is passed at restart of the activity to the `onCreate()` method and `onRestoreInstanceState()` as parameter.

If you override `onSaveInstanceState()` and `onRestoreInstanceState()` you should call the super implementation of it, because the default views of Android store their data via a call to `View.onSaveInstanceState` from the `onSaveInstanceState()` method of the activity. For example EditText stores its content via the default call of this method.

The `onRestoreInstanceState()` or the `onCreate()` methods can be used to recreate the instance scope of an activity if it is restarted.

Exercise:

Key point to study: Description about Activity LifeCycle

Develop an application demonstrating Activity LifeCycle

Assignment 4

Aim: Study of Intents in android

Objective: Student should know what is intent in android and what are the types of it.

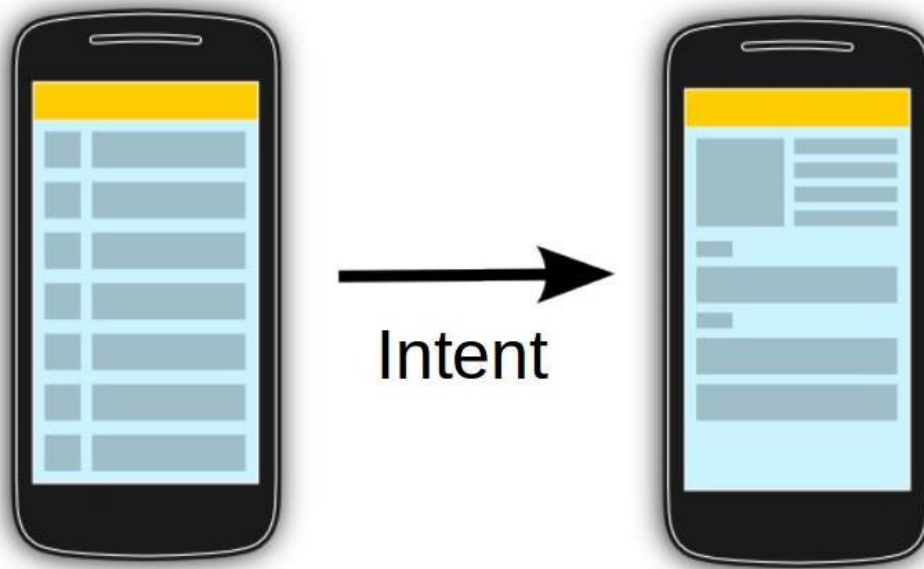
Outcome: Student will develop a good application using the intents and its properties efficiently

Intents and intent filter

What are intents?

Intents are asynchronous messages which allow application components to request functionality from other Android components. Intents allow you to interact with components from the same applications as well as with components contributed by other applications. For example, an activity can start an external activity for taking a picture.

Intents are objects of the `android.content.Intent` type. Your code can send them to the Android system defining the components you are targeting. For example, via the `startActivity()` method you can define that the intent should be used to start an activity. An intent can contain data via a `Bundle`. This data can be used by the receiving component. Starting activities. To start an activity, use the method `startActivity(intent)`. This method is defined on the `Context` object which activity extends.



The following code demonstrates how you can start another activity via an intent.

```
# Start the activity connect to the  
# specified class  
  
Intent i = new Intent(this, ActivityTwo.class);  
startActivity(i);
```

Sub-activities

Activities which are started by other Android activities are called *sub-activities*. This wording makes it easier to describe which activity is meant.

Starting services

You can also start services via intents. Use the `startService(Intent)` method call for that.

Different types of intents

Android supports explicit and implicit intents. An application can define the target component directly in the intent (*explicit intent*) or ask the Android system to evaluate registered components based on the intent data (*implicit intents*).

Explicit Intents

Explicit intents explicitly define the component which should be called by the Android system, by using the Java class as identifier.

The following shows how to create an explicit intent and send it to the Android system. If the class specified in the intent represents an activity, the Android system starts it.

```
Intent i = new Intent(this, ActivityTwo.class);  
i.putExtra("Value1", "This value one for ActivityTwo ");  
i.putExtra("Value2", "This value two ActivityTwo");
```

Explicit intents are typically used within an application as the classes in an application are controlled by the application developer.

Implicit Intents

Implicit intents specify the action which should be performed and optionally data which provides content for the action.

For example, the following tells the Android system to view a webpage. All installed web browsers should be registered to the corresponding intent data via an intent filter.

```
Intent i = new Intent(Intent.ACTION_VIEW,  
Uri.parse("http://www.exercise.com"));
```

```
startActivity(i);
```

If an implicit intent is sent to the Android system, it searches for all components which are registered for the specific action and the fitting data type.

If only one component is found, Android starts this component directly. If several components are identified by the Android system, the user will get a selection dialog and can decide which component should be used for the intent.

Data transfer between activities

Data transfer to the target component: An intent contains certain header data, e.g., the desired action, the type, etc. Optionally an intent can also contain additional data based on an instance of the Bundle class which can be retrieved from the intent via the `getExtras()` method.

You can also add data directly to the Bundle via the overloaded `putExtra()` methods of the Intent objects. Extras are key/value pairs. The key is always of type String. As value you can use the primitive data types (int, float, ...) plus objects of type String, Bundle, Parcelable and Serializable.

The receiving component can access this information via the `getAction()` and `getData()` methods on the Intent object. This Intent object can be retrieved via the `getIntent()` method.

The component which receives the intent can use the `getIntent().getExtras()` method call to get the extra data. That is demonstrated in the following code snippet.

```
Bundle extras = getIntent().getExtras();  
if (extras == null) {  
    return;  
}
```



```
// get data via the key
String value1 = extras.getString(Intent.EXTRA_TEXT);
if (value1 != null) {
    // do something with the data
}
```

Example: Using the share intent

Lots of Android applications allow you to share some data with other people, e.g., the Facebook, G+, Gmail and Twitter application. You can send data to one of these components. The following code snippet demonstrates the usage of such an intent within your application.

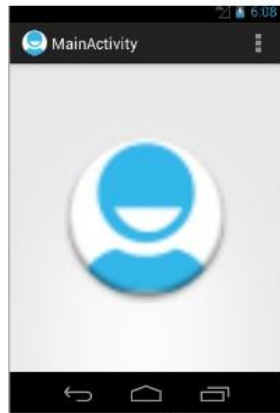
```
// this runs, for example, after a button click
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("text/plain");
intent.putExtra(android.content.Intent.EXTRA_TEXT, "News for you!");
startActivity(intent);
```

Retrieving result data from a sub-activity

An activity can be closed via the back button on the phone. In this case the `finish()` method is performed. If the activity was started with the `startActivity(Intent)` method call, the caller requires no result or feedback from the activity which now is closed.

If you start the activity with the `startActivityForResult()` method call, you expect feedback from the sub-activity. Once the sub-activity ends, the `onActivityResult()` method on the sub-activity is called and you can perform actions based on the result.

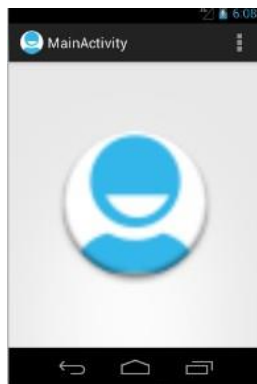
In the `startActivityForResult()` method call you can specify a result code to determine which activity you started. This result code is returned to you. The started activity can also set a result code which the caller can use to determine if the activity was canceled or not.



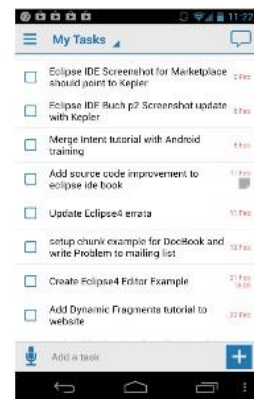
Intent resolution by the
Android system



One activity is
started



Intent + resultCode
provided by called
activity



`onActivityResult(requestCode, resultCode, intent)`

requestCode
provided by Android to
identify which activity
type was started

The sub-activity uses the `finish()` method to create a new intent and to put data into it. It also sets a result via the `setResult()` method call.

The following example code demonstrates how to trigger an intent with the `startActivityForResult()` method.

```

public void onClick(View view) {
    Intent i = new Intent(this, ActivityTwo.class);
    i.putExtra("Value1", "This value one for ActivityTwo ");
    i.putExtra("Value2", "This value two ActivityTwo");
    // set the request code to any code you like,
    // you can identify the callback via this code
    startActivityForResult(i, REQUEST_CODE);
}

```

If you use the `startActivityForResult()` method, then the started activity is called a sub-activity.

If the sub-activity is finished, it can send data back to its caller via an Intent. This is done in the `finish()` method.

@Override

```

public void finish() {
    // Prepare data intent
    Intent data = new Intent();
    data.putExtra("returnKey1", "Swinging on a star. ");
    data.putExtra("returnKey2", "You could be better then you are. ");
    // Activity finished ok, return the data
    setResult(RESULT_OK, data);
    super.finish();
}

```

Once the sub-activity finishes, the `onActivityResult()` method in the calling activity is called.

@Override

```

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

```

```
if (resultCode == RESULT_OK && requestCode == REQUEST_CODE) {  
    if (data.hasExtra("returnKey1")) {  
        Toast.makeText(this, data.getExtras().getString("returnKey1"),  
            Toast.LENGTH_SHORT).show();  
    }  
}  
}
```

Defining intent filters

Intent filter

Intents are used to signal to the Android system that a certain event has occurred. Intents often describe the action which should be performed and provide data upon which such an action should be done. For example, your application can start a browser component for a certain URL via an intent. This is demonstrated by the following example.

```
String url = "http://www.exercise.com";  
Intent i = new Intent(Intent.ACTION_VIEW);  
i.setData(Uri.parse(url));  
startActivity(i);
```

But how does the Android system identify the components which can react to a certain intent?

A component can register itself via an *intent filter* for a specific action and specific data. An intent filter specifies the types of intents to which an activity, service, or broadcast receiver can respond to by declaring the capabilities of a component.

Android components register intent filters either statically in the *AndroidManifest.xml* or in case of a broadcast receiver also dynamically via code. An intent filter is defined by its category, action and data filters. It can also contain additional meta-data.

If an intent is sent to the Android system, the Android platform runs a receiver determination. It uses the data included in the intent. If several components have registered for the same intent filter, the user can decide which component should be started.

Defining intent filter

You can register your Android components via intent filters for certain events. If a component does not define one, it can only be called by explicit intents. This chapter gives an example for registering a component for an intent. The key for this registration is that your component registers for the correct action, mime-type and specifies the correct meta-data.

If you send such an intent to your system, the Android system determines all registered Android components for this intent. If several components have registered for this intent, the user can select which one should be used.

Exercise:

Key points to study: Description about Intent. Types of Intents.

Develop an application which demonstrate the efficient use of explicit and implicit intents

Assignment 5

Aim: Study of Adapters and Views (List View)

Objective: Student should use the adapters and list views in their applications efficiently.

Outcome: Student will demonstrate the efficient use of List views and adapters through the application.

Android and Lists

Using lists in Android

The display of elements in a list is a very common pattern in mobile applications. The user sees a list of items and can scroll through them. Such an activity is depicted in the following picture.

Typically the user interacts with the list via the action bar, for example, via a refresh button. Individual list items can be selected. This selection can update the action bar or can trigger a detailed screen for the selection. The following graphic sketches that. On the selection of a list item another activity is started.

Views for handling lists

Android provides the `ListView` and the `ExpandableListView` classes which are capable of displaying a scrollable list of items.

The `ExpandableListView` class supports a grouping of items.

Possible input types for lists

The input to the list (items in the list) can be arbitrary Java objects. The adapter extracts the correct data from the data object and assigns this data to the views in the row of the `ListView`.

These items are typically called the *data model* of the list. An adapter can receive data as input.

Adapters

An adapter manages the data model and adapts it to the individual entries in the widget. An adapter extends the `BaseAdapter` class.

Every line in the widget displaying the data consists of a layout which can be as complex as you want. A typical line in a list has an image on the left side and two text lines in the middle as depicted in the following graphic.



A layout file for such a line might look like the following.

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:padding="6dip" >

    <ImageView
        android:id="@+id/icon"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_alignParentBottom="true"
        android:layout_alignParentTop="true"
        android:layout_marginRight="6dip"
        android:contentDescription="TODO"
        android:src="@drawable/ic_launcher" />

    <TextView
        android:id="@+id/secondLine"
        android:layout_width="fill_parent"
        android:layout_height="26dip"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_toRightOf="@id/icon"
        android:ellipsize="marquee"
```

```
        android:singleLine="true"
        android:text="Description"
        android:textSize="12sp" />

<TextView
    android:id="@+id/firstLine"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_above="@id/secondLine"
    android:layout_alignParentRight="true"
    android:layout_alignParentTop="true"
    android:layout_alignWithParentIfMissing="true"
    android:layout_toRightOf="@id/icon"
    android:gravity="center_vertical"
    android:text="Example application"
    android:textSize="16sp" />

</RelativeLayout>
```

The adapter would inflate the layout for each row in its `getView()` method and assign the data to the individual views in the row.

The adapter is assigned to the `ListView` via the `setAdapter` method on the `ListView` object.

Filtering and sorting

Filtering and sorting of the data is handled by the adapter. You need to implement the logic in your custom adapter implementation.

Data updates in the adapter

The `notifyDataSetChanged()` method on the adapter is called if the data has changed or if new data is available.

The `notifyDataSetInvalidated()` method is called if the data is not available anymore.

Listener

To react to selections in the list, set an `OnItemClickListener` to your `ListView`.

```
listView.setOnItemClickListener(new OnItemClickListener() {  
    @Override  
    public void onItemClick(AdapterView<?> parent, View view,  
        int position, long id) {  
        Toast.makeText(getApplicationContext(),  
            "Click ListItem Number " + position, Toast.LENGTH_LONG)  
            .show();  
    }  
});
```

Default adapter

Default platform adapter

Android provides default adapter implementations; the most important are `ArrayAdapter` and `CursorAdapter`.

`ArrayAdapter` can handle data based on Arrays or `java.util.List`.

`SimpleCursorAdapter` can handle database related data.

Using ArrayAdapter

The `ArrayAdapter` class can handle a list or array of Java objects as input. Every Java object is mapped to one row. By default, it maps the `toString()` method of the object to a view in the row layout.

You can define the ID of the view in the constructor of the `ArrayAdapter` otherwise the `android.R.id.text1` ID is used as default.

The `ArrayAdapter` class allows to remove all elements in its underlying data structure with the `clear()` method call. You can then add new elements via the `add()` method or a Collection via the `addAll()` method.

You can also directly modify the underlying data structure and call the `notifyDataSetChanged()` method on the adapter to notify it about the changes in data.

Warning

If you want to change the data in your adapter, the underlying data structure must support this operation. This is, for example, the case for the ArrayList class, but not for arrays.

ListView example with ArrayAdapter

The following listing shows a layout file called `activity_listviewexampleactivity.xml` which includes a ListView.

```
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/listview"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

The following example shows the usage of the ListView view in an activity. It uses a default layout from the Android platform for the row layout. It also demonstrates the removal of list items and uses animations for the removal.

```
package com.exerciseneme.android.listview.withanimation;

public class ListViewExampleActivity extends Activity {

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

        final ListView listview = (ListView) findViewById(R.id.listview);
        String[] values = new String[] { "Android", "iPhone", "WindowsMobile",
            "Blackberry", "WebOS", "Ubuntu", "Windows7", "Max OS X",
            "Linux", "OS/2", "Ubuntu", "Windows7", "Max OS X", "Linux",
            "OS/2", "Ubuntu", "Windows7", "Max OS X", "Linux", "OS/2",
            "Android", "iPhone", "WindowsMobile" };

        final ArrayList<String> list = new ArrayList<String>();
```

```

for (int i = 0; i < values.length; ++i) {
    list.add(values[i]);
}

final StableArrayAdapter adapter = new StableArrayAdapter(this,
    android.R.layout.simple_list_item_1, list);
listview.setAdapter(adapter);

listview.setOnItemClickListener(new AdapterView.OnItemClickListener() {

    @Override
    public void onItemClick(AdapterView<?> parent, final View view,
        int position, long id) {
        final String item = (String) parent.getItemAtPosition(position);
        view.animate().setDuration(2000).alpha(0)
            .withEndAction(new Runnable() {
                @Override
                public void run() {
                    list.remove(item);
                    adapter.notifyDataSetChanged();
                    view.setAlpha(1);
                }
            });
    }

});

}

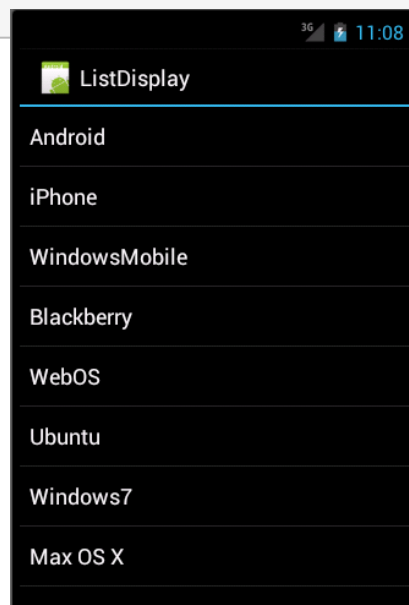
private class StableArrayAdapter extends ArrayAdapter<String> {

    HashMap<String, Integer> mIdMap = new HashMap<String, Integer>();

    public StableArrayAdapter(Context context, int textViewResourceId,
        List<String> objects) {
        super(context, textViewResourceId, objects);
        for (int i = 0; i < objects.size(); ++i) {
            mIdMap.put(objects.get(i), i);
        }
    }
}

```

```
}  
@Override  
public long getItemId(int position) {  
    String item = getItem(position);  
    return mIdMap.get(item);  
}  
@Override  
public boolean hasStableIds() {  
    return true;  
}  
}  
}
```



Exercise:

Key points to study: Description of List View and Types of adapters

Develop an application for the demonstrate ListView creation.

Assignment 6

Aim: Study of Dialogs in android

Objective: Student should understand the concept of dialogs and alert dialogs in android.

Outcome: Student will demonstrate the use of dialogs in applications

Android Dialogs

Using dialogs in Android

You can open dialogs from your activity via the `showDialog(int)` method. Dialogs which are created via an activity are bound to this activity. A dialog gets the focus until the user closes it.

The `Dialog` class is the base class for dialogs. Typically you use one of its subclasses, e.g., `AlertDialog`, `ProgressDialog`, `DatePickerDialog` or `TimePickerDialog`.

```
// constant for identifying the dialog  
private static final int DIALOG_ALERT = 10;  
  
public void onClick(View view) {  
    showDialog(DIALOG_ALERT);  
}
```

If the dialog is displayed, the Android system calls the `onCreateDialog(int)` method. In this method you instantiate the correct dialog based on the input parameter. You should always create a dialog from the `onCreateDialog(int)` method as in this case the Android system manages the dialog for you.

```
@Override  
protected Dialog onCreateDialog(int id) {  
    switch (id) {  
        case DIALOG_ALERT:  
            Builder builder = new AlertDialog.Builder(this);
```

```

        builder.setMessage("This will end the activity");
        builder.setCancelable(true);
        builder.setPositiveButton("I agree", new OnClickListener());
        builder.setNegativeButton("No, no", new CancelOnClickListener());
        AlertDialog dialog = builder.create();
        dialog.show();
    }
    return super.onCreateDialog(id);
}

private final class CancelOnClickListener implements
    DialogInterface.OnClickListener {
    public void onClick(DialogInterface dialog, int which) {
        Toast.makeText(getApplicationContext(), "Activity will continue",
            Toast.LENGTH_LONG).show();
    }
}

private final class OkOnClickListener implements
    DialogInterface.OnClickListener {
    public void onClick(DialogInterface dialog, int which) {
        AlertExampleActivity.this.finish();
    }
}

```

onCreateDialog(int) is only called the first time if you want to later influence the dialog to use the optional onPrepareDialog(int, Dialog) method.

ProgressDialog

Android also provides a ProgressDialog, which can be opened via a ProgressDialog.open() method call.

Custom dialogs

If you want to create your custom dialogs, you create a layout file for the dialog. This layout file is assigned to the dialog via setContentView() method.

You would then use the `dialog.findViewById()` to find the elements in your layout and assign values to it.

The title of the dialog can be set via the `setTitle()` method.

Exercise:

Develop an application and Displaying an alert dialog

Adding a dialog to your activity

The following demonstrates the usage of the `AlertDialog` dialog in an existing activity. An instance of this class can be created by the builder pattern, e.g., you can chain your method calls.

Ensure that the layout file of your activity contains a button with the `android:onClick` pointing to a method called `onClick`.

Change the code of your activity to the following.

```
// constant for identifying the dialog
private static final int DIALOG_ALERT = 10;

// existing code.....

// adjust this method if you have more than
// one button pointing to this method
public void onClick(View view) {
    showDialog(DIALOG_ALERT);
}

@Override
protected Dialog onCreateDialog(int id) {
    switch (id) {
        case DIALOG_ALERT:
            Builder builder = new AlertDialog.Builder(this);
            builder.setMessage("This ends the activity");
            builder.setCancelable(true);
            builder.setPositiveButton("I agree", new OnClickListener());
            builder.setNegativeButton("No, no", new OnClickListener());
```

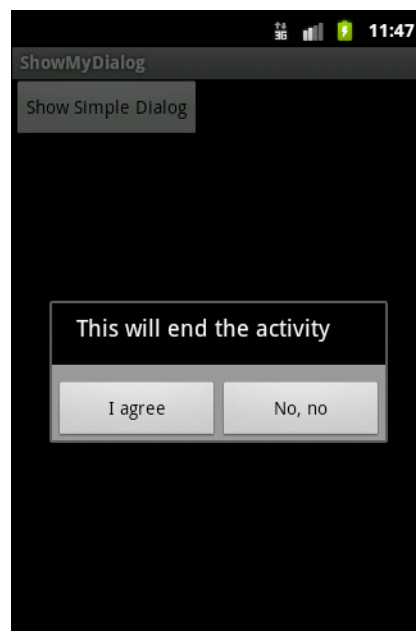
```
        AlertDialog dialog = builder.create();
        dialog.show();
    }
    return super.onCreateDialog(id);
}

private final class CancelOnClickListener implements
    DialogInterface.OnClickListener {
    public void onClick(DialogInterface dialog, int which) {
        Toast.makeText(getApplicationContext(), "Cancle selected, activity continues",
            Toast.LENGTH_LONG).show();
    }
}

private final class OkOnClickListener implements
    DialogInterface.OnClickListener {
    public void onClick(DialogInterface dialog, int which) {
        AlertExampleActivity.this.finish();
    }
}
```

Test dialog usage

If you run your application and click the corresponding button, your dialog is displayed.



Assignment 7

Aim: Developing application for TextToSpeech Converter

Objective: Student should learn the advanced application development.

Outcome: Student will develop the TextToSpeech converter in android.

TextToSpeech Converter

Android allows you convert your text into voice. Not only you can convert it but it also allows you to speak text in variety of different languages.

Android provides TextToSpeech class for this purpose. In order to use this class, you need to instantiate an object of this class and also specify the initListnere. Its syntax is given below:

```
private EditText write;

ttobj=new TextToSpeech(getApplicationContext(), new
TextToSpeech.OnInitListener() {

    @Override

    public void onInit(int status) {

        }

    }

});
```

In this listener , you have to specify the properties for TextToSpeech object , such as its language ,pitch e.t.c. Language can be set by calling setLanguage() method. Its syntax is given below:

```
ttobj.setLanguage(Locale.UK);
```

The method `setLanguage` takes an `Locale` object as parameter. The list of some of the locales available are given below:

Sr.No	Locale
1	US
2	CANADA_FRENCH
3	GERMANY
4	ITALY
5	JAPAN
6	CHINA

Once you have set the language, you can call `speak` method of the class to speak the text. Its syntax is given below:

```
ttobj.speak(toSpeak, TextToSpeech.QUEUE_FLUSH, null);
```

Apart from the `speak` method, there are some other methods available in the `TextToSpeech` class. They are listed below:

Sr.No	Method & description
1	<code>addSpeech(String text, String filename)</code> This method adds a mapping between a string of text and a sound file.
2	<code>getLanguage()</code> This method returns a <code>Locale</code> instance describing the language.

3	<code>isSpeaking()</code> This method checks whether the TextToSpeech engine is busy speaking.
4	<code>setPitch(float pitch)</code> This method sets the speech pitch for the TextToSpeech engine.
5	<code>setSpeechRate(float speechRate)</code> This method sets the speech rate.
6	<code>shutdown()</code> This method releases the resources used by the TextToSpeech engine.
7	<code>stop()</code> This method stop the speak.

Example

The below example demonstrates the use of TextToSpeech class. It crates a basic application that allows you to set write text and speak it.

To experiment with this example , you need to run this on an actual device.

Steps	Description
1	You will use Eclipse IDE to create an Android application and name it as TextToSpeech under a package <code>com.example.texttospeech</code> . While creating this project, make sure you Target SDK and Compile With at the latest version of Android SDK to use higher levels of APIs.

2	Modify src/MainActivity.java file to add TextToSpeech code.
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
4	Modify res/values/string.xml file and add necessary string components.
5	Run the application and choose a running android device and install the application on it and verify the results.

Exercise: Develop an application for TextToSpeech converter.

```
package com.example.texttospeech;

import java.util.Locale;
import java.util.Random;

import android.app.Activity;
import android.os.Bundle;
import android.speech.tts.TextToSpeech;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {

    TextToSpeech ttobj;
    private EditText write;
    @Override
```

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    write = (EditText)findViewById(R.id.editText1);
    ttobj=new TextToSpeech(getApplicationContext(),
    new TextToSpeech.OnInitListener() {
    @Override
    public void onInit(int status) {
        if(status != TextToSpeech.ERROR){
            ttobj.setLanguage(Locale.UK);
        }
    }
    });
}

@Override
public void onPause(){
    if(ttobj !=null){
        ttobj.stop();
        ttobj.shutdown();
    }
    super.onPause();
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

```

```

public void speakText(Viewview){
    String toSpeak = write.getText().toString();
    Toast.makeText(getApplicationContext(), toSpeak,
    Toast.LENGTH_SHORT).show();
    ttobj.speak(toSpeak, TextToSpeech.QUEUE_FLUSH, null);

}
}

```

Here is the content of activity_main.xml

```

<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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true"
    android:layout_marginBottom="188dp"
    android:layout_marginRight="67dp"
    android:onClick="speakText"

```

```
android:text="@string/text1" />
```

```
<EditText
```

```
    android:id="@+id/editText1"
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:layout_above="@+id/button1"
```

```
    android:layout_centerHorizontal="true"
```

```
    android:layout_marginBottom="81dp"
```

```
    android:ems="10" >
```

```
    <requestFocus />
```

```
</EditText>
```

```
<TextView
```

```
    android:id="@+id/textView1"
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:layout_alignParentTop="true"
```

```
    android:layout_centerHorizontal="true"
```

```
    android:layout_marginTop="20dp"
```

```
    android:text="@string/write"
```

```
    android:textAppearance="?android:attr/textAppearanceLarge" />
```

```
</RelativeLayout>
```

Here is the content of Strings.xml.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<resources>
```

```
    <string name="app_name">TextToSpeech</string>
```


```
    <string name="action_settings">Settings</string>
```

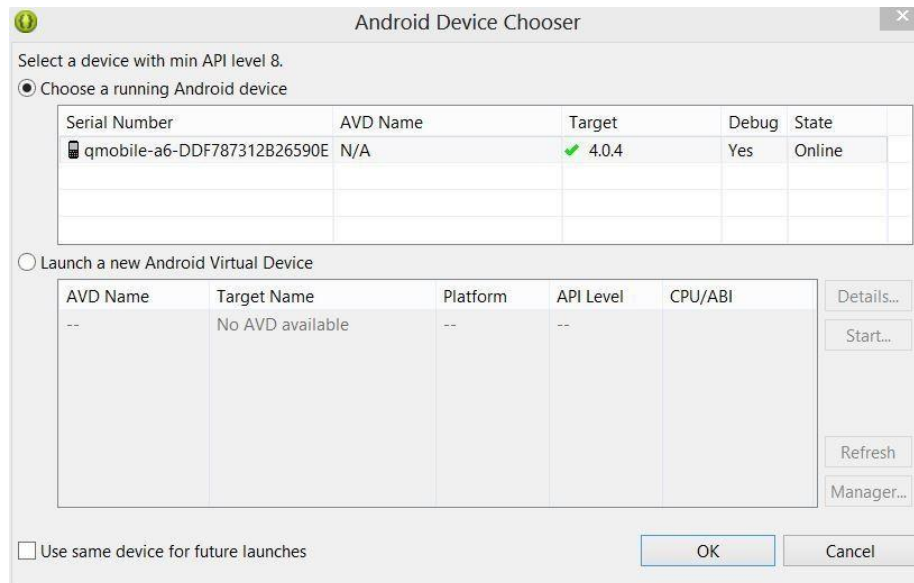


```
<string name="hello_world">Hello world!</string>
<string name="text1">Text to Speech</string>
<string name="write">Write Text</string>
</resources>
```

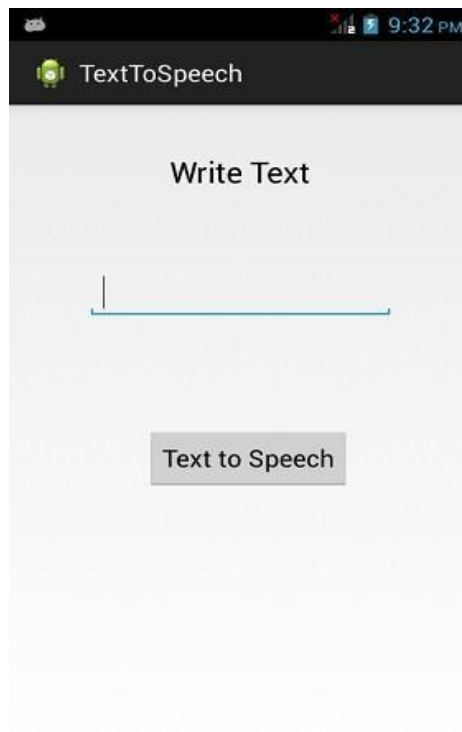
Here is the content of AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.texttospeech"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.texttospeech.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Let's try to run your TextToSpeech application. I assume you have connected your actual Android Mobile device with your computer. To run the app from Eclipse, open one of your project's activity files and click Run  icon from the toolbar. Before starting your application, Eclipse will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display following screen.



Now just type some text in the field and click on the text to speech button below. A notification would appear and text will be spoken. It is shown in the image below:



Now type something else and repeat the step again with different locale. You will again hear sound. This is shown below:



Assignment 8

Aim: To Study the Telephony (SmsManager) in android.

Objective: Student should know how to use the telephony package in android.

Outcome: Student will develop an application for sending sms using SmsManager in android.

SmsManager

There are following two ways to send SMS using Android device:

Using SmsManager to send SMS

Using Built-in Intent to send SMS

Using SmsManager to send SMS

The SmsManager manages SMS operations such as sending data to the given mobile device. You can create this object by calling the static method `SmsManager.getDefault()` as follows:

```
SmsManager smsManager = SmsManager.getDefault();
```

Once you have SmsManager object, you can use *sendDataMessage()* method to send SMS at the specified mobile number as below:

```
smsManager.sendTextMessage("phoneNo", null, "SMS text", null, null);
```

Apart from the above method, there are few other important functions available in SmsManager class. These methods are listed below:

S.N.	Method & Description
1	<code>ArrayList<String> divideMessage(String text)</code> This method divides a message text into several fragments, none bigger than the maximum SMS message size.

2	static SmsManager getDefault() This method is used to get the default instance of the SmsManager
3	void sendDataMessage(String destinationAddress, String scAddress, short destinationPort, byte[] data, PendingIntent sentIntent, PendingIntent deliveryIntent) This method is used to send a data based SMS to a specific application port.
4	void sendMultipartTextMessage(String destinationAddress, String scAddress, ArrayList<String> parts, ArrayList<PendingIntent> sentIntents, ArrayList<PendingIntent> deliveryIntents) Send a multi-part text based SMS.
5	void sendTextMessage(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent) Send a text based SMS.

Example

Following example shows you in practical how to use SmsManager object to send an SMS to the given mobile number.

To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.

Step	Description
1	You will use Eclipse IDE to create an Android application and name it as <i>SendSMSDemo</i> under a

	package <i>com.example.sendsmsdemo</i> . While creating this project, make sure you <i>Target SDK</i> and <i>Compile With</i> at the latest version of Android SDK to use higher levels of APIs.
2	Modify <i>src/MainActivity.java</i> file and add required code to take care of sending email.
3	Modify layout XML file <i>res/layout/activity_main.xml</i> add any GUI component if required. I'm adding a simple GUI to take mobile number and SMS text to be sent and a simple button to send SMS.
4	Modify <i>res/values/strings.xml</i> to define required constant values
5	Modify <i>AndroidManifest.xml</i> as shown below
6	Run the application to launch Android emulator and verify the result of the changes done in the application.

Exercise:

Key Points to study: telephony, SmsManager in android.

Develop an application for the sending sms from application.

Assignment 9

Aim: To study the database in android (SQLite)

Objective: Student should know the database development in android

Outcome: Student will demonstrate the database development using SQLite for basic queries.

SQLite and Android

What is SQLite?

SQLite is an Open Source database. SQLite supports standard relational database features like SQL syntax, transactions and prepared statements. The database requires limited memory at runtime (approx. 250 KByte) which makes it a good candidate from being embedded into other runtimes.

SQLite supports the data types TEXT (similar to String in Java), INTEGER (similar to long in Java) and REAL (similar to double in Java).

All other types must be converted into one of these fields before getting saved in the database. SQLite itself does not validate if the types written to the columns are actually of the defined type, e.g. you can write an integer into a string column and vice versa.

SQLite in Android

SQLite is embedded into every Android device. Using an SQLite database in Android does not require a setup procedure or administration of the database.

You only have to define the SQL statements for creating and updating the database. Afterwards the database is automatically managed for you by the Android platform.

Access to a SQLite database involves accessing the file system. This can be slow. Therefore it is recommended to perform database operations asynchronously.

If your application creates a database, this database is by default saved in the directory DATA/data/APP_NAME/databases/FILENAME.

The parts of the above directory are constructed based on the following rules. DATA is the path which the `Environment.getDataDirectory()` method returns. APP_NAME is your application name. FILENAME is the name you specify in your application code for the database.

SQLite architecture

Packages

The `android.database` package contains all necessary classes for working with databases. The `android.database.sqlite` package contains the SQLite specific classes.

Creating and updating database with SQLiteOpenHelper

To create and upgrade a database in your Android application you create a subclass of the `SQLiteOpenHelper` class. In the constructor of your subclass you call the `super()` method of `SQLiteOpenHelper`, specifying the database name and the current database version.

In this class you need to override the following methods to create and update your database.

onCreate() - is called by the framework, if the database is accessed but not yet created.

onUpgrade() - called, if the database version is increased in your application code. This method allows you to update an existing database schema or to drop the existing database and recreate it via the `onCreate()` method.

Both methods receive an `SQLiteDatabase` object as parameter which is the Java representation of the database.

The `SQLiteOpenHelper` class provides the `getReadableDatabase()` and `getWritableDatabase()` methods to get access to an `SQLiteDatabase` object; either in read or write mode.

The database tables should use the identifier `_id` for the primary key of the table. Several Android functions rely on this standard.

Note:

It is good practice to create a separate class per table. This class defines static `onCreate()` and `onUpgrade()` methods. These methods are called in the corresponding methods of `SQLiteOpenHelper`. This way your implementation of `SQLiteOpenHelper` stays readable, even if you have several tables.

SQLiteDatabase

`SQLiteDatabase` is the base class for working with a SQLite database in Android and provides methods to open, query, update and close the database.

More specifically `SQLiteDatabase` provides the `insert()`, `update()` and `delete()` methods.

In addition it provides the `execSQL()` method, which allows to execute an SQL statement directly.

The object `ContentValues` allows to define key/values. The key represents the table column identifier and the value represents the content for the table record in this column. `ContentValues` can be used for inserts and updates of database entries.

Queries can be created via the `rawQuery()` and `query()` methods or via the `SQLiteQueryBuilder` class .

`rawQuery()` directly accepts an SQL select statement as input.

`query()` provides a structured interface for specifying the SQL query.

`SQLiteQueryBuilder` is a convenience class that helps to build SQL queries.

`rawQuery()` Example

The following gives an example of a `rawQuery()` call.

`Cursor cursor = getReadableDatabase().`

```
rawQuery("select * from todo where _id = ?", new String[] { id });
```

`query()` Example

The following gives an example of a query() call.

```
return database.query(DATABASE_TABLE, new String[] { KEY_ROWID,  
KEY_CATEGORY, KEY_SUMMARY, KEY_DESCRIPTION }, null, null, null,  
null, null);
```

The method query() has the following parameters.

Table 1. Parameters of the query() method

Parameter	Comment
String dbName	The table name to compile the query against.
String[] columnNames	A list of which table columns to return. Passing "null" will return all columns.
String whereClause	Where-clause, i.e. filter for the selection of data, null will select all data.
String[] selectionArgs	You may include ?s in the "whereClause". These placeholders will get replaced by the values from the selectionArgs array.
String[] groupBy	A filter declaring how to group rows, null will cause the rows to not be grouped.
String[] having	Filter for the groups, null means no filter.
String[] orderBy	Table columns which will be used to order the data, null means no ordering.

If a condition is not required you can pass null, e.g. for the group by clause.

The "whereClause" is specified without the word "where", for example a "where" statement might look like: "_id=19 and summary=?".

If you specify placeholder values in the where clause via ?, you pass them as the selectionArgs parameter to the query.

Cursor

A query returns a Cursor object. A Cursor represents the result of a query and basically points to one row of the query result. This way Android can buffer the query results efficiently; as it does not have to load all data into memory.

To get the number of elements of the resulting query use the getCount() method.

To move between individual data rows, you can use the moveToFirst() and moveToNext() methods. The isAfterLast() method allows to check if the end of the query result has been reached.

Cursor provides typed get*() methods, e.g. getLong(columnIndex), getString(columnIndex) to access the column data for the current position of the result. The "columnIndex" is the number of the column you are accessing.

Cursor also provides the getColumnIndexOrThrow(String) method which allows to get the column index for a column name of the table.

A Cursor needs to be closed with the close() method call.

ListView, ListActivities and SimpleCursorAdapter

ListView are Views which allow to display a list of elements. ListActivities are specialized activities which make the usage of ListView easier. To work with databases and ListView you can use the SimpleCursorAdapter. The SimpleCursorAdapter allows to set a layout for each row of the ListView.

You also define an array which contains the column names and another array which contains the IDs of Views which should be filled with the data.

The SimpleCursorAdapter class will map the columns to the Views based on the Cursor passed to it. To obtain the Cursor you should use the Loader class.

Exercise:

Develop an application for the demonstration of basic queries (create table, insert, select, update, delete) in SQLite.

Assignment 10

Aim: Develop an application using all components of android and database.

Objective: Student should able to develop and deploy the android application.

Outcome: Student will develop the android application using database (SQLite/ External, MySQL) and web services.

Assignment: Student should come with their ideas and discuss with faculty and develop application on android. Application with web service and database has more weightage and with all necessary database operations.

Student should use maximum UI components to make UI better and user-friendly.