

Developing Android Applications Learner's Guide

Are you registered with
Onlinevarsity.com?

Yes



No



Did you download this book
from **Onlinevarsity.com**?

Yes



No



Scores

For each **YES** you score **50**

For each **NO** you score **0**

If you score less than 100 this book is illegal.

Register on **www.onlinevarsity.com**

Developing Android Applications Learner's Guide

© 2013 Aptech Limited

All rights reserved.

No part of this book may be reproduced or copied in any form or by any means – graphic, electronic or mechanical, including photocopying, recording, taping, or storing in information retrieval system or sent or transferred without the prior written permission of copyright owner Aptech Limited.

All trademarks acknowledged.

APTECH LIMITED

Contact E-mail: ov-support@onlinevarsity.com

First Edition - 2013



Dear Learner,

We congratulate you on your decision to pursue an Aptech course.

Aptech Ltd. designs its courses using a sound instructional design model – from conceptualization to execution, incorporating the following key aspects:

- Scanning the user system and needs assessment

Needs assessment is carried out to find the educational and training needs of the learner

Technology trends are regularly scanned and tracked by core teams at Aptech Ltd. TAG* analyzes these on a monthly basis to understand the emerging technology training needs for the Industry.

An annual Industry Recruitment Profile Survey# is conducted during August - October to understand the technologies that Industries would be adapting in the next 2 to 3 years. An analysis of these trends & recruitment needs is then carried out to understand the skill requirements for different roles & career opportunities.

The skill requirements are then mapped with the learner profile (user system) to derive the Learning objectives for the different roles.

- Needs analysis and design of curriculum

The Learning objectives are then analyzed and translated into learning tasks. Each learning task or activity is analyzed in terms of knowledge, skills and attitudes that are required to perform that task. Teachers and domain experts do this jointly. These are then grouped in clusters to form the subjects to be covered by the curriculum.

In addition, the society, the teachers, and the industry expect certain knowledge and skills that are related to abilities such as *learning-to-learn, thinking, adaptability, problem solving, positive attitude etc.* These competencies would cover both cognitive and affective domains.

A precedence diagram for the subjects is drawn where the prerequisites for each subject are graphically illustrated. The number of levels in this diagram is determined by the duration of the course in terms of number of semesters etc. Using the precedence diagram and the time duration for each subject, the curriculum is organized.

- Design & development of instructional materials

The content outlines are developed by including additional topics that are required for the completion of the domain and for the logical development of the competencies identified. Evaluation strategy and scheme is developed for the subject. The topics are arranged/organized in a meaningful sequence.

The detailed instructional material – Training aids, Learner material, reference material, project guidelines, etc.- are then developed. Rigorous quality checks are conducted at every stage.

➤ Strategies for delivery of instruction

Careful consideration is given for the integral development of abilities like thinking, problem solving, learning-to-learn etc. by selecting appropriate instructional strategies (training methodology), instructional activities and instructional materials.

The area of IT is fast changing and nebulous. Hence considerable flexibility is provided in the instructional process by specially including creative activities with group interaction between the students and the trainer. The positive aspects of web based learning –acquiring information, organizing information and acting on the basis of insufficient information are some of the aspects, which are incorporated, in the instructional process.

➤ Assessment of learning

The learning is assessed through different modes – tests, assignments & projects. The assessment system is designed to evaluate the level of knowledge & skills as defined by the learning objectives.

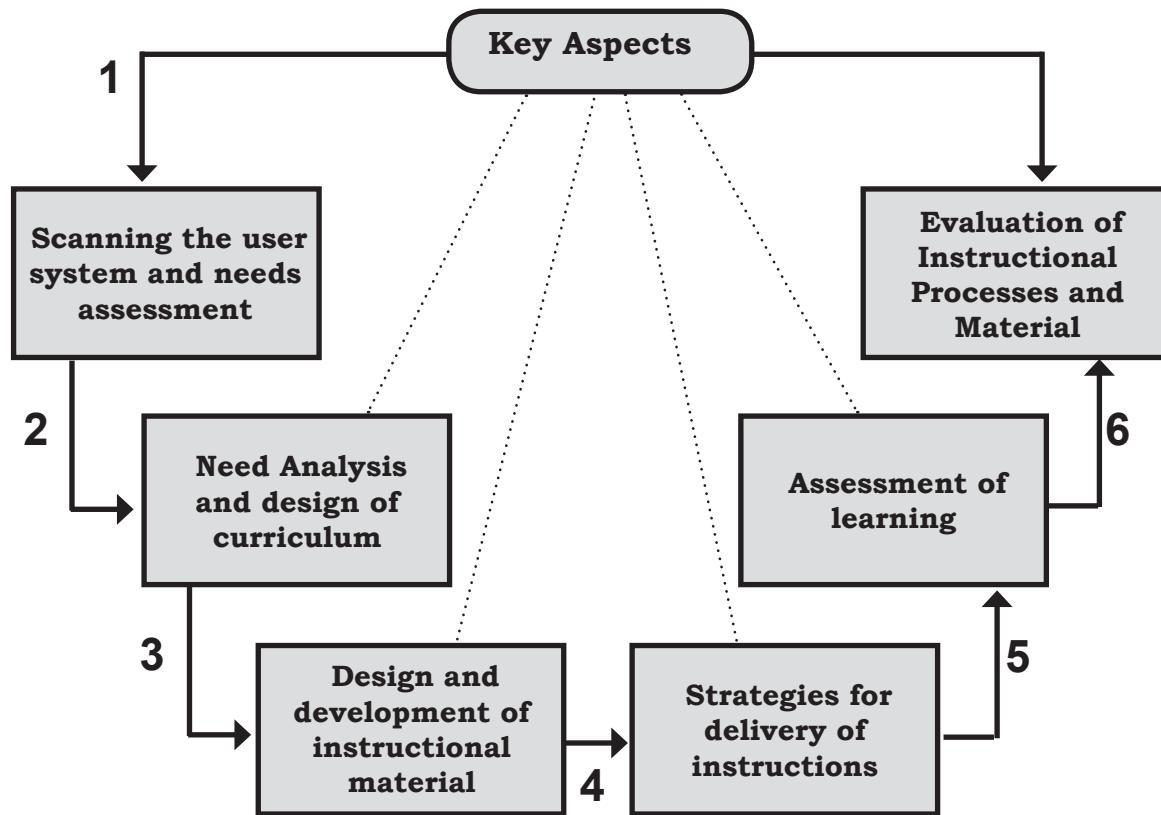
➤ Evaluation of instructional process and instructional materials

The instructional process is backed by an elaborate monitoring system to evaluate - on-time delivery, understanding of a subject module, ability of the instructor to impart learning. As an integral part of this process, we request you to kindly send us your feedback in the reply pre-paid form appended at the end of each module.

*TAG – Technology & Academics Group comprises of members from Aptech Ltd., professors from reputed Academic Institutions, Senior Managers from Industry, Technical gurus from Software Majors & representatives from regulatory organizations/forums.

Technology heads of Aptech Ltd. meet on a monthly basis to share and evaluate the technology trends. The group interfaces with the representatives of the TAG thrice a year to review and validate the technology and academic directions and endeavors of Aptech Ltd.

Aptech New Products Design Model



“

Nothing is a waste of time,
if you use the experience wisely

”

Preface

The book, Developing Android Applications, aims to teach the basics of developing Android applications for Android enabled devices. The book introduces the developer to the new features of Android Jellybean with a step-by-step approach and clearly explained sample codes using Google Android's Software Development Kit (SDK).

Android programming has undergone several reformations since its inception. This book intends to familiarize the developer with the new features and APIs of Android 4.1. The book begins with an introduction to the basic concepts of developing Android applications using Eclipse IDE. The book proceeds with the explanation of the various new user interface components such as Action Bar and GridLayout. It walks the developer through the various basic views that can be used to build the Android UI. It also teaches the different methods used for saving and storing data in the application. All these will enable the developer to create professional Android applications. The book proceeds and explains the connectivity APIs such as Wi-Fi and develop location-based service application using Google Maps. Finally, it concludes with an explanation of publishing an Android application.

This book is the result of a concentrated effort of the Design Team, which is continuously striving to bring you the best and the latest in Information Technology. The process of design has been a part of the ISO 9001 certification for Aptech-IT Division, Education Support Services. As part of Aptech's quality drive, this team does intensive research and curriculum enrichment to keep it in line with industry trends.

We will be glad to receive your suggestions.

Design Team

“

To learn,
you must want to be taught

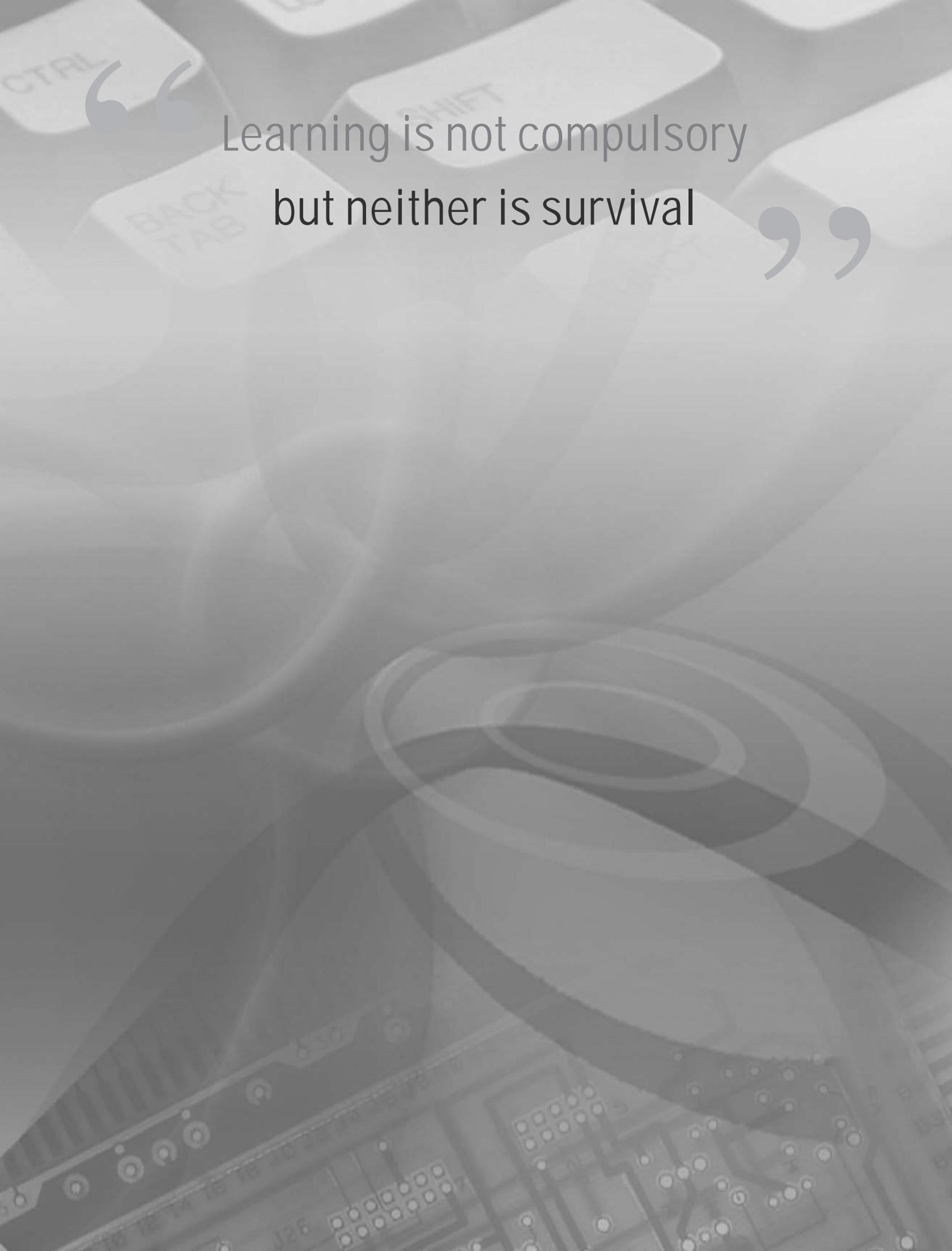
”

Table of Contents

Module

1.	Introduction to Android	1
2.	Getting Started with Android	47
3.	Android User Interface (UI)	111
4.	Android System Overview	187
5.	More UI Elements	275
6.	Media Handling	325
7.	Data Handling in Android	365
8.	Using Google API	403
9.	Services, Broadcast Receivers, and Intent Filters	441
10.	Bluetooth, Network, Wi-Fi, and Sensors	501
11.	SMS and Telephony	545
12.	Development for Android Market	567

“ Learning is not compulsory
but neither is survival ”



JELLYBEAN
ICE CREAM SANDWICH

Session - 1

Introduction to Android

Welcome to the Session, **Introduction to Android**.

This session explains what Android is, its history and advantages. It also describes the Android architecture, the layers, the application framework, libraries, and the Linux kernel. Further, it explains how to install the Android Software Development Kit (SDK).

In this session, you will learn to:

- ➔ Explain Android
- ➔ Explain the history of Android
- ➔ Describe the architectural framework of Android
- ➔ Explain the downloading and installation process of Android SDK



1.1 Introduction

An Operating System (OS) is a set of programs that is executed on any computer to manage the computer hardware resources and also provide services to the application program. Windows 7 and Linux Kernel are some of the common operating systems being used.

Following are some of the features of an operating system:

- Keeps track of files and folders, the peripheral devices such as disk drives and printers.
- Assists in security settings in a computer.
- Assists in converting keyboard instructions into an output.

Having understood the concept of an OS, let us assume that a mobile user wants to view his/her mobile as his/her desktop. What do you think are the advantages if one was able to substitute a PC with mobile? The profitability list is endless.

The number of Smartphones and the corresponding OS available in the market today are constantly on the rise. Some of the common OS for Smartphones include iPhone OS (iOS), Blackberry OS, Symbian, Windows Phone 7, and Android. These mobile phones execute different applications because of the availability of different mobile OS.

1.2 Introduction to Android

This section will explain in detail the history, evolution, and features of Android.

1.2.1 What is Android?

Android is an Operating System (OS) for mobile devices. Mobile devices include Smartphone and tablets as shown in figure 1.1.



Figure 1.1: Smartphone and Tablet

Android can be defined as a software stack for mobile devices. Apart from the OS, it includes middleware and key applications. In brief, Android is:

- A free open source Linux based mobile OS
- A development platform used for creating mobile applications

- A device used for executing the OS and applications created for it

Android is being used by a large community of developers to create applications. The Android applications are written in Java programming language. However, Android does not support a Java Virtual Machine (JVM). Hence, a new virtual machine which is exclusively meant for Android is used to compile the Java classes. The virtual machine supported by Android is the Dalvik Virtual Machine (VM). A distinctive feature of the Dalvik VM is that it administers low level memory management. The Dalvik VM depends on Linux Kernel for security, memory, and process management. The files executed by the Dalvik VM are available in the **.dex** format.

Additionally, the software is also popular because it is owned by Google, and a user has access to all the well-known Google applications (apps) as shown in figure 1.2.



Figure 1.2: Linux and Google Applications

It has become the most used and powerful tool in mobile systems today. All over the world, many device manufacturers have brought to the market devices that are capable of running Android OS.

1.2.2 History of Android

Android Inc. began in 2003 when its founders started the company with the intention of developing software for mobile phones. In 2005, Google acquired Android Inc as shown in figure 1.3.



Figure 1.3: Android Inc and Google

The first version of the Android was released on November 5, 2007. On this eventful day, Google, in coalition with a number of other companies, including mobile phone manufacturers and wireless providers revealed Android, which was built on the Linux platform. This

group included key players, such as, Texas Instruments, High Tech Computer Corporation (HTC), Google, Intel, LG, Motorola, Samsung Electronics, Broadcom Corporation, Marvell Technology Group, Nvidia, QualComm, Sprint Nextel, and T-Mobile. This coalition, termed as the Open Handset Alliance (OHA) aimed at providing a free and open source platform for mobile devices. The OHA came out with the release of Android's first version, built on the Linux kernel version 2.6., with an open source Apache License.

→ Evolution of Android Versions

It is important to learn about Android's history to better understand its growing popularity and demand. From the time of its market release in 2008, Android has since released a number of versions. With each version came a new set of features and options.

Table 1.1 lists the different Android versions and related features.

Android Version and Release	Description
Android 1.0 – 2008	<ul style="list-style-type: none"> ➔ The first commercial version of Android was released in 2008. It was significant for offering a wide variety of applications as opposed to Apple's iPhone. <p>Prominent Features:</p> <ul style="list-style-type: none"> ➔ A user could surf the Web quickly and enjoy the experience of Internet browsing as it was more user friendly ➔ Google Maps came with this version of Android, making it a useful device for travel ➔ It also allowed the user to sync e-mail and calendar with their Gmail account
Android Cupcake Version 1.5 – May 2009	<ul style="list-style-type: none"> ➔ The subsequent versions of Android are based on the names of desserts. The Cupcake version was incorporated with significant number of new features. The most popular feature amongst the set of new features was its virtual keyboard. <p>Prominent features:</p> <ul style="list-style-type: none"> ➔ A number of shortcuts and widgets on the home screen ➔ Virtual onscreen keyboard made Android phones lighter and smarter ➔ Video recording made it possible for the user to upload videos to YouTube ➔ Stereo Bluetooth enabled wireless music ➔ Copy and Paste function enabled on the Web browser

Android Version and Release	Description
Android Donut Version 1.6 – October 2009	<ul style="list-style-type: none"> ➔ The Donut had only a few major upgrades. It entered a new user market by supporting Code Division Multiple Access (CDMA) phones. <p>Prominent Features:</p> <ul style="list-style-type: none"> ➔ Universal search function ➔ More screen resolutions ➔ Improved Google Maps navigation features
Android Éclair Versions 2.0 and 2.1 – November 2009 and January 2010	<ul style="list-style-type: none"> ➔ The main upgrade that Éclair offered was support for the Microsoft Exchange Server and HTML 5. Thus, it opened the Android out to more business users. <p>Prominent features:</p> <ul style="list-style-type: none"> ➔ Access to Outlook mail ➔ Access to multiple Google accounts ➔ Support for flash, digital zoom, and other features in Camera settings ➔ Search for Text and Multimedia Messaging Services (MMS) messages ➔ Auto complete with a dictionary that included names in the phone directory ➔ Additional Web browser features
Android Froyo Version 2.2 – May 2010	<ul style="list-style-type: none"> ➔ Android Froyo was a popular version because it supported Flash. This dissolved one of the significant differences between the Android and the iPhone. <p>Prominent features:</p> <ul style="list-style-type: none"> ➔ Flash Player enhanced the Web, videos, photo viewing, and streaming audio experience ➔ Improved back-up storage for contacts and e-mail data ➔ Improved Outlook features ➔ Shared 3G Internet connection with other gadgets using Wi-Fi ➔ Better Web surfing and Bluetooth support

Android Version and Release	Description
Android Gingerbread Version 2.3 – December 2010	<ul style="list-style-type: none"> ➔ Android Gingerbread did not have major upgrades from its earlier version. The major difference was that single-core phones could now run applications that were previously designed for dual-core phones.
Android Honeycomb Version 3.0 – February 2011	<ul style="list-style-type: none"> ➔ With Honeycomb, Android could be used for bigger screens, including tablet PCs. This version was specifically meant for tablets and cannot be used for mobile phones. ➔ The main attraction in this release was the User Interface (UI). It was more interactive, and offered easier navigation. It also offered a more personalized experience. The highlight was that these enhanced features were provided for all applications, including the ones developed for earlier Android versions. For instance, the applications in Android version 3.0 had better capabilities in terms of graphics, UI, and multimedia features. <p>Prominent Features:</p> <ul style="list-style-type: none"> ➔ Improvements for a large range of tablets ➔ Zoom feature for apps with a fixed size ➔ Ability to download media files from a Secure Digital (SD) card – Users with devices that provide support for an SD card can download media files from the card. Apps in the device can then play the files
Android Ice Cream Sandwich (ICS) Version 4.0 – October 2011	<ul style="list-style-type: none"> ➔ Android ICS was built to merge Gingerbread and Honeycomb versions. However, this version does not support Flash. <p>Prominent Features:</p> <ul style="list-style-type: none"> ➔ A faster, smoother browser ➔ Data traffic monitor ➔ User-friendly action bar instead of the Menu button ➔ Better storage space for applications ➔ Pre-written text messaging feature to decline calls ➔ Face recognition feature for unlocking phone ➔ Live video effects

Android Version and Release	Description
Android Version 4.1 and 4.2 Jellybean – June 2012 and October 2012	<p>→ Android Jellybean 4.1 and 4.2 aimed at a better and faster UI. The features are said to be based on ‘Project Butter’, which enable Jelly Bean to function like ‘Butter’. This means that the touch related features are synchronized better and works faster.</p> <p>Prominent Features in 4.1:</p> <ul style="list-style-type: none"> → Touch Prediction, which attempts to predict where the user will touch the screen → Google Now, which tries to predict and provide the user with information based on the user’s existing patterns → Voice Typing without Online Access – When the user has a bad Internet connection, or is not connected, the user can still use the voice-to-text feature. Previously voice-to-text required the user to be connected to the Internet → Voice Search, which enables user to search for information through voice rather than text. This makes the search process faster and easier → Improvements in the Notifications feature <p>Prominent Features in 4.2:</p> <ul style="list-style-type: none"> → Developer Options added to the Settings menu, which was previously stand-alone → Lock Screen can now display widgets, which could earlier be viewed only on the home screen → Disable Lock Screen option is made available → Better image editing options → New screensavers named Daydreams → Photo Sphere, which allows user to take a 360-degree picture

Table 1.1: Android Versions and Features

1.2.3 Advantages of Android

Android has several advantages as an OS. The main advantage is that it offers a uniform approach for developing Android applications. It provides support for the following:

- ➔ **Java** – Java is the language used for Android applications. A large number of developers are familiar with Java, which makes it easier for them to develop applications in Android. Java also provides numerous developer tools.
- ➔ **2D and 3D Graphics and Animation** – One of the main advantages of Android is that it comes with several APIs for developing 2D and 3D graphics. It also has several features for adding animation to the UI.
- ➔ **Multiple Languages** – Android provides multiple language support for apps powering voice and text features. The user also can also customize the language for the UI.
- ➔ **Web Browsers** – Android provides support for most Web browsers, such as, Firefox, Chrome, Opera, SkyFire, XScope, and Dolphin.
- ➔ **Multimedia Formats** – Android offers support for several multimedia formats, such as, MPEG4, MIDI, 3GP, and so on.
- ➔ **Video Telephony** – Android phones provide a variety of video telephony features, such as, video conferencing, video chat, video recorder, and so on.
- ➔ **Additional Hardware** – Developers add numerous apps to Android. Many apps require proper hardware to function properly. Android provides support for additional hardware to help such apps to function properly. For instance, Android provides support for hardware, such as, camera and sensors.
- ➔ **Flash** – Many mobile platforms do not provide support for Flash. Most Android versions support Flash, which enhances the Web browsing experience, and helps user to play free games, as well as watch live events. Android Jellybean does not support Flash. However, it can be manually downloaded and installed.
- ➔ **Google Map Applications** – With the display, annotate, and manipulate functionality, the map based applications can be developed.
- ➔ **Services** (Background services and applications) – Provides services and applications that run in the background. For example, one could use a different ringtone and volume based on location.
- ➔ **Data Sharing** – Intent and Content Providers are two components of Android. These components enable the transfer of messages and sharing of data across applications and components.
- ➔ **P2P Communication Facility** – Peer-to-peer inter-device application messaging facility.

- **All Applications are Equal** – This implies that the native applications can be replaced by third party alternatives. Exceptions are, the ‘unlock’ and ‘in call experience’ screens.

Other advantages include:

- Open source framework
- Free availability of a variety of Google applications
- Ease of use
- Social networking friendly
- Popular User Interface (UI) design updates

1.2.4 Comparison of Android Phone and iPhone

The Android, because of its open source software has a number of advanced capabilities that its competitors do not possess. Its primary competitor is iPhone and Android Phone possesses certain advantages over iPhone as shown in figure 1.4.



Figure 1.4: Android Phone and iPhone

Some of the advantages that Android phone has over iPhone are as follows:

- One of the main advantages of the Android smartphone over the iPhone is automation. This is possible using the feature, Tasker. The user can use a set of conditions to turn certain settings on or off. For instance, a user can automate the phone ringer. If there is a regular meeting scheduled every day, the user can set the ringer to vibrate at the meeting start time, and exit the setting at the meeting end time. This feature is not currently possible with other platforms.
- An Android smartphone user can add widgets to the home screen. Widgets help the user to monitor several applications that are in use. Widgets are also useful to view reminders on the home screen. Widgets cannot be added to the home screen of the iPhone.

- ➔ Android smartphone and the iPhone allow users to customize their home screen. However, Android smartphone allows external users to access a user's home screen and add new features. This helps the user to upgrade the platform and improve the performance.
- ➔ The user has access to a wide variety of free applications through Android Market or other Web sites. The user does not have to browse for applications on the phone. Downloading and installing the apps on iPhone is more complicated.
- ➔ An interesting feature of Android phone is that the user is able to remove and replace the battery. The memory card can be replaced as well. With iPhone, a user has no options if the device exceeds the storage limit.
- ➔ Since Android is an open source platform it allows various developers to make improvements to it. Users can easily access, upgrade, and download them instead of waiting for the next release. With iPhone, however, users are required to wait for the next version for upgrades of the OS.
- ➔ Android phone users can access several functions of their phones from a PC. They can send text messages from a Web browser as well.
- ➔ With Android, apps can be integrated with the platform. For instance, when Google Voice is installed, all functions related to voice calls can be routed through Google Voice when it is integrated with the OS.
- ➔ Flash can be accessed through Android phone, while it is not accessible through iPhone.
- ➔ An Android user can install multiple keyboards for different kinds of text functions.

1.2.5 What isn't Android?

The Android is mistaken for several other things which it is not. They are as follows:

- ➔ A mobile phone – The Android is often mistaken for a mobile phone with advanced and user friendly features. It is however, a platform for mobile devices.
- ➔ Google's counterpart to the iPhone – Although iPhone is a close competitor to Android smartphone, the main difference is that the iPhone is solely owned by a single company (Apple), and all releases and versions are made by Apple. Android, on the other hand, can function on any device that is compatible. It is built using open source code, and is supported by the Open Handset Alliance (OHA).

- A single application layer – Android includes an application layer. It also consists of an OS, applications, API libraries, and the Linux Kernel.
- An implementation of a Java Mobile (ME) Edition – Although developers use Java to write applications for Android, the applications cannot be run in a Java ME.
- Parts of Standards – There are organizations that provide mobile phone standards, such as, the Linux Phone Standards Forum (LiPS), or the Open Mobile Alliance (OMA). Although Android functions on open source, Android has more to offer than these standards.

1.2.6 Android and the Open Source Platform

Android, as specified earlier, is based on Linux platform. The code is open source as shown in figure 1.5, that is, the code for the product's design and implementation is freely distributed.



Figure 1.5: Open Source

The code can be freely accessed, and any developer can make changes to the existing code. This feature enables a wide variety of Android enthusiasts, service providers, and mobile device providers to modify the code. It also makes a whole plethora of applications available to Android. The open source code makes Android a low cost, high quality, and customizable platform, thus turning it into a widely used and extremely popular option. Another advantage of modified code is that new features and updates are added at a rapid rate.

However, there are certain disadvantages to the open source platform as well. The updates are made quicker than the official releases, and Android phones can be easily 'hacked'. Although the OS itself has a robust security and privacy system, the vast number of developers and applications weaken the system.

→ Features of Android

Android's platform being based on open source, the code can be freely accessed by developers, manufacturers, service providers, and other enthusiasts and customized. The platform has no configuration limitations for hardware or software. By itself, however, Android supports certain features. They are as follows:

- **Data Storage:** SQLite is the Database used for Android. This is a popular Relational Database Management System (RDBMS). The SQLite is a lightweight RDBMS, and stands apart because it runs as a part of the client application. Other DBMS run as an independent process.
- **Networking:** Android supports a wide variety of technology used for network connectivity, such as, Global System for Mobile Communications (GSM), Integrated Digital Enhanced Network (iDEN), Bluetooth, Enhanced Data rates for GSM Evolution (EDGE), Code Division Multiple Access (CDMA), Long Term Evolution (LTE), and Wi-Fi.
- **Media Formats and Files:** Android provides support for a large number of media files formats, such as, Moving Picture Experts Group 4 (MPEG-4) Simple Profile (SP), MP3, MP4, Advanced Audio Coding (AAC), H.263, Musical Instrument Digital Interface (MIDI), Joint Photographic Experts Group (JPEG), Graphics Interchange Format (GIF), and so on.
- **Messaging Support:** Android offers support for both, Multimedia Messaging Service (MMS) and Short Message Service (SMS).
- **Browser Support:** Android provides support for several browsers, such as, Google Chrome, Opera, Firefox, SkyFire, and so on. The platform includes the WebKit, and Google Chrome's JavaScript Engine, which enable the Web browsers to function effectively.
- **Hardware:** Android includes support for a variety of hardware, such as, camera, proximity sensor, Global Positioning System (GPS), Accelerometer Sensor, and digital compass.
- **Touch Screen:** Android provides support for multi-touch screens.
- **Multi-tasking Applications:** Android offers support for multi-tasking applications, such as, Tasker.
- **Internet Connection Sharing:** Android enables sharing both wired and wireless Internet connections.
- **Flash Support:** The Android 2.3 version provides support for Flash 10.1. This provides users with a better animation and graphics experience.

1.3 Android Architecture

The Android architecture as shown in figure 1.6 is made up of different layers. Each layer consists of several program components. All the layers are connected to each other. The Android architecture consists of the following four layers:

- ➔ Linux Kernel
- ➔ Libraries

- Application Framework
- Applications

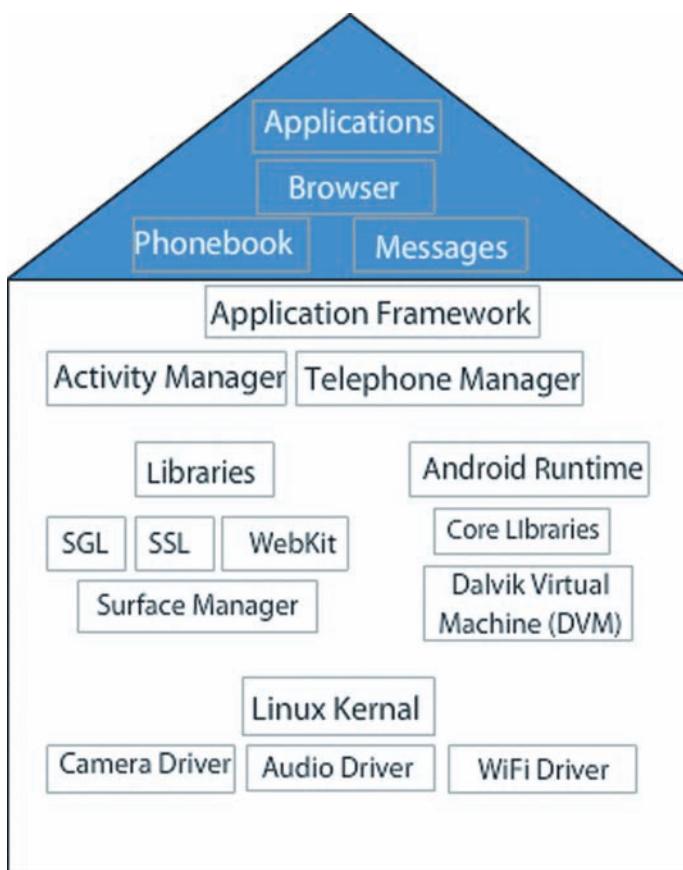


Figure 1.6: Android Architecture

Table 1.2 lists the four layers and provides a description of each layer.

Layer	Description
Linux Kernel	<ul style="list-style-type: none"> → The Linux kernel is the base layer. This is the OS layer upon which the entire framework is built. → The layer communicates with the hardware and consists of the key hardware drivers. For example, for the camera to function on an Android phone, the kernel must have a camera driver installed. → The kernel is also used for important functionalities, such as, managing memory, processes, network, and security. → The Linux kernel is a robust OS that enables the Android to adapt to a variety of hardware. → The Linux kernel interacts with the Libraries layer. It also serves as an interface between the hardware and all other elements of the stack.

Layer	Description
Libraries	<p>→ The layer immediately above the Linux kernel consists of the libraries. In generic terms, a Library is nothing but a resource or information stack.</p> <p>→ For a developer, a library is a resource center to assist in software development. The libraries contain data meant for specific hardware. The library consists of APIs and C/C++ core libraries. Different components make use of these libraries. Developers can access the libraries through the Application Framework layer.</p> <ul style="list-style-type: none"> • Surface Manager – The Surface Manager enables combining different functions on a single frame. It helps in display management, and also determines how the applications should be layered. • Open Graphics Libraries (GL) ES and Scalable Graphics Library (SGL) – Open Graphics libraries support 2D and 3D graphics implementation and related aspects. <ul style="list-style-type: none"> ◆ Media Framework – Provides support for audio and video formats, recording, playback, and other related functions. It also helps to manage static images. ◆ FreeType – Assists in bitmap and vector font rendering. Also supports fonts and font related functions. ◆ Secure Sockets Layer (SSL) – Helps to provide security related features. ◆ SQLite – SQLite is an open source storage database that manages all database related queries. ◆ WebKit – Web browser provides the tools for building a Web browser. ◆ Libc: A small size custom library for the C compiler. This library needs to be loaded in all the processes.

Layer	Description
Android Runtime	<p>→ The Android Runtime is also part of the Libraries layer. The execution of the applications is dependent on the Android Runtime which consists of the core libraries and the Dalvik VM.</p> <ul style="list-style-type: none"> • The Dalvik VM executes applications that require minimum processing power and memory. • It is similar to the Java Virtual Machine (JVM). The JVM runs .class files whereas the Dalvik VM runs .dex files, which are tailored to provide higher efficiency. The Dalvik VM provides support for security, memory management, isolation, and threading. DVM allows every Android application to run as a separate process, but with its own instance of DVM. DVM is a virtual machine that is created when a process starts and is destroyed when the process exits. • Core Java libraries contain all the Java files, which enable the utilities and tools in a mobile device. These libraries consist of .class files. It includes the basic set of libraries that are imported into an application for its implementation. <p>→ The Libraries layer interacts with the Application Framework layer.</p>

Layer	Description
Application Framework	<p>→ The Application Framework layer manages the applications for the basic functioning of the device. This layer enables important functions, such as, voice calls, data sharing, activities, and location management.</p> <ul style="list-style-type: none"> • Activity Manager – Manages the lifecycle of all Android applications, such as, Start, Run, Background, and Kill. It provides a common stack for applications running in various processes so that navigation between applications is synchronized. • Packages Manager – Monitors the applications installed on the mobile device. The user might download and install an application online. Also, the user may use a Compact Disc (CD) and package manager to track the application and its capabilities. • Windows Manager – Manages active windows. It is more of a lower layer service offered by the Surface Manager. • Telephony Manager – An important element that consists of the APIs to construct the phone application, including voice SMS, and dialing services. • Content Providers – The content providers are important to the Android platform, as they enable applications to share data with each other. For example, the messaging application may require information from the contacts application. • Resources Manager – Stores the external elements of an application that do not include code, such as, layout files, strings storage, and bitmaps. • View System – Contains the basic applications for UI, such as, lists and buttons. It also performs other tasks, such as, dispatching events, drawing, and layouts. • Location Manager – Manages GPS, Wi-Fi, and cell tower applications. • Notification Manager – Manages notifications, such as, reminders, task lists, alarms, and so on. Displays updates on the status bar. • Extensible Messaging and Presence Protocol (XMPP) Manager – Manages XMPP service related applications. Provides support for Internet Messaging, Instant message, chat, offline messaging, and Voice Over IP (VOIP). <p>→ This layer interacts with the Applications layer.</p>

Layer	Description
Applications	<ul style="list-style-type: none"> → The topmost layer consists of all the applications. A few applications, such as, the Web browser and Contacts manager come with the standard installation. However, a variety of other applications can be added to the phone. → The layer interacts with the other layers.

Table 1.2: Android Architecture

1.4 Android Software Development Kit

A Software Development Kit (SDK) as shown in figure 1.7 is a package that consists of all the tools required for developing applications.



Figure 1.7: Android SDK

The applications are typically developed for specific software, hardware, OS, or any other platform. The Android Software Development Kit (SDK) provides a comprehensive set of tools that help the developer to build applications for Android. The SDK also offers tools for testing and debugging apps. It includes libraries, sample code, an emulator, and tutorials among other tools.

The advantage of using the SDK is that it is easier to download. Once the SDK is downloaded, developers can then login to the Android App Developer Web site and use the emulators to understand the application. The developer can avoid uploading, downloading, or reloading parts of software.

1.4.1 System Requirements for SDK Installation

Table 1.3 lists the system requirements for the Android SDK.

Requirement	Description
Operating Systems Supported and Requirements	<ul style="list-style-type: none"> → Windows XP (32-bit), Vista (32- or 64-bit), or Windows 7 (32- or 64-bit) → Mac OS X 10.5.8 or later (x86 only) → Linux

Development Environment Supported and Requirements	<ul style="list-style-type: none"> ➔ Eclipse 3.6 (Helios) or greater <p>Note - Eclipse 3.5 (Galileo) is no longer supported with the latest version of Android Development Tools (ADT).</p> <ul style="list-style-type: none"> ➔ Eclipse Java Development Tools (JDT) plugin ➔ Several types of Eclipse packages are available for each platform. For developing Android applications, it is recommended that one of these packages be installed: <ul style="list-style-type: none"> • Eclipse Integrated Development Environment (IDE) for Java Developers • Eclipse Classic • Eclipse IDE for Java Enterprise Edition (EE) Developers ➔ JDK 6 - Java Runtime Environment (JRE) alone is not sufficient ➔ ADT plugin recommended
Recommended Hardware Requirements	<ul style="list-style-type: none"> ➔ Minimum 2 Gigabyte (GB) of Random Access Memory (RAM) ➔ Intel Core2Due or equal processor for x86 architecture ➔ 4 GB hard disk space

Table 1.3: System Requirements for Android SDK Installation

1.4.2 Downloading Android SDK

There are two options for downloading the Android SDK. A relatively inexperienced developer can download the ADT Bundle. This includes:

- ➔ Eclipse and ADT plugin
- ➔ Android SDK tools
- ➔ Android Platform tools
- ➔ Latest Android platform
- ➔ Latest Android system image for emulator
- ➔ If the developer already has Eclipse, or another IDE, the SDK Tools only option can be used.

Note - Internet links are subject to change. If the link provided does not function, navigate to the <http://www.android.com> Web page, and then navigate to the Developers page to find the updated link.

→ Installing the Android ADT Bundle

- Installation steps for Windows OS:

1. Check system requirements for Windows OS. System requirements are provided in **1.5.1. System Requirements for SDK Installation**.
2. If the Java Development Kit is not installed on your system, go to www.oracle.com/technetwork/java/javase/downloads/index.html. Scroll down to locate Java SE 6 and click DOWNLOAD below JDK as shown in figure 1.8.

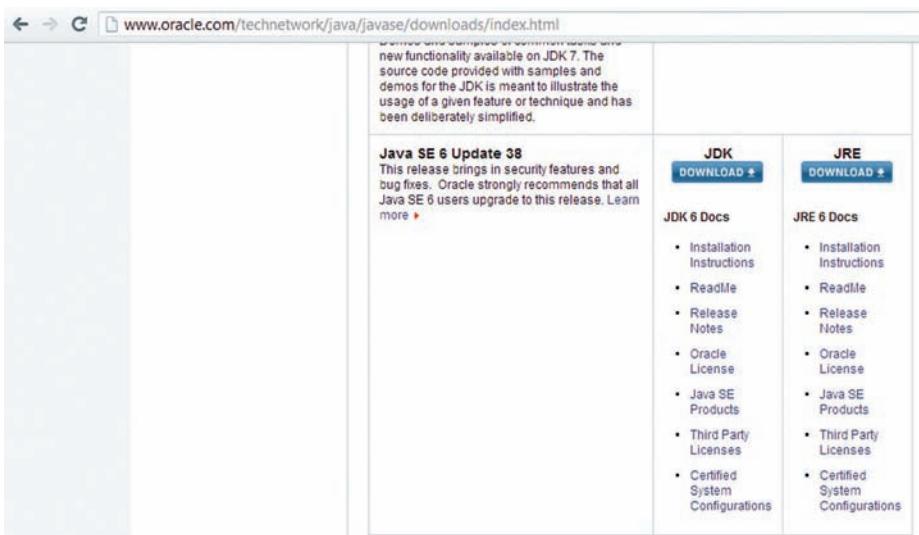


Figure 1.8: JDK Download Page

Note - Internet links are subject to change. If the link provided does not function, go to the <http://www.oracle.com> Web page, and navigate to the Downloads page to find the updated link.

3. To install Eclipse IDE, and the ADT Bundle, go to <http://developer.android.com/sdk/index.html>.

Note - If the Eclipse IDE or another IDE is already installed, go to steps for **Setting Up An Existing IDE**.

4. In the **Get the Android SDK** page, on the right side, click **Download the SDK ADT Bundle for Windows** as shown in figure 1.9. The **Terms and Conditions** page is displayed.



Figure 1.9: Download the SDK ADT Bundle for Windows

5. In the **Terms and Conditions** page as shown in figure 1.10, read the terms. Click "**I have read and agree with the above terms and conditions**" checkbox. Depending on the capacity of the processor, select **32-bit** or **64-bit**.

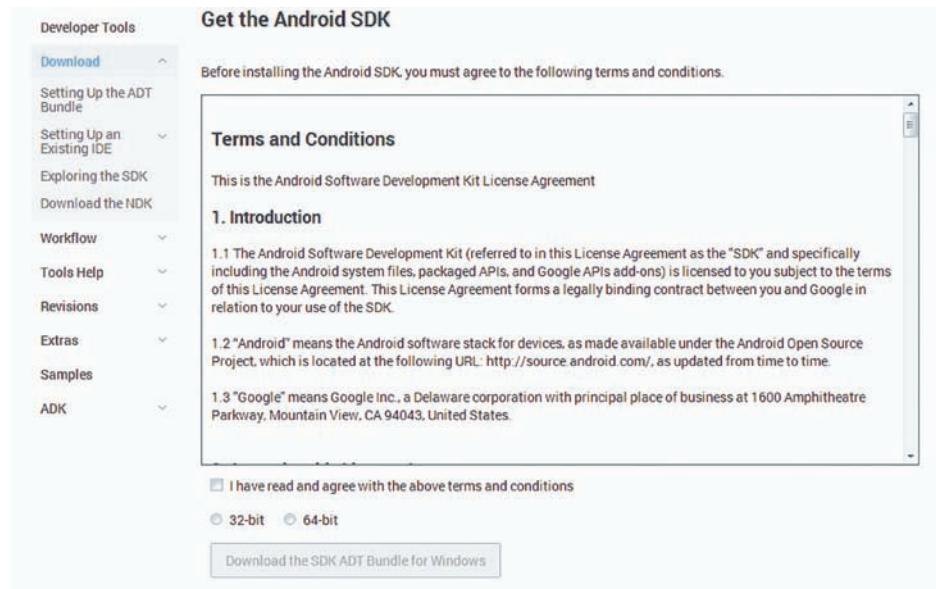


Figure 1.10: Terms and Conditions Page

6. Click **Download the SDK ADT Bundle for Windows** button as shown in figure 1.11.

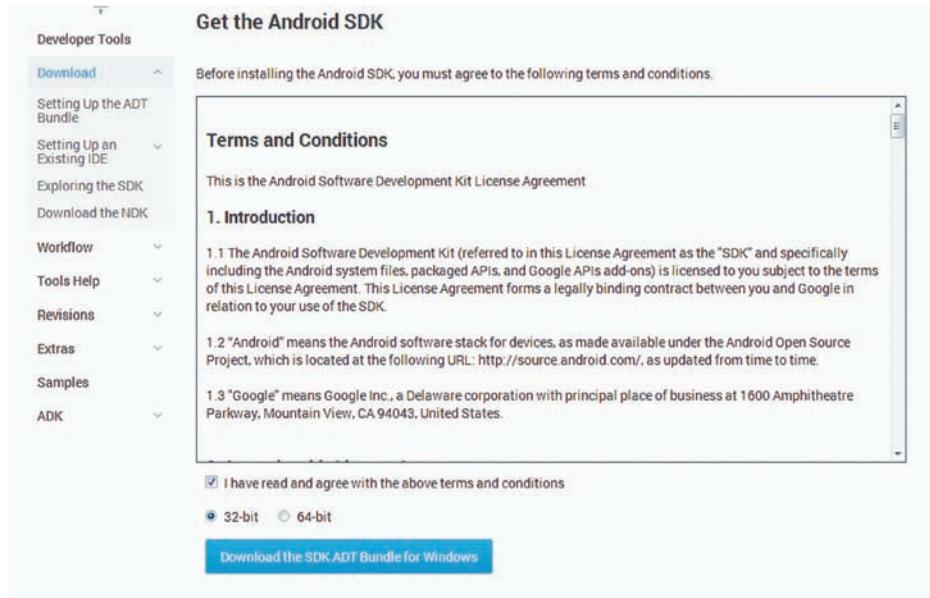


Figure 1.11: Download enabled after accepting Terms and Conditions

The **File Download** dialog box is displayed as shown in figure 1.12.

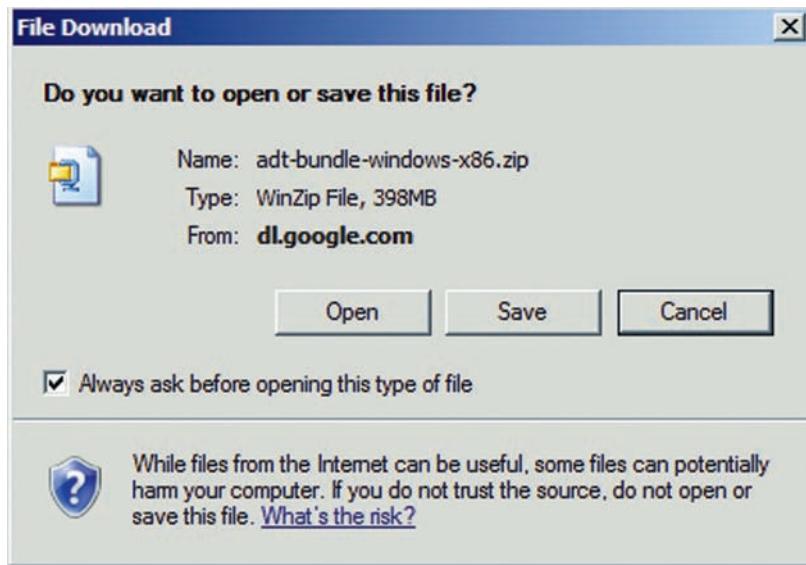


Figure 1.12: File Download Dialog Box

7. Click **Save**. The **Save As** dialog box appears.

8. In the **home** directory create a new directory, for example, **Android SDK**. Double-click the **Android SDK** directory and click **Save** as shown in figure 1.13. The **adt-bundle-windows-x86** file is saved to the directory.

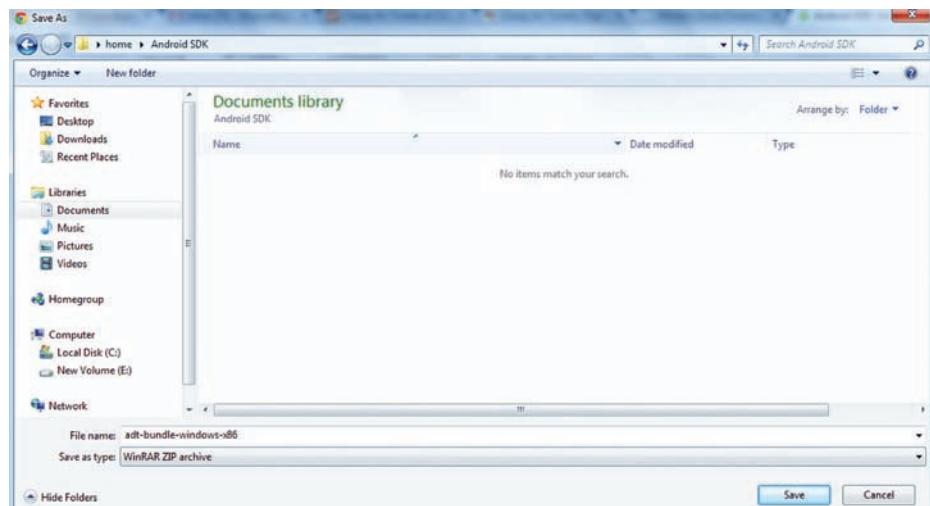


Figure 1.13: Save As Dialog box

9. Right-click the **adt-bundle-windows-x86** file and select **Extract to adt-bundle-windows-x86** as shown in figure 1.14.

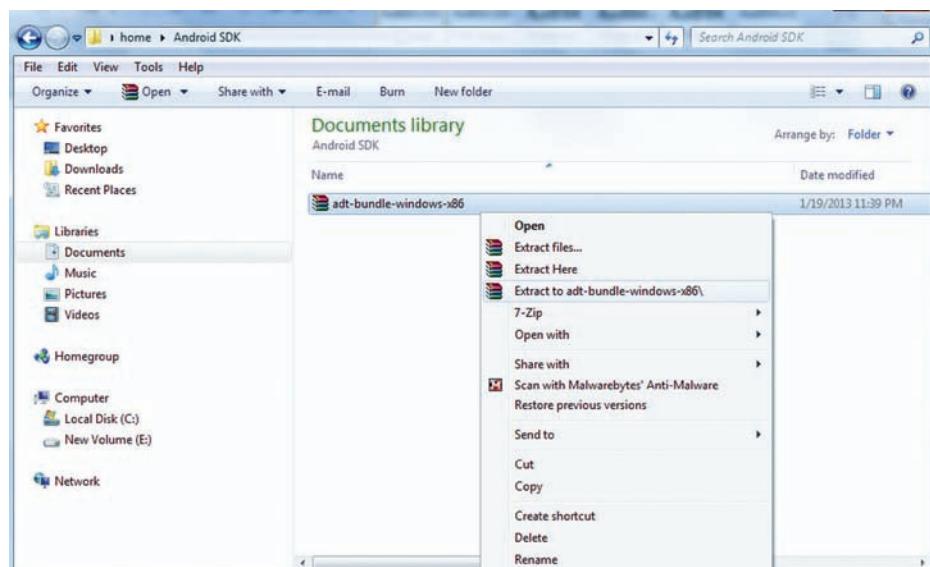


Figure 1.14: Extracting the ADT Bundle

The files are extracted to the **adt-bundle-windows-x86** folder in the **Android SDK** folder as shown in figure 1.15.

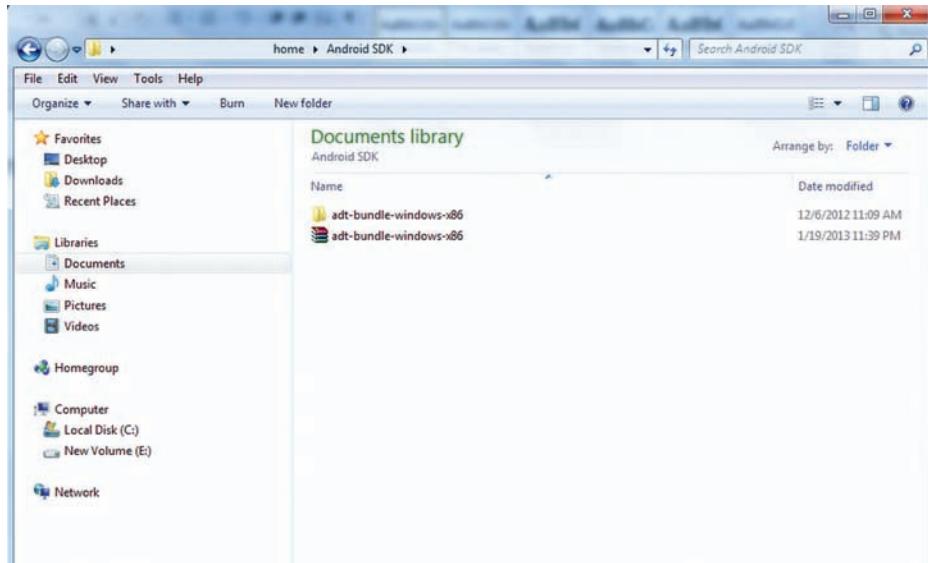


Figure 1.15: After Extraction

- Double-click the **adt-bundle-windows-x86** folder to display the eclipse folder as shown in figure 1.16.

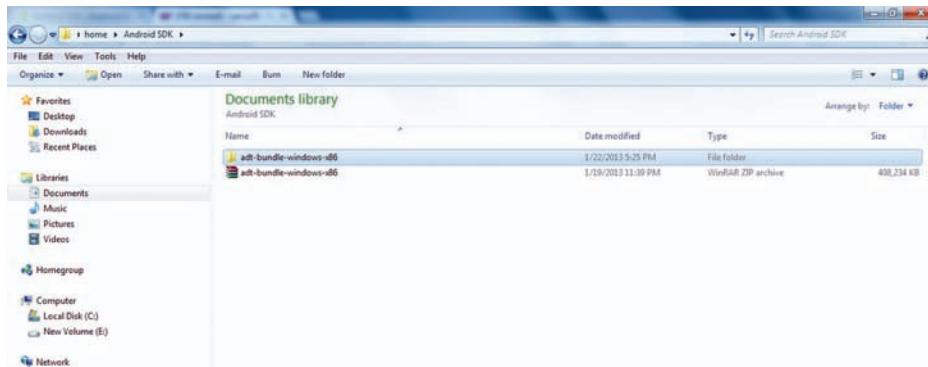


Figure 1.16: adt-bundle-windows-x86 Folder

11. Double-click the **eclipse** folder to open it as shown in figure 1.17.

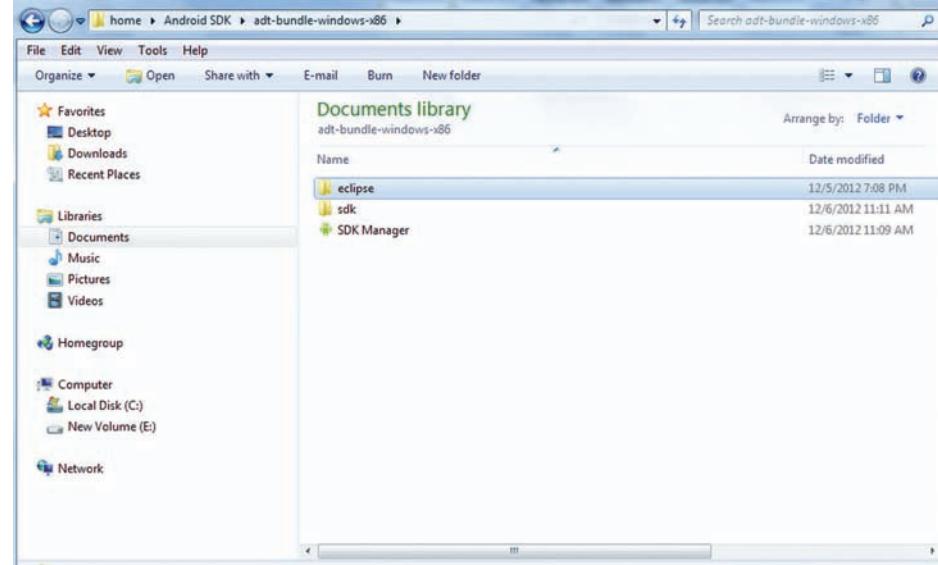


Figure 1.17 Eclipse Folder

The contents of the eclipse folder are displayed as shown in figure 1.18.

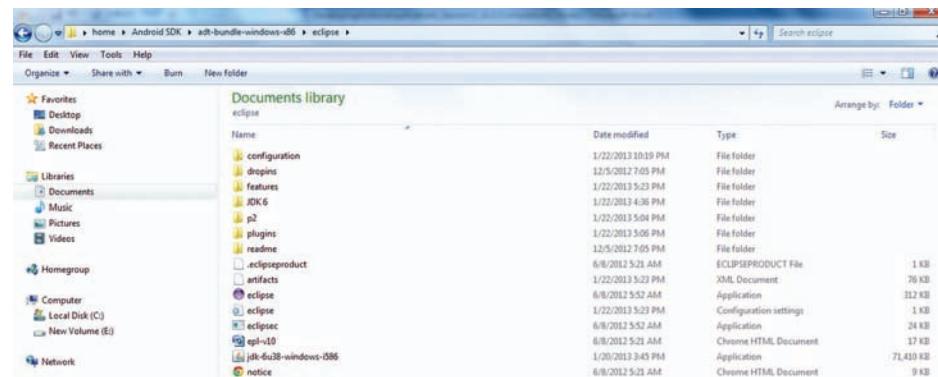


Figure 1.18: Detailed Eclipse IDE

12. Double-click the **eclipse** icon to start the IDE as shown in figure 1.19.

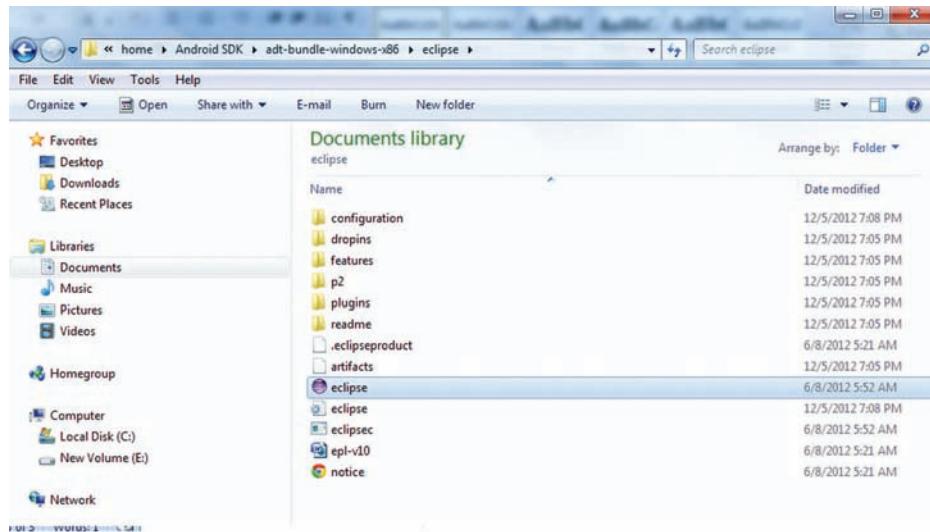


Figure 1.19 eclipse Icon in Eclipse Folder

The **Workspace Launcher** dialog box is displayed as shown in figure 1.20.

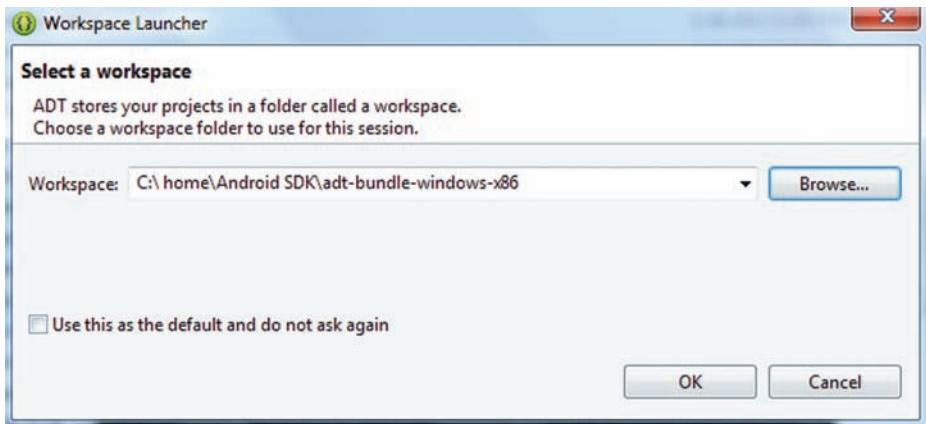


Figure 1.20: Workspace Launcher

- **Installation for Other Platforms:**

General installation steps for Platforms other than Windows are as follows:

1. Check system requirements.
2. If the Java Development Kit is not installed on your system, go to www.oracle.com/technetwork/java/javase/downloads/index.html. Scroll down to locate Java SE 6 and click **DOWNLOAD** below JDK.
3. Go to the site <http://developer.android.com/sdk/index.html>.
4. Go to the **Get the Android SDK** page. At the bottom click **DOWNLOAD FOR OTHER PLATFORMS** link. The **ADT Bundle** table appears.

5. In the **ADT Bundle** table, in the **Platform** list, identify the OS.
6. In the **ADT Bundle** table, in the **Package** list, click the required .zip file link to download the ADT Bundle for your platform.
7. Follow steps 5 – 10 provided for Installing SDK for Windows OS to complete the installation process.

→ Setting Up an Existing IDE:

If the developer has already installed the IDE, the SDK installation can be completed as per the OS platform.

- **SDK Installation for Windows:**

The download package consists of an executable file that initiates the installer. The installer checks if the machine or device has the Java SE Development Kit (JDK) installed and if not, installs the kit. The installer saves the Android SDK Tools to a specific location. The developer can also select the directory where the file needs to be saved.

1. Go to the site <http://developer.android.com/sdk/index.html>.
2. In the **Get the Android SDK** page, at the bottom, click **USE AN EXISTING IDE** link as shown in figure 1.21.

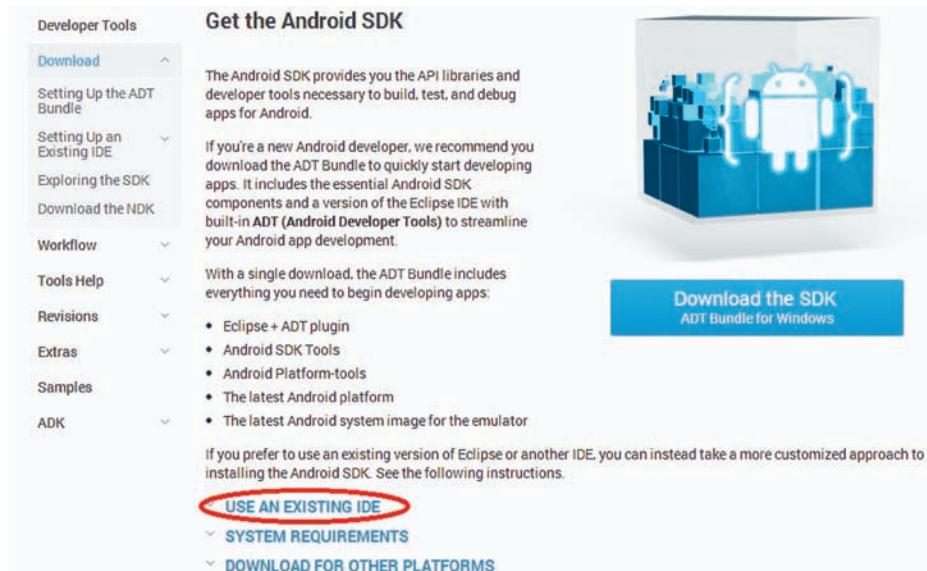


Figure 1.21: Use An Existing IDE

3. Click **Download the SDK Tools for Windows** as shown in figure 1.22. The **Terms and Conditions** page appears.

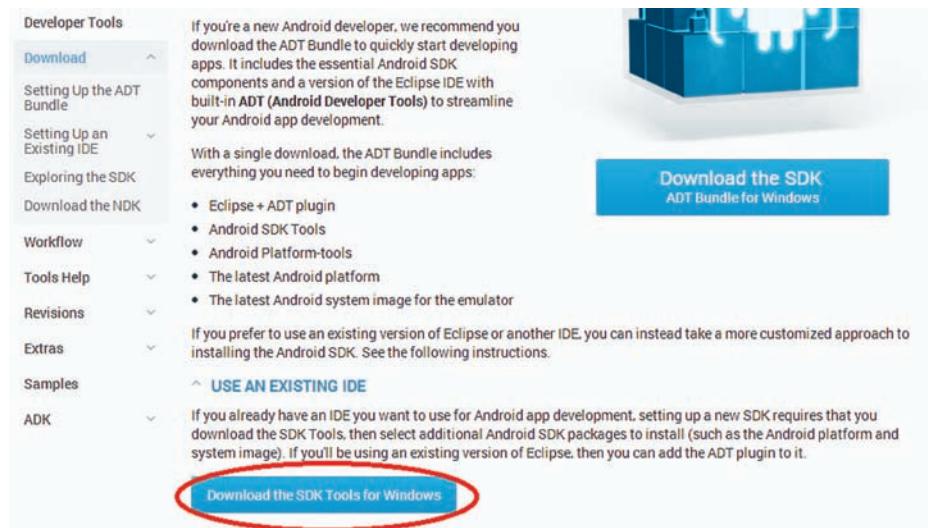


Figure 1.22: Download the SDK Tools for Windows Page

4. In the **Terms and Conditions** page, read the terms as shown in figure 1.23. Click “**I have read and agree with the above terms and conditions**” checkbox.

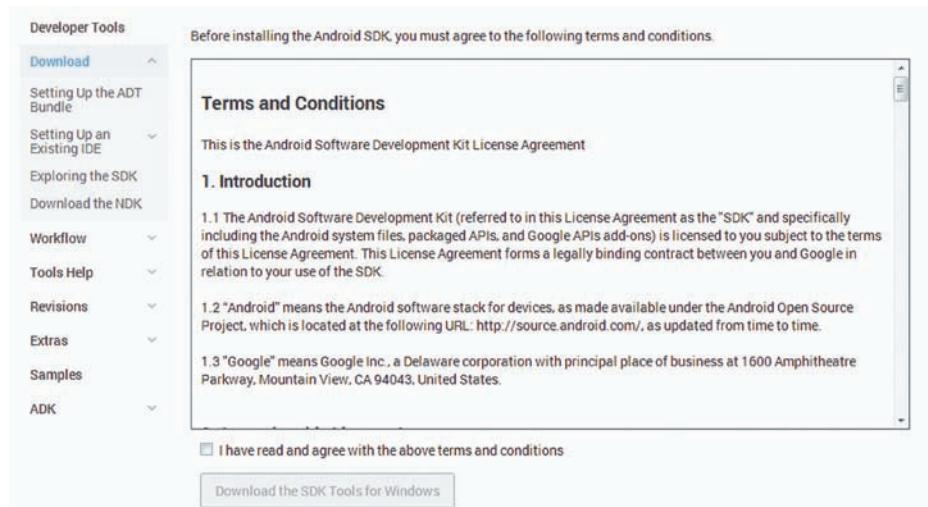


Figure 1.23: Terms and Conditions Page

5. Click **Download the SDK Tools for Windows** button as shown in figure 1.24. The **Save As** dialog box appears.

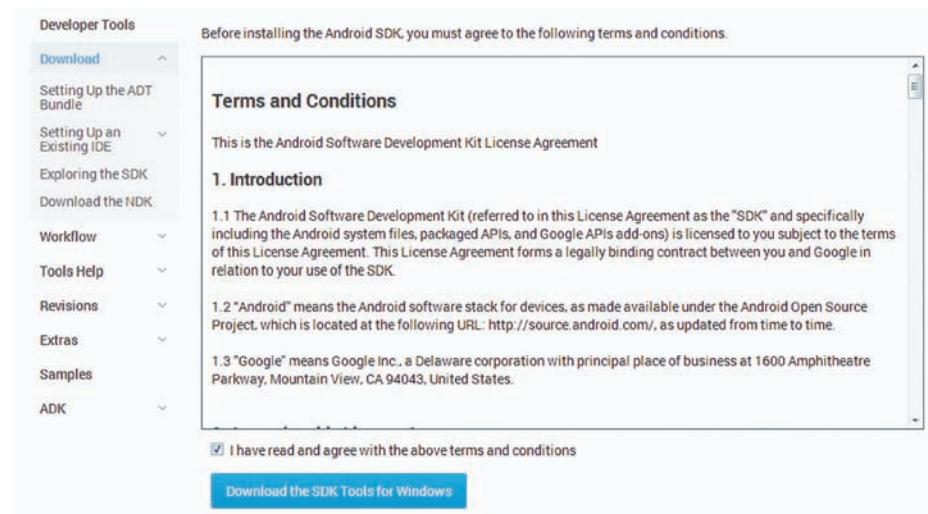


Figure 1.24: Terms and Conditions agreed to proceed with download

6. The **File Download** page is displayed as shown in Figure 1.25.



Figure 1.25: File Download Page

Note - This window is displayed for Windows OS other than Windows 7

7. Click **Save**.
8. In the **home** directory create a new directory, for example, **Android SDK Tools**.

9. Double-click the **Android SDK Tools** directory and click **Save** as shown in figure 1.26. The **installer_r21.0.1-windows** file is saved to the directory.

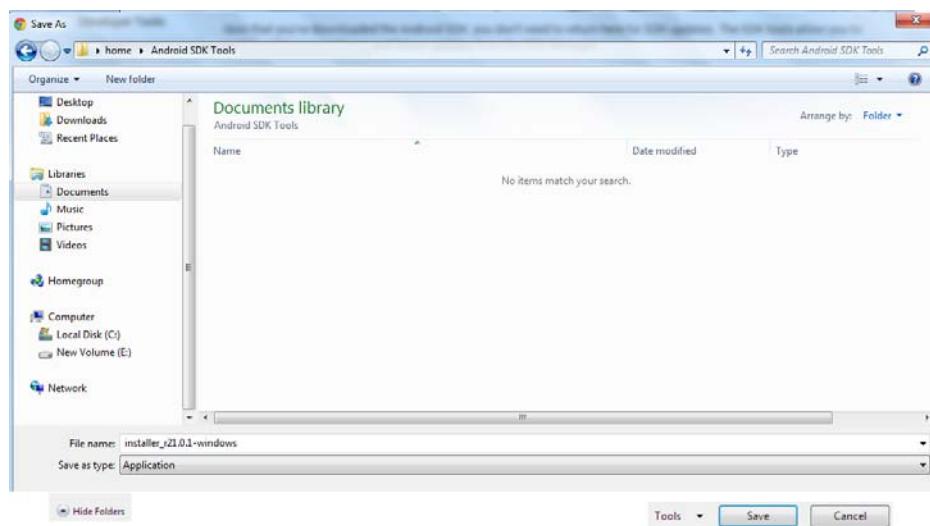


Figure 1.26: Android SDK Tools

10. Double-click the **installer_r21.0.1-windows** file to view the **Setup Wizard** as shown in figure 1.27.

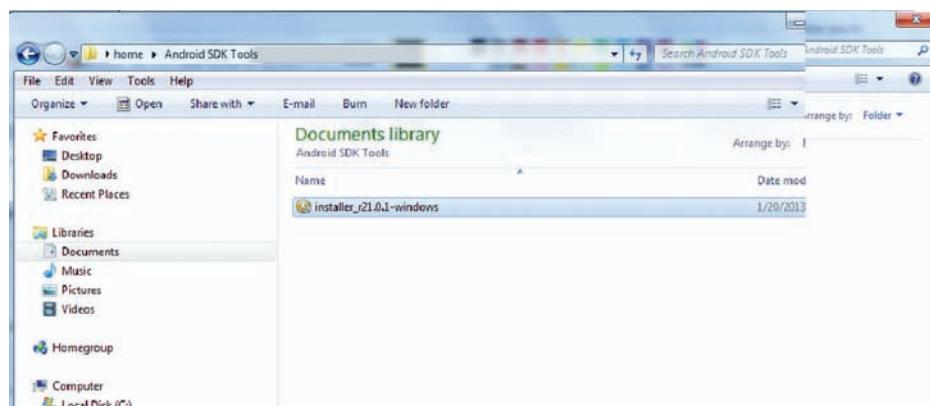


Figure 1.27: Installer File

11. Click **Next** as shown in figure 1.28.



Figure 1.28: Welcome to the Android SDK Tools Setup Wizard

The **Java SE Development Kit** pane of **Android SDK Tools Setup** dialog box is displayed as shown in figure 1.29.

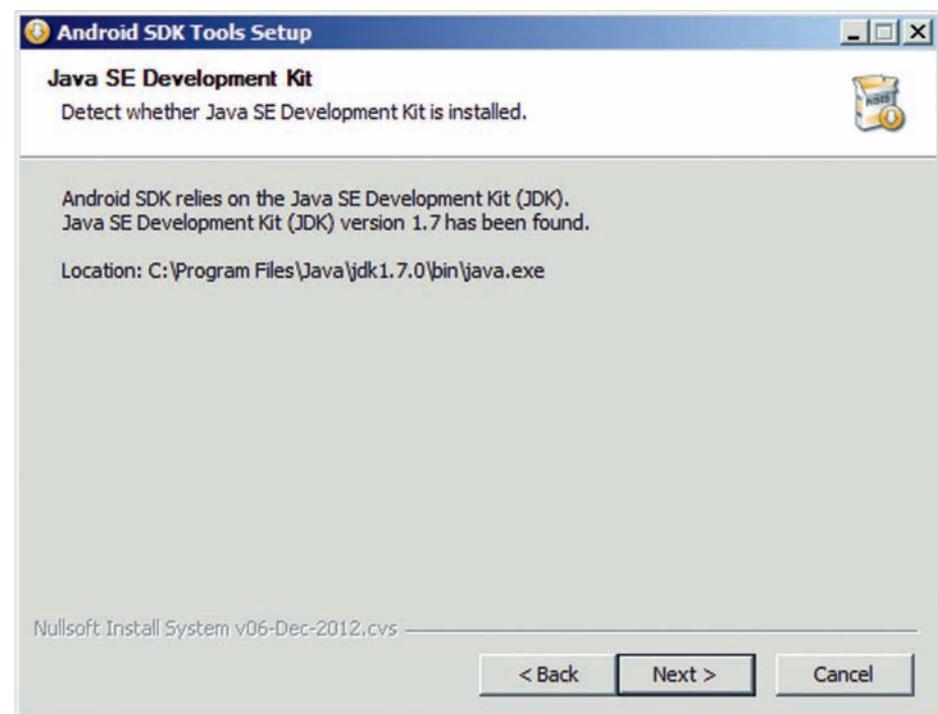


Figure 1.29: Java SE Development Kit Detection Pane

12. Click **Next** to display the **Choose Users** pane of **Android SDK Tools Setup** dialog box.
13. Select **Install for anyone using this computer** and click **Next** as shown in figure 1.30.



Figure 1.30: Choose Users Pane

The **Browse For Folder** dialog box is displayed.

14. Select the **Android SDK Tools** folder and click **OK** as shown in figure 1.31.

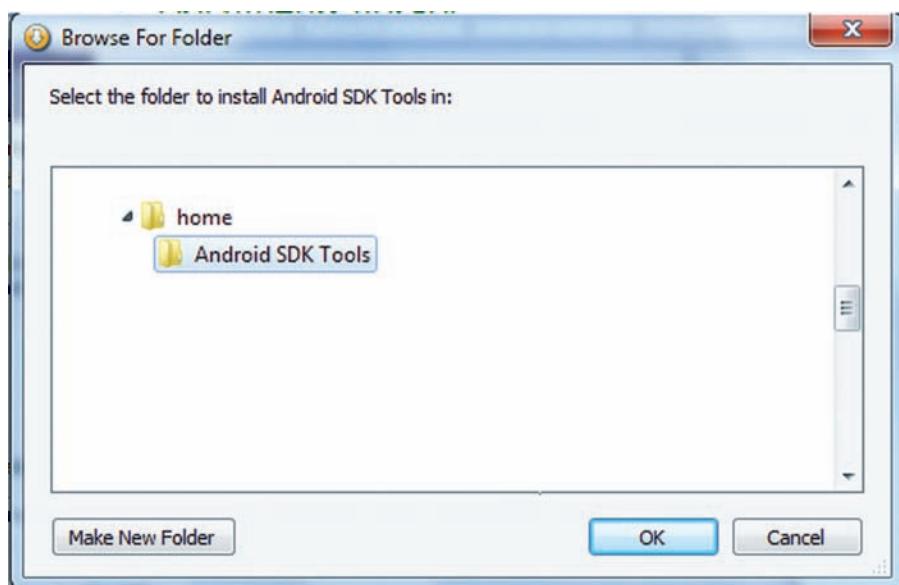


Figure 1.31: Android SDK Tools Folder

15. Select a folder in the **Choose a Start Menu Folder** in the **Android SDK Tools Setup** dialog box. For example, choose **Accessories** and click **Install** as shown in figure 1.32. After the files are installed, the **Installation Complete** window appears.

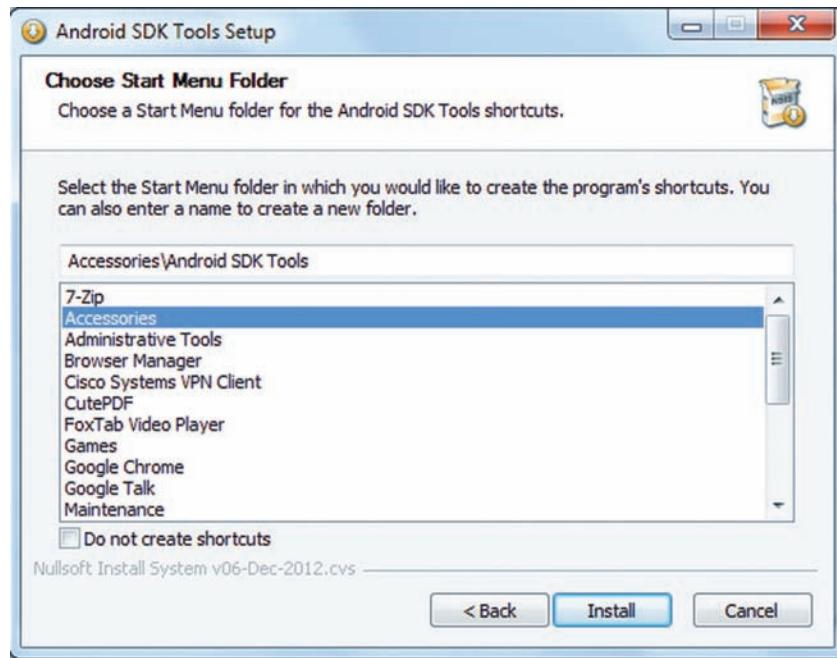


Figure 1.32: Choose Start Menu Folder

16. Click **Next** in the **Installation Complete** pane of **Android SDK Tools Setup** dialog box as shown in figure 1.33.

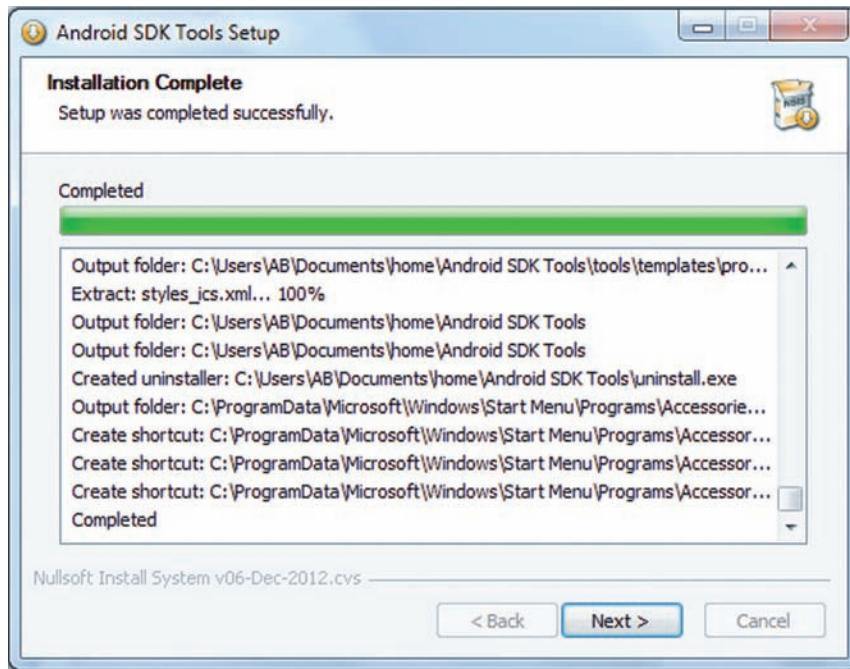


Figure 1.33: Installation Complete

17. Click **Start SDK Manager** to deselect it as shown in figure 1.34.
18. Click **Finish**.

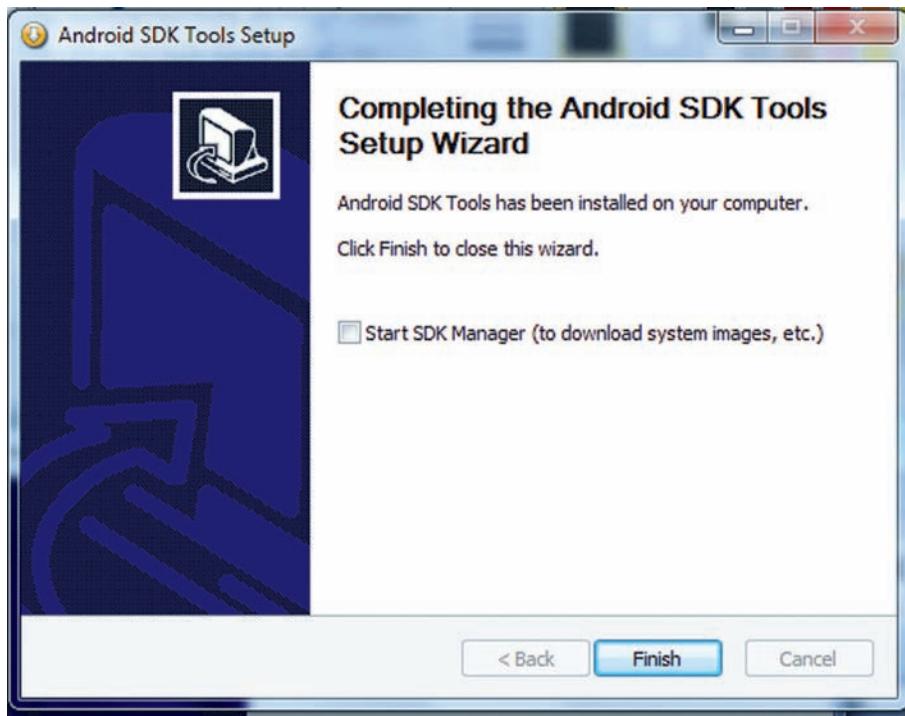


Figure 1.34: Completing the Android SDK Tools Setup Wizard

You will see the Android SDK Manager window with the installation status displayed as shown in figure 1.35.

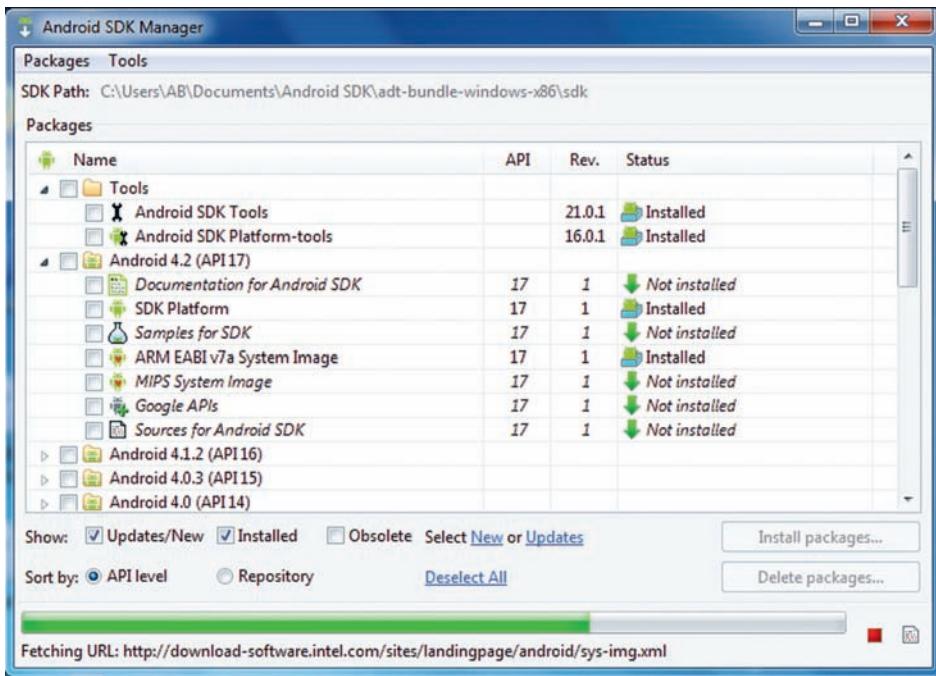


Figure 1.35: Android SDK Manager Installation Status

- **SDK Installation for Platforms Other than Windows:**
 1. Check system requirements.
 2. Go to the **Get the Android SDK** page, at the bottom; click the **DOWNLOAD FOR OTHER PLATFORMS** link. The **ADT Bundle** table appears.
 3. Scroll down and locate the **SDK Tools Only** table.
 4. In the **SDK Tools Only** table, in the **Platform** list, identify the OS.
 5. In the **SDK Tools Only** table, in the **Package** list, click the required compressed file link to download the SDK Tools for your platform.
- **SDK Installation for Macintosh (Mac):**
 1. Decompress the .zip file named **android-sdk_r21.0.1-macosx.zip**. The unzipped files are saved by default in a directory named **android-sdk_r21.0.1-macosx**.
 2. Save the files to a specific directory, for example, **Android SDK** in the **home** directory. Note the name and location of the folder where the SDK files are saved. This will be required for setting up the ADT plugin, and to access the SDK files from the command prompt.
 3. If the IDE is **Eclipse**, go to steps for **Installing the Eclipse ADT Plugin**.
 4. If the IDE is other than Eclipse, go to the <http://developer.android.com/sdk/installing/index.html> page, and read the **Adding Platforms and Packages** tutorial.

Note - Internet links are subject to change. If the link provided does not function, go to the <http://www.android.com> Web page, and navigate to the Developers page to find the updated link.
- **SDK Installation for Linux:**
 1. Decompress the .tgz file named **android-sdk_r21.0.1-linux.tgz**. The unpacked files are saved by default to a directory named **android-sdk_r21.0.1-linux**. Note the name and location of the folder where the SDK files are saved. This will be required for setting up the ADT plugin, and to access the SDK files from the command prompt.
 2. If the IDE is **Eclipse**, go to steps for **Installing the Eclipse ADT Plugin**.
 3. If the IDE is other than Eclipse, go to the <http://developer.android.com/sdk/installing/index.html> page, and read the **Adding Platforms and Packages** tutorial.

Note - Internet links are subject to change. If the link provided does not function, go to the <http://www.android.com> Web page, and navigate to the Developers page to find the updated link.

→ Installing the Eclipse ADT Plugin:

The ADT is Android's customized plugin for the Eclipse IDE. It offers a dynamic and all-inclusive environment for developing Android apps. The ADT makes it easier to create new projects, construct a UI for the apps, debug, and export application packages (APK). The Eclipse ADT plugin is required if the developer has already installed the Eclipse IDE, but has not downloaded and installed the ADT Bundle.

To install the Eclipse ADT plugin, go to eclipse.org/mobile and download the installation file.

To install the ADT Plugin for Eclipse IDE, perform the following steps:

1. Start **Eclipse** as shown in figure 1.36.

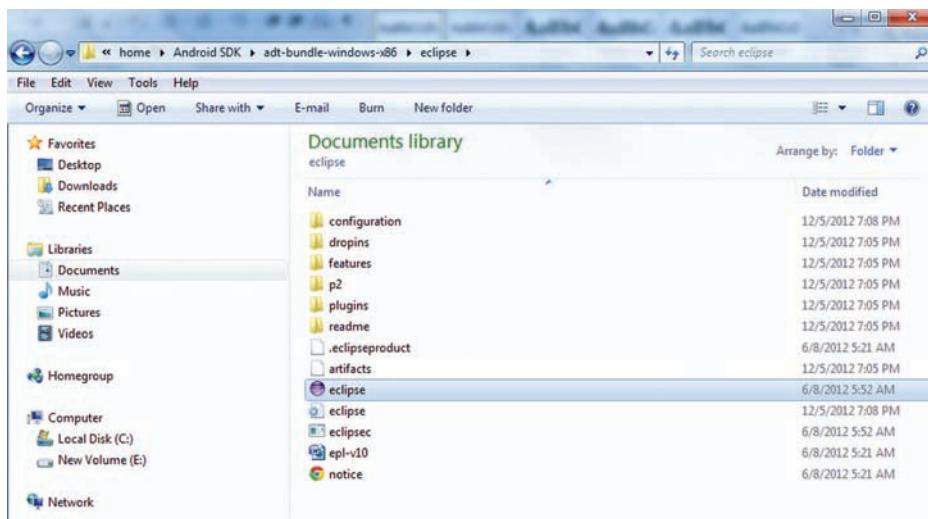


Figure 1.36: Starting Eclipse

2. Click **Help → Install New Software** as shown in figure 1.37.

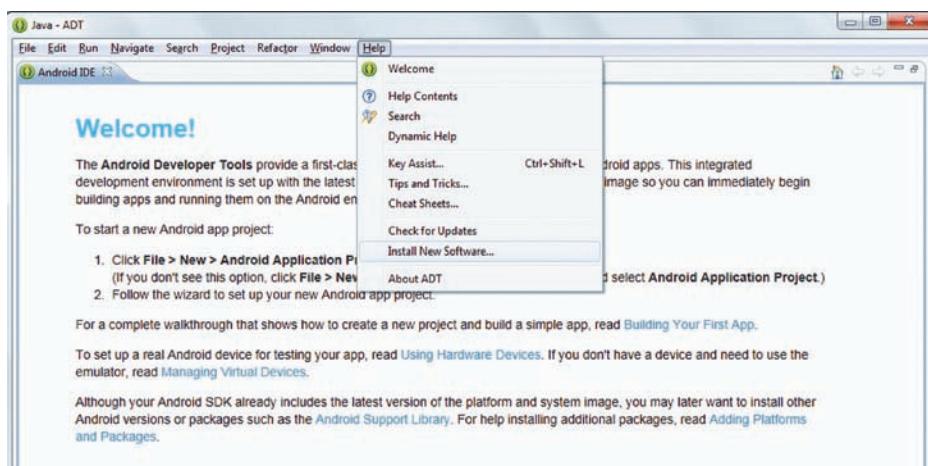


Figure 1.37: Help → Install New Software

3. In the **Available Software** pane of the **Install** dialog box, click **Add** next to the **Work with** list, as shown in figure 1.38.

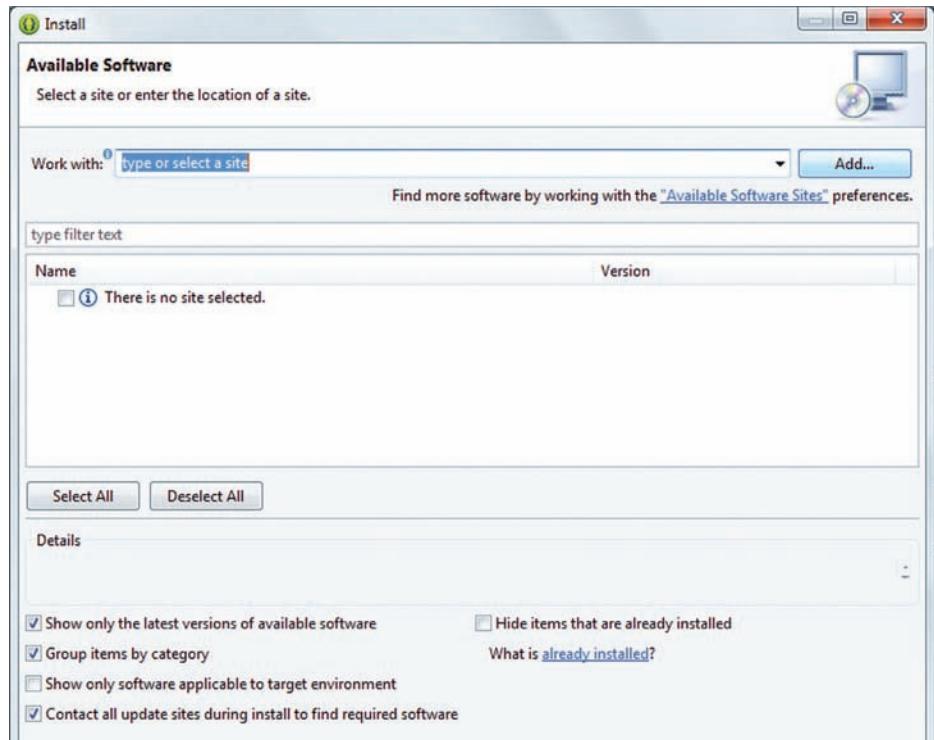


Figure 1.38: Install Available Software

4. In the **Add Repository** dialog box, in the **Name** box, type **ADT Plugin** as shown in figure 1.39.

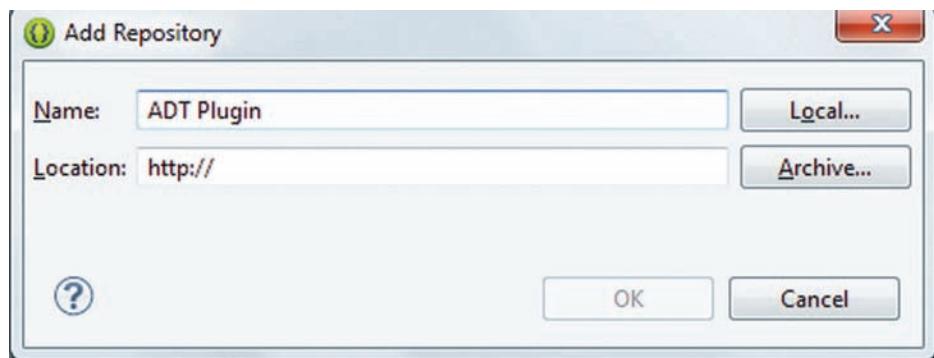


Figure 1.39: Add Repository

5. In the **Location** box, type the URL <https://dl-ssl.google.com/android/eclipse/> and click **OK** as shown in figure 1.40.

Note - If there is a problem in acquiring the plugin, try providing **http://** instead of **https://** in the **Location** box URL. Internet links are subject to change. If the link provided does not function, go to the <http://www.android.com> Web page, and navigate to the **Developers** page to find the updated link.



Figure 1.40: ADT Plugin - Details

6. In the **Available Software** pane of the **Install** dialog box, select **Developer Tools** as shown in figure 1.41.

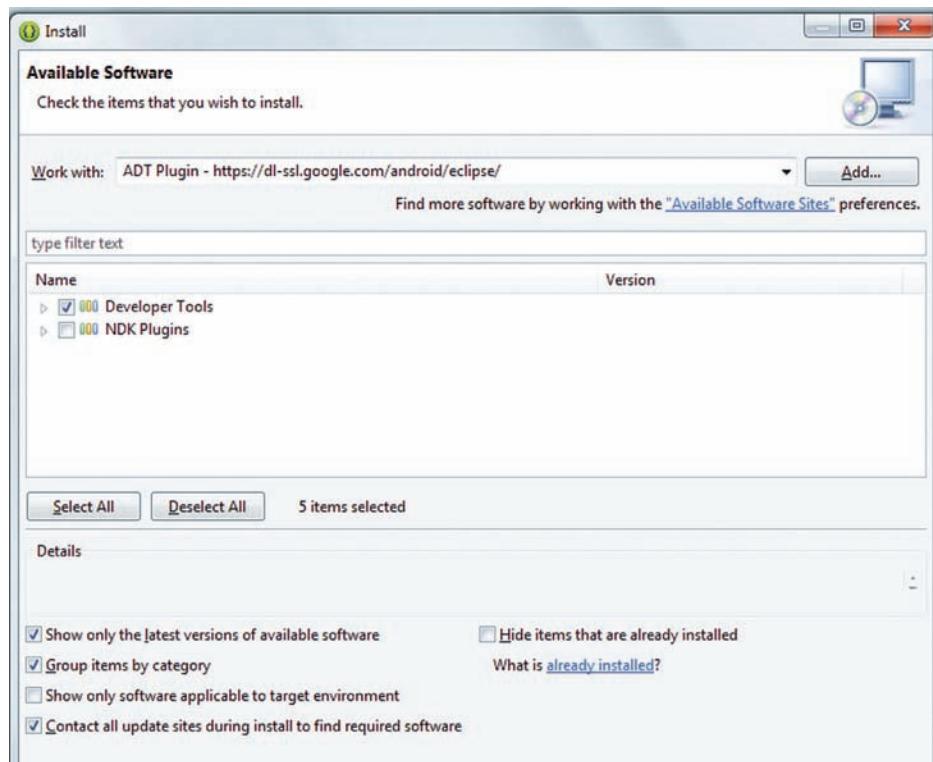


Figure 1.41: Available Software Dialog Box

7. In the **Install Details** pane of the **Install** dialog box, click **Next** as shown in figure 1.42.

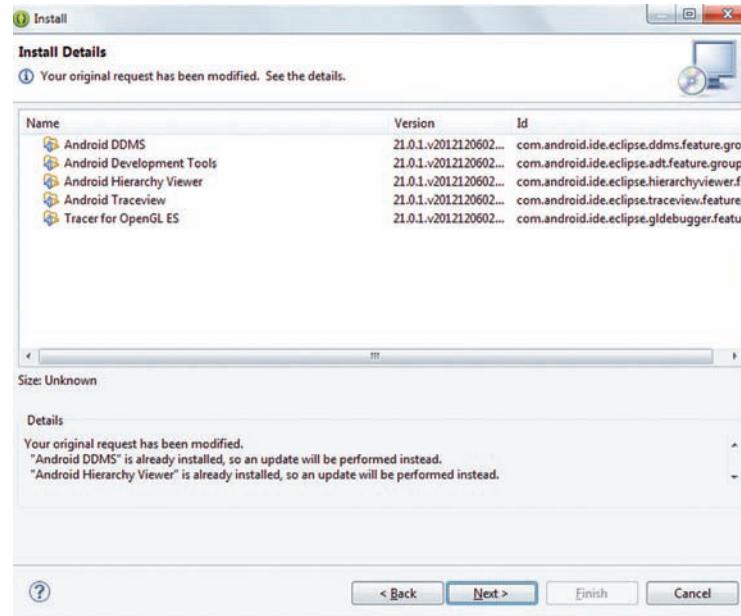


Figure 1.42: Install Details

8. Read and accept the license agreement and click **Finish** as shown in figure 1.43.

Note - If a securing warning appears stating that the authenticity and validity of the software can't be established, click **OK**.

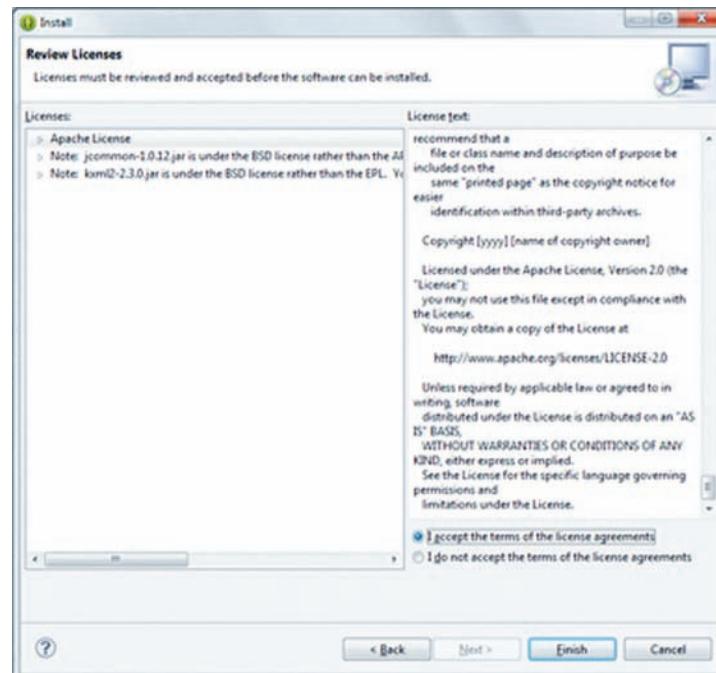


Figure 1.43: Review Licenses

9. After the installation, **Eclipse** restarts automatically. If Eclipse does not restart automatically, click **File → Restart** as shown in figure 1.44.



Figure 1.44: Restarting Eclipse

10. In the **Select a workspace** pane in the **Workspace Launcher** dialog box, browse and select the Android SDK directory where the SDK was downloaded and unpacked as shown in figure 1.45. Click **OK**.

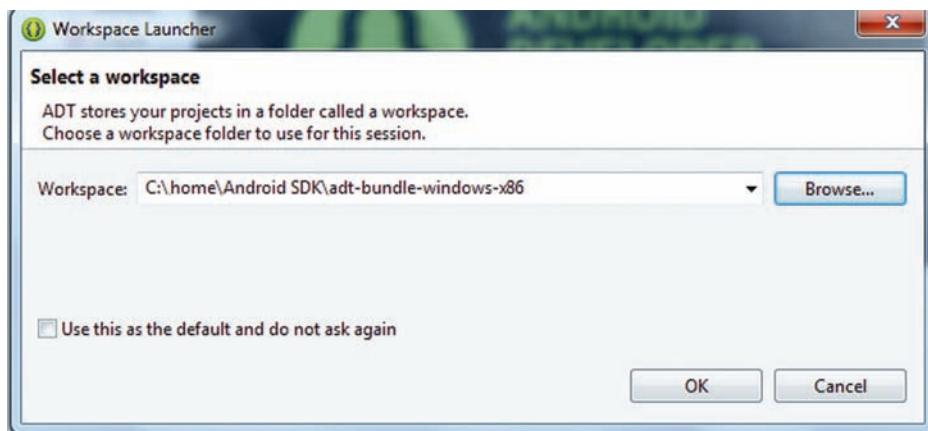


Figure 1.45: Select a Workspace

The installation is complete as shown in figure 1.46.

Note - To get the latest SDK Tools and packages, go to [Adding Platforms and Packages](#). Internet links are subject to change. If the link provided does not function, go to the <http://www.android.com> Web page, and navigate to the Developers page to find the updated link.

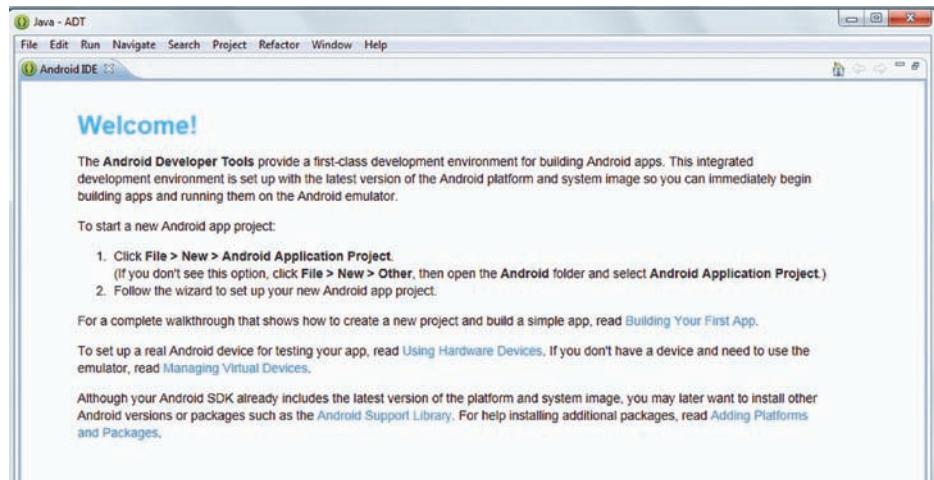


Figure 1.46: Eclipse – Welcome Page

1.5 Check Your Progress

1. Which of the following options defines Android correctly?

(A)	Mobile Phone	(C)	Operating System (OS) for mobile devices
(B)	Google's counterpart to the iPhone	(D)	A single application layer

2. Android applications are written in _____ programming language.

(A)	C	(C)	Perl
(B)	C++	(D)	Java

3. With which version could Android be used with tablet PCs?

(A)	Android Froyo	(C)	Android Jellybean
(B)	Android Honeycomb	(D)	Android Ice Cream Sandwich (ICS)

4. Which of the following features are unique to Android?

(A)	Wireless Connectivity	(C)	SQLite Database
(B)	Touch Screen	(D)	Removable Battery and Storage

5. Match the Android architecture layer with the correct component.

Architecture Layer		Component	
(A)	Linux Kernel	1.	Activity Manager
(B)	Libraries	2.	Camera Driver
(C)	Application Framework	3.	Browser
(D)	Applications	4.	Surface Manager

(A)	A-2, B-4, C-1, D-3	(C)	A-3, B-1, C-2, D-4
(B)	A-1, B-2, C-3, D-4	(D)	A-4, B-3, C-2, D-1

6. Which of the following options correctly represent the features of the Dalvik Virtual Machine (DVM)?

(A)	Runs applications that require minimum processing power and memory	(C)	Runs .dex files
(B)	Runs .class files	(D)	Runs multiple applications as a single process

7. Which of the following options represents the correct system requirements options that should be fulfilled for installing the Android Software Development Kit (SDK) for the Windows OS?

(A)	Java Runtime Environment (JRE)	(C)	2GB Hard Disk Space
(B)	Java Development Kit (JDK)	(D)	Eclipse

1.5.1 Answers

1.	C
2.	D
3.	B
4.	C, D
5.	A
6.	A, C
7.	B, D



- Android refers to an open source Operating System that is largely used in mobile devices that have the touch screen feature.
- Android is popular because it provides developers a flexible and easy to use platform, and users a variety of Google apps.
- Android Inc was started in 2003 and was acquired by Google in 2005. Android has released several versions of the product and the latest version is Android 4.2.
- The platform provides support for a variety of features and options. It also has several advantages compared to its competitor, iPhone. Automation of tasks, customizable home screen, removable battery and storage, access to a wide variety of free applications, and open source platform are some of the advantages of Android over iPhone.
- Android architecture consists of four layers that interact with each other and offer a robust, efficient, and flexible platform.
- Android Software Development Kit (SDK) is easy to download and installs set of tools that helps to build applications for Android.
- Developer can download and install the ADT Bundle, which comes with the Eclipse IDE, or only the SDK Tools, if another IDE or Eclipse is already installed.



1. Find a touch screen mobile phone that has the Android platform installed. Browse through the User Interface (UI), Messaging menu, and the Camera. Find out how these options work. Assume you are a developer and complete the following tasks:
 - Find the processes that run each of the applications in the UI, messaging menu, and the camera.
 - For each application, identify the corresponding Android architecture layer element that supports the application.
2. On an Android phone, use the Web browser to download and install a new application.
3. Identify and list the different hardware installed in the Android phone.
4. Find an Android phone and an iPhone. Compare the features of both phones.
 - Identify the features that Android phone has and iPhone does not have.
 - Identify the features that iPhone has and Android phone does not have.
5. On a PC with Windows OS, download the Android SDK and complete the installation steps.

“First say to yourself what you would
be; and then do what you have to do”

JELLYBEAN
ICE CREAM SANDWICH

Session - 2

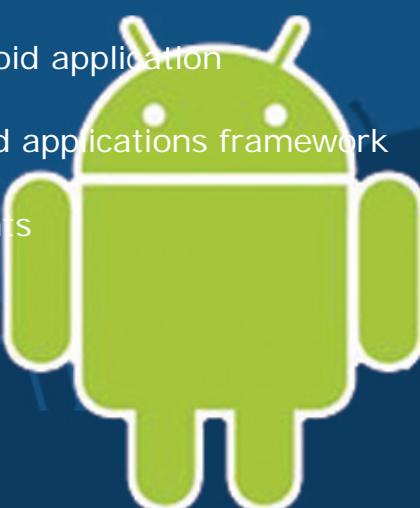
Getting Started with Android

Welcome to the Session, **Getting Started with Android**.

This session explains how developers can start using the Android platform with the Eclipse Integrated Development Environment (IDE). It proceeds further to explain how to create an Android application, which includes creating an Android project, an Android Virtual Device (AVD), launch configurations, running and debugging the project, and understanding the project. It also describes the Android application fundamentals and its composition, communication components, and pre-existing components.

In this session, you will learn to:

- ➔ Explain the process of creating an Android application
- ➔ Explain the fundamentals of Android application
- ➔ Explain the composition of Android applications framework
- ➔ Explain communication components
- ➔ Explain pre-existing components



2.1 Introduction

A new Android application developer needs to understand the basics of creating an Android application. In order to get started with Android, the developer should first learn to create a simple project, create the AVD and launch configurations, and run the application. The developer should also understand the application fundamentals and its composition.

The developer can create an Android application, either using the Eclipse IDE, or any other IDE. Before executing the application the developer needs to create an AVD, as the AVD represents an emulator or model of devices, such as, the tablet or Smartphone. Creation of the AVD will generate a virtual device that the developer can use to test the application.

In Eclipse IDE, to run an Android application, a run configuration needs to be created. The run configuration specifies which project should be executed, the activity that needs to be launched, and the specific device the application should connect to. For the AVD to launch the configuration correctly, the developer needs to create launch configurations. Once the AVD and launch configurations have been created, the developer can run and debug the application.

As discussed earlier, developers need to write Android applications using Java syntax. A majority of the features present in core Java Application Programming Interfaces (APIs) are found in Android's core libraries. This session describes the different components of Android applications. The various components of an Android application are bound together by the Android manifest file. The Android manifest file is also explained in detail.

2.2 Creating an Android Application

Creating an Android application first involves understanding what an Android project is. The Android project contains the source code for an Android application. The developer needs to create an Android project using the Eclipse IDE, or another IDE. This section describes how to create an Android application using the Eclipse IDE in a Windows Operating System (OS).

2.2.1 What is an Android Project?

An Android project essentially consists of the entire source code for an Android application. The project helps to create an **.apk** file for installation of the application on any device. When the Android Software Development Kit (SDK) has been downloaded and installed, the process of creating a new Android project becomes simple and easy.

An Android project can be created in two ways:

- **With the Eclipse IDE** – For this, the Android Software Development Kit (SDK) should have been downloaded and installed along with Eclipse ADT plugin.

- **With another IDE** – For this, Android SDK Tools should have been downloaded and installed. The developer can then create the project using the command prompt. For guidelines, visit <http://developer.android.com/training/basics/firstapp/creating-project.html> and scroll down to view the section, *Create a Project with Command Line Tools*.

Note - Internet links are subject to change. If the link provided does not function, go to the <http://www.android.com> Web page, and navigate to the Developers page to find the updated link.

2.2.2 Creating a HelloWorld Android Project Using Eclipse

In order to create a project using Eclipse, the developer should have first installed the Eclipse IDE and ADT plugin. Once Eclipse has been installed, the first application can be created.

Steps for creating a **HelloWorld** Android project using Eclipse IDE in the Windows OS are as follows:

1. Access the **home/Android SDK/adt-bundle-windows-x86 folder/eclipse** folder as shown in figure 2.1.

Note - Access eclipse from the location where the Android SDK installation files were originally saved.

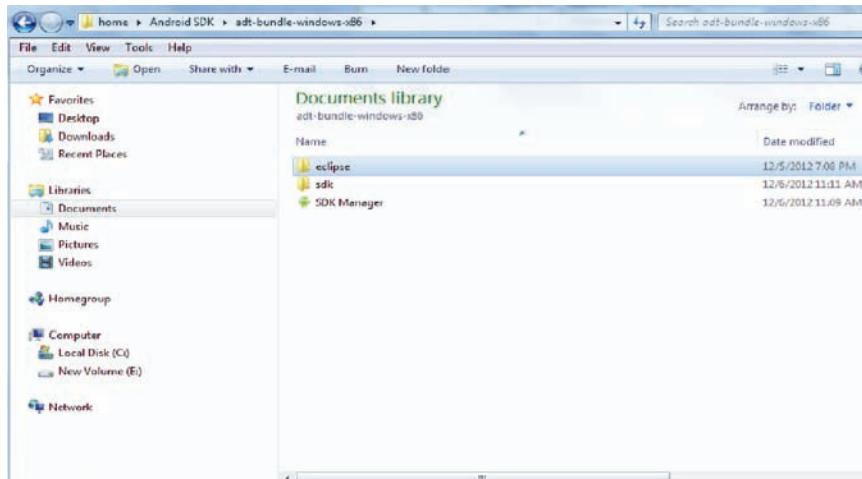


Figure 2.1: Eclipse Folder

2. Double-click the **eclipse** icon to start the IDE as shown in figure 2.2.

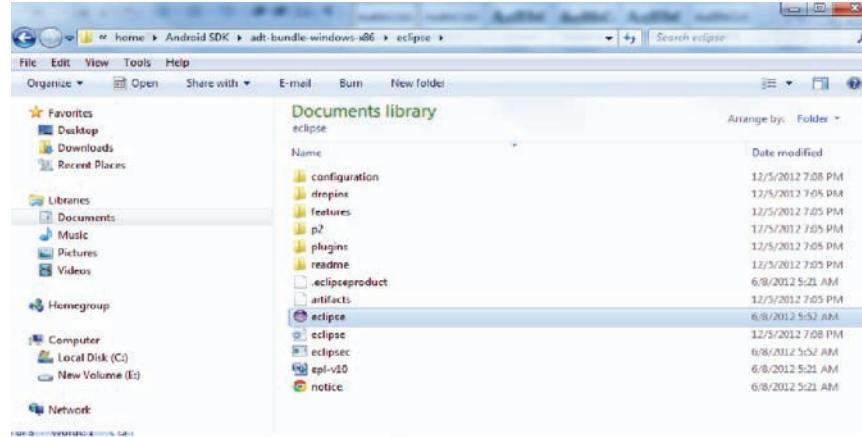


Figure 2.2: Eclipse Icon

3. In the **Workspace Launcher** dialog box, click **Browse** and select the **Android SDK** folder as shown in figure 2.3.

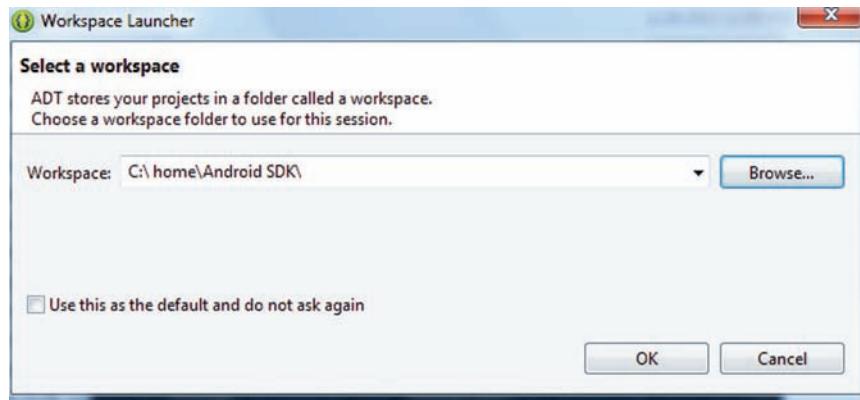


Figure 2.3: Workspace Launcher

4. Click **OK**.

Within Eclipse IDE, the **Android IDE** window is displayed as shown in figure 2.4.

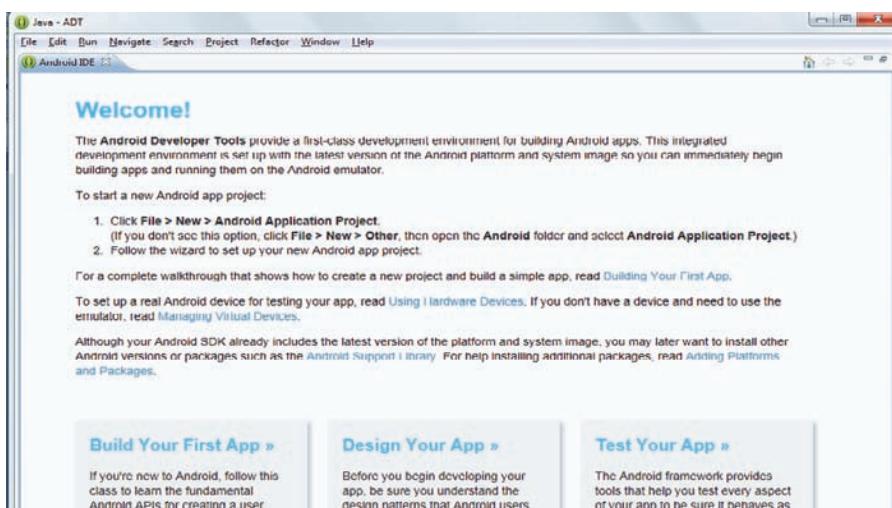


Figure 2.4: Android IDE

- a. Click **File → New → Android Application Project** in Eclipse IDE to start a new Android project as shown in figure 2.5.

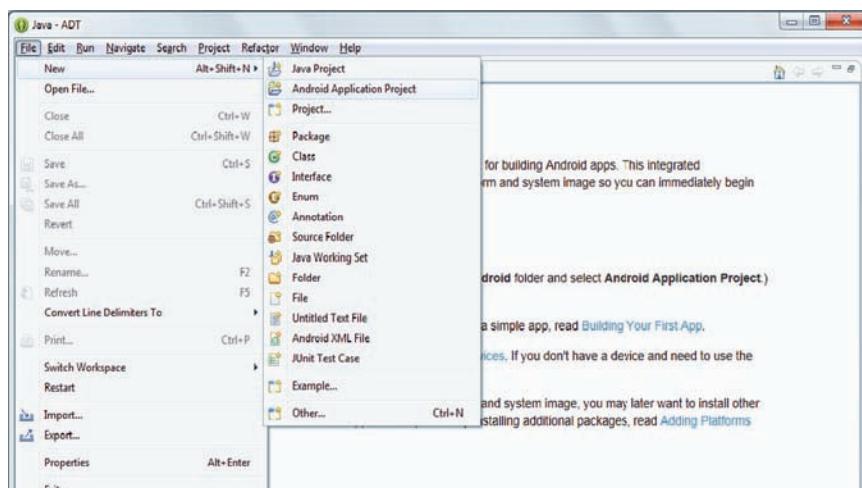


Figure 2.5: New Android Application Project

The **New Android Application** dialog box is displayed as shown in figure 2.6.

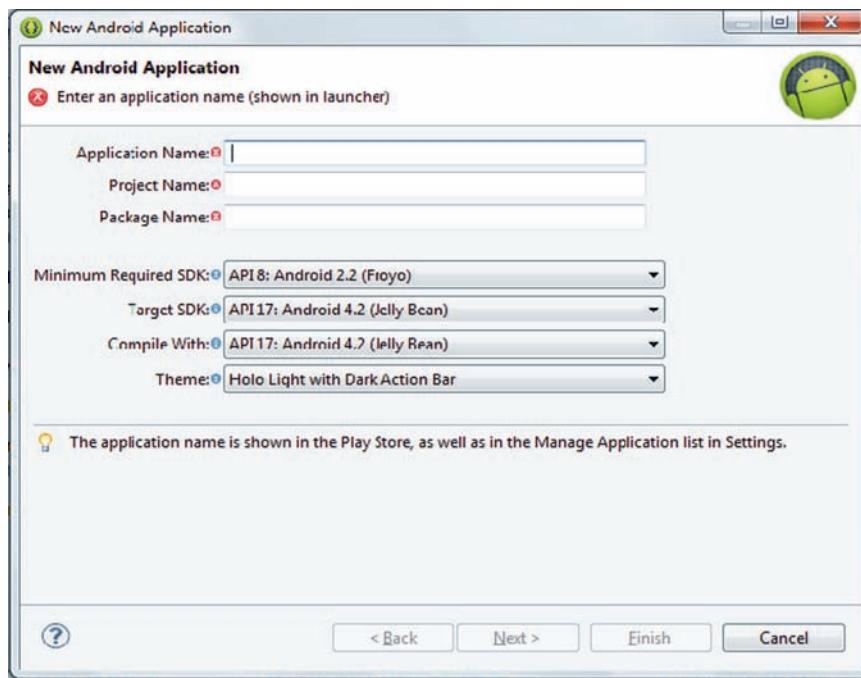


Figure 2.6: New Android Application Dialog Box

5. In the **Application Name** box, type **HelloWorld** as shown in figure 2.7. The **Application Name** indicates the name of the application that the users will view.

When the application name is typed, the **Project Name** and **Package Name** boxes are automatically filled.

The project name is the directory in which the project files are saved and can be viewed in Eclipse.

The package name for the application has to be a unique name and cannot be the same as another package installed in Android. To ensure that the name is unique, the reversed domain name of the organization or other entity is provided.

The **Minimum Required SDK**, **Target SDK**, **Compile With**, and **Theme** boxes are filled by default.

The **Minimum Required SDK** refers to the lowest Android version that the application (app) can support. The **Target SDK** refers to the highest Android version. The **Compile With** option is the version of the platform used to create the app. This is set by default to the highest Android version. **Theme** is the style of the User Interface (UI) for the app being created.

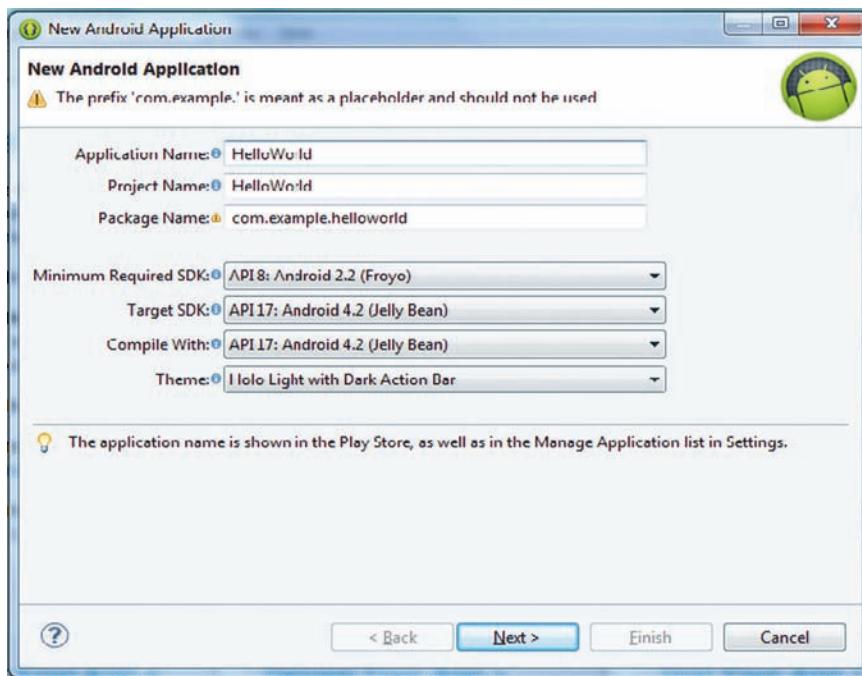


Figure 2.7: New Android Application Parameters

6. Click **Next**.
7. Click **Next** in the **Configure Project** pane of the **New Android Application** dialog box as shown in figure 2.8.

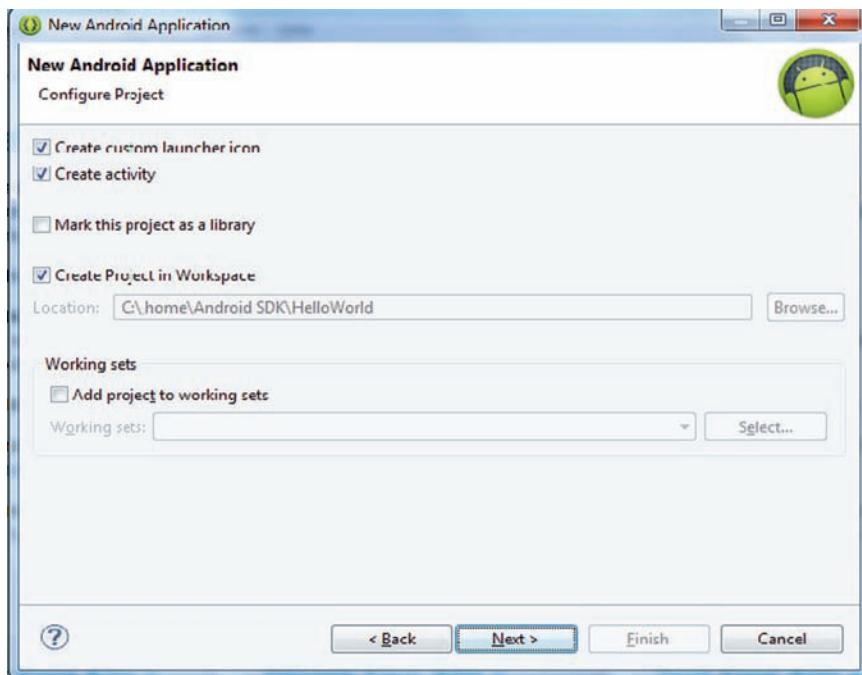


Figure 2.8: Configure Project Pane

The **Configure Launcher Icon** pane of the **New Android Application** dialog box is displayed as shown in figure 2.9.

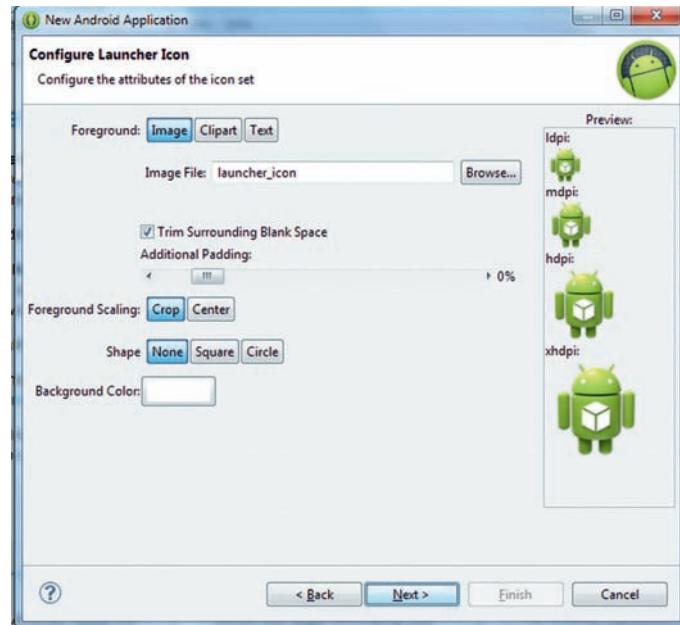


Figure 2.9: Configure Launcher Icon Pane

8. In the **Configure Launcher Icon** pane of **New Android Application** dialog box, customize the appearance of the launcher icon as shown in figure 2.10. After customization, click **Next**.

Note - Ensure that the icon fulfills the specifications provided in the Iconography guide.



Figure 2.10: Launcher Icon Details

9. Click **Next** in the **Create Activity** pane of the **New Android Application** dialog box as shown in figure 2.11. **BlankActivity** is selected by default. This dialog box provides the template for building the app.

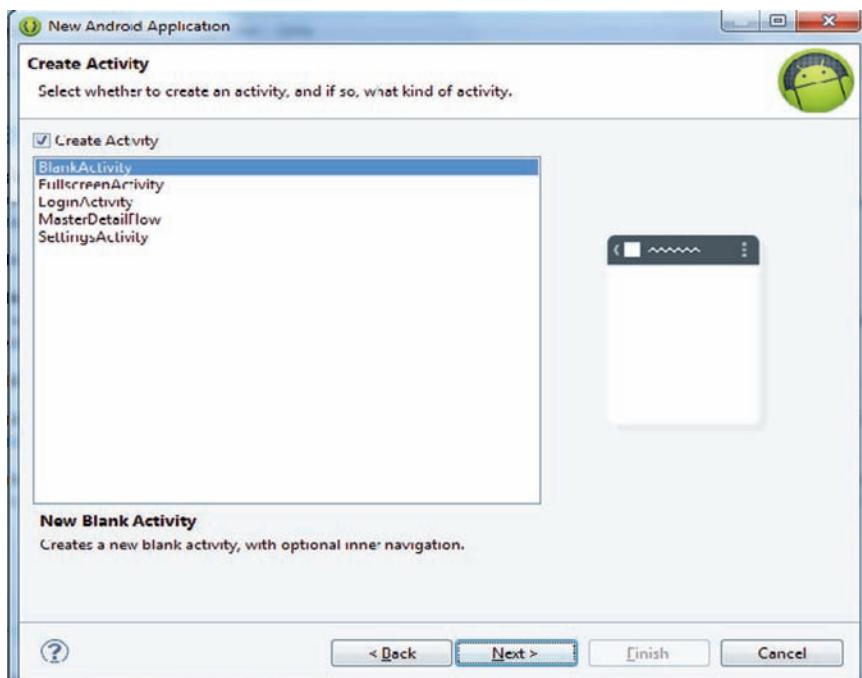


Figure 2.11: Create Activity

10. In the **New Blank Activity** pane of the **New Android Application** dialog box, click **Finish** as shown in figure 2.12.

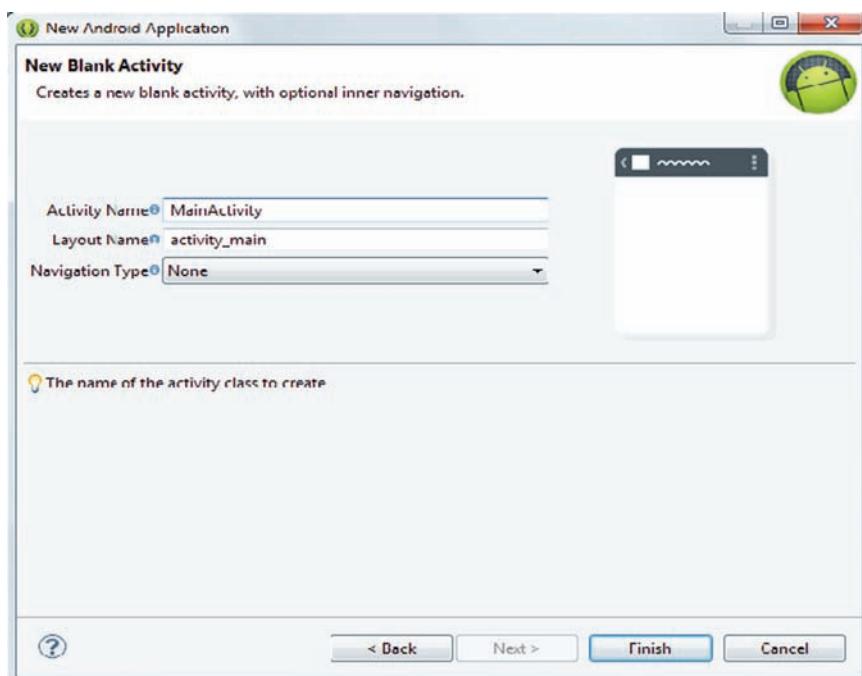


Figure 2.12: New Blank Activity

The Eclipse IDE main window is displayed as shown in figure 2.13. The directory structure for the newly created **HelloWorld** project can be seen in the **Package Explorer** pane of Eclipse IDE as shown in figure 2.13.

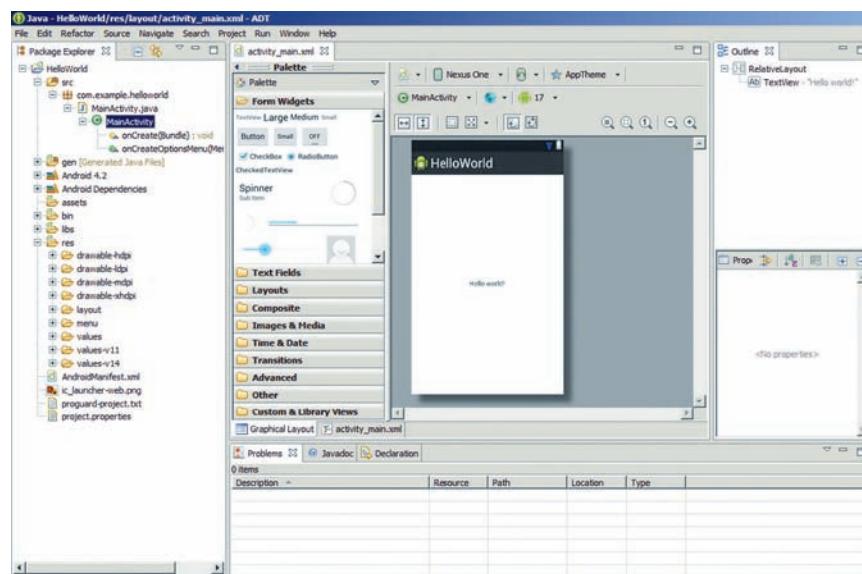


Figure 2.13: HelloWorld Project

2.2.3 Creating Android Virtual Device (AVD)

The AVD is a configuration that enables the developer to sample various devices using an emulator as shown in figure 2.14. The emulator functions exactly similar to a mobile device. The AVD identifies hardware and software options for the model device, which the Android emulator duplicates. The emulator is based on AVD configurations which the developer can also specify. AVD consists of:

- Hardware features for the device, such as, keyboard, camera, processor, and storage.
- Version of Android OS to be used on the emulator.
- Software features for the device, such as, games, social networking apps, e-mail, Internet, Microsoft Outlook, multimedia, and so on.
- The look of the screen, size, specific storage space for data, settings, and applications.



Figure 2.14: Android Emulator

To use an emulator, the AVD configurations need to be created for the model device. The developer can create many AVDs for different types of devices. Each AVD is independent of the others. Once the AVD configuration is created, the developer can run the application on a single or multiple emulators.

→ Steps for creating an AVD in Eclipse:

1. Click **Window → AVD Manager** to start the AVD Manager as shown in figure 2.15.

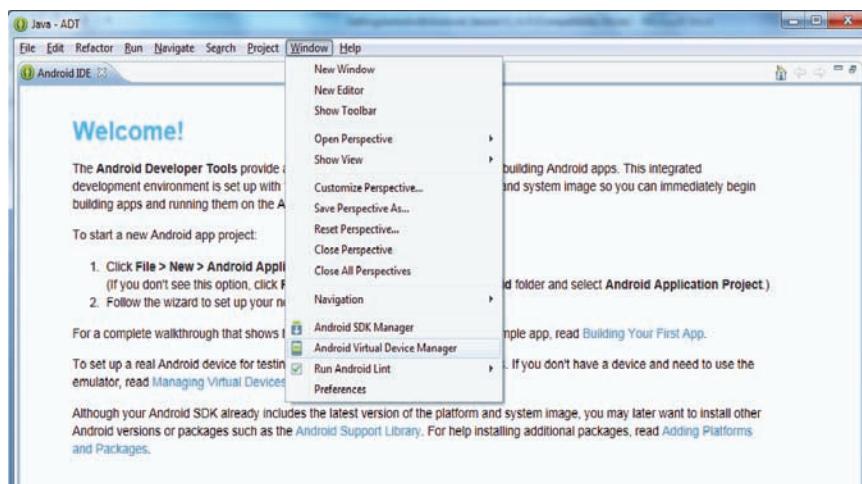
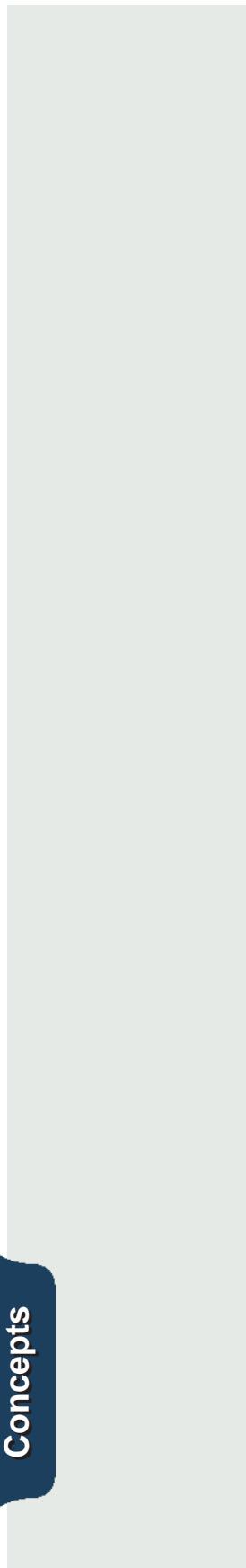


Figure 2.15: Start AVD Manager



The **Android Virtual Device Manager** dialog box is displayed as shown in figure 2.16.

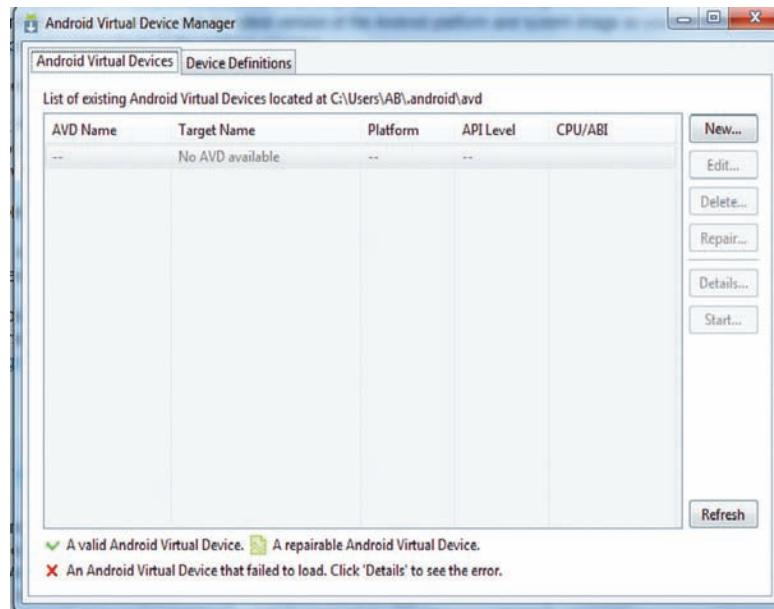


Figure 2.16: AVD Manager

2. Click **New** to create a new AVD.

The **Create new Android Virtual Device (AVD)** dialog box is displayed as shown in figure 2.17.

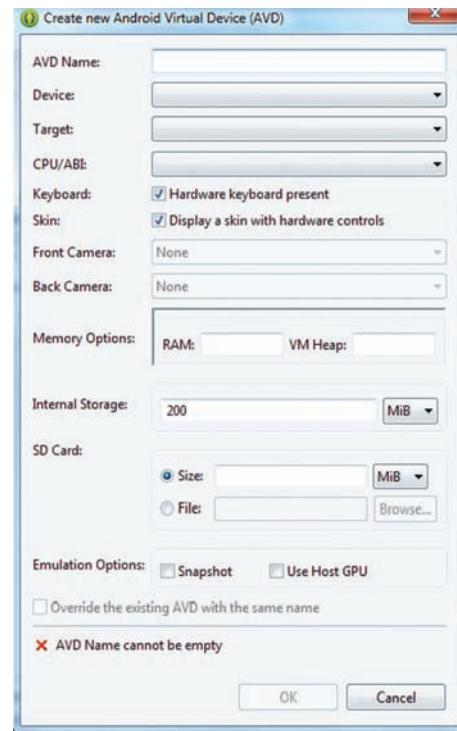


Figure 2.17: Create New AVD

3. In the **Create new Android Virtual Device** dialog box, provide the details for the AVD as shown in figure 2.18.

- In the **AVD Name** box, type a name that is easily identifiable.
- In the **Device** list, select the device that is being emulated.
- When the device is selected, the **Target** is selected by default.
- Complete other details as required, such as, **Keyboard**, **Skin**, **Front and Back Camera**, **SD Card**, and **Emulation Options**. The **Memory Options** field gets filled by default.

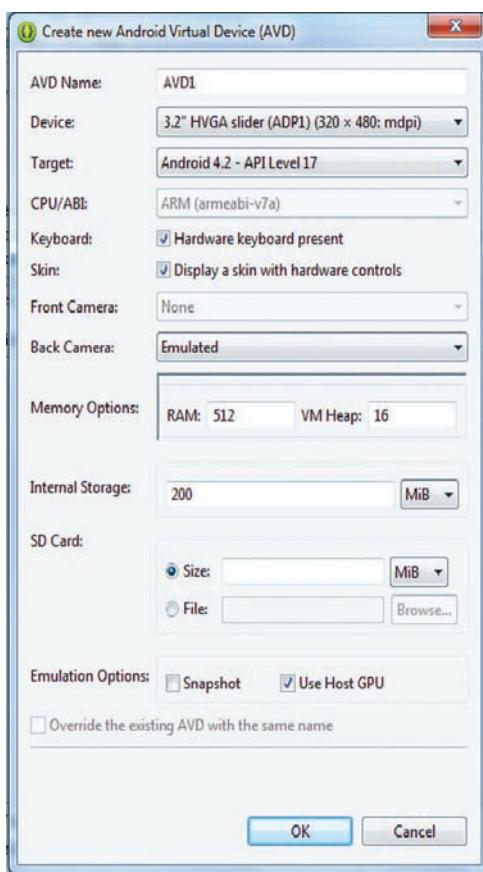


Figure 2.18: Create New AVD Details

- Click **OK** after all the requirements are complete.

The **Android Virtual Device Manager** dialog box appears displaying the newly created AVD, as shown in figure 2.19.

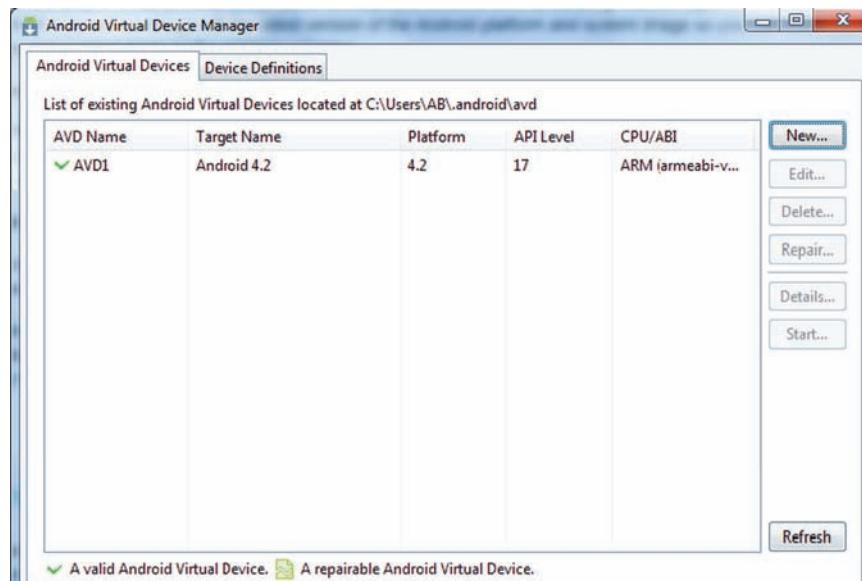


Figure 2.19: AVD Manager with new AVD

- Select the **AVD** and click **Start** to start the emulator as shown in figure 2.20.

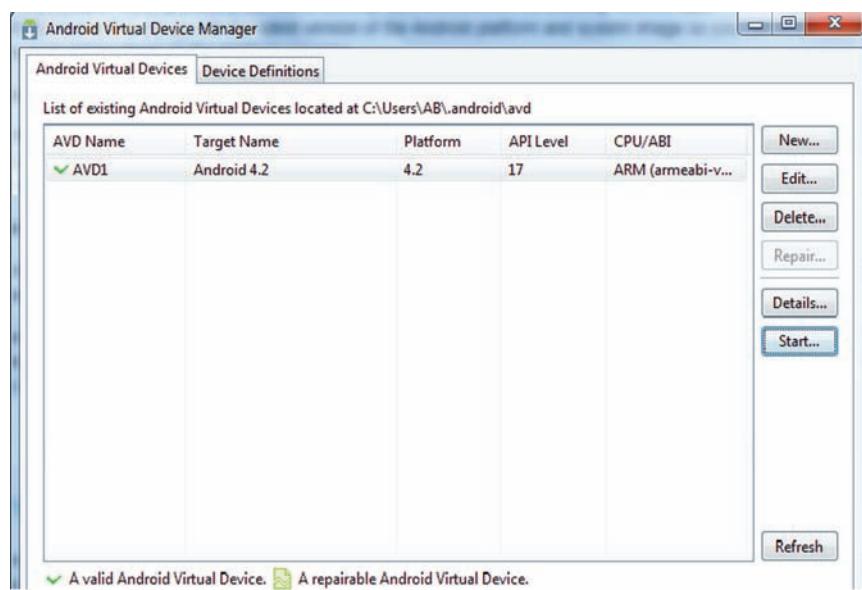


Figure 2.20: Start Emulator

6. Click **Launch** in the **Launch Options** dialog box as shown in figure 2.21.

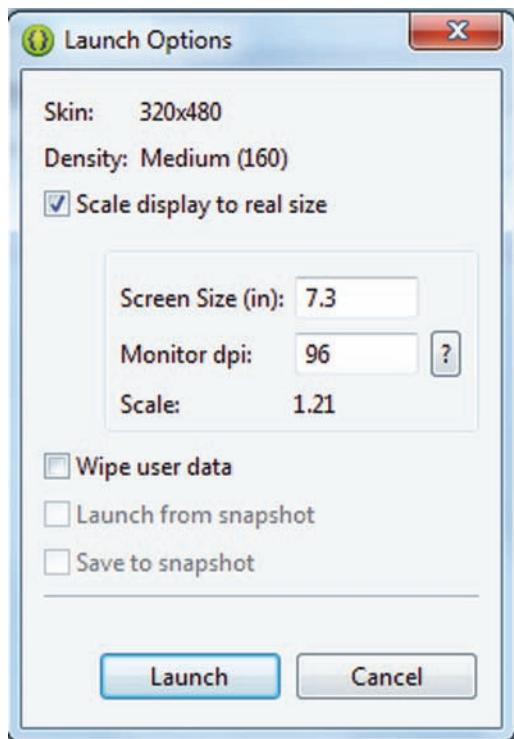


Figure 2.21: Launch Options

The Emulator is launched as shown in figure 2.22.

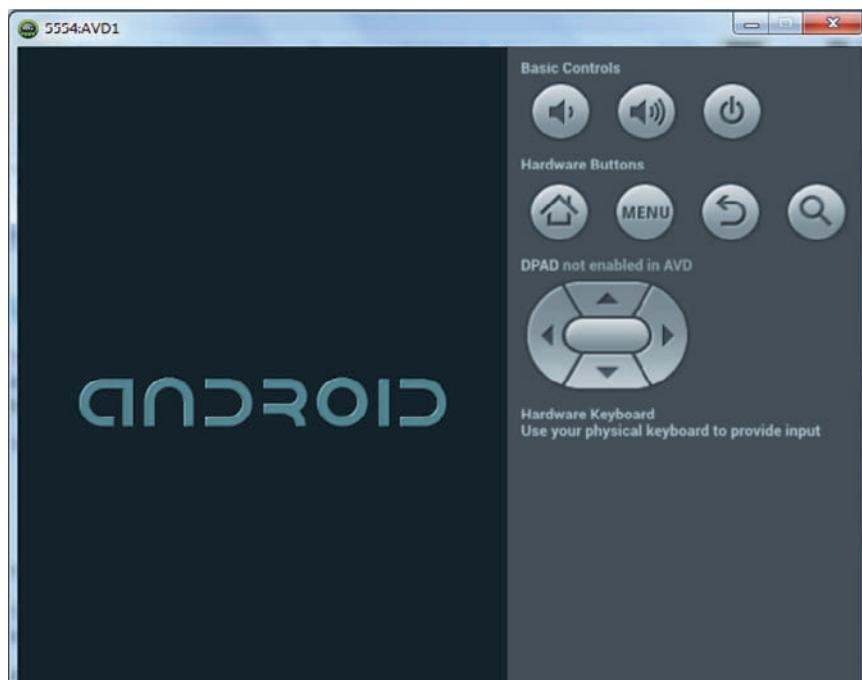


Figure 2.22: Android Emulator

2.2.4 Creating Launch Configurations

Although the emulator has been launched, a Run configuration is required to be set up for the application to run correctly. The Run configuration identifies the project that has been created and is required to be executed, the main activity that needs to be started, and the emulator that is going to be used to run the application. When there is an actual device, the Run configuration identifies the device.

A launch configuration can also include a debug configuration, which enables debugging the application as and when it is being run on a device.

→ Steps to create Run configurations:

1. In Eclipse, click **Run → Run Configurations** as shown in figure 2.23.

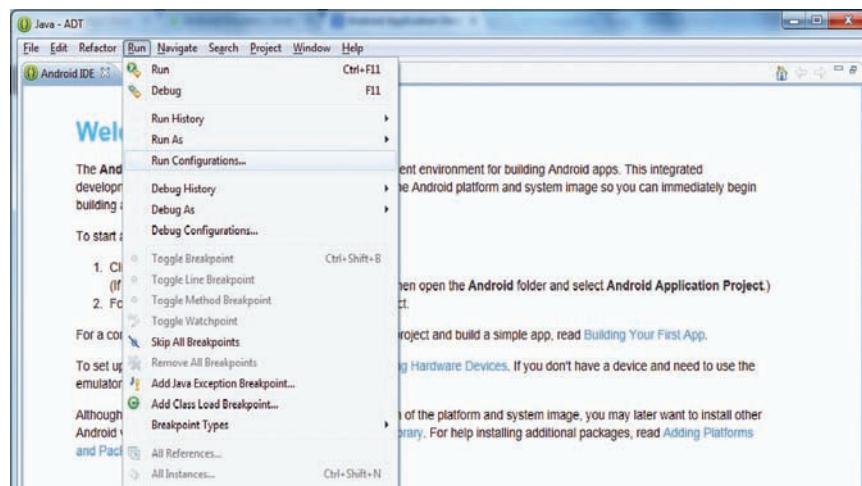


Figure 2.23: Selecting Run Configurations

2. In the **Create, manage, and run configurations** dialog box, select **Android Application** and click **New launch configuration** icon as shown in figure 2.24.

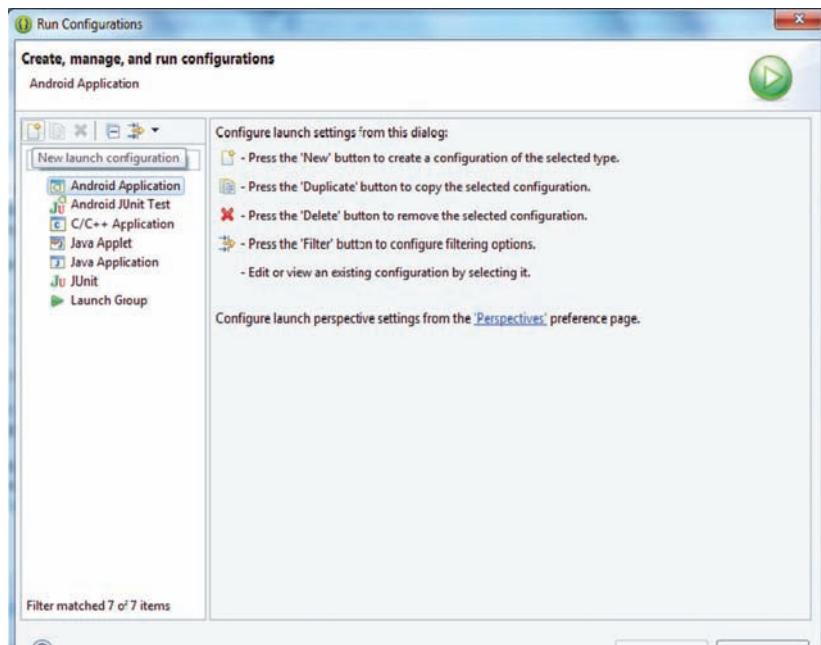


Figure 2.24: New launch configuration Icon

3. In the **Name** box of the **Create, manage, and run configurations** dialog box, you can type the name of the new configuration.
4. Click **Browse** as shown in figure 2.25 to display the **Project Selection** dialog box.

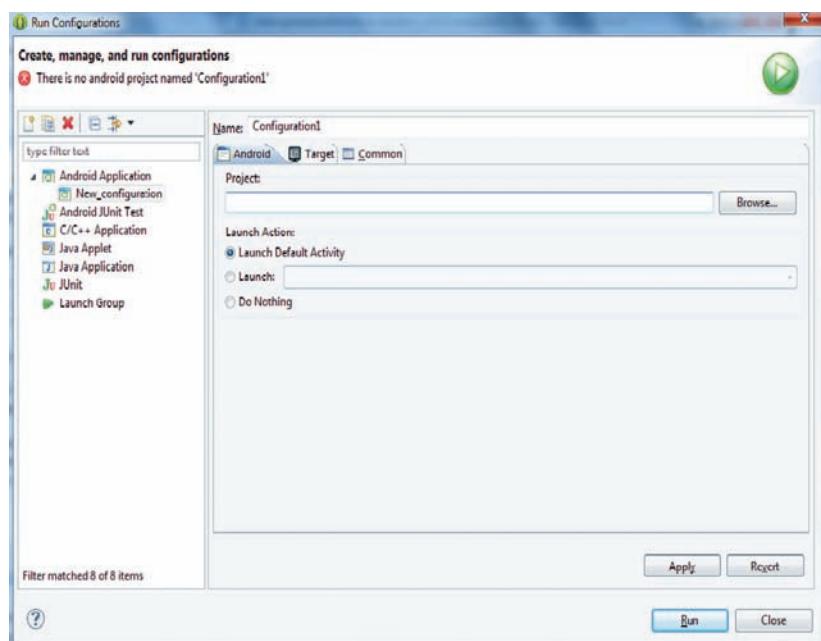


Figure 2.25: Create, manage, and run configurations

5. Select **HelloWorld** and click **OK** as shown in figure 2.26.

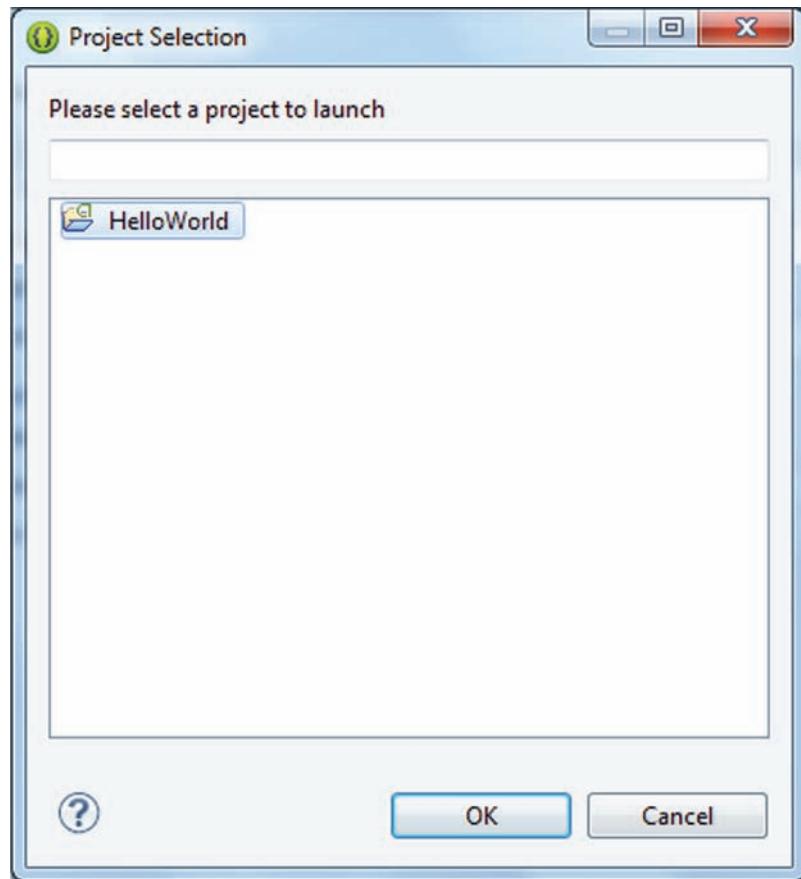


Figure 2.26: Project Selection Dialog Box

The **Create, manage, and run configurations** dialog box now displays the project named, **HelloWorld** as shown in figure 2.27.

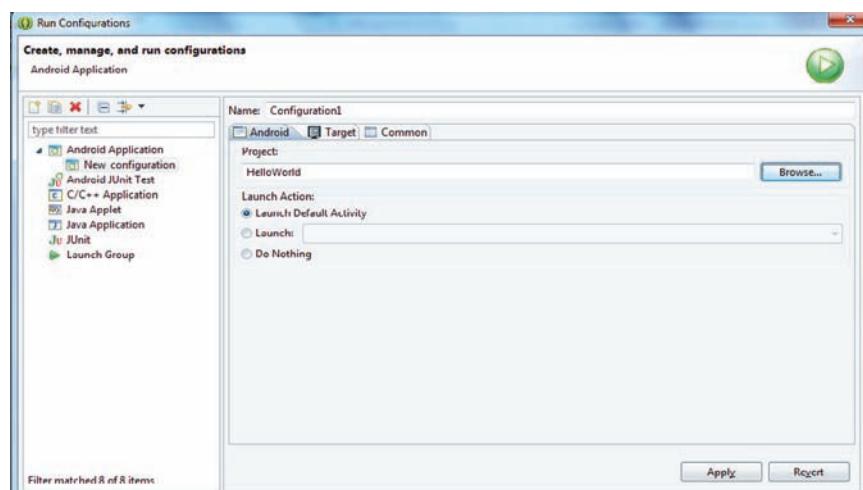


Figure 2.27: Android Project Name

6. Click the **Target** tab in the **Create, manage, and run configurations** dialog box and select the newly created AVD, as shown in figure 2.28.
7. Click **Apply**.

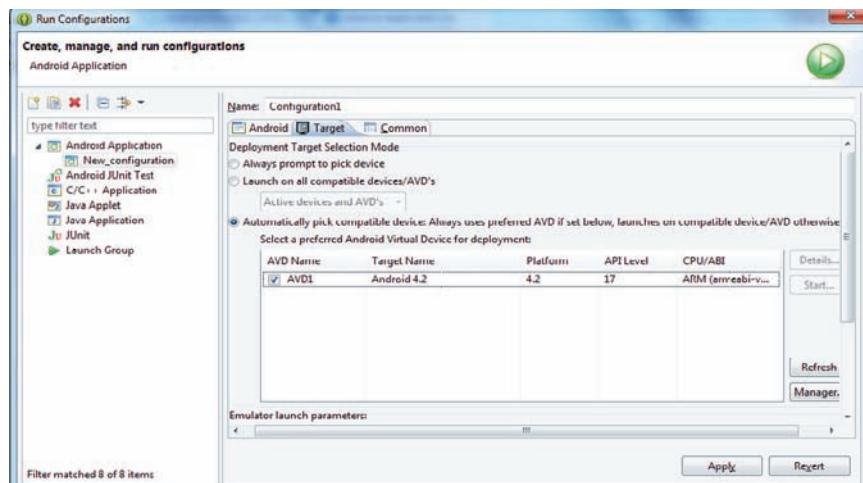


Figure 2.28: Target Tab Options

The launch configuration has been created.

8. Switch to the **Android** tab and click **Close** to exit the **Create, manage, and run configurations** dialog box as shown in figure 2.29.

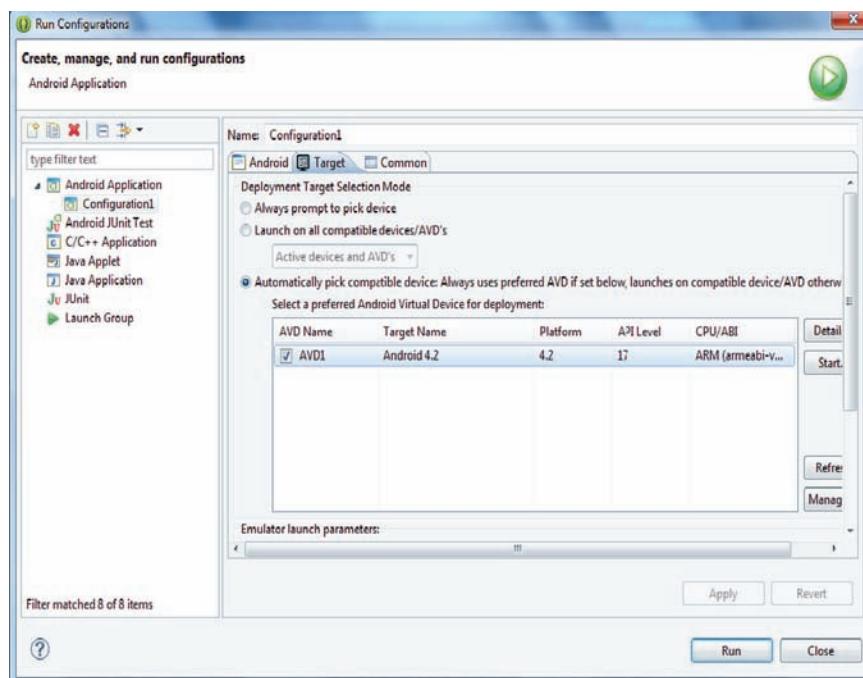


Figure 2.29: Run Configurations Dialog Box

→ **Steps to create Debug configurations:**

The steps for creating a debug configuration are the same as for creating a run configuration. However, when a run configuration has been created for an activity, few steps need to be followed for creating a debug configuration for the same activity.

1. In Eclipse, click **Run → Debug Configurations** as shown in figure 2.30.

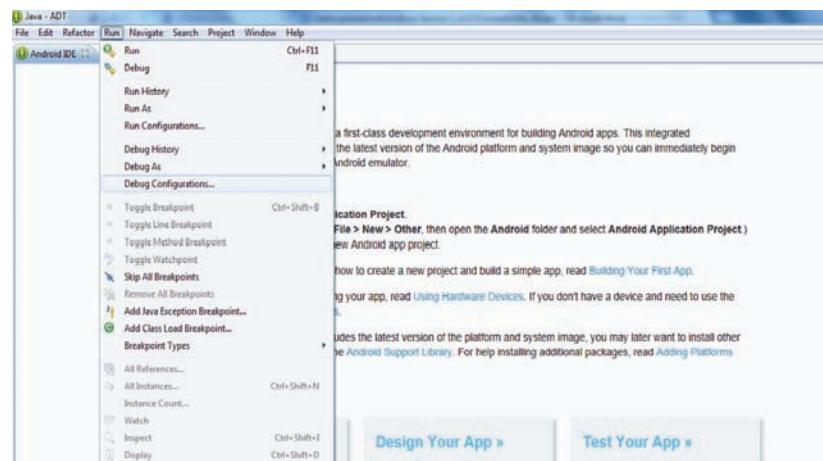


Figure 2.30: Debug Configurations Menu

2. In the **Create, manage, and run configurations** dialog box, in the **Android Application** list, select a configuration as shown in figure 2.31.

In the **Create, manage, and run configurations** dialog box, in the **Android** tab, the project name for the selected configuration is displayed by default.

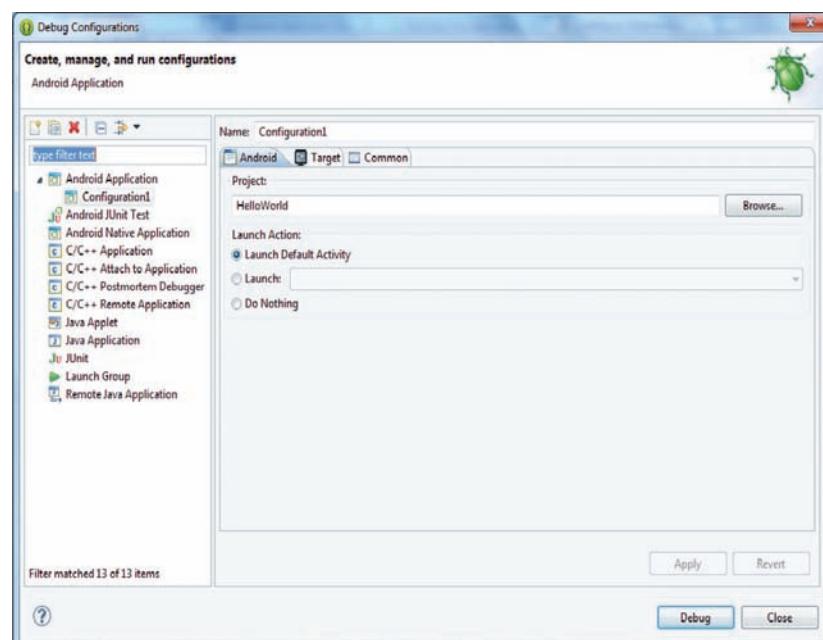


Figure 2.31: Debug Configuration Dialog Box

3. In the **Create, manage, and run configurations** dialog box, click the **Target** tab. The AVD for the configuration is selected by default as shown in figure 2.32.
4. Click **Debug**.

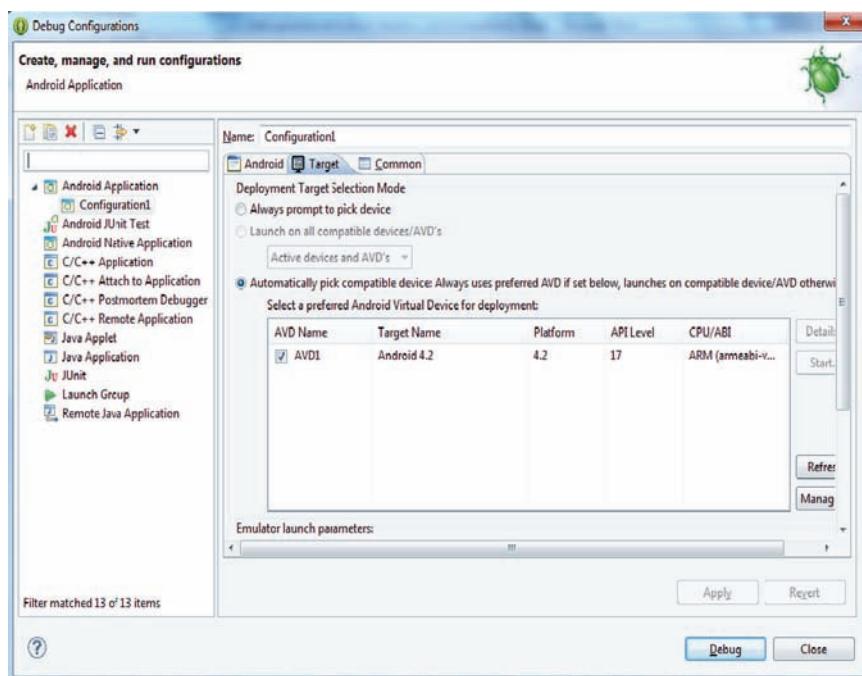


Figure 2.32: Target Tab

The launch configuration has been created.

5. Click **Run → Run** as shown in figure 2.33.

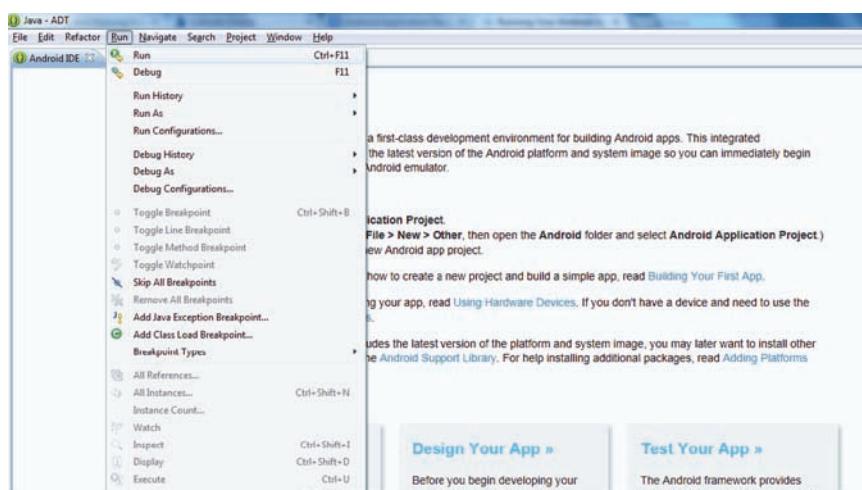


Figure 2.33: Run Application

6. The Emulator is displayed as shown in figure 2.34.

Note - The emulator takes time to load.



Figure 2.34: Emulator

2.2.5 Running and Debugging the HelloWorld Project

There are two ways for the developer to run and debug the **HelloWorld** project. The developer can run or debug the application using an Android device, or through the emulator.

Note - In order to run the app, the developer should ensure that the Android manifest file for the project is present. The file is very important for running all Android applications. In the file, the developer should verify that the highest version of Android is included. To do this, the developer has to alter the `<uses-sdk>` element in the manifest file.

The developer has to ensure that the required directories and files for running the project are present. When the application is built, a **.apk** file is created, which contains the binary code for the application. The **.apk** file consists of all the files and resources required to run the application. The file includes **.dex** files that are Dalvik byte code files, **resource.arsc** file that are compiled and uncompiled resources required by the application, and a binary form of the **AndroidManifest.xml** file.

In Eclipse, the **.apk** file is automatically created in the project's **bin** folder. To run an application using a device or an emulator, the application needs to be signed in the debug or release mode. This is because while building the application, a debug key is created with an identifiable password so that the developer need not keep providing the password while building the application. For better security, however, the developer needs to sign the application using the release mode while releasing the application. This requires designating a private key.

→ **Steps to Run the HelloWorld Project on a Real Device (Windows OS):**

1. Connect the Android device to a machine that has the Android applications installed. The machine must have a Universal Serial Bus (USB) cable using which the Android device is connected. Ensure that the appropriate USB driver is installed in Windows processors.
2. On the Android device, enable debugging for the USB by going to **Menu → Settings → Applications → Development (Settings → Developer Options in newer versions)**.

→ **Steps to Run the HelloWorld Project on the Emulator (Windows OS):**

1. In **Eclipse**, click **Run → Run** as shown in figure 2.35.

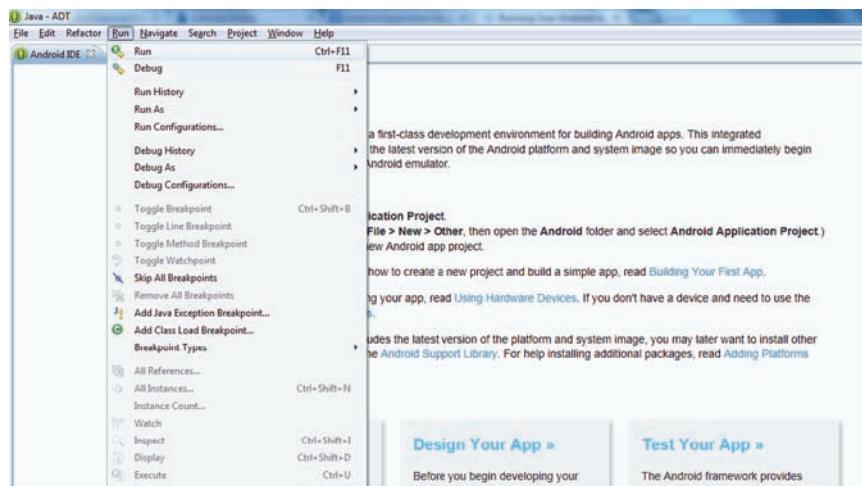


Figure 2.35: Run Menu

The emulator is displayed.

Note - The emulator takes time to load.

2. When the emulator first loads, the home screen is locked as shown in figure 2.36. Click and drag the **lock** icon to the right to unlock the home screen.

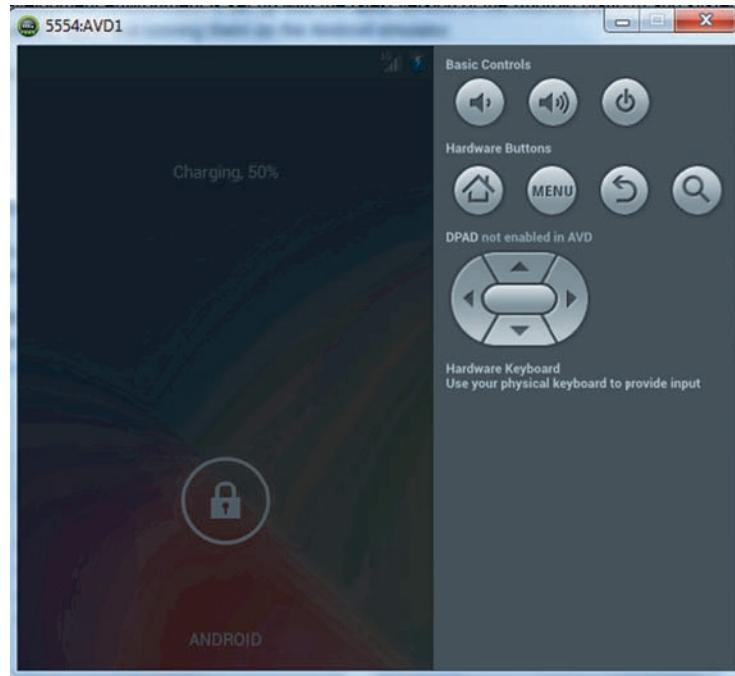


Figure 2.36: Emulator with home screen locked

3. On the home screen, click the **circle** and click **OK** as shown in figure 2.37.

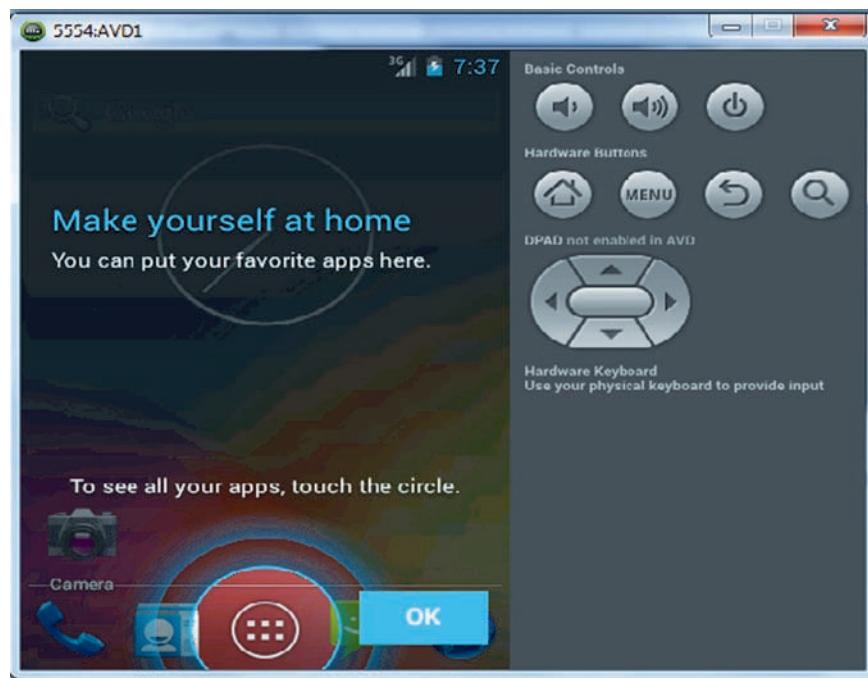


Figure 2.37: Home screen circle

4. When the apps load, click the **HelloWorld** app icon as shown in figure 2.38.



Figure 2.38: HelloWorld App Icon

The text **Hello world!** is displayed on the screen as shown in figure 2.39.

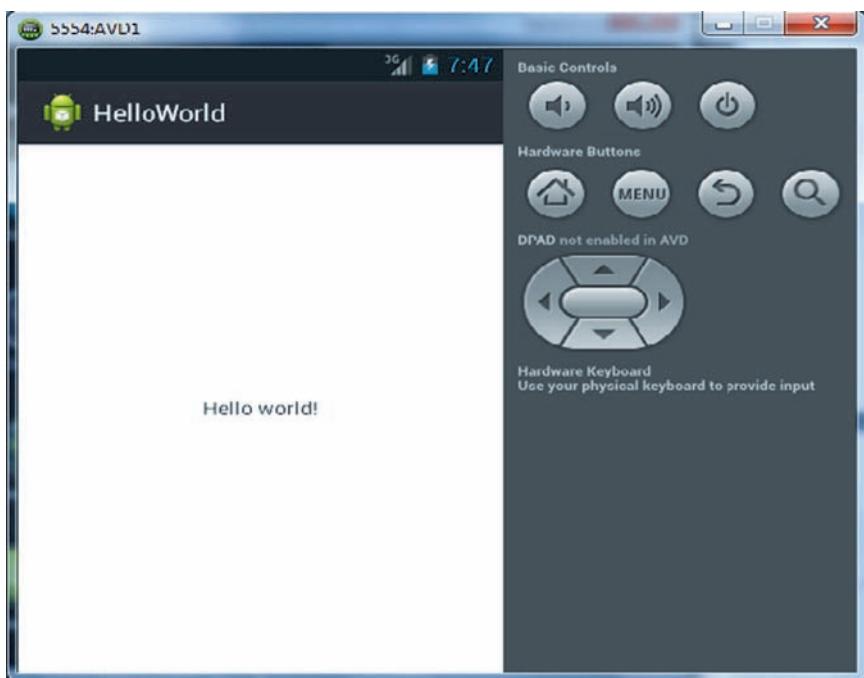


Figure 2.39: HelloWorld Project - Output

→ Steps to Debug the HelloWorld Project in Eclipse (Windows OS):

1. In Eclipse, click Run → Debug as shown in figure 2.40.

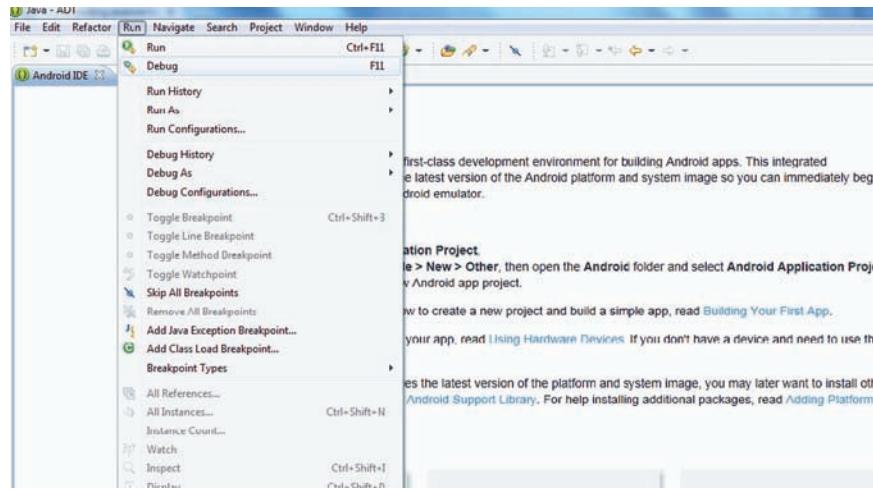


Figure 2.40: Debug Application

The line by line debug reports are displayed in eclipse as shown in figure 2.41.

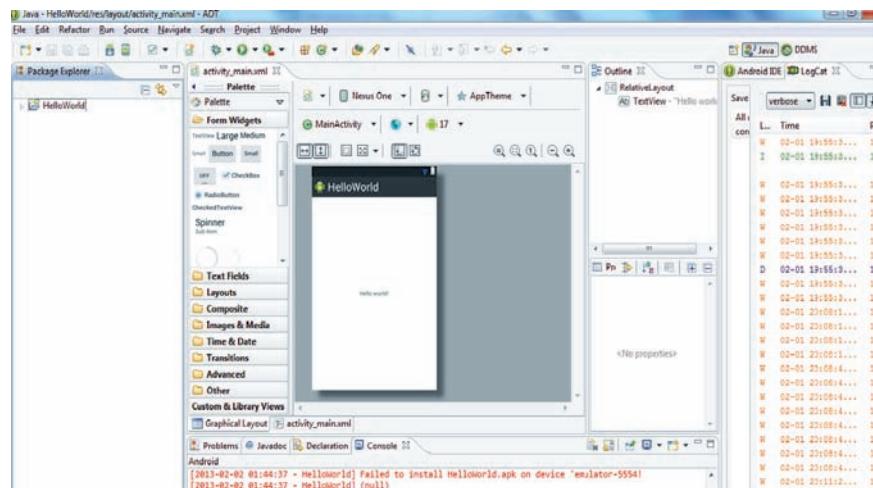


Figure 2.41: Eclipse Debugging Applications

Note - For debugging applications through the emulator, the developer can also go to Run → Debug Configurations.

2.3 Understanding the HelloWorld Project

This section explains the file structure of an Android project when it is created. In Eclipse IDE, the file structure is created automatically. The **HelloWorld** project, when it is created, will consist of some main folders. In the Eclipse IDE, these can be viewed using the **Package Explorer** pane on the left hand side of the Eclipse screen, as shown in figure 2.42.



Figure 2.42: HelloWorld Project Folders

- **src** – this folder consists of the **.java** and **.aidl** source files of the **HelloWorld** project. It contains all the code required by the **HelloWorld app**. It consists of the stub **MainActivity.java** file by default.
- **gen** – this folder consists of both compiler generated Java files and other source code files required by the project. It is important not to directly edit the files in this folder. For instance, the folder consists of the **R.java** file which has references to resources present in the application. It acts as an index to all the resources in the project and the **.gen** files always need to be writable.
- The **Android 4.2** folder contains all the **.class** libraries required by the application.
- **assets** – this folder is used to save raw asset files that contain assets required by the application to function properly. The folder is originally empty and asset files get added when assets are added for the application. The Asset Manager helps to read and organize asset files similar to a regular file system. It consists of HTML files, text files, databases, and so on.
- **bin** – this folder consists of several compiled resources and the final **.apk** file.
- **res** – this folder consists of all the applications resources, such as, values, layouts, and images. In other words, it consists of all the resources used in the application. Resources refer to files that are not part of the code files but are used by the application code. Resources are worked into the application during build time. The **res/layout/** folder consists of XML files that are compiled as screen layouts, or parts of a screen. The **res/color** consists of XML file that describe colors. The **res/drawable** folder consists of bitmap files, image files, and other XML files for

describing the drawable objects or shapes. The **res/values** folder consists of XML files that are compiled into different type of resources.

- **AndroidManifest.xml** – this is the most important file for each Android application. It is stored in the root folder of the application. It defines the components of the application, such as, services and activities. Using this file, a developer can also define Intent filters, content providers, and broadcast receivers. The file contains the name of the application's Java package, which acts as a unique identifier for the application. It contains the permissions for the application, and also lists the libraries for the application. It also provides information about the version code, version name, minSdkVersion, and targetSdkVersion as shown in figure 2.43.

```
1 <?xml version="1.0" encoding="utf-8"?>
2<manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.example.helloworld"
4     android:versionCode="1"
5     android:versionName="1.0" >
6
7     <uses-sdk
8         android:minSdkVersion="8"
9         android:targetSdkVersion="17" />
10
11    <application
12        android:allowBackup="true"
13        android:icon="@drawable/ic_launcher"
14        android:label="@string/app_name"
15        android:theme="@style/AppTheme" >
16        <activity
17            android:name=".MainActivity"
18            android:label="@string/app_name" >
19            <intent-filter>
20                <action android:name="android.intent.action.MAIN" />
21
22                <category android:name="android.intent.category.LAUNCHER" />
23            </intent-filter>
24        </activity>
25    </application>
26
27 </manifest>
```

Figure 2.43 AndroidManifest.xml

- **default.properties** – this file sets the default properties for the Android project, and consists of settings for the project.

→ activity_main.xml File

1. Expand **res** → **layout** folder. It will show the **activity_main.xml** in **Graphical Layout** screen. Click the **activity_main.xml** tab at the bottom to view the code as shown in figure 2.44.

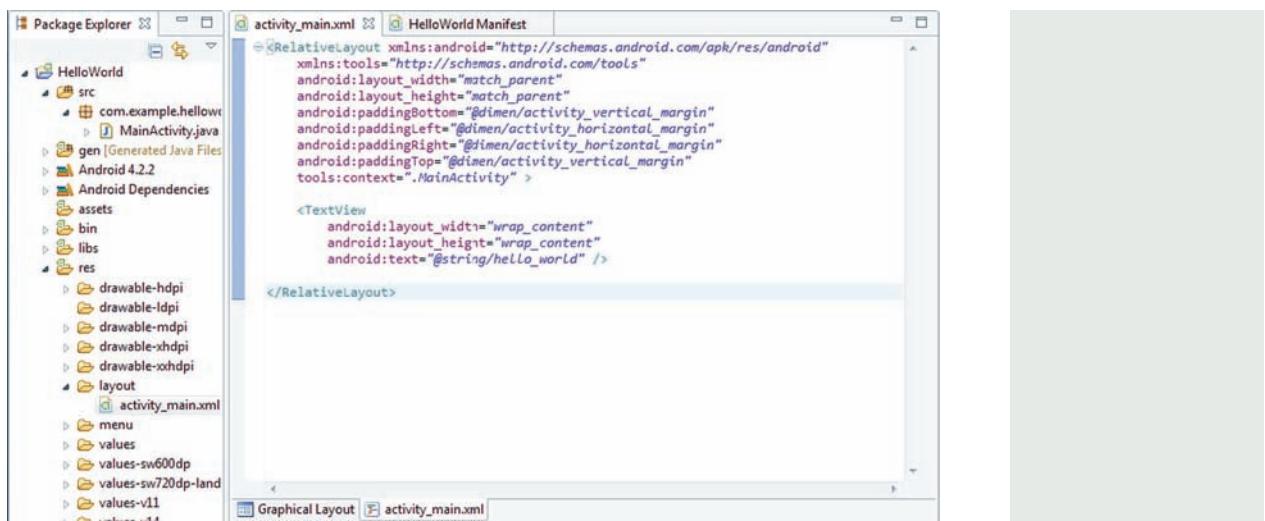


Figure 2.44: activity_main.xml Tab

When the developer creates the new **HelloWorld** project, android creates by default the **activity_main.xml** file. This file is stored in the **res/layout/** folder and contains the UI code. **activity_main.xml** is an XML layout file and is a tree of XML elements. Each node of the tree represents a View class which is used to design a UI layout. The different elements it contains and their description are as follows:

"<RelativeLayout" - specifies the type of layout used for the application. In this case RelativeLayout has been used. There are many other layouts, such as, linear layout, table layout, grid view, Web view, and so on. In the relative layout, the controls are relative to each other, or to the parent elements.

`android:layout_width="match_parent"` - indicates that the width should be of same size as the parent node. In this case the width will be of same size as screen size. This is the same for the height as well. It is used for specifying the height that will be used by the View.

"<TextView ..." creates a view component that prints the text given in the `android:text` attribute. In other words, it specifies the text that will be displayed by the TextView. Here, we are reading the "HelloWorld" text from another string value. The string value is defined in **strings.xml** file which is present in the **res/value** folder.

→ .java File

1. Double-click the **MainActivity.java** file in the **/src** folder to view the code as shown in figure 2.45.

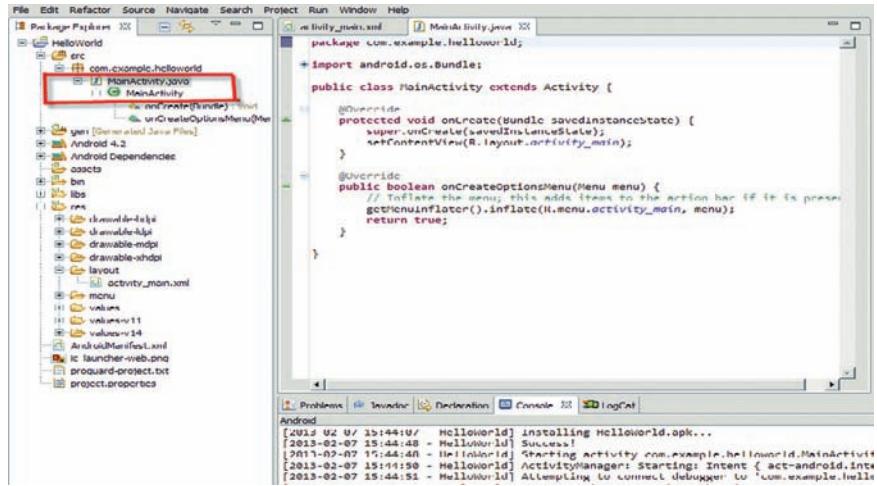


Figure 2.45 MainActivity.java – Default Code

MainActivity is a class which extends the `Activity` class. In the `HelloWorld` project, **MainActivity** is the starting activity. Here the `onCreate()` method which is part of activity life cycle has been overridden. The `setContentView()` method is defined in the `Activity` class through which the `activity_main.xml` for the UI is mapped. In other words, it accepts the reference of the layout resource which is identified as `R.layout.activity_main`. It actually refers to the layout defined in the `/res/layout/activity_main.xml` file. In android an `R.java` file gets auto generated for the changes in the resources. So there will be a static entry in `R.java` for `activity_main.xml`. This is the reason why `activity_main.xml` is accessible using `R.layout.activity_main`.

2.4 Android Application

Using the Android application layer, the developer can create a wide variety of applications. The Android components are reusable and can be used effectively to build applications.

2.4.1 Android Application Fundamentals

The Android application is written in Java programming language. The distinct feature of the Android application is its ability to live in its own ‘security sandbox’. The reasons for this distinct feature are as follows:

- ➔ Every application has a different user (a different Linux User ID assigned by the system).
- ➔ The User ID is recognized by the system and permission is granted to the application files so that only valid users can access them.

- The presence of a Virtual Machine (VM) provides scope for an application to function independently.

This understanding leads to the concept of ‘Principle of Least Privilege’. The ‘Principle of Least Privilege’ states that every application is dependent on components. Therefore, an application can access only those dependent components which it requires to perform the task. In this manner, a secure environment is maintained.

Exceptions to the ‘Principle of Least Privilege’ (shared access) are as follows:

- Two applications can function using the same Linux User ID and VM, and thus can access each other’s files.
- At the time of installation, permission can be granted to the application to access data beyond its boundary.

This leads to the core framework components that define the application, the manifest file where the components are declared along with the device features required by the application, and finally the resources that are separate from the application code and allows the application to optimize its behavior for different device configurations.

2.4.2 *Android Application Components*

Application components refer to the points that the system can use to enter an application. Each component also determines the way in which the overall application performs. Each component can be independently activated, and affects the behavior of the application. All components must be identified in the Android Manifest file.

The four types of application components are activities, services, content providers, and broadcast receivers. Three of the application components are activated by Intents. Intent is a message used for activating components and are created using Intent Objects, and can be explicit or implicit. Intents define the action that needs to be performed for activities and services. They define the announcement to be broadcasted for broadcast receivers. Intents do not activate content providers, which are activated by requests from a ContentResolver.

Android application components include:

→ Activities

An activity represents the presentation layer. In other words, it is a screen containing the User Interface (UI). It allows the user to interact with the application. An application can have one or more activities. These activities are loosely coupled. Every screen in an application is an activity.

For example, a Phonebook application can have different activities, such as, a Contacts list, Call log, Phone directory, SIM directory, and so on. The activities

together form the Phonebook application, but are distinct from each other. Other applications, such as, the Messaging application could start the Phonebook application as well.

There are several activities that are tied together within each application. However, every activity is independent of the other. A specific activity in an application is designated as the main activity, which is typically viewed by the user when the application is first launched. Every activity in the application can initiate another activity. For every new activity that is initiated, the previous activity stops and waits in the background. The changes in activity states are managed by the activity lifecycle.

It is similar to a form. To create the UI screen, the developer must create a Java class that extends the `Activity` class. Within this class the developer has to define the UI and implementation of the functionality. This class must implement the callback methods such as `onCreate()`. These callback methods are invoked by the system, when the transition of the `Activity` takes place between different life cycle states.

UI of the `Activity` class is implemented using `Views`. `Views` are the UI controls that display data and allow user interaction. The `setContentView()` method is used within the `onCreate()` method to assign an UI to an `Activity`.

- **Activity Lifecycle**

From the time an activity is initiated, it goes through several stages. The lifecycle of the activity changes depending on other related activities that are performed. Essentially when the activity is first initiated, the system calls certain lifecycle processes for the activity. The lifecycle processes for the original activity change when the user initiates an action that triggers another activity or accesses another application. The lifecycle of an activity is managed by implementing the callback methods.

In the course of an activity's lifecycle, there are activity states. An `Activity` goes through four transition states during its lifetime that are as follows:

- **Active State:** When an `Activity` is present at the top of the stack, it is the currently visible and focused `Activity` and all the user inputs are provided to it. In order to keep this `Activity` active, Android provides it with all the required resources and also terminates the previous activities if required. When another `Activity` becomes active, this `Activity` will be pushed to the paused state.
- **Paused State:** An `Activity` in this state is visible, but another `Activity` will have the focus and is present in the foreground. An `Activity` in the paused state is treated in the same way as it was treated when it was in the active state. The only difference is that it will not receive any user input. In serious cases, Android terminates a paused `Activity` to ensure availability of resources to the

current Activity. An Activity is stopped when it becomes totally obscure.

- **Stopped State:** In this state, the Activity is not visible. It is however, present in the memory with all the state information. All such activities are now ready for termination when the system requires memory. When an Activity is stopped, its data and UI information needs to be saved. An Activity becomes inactive when it is closed or exited.
- **Inactive State:** When the Activity is no longer in the memory, it is said to be in the inactive state. An Activity goes to the inactive state when it is terminated. All such activities need to be restarted before they are used again.

As and when a new activity instance is created, callback methods move the activity state one step forward. When the user is in the process of completing or leaving the activity, callback methods move the activity state one step down. In some cases callback methods move the activity state part way down when it is to be moved to the top. There are several methods that notify the activity of the different states. Some of the callback methods include `onCreate()`, `onRestart()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, and `onDestroy()` as shown in figure 2.46.

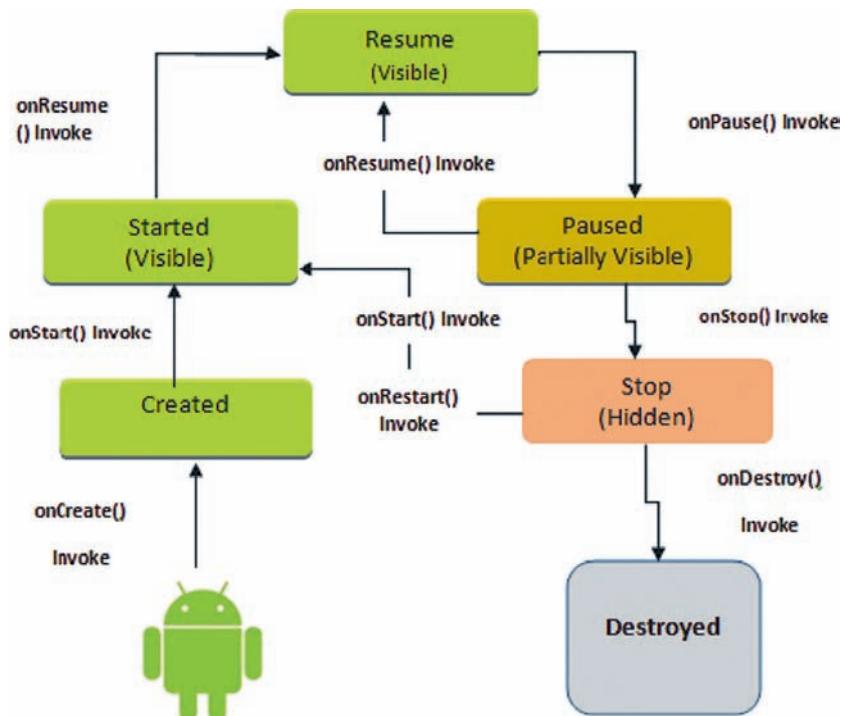


Figure 2.46: Activity Life Cycle

Table 2.1 lists the callback method of the Activity class.

Callback	Description
onCreate()	This callback method that is invoked when the activity is first created. The developer should set up static tasks, such as, creating views and binding data. This method is always followed by the onStart() method.
onStart()	This callback method is invoked when the user is beginning to view the activity. The method is followed by onResume() when the activity is in the foreground, and by onStop() if the activity is in the background.
onResume()	This callback method is invoked when the user begins to interact with the activity.
onPause()	This callback method is invoked when the activity is receding to the background but has not been killed.
onStop()	This callback method is invoked when the activity cannot be viewed by the user anymore. It could be followed by onRestart() or onDestroy method based on user activity.
onDestroy()	This callback method is invoked when the activity is about to be destroyed. The method is called when the activity is about to end, or when the activity is being terminated due to lack of memory or space.
onRestart()	This callback method is invoked once the activity has stopped, and before the activity is started again. This method is also always followed by onStart() method.

Table 2.1 Activity-Lifecycle Callback Methods

These methods define the entire lifecycle of an activity. Once the developer implements this method, he/she can monitor three nested loop in the activity life cycle. These are as follows:

- **The Full Lifetime:** This lifetime occurs between onCreate() to onDestroy() method invocation. In this lifetime, the activity normally performs global activity and releasing of resources. For example, it can create a thread in the onCreate() method and stop the thread in the onDestroy() method.
- **Visible Lifetime:** This lifetime occurs between onStart() and onStop() callback methods. The developer maintains the resources between these methods which are required to be displayed to the user.
- **Active Lifetime:** This lifetime occurs between the onResume() and onPause() method invocation. In this lifetime, the activity is in the foreground and has the user input focus.

To see the working of the lifecycle methods, perform the following steps:

1. Open the **HelloWorld** project folder in Eclipse IDE created earlier.
2. Click the **src** folder and double-click the **MainActivity.java** file to view the code as shown in figure 2.47.

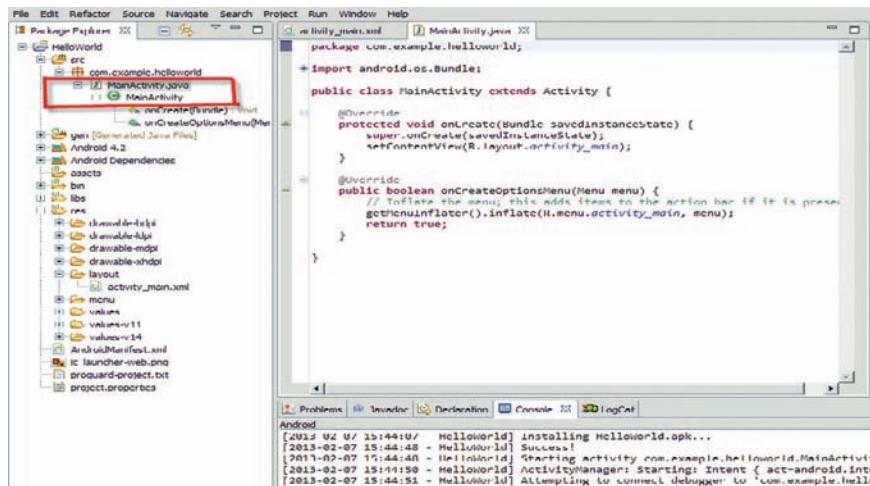


Figure 2.47: MainActivity.java

3. Modify the code as shown in code snippet 1.

Code Snippet 1:

```
...
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toast.makeText(getApplicationContext(), "oncreate() callback",
                Toast.LENGTH_SHORT).show();
    }
    /** Called when the activity is about to become visible. */
    @Override
    protected void onStart() {
        super.onStart();
    }
}
```

```
        Toast.makeText(getApplicationContext(), "The onStart() callback", Toast.LENGTH_SHORT).show();

    }

    /** Called when the activity has become visible. */

    @Override

    protected void onResume() {

        super.onResume();

        Toast.makeText(getApplicationContext(), "The onResume() callback", Toast.LENGTH_SHORT).show();

    }

    /** Called when another activity is taking focus. */

    @Override

    protected void onPause() {

        super.onPause();

        Toast.makeText(getApplicationContext(), "The onPause() callback", Toast.LENGTH_SHORT).show();

    }

    /** Called when the activity is no longer visible. */

    @Override

    protected void onStop() {

        super.onStop();

        Toast.makeText(getApplicationContext(), "The onStop() callback", Toast.LENGTH_SHORT).show();

    }

    /** Called just before the activity is destroyed. */

    @Override

    public void onDestroy() {

        super.onDestroy();

        Toast.makeText(getApplicationContext(), "The onDestroy() callback", Toast.LENGTH_SHORT).show();

    }

}
```

```

@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);
returntrue;
}
}

```

Note - In the code the `Toast` class is used to display a message to the user. It will be discussed in detail later.

Figure 2.48 displays the output of the modified code.

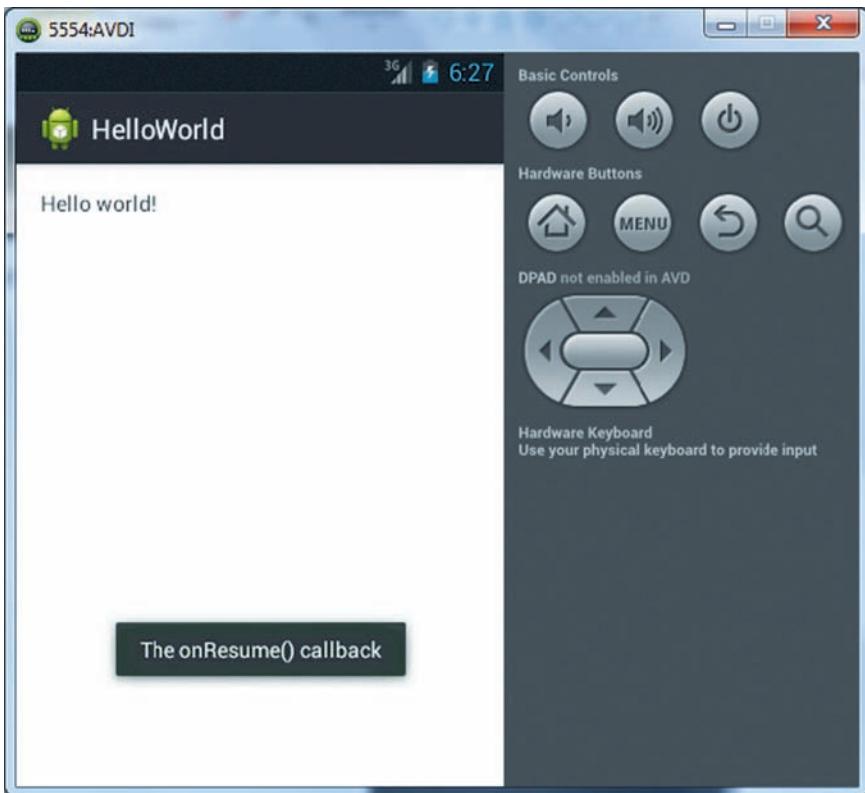


Figure 2.48: Lifecycle CallBack Methods - Output

→ Services

A service as a component is not displayed on the UI. In other words, it does not help in the design of the UI. It is a background component that processes lengthy operations, or is connected to processes that run remotely. Consider an example of music being played in the background while the user is surfing the net or checking emails. Music in the background is considered as a service. This will not affect user interaction with an activity.

Services include gathering data without interrupting another activity, and running a process while the user is involved in another activity task. Other activities can start a service, run it, and get attached to the service as well.

Each service has two forms – Started and Bound.

A **Started** service has the following features:

- Starts when the `startService()` method is invoked by an application component, such as an Activity.
- Runs indefinitely in the background irrespective of the state of the invoking component.
- Stops once the operation is complete.

A **Bound** service has the following features:

- Starts when the `bindService()` method is invoked by the application component to bind to it.
- It interacts with the application components, such as by sending requests, obtaining results, and performing Inter Process Communication (IPC).
- Runs as long as the application component is bound. Several components can bind to a single service.

Note - A service also needs to be identified in the **AndroidManifest.xml** file.

A new service class must extend the `Service` class. Some of the callback methods must be overridden in the implementation class to handle key aspects of the service lifecycle and provide a mechanism for the components to bind to the service.

Service methods include `Context.startService()`, `onStartCommand`, `Context.stopService`, and `Context.bindService()`.

Table 2.2 lists the methods of the `Service` class.

Methods	Description
<code>Context.startService ()</code>	This method retrieves a service by creating or calling the <code>onCreate ()</code> method when required. It then calls the <code>onStartCommand() </code> method using the arguments provided by the client.
<code>onStartCommand()</code>	This method is overridden for the following reasons: <ul style="list-style-type: none"> ➔ To ensure that the service is executed infinitely. ➔ When an application component, such as an Activity invokes a service using the <code>startService()</code> method. ➔ Developer should stop the service by invoking the <code>stopSelf()</code> or <code>stopService()</code> method.

Methods	Description
<code>onCreate()</code>	This method is invoked to perform the setup procedures only once when the service is first created.
<code>Context.stopService()</code> or <code>stopSelf()</code>	This method is used to stop the service. The method is also used to stop multiple instances of a service. The <code>stopSelf (int)</code> method can be used to prevent the service from stopping until all the intents that were started are processed.
<code>Context.bindService()</code>	This method is used to keep the service connected until the connection remains. An <code>IBinder</code> object is received by the client when the service returns from the <code>onBind()</code> method. This will allow the client to make calls back to the client.
<code>onBind()</code>	This method is invoked when another component wants to bind with the Service class.

Table 2.2: Service Class - Methods

Figure 2.49 displays the working of the service lifecycle methods.

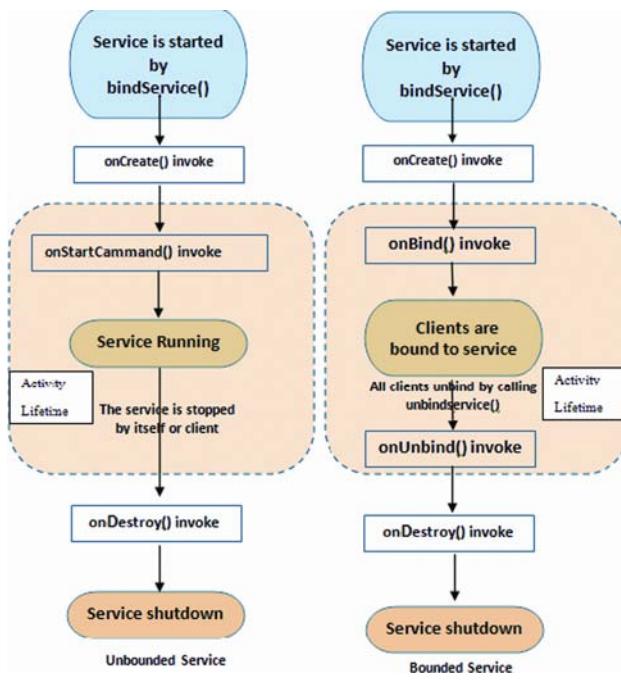


Figure 2.49: Service Life Cycle

Now, let us work with an example of music player where the music is played and stopped with the help of an android service. The steps to be performed are as follows:

1. Open the **HelloWorld** Project in Eclipse which developer created earlier.
2. Navigate to **src → package**.
3. To create a new class right-click the package folder to display the context menu.
4. Select **New → Class** as shown in figure 2.50.

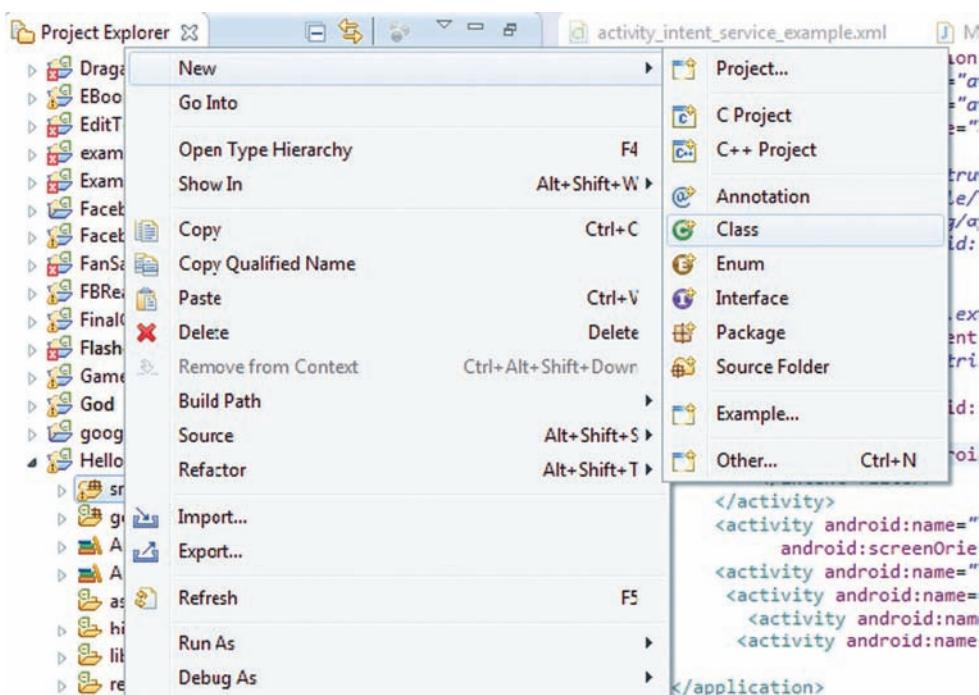


Figure 2.50: Create Class

The **New Java Class** dialog box is displayed as shown in figure 2.51.

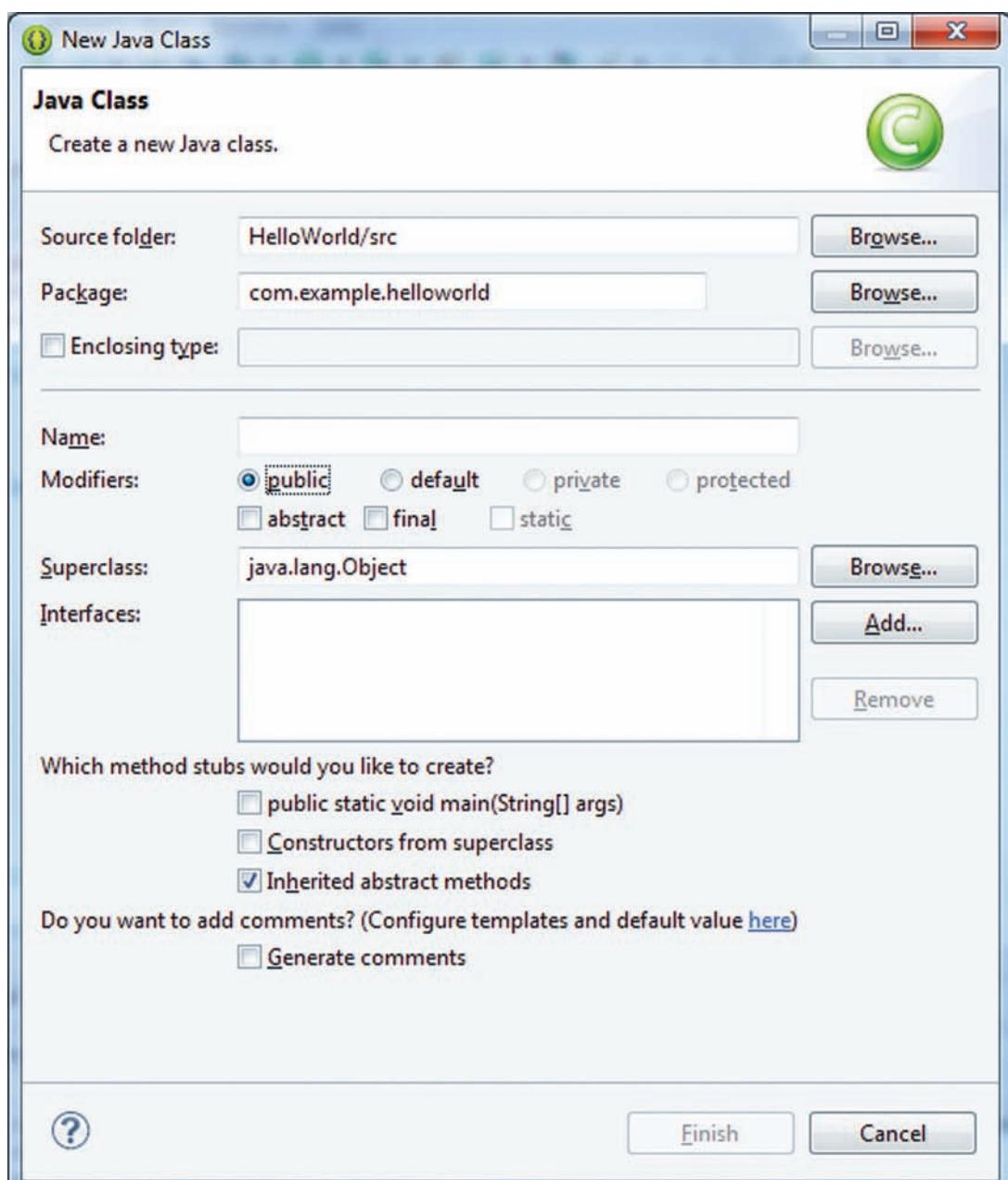
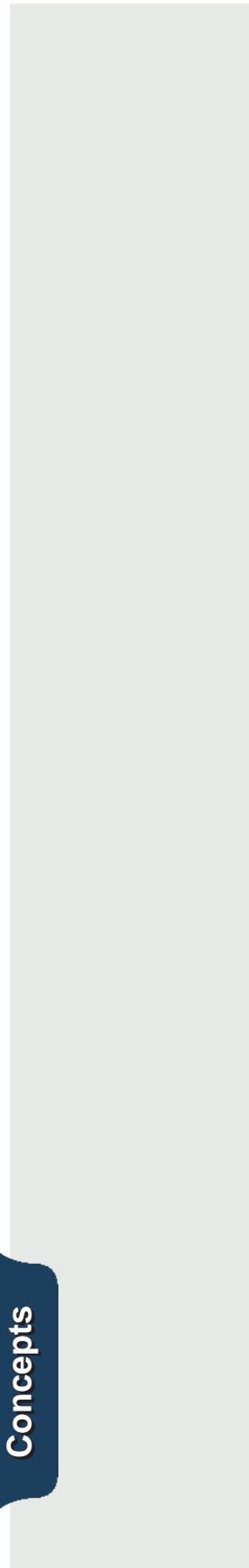


Figure 2.51: New Java Class

5. Type **ServiceRun** in the **Name** box.
6. Click **Finish**.
7. Modify the code as shown in code snippet 2.



Code Snippet 2:

```
package com.example.helloworld;

import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;

public class ServiceRun extends Service {
    MediaPlayer mp_object;

    @Override
    public IBinder onBind(Intent arg0) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public void onCreate() {
        // TODO Auto-generated method stub
        super.onCreate();
        mp_object = MediaPlayer.create(getApplicationContext(), R.raw.sleepaway);
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        mp_object.start();
        return 0;
    }

    @Override
    public void onDestroy() {
        mp_object.release();
        super.onDestroy();
    }
}
```

8. Create a new folder named “**raw**” inside the **/res** directory by right-clicking the **/raw folder** and selecting **New → Folder**.
9. Copy a song inside it as shown in figure 2.52. For example, it is **flysong.mp3**.

Note - The song name should be in lowercase and without space.

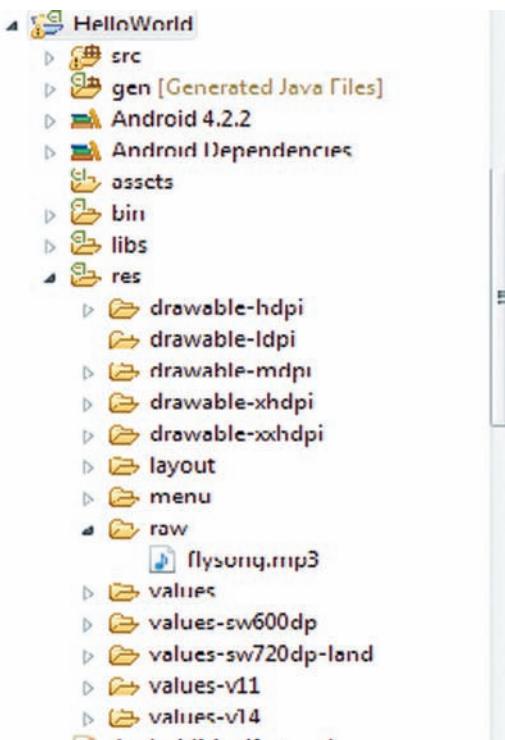


Figure 2.52: raw Folder

10. Navigate to the **res → layout** folder.

11. Double-click **activity_main.xml** file to display the file in **Graphical Layout** mode and add two buttons by dragging the button as shown in figure 2.53.

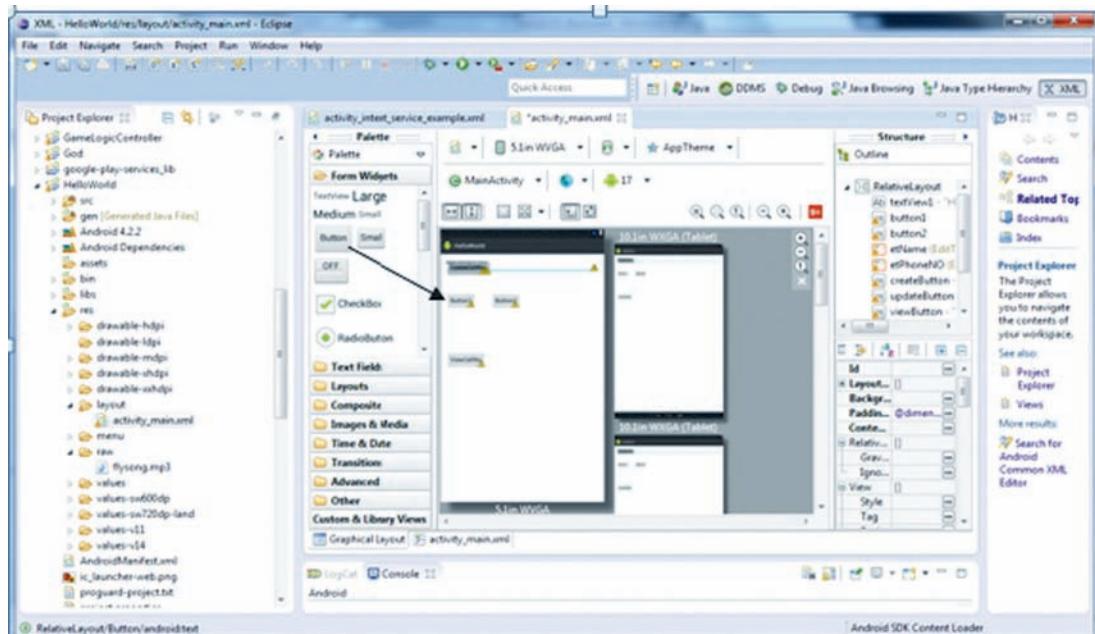


Figure 2.53: Create a Button

12. Click **activity_main.xml** tab at the bottom to open the file in editable screen.
13. Modify the code as shown in code snippet 3.

Code Snippet 3:

```
<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"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world"/>
```

```

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView1"
    android:layout_below="@+id/textView1"
    android:layout_marginTop="80dp"
    android:text="Start_Music" />

<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/button1"
    android:layout_alignBottom="@+id/button1"
    android:layout_marginLeft="45dp"
    android:layout_toRightOf="@+id/button1"
    android:text="Stop_Music" />
</RelativeLayout>

```

14. Navigate to **res → layout** folder.
15. Double-click **activity_main.xml** file to display the file in **Graphical Layout** screen as shown in figure 2.54.

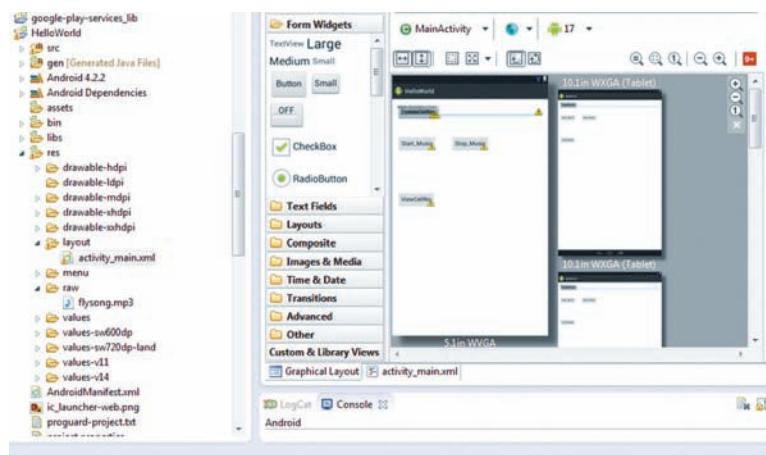


Figure 2.54: activity_main.xml – Graphical Layout

16. Open and add the <service> tag in **AndroidManifest.xml** file as shown in code snippet 4.

Code Snippet 4:

```
<?xmlversion="1.0"encoding="utf-8"?>

<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="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">

        <activity
            android:name="com.example.helloworld.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>

        <service android:name=".ServiceRun">
        </service>
    </application>

</manifest>
```

17. Set the `onClickListeners` for both the buttons inside the **MainActivity.java** by adding the code as shown in code snippet 5.

Code Snippet 5:

```
Button play_song, stop_song;
...
play_song = (Button) findViewById(R.id.button1);
stop_song = (Button) findViewById(R.id.button2);
play_song.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        /* way to call the service class */
        Intent service = new Intent(MainActivity.this, ServiceRun.class);
        startService(service);
    }
});
stop_song.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent name = new Intent(MainActivity.this, ServiceRun.class);
        stopService(name);
    }
});
```

18. Code snippet 6 displays the code in **MainActivity.java** file.

Code Snippet 6:

```
package com.example.helloworld;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
```

```
import android.widget.Button;
import android.widget.Toast;
public class MainActivity extends Activity {
    Button play_song, stop_song;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toast.makeText(getApplicationContext(), "oncreate() callback",
            Toast.LENGTH_SHORT).show();
        play_song = (Button) findViewById(R.id.button1);
        stop_song = (Button) findViewById(R.id.button2);
        play_song.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                /* way to call the service class */
                Intent service = new Intent(MainActivity.this, ServiceRun.class);
                startService(service);
            }
        });
        stop_song.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent name = new Intent(MainActivity.this, ServiceRun.class);
                stopService(name);
            }
        });
    }
}
```

```
/** Called when the activity is about to become visible. */
@Override
protected void onStart() {
    super.onStart();
    Toast.makeText(getApplicationContext(), "The
onStart() callback",
        Toast.LENGTH_SHORT).show();
}

/** Called when the activity has become visible. */
@Override
protected void onResume() {
    super.onResume();
    Toast.makeText(getApplicationContext(), "The
onResume() callback",
        Toast.LENGTH_SHORT).show();
}

/** Called when another activity is taking focus. */
@Override
protected void onPause() {
    super.onPause();
    Toast.makeText(getApplicationContext(), "The
onPause() callback",
        Toast.LENGTH_SHORT).show();
}

/** Called when the activity is no longer visible. */
@Override
protected void onStop() {
    super.onStop();
    Toast.makeText(getApplicationContext(), "The
onStop() callback",
        Toast.LENGTH_SHORT).show();
}
```

```

    /** Called just before the activity is destroyed. */
    @Override
    public void onDestroy() {
        super.onDestroy();
        Toast.makeText(getApplicationContext(), "The
onDestroy() callback",
                Toast.LENGTH_SHORT).show();
    }
    @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;
    }
}

```

19. Build and execute the application by clicking **Run → Run**.

Once you run the project you will see the output as shown in figure 2.55. When you click **Start Music** and **Stop Music** you will start and stop the playing of the song respectively.

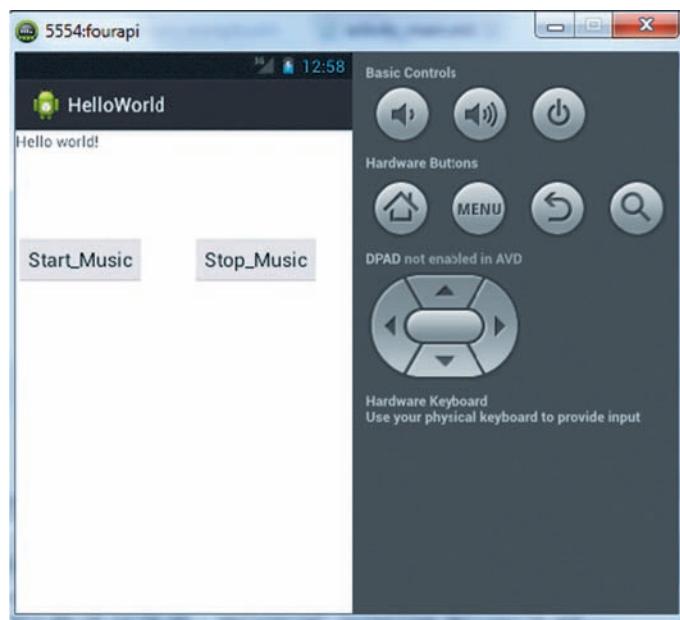


Figure 2.55: Start and Stop the Music

→ Content Providers

Content providers are used for managing shared data sets. The data can be stored in the SQLite Database, Web, or other easily accessible storage location. The content providers help other applications to query for data and make changes to the data for a specific application.

Content providers are those components that provide data to the applications. In other words, it is a shareable data store. For example, if the content provider allows access, an application can query personal information, for instance, information in a phonebook. The content provider also enables reading and writing data that is not shared. A content provider is implemented as a subclass of `ContentProvider`. `ContentResolver` object present in the client provides access to data in a content provider.

Content providers are simple in their existence as they use the standard methods of `insert()`, `update()`, `delete()`, and `query()` to work with data. Some examples, of native Android content providers are media store, settings provider, contacts provider, and so on.

Content providers are accessed using a URI model. The main feature of the content provider is its flexibility to extend so that the native android databases can be modified. Access to data from content providers is managed by the `ContentResolver` class.

An object of the `ContentResolver` class is obtained using the `getContentResolver()` method.

The `ContentResolver` class has methods to query and modify content providers. The query string used for querying a content provider is in the form of a URI. Content providers have two forms of URI. The first form will access all data and the second form will access a single row. The expression of the two URI forms is:

```
<standard _ prefix>://<authority>/<data _ path>/<id>
```

Parts of the URI are as follows:

- `content://` is the standard prefix for all content providers.
- The name of the content provider will be specified by the authority. Contacts for the built-in contacts content provider can be stated as an example.
- The type of data requested will be specified by the data path. For example, if all the contacts are obtained from the contacts content provider, the URI would be `content://contacts/people`.
- The specific record that is requested will be specified by the id. If contact number 5 is being looked for, then the URI would be: `content://contacts/people/5`.

The methods that need to be used for content providers are as follows:

- `onCreate ()` – Is used to initialize the content provider for the first time.
- `query (Uri, String [], String, String [], String)` – Provides data to the caller. In other words, it is used to retrieve the records.
- `insert (Uri, ContentValues)` – Is used to insert data into the content provider.
- `update (Uri, ContentValues, String, String [])` – Is used to update data in the content provider.
- `delete (Uri, ContentValues, String, String [])` – Is used to delete data from the content provider.
- `getType (Uri)` – Is used to provide the MIME data type in the content provider.

→ Broadcast Receivers

The broadcast receiver, as the name suggests, is a component that receives or responds to announcements broadcast by the system. These could be announcements for low battery, locked screen, data downloaded, and so on. Broadcast receivers also inform other applications of data updates. Broadcast receivers do not display a UI as well. The broadcast receiver acts as an entry point for other components and is simply an announcement component. Most of the broadcasts are initiated by the system but an application can also initiate a broadcast.

Broadcasts are of two types namely, Normal and Ordered. Normal broadcasts are received and executed in an asynchronous order. Ordered broadcasts are received one at a time and are synchronous. With normal broadcasts, however, at times, the system can deliver one broadcast at a time.

The Broadcast Receiver class' lifecycle includes security, receiver lifecycle, and process lifecycle. Developer can dynamically register an instance of this class with `Context.registerReceiver()` or statically with the help of `<receiver>` tag in **AndroidManifest.xml** file.

Receivers are used with Context APIs and interact with other applications. This makes them vulnerable to abuse by other applications. Some considerations with regard to security are as follows:

- When using `registerReceiver (BroadcastReceiver, IntentFilter)`, the registered receiver is open to broadcasts from any application. Permissions need to be set for controlling the broadcasts that are sent. In the latest version you can specify the package name of the application that will receive the broadcast.
- Intent names should be written in a namespace owned by the developer to avoid applications conflict.

- When a receiver is published in the manifest file and intent filters are specified for the file, enter `android:exported="false"` to ensure security.

Receiver Lifecycle – This is important because a BroadcastReceiver is only active for the call for `onReceive()` method. Once the function is terminated the receiver becomes inactive. A broadcast receiver extend the BroadcastReceiver abstract class which means that the developer have to implement the base class method `onReceive()`. Whenever an event occur Android invokes the `onReceive()` method on the registered broadcast receiver.

A process lifecycle keeps the BroadcastReceiver running until `onReceive()` method is active.

Methods for BroadcastReceiver are as follows:

- `abortBroadcast()` – This method call to the broadcast receiver indicates that the receiver should abort a current broadcast and work for broadcasts sent via `Context.sendOrderedBroadcast`.
- `clearAbortBroadcast()` – This method call to the broadcast receiver indicates that the current broadcast should be terminated.
- `getAbortBroadcast()` – This method returns boolean value indicating whether the receiver should abort the current broadcast.
- `getDebugUnregister()` – This method returns the last value from `setDebugUnregister(boolean)` method.
- `getResultCode()` – This method provides the result code that is set by the previous receiver.
- `getResultData()` – This method receives the current data, that is set by the previous owner.

Note - Avoid using long running task in the broadcast receiver. For performing any long task we can start a service within a receiver.

2.5 Communication Components

The mode of communication between the components includes Intents and Intent filters.

→ Intent

- An Intent activates the three components - activities, services, and broadcast receivers. When several actions from different components are requested at runtime for an application, the Intent brings the components together.
- An Intent object creates an Intent. In other words, it is a gamut of information used for activating the components. The Intent object identifies the message that should activate a component or component type. There are two types of Intents – implicit and explicit.

- Explicit Intents are Intents that identify a component by the component's name. The Java class name is used as the identifier. Implicit Intents do not identify a component by the name. Implicit Intents are most commonly used for identifying components in other applications.
- In case of activities and services, an Intent determines the action that must be performed. For example, sending a message, or viewing an e-mail.
- An Intent can also communicate a request for data. For instance, an Intent can communicate a request to display a file. For activities that involve a result, Intents can communicate the request for the data and the data result.
- In case of broadcast receivers, the Intent merely communicates the announcement, for instance, "data downloaded".
- Intents do not activate content providers. Content providers are activated by content resolvers.

→ Intent Filter

Activities, services, and broadcast receivers use Intent filters to inform the system about the Intents that they can manage.

Note - In an Intent Filter, two components are unaware of each other's existence. However, they work in partnership to deliver the expected result.

- Every filter provides information on a particular ability of the component, and the Intents that the component can receive.
- An explicit Intent does not require a filter as it directly communicates with the component. An implicit Intent can communicate with a component only if the filters allow it to.
- Although implicit Intents are filtered, the Intent filter does not provide any security.
- A component contains a different filter for each activity.
- Intent filters are executed as an object of `IntentFilter` class.

Some of the points to remember about Intent filters are as follows:

- A component can claim any number of intent filters.
- If there is more than one component claiming the same intent filter, the user can decide to select the appropriate component.
- Priorities can be set for intent filters to track the order of responses.

→ Pre-existing Components

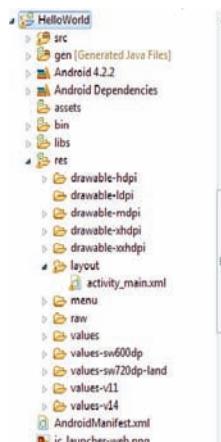
Pre-existing components are determined by the Android manifest file. The pre-existing components are vital for the execution of implicit intents. They include:

- Intent Objects – An Intent object consists of information of interest that is received by a component. It also contains information, such as category of the component. The Intent should be handled by the component name which is provided in the manifest file as a component name object. The component name is optional, and the Intent object can deliver the information to the designated class or to another target it identifies.
- Intent Matching – Intents are matched with Intent filters to identify a target component that needs to be activated. They are also matched to identify more information on the device components. Intent matching can be specified using “**android.intent.category.Launcher**” and “**android.intent.category.HOME**”.
- Intent Resolution – Intents are of two types, explicit and implicit. Explicit intents identify target components by their name. These are typically used for messages internal to an application. Implicit intents do not identify components by name and are used for messages in other applications. An activity or service is identified that can best perform the specified action and the intent is delivered accordingly. This is done by comparing the intent objects to intent filters. Filters display the abilities of components so that they can receive intents. Intent resolution basically functions around matching Intent against all the `<intent-filter>` descriptions in the installed application packages.

→ Android Manifest File

The manifest file is important because all the components required for an application need to be present in this file. Before an application component is initiated, the system looks for the component in the **AndroidManifest.xml** file.

Every Android project contains the **AndroidManifest.xml** file which contains the detailed configuration information as shown in figure 2.56. It can be considered as a metadata file and is present in the root directory of the project. It contains the details of the components that exist in an application.



```

5     android:versionName="1.0" >
6     <uses-sdk
7         android:minSdkVersion="8"
8         android:targetSdkVersion="17" />
9
10    <application
11        android:allowBackup="true"
12        android:icon="@drawable/ic_launcher"
13        android:label="@string/app_name"
14        android:theme="@style/AppTheme" >
15            <activity
16                android:name=".MainActivity"
17                android:label="@string/app_name" >
18                <intent-filter>
19                    <action android:name="android.intent.action.MAIN" />
20
21                    <category android:name="android.intent.category.LAUNCHER" />
22                </intent-filter>
23
24            </activity>
25            <service
26                android:name=".ServiceRun"
27            >
28        </service>
29
30    </application>
31
32 </manifest>
33

```

Figure 2.56: Android Manifest File

The first part of the code is displayed in figure 2.57.

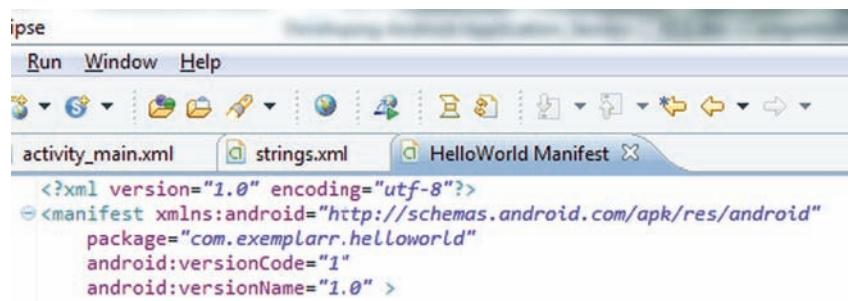


Figure 2.57: Android Version

This node specifies the following attributes:

- package – specifies the package name of the application.
- android:versionCode – specifies the internal version of the project.
- android:versionName – specifies the version number shown to the user.

Code snippet 7 shows the `<uses-sdk>` tag present within the `<manifest>` tag.

Code Snippet 7:

```

<uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />

```

`android:minSdkVersion` – specifies an integer value indicating the minimum Android API version number.

`android:targetSdkVersion`— specifies an integer value indicating the target Android API version number.

Code snippet 8 displays the `<application>` node tag.

Code Snippet 8:

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
```

The `<application>` tag is a container as it includes the tags for declaring Services, Activities, ContentProvider, and BroadcastReceiver present in the application. There is only one application node present in the **AndroidManifest.xml** file. Android application related configuration information such as app icon, name, theme, and so on are also declared and forms the metadata of information of the application.

Code snippet 9 displays the `<activity>` node tag.

Code Snippet 9:

```
<activity
    android:name="com.example.helloworld.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>
```

Every activity in android should be registered here. In the case of **HelloWorld** project, the **MainActivity** class is registered and also made as the launcher Activity class. In the code, the `android:name` attribute is used to specify the Activity class name. This displays the main launch Activity and any other screen/dialogs. A runtime exception is thrown when an Activity has not been defined but an attempt is made to start the Activity.

The `<activity>` tag also specifies which Intents will launch the Activity and is specified by their child tags `<intent-filter>`.

Other node tags that can be used in **AndroidManifest.xml** file are as follows:

- ➔ `<service>`: The tag is used for declaring each Service class used in the Android application development. The `<service>` tag supports the `<intent-filter>` child tags.

- ➔ <provider>: The tag is used for declaring content provider.
- ➔ <receiver>: The tag is used for registering a broadcast receiver.
- ➔ <uses-permission>: The developer uses the tag to set permissions for applications so that it can function correctly. In other words, this tag is used to grant access to a feature that is protected by permission. During installation, the user is presented with the option of granting or denying permission to access certain applications. Even execution of native applications, such as receiving of SMS or dialing, requires permission.
- ➔ <permission>: The tag helps in defining permissions in the manifest file. These permissions refer to restricting access to certain shared application components. The android:name attribute specifies the name of the permission that will be referred in the code and the android:protectionLevel characterizes the level of access the permission grants.

The values for android:protectionLevel are as follows:

- normal: signifies low risk and is the default value.
- dangerous: signifies high risk.
- signature: signifies that the requesting application and the application declaring permission should be signed with the same certificate.
- signatureOrSystem: signifies that the requesting application should be signed with the same certificate as that which is present in the system image.

The android:label and the android:description explains the risk involved in granting this level of permission.

- ➔ <instrumentation>: The <instrumentation> tag is used for monitoring the application and its response to the system resources. The monitoring is done by conducting tests on activities and services at runtime.

Advantages of the Manifest File

The advantages of using manifest file are as follows:

- Assists in granting permission to specific users. An example could be permission to access the Internet.
- Helps in declaring hardware and software requirement for a particular application. For example, Bluetooth, multi-touch screen, and so forth.
- Declares application components.
- Helps in specifying the minimum API level required by the application.

- Helps in linking the application to the API libraries. For example, contains link to the Google Maps library.
- The manifest file contains a root <manifest> tag and a <package> attribute. This <package> attribute is associated to the project's package. The versionCode and versionName attributes are used to specify the application version number and the version number that will be displayed to the user respectively.

Apart from stating the components of the application, the manifest file has a number of other uses. Some of these include:

- If an application requires user permissions, the manifest file defines the permissions.
- Depending on the APIs the application uses, the manifest file determines the minimum level of API the application requires.
- Identifies the hardware features that the application makes use of, or requires. It also determines software features needed by the application.
- Apart from the libraries in the Android architecture, the application needs to be connected to other application libraries. The manifest file determines the libraries as well.

2.6 Check Your Progress

1. Which of the following option specifies the first step for creating an Android application?

(A)	Creating a Debug Configuration	(C)	Creating a Project
(B)	Creating an Android Virtual Device (AVD)	(D)	Creating a Run Configuration

2. An Android project consists of _____ for the application.

(A)	Android Software Development Kit (SDK)	(C)	Android SDK Tools
(B)	Emulator	(D)	Source code

3. Which of the following option is true for Android Virtual Device (AVD)?

(A)	A mobile device	(C)	A configuration for a device
(B)	A User Interface (UI)	(D)	A Web page for testing applications

4. Which of the following options are required for the Run configuration?

(A)	Android project to be run	(C)	Android device to connect to
(B)	Android Manifest file to connect to	(D)	Android application to be run

5. Which of the following Android Application components display the UI?

(A)	Activities	(C)	Content Providers
(B)	Services	(D)	Broadcast Receivers

6. Identify the functions of the Android Manifest file.

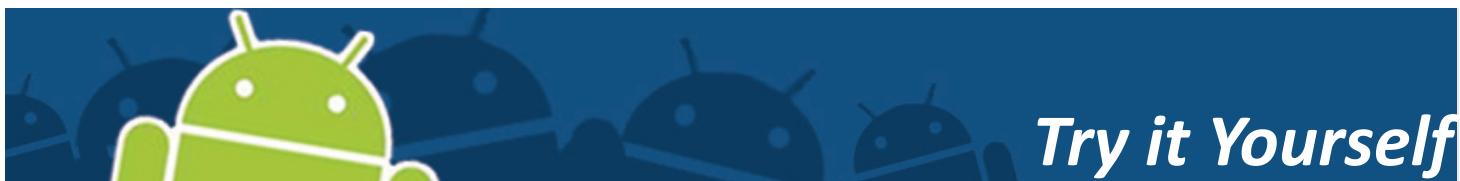
(A)	Running an application	(C)	Defining user permissions
(B)	Identifying hardware and software features	(D)	Debugging an application

2.6.1 Answers

1.	C
2.	D
3.	C
4.	A, C
5.	A, D
6.	B, D



- Android project consists of all the files and resources required to build the project into a .apk file for installation.
- The 'Principle of Least Privilege' states that every application is dependent on components. The four main components are Activities, Services, Broadcast Receivers, and Content Providers.
- The Android Virtual Device (AVD) is a configuration that enables sampling various Android devices using the emulator.
- For the emulator to run the application correctly, a Run configuration is required. The Launch configuration identifies the project to be executed, the main activity to be started, and the emulator or device to run the application. It also enables debugging the application.
- The Android framework includes components such as Activities, Services, Content providers, and Broadcast receivers which form the building blocks of an Android application.
- An Activity is a component that helps the developer to specify the UI of the application. Activity state is determined by the position of the Activity in the Activity stack.
- The difference between an Activity and a Service is the absence of a UI in a Service class as services are executed in the background.
- Content providers are the data providers that control the access to structured data. These connect the data on one process to the code of another process.
- Communication components include Intents and Intent filters. Intents are messages transferred between components, such as Services, Activities, Content Providers, and Broadcast Receivers.
- Explicit intent explicitly specifies the name of the component which should be invoked by the Android system. Implicit intents refer to those intents that do not target any particular component for assistance.
- The pre-existing components include Intent objects, intent matching, and intent resolution.
- Android Manifest file is present in the root directory and presents information about the application to the Android system.



Samantha is interested in learning new technologies that are being introduced in the market. Since Android programming is gaining popularity, Samantha has thought of learning Android programming. You have been assigned as the teacher for teaching her Android programming. She wants to develop an application which will perform the following tasks:

- Display “I am learning Android” string in the Emulator.
- Create a **Service** class with two buttons labeled as **Start Service** and **Stop Service**. When the user clicks the button a message should be displayed using a **Toast** such as “**Service Started**” and “**Service Stopped**” respectively.

“ Learning how to learn is
life's most important skill ”

JELLYBEAN
ICE CREAM SANDWICH

Session - 3

Android User Interface (UI)

Welcome to the Session, **Android User Interface (UI)**.

This session gives an overview of Android User Interface (UI). It describes the views, layouts, User Interface (UI) components, styles and themes, and the procedure for handling user events.

In this session, you will learn to:

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



3.1 Introduction

The UI is a space that enables interaction between the user and the machine. The basic purpose of UI is to enable the user to operate and control the machine. Feedback from the machine helps the user to manage operating decisions. UI is made up of both hardware and software components.

UI enables the following actions:

- ➔ Input: User manipulates the controls.
- ➔ Output: Machine responds to the user's actions.

Having understood the basic concept of UI, it is easy to appreciate how crucial UI is for PCs, mobile phones, smartphones, and other electronic gadgets. Without UI, it is impossible to interact with the gadget and use it for any purpose.

Touch screen, Graphic User Interface (GUI), and Text-based User Interface (TUI) are just a few examples of the different types of UIs that are available.

3.2 Android User Interface

User interface is what the user will see and interact with to perform some operations.

Android comes with many friendly UI elements and layouts which helps to build interactive applications.

3.2.1 Overview of UI

Android offers various tools for the developer to build the GUI for an application. It also enables the developer to build an interface for menus, dialogs, and notifications. The tools that Android offers for UI developers include pre-built components and controls for the UI. The developer must understand certain concepts to build the UI. They are as follows:

- ➔ Views
- ➔ Layouts
- ➔ UI Components

View objects are the primary UI units of an Android application. A View, being an UI object, forms the common platform of interaction between the user and the application. View is a built-in class used for building different UI components. Using the view class, the developer has to specify the screen area that the user needs to interact with. Views are used to represent elements, such as, lists, buttons, checkboxes, and so on. Android offers various built-in views. However, if these are not sufficient, the developer can create specific views for apps.

Another important concept that the developer needs to understand while building the UI for an app, is layouts. **Layouts act as containers for views.** The type of layout determines the location of the View object on the UI screen. Layouts consist of linear layout, relative layout, list view, and grid view.

UI components involve understanding of several concepts such as:

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

3.2.2 Views

All UI components in Android are built using `View` and `ViewGroup` objects. Views enable the developer to create the UI. Each item in the Android UI belongs to the `View` class. Views are also known as widgets and help to provide interactivity using buttons, text fields, and so on. There are a number of existing built-in views that Android provides.

To build a unique UI for an app, the developer needs to understand the `View` object, as well as, the objects belonging to `ViewGroup`. A set of several views make a `ViewGroup`. In other words, a **`ViewGroup` is a container that contains and organizes other child views.** Using the subclasses of `ViewGroup` such as `RelativeLayout`, `LinearLayout` and `View` objects such as `Button`, `TextView`, `ListView`, and so on, the developer can build layout of the UI. `ViewGroup` and `View` objects are arranged in a hierarchy for each

component, as shown in figure 3.1. Each ViewGroup can have other ViewGroup and View within. These are called child Views.

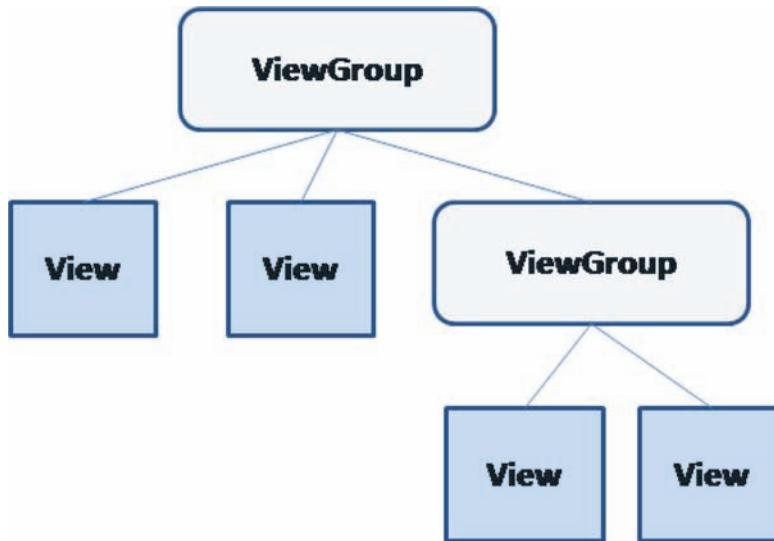


Figure 3.1: View Hierarchy

Upon creating a hierarchy, the developer can add criteria for interactivity. In other words, the developer can perform certain common operations. For instance, the developer can use set properties option to determine the text in TextView. The properties related to this function are different for various View subclasses. Some of the other types of operations that a developer can perform on views are setup listeners, set focus, and set visibility.

- **Set Focus:** When this criterion has been added, it controls the moving focus for views depending on the inputs from the user.
- **Setup Listeners:** This provides the option of setting notifications for listeners for certain changes in the View. For instance, when there is a change in the focus, the listener can be notified.
- **Set Visibility:** When this criterion has been added, the user can hide or show Views.

Other concepts related to Views include:

→ **Attributes:**

Every View and ViewGroup object possess their own range of XML attributes. Some attributes are specific to a View object - for example the TextView supports the textSize attribute. But these attributes are also inherited by any View objects that may extend this class. There are other attributes that are common to all View objects, because they are inherited from the root View class. One of these attributes

is ID which is common to all. The following lists some of the common attributes present in an object of the View class:

- ID
- Setting the Padding and Size of a View
- Setting View Location

→ ID

Each View object will have an integer ID associated with it, which uniquely identifies it as a resource within the tree. The attribute is used to provide reference to an application when it is compiled. The ID attribute will be assigned in the XML file as a string. This is an XML attribute common to all View objects and is defined by the View class.

The syntax used for an ID inside an XML is as follows:

Syntax:

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

where,

- '@' - symbol indicates that the XML parser should parse and expand the ID string and also recognize it as a resource.
- '+' - symbol identifies it as a new resource and adds it as a resource when required.

Note - When the developer provides a reference to an Android resource ID, the name of the Android package must replace the '+' symbol.

→ Setting the Padding and Size of a View

The developer can set the size and padding of a View. Setting the padding helps to measure the dimensions of the View. The padding can be provided for the left, right, top, and bottom of the View area in pixels. The `setPadding (int, int, int, int)` method can be used to set the padding for a View. The developer can use the different `getPaddingXXX()` method to obtain the current setting. XXX stands for left, top, and so on. For instance, the query `getPaddingLeft()` displays the current setting for the left part of the View area.

The width and height of the View area can be set to determine the View size. The developer can set the width and height using measured width and measured height options in the layout. For instance, specifying `measure (int, int)` in the layout determines the width and height of all Views in the layout. Using `layout (int, int, int, int)` specifies the measurement for the child views.

→ Setting View Location

A View is in the shape of a rectangle. The View's location is specified using left and top co-ordinates, and the dimensions of View is expressed as, width and height. The unit of measurement is in pixels. A View's location can be obtained using `getLeft()` and `getTop()` method which returns the X and Y co-ordinates respectively. The location of the view is returned which is relative to the parent view.

→ Layout Parameters

The attribute `layout_samplename` defines the layout parameters for a View object and these parameters conform to the ViewGroup in which the View resides. The nested class inside the ViewGroup class extends the `ViewGroup.LayoutParams`. This nested subclass contains property types for defining the size and position of child views. The ViewGroup class also includes many other parameters such as height, width, and so on and each view is required to define them. The width and height of the view is normally set using the built-in constants as it ensures that the application will display the UI properly in varying screen sizes. The two constants used are as follows:

`wrap_content` – indicates that the view should size itself according to the dimension required by the content

`match_parent` – indicates that the view should be as large as the parent view group

Some of the common types of views in Android includes:

- **Basic Views** – Views such as 'TextView', 'EditText', and 'Button' are most frequently used.
- **Picker Views** – Views that allows users to choose a desired format/design from a list. Examples are the 'TimePicker' and 'DatePicker' views.
- **List Views** – Views that display a detailed list of items. This includes the 'ListView' and the 'Spinner' views.
- **Display Views** – Views that display images, such as the 'Gallery' and 'ImageSwitcher' views.
- **Menus** – Views that display additional and context-related menu items.

Figure 3.2 displays the different views in Android.



Figure 3.2: Different Views in Android

Other settings that the developer can work on include drawing, focus handling, event handling, settings for touch mode, and security settings.

- When drawing settings are provided, the View is able to draw according to the specified setting without invoking the `OnDraw()` option.
- Focus handling refers to the way in which movement of Views responds to user inputs. The `setFocusable()` option can be used for focus handling. For the touch mode, Views determine the focus from the `setFocusableInTouchMode()` option.
- View settings can be customized in a way that focus is handled smoothly irrespective of whether the user is in the touch mode or not. Interactive options, such as, buttons need to be provided for touch mode. When the user switches from the touch mode, the device will find a View and establish focus.
- When it comes to event handling, a View is designated an event, which it handles, and makes notifications to the listeners if required. During event processing, the View's appearance and measurements can be changed.

Finally, a View can also be customized by overriding the standard options. Overrides can be set for View creation, layout, drawing, event processing, focus, and window attachment.

3.2.3 Overview of Layouts

A layout is an extension of `ViewGroup` class. It defines the visual structure of the UI. It mainly comprises interconnected child views. It helps to define the architecture of the UI present in an `Activity`. Therefore, it can be used for developing complex interfaces.

The developer can either write code for specifying a layout, or create an XML file. Creating an XML file is a simpler way of specifying a layout. Every element in the XML file is either a `View` or a `ViewGroup` object. The Android framework gives the developer flexibility to use either or both methods to declare and control the UI layout.

→ Advantages of declaring UI in XML

The advantages of declaring the UI in XML are as follows:

- It enables the developer to separate the presentation of your application from the application logic that controls its behavior.
- The developer's UI descriptions are external to the application code, so that the developer can make changes without having to modify the source code and recompile.
- Declaring the layout in XML also helps in easy visualization of the structure of the UI.

As a result, it is easier to debug problems.

→ General rules for writing layout in XML

Using Android's XML vocabulary, the developer can quickly design UI layouts and their screen elements. It is very similar to creating Web pages with the help of HTML that is, by using a series of nested elements. The layout is specified in **activity_main.xml** file present in the `res/layout/` folder.

Here are some general rules for writing layout in XML:

1. Declaring UI elements in XML follows the same structure that one follows while naming classes and methods. In other words, it means that element and attribute name of UI elements matches the class name and method names respectively.
2. The direct relation between the class name and methods with that of UI element name and its attributes helps the developer to understand easily.
3. Vocabulary is same except for some instances where there is naming differences. For example, the `EditText` element has a `text` attribute that corresponds to `EditText.setText()`.
4. Each layout file must contain exactly one root element, which must be a `View` or `ViewGroup` object.
5. The developer needs to define the root element and then add additional layout objects or widgets as child elements to build a 'View' hierarchy that defines the layout.

6. Once the developer declares the default layout in a XML file, it is necessary to save it with the .xml extension in Android projects **res/layout/** directory to compile it properly.

Thus, to provide a layout in the XML file, the developer must name the XML element for a View with relation to the View class it signifies. For instance, the <EditText> element creates an <EditText> widget.

Code snippet 1 displays a simple RelativeLayout with an TextView and a Button.

Code Snippet 1:

```
<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">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world"/>

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="34dp"
        android:text="Start Music" />
</RelativeLayout>
```

→ Load the XML Resource

As already discussed, when the developer has finished creating the XML layout, it must be saved in the **res/layout/** directory with a **.xml** extension. The XML file in an application is compiled into a **View** resource. The developer must load the **View** resource in the application code. This is done in the **onCreate()** callback method of the **Activity** class. A reference to the **View** resource must be provided in the **setContentView()** callback feature as well. For instance, if the XML file is saved as **activity_main.xml**, it must be loaded in the application code for the **Activity** class.

Code snippet 2 displays the XML file **activity_main.xml** being loaded in the application code.

Code Snippet 2:

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

→ Reference the Widget Programmatically

The developer can also use the XML layout to add **View** object and provide references to them in the application source code. To do this, the developer has to perform the following steps:

1. Define a view or widget in the layout file, **activity_main.xml** and assign a unique ID. Code snippet 3 shows the declarations in the layout file of a **Button** widget.

Code Snippet 3:

```
...
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button" />
```

Note - It is important to designate an ID for a relative layout to enable sibling views to determine their layout relative to other sibling views based on the ID.

2. Create an instance for the View object, and obtain its reference programmatically from the layout file in the onCreate() method of the Activity class. Code snippet 4 demonstrates how to capture the widget from the layout.

Code Snippet 4:

```
Button buttonName = (Button) findViewById(R.id.button1);
```

Android supports different layouts or ViewGroup to construct the UI. The most common **layouts** include:

1. Linear layout.
2. Relative layout.
3. Listview.
4. Gridview.

Table 3.1 lists some of the common attributes of a ViewGroup.

Attributes	Description
layout_width	Defines the width of the ViewGroup
layout_height	Defines the height of the ViewGroup
layout_marginTop	Defines the space on the top of the ViewGroup
layout_marginBottom	Defines the space on the bottom of the ViewGroup
layout_marginLeft	Defines the space on the left of the ViewGroup
layout_marginRight	Defines the space on the right of the ViewGroup
layout_x	Defines the x-coordinate of the ViewGroup
layout_y	Defines the y-coordinate of the ViewGroup

Table 3.1: Common Attributes of ViewGroup

3.3 Working with Layouts

While working with layouts, it is preferable that the developers work using XML from external resources. The XML layout will comprise a root node. This root node can have multiple nested layouts and views for designing the UI. The various layouts for an Android application are explained in detail.

→ XML for UI Layouts

The Android development plugin for Eclipse includes a layout resource designer for designing and previewing layout resources. The tool has following two tab views:

1. **Graphical Layout View:** This allows the user to preview how the controls will appear on various screens and orientation and is shown in figure 3.3.

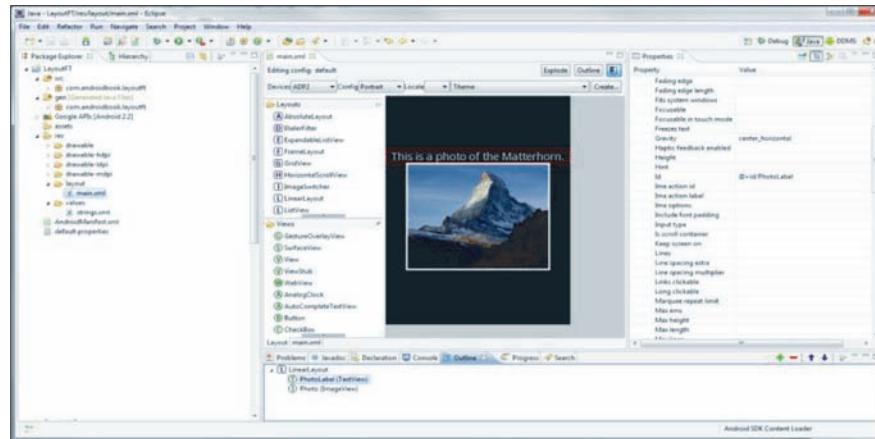


Figure 3.3: Layout Resource Designer in Eclipse

2. **XML View:** This shows the user the XML definition of the resource as shown in figure 3.4.



Figure 3.4: Project Folder Structure

The Android SDK also has several tools that can help the user design, debug, and optimize layout resources. In addition to the layout resource designer built into the Android plugin for Eclipse, the Hierarchy Viewer and ‘**layoutopt**’ tools provided with the Android SDK can also be used. These tools are available in the **/tools** directory of Android SDK.

- The ADT Plugin for Eclipse offers a layout preview of the user’s XML file by opening the XML view of the file. To see the graphical view select the **Graphical Layout** tab.
- The user can try the **Hierarchy Viewer** tool, for debugging layouts — it reveals layout property values, draws wireframes with padding/margin indicators, and displays fully rendered views while the user debug on the emulator or device.

- The **layoutopt** tool lets the user quickly analyze the layouts and hierarchies for inefficiencies or other problems.

While working with the layout resource designer in Eclipse:

- Use the Outline pane to add or remove controls to the layout resource.
- Select a specific control (either in the Preview or the Outline) and use the Property pane to adjust a particular control's attributes.
- Use the XML tab to edit XML definition directly.

3.3.1 Linear Layout

It is the simplest type of layout. `LinearLayout` aligns all its child nodes in a single direction, either vertically or horizontally, depending on the `orientation` attribute specified by the user. A scrollbar appears if the window length exceeds the length of the screen. The `android: orientation` attribute helps the developer to specify the layout direction.

This section explains how to create a `LinearLayout` with vertical orientations.

The steps to create a `LinearLayout` are as follows:

- Start Eclipse IDE.
- Click **File → New → Project** to display the **New Project** window.
- Select **Android → Android Application Project**.
- Click **Next** to display the **New Android Application** screen.
- Type the application name as **Layoutexample** and leave the default settings.
- Click **Next**. Leave the default options in the **Configure Project** pane of **New Android Application** dialog box.
- Click **Next**. Can change the image if required in the **Configure Launcher Icon** pane of **New Android Application** dialog box.
- Click **Next** and select **BlankActivity** in the **Create Activity** pane of **New Android Application** dialog box.
- Click **Next**. Modify the name of the `Activity` class if required in the **Blank Activity** pane of **New Android Application** dialog box.
- Click **Finish**. By default android creates `RelativeLayout`.
- To create `linearlayout.xml` file right-click the `res/layout/` folder.

12. Click **New** → **Android XML File** to display the **New Android XML File** dialog box as shown in figure 3.5.

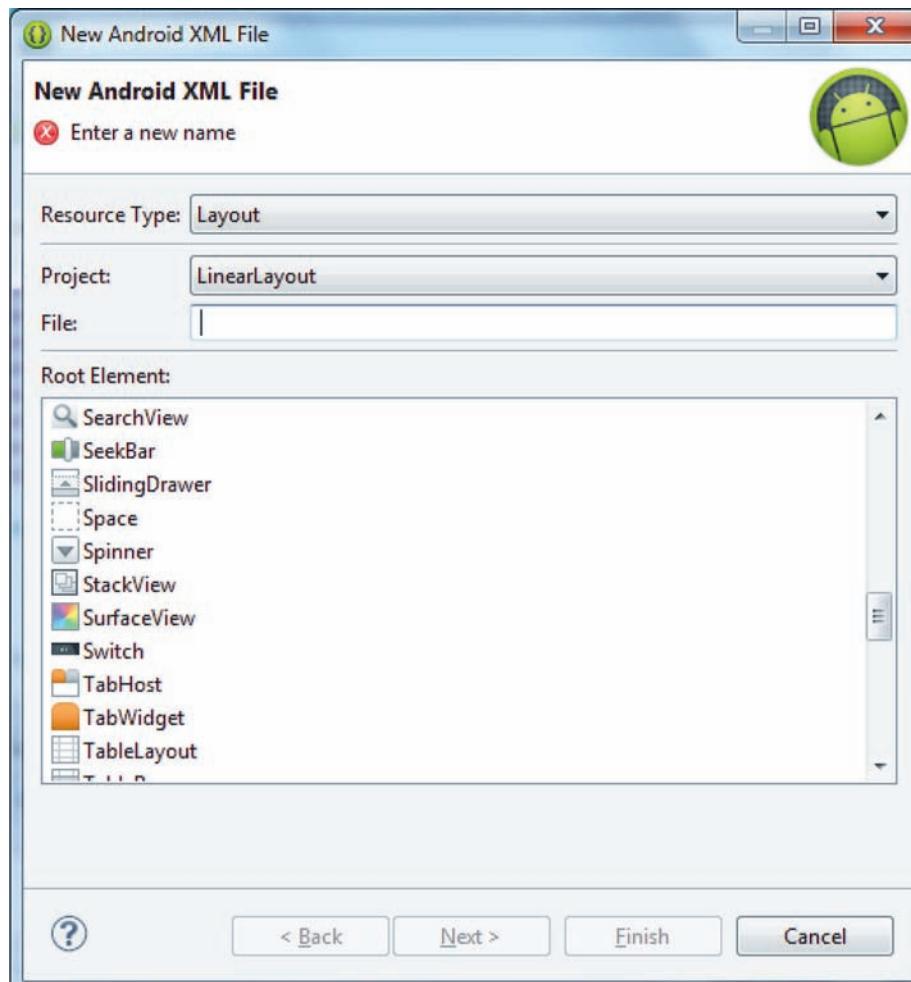


Figure 3.5: New Android XML File

In the **New Android XML File** dialog box make sure that the **Resource Type** is set to **Layout**.

13. Type **linearlayout** in the **File** box and in **Root Element** options choose **Linear Layout**.
14. Click **Next**.
15. Click **Finish**.
The **linearlayout.xml** file is displayed and if it is not displayed then double-click and open the file from the project explorer pane.
16. Modify the code in the **linearlayout.xml** file as shown in code snippet 5.

Code Snippet 5:

```
<?xmlversion="1.0"encoding="utf-8"?>  
  
<LinearLayoutxmlns: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:orientation="vertical"  
    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=".LinearLayout">  
  
    <EditText  
        android:id="@+id/editText"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:ems="10"  
        android:hint="Enter Some Text">  
  
        <requestFocus/>  
    </EditText>  
  
    <Button  
        android:id="@+id/button"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_marginTop="10dp"  
        android:text="Submit"/>  
  
    <TextView  
        android:id="@+id/textView"  
        android:layout_width="wrap_content"
```

```

    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:text=""
    android:textAppearance="?android:attr/textAppearanceLarge" />

    <Button
        android:id="@+id/showNotification"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:text="Show Notification"/>

    <Button
        android:id="@+id/buttonNext"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:text="Next"/>

</LinearLayout>

```

17. Modify the code in **MainActivity.java** file to load the **linearlayout.xml** file as the layout view as shown in code snippet 6.

Code Snippet 6:

```

package com.example.linearLayout;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class MainActivity extends Activity {

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

```

```
@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;  
}  
}
```

18. Click **File → Save All** to save all the changes.

19. Click **Run → Run**.

The Emulator starts with the **LinearLayout** project. The output of the project will be as shown in figure 3.6.

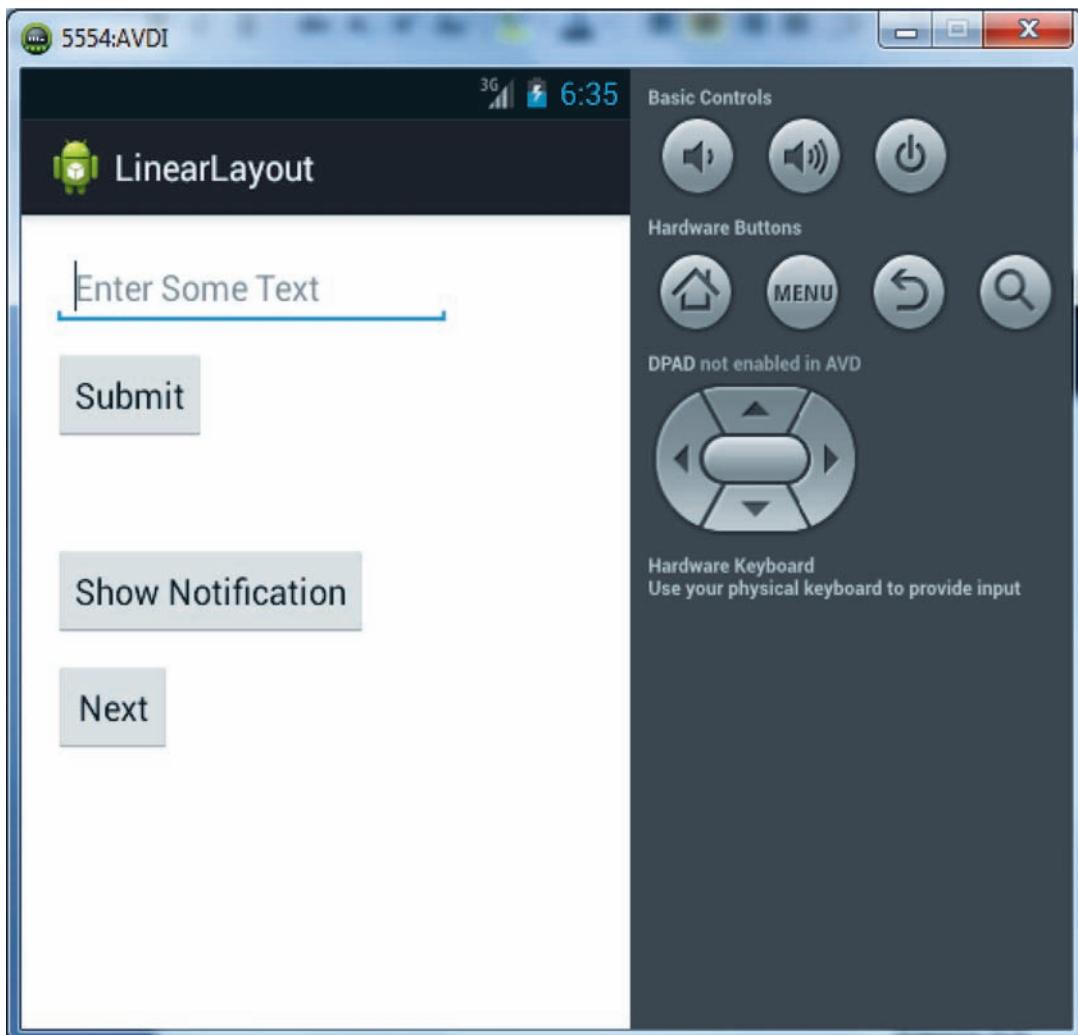


Figure 3.6: Linear Layout

In the code snippet the attributes used are as follows:

`android:layout_width="match_parent"`: It specifies the width of the layout. The width may either have the value set to `wrap_content` or `match_parent` which means that the view must be as large to enclose its content or matches the size of the parent view respectively.

`android:layout_height="match_parent"`: It specifies the height of the layout.

`android:orientation="vertical"`: It specifies the orientation of the layout.

Two types of orientations are vertical and horizontal, which displays layout either horizontally and vertically.

3.3.2 Relative Layout

In this type of layout, the developer can specify the location of child objects relative to each other or to the parent. A relative layout is very useful for designing a user interface because it can remove the need for nested view groups keeping the layout tree simple, thus improving performance. If a developer has several nested linear layout groups, the groups could be replaced with a single relative layout instead.

The steps to create a Relative Layout are as follows:

1. Create a new project named **RelativeLayout**.
2. Copy a picture or an image in the `/res/drawable-hdpi` folder to display an image. Here the image file is named as **koala.jpg**.

Note - Name of the image file should be in lower case and there should be no spaces in the file name.

3. Double-click the **activity_main.xml** file from the `res/layout/` folder to open it.
4. Modify the code in the **activity_main.xml** file as shown in code snippet 7.

Code Snippet 7:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
```

```
<ImageView  
    android:id="@+id/imageView"  
    android:layout_width="100dp"  
    android:layout_height="100dp"  
    android:layout_alignParentLeft="true"  
    android:layout_below="@+id/button"  
    android:layout_marginLeft="27dp"  
    android:layout_marginTop="32dp"  
    android:src="@drawable/koala" />  
  
<Spinner  
    android:id="@+id/spinner"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignBottom="@+id/imageView"  
    android:layout_alignParentRight="true"  
    android:layout_marginBottom="15dp"  
    android:layout_marginRight="28dp"  
    android:layout_marginLeft="20dp"  
    android:layout_toRightOf="@+id/imageView" />  
  
<TextView  
    android:id="@+id/timeView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignLeft="@+id/button"  
    android:layout_below="@+id/button"  
    android:textAppearance="?android:attr/textAppearanceMedium" />  
  
<Button  
    android:id="@+id/button"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_alignParentTop="true"
```

```
        android:layout_centerHorizontal="true"
        android:layout_marginTop="31dp"
        android:layout_marginLeft="50dp"
        android:layout_marginRight="50dp"
        android:text="Get time"/>

<CheckBox
    android:id="@+id/checkBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
        android:layout_alignRight="@+id/imageView"
    android:layout_below="@+id/imageView"
    android:layout_marginTop="20dp"
    android:text="CheckBox" />

<RatingBar
    android:id="@+id/ratingBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/checkBox"
    android:layout_below="@+id/checkBox"
    android:layout_marginTop="20dp" />

<Button
    android:id="@+id/exit"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignRight="@+id/spinner"
    android:layout_marginBottom="20dp"
    android:text="Exit" />
</RelativeLayout>
```

5. Click **File → Save All** to save all the changes.
6. Click **Run → Run**.

The Emulator starts with the **RelativeLayout** project. The output of the project will be as shown in figure 3.7.

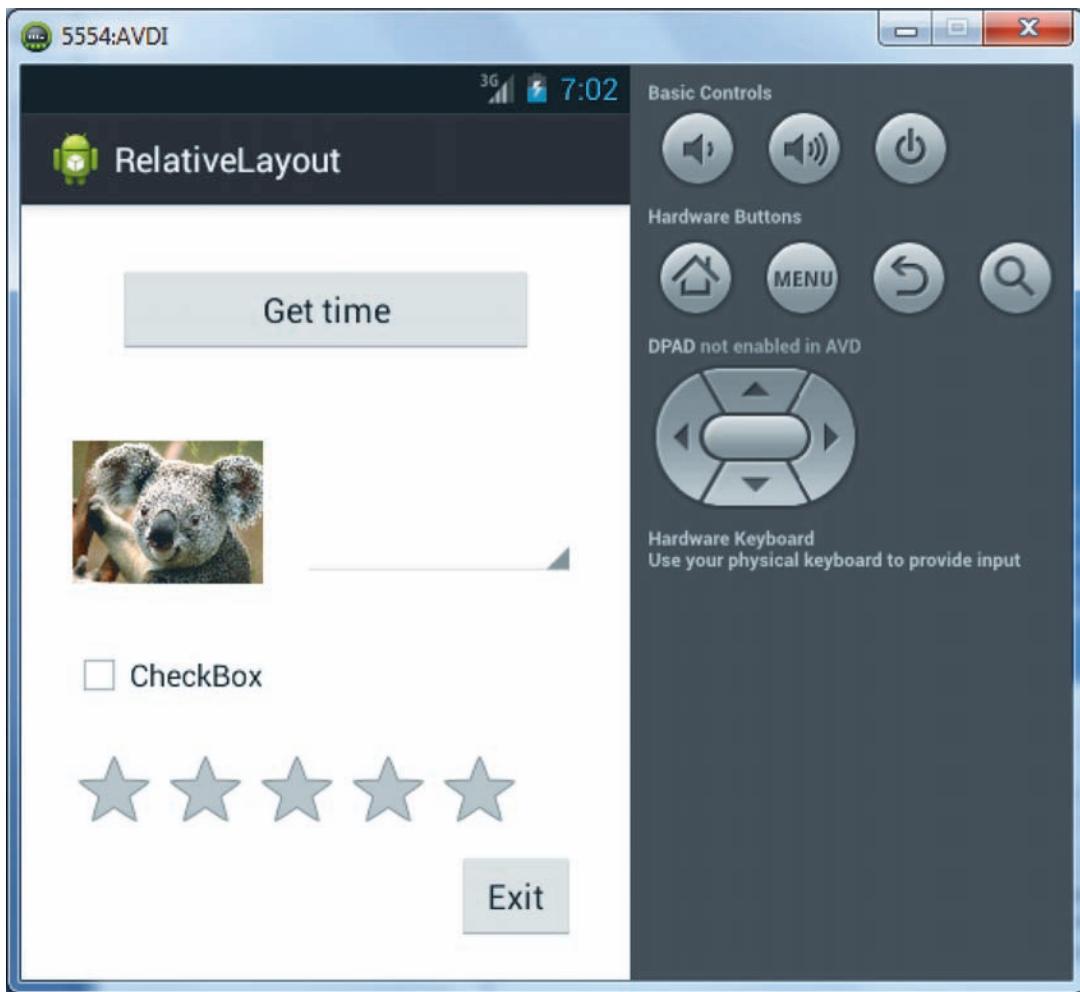


Figure 3.7: RelativeLayout – Output

3.3.3 List View

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

3.3.4 Grid View

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

The steps to create a **GridView** layout are as follows:

1. Create a new project named **GridViewLayout** in Eclipse.
2. Navigate to **GridViewLayout** → **res** → **drawable-hdpi** in the **Package Explorer** pane.
3. Copy - paste **.jpg** or **.png** files that are to be displayed in the image grid. Save the image files into the project's **res/drawable-hdpi/** directory.

Note - Name of the image should be in lower case and there should be no spaces in the file name.
4. Navigate to **GridViewLayout** → **res** → **layout** → **activity_main.xml** in the **Package Explorer** pane.
5. Double-click the **activity_main.xml** file to open it.
6. Modify the code in **activity_main.xml** file as shown in code snippet 8.

Code Snippet 8:

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

</GridView>
```

7. To create **animals.xml** file right-click the **res/layout/** folder.
8. Click **New** → **Android XML File** to display the **New Android XML File** dialog box.
9. Type **animals** in the **File** box.
10. Click **Next**.

11. Click **Finish**.
12. The **animals.xml** file is displayed and if it is not displayed then double-click and open the file from the project explorer pane.
13. Modify the code in the **animals.xml** file as shown in code snippet 9.

Code Snippet 9:

```
<?xmlversion="1.0"encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com/apk/
res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="5dp" android:orientation="vertical">

    <ImageView
        android:id="@+id/grid_item_image"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_marginRight="20dp"
        android:src="@drawable/ic_launcher">
    </ImageView>

    <TextView
        android:id="@+id/grid_item_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@+id/label"
        android:layout_marginTop="5px"
        android:textSize="18sp">
    </TextView>

</LinearLayout>
```

14. Open the Activity class named **MainActivity.java** file present in the **src** folder.

15. Modify the code `onCreate()` method as shown in code snippet 10.

Code Snippet 10:

```
package com.example.gridviewlayout;

import android.os.Bundle;

import android.app.Activity;

import android.view.Menu;

import android.view.View;

import android.widget.AdapterView;

import android.widget.GridView;

import android.widget.TextView;

import android.widget.Toast;

import android.widget.AdapterView.OnItemClickListener;

public class MainActivity extends Activity {

    GridView gridView;

    static final String[] IMAGES = new String[] { "Image 1",

        "Image 2", "Image 3", "Image 4" };

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        gridView = (GridView) findViewById(R.id.gridView1);

        gridView.setAdapter(new ImageAdapter(this, IMAGES));

        gridView.setOnItemClickListener(new

        OnItemClickListener() {

            public void onItemClick(AdapterView<?> parent,

                View vw,

                int position, long id) {

                Toast.makeText(getApplicationContext(),

                    "Click ListItem Number " + position, Toast.LENGTH_-

                    LONG)
            }
        });
    }
}
```

```
        .show( ) ;

    }

} ) ;

}

@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);

    returntrue;

}

}
```

In the code, the **activity_main.xml** layout is set for the content view and the Grid is captured from the layout using `findViewById(int)` method.

The `setAdapter()` method then sets a custom adapter named `ImageAdapter` as the source for all items to be displayed in the grid.

To initiate an action when an item in the grid is clicked, the `setOnItemClickListener()` method passes a new anonymous instance of `AdapterView.OnItemClickListener`. This anonymous instance defines the `onItemClick()` callback method to display a `Toast` message. The message displays the index position (zero-based) of the selected item (in a real world scenario, the position could be used to get the full sized image for some other task).

16. To create a new class called **ImageAdapter** that extends the **BaseAdapter** class, right-click **src** → **com.example.gridviewlayout** folder to display the context menu.
 17. Click **New** → **Class** to display the **New Java Class** dialog box.
 18. Type **ImageAdapter** in the **Name** box.
 19. Type the code as shown in code snippet 11.

Code Snippet 11:

```
package com.example.gridviewlayout;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
import android.widget.TextView;

imageView.setImageResource(R.drawable.lighthouse);

}

} else {

    gridView = (View) convertView;

}

return gridView;

}

@Override

public int getCount() {

    return values.length;

}

@Override

public Object getItem(int position) {

    return null;

}

@Override

public long getItemId(int position) {

    return 0;

}

}
```

The code implements some required methods inherited from `BaseAdapter` class. The constructor and `getCount()` method are self-explanatory. The `getView()`

method creates a new `View` object for each image added to the `ImageAdapter` class. In other words, it displays the image at the specified position in the data set. When this method is invoked, a `View` object is passed and a check is performed to see if the object is null. If it is null, an object of `ImageView` class is instantiated and configured with desired properties for the image presentation.

This class will be used to access the data. In this case, it will access the images. The `getCount()` method is overridden and returns a count of the total number of images present in the data set. The `getItem()` method returns the image present at the specified position in the data set. Finally, the `setImageResource()` method is used to initialize the `Image` object by passing image name.

20. Execute the application by selecting **Run → Run**.

The output of the application will be as shown in figure 3.8.

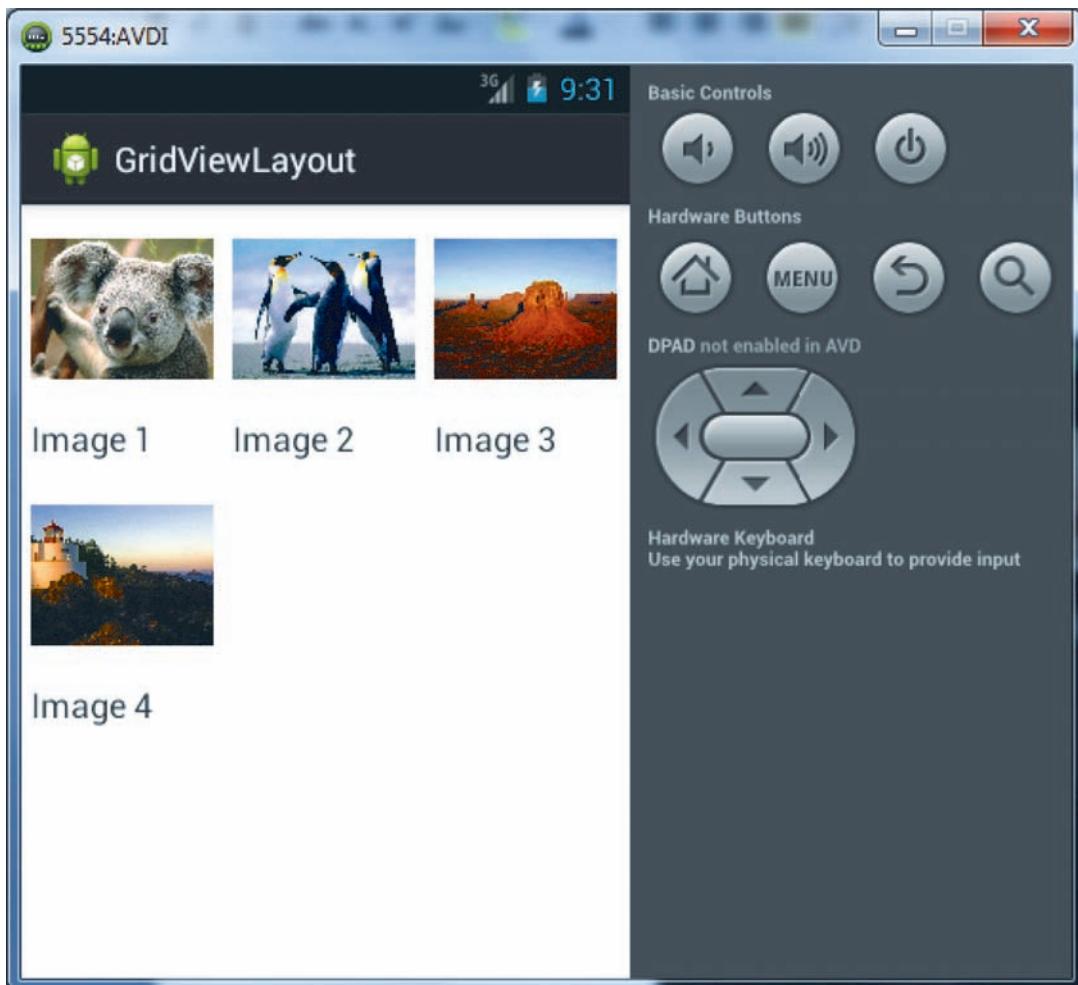


Figure 3.8: GridView Layout

Note - If any of the image is clicked, its position is displayed.

3.4 UI Components

In Android, UI components are interactive controls in an app's UI that enables the user to perform a wide range of actions. Through UI components the user can interact with application. Some of the basic UI components are Button, TextView, DatePicker, ProgressBar, and so on as shown in figure 3.9.

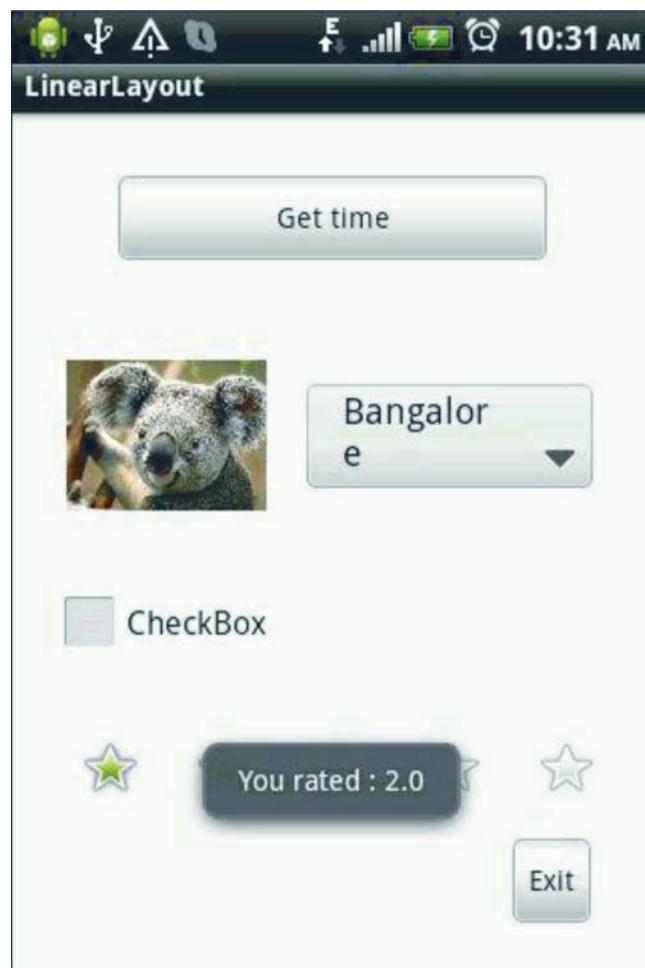


Figure 3.9: UI Input Control Components

3.4.1 Input Controls

As discussed, Android provides a number of input controls. The different input controls are listed in table 3.2.

Input Controls	Description	Code Snippet
Button	It consists of a text or icon or a combination of both, that communicates what action would occur if user clicks it.	<pre><Button android:layout_height="wrap_content" android:text="@string/button_text" android:drawableLeft="@drawable/button1_icon"/></pre>
TextField	<p>It allows user to type text. When a user touches a text field, it places the cursor and automatically pops up the keyboard. In addition to typing, text fields allow other activities, such as text selection (cut, copy, paste) and data look-up via auto-completion.</p> <p>It is possible to add a text field to the layout with the <code>EditText</code> object. The user can do this in XML layout with a <code><EditText></code> element. The type of keyboard required can be specified for your <code>EditText</code> object with the <code>android:inputType</code> attribute. Some of the commonly used input type values that define keyboard behaviors:</p> <ul style="list-style-type: none"> • “text” - Normal text keyboard. • “textEmailAddress” - Normal text keyboard with the @ character. • “textUri” - Normal text keyboard with the / character. • “number” - Basic number keypad. • “phone” - Phone-style keypad. 	<pre><EditText android:id="@+id/number" android:layout_width="fill_parent" android:layout_height="wrap_content" android:hint="@string/sms_hint" android:inputType="number"/></pre>

Input Controls	Description	Code Snippet
Checkbox	It allows the user to select one or multiple options from available set. When the user selects a CheckBox, the CheckBox object receives an on-click event. To create each checkbox option, create a CheckBox in the layout. Each checkbox is managed separately, hence it is necessary to register an event listener for each.	<pre><CheckBox xmlns:android="http://schemas. android.com/apk/res/android" android:id="@+id/ checkBox1" android:layout_ width="wrap_content" android:layout_ height="wrap_content" /></pre>
Toggle button	It allows user to change a setting between two states. The user can add a basic toggle button to the layout with the ToggleButton object. Android 4.0 introduces another kind of toggle button called a Switch. This provides a slider control, which can be added with a Switch object. When the user selects a ToggleButton and Switch, the object receives an onClick event.	<pre><ToggleButton android:id="@+id/ togglebutton1" android:layout_width="wrap_ content" android:layout_ height="wrap_content" android:textOn="ON" android:textOff="OFF" android:onClick="onToggleCl icked" /></pre>

Input Controls	Description	Code Snippet
Radio button	<p>It allows users to select one option from a set. When the user selects one of the radio buttons, the corresponding RadioButton object receives an onClick event.</p>	<pre><?xml version="1.0" encoding="utf-8"?> <RadioGroup xmlns:android= "http://schemas.android.com/ apk/res/android" android:layout_width="fill_ parent" android:layout_ height="wrap_content" android:orientation="verti cal"> <RadioButton android:id="@+id/ radio1" android:layout_ width="wrap_content" android:layout_ height="wrap_content" android:text="@ string/Male" android:onClick="onRadi oButtonClicked"/> <RadioButton android:id="@+id/ radio2" android:layout_ width="wrap_content" android:layout_ height="wrap_content" android:text="@string/ Female" android:onClick="onRadi oButtonClicked"/> </RadioGroup></pre>
Spinner	<p>It enables the user to quickly select an option or value from a drop-down list. Spinners provide a quick way to select one value from a set. In the default state, a spinner shows its currently selected value. When the user touches the spinner, it displays a drop-down menu with all other available values, from which the user can select a new one.</p>	<pre><Spinner xmlns:android="http:// schemas.android.com/apk/res/ android" android:id="@+id/spinner1" android:layout_width="wrap_ content" android:layout_ height="wrap_content" /></pre>

Input Controls		Description	Code Snippet
Picker		<p>It enables the user to select time or pick a date in required format like AM/PM or DD-MM-YY. It is recommended that the developer utilizes DialogFragment to host each time or date picker. The DialogFragment manages the dialog lifecycle and allows display of the pickers in different layout configurations, such as in a basic dialog on handsets or as an embedded part of the layout on large screens.</p>	<pre><DatePicker xmlns:android="http://schemas.android.com/apk/res/android" android:id="@+id/datePicker1" android:layout_width="wrap_content" android:layout_height="wrap_content" /></pre>

Table 3.2: Input Controls

3.4.2 Menus

Menu is an UI component which holds several items that provide navigation or settings or more functionality to an application. It is a common interface component seen in android phones and appears when menu buttons on the device are clicked. The menu displays all available options. Android 3.0 replaces the standard menu button on device with the Action bar menu button as shown in figure 3.10.

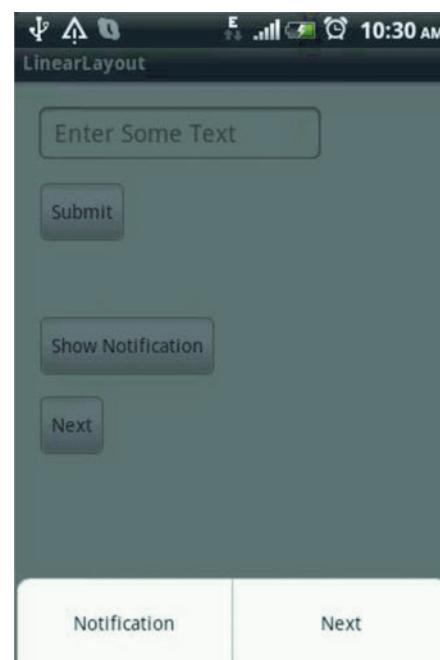


Figure 3.10: Action Menu in Android

3.4.3 Action Bar

Action bar is always present at the top of the android screen, which is used to identify the application, user location, user action, and other user interface elements, through which user can navigate or perform an action. It provides user actions and navigation modes. Commonly used actions can directly be taken from options menu and placed in the action bar. Other options are available in ‘overflow menu’ button in the action bar as shown in figure 3.11.

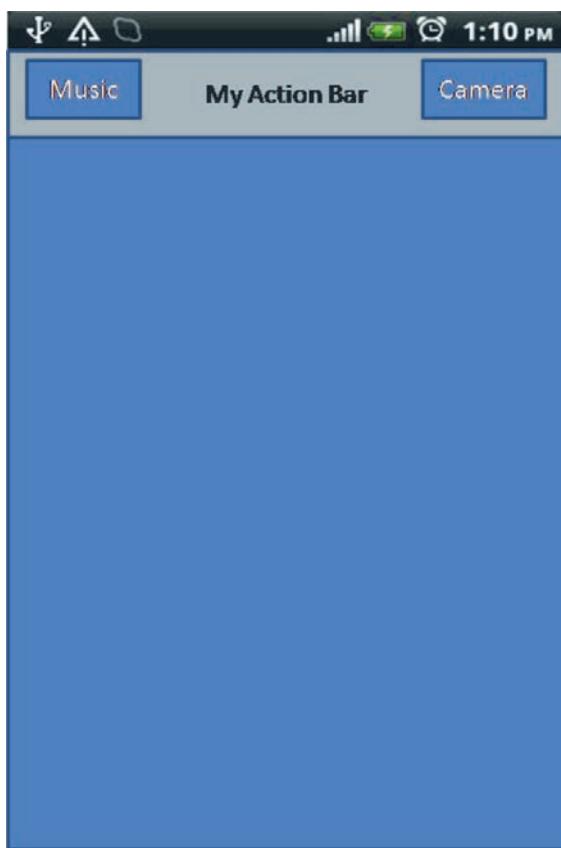


Figure 3.11: Action Bar

3.4.4 Settings

Settings allows user to customize the behavior or appearance of the application or a mobile. Examples include setting the ‘save message’ limit or specifying whether notifications can be saved. Changing the network status, managing installed applications, and so on are other settings that can be performed.

Click the launcher icon in emulator as shown in figure 3.12.



Figure 3.12: Launcher Icon

This will display the **Settings** icon as shown in figure 3.13.

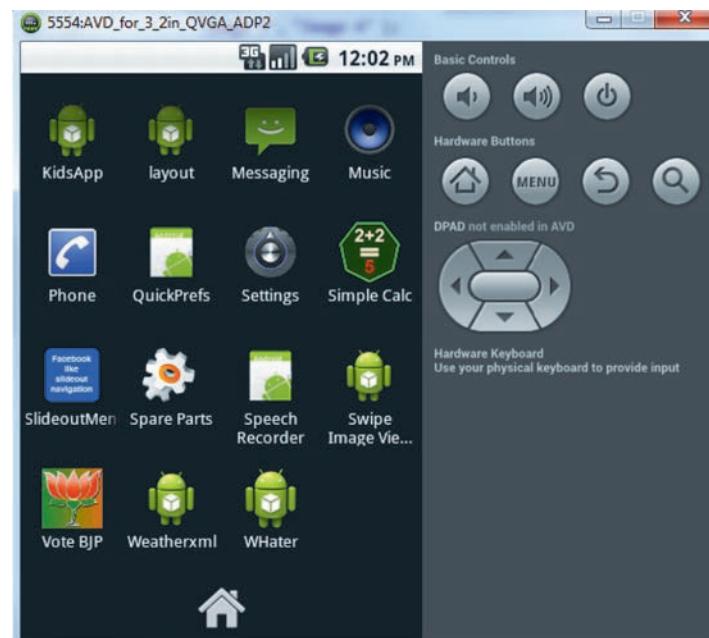


Figure 3.13: Settings Icon

Clicking the **Settings** icon will display the different elements whose value can be set as shown in figure 3.14.

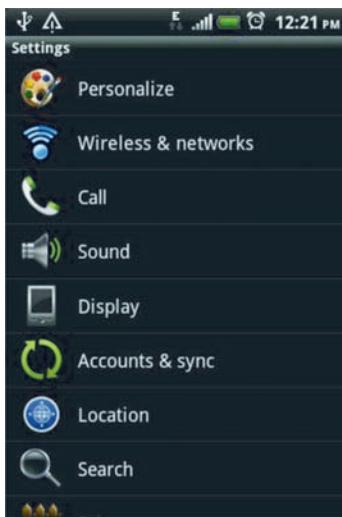


Figure 3.14: Setting View

3.4.5 Dialogs

Dialogs are prompt or alert displayed to the user to take a decision or to input any information. The dialogs are also used to notify user when a task has been completed. It does not fill the entire screen and usually appears when a user has to take a particular action before proceeding.

The different types of Dialogs are **AlertDialog**, **Toast**, **TimePicker**, **DatePicker**, and so on.

3.4.6 Some Additional Views

Other than the views discussed, Android contains some important views such as:

- **videoView** – Provides layout to play video files.
- **GalleryView** – Provides layout to view all the images in a gallery view.
- **TabHost** – Is used to maintain tabs in an application such as in default contacts application. Each tab contain child layout to navigate within.
- **WebView** – Is used to load URL. WebView is like a browser which will display the Web content within.

3.4.7 Status Notifications

Notifications are used to notify or provide alerts to the user when a message or notification arrives. Notification contains icon, title, body, and the notification arrival time. The Notification Manager ensures that the status bar icons are updated regularly. A status bar notification displays an icon on the status bar along with a message. When the notification is chosen, an Intent is sent by Android to launch the Activity. The status bar notification

can be initiated by an Activity or Service class.

A status bar notification is created when the background service wants to inform the user about an event that requires the user to respond.

The user can view the details by opening the ‘Notifications drawer’ as shown in figure 3.15.

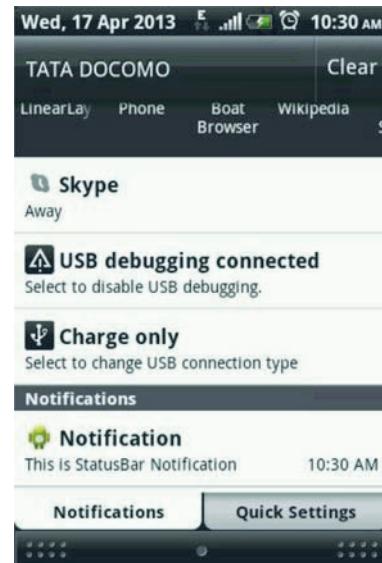


Figure 3.15: Notification in Notification Drawer

3.4.8 Toasts

These are simple messages that provide feedback about a user’s action in a popup window. It is displayed in a small window and does not interfere with the user’s ongoing activity. Figure 3.16 displays a Toast message.

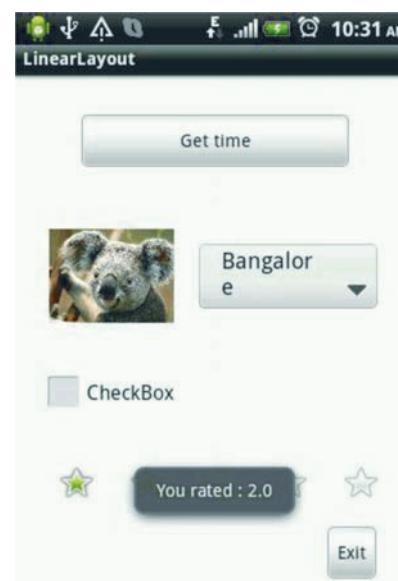


Figure 3.16 Toast Message

3.4.9 Search

It is an important feature in Android and enables user to search for an item in the gadget or the Internet by entering a keyword, as shown in figure 3.17.



Figure 3.17: Search

3.4.10 Drag and Drop

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

3.4.11 Accessibility

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

3.5 Styles and Themes

Presentation and formatting of an application can be improved using Styles and Themes. Android does possess default styles and themes. A style can be referred as a collection of properties that specify the look and format for a View or window in UI. A style can specify properties such as height, padding, font color, font size, background color, and much more. A style is defined in an XML resource that is separate from the XML that specifies the layout.

Styles in Android are similar to cascading stylesheets in Web design. In other words, they allow the user to separate the design from the content.

All of the attributes related to style can be removed from the XML layout file and incorporated into a style definition file. This can be named any arbitrary name. This is then applied with the style attribute.

A theme is a style applied to an entire Activity or application, rather than an individual View. When a style is applied as a theme, every View in the Activity or application will apply each style property that it supports.

The user can create a set of styles and save them in an XML file in the **res/values/** directory of the project. The name of the XML file is arbitrary, but it is mandatory to use the **.xml** extension and save the file in the **res/values/** folder. It should also be noted that the root node of the XML file must be `<resources>`.

3.5.1 Defining Styles and Theme

To create a style it is necessary to add a `<style>` element to the file with a name attribute that uniquely identifies the style. This attribute is not optional. To this `<style>` element, an `<item>` element is added to represent each property of that style. The name attribute of the item element is used for declaring the style property and a value to go with it. This attribute is not optional (this attribute is required). The value for the `<item>` element can be a keyword string, color represented in a hexadecimal format, a reference to another resource type, or other value depending on the style property.

The example will explain how to apply styles in detail.

The steps to create a **StyleExample** project are as follows:

1. Start Eclipse IDE.
2. Click **File → New → Project** to display the **New Project** window.
3. Select **Android → Android Application Project**.
4. Click **Next** to display the **New Android Project** screen.
5. Type **StyleExample** as the application name and leave the default settings.
6. Click **Next**. Leave the default options selected in the **Configure Project** screen of the **New Android Application** dialog box.
7. Click **Next. Configure Launcher Icon** pane of the **New Android Application** dialog box is displayed. Developer can change the image if required.
8. Click **Next**. Select **BlankActivity**.
9. Click **Next. Blank Activity** pane of the **New Android Application** dialog box is displayed.

10. Click **Finish**.
11. Create a new xml file named **newstyle** inside the **/res/value** directory by right-clicking the **res/value** folder to display the context menu.
12. Select **New → XML** file.
13. Modify the code as shown in code snippet 12.

Code Snippet 12:

```
<?xml version="1.0" encoding="UTF-8"?>  
  
<resources>  
  
    <style name="NewStyle" parent="@android:style/TextAppearance.Medium">  
        <item name="android:layout_width">fill_parent  
        </item>  
        <item name="android:layout_height">wrap_content  
        </item>  
  
        <item name="android:background">#551033  
        </item>  
        <item name="android:textColor">#86C67C  
        </item>  
        <item name="android:typeface">monospace  
        </item>  
    </style>  
</resources>
```

14. Navigate and open **res → layout → activity_main.xml** file.
15. Add the **<style>** element in **<TextView>** as shown in code snippet 13.

Code Snippet 13:

```
<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">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        style="@style/NewStyle"
        android:text="@string/hello_world"/>

</RelativeLayout>
```

Once you run the application, the output will be as shown in figure 3.18.

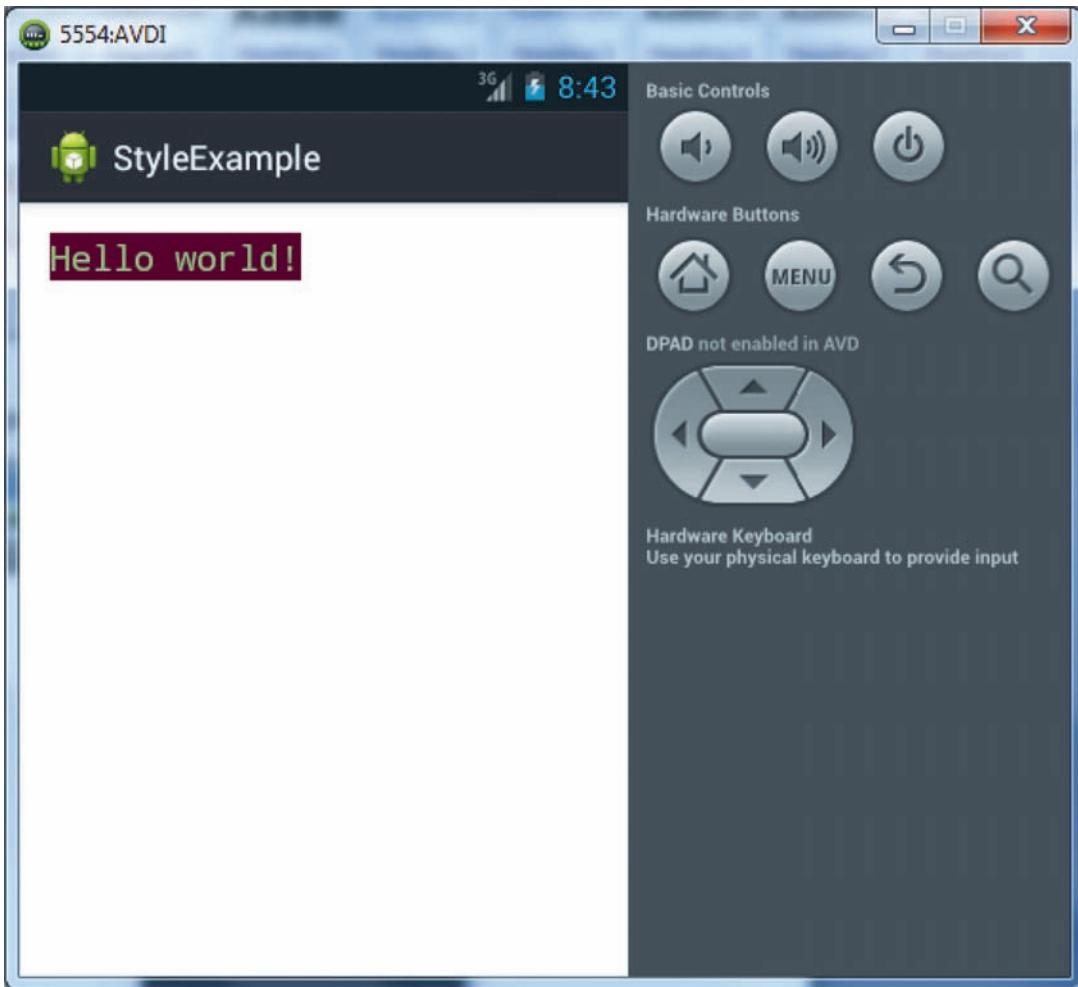


Figure 3.18: Adding Styles

3.5.2 Inheriting Styles

The `<style>` element has the `parent` attribute which allows the user to specify a style from the parent and inherit its properties. In other words, the user can utilize this to inherit properties from an existing style and then can define only the properties that are required to be modified.

It is possible to inherit the Android platform's default text appearance and then modify it according to the developer's needs.

If the developer wishes to inherit from self-defined styles, it is not necessary to use the `parent` attribute. Instead, the developer can just prefix the name of the style to be inherited to the name of the new style, separated by a period.

For example, to create a new style that inherits the `NewStyle` style defined, but make the color red, the user can author the new style as shown in code snippet 14.

Code Snippet 14:

```
<style name="NewStyle.Red">
    <item name="android:textColor">#FF0000</item>
</style>
```

3.5.3 Style Properties

The best place to find properties that apply to a specific View is the corresponding class reference, which lists all the supported XML attributes. For example, all the attributes listed in the table of `TextView` XML attributes can be used in a style definition for a `TextView` element (or one of its subclasses).

Some style properties are not supported by any `View` element and can only be applied as a theme. These style properties apply to the entire window and not to any type of `View`. For example, style properties for a theme can hide the application title, hide the status bar, or change the window's background. This kind of style properties do not belong to any `View` object. To find out theme-only style properties, the user can look at the `R.attr` reference for attributes that begin with `window`.

3.5.4 Application of Styles and Themes to the UI

A style can be set in the following two ways:

1. Adding the `style` attribute to a `View` element in the XML for the layout. This is added to an individual `View`.
2. Adding the `android:theme` attribute to the `<activity>` or `<application>` in the Android manifest file that sets the style to the entire application.

When the developer applies a style to a single `view` in the layout, the properties defined by the style are applied only to that `view`. If a style is applied to a `ViewGroup`, the child `View` elements will not inherit the style properties. It will be applied to only the element where the style and its properties are applied. However, the style can be applied as a theme by the developer so that it applies to all `View` elements.

To apply a style definition as a theme, the user must apply the style to an `Activity` or `application` in the Android manifest. Every `View` within the `Activity` or `application` will apply each property that it supports.

3.5.5 Application of a Theme to an Activity or Application

To set a theme for all the activities of an application, open the `AndroidManifest.xml` file and edit the `<application>` tag to include the `android:theme` attribute with the style name.

If the theme is to be applied to a single Activity in the application, then the android:theme attribute is required to be added to the <activity> tag instead. Just as Android provides other built-in resources, there are many pre-defined themes that can be used.

If you like a theme, but want to tweak it, just add the theme as the parent of your custom theme.

The example will explain how to use Theme in an application.

The steps to create a **ThemeExample** project are as follows:

1. Start Eclipse IDE.
2. Click **File → New → Project** to display the **New Project** window.
3. Select **Android Project**.
4. Click **Next** to display the **New Android Project** screen.
5. Type **ThemeExample** as the application name and leave the default settings.
6. Click **Next**. Leave the default options selected.
7. Click **Next** to display the **Configure Launcher Icon** pane. Developer can change image if required.
8. Click **Next** and select **BlankActivity**.
9. Click **Next**.
10. Click **Finish**.
11. Navigate to **res → values → styles.xml** file in the **Package Explorer** pane.
12. Double-click **styles.xml** file.
13. Modify the code as shown in code snippet 15.

Code Snippet 15:

```
<resources>

<style name="NewStyle">
<item name="android:windowNoTitle">true</item>
<item name="android:text">This is a theme.</item>
<item name="android:textColor">#66FF66</item>
<item name="android:textSize">30sp</item>
<item name="android:windowBackground">@drawable/lighthouse</item>
</style>

</resources>
```

14. Double-click **AndroidManifest.xml** file.
15. Modify the code under **<application>** tag in **AndroidManifest.xml** file to apply the **android:theme** attribute as shown in code snippet 16.

Code Snippet 16:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.themeexample"
    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/NewStyle" >
        <activity
            android:name="com.example.themeexample.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>
```

Once you execute the project, the output will be as displayed in figure 3.19.

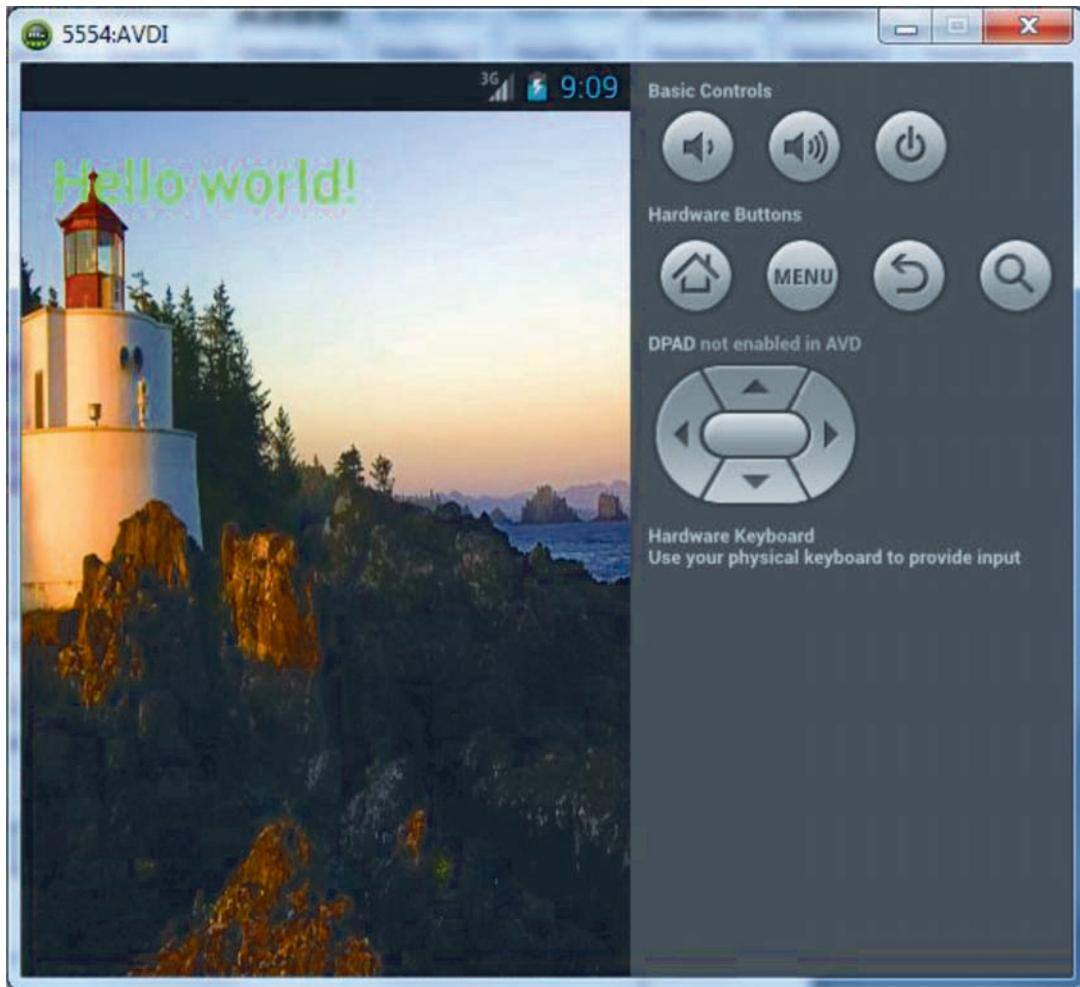


Figure 3.19: Themes in Application

Using Platform Styles and Themes:

The Android platform provides a large collection of styles and themes that can be used in applications. The user can find a reference of all available styles in the `R.style` class. To use the styles listed there, replace all underscores in the style name with a period.

Code snippet 17 displays the use of **Dialog** theme in **AndroidManifest.xml**.

Code Snippet 17:

```
<activity android:theme="@android:style/Theme.Dialog">
```

It should be noted that the `R.style` reference is not well documented and does not thoroughly describe the styles. So, viewing the actual source code for these styles and themes will give the user a better understanding of what style properties each one provides.

3.6 Handling User Events

Android provides a number of ways to intercept the events generated by a widget present in an Android application. In Android, an event refers to a response generated for an external action. An example of an event is a user interacting with the help of a Touchscreen. The Android framework maintains an Event Queue in which the events are arranged as they occur. The events are removed in First-In-First-Out (FIFO) basis.

When a user touches the screen, it is an input event. The event is passed to the `View` positioned on the screen along with information. The `View` requires an ‘event listener’ to handle the event. The Android `View` class comes equipped with a range of event listener interfaces. Each of these, in turn, contains an abstract declaration for a callback method.

The two actions that needs to be performed by the developer to inform the user about the user input events are as follows:

- Defining an event listener and registering it with the view.
- Overriding an existing callback method for the view.

In order to handle a particular event, it is important that the Event Listener implements the corresponding callback method. For example, if a button should respond to a click event when a user touches it on the screen, then it must register the `View` with a click listener and implement the corresponding `onClick` callback procedure.

`View` class consists of a collection of nested interfaces with callback methods. These interfaces are known as event listeners and are generally named as ‘`On <something> Listener`’. The callback methods has the following naming pattern, `On <something>()`. Callback methods are the methods adopted by the system to respond to an event.

3.6.1 Event Listener

It is an interface which exists in the `View` class. Each of the event listeners consist of a single callback method which are invoked when the view to which the listener is registered generates an event due to user interaction.

Table 3.3 lists the event listener interfaces and their associated callback methods.

Event Listener Interface	Callback Methods	Description
<code>View.OnClickListener</code>	<code>onClick()</code>	Invoked when the user touches or focuses on the item that uses the navigation key or trackball or presses the Enter key
<code>View.OnLongClickListener</code>	<code>onLongClick()</code>	Invoked when the user touches and holds the items or navigates and holds the trackball for a second and the callback method returns a boolean value of true or false
<code>View.OnFocusChangeListener</code>	<code>onFocusChange()</code>	Invoked while the user navigates to or away from the item that uses the trackball or the navigation keys
<code>View.OnKeyListener</code>	<code>onKey()</code>	Invoked when the user is focused on the item and releases/press a key on the device and the callback method returns a Boolean value of true if the event is handled and should stop else it returns false
<code>View.OnTouchListener</code>	<code>onTouch()</code>	Invoked when the user performs any action such as touching, pressing or movement on the screen and the callback method returns a boolean value of true or false
<code>View.OnCreateContextMenuListener</code>	<code>onCreateContextMenu()</code>	Invoked when there is a construction of a context menu

Table 3.3: Event Listener Interfaces and their Associated Callback Methods

3.6.2 Handling User Events

When a user is building a custom component from View, it is possible to define several callback methods as default event handlers. Some of the common callbacks method used for event handling include:

`onKeyDown(int, KeyEvent)` - Invoked when a key has been pressed and was not handled by the views within an Activity.

`onKeyUp(int, KeyEvent)` - Invoked when a key was released and was not handled by the views within an Activity.

`onTrackballEvent(MotionEvent)` - Invoked when a trackball motion event occurs.

`onTouchEvent(MotionEvent)` - Invoked when a touch screen motion event occurs.

`onFocusChanged(boolean, int, Rect)` - Invoked when the view gains or loses focus.

3.7 Working with Listeners

To understand the concept of listener interface an UI is created. Whenever a user interacts with a View object, an event will be generated. This event will be handled and a corresponding message will be displayed.

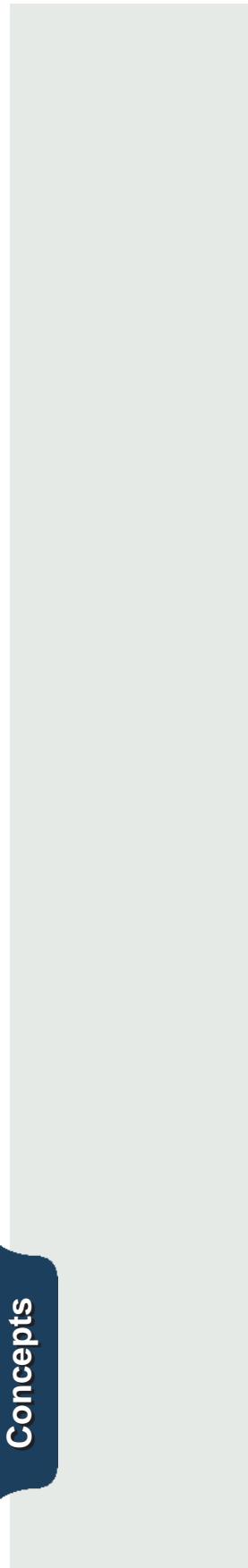
The steps to create a **UIComponent** project are as follows:

1. Start Eclipse IDE.
2. Click **File → New → Project** to display the **New Project** window.
3. Select **Android → Android Application Project**.
4. Click **Next** to display the **New Android Project** screen.
5. Type **UIComponent** as the application name and leave the default settings.
6. Click **Next** and leave the default options selected.
7. Click **Next** to display the **Configure Launcher Icon** pane. Developer can change image if required.
8. Click **Next** and select **BlankActivity**.
9. Click **Next**.
10. Click **Finish**.

11. Navigate to **res → layout** folder to create a new xml file named **activity_linear.xml**.
12. Right-click the folder to display the context menu and select **New → Other → XML File**.
13. Click **Next**.
14. Type **activity_linear.xml** in the **Name** box.
15. Click **Finish**.
16. Double-click and open **activity_linear.xml** file.
17. Modify the code as shown in code snippet 18.

Code Snippet 18:

```
<?xml version="1.0" encoding="utf-8"?>  
  
<LinearLayout xmlns:android="http://schemas.android.com/apk/  
res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:id="@+id/idlienr"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:orientation="vertical"  
    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=".LinearLayout" >  
  
    <EditText  
        android:id="@+id/editText"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:hint="Enter Some Text" />
```



```
        android:ems="10" >

        <requestFocus />
    </EditText>

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:text="Submit" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""
        android:layout_marginTop="10dp"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <Button
        android:id="@+id/showNotification"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:text="Show Notification" />

    <Button
        android:id="@+id/buttonNext"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:text="Next" />
</LinearLayout>
```

18. Modify the code in `activity_main.xml` file as shown in code snippet 19.

Code Snippet 19:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="vertical" >  
  
<ImageView  
    android:id="@+id/imageView"  
    android:layout_width="100dp"  
    android:layout_height="100dp"  
    android:layout_alignParentLeft="true"  
    android:layout_below="@+id/button"  
    android:layout_marginLeft="27dp"  
    android:layout_marginTop="32dp"  
    android:src="@drawable/desert" />  
  
<Spinner  
    android:id="@+id/spinner"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignBottom="@+id/imageView"  
    android:layout_alignParentRight="true"  
    android:layout_toRightOf="@+id/imageView" />  
  
<TextView  
    android:id="@+id/timeView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignLeft="@+id/button"  
    android:layout_below="@+id/button"  
    android:textAppearance="?android:attr/textAppearanceMedium" />  
  
<Button  
    android:id="@+id/button"
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginLeft="50dp"
        android:layout_marginRight="50dp"
        android:layout_marginTop="31dp"
        android:text="Get time" />

<CheckBox
    android:id="@+id/checkBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignRight="@+id/imageView"
    android:layout_below="@+id/imageView"
    android:layout_marginTop="20dp"
    android:text="CheckBox" />

<RatingBar
    android:id="@+id/ratingBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/checkBox"
    android:layout_below="@+id/checkBox"
    android:layout_marginTop="20dp" />

<Button
    android:id="@+id/exit"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignRight="@+id/spinner"
    android:layout_marginBottom="20dp"
    android:text="Back" />

</RelativeLayout>
```

19. Navigate to **res → layout** folder to create a new Android xml file named **activitymaaino.xml**.
20. Modify the code as given in code snippet 20.

Code Snippet 20:

```
<?xml version="1.0" encoding="utf-8"?>  
  
<RelativeLayout xmlns:android="http://schemas.android.com/  
apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" >  
  
    <TextView  
        android:id="@+id/text_picked_time"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_centerHorizontal="true"  
        android:layout_centerVertical="true" />  
  
    <Button  
        android:id="@+id/button_pick_time"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_centerHorizontal="true"  
        android:layout_centerVertical="true"  
        android:layout_below="@+id/text_picked_time"  
        android:text="Click for Time" />  
  
    <TextView  
        android:id="@+id/textView1"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_alignParentTop="true"  
        android:layout_alignRight="@+id/text_picked_time"  
        android:layout_marginTop="72dp"  
        android:text="Time Picker Example" />  
  
</RelativeLayout>
```

21. Create a new xml named “menu” inside the **/res/menu** directory by right-clicking the **/res/menu** folder to display the context menu.
22. Selecting **New → Other → XML File**.
23. Type **menu.xml** in the **Name** box.
24. Click **Finish**.
25. Modify the code in **menu.xml** file as shown in code snippet 21.

Code Snippet 21:

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

    <item
        android:id="@+id/first"
        android:title="Notification"/>

    <item
        android:id="@+id/second"
        android:title="Next"/>
</menu>
```

26. Navigate to **src → com.example.uicomponents → MainActivity.java file**.
27. Double-click to open the file.
28. Modify the code as shown in code snippet 22.

Code Snippet 22:

```
package com.example.uicomponents;

import android.os.Bundle;
import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
```

```
import android.content.Intent;
import android.graphics.Color;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.view.View.OnClickListener;
public class MainActivity extends Activity implements OnClickListener {
    EditText textField;
    Button submit;
    Button next;
    TextView result;
    String text;
    Button showNotification;
    public LinearLayout activity;
    long when;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_linear);
        textField = (EditText) findViewById(R.id.editText);
        submit = (Button) findViewById(R.id.button);
        next = (Button) findViewById(R.id.buttonNext);
        result = (TextView) findViewById(R.id.textView);
        showNotification = (Button) findViewById(R.
                id.showNotification);
        when = System.currentTimeMillis();
    }
}
```

```
submit.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View arg0) {  
        // TODO Auto-generated method stub  
        text = textField.getText().toString();  
        result.setText("The Text Entered is : " + text);  
    }  
});  
  
next.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View arg0) {  
        // TODO Auto-generated method stub  
        Intent nextIntent = new Intent(MainActivity.  
                this, RelativeLayout.class);  
        startActivity(nextIntent);  
    }  
});  
  
showNotification.setOnClickListener(new  
OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        // TODO Auto-generated method stub  
        showNotification();  
    }  
});  
  
}  
  
public boolean onCreateOptionsMenu(Menu menu) {  
    super.onCreateOptionsMenu(menu);  
    MenuInflater inflater = getMenuInflater();  
    inflater.inflate(R.menu.menu, menu);  
    return true;  
}
```

```
public boolean onOptionsItemSelected(MenuItem item)
{
    switch (item.getItemId())
    {
        case R.id.first:
            showNotification();
            return true;
        case R.id.second:
            // startActivity(new Intent(this, RelativeLayout.
            // class));
            return true;
    }
    return false;
}

@SuppressWarnings("deprecation")
public void showNotification()
{
    NotificationManager notificationManager
    =(NotificationManager)MainActivity.this.
    getSystemService(Context.NOTIFICATION_SERVICE);

    Intent intent = new Intent(this, MainActivity.class);
    intent.setFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
    PendingIntent contentIntent = PendingIntent.
    getActivity(this, 0, intent, 0);

    Notification notification = new Notification(R.drawable.
    ic_launcher, "New Message", when);

    notification.ledARGB = Color.RED;
    notification.ledOffMS = 300;
    notification.ledOnMS = 300;
    notification.defaults |= Notification.DEFAULT_SOUND;
    notification.defaults |= Notification.DEFAULT_LIGHTS;
```

```

        notification.flags |= Notification.FLAG_AUTO_CANCEL |
        Notification.FLAG_SHOW_LIGHTS;

        notification.setLatestEventInfo(this,
            "Notification", "This is StatusBarNotification",
            contentIntent);

        notificationManager.notify(10001, notification);
    }

    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
    }
}

```

29. Create a new Java class named **RelativeLayout** inside the **/src/com.example.uicomponents** directory by right-clicking the **/com.example.uicomponents** folder to display the context menu.
30. Select **New → Class**.
31. Type **RelativeLayout** as the Java class in the **Name** box.
32. Modify the code in **RelativeLayout.java** file as shown in code snippet 23.

Code Snippet 23:

```

package com.example.uicomponents;

import java.util.ArrayList;
import java.util.Calendar;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.app.TimePickerDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;

```

```
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.ImageView;
import android.widget.RatingBar;
import android.widget.RatingBar.OnRatingBarChangeListener;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.TimePicker;
import android.widget.Toast;

public class RelativeLayout extends Activity {

    Button getTime;
    ImageView imageView;
    Spinner spinner;
    TextView timeView;
    CheckBox checkBox;
    RatingBar ratingBar;
    Button back;

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

    static final int TIME_DIALOG_ID = 999;
    int hour;
    int minute;
    ArrayAdapter<String> dataAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
```

```

super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

getTime = (Button) findViewById(R.id.button);
imageView = (ImageView) findViewById(R.id.imageView);
spinner = (Spinner) findViewById(R.id.spinner);
timeView = (TextView) findViewById(R.id.timeView);
checkBox = (CheckBox) findViewById(R.id.checkBox);
ratingBar = (RatingBar) findViewById(R.id.ratingBar);
back = (Button) findViewById(R.id.exit);

final Calendar c=Calendar.getInstance();
hour=c.get(Calendar.HOUR_OF_DAY);
minute=c.get(Calendar.MINUTE);

list.add("Bangalore");
list.add("Chennai");
list.add("Hyderabad");
list.add("Delhi");

dataAdapter = new ArrayAdapter<String>(RelativeLayout.
    this, android.R.layout.simple_list_item_1, list);

spinner.setAdapter(dataAdapter);

getTime.setOnClickListener(new OnClickListener() {

    @SuppressWarnings("deprecation")
    @Override
    public void onClick(View arg0) {
        // TODO Auto-generated method stub
        showDialog(TIME_DIALOG_ID);

    }
});

spinner.setOnItemSelectedListener(new
OnItemSelectedListener() {

    @Override
    public void onItemSelected(AdapterView<?> arg0, View
        arg1, int arg2, long arg3) {
        // TODO Auto-generated method stub
        Toast.makeText(RelativeLayout.this,

```

```
        "SelectedCity : " + dataAdapter.getItem(arg2) , Toast.LENGTH_LONG).show();
    }

    @Override
    public void onNothingSelected(AdapterView<?> arg0) {
        // TODO Auto-generated method stub
    }
}) ;

checkBox.setOnCheckedChangeListener(new
    OnCheckedChangeListener() {

    @Override
    public void onCheckedChanged(CompoundButton
        buttonView boolean isChecked) {
        // TODO Auto-generated method stub

        if (isChecked) {
            Toast.makeText(RelativeLayout.this, "You
                checked" , Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(RelativeLayout.this, "You
                unchecked" , Toast.LENGTH_LONG).show();
        }
    }
}) ;

ratingBar.setOnRatingBarChangeListener(new
    OnRatingBarChangeListener() {

    @Override
    public void onRatingChanged(RatingBar ratingBar, float
        rating, boolean fromUser) {
        // TODO Auto-generated method stub
    }
}) ;
```

```
        Toast.makeText(RelativeLayout.this, "You rated : " +  
        rating,  
        Toast.LENGTH_LONG).show();  
    }  
});  
back.setOnClickListener(new OnClickListener() {  
  
    @Override  
    public void onClick(View v) {  
        // TODO Auto-generated method stub  
        AlertDialog.Builder alert_dialog = new AlertDialog.  
            Builder(RelativeLayout.this);  
        alert_dialog.setTitle("Confirmation...");  
        alert_dialog.setNegativeButton("Cancel",  
            new DialogInterface.OnClickListener()  
        {  
            @Override  
            public void onClick(DialogInterface dialog,  
                int which) {  
            }  
        });  
        alert_dialog.setPositiveButton("Leave current",  
            new DialogInterface.OnClickListener() {  
                @Override  
                public void onClick(DialogInterface dialog,  
                    int which) {  
                    finish();  
                }  
            });  
        alert_dialog.show();  
    }  
});  
}  
  
@Override
```

```

protected Dialog onCreateDialog(int id) {
    switch (id) {
        case TIME_DIALOG_ID:
            // set time picker as current time
            return new TimePickerDialog(this,
                timePickerListener, hour, minute, false);
    }
    return null;
}

private TimePickerDialog.OnTimeSetListener
timePickerListener = new TimePickerDialog.
OnTimeSetListener() {
    public void onTimeSet(TimePicker view, int
        selectedHour, int selectedMinute) {
        hour = selectedHour;
        minute = selectedMinute;
        // Displaying time using toast
        timeView.setText("Time is : "
            + new
            StringBuilder().append(hour).append(":")
            .append(minute));
    }
};
}
}

```

33. Create a new Java class named **TimePickerFragment** inside the **/src/com.example.uicomponents** directory by right-clicking the **/com.example.uicomponents** folder to display the context menu.
34. Select **New → Class**.
35. Type **TimePickerFragment** as the Java class in the **Name** box.
36. Modify the code in **TimePickerFragment.java** file as shown in code snippet 24.

Code Snippet 24:

```
package com.example.uicomponents;

import android.app.Activity;
import android.app.Dialog;
import android.app.TimePickerDialog;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;
import android.text.format.DateFormat;
import android.widget.TimePicker;

import java.util.Calendar;

public class TimePickerFragment extends DialogFragment
    implements
        TimePickerDialog.OnTimeSetListener {
    private TimePickedListener mListener;

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // use the current time as the default values for the picker
        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);

        // create a new instance of TimePickerDialog and return it
        return new TimePickerDialog(getActivity(), this, hour,
            minute, DateFormat.is24HourFormat(getActivity()));
    }

    @Override
    public void onAttach(Activity activity) {
        // when the fragment is initially shown (i.e. attached to the
        // activity),
        // cast the activity to the callback interface type
        super.onAttach(activity);
        try {
            mListener = (TimePickedListener) activity;
```

```

    } catch (ClassCastException e) {
        throw new ClassCastException(activity.toString()
            + " must implement " +
            TimePickedListener.class.getName());
    }
}

public void onTimeSet(TimePicker view, int hourOfDay, int
minute) {
    // when the time is selected, send it to the activity via its
    // callback
    // interface method
    Calendar c = Calendar.getInstance();
    c.set(Calendar.HOUR_OF_DAY, hourOfDay);
    c.set(Calendar.MINUTE, minute);
    mListener.onTimePicked(c);
}

public static interface TimePickedListener {
    public void onTimePicked(Calendar time);
}
}

```

37. Create a new Java class named **Timepicker** inside the **/src/com.example.uicomponents** directory by right-clicking the **/com.example.uicomponents** folder to display the context menu.
38. Select **New → Class**.
39. Type **Timepicker** as the Java class in the **Name** box.
40. Modify the code in **Timepicker.java** file as shown in code snippet 25.

Code Snippet 25:

```

package com.example.uicomponents;

import android.os.Bundle;
import android.support.v4.app.DialogFragment;
import android.support.v4.app.FragmentActivity;

```

```
import android.text.format.DateFormat;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

import java.util.Calendar;
import com.example.uicomponents.TimePickerFragment.TimePickedListener;

public class Timepicker extends FragmentActivity implements TimePickedListener
{
    private TextView mPickedTimeText;
    private Button mPickTimeButton;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maaino);
        mPickedTimeText = (TextView) findViewById(R.id.text_picked_time);
        mPickTimeButton = (Button) findViewById(R.id.button_pick_time);
        mPickTimeButton.setOnClickListener(new OnClickListener()
        {
            public void onClick(View v)
            {
                // show the time picker dialog
                DialogFragment newFragment = new TimePickerFragment();
                newFragment.show(getSupportFragmentManager(), "timePicker");
            }
        });
    }

    public void onTimePicked(Calendar time)
    {
```

```

        // display the selected time in the TextView
        mPickedTimeText.setText(DateFormat.format("h:mma",
        time));
    }
}

```

41. Navigate to **res → drawable-mdpi** folder to copy an image named **desert.jpg**.

Note - The image name should be in lowercase and without space.

42. Navigate to **AndroidManifest.xml** file.

43. Add the following code as shown in code snippet 26.

Code Snippet 26:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/
    android"
        package="com.example.uicomponents"
        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.uicomponents.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>  
  
<activity android:name="com.example.uicomponents.  
RelativeLayout" >  
</activity>  
  
<activity android:name="com.example.uicomponents.  
Timepicker" >  
</activity>  
  
</application>  
  
</manifest>
```

Once you execute the application, the output will be as shown in figure 3.20.

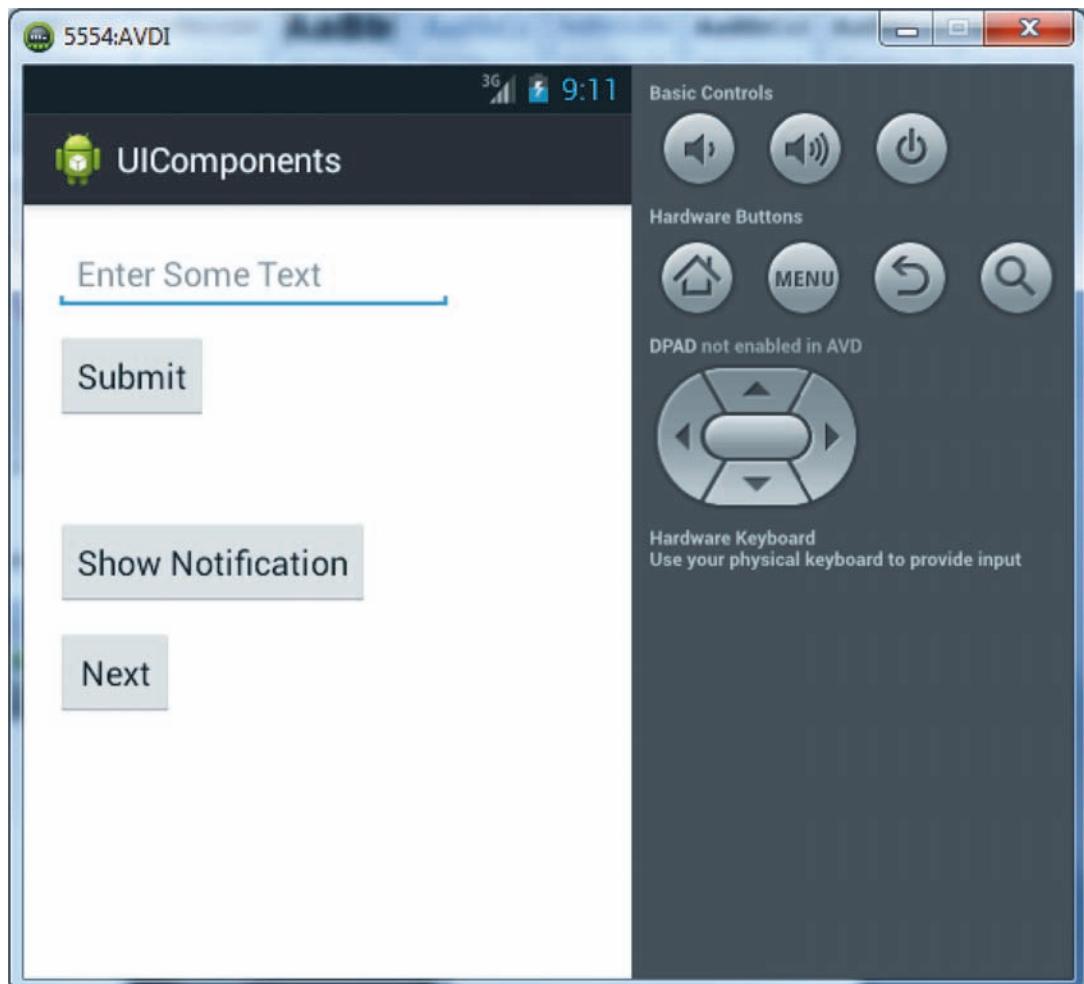


Figure 3.20: Output of the Application

Type **John Daniel** in the text box to display the output as shown in figure 3.21.

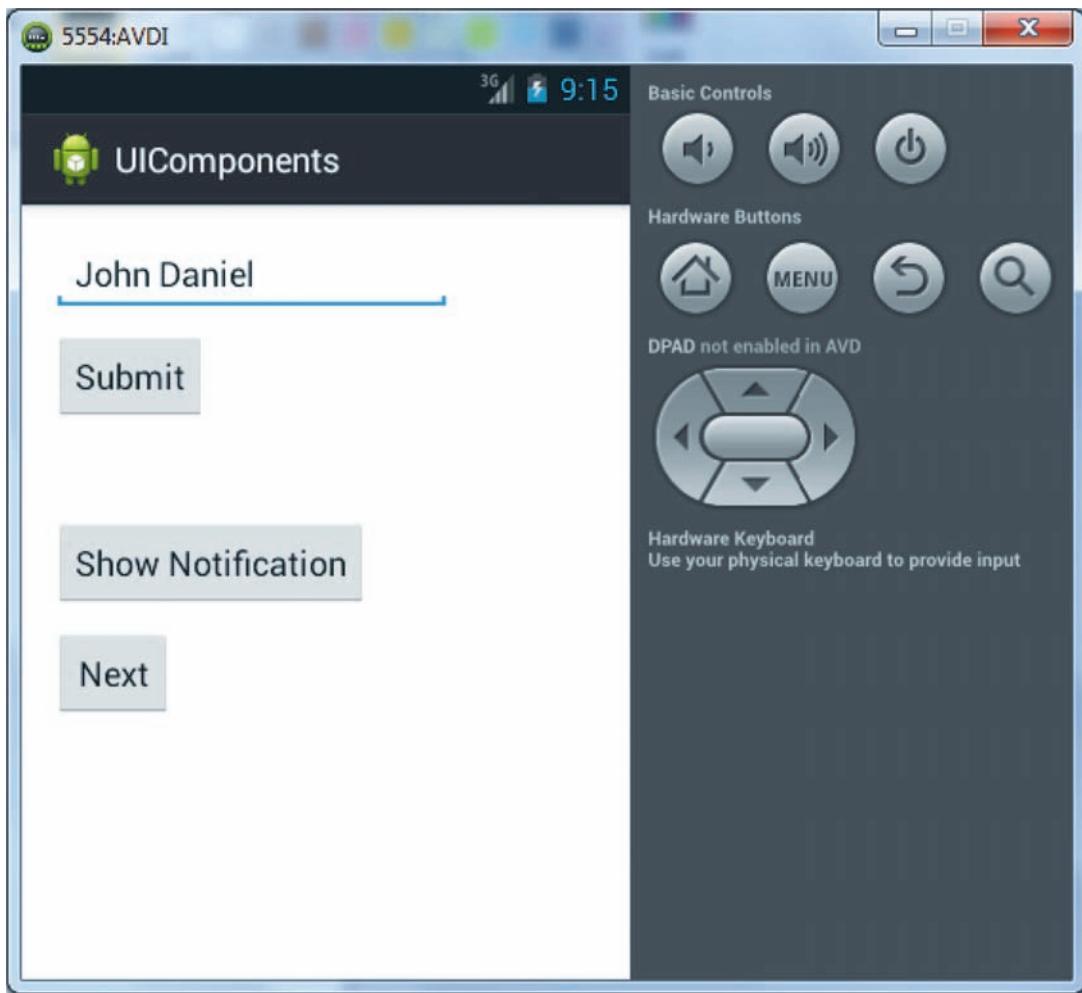


Figure 3.21: Name in Text Box

Once you click **Submit**, you will obtain the output as shown in figure 3.22.

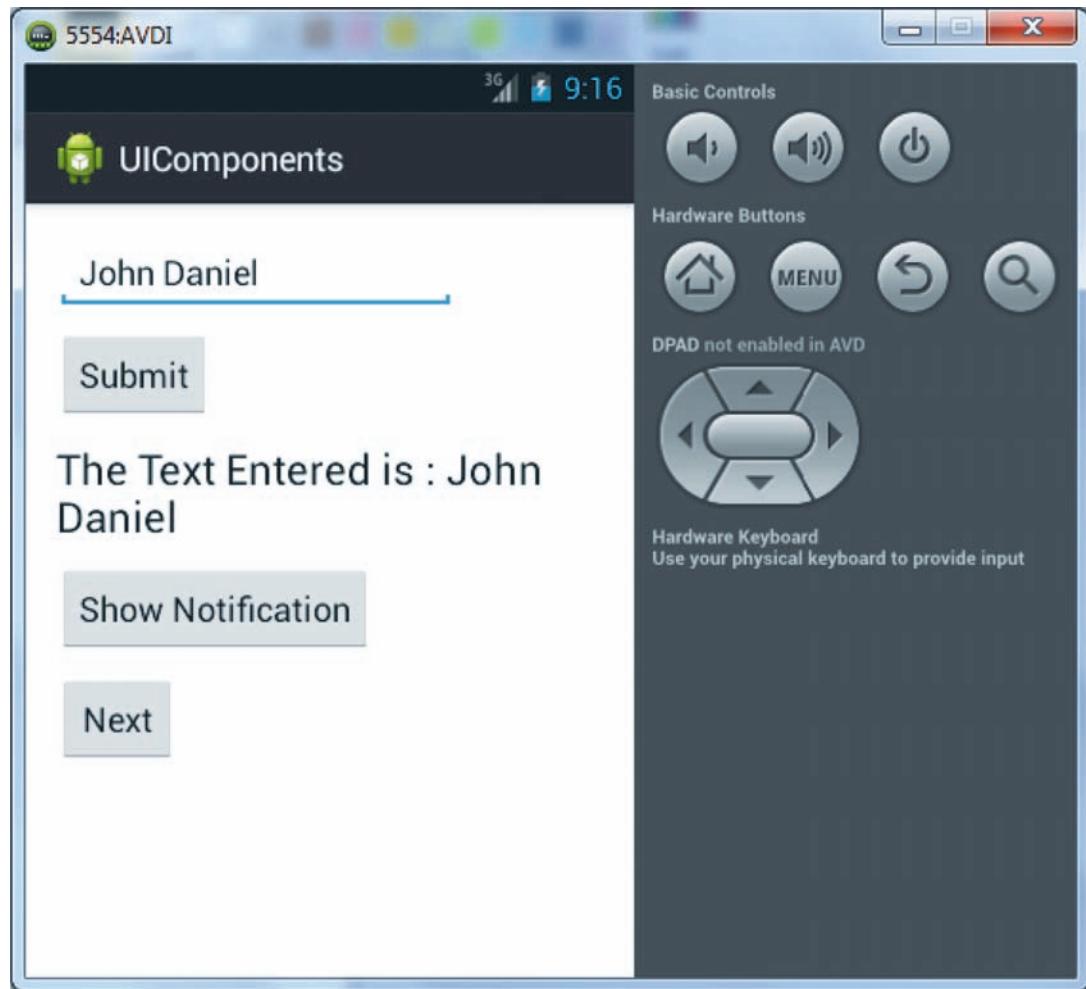


Figure 3.22: Message Display

Clicking **Next** will display the output as shown in figure 3.23.

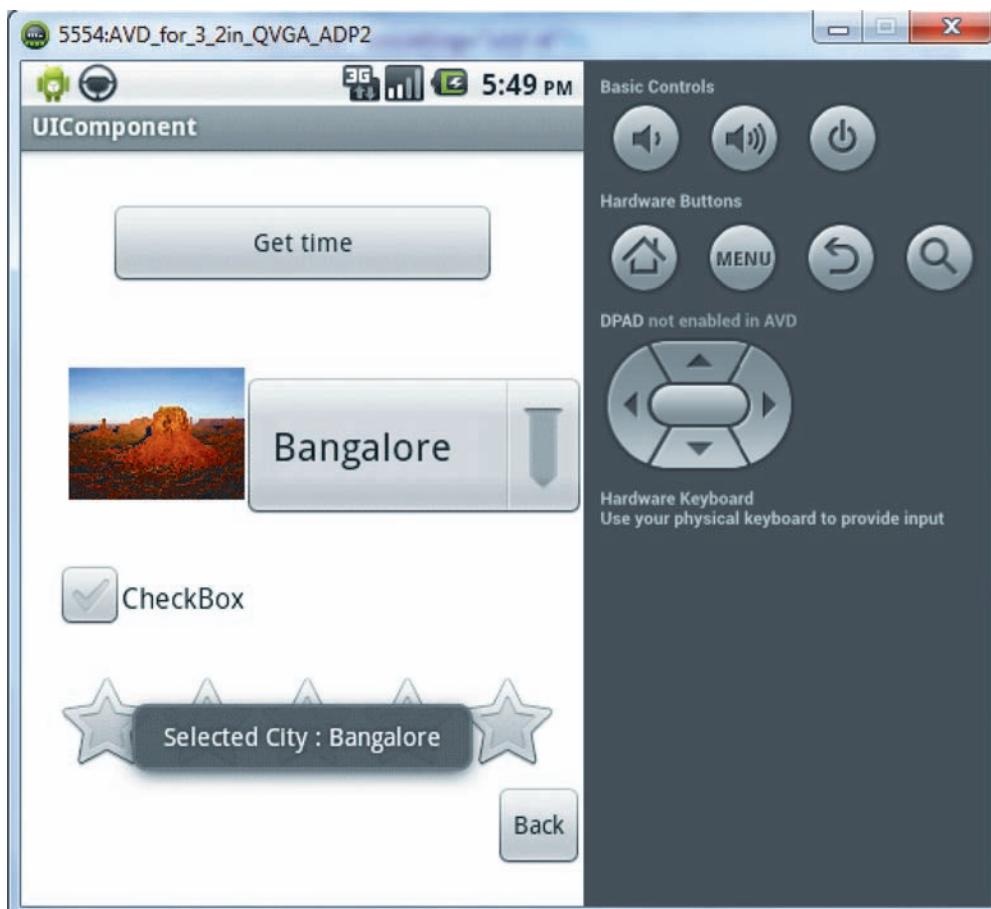


Figure 3.23: UI Component

Note - Selecting any of the UI components will result in the display of a corresponding Toast message.

3.8 Check Your Progress

1. Each layout file in Android can contain a maximum of how many root elements?

(A)	1	(C)	2
(B)	3	(D)	4

2. Which of the following Graphic UI component allows user to change a setting between two states?

(A)	Picker	(C)	Toggle
(B)	Spinner	(D)	Seek bar

3. Which of the following Graphic UI allows user to select an option from a drop-down list?

(A)	Toggle	(C)	Seek bar
(B)	Radio button	(D)	Spinner

4. Which of the following UI component provides feedback about a user's action in a popup window?

(A)	Notification	(C)	Dialog
(B)	Layout	(D)	Toast

5. What is the collection of properties that specify look and format for a View window called?

(A)	Event	(C)	Theme
(B)	Style	(D)	Widget

6. A user touching a screen in a smartphone is an example of _____.

(A)	Input event	(C)	Notification
(B)	Linear layout	(D)	Intent

7. Which UI component displays popup window and allows user to take decision or enter information?

(A)	Toast	(C)	Notification
(B)	Dialog	(D)	Menu

3.8.1 Answers

1.	A
2.	C
3.	D
4.	D
5.	B
6.	A
7.	B



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



Samantha who is new to Android programming wants to create an app which will enable her to incorporate the following functionalities:

1. Create a linear layout with text view and buttons.
2. Create a relative layout which will display a text field, two spinners, and a button as shown in figure 3.24.

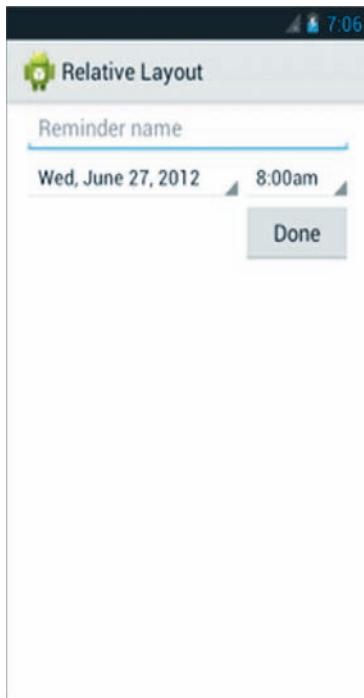
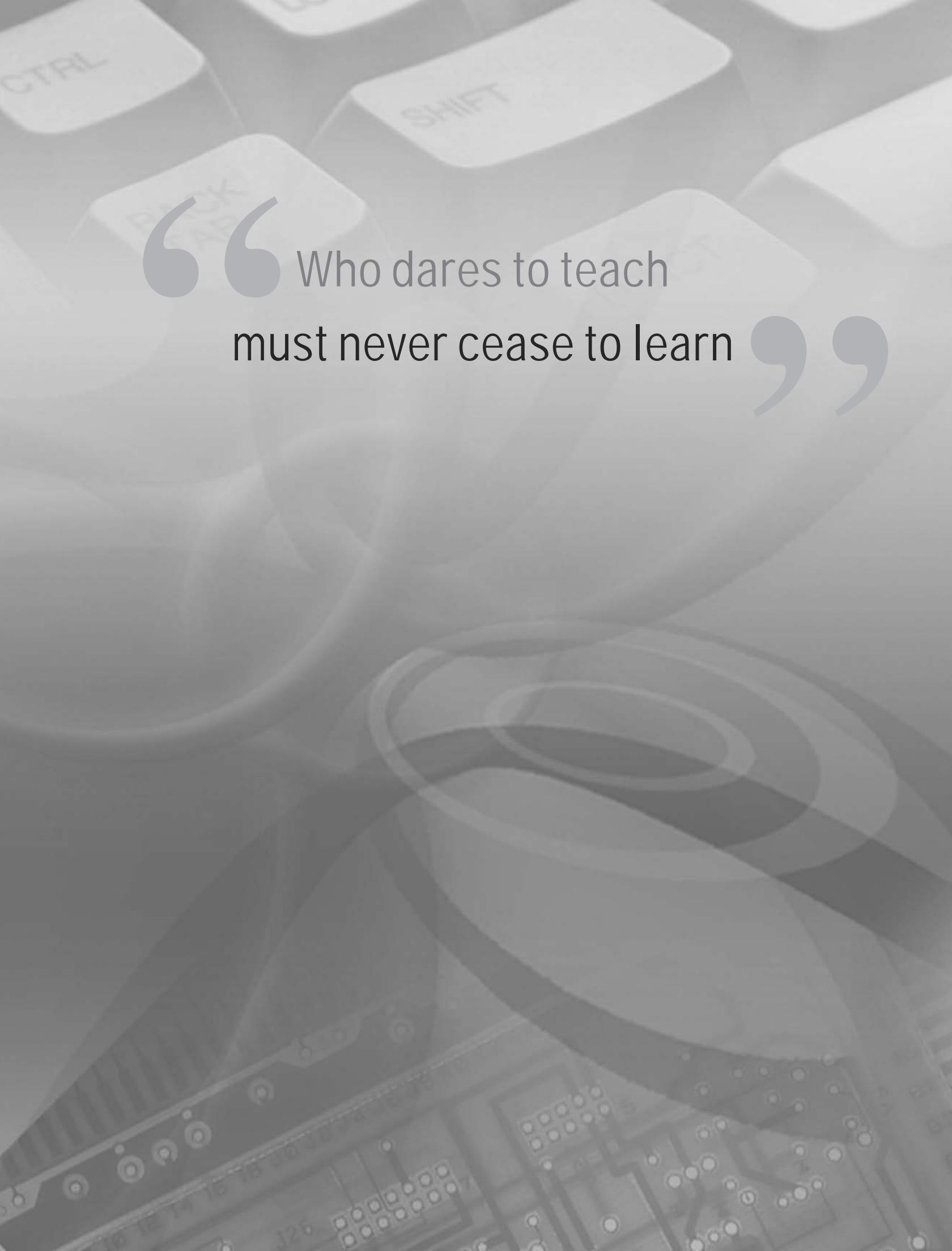


Figure 3.24: UI View

3. Create a layout that will have two radio button objects.
4. For each of the View object there will be an event listeners and event handlers attached for handling user events and displaying a Toast message.

“ Who dares to teach
must never cease to learn ”



JELLYBEAN
ICE CREAM SANDWICH

Session - 4

Android System Overview

Welcome to the Session, **Android System Overview**.

This session provides an overview of the Android security system, preferences, file system, and notification. These are important aspects which the developer has to consider for developing an android application.

In this session, you will learn to:

- ➔ Explain Preferences
- ➔ Explain Shared Preferences Storage Structure
- ➔ Explain the Android File System
- ➔ Explain Notifications
- ➔ Explain the Android Security Model



4.1 Introduction

An Android developer needs to be aware of setting preferences, saving files, sending different types of notifications, and the Android security model. The developer while developing applications also has to provide the Android user with options to customize their applications. This can be done by setting preferences.

While building different types of applications, there might be a requirement to read and write a large amount of data to and from files. Thus, the concept of file system is also explained in this session in detail.

The session explains the different types of notifications and the ways to display them. Finally, to implement security in the application, the session explains the Android security model which helps to comprehend concepts, such as, user permissions, password protection, and application security.

4.2 Android System Overview

There are several options in android to save persistent application data. The solution you choose depend upon the requirement of the project such as, whether data should be private to your application or accessible to other application also. Following are the options which can be used for storing the data.

- **Shared Preferences:** Store data as key-value pairs.
- **Internal Storage:** Store data on the device memory. Data will not be accessed by other application.
- **External Storage:** Store public data on the shared external storage. Data will be accessed by other application.
- **SQLite Databases:** Store structured data in a private database.
- **Network Connection:** Store data on the Web with your own network server.

Shared preferences (`android.content.SharedPreferences`) can be used to easily store application `data.android.preference` package provide classes to manage and implement UI preferences. This ensures that the user experience with applications remains consistent. The Preference subclass can be used to determine settings that enable users to customize their applications. There are two ways in which a developer can specify preferences. The first way is to create an XML file. The developer can also specify preferences directly in the code. Preference object is used as the building block for a single setting. Each of the preference settings is saved as a key-value pair in the default `SharedPreferences` file for your app's settings.

The three most commonly used Preferences are as follows:

- **CheckBoxPreference:** CheckBoxPreference is used to store data in preferences using CheckBox widget. In other words, it provides the checkbox widget functionality. It returns a true or false value as shown in figure 4.1.

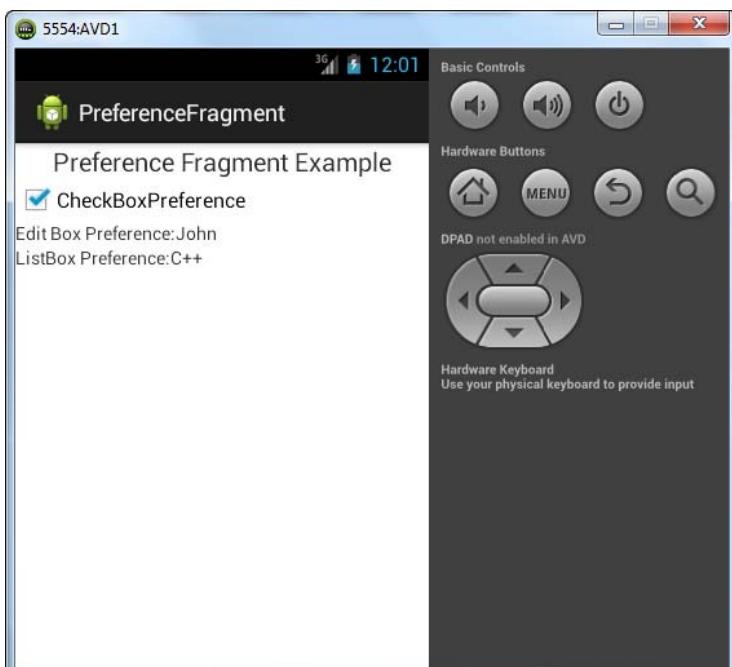


Figure 4.1: CheckBoxPreference

- **ListPreference:** ListPreference is used to display the list of entries as a dialog from which a user can choose a single selection. This displays a group of radio buttons as shown in figure 4.2 among which only one button can be selected.

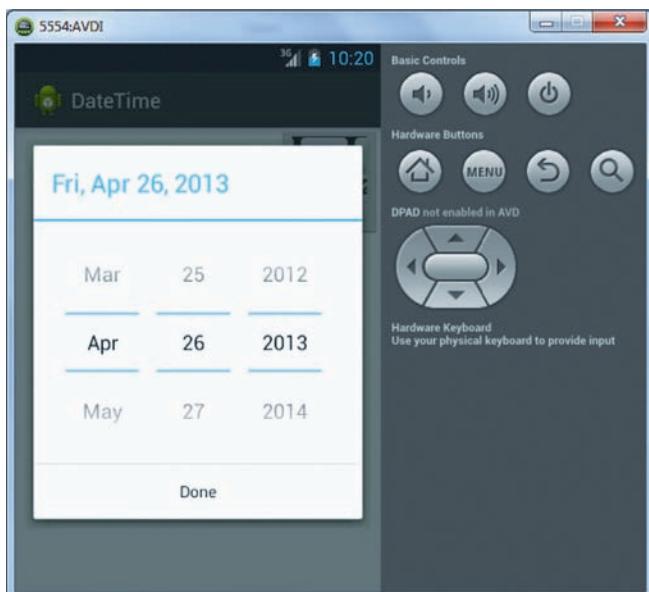


Figure 4.2: ListPreference

- **EditTextPreference:** This opens up the dialog box where user can enter a value as shown in figure 4.3. When text is typed into the field, it returns a string value.

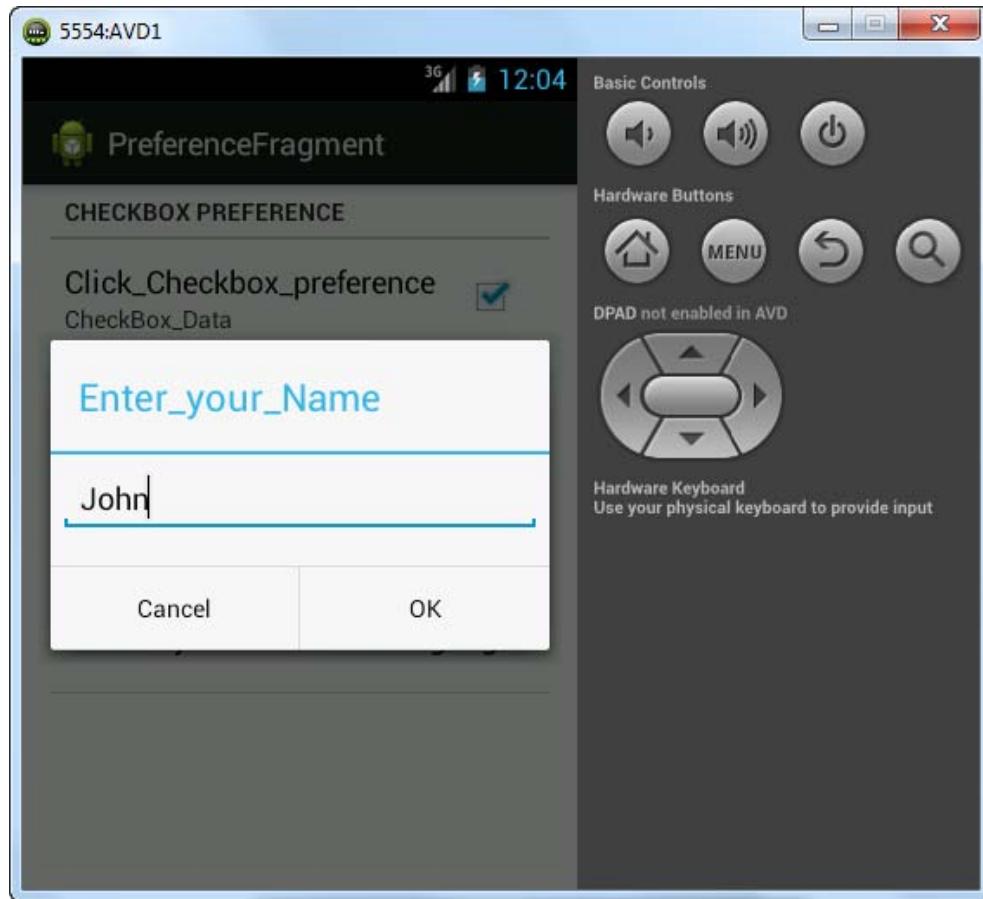


Figure 4.3: EditTextPreference

In the following example, an application is created where all the different types of preferences are integrated.

The steps to create the application are as follows:

1. Start **Eclipse IDE**.
2. Create a project named **PreferenceFragment**.
3. Create a folder named **xml** under **/res** folder.
4. Create an xml file, to define the preference under **/res/xml/namedpreferences.xml**.
5. Add the code to **preferences.xml** file as shown in code snippet 1.

Code Snippet 1:

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

<PreferenceScreen xmlns:android="http://schemas.android.com/apk/
res/android" >

    <PreferenceCategory android:title="Checkbox Preference" >
        <CheckBoxPreference
            android:key="checkbox_preferencevalue"
            android:summary="CheckBox_Data"
            android:title="Click_Checkbox_preference" />
    </PreferenceCategory>
    <PreferenceCategory android:title="EditTextPreference" >
        <EditTextPreference
            android:dialogTitle="Enter_your_Name"
            android:key="edittext_preferencevalue"
            android:summary="summary_edittext_preference"
            android:title="title_edittext_preference" />
    </PreferenceCategory>
    <PreferenceCategory android:title="ListViewPreference" >
        <ListPreference
            android:entries="@array/array_lp_entries"
            android:entryValues="@array/array_lp_entry_values"
            android:key="lp_android_choice"
            android:title="@string/str_lp_android_choice_title" />
    </PreferenceCategory>
</PreferenceScreen>
```

6. Create an xml file to define the menu in the **res → menu → main.xml** file.
7. Modify the code as shown in code snippet 2 in **main.xml** file.

Code Snippet 2:

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

</menu>
```

8. Navigate to **res → values → strings.xml** file.
9. Modify the code in **strings.xml** file as shown in code snippet 3.

Code Snippet 3:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">PreferenceFragment</string>
    <string name="action_settings">Preference Example</string>
    <string name="hello_world">Hello world!</string>
    <string name="str_lp_android_choice_title">Choose your
Favorite Language</string>

    <string-array name="array_lp_entries">
        <item>Java</item>
        <item>C</item>
        <item>C++</item>
        <item>Fortan</item>
        <item>Perl</item>
    </string-array>
    <string-array name="array_lp_entry_values">
        <item>Java</item>
        <item>C</item>
        <item>C++</item>
        <item>Fortan</item>
    </string-array>
```

```

<item>Perl</item>
</string-array>
</resources>

```

10. Create a Java class named **PrefsFragment.java** under the **src → com.example.preferencefragment** folder.
11. Add the code to **PrefsFragment.java** file as shown in code snippet 4.

Code Snippet 4:

```

package com.example.PreferenceFragment;

import android.app.Fragment;
import android.os.Bundle;
import android.preference.PreferenceFragment;
public class PrefsFragment extends PreferenceFragment {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);

        // Load the preferences from an XML resource
        addPreferencesFromResource(R.xml.preferences);
    }
}

```

12. Create a Java class named **SetPreferenceActivity** under **src → com.example.preferencefragment** folder.
13. Add the code to **SetPreferenceActivity.java** file as shown in code snippet 5.

Code Snippet 5:

```

package com.example.PreferenceFragment;
import android.app.Activity;
import android.os.Bundle;
public class SetPreferenceActivity extends Activity{

```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    // TODO Auto-generated method stub
    super.onCreate(savedInstanceState);

    getFragmentManager().beginTransaction()
        .replace(android.R.id.content, new PrefsFragment()).commit();
}

```

14. Navigate to **src → com.example.preferencefragment → MainActivity.java** file.
15. Add the code to **MainActivity.java** file as shown in code snippet 6.

Code Snippet 6:

```

package com.example.PreferenceFragment;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.CheckBox;
import android.widget.TextView;

public class MainActivity extends Activity {

    CheckBox prefCheckBoxvalue;
    TextViewprefEditTextvalue, prefEditTextvalueone;

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

```

```
    prefCheckBoxvalue = (CheckBox) findViewById(R.  
id.prefCheckBoxvalue);  
  
    prefEditTextvalue = (EditText) findViewById(R.  
id.prefEditTextvalue);  
  
    prefEditTextvalueone=(EditText) findViewById(R.  
id.prefEditTextvalueone);  
  
    loadPrefvalue();  
}  
  
@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;  
}  
  
@Override  
  
public boolean onOptionsItemSelected(MenuItem item) {  
  
    Intent intent = new Intent();  
  
    intent.setClass(MainActivity.this,  
SetPreferenceActivity.class);  
  
    startActivityForResult(intent, 0);  
  
    return true;  
}  
  
@Override  
  
protected void onActivityResult(int requestCode, int  
resultCode, Intent data) {  
  
    loadPrefvalue();  
}  
  
private void loadPrefvalue(){  
  
    SharedPreferences mySharedPreferences =  
PreferenceManager.getDefaultSharedPreferences(this);
```

```

        boolean my_checkbox_preference = mySharedPreferences.
        getBoolean("checkbox_preferencevalue", false);

        prefCheckBoxvalue.setChecked(my_checkbox_preference);

        String my_edittext_preference = mySharedPreferences.
        getString("edittext_preferencevalue", "");

        prefEditTextvalue.setText("Edit Box Preference:" +my_
        edittext_preference);

        String my_edittext_preference1 = mySharedPreferences.
        getString("lp_android_choice", "");

        prefEditTextvalueone.setText("ListBox Preference:" +my_
        edittext_preference1);

    }

}

```

16. Navigate to **AndroidManifest.xml** file.
17. Add the code to **AndroidManifest.xml** file as shown in code snippet 7.

Code Snippet 7:

```

<activity android:name="com.example.preferencefragment.
SetPreferenceActivity" >
</activity>

```

AndroidManifest.xml file will contain the code as shown in code snippet 8.

Code Snippet 8:

```

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

    package="com.example.PreferenceFragment"
    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.PreferenceFragment.  
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>  
    <activity  
        android:name=".SetPreferenceActivity" >  
    </activity>  
    </application>  
</manifest>
```

Once you execute the application you will get the output as shown in figure 4.4.

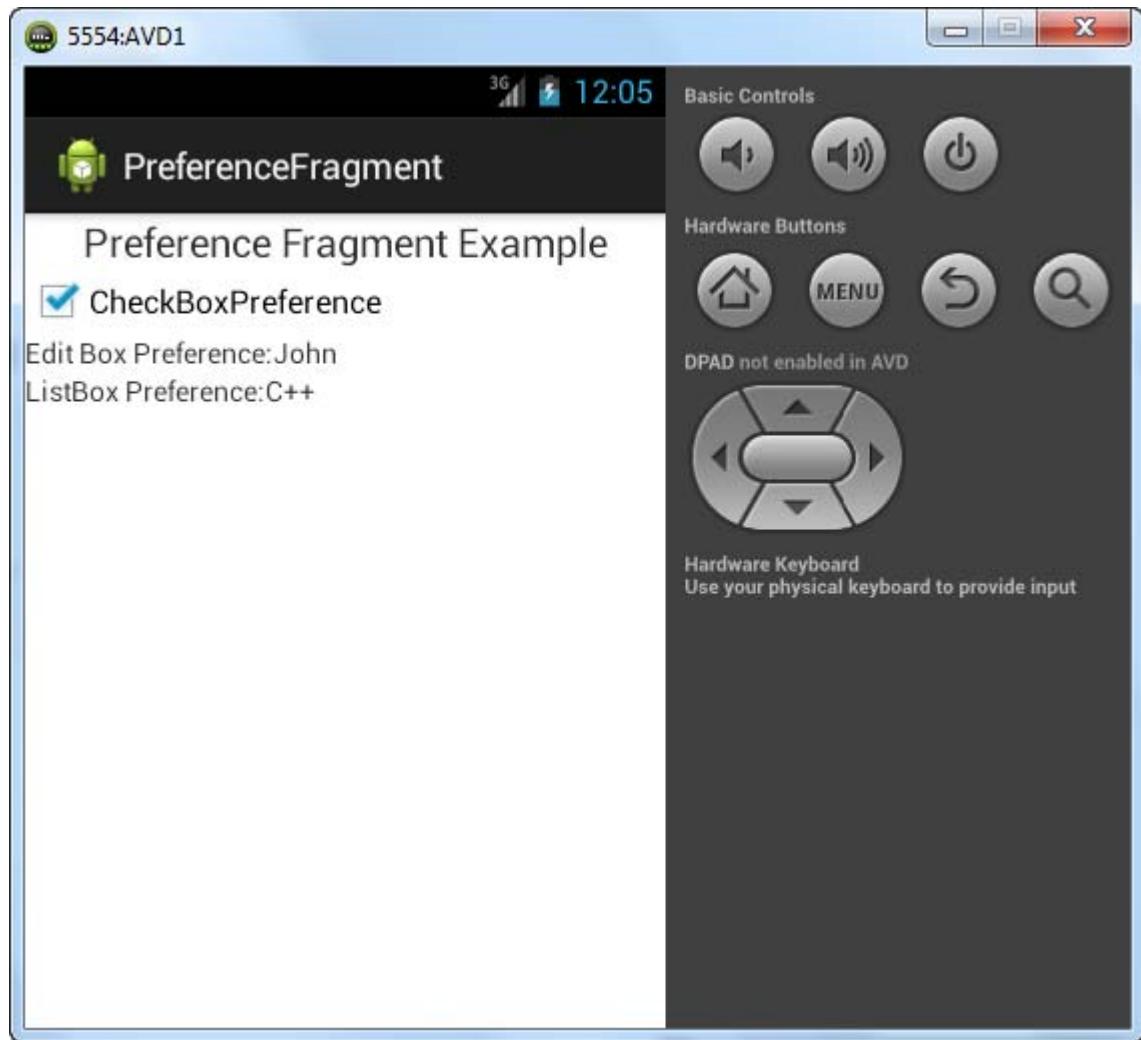


Figure 4.4: Preference Fragment

Click **Menu** to display the output as shown in figure 4.5.

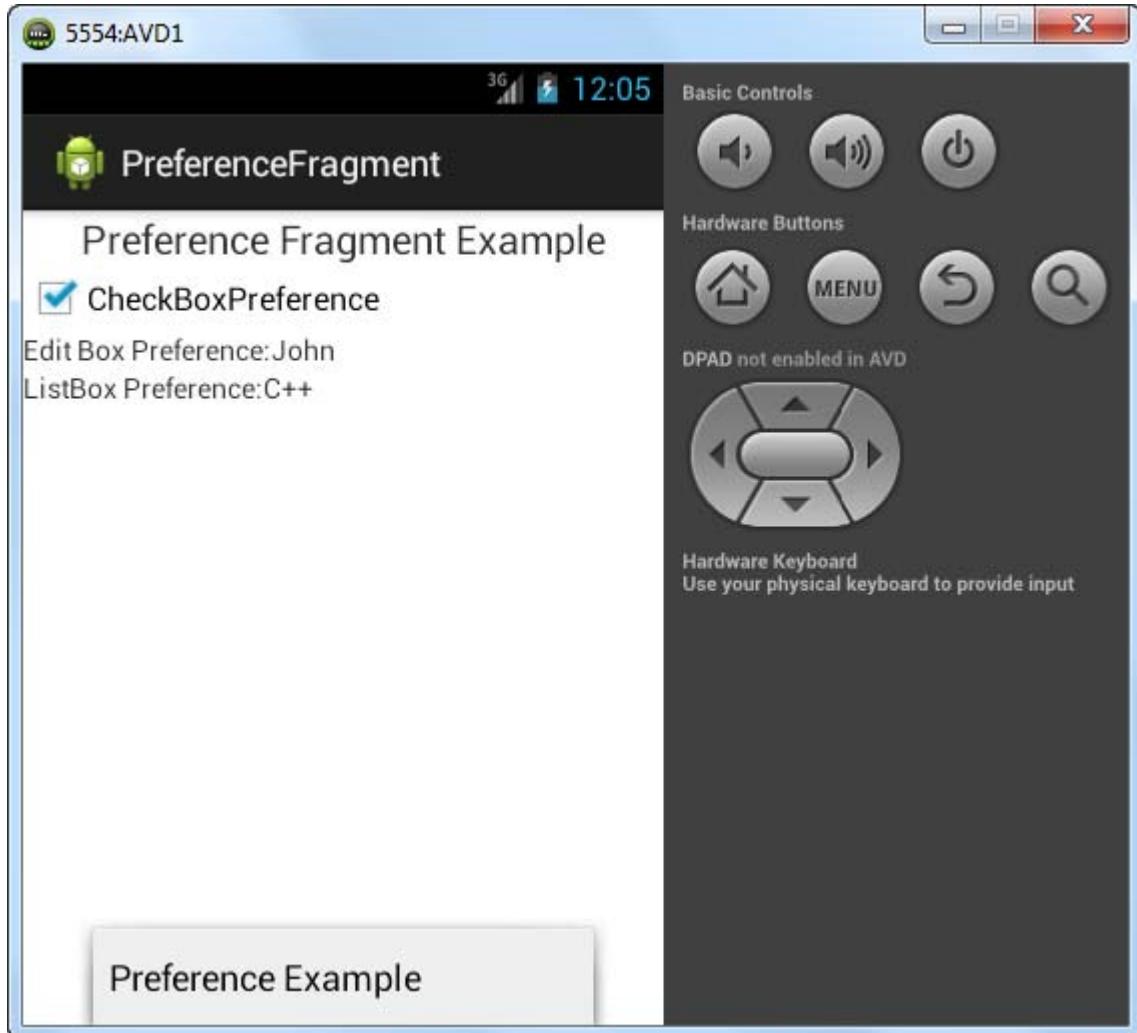


Figure 4.5: Different Types of Preferences

Click Preference Example to display the output as shown in figure 4.6.

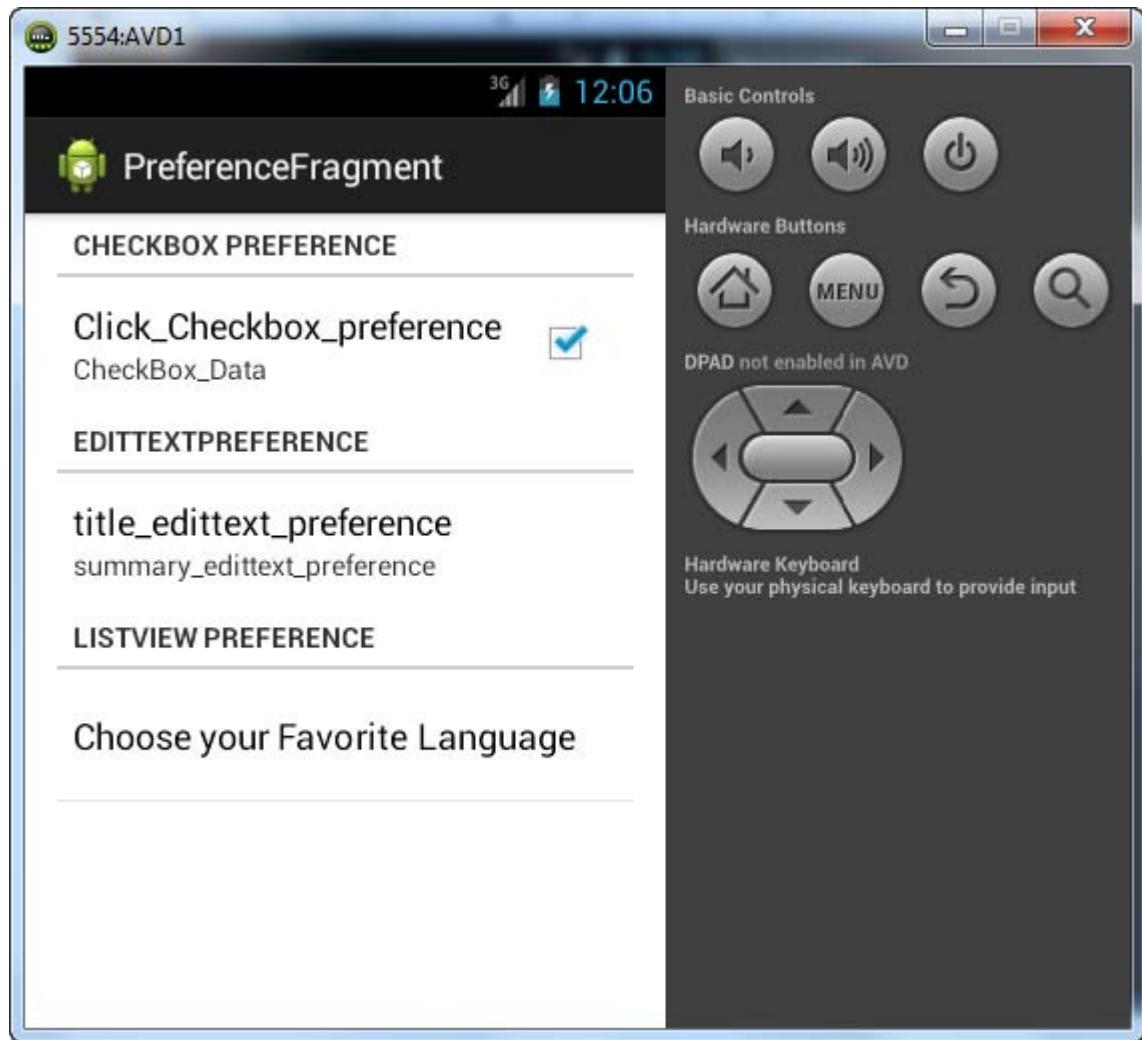


Figure 4.6: Preference Options

Click **Choose your Favorite Language** to display the output as shown in figure 4.7.

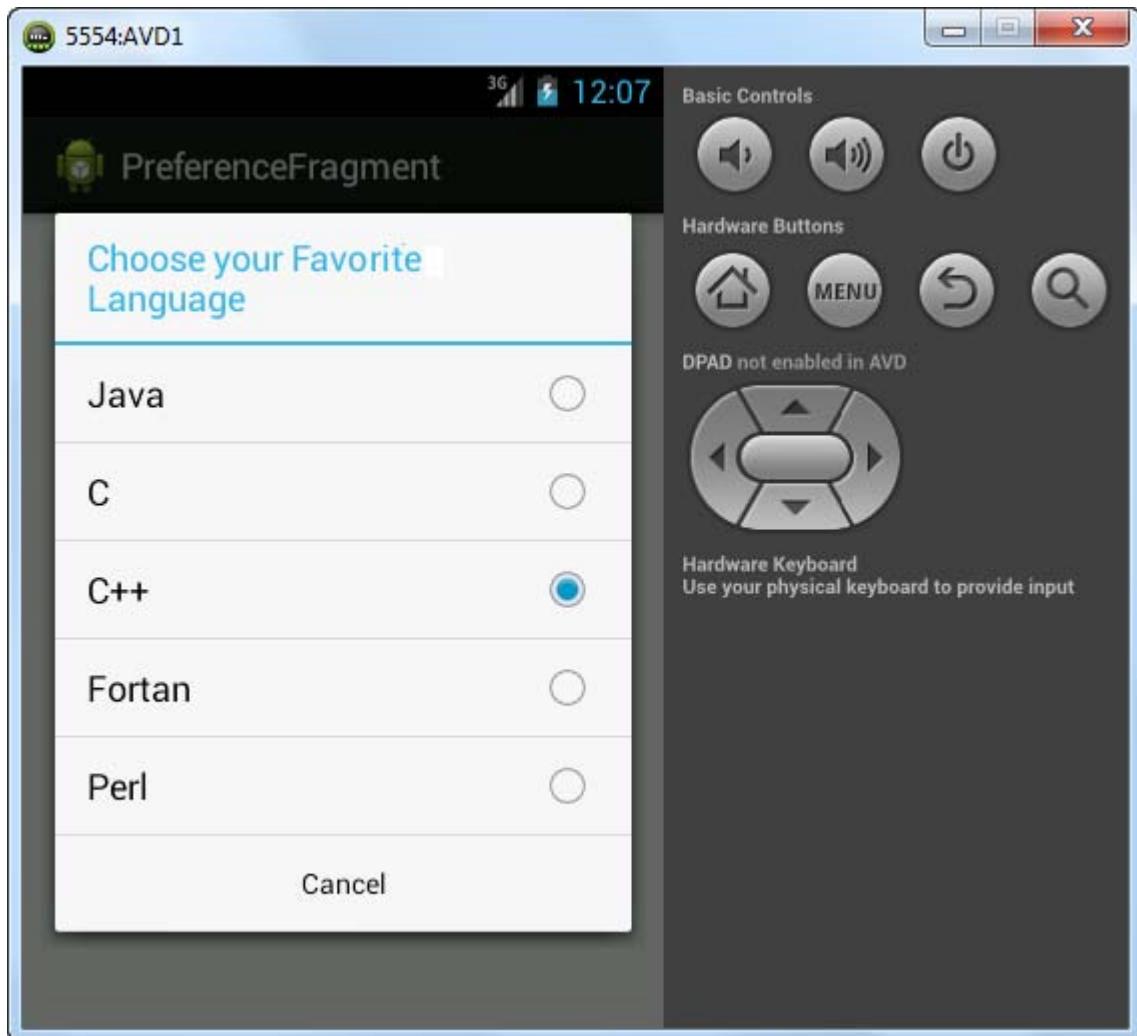


Figure 4.7: Choose Favorite Language CheckBox Preference

Click `title_edittext_preference`, to display the output as shown in figure 4.8.

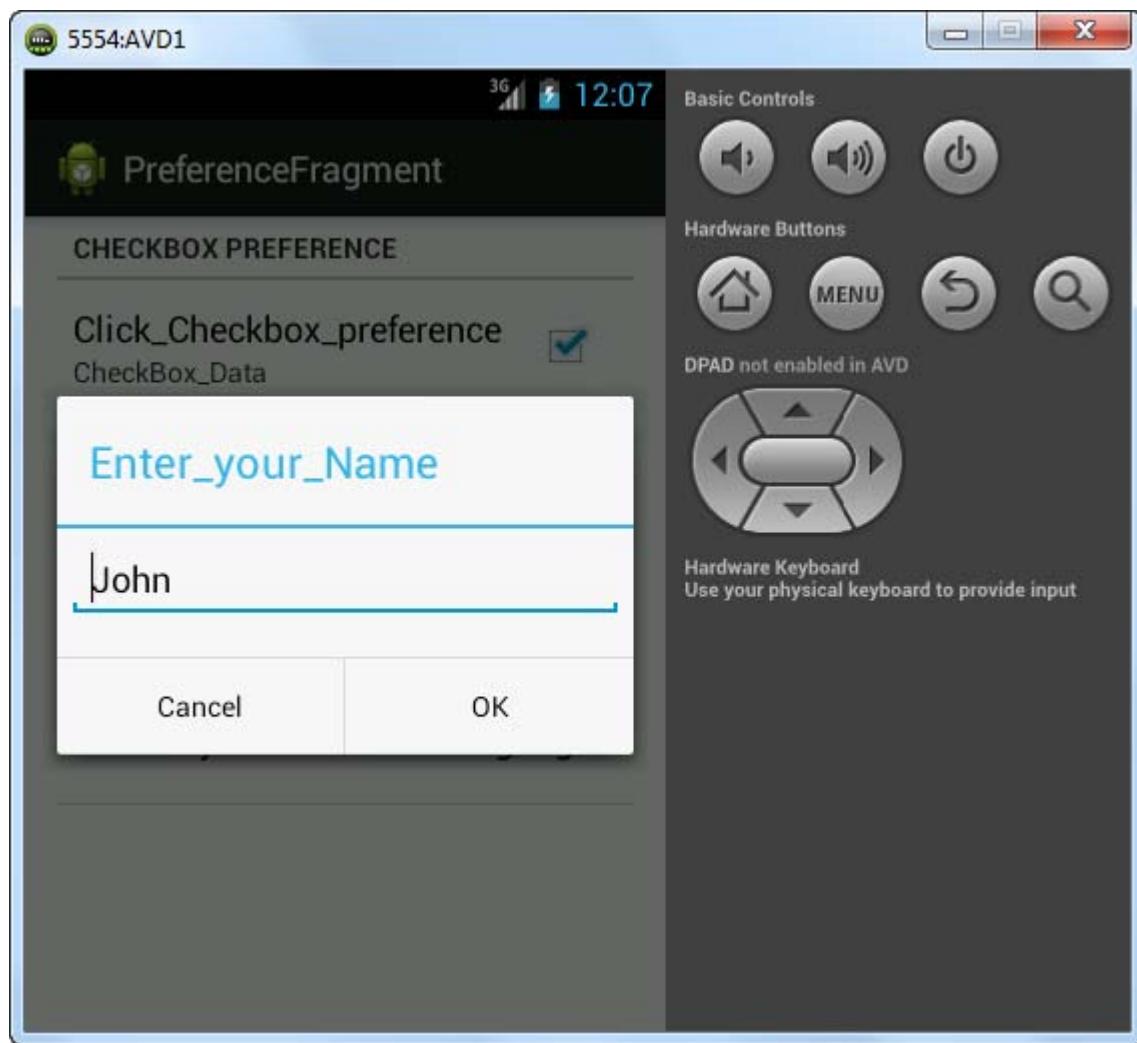


Figure 4.8: Title EditText Preference

As seen in code snippet 1 the developer creates an XML file and specifies the preferences in the file. The XML file provides a simple way of specifying preferences with a structure that is easily understood and managed.

The XML contains a list of preference objects and preference subclasses. The `<PreferenceScreen>` element refers to the top most level of a hierarchy and forms the basis of a preference hierarchy. In other words, the root element of the xml file is `<PreferenceScreen>` element. Each XML element can be matched to the class name, and will have a preference subclass assigned to it, such as, `<CheckBoxPreference>`. Every child that is added within the `<PreferenceScreen>` element exists as a single item within the list of settings.

Users may find it difficult to go through a list of 10 or more settings. In such cases, you can group the settings.

Within each `<PreferenceScreen>` element you can include any combination of `<PreferenceCategory>` and `Preference <control>` elements as shown in code snippet 9.

Code Snippet 9:

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

    <PreferenceCategory android:title="Checkbox Preference" >
        <CheckBoxPreference
            android:key="checkbox_preferencevalue"
            android:summary="CheckBox_Data"
            android:title="Click_Checkbox_preference" />
    </PreferenceCategory>
    <PreferenceCategory android:title="EditTextPreference" >
        <EditTextPreference
            android:dialogTitle="Enter_your_Name"
            android:key="edittext_preferencevalue"
            android:summary="summary_edittext_preference"
            android:title="title_edittext_preference" />
    </PreferenceCategory>
    <PreferenceCategory android:title="ListView Preference" >
        <ListPreference
            android:entries="@array/array_lp_entries"
            android:entryValues="@array/array_lp_entry_
values"
            android:key="lp_android_choice"
            android:title="@string/str_lp_android_choice_
title" />
    </PreferenceCategory>
</PreferenceScreen>
```

The XML file is typically named as `preferences.xml` and should be saved in the `res/xml/`

directory. For a single-pane layout, it is sufficient to create a single XML file. For multiple pane layouts, more than one XML file is required for specifying preferences. Code snippet 9 shows settings for three preferences – Checkbox, Editbox, and ListPreference. The attributes that can be defined for these preferences are as follows:

- `android:key` – This attribute is mandatory data value. It provides a distinct key for the system to save the data value. The data value is saved in the Shared Preferences file. The key is the unique value used to access the Preference.
- `android:title` – This attribute specifies a name for the setting which is visible to the user.
- `android:defaultValue` – The default value that will be displayed (and selected) if no preference value has been assigned to this preference key.

4.2.1 Dividing Settings into Groups

When there are more than ten settings to specify, the developer can divide the settings into groups. There are three ways in which group settings can be specified. These are by using titles, Intents, and sub screens.

- **Titles:** The developer can provide headings between groups of settings in the XML file so as to identify each group. To do this, each group of preference objects should be placed inside the `<PreferenceCategory>` element as shown in code snippet 10.

Code Snippet 10:

```
<PreferenceCategory android:title="ListView Preference" >
    <ListPreference
        android:entries="@array/array_lp_entries"
        android:entryValues="@array/array_lp_entry_values"
        android:key="lp_android_choice"
        android:title="@string/str_lp_android_choice_title" />
</PreferenceCategory>
```

- **Sub screen:** The developer can also place settings groups within a sub screen. To do this, the preference objects group needs to be inside the `<PreferenceScreen>` element. When a user makes changes to a preference, the changes are saved to the **Shared Preferences** file.

4.2.2 Providing Default Values

It is important to provide a default value for each preference object in the XML file. The Shared Preferences file is initialized using the default values specified when the user first

accesses the application. The default value is specified using the `android:defaultValue` attribute in the XML preference file. The value specified depends on the preference object as shown in code snippet 11.

Code Snippet 11:

```
<PreferenceCategory
    android:title="Checkbox Preference">

    <CheckBoxPreference
        android:key="checkbox_preferencevalue"
        android:title="Click_Checkbox_preference"
        android:summary="CheckBox_Data"
        android:defaultValue="true" />

</PreferenceCategory>
```

The developer then has to ensure that the default values are assigned by invoking the `setDefaultValues()` method when the user first accesses the application. Thus, the method is invoked in the `onCreate()` method of the main Activity class. The application will start with the default settings.

The `setDefaultValues()` method accepts three arguments. They are as follows:

- The context for the application.
- The resource identity for the XML file for which the default values are being set.
- A boolean value that determines the number of times the default values are set. If it is set to false, the default values are called only if the values have not been called before. If set to true, previous default values will be ignored. Code snippet 12 demonstrates the use of the `setDefaultValues()` method.

Code Snippet 12:

```
PreferenceManager.setDefaultValues(this, R.xml.user_profile, false);
```

4.2.3 Saving Data in SharedPreference

The following application shows how to create and store the shared preference data from the code.

The steps to create the application are as follows:

1. Start **Eclipse IDE**.
2. Create a project named **SharePreferenceExample**.

3. Navigate to **res → layout → activity_main.xml** file.
4. Modify the code of **activity_main.xml** file as shown in code snippet 13.

Code Snippet 13:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="vertical" >  
  
    <EditText  
        android:id="@+id/editText"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:ems="10" >  
  
        <requestFocus />  
    </EditText>  
  
    <Button  
        android:id="@+id/button"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_below="@+id/editText"  
        android:layout_centerHorizontal="true"  
        android:layout_marginTop="18dp"  
        android:text="Submit" />  
  
    <Button  
        android:id="@+id/buttonShow"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_centerHorizontal="true"  
        android:layout_centerVertical="true"  
        android:text="Show Stored Data" />
```

```
<TextView  
    android:id="@+id/textViewData"  
    android:layout_width="wrap_content"  
  
    android:layout_height="wrap_content"  
    android:layout_below="@+id/button"  
    android:layout_centerHorizontal="true"  
    android:layout_marginTop="41dp"  
    android:text=""  
    android:textAppearance="?android:attr/  
textAppearanceLarge" />  
  
</RelativeLayout>
```

5. Navigate to **src → com.example.sharedpreferenceexample → MainActivity.java** file.
6. Modify the code of **MainActivity.java** file as shown in code snippet 14.

Code Snippet 14:

```
package com.example.sharedpreferenceexample;  
  
import android.os.Bundle;  
import android.app.Activity;  
import android.view.Menu;  
import android.content.SharedPreferences;  
import android.preference.PreferenceManager;  
import android.view.View;  
import android.view.View.OnClickListener;  
import android.widget.Button;  
import android.widget.EditText;  
import android.widget.TextView;  
import android.widget.Toast;
```

```
public class MainActivity extends Activity {  
  
    SharedPreferences preferences;  
    SharedPreferences.Editor editor;  
  
    EditText editText;  
    Button submit;  
    Button show;  
    TextView dataText;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        preferences = PreferenceManager  
                .getDefaultSharedPreferences(MainActivity.this);  
        editText = (EditText) findViewById(R.id.editText);  
        submit = (Button) findViewById(R.id.button);  
        show = (Button) findViewById(R.id.buttonShow);  
        dataText = (TextView) findViewById(R.id.textViewData);  
        submit.setOnClickListener(new OnClickListener() {  
  
            @Override  
            public void onClick(View arg0) {  
                // TODO Auto-generated method stub  
  
                editor = preferences.edit();  
                editor.putString("data", editText.getText().  
                        toString());  
                editor.commit();  
                Toast.makeText(getApplicationContext(),  
                        "Data saved in preferences", Toast.LENGTH_LONG).  
                        show();  
            }  
        });  
    }  
}
```

```
        }

    });

show.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View arg0) {
        // TODO Auto-generated method stub
        String data = preferences.getString("data", "");
        if (!data.equals("")) {
            dataText.setText(data);
        } else {
            Toast.makeText(getApplicationContext(),
                    "No Data Available", Toast.LENGTH_
                    LONG).show();
        }
    }
});

@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;
}
}
```

When the application is executed the output will be as shown in figure 4.9.

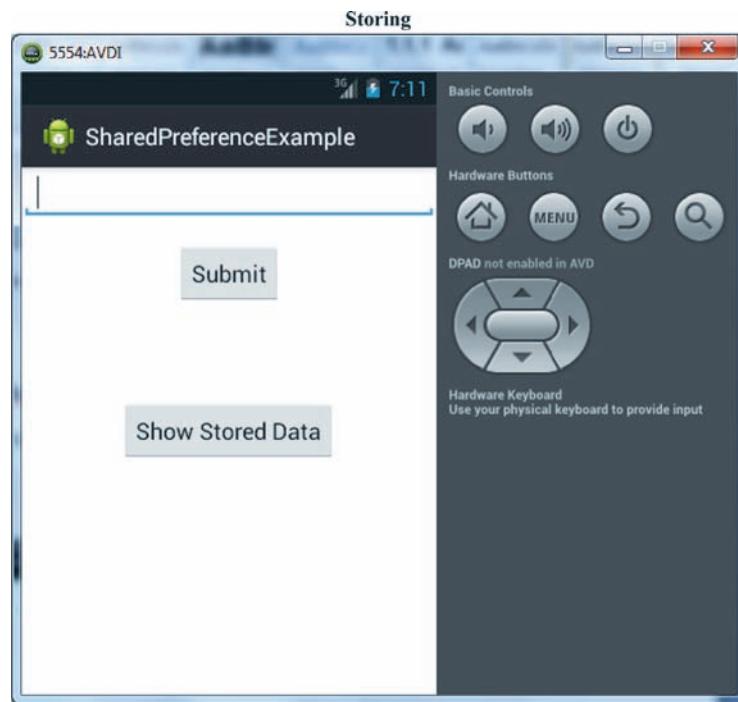


Figure 4.9: SharedPreferences

On typing the content in the edit box and clicking **Submit**, the output will be as shown in figure 4.10.

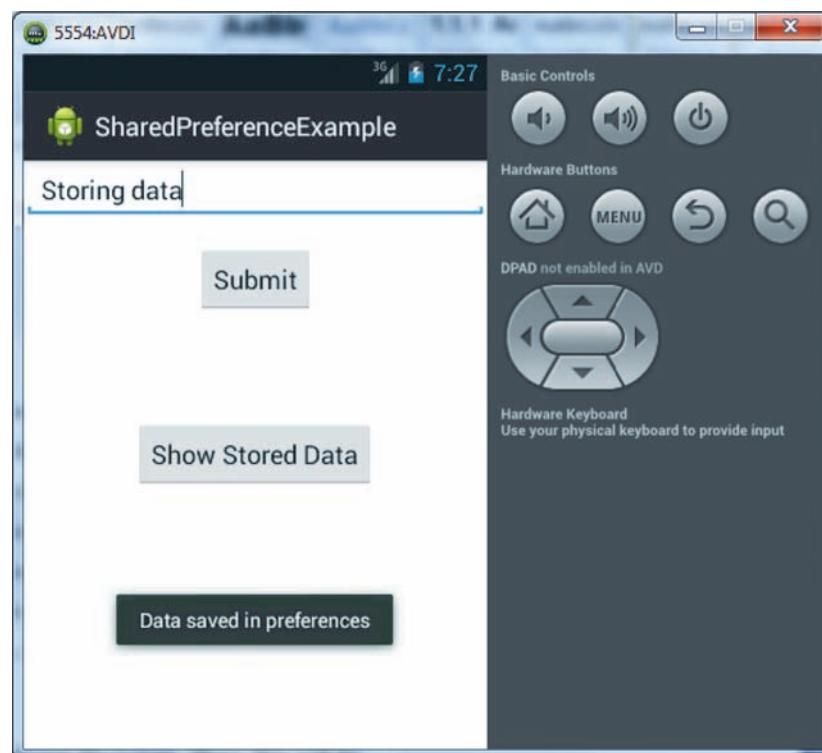


Figure 4.10: Enter Data to Store

On clicking the **Show Stored Data**, the output will be as shown in figure 4.11.

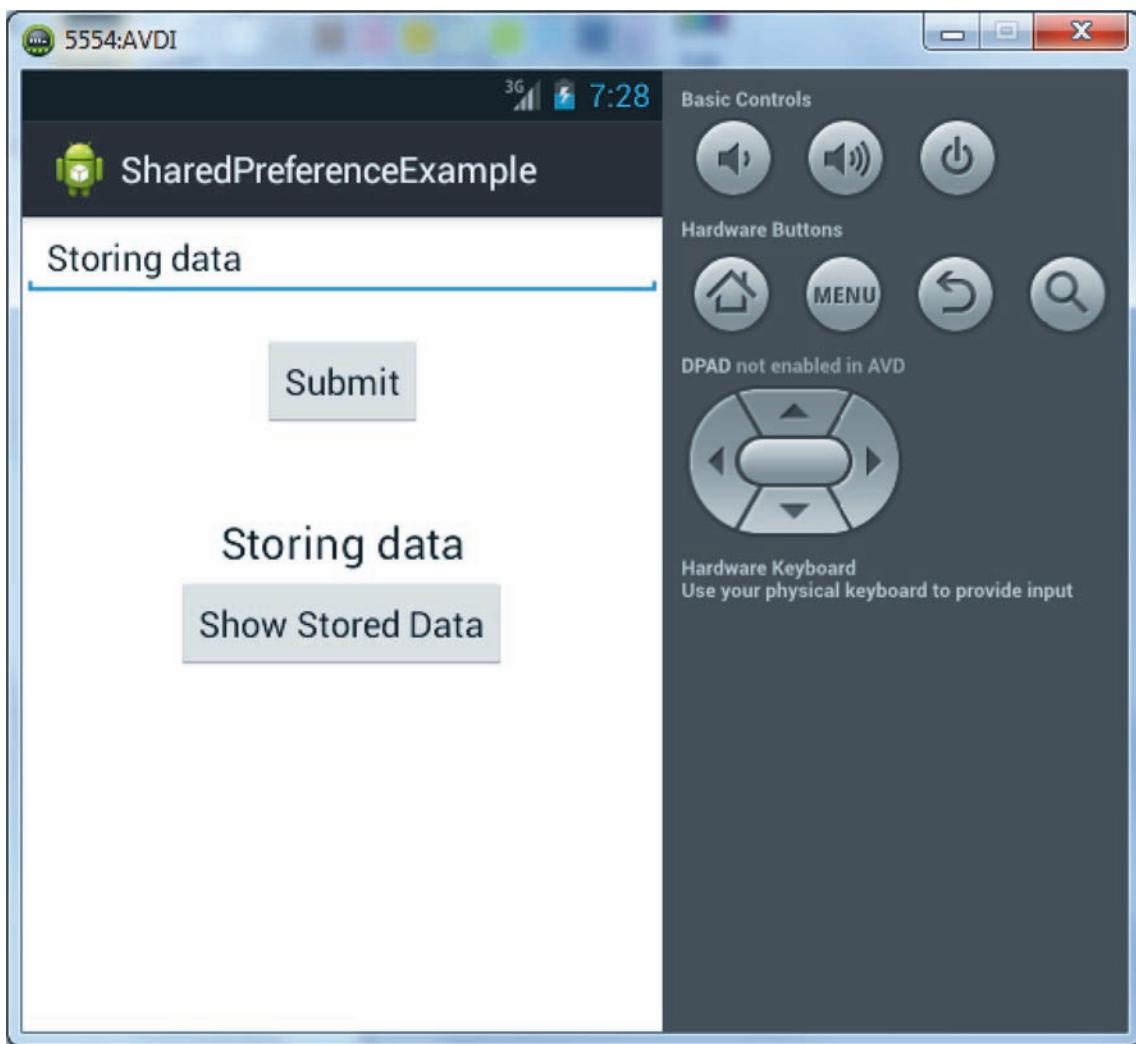


Figure 4.11: Displaying Data from SharedPreference

4.3 File System

Android's file system is based on physical storing options such as diskettes, CD ROMs, and hard disks. The developer can read and write files using the File APIs. Large chunks of data can be viewed and written without interruption using a file object. Android is characterized by two storage options namely, internal and external. Typically, the built-in memory is internal storage, while removable memory is external storage. However, in some cases, when there is no removable memory option, the existing storage space is partitioned into internal and external storage. Irrespective of whether removable storage exists, the API activities remain the same.

4.3.1 Internal Storage

Developers would sometimes like to store data in traditional file system. These files will be saved on the device's internal storage. The data saved by this method cannot be accessed by other applications. The data can be saved using the classes present in the `java.io` package.

Features of internal storage are as follows:

- More secure because only the related application can access files stored.
- Files are always accessible.
- When an application is uninstalled from the system, all related files are removed from internal storage.

→ Saving to Internal Storage

The internal storage directory for saving files can be obtained by using, the `getFilesDir()` and `getCacheDir()` methods. These `getCacheDir()` returns the path to the cache files whereas `getFilesDir()` returns the path to files created and stored in the internal storage of your app.

Code snippet 15 explain the use of this method.

Code Snippet 15:

```
File srDir=newFile(getFilesDir(), "file_name");
srDir.mkdirs();
• mkdirs(): used to create a new directory if not exists.
```

To create and write data to a file saved in the internal storage, perform the following steps:

1. Obtain an object of the `FileOutputStream` class by invoking the `openFileOutput()` method which accepts the filename and the operating mode. The operating mode can be `MODE_APPEND`, `MODE_WORLD_READABLE`, `MODE_WORLD_WRITABLE`, or `MODE_APPEND`.
2. Write to a file using the `write()` method of the input stream.
3. Invoke the `close()` method to close the stream.

Code Snippet 16 explains the creation and writing of data to a private file which will be present in the internal storage of the Android system.

Code Snippet 16:

```

...
String FILENAME = "android";
String string = "Learning Android programming!!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_
PRIVATE);
fos.write(string.getBytes());
fos.close();
...

```

To read data from a file saved in the internal storage, perform the following steps:

- Obtain an object of the `FileInputStream` class by invoking the `openFileInput()` method which accepts the filename.
- Read the data using the `read()` method.
- Invoke the `close()` method to close the stream.

4.3.2 Working with Storing Files in Internal Storage

This section explains the procedure to store files to the internal storage of an Android device. In this example, a new project will be created with two editable text boxes and a button. Text can be typed in the editable text box and the button will be used to save the text. This saved text can be viewed in the internal storage of an Android device using the DDMS perspective.

The steps to store files to the internal storage are as follows:

1. Start **Eclipse IDE**.
2. Create a project **InternalStore**.
3. Navigate to **InternalStore** → **res** → **layout** → **activity_main.xml** file in the Package Explorer.
4. Double-click **activity_main.xml**.
5. Modify the code of the **activity_main.xml** file as shown in code snippet 17.

Code Snippet 17:

```
<?xmlversion="1.0"encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/
    android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Enter Desired File Name" />

    <EditText
        android:id="@+id/filename"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Enter the File Content" />

    <EditText
        android:id="@+id/content"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/store"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Save File" />

</LinearLayout>
```

6. Navigate to **src → com.example.internalstore → MainActivity.java** file, in the **Package Explorer** pane of **Eclipse IDE**.
7. Double-click **MainActivity.java** class.
8. Modify the code of the **MainActivity.java** class.

Code snippet 18 displays the modified code for **MainActivity.java** class.

Code Snippet 18:

```
package com.example.internalstore;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {

    EditText etFName, etContent;
    Button buttonStore;

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

        etFName = (EditText) findViewById(R.id.filename);
        etContent = (EditText) findViewById(R.id.content);
        buttonStore = (Button) findViewById(R.id.store);
        buttonStore.setOnClickListener(new Button.OnClickListener() {
```

```
public void onClick(ViewV) {  
    // TODO Auto-generated method stub  
    String fileName = etFName.getText().toString();  
    String content = etContent.getText().toString();  
    FileOutputStream fos;  
    try {  
        fos = openFileOutput(fileName, Context.MODE_  
PRIVATE);  
        fos.write(content.getBytes());  
        fos.close();  
        Toast.makeText(MainActivity.this, fileName +  
"stored",  
        Toast.LENGTH_LONG).show();  
    } catch (FileNotFoundException e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    } catch (IOException e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    }  
}  
});  
}  
  
@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;  
}  
}
```

Figure 4.12 displays the emulator with required details entered once the application has been executed.

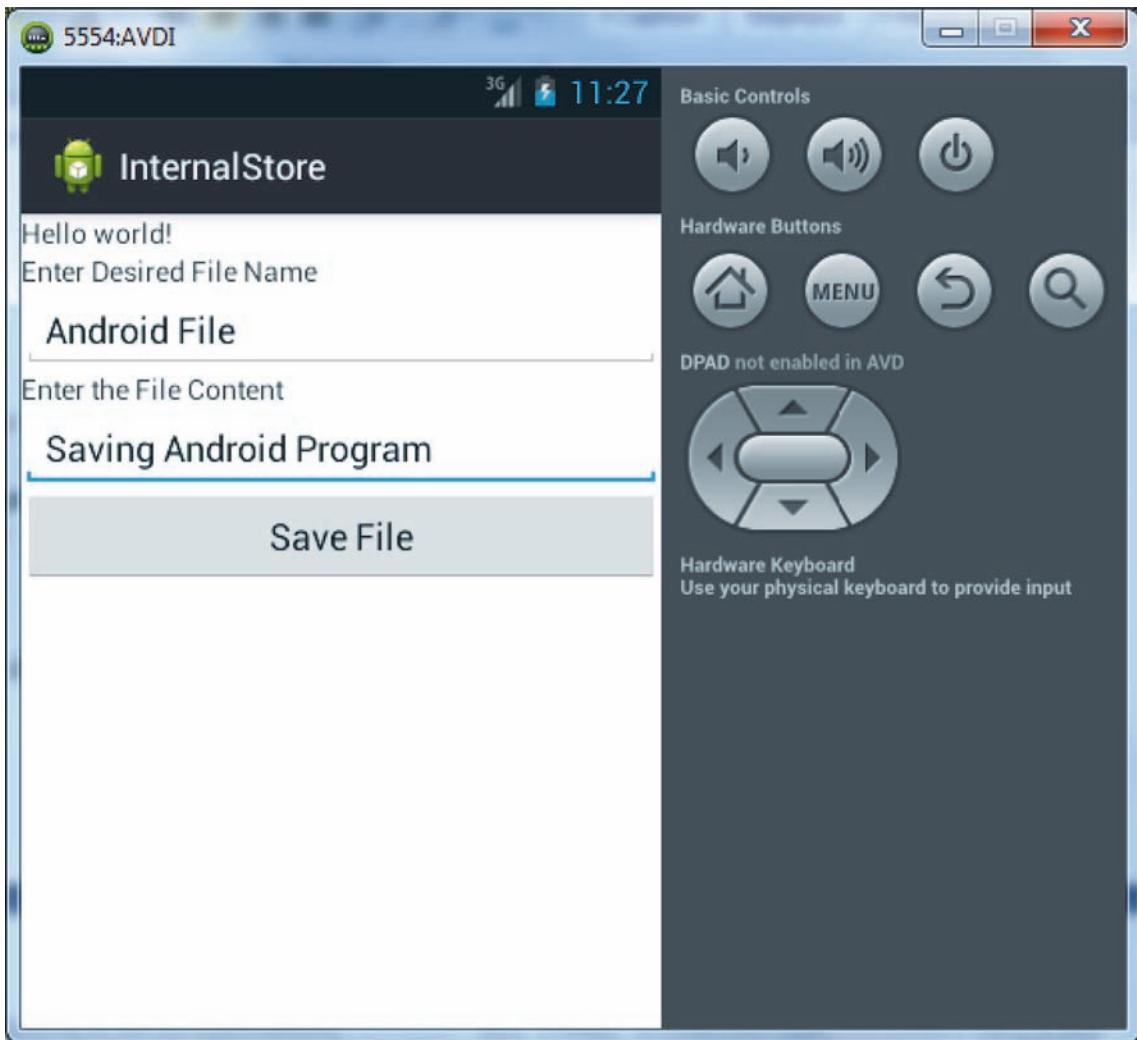


Figure 4.12: Internal Storage

9. Clicking **Save File** will display the output as shown in figure 4.13.

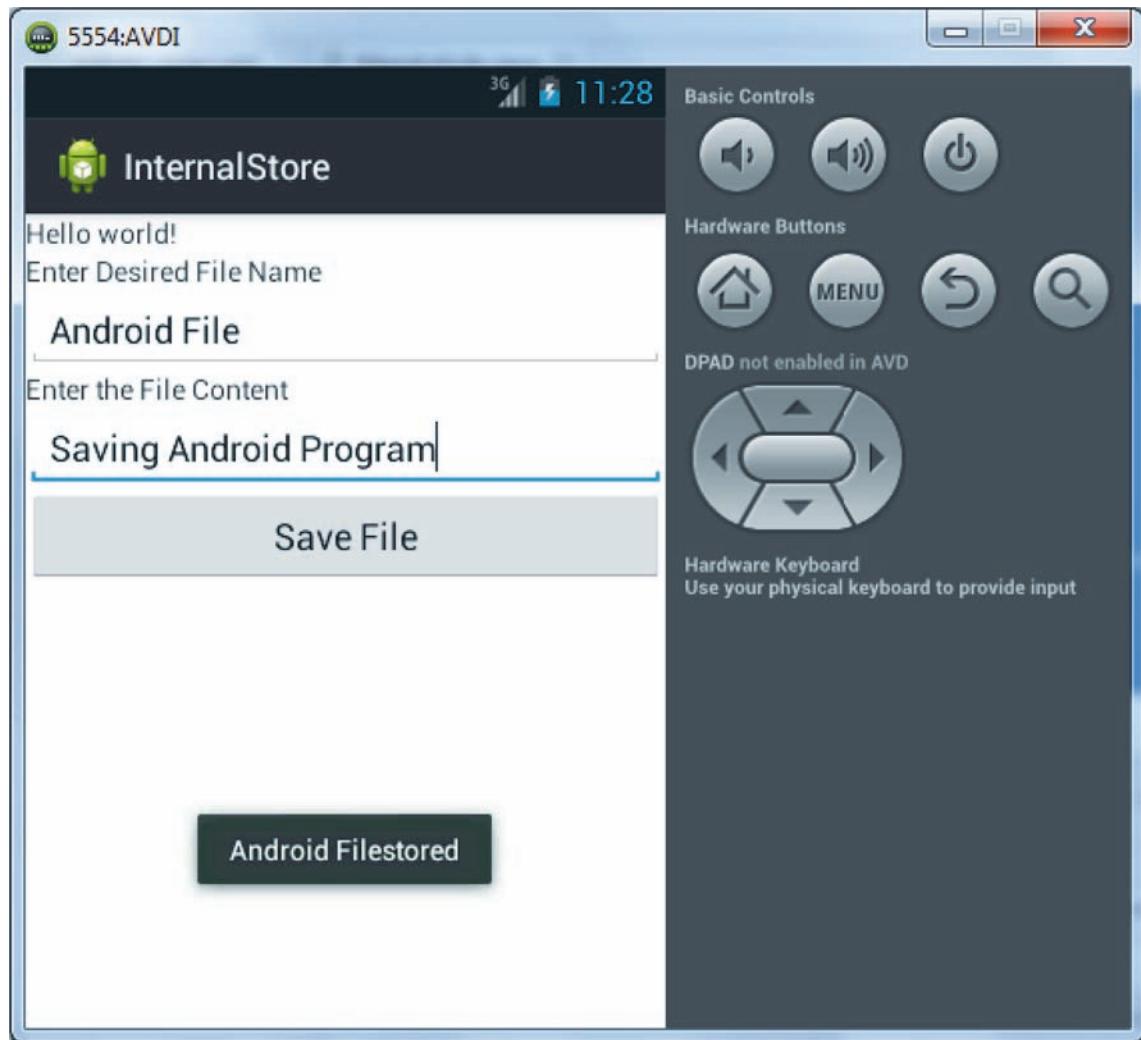


Figure 4.13: Toast Message – Saving File

The steps to view the stored file in the **Internal Store** are as follows:

10. Navigate to **Window → Open Perspective → Other** in the **Eclipse IDE**.
11. Click **Other**.

Figure 4.14 displays the **DDMS** pane.

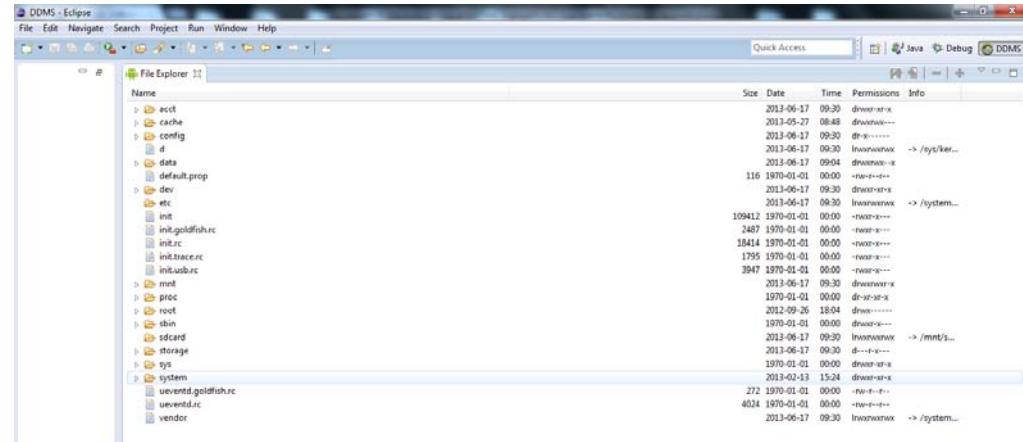


Figure 4.14: File Explorer

Navigate to **data → data → com.example.internalstore** folder. Figure 4.15 displays the folder with file named **Android File** stored in it.

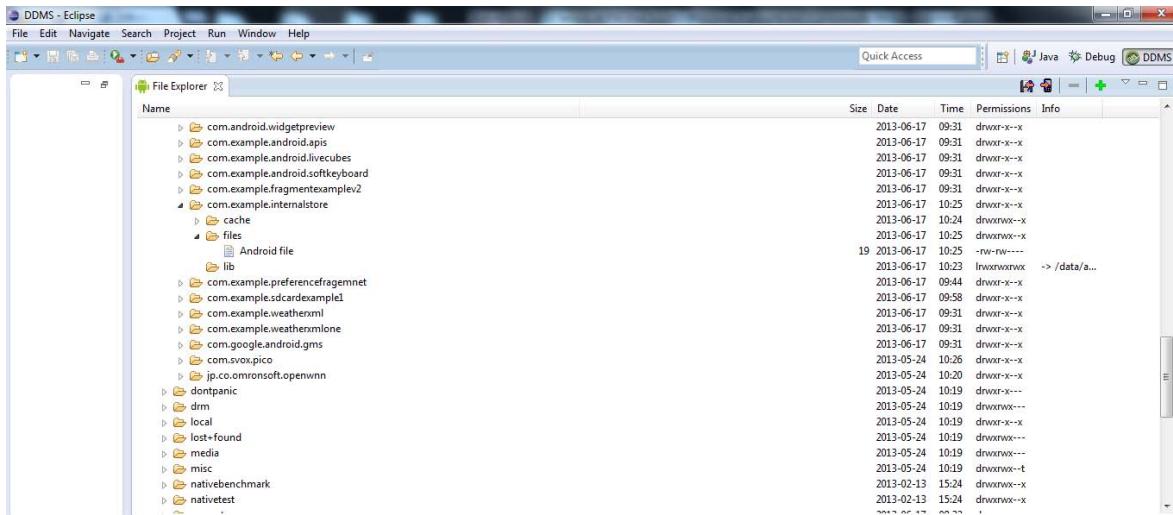


Figure 4.15: File store in Share Preference

The entered **Android File** with contents is stored in the internal database.

4.3.3 External Storage

Developers also have the facility to save files to an external storage, such as an SD card. When a file is saved to an external storage, it can be modified and shared by other users.

Some of the features of external storage are as follows:

- It cannot be accessed all the time, because it is removable.
- Files are not very secure as they are accessible by others. All apps can be accessed without permissions in external storage.
- When an application is uninstalled, the related files are completely removed only if they are saved to the `getFilesDir()` folder.

It is best to use external storage for files without access restrictions and for files that can be shared without user permissions for access.

→ Writing and Saving to External Storage

To read and write data to an external storage directory, perform the following steps:

- Invoke the `getExternalStorageDirectory()` method to obtain the absolute path of the SD card.
- Create a directory in the SD card.
- Read or write data using the `read()` or `write()` method.
- Provide the `WRITE_EXTERNAL_STORAGE` permission in the `AndroidManifest.xml` file.

Code snippet 19 displays the use of `WRITE_EXTERNAL_STORAGE` permission in the Android manifest file.

Code Snippet 19:

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

To save to external storage, the developer has to first check if the storage medium is available. Using the `getExternalStorageState()` method the developer can determine if the media is mounted and then save files to the medium. External files can be classified as public and private files. Public files are files that can be accessed by all users and remain undisturbed even after the app is uninstalled. The `getExternalStoragePublicDirectory()` method can be used to save a file to external storage.

To get the absolute path of external storage, one can access as shown in code snippet 20.

Code Snippet 20:

```
File file=new  
File(Environment.getExternalStorageDirectory().getAbsolutePath());
```

Private files are stored in a particular external directory when the app is installed and are automatically deleted when the app is uninstalled. The `getExternalFilesDir()` method can be used to save private files.

→ Deleting Files

It is good practice to delete unwanted files to ensure the availability of disk space. The developer can use the `delete()` method. For instance, `examplefile.delete()` : can be used to directly delete the file. For files in internal storage, the `deleteFile()` method can be used to locate the file and then delete it.

Note - The developer should periodically delete all files in cache using the `getCacheDir()` option.

Writing File in External SD Card:

1. Start Eclipse IDE.
2. Create a project named **ExternalStoreSDCard**.
3. Navigate to **ExternalStoreSDCard** → **res** → **layout** → **activity_main.xml** file in the **Package Explorer**.
4. Double-click **activity_main.xml**.
5. Modify the code of the **activity_main.xml** file as shown in code snippet 21.

Code Snippet 21:

```
<LinearLayout  
    android:id="@+id/widget28"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
  
    android:orientation="vertical"  
    xmlns:android="http://schemas.android.com/apk/res/android">
```

```
<EditText  
    android:id="@+id/txtData"  
    android:layout_width="fill_parent"  
    android:hint="Enter your name to write in sd card"  
    android:layout_height="180px"  
    android:textSize="18sp" />  
  
<Button  
    android:id="@+id	btnWriteSDFile"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Click for writing in SD File" />  
  
<Button  
    android:id="@+id	btnClearScreen"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Clear the Screen" />  
  
<Button  
    android:id="@+id	btnReadSDFile"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Click for reading content from sd card" />  
  
</LinearLayout>
```

6. Navigate to **src → com.example.externalstoresdcard → MainActivity.java** file, in the **Package Explorer** pane of **Eclipse IDE**.
7. Modify the code of the **MainActivity.java** class.
8. Code snippet 22 displays the modified code for **MainActivity.java** class.

Code Snippet 22:

```
package com.example.externalstoresdcard;

package com.example.sdcardexample1;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import java.io.*;
import android.app.Activity;
import android.os.Bundle;
import android.view.*;
import android.view.View.OnClickListener;
import android.widget.*;

public class MainActivity extends Activity {

    EditText txtData;
    Button btnWriteSDFile;
    Button btnReadSDFile;
    Button btnClearScreen;
    Button btnClose;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtData = (EditText) findViewById(R.id.txtData);
        txtData.setHint("Enter some lines of data here...");

        btnWriteSDFile = (Button) findViewById(R.id.btnWriteSDFile);
        btnWriteSDFile.setOnClickListener(new OnClickListener() {

            public void onClick(View v) {
                // write on SD card file data in the text box
            }
        });
    }
}
```

```
try {  
    File myFile = new File("/sdcard/mysdfile.txt");  
    myFile.createNewFile();  
    FileOutputStream fOut = new  
        FileOutputStream(myFile);  
    OutputStreamWriter myOutWriter = new  
        OutputStreamWriter(fOut);  
    myOutWriter.append(txtData.getText());  
    myOutWriter.close();  
    fOut.close();  
    Toast.makeText(getApplicationContext(),  
        "Done writing SD 'mysdfile.txt'",  
        Toast.LENGTH_SHORT).show();  
} catch (Exception e) {  
    Toast.makeText(getApplicationContext(), e.getMessage(),  
        Toast.LENGTH_SHORT).show();  
}  
}  
}); // onClick  
}); // btnWriteSDFfile  
  
btnReadSDFfile = (Button) findViewById(  
    R.id.btnReadSDFfile);  
  
btnReadSDFfile.setOnClickListener(new  
    OnClickListener() {  
        public void onClick(View v) {  
            // write on SD card file data in the text box  
            try {  
                File myFile = new File("/sdcard/mysdfile.txt");  
                FileInputStream fIn = new FileInputStream(myFile);  
                BufferedReader myReader = new BufferedReader(  
                    new InputStreamReader(fIn));  
            }  
        }  
    }  
});
```

```

String aDataRow = "";
String aBuffer = "";
while ((aDataRow = myReader.readLine()) != null) {
    aBuffer += aDataRow + "\n";
}
txtData.setText(aBuffer);
myReader.close();
Toast.makeText(getApplicationContext(),
        "Done reading SD 'mysdfile.txt'",
        Toast.LENGTH_SHORT).show();
} catch (Exception e) {
    Toast.makeText(getApplicationContext(), e.getMessage(),
        Toast.LENGTH_SHORT).show();
}
} // onClick
}); // Reading from sd card and displaying in the textView
btnClearScreen = (Button) findViewById(R.
id.btnClearScreen);
btnClearScreen.setOnClickListener(new
OnClickListener() {
    public void onClick(View v) {
        // clear textBox
        txtData.setText("");
    }
}); // btnClearScreen close

} // onCreate close
}

```

9. Type the code as shown in code snippet 23 in **AndroidManifest.xml** file.

Code Snippet 23:

```
<?xmlversion="1.0"encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/
    android"
    package="com.example.externalstoresdcard"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-sdk
        android:minSdkVersion="9"
        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.externalstoresdcard.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>

    <uses-permission android:name="android.permission.READ_EXTERNAL_
        STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_
        STORAGE" />

</manifest>
```

Once you execute the application, the output will be as shown in figure 4.16.

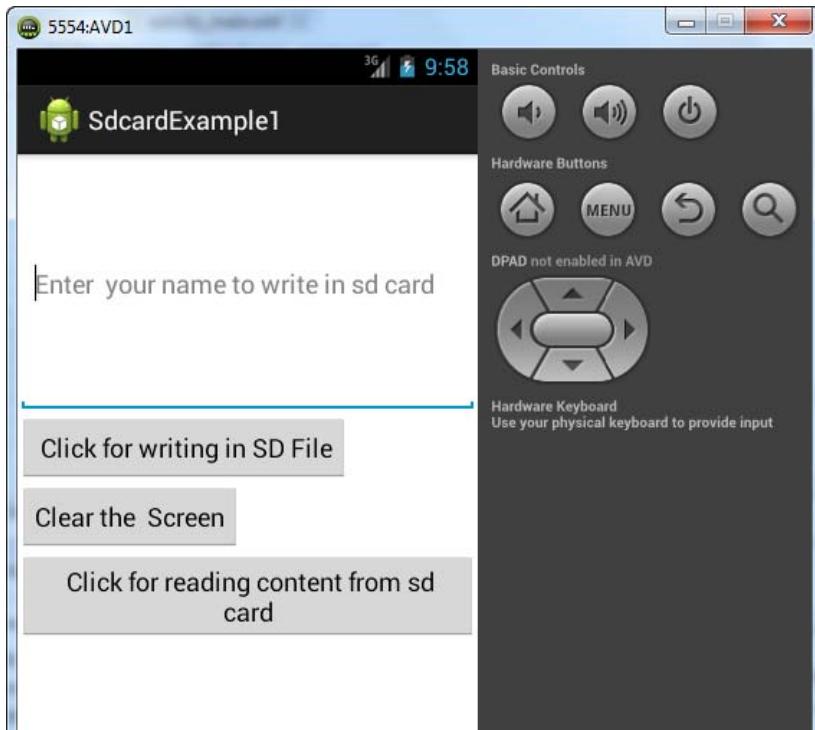


Figure 4.16: Writing to Store in SD card

After typing the content, the user should click the button to save it in the SD Card. The output will be as displayed in figure 4.17.

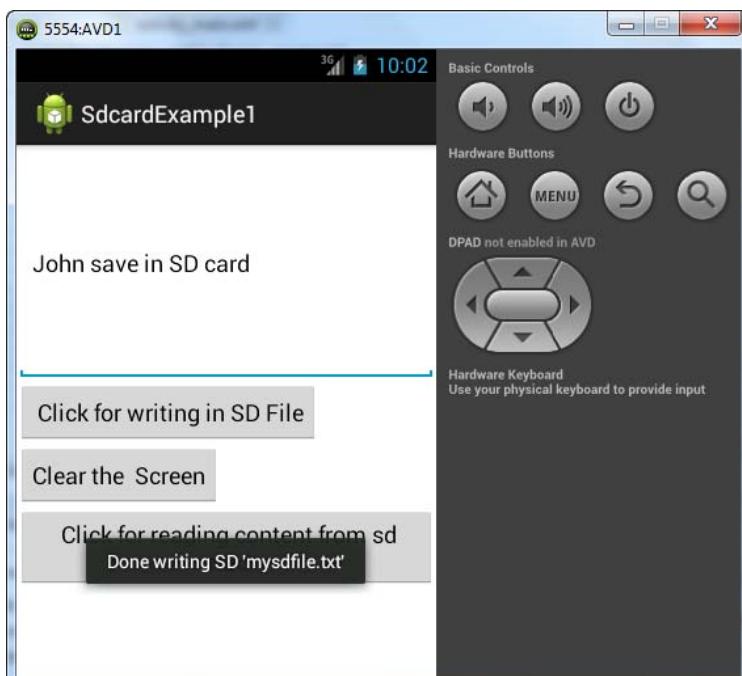


Figure 4.17: Writing in SD Card

Click the required button for retrieving the data and the output will be as shown in figure 4.18.

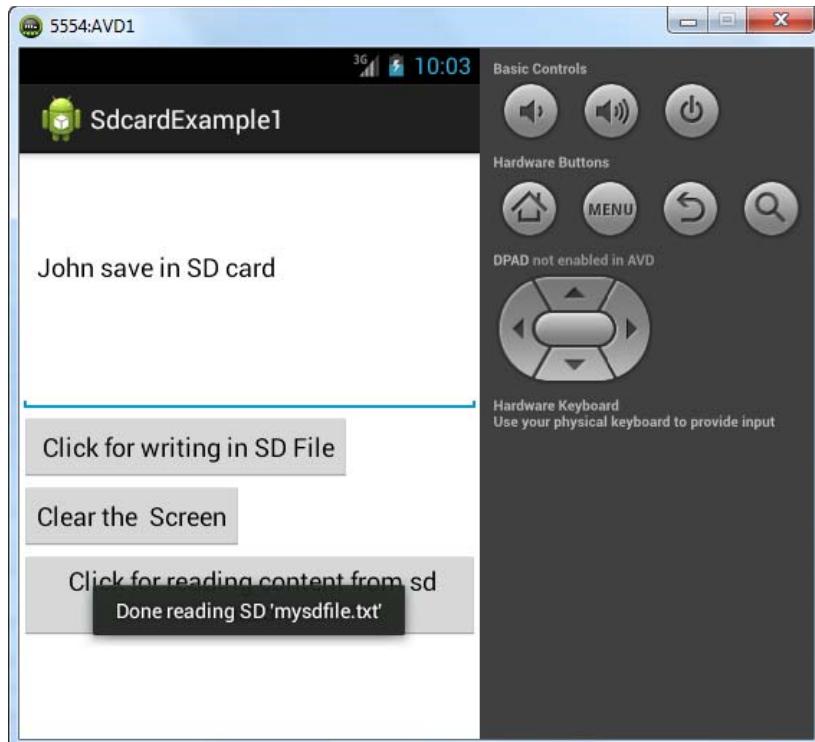


Figure 4.18: Reading SD Card

4.4 Notifications

Android's notifications enable an app to inform the user about events. Events include calendar events and other alerts. The different types of notifications include dialogs, popups, and toasts, and status notifications. A Service notifies the user of any event taking place using either **Toast Notifications** or **Status Bar Notifications**. Toast notifications displays a non-modal dialog box containing the message and pops on the window surface for a few seconds. The current activity remains visible and the user can interact with it. On the other hand, a status bar notification displays an icon in the status bar that contains a message, according to which the user can take necessary actions.

4.4.1 Dialogs and Alerts

A dialog is displayed in front of an activity as a small window. Dialog boxes help users in selecting and confirming actions, or alerting, or displaying warning messages. When dialogs are active, the activity loses the focus. It is used for interrupting the user activity and forces the user to perform a task related to the current activity. A dialog can be created by extending the `Dialog` class. A dialog should not be instantiated directly. It should be done by using any one of the following subclasses:

1. **DatePickerDialog** – This helps the user to select a date from the DatePicker View.
2. **TimePickerDialog** – This helps the user to select a time from the TimePicker View.
3. **ProgressDialog** – This displays a progress wheel or progress bar to the user.
4. **AlertDialog** – It can manage zero, one, two, or three buttons. It also manages a specified set of items that include checkboxes or radio buttons.

→ **DatePickerDialog and TimePickerDialog**

A DatePickerDialog helps the user to select a date from the DatePicker View and a TimePickerDialog helps the user to select a time from the TimePicker View.

This section explains the procedure to create a DateTime application to illustrate the use of a DatePickerDialog and TimePickerDialog to set the date and time.

The steps to create the DateTime project are as follows:

1. Start **Eclipse IDE**.
2. Create a project named **DateTime**.
3. Navigate to **DateTime** → **res** → **layout** → **activity_main.xml** in the **Package Explorer** pane.
4. Double-click the **activity_main.xml** file.
5. Modify the code in **activity_main.xml** file as shown in code snippet 24.

Code Snippet 24:

```
<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:onClick="selectDate" >

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" />

```

```
        android:layout_marginLeft="20dp"
        android:layout_marginTop="56dp"
        android:ems="10"
        android:inputType="date" >
        <requestFocus />
    </EditText>

    <ImageButton
        android:id="@+id/imageButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/editText1"
        android:layout_toRightOf="@+id/editText1"
        android:cropToPadding="true"
        android:onClick="selectDate"
        android:src="@drawable/ic_datepicker" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignRight="@+id/editText1"
        android:layout_centerVertical="true"
        android:layout_marginRight="50dp"
        android:text="Next" />

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
```

```

        android:layout_alignRight="@+id/button1"
        android:layout_marginRight="32dp"
        android:layout_marginTop="20dp"
        android:text="Date Picker Example" />
</RelativeLayout>
```

6. Create an Android xml file named **activitymain.xml** in **res/layout/** folder.
7. Modify the code as shown in code snippet 25.

Code Snippet 25:

```

<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:id="@+id/text_picked_time"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true" />

    <Button
        android:id="@+id/button_pick_time"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/text_picked_time"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="Click for Time" />

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignRight="@+id/text_picked_time"
        android:layout_marginTop="72dp"
        android:text="Time Picker Example" />

</RelativeLayout>
```

8. Navigate to `src → com.example.datetime` package.
9. Create a Java class named `TimePickerFragment.java` file.
10. Modify the code as given in code snippet 26.

Code Snippet 26:

```
package com.example.datetime;
import android.app.Activity;
import android.app.Dialog;
import android.app.TimePickerDialog;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;
import android.text.format.DateFormat;
import android.widget.TimePicker;

import java.util.Calendar;

public class TimePickerFragment extends DialogFragment implements
TimePickerDialog.OnTimeSetListener{

    private TimePickedListener mListener;
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState)
    {
        // use the current time as the default values for the picker
        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);

        // create a new instance of TimePickerDialog and return it
        return new TimePickerDialog(getActivity(), this, hour, minute,
        DateFormat.is24HourFormat(getActivity()));
    }

    @Override
    public void onAttach(Activity activity)
    {
```

```

// when the fragment is initially shown (i.e. attached to
// the activity), cast the activity to the callback interface
// type

    super.onAttach(activity);

    try
    {
        mListener = (TimePickedListener) activity;
    }
    catch (ClassCastException e)
    {
        throw new ClassCastException(activity.
                toString() + " must implement " +
                TimePickedListener.class.getName());
    }
}

public void onTimeSet(TimePicker view, int hourOfDay, int
minute)
{
    // when the time is selected, send it to the activity via
    // its callback interface method

    Calendar c = Calendar.getInstance();
    c.set(Calendar.HOUR_OF_DAY, hourOfDay);
    c.set(Calendar.MINUTE, minute);
    mListener.onTimePicked(c);
}

public static interface TimePickedListener
{
    public void onTimePicked(Calendar time);
}
}

```

11. Create a Java class named **Timepicker.java** file, under **src/com.example.datetime** package.
12. Modify the code as given in code snippet 27.

Code Snippet 27:

```
package com.example.datetime;

import android.os.Bundle;
import android.support.v4.app.DialogFragment;
import android.support.v4.app.FragmentActivity;
import android.text.format.DateFormat;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

import java.util.Calendar;

import com.example.datetime.TimePickerFragment.TimePickedListener;

public class Timepicker extends FragmentActivity implements
TimePickedListener {

    private TextView mPickedTimeText;
    private Button mPickTimeButton;

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

        mPickedTimeText = (TextView) findViewById(R.id.text_picked_time);
        mPickTimeButton = (Button) findViewById(R.id.button_pick_time);
        mPickTimeButton.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
```

```

        // show the time picker dialog

        DialogFragment newFragment = new
            TimePickerFragment();

        newFragment.show(getSupportFragmentManager(),
"timePicker");
    }

}

public void onTimePicked(Calendar time) {
    // display the selected time in the TextView

    mPickedTimeText.setText.DateFormat.format("h:mma", time));
}
}

```

13. Navigate to **src → com.example.datetime → MainActivity.java** file.

14. Modify the code as given in code snippet 28.

Code Snippet 28:

```

package com.example.datetime;

import java.util.Calendar;
import android.annotation.SuppressLint;
import android.app.DatePickerDialog;
import android.app.Dialog;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.support.v4.app.FragmentActivity;
import android.support.v4.app.DialogFragment;
import com.example.datetime.R;

import android.widget.Button;
import android.widget.DatePicker;

```

```
import android.widget.EditText;

@SuppressWarnings("ValidFragment")
public class MainActivity extends FragmentActivity {
    EditText mEdit;

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

        Button next = (Button) findViewById(R.id.button1);
        next.setOnClickListener(new OnClickListener() {

            public void onClick(View v) {
                Intent o=new
                Intent(getApplicationContext(),Timepicker.class);
                startActivity(o)
            ;
            }
        });
    }

    public void setDate(View view) {
        DialogFragment newFragment = new SelectDateFragment();
        newFragment.show(getSupportFragmentManager(), "DatePicker");
    }

    public void populateSetDate(int year, int month, int day) {
        mEdit = (EditText) findViewById(R.id.editText1);
        mEdit.setText(month+"/"+day+"/"+year);
    }
}
```

```
public class SelectDateFragment extends DialogFragment
implements DatePickerDialog.OnDateSetListener {

    @Override

    public Dialog onCreateDialog(Bundle savedInstanceState) {
        final Calendar calendar = Calendar.getInstance();
        int yy = calendar.get(Calendar.YEAR);
        int mm = calendar.get(Calendar.MONTH);
        int dd = calendar.get(Calendar.DAY_OF_MONTH);
        return new DatePickerDialog(getActivity(), this, yy,
mm, dd);
    }

    public void onDateSet(DatePicker view, int yy, int mm, int dd) {
        populateSetDate(yy, mm+1, dd);
    }
}

@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;
}
}
```

15. Navigate to **AndroidManifest.xml** file.

16. Add the line of code, <activity android name=".Timepicker"></activity>

Now, the **AndroidManifest.xml** file will look like the code present in code snippet 29.

Code Snippet 29:

```

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

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.datetime"
    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.datetime.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>
        <activity android:name=".Timepicker"></activity>
    </application>
</manifest>

```

17. Navigate to **res → drawable-hdpi** folder.

18. Paste the image **ic_datepicker.png**.

Note - The image name should be in small letter and there should be no space.

19. Navigate to **res → values → strings.xml**.

20. Modify the code as given in code snippet 30.

Code Snippet 30:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">DateTime</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_main">MainActivity</string>
    <string name="select">Select</string>
    <string name="selectdate">Select Date</string>
</resources>
```

Once you execute the application the output will be as shown in figure 4.19.

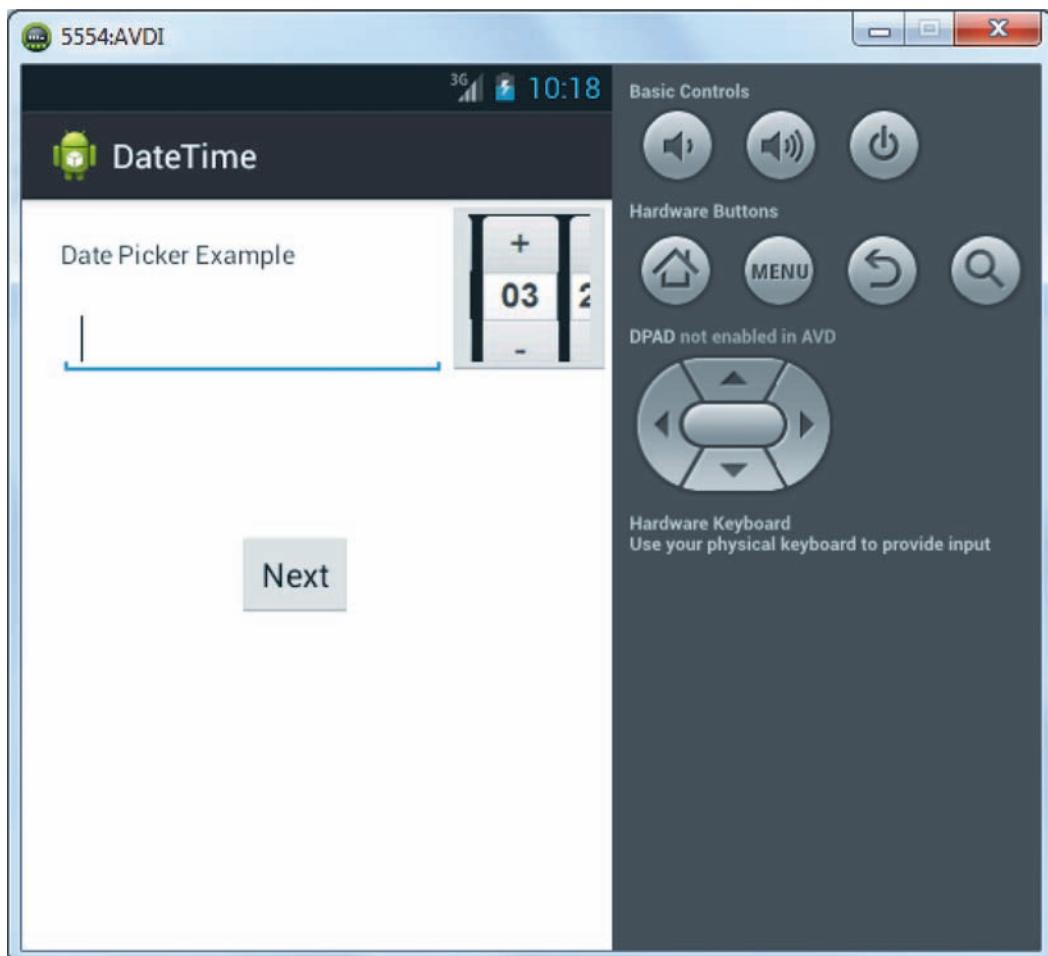


Figure 4.19: Date Picker

Once the date icon is clicked, the output will be as shown in figure 4.20.

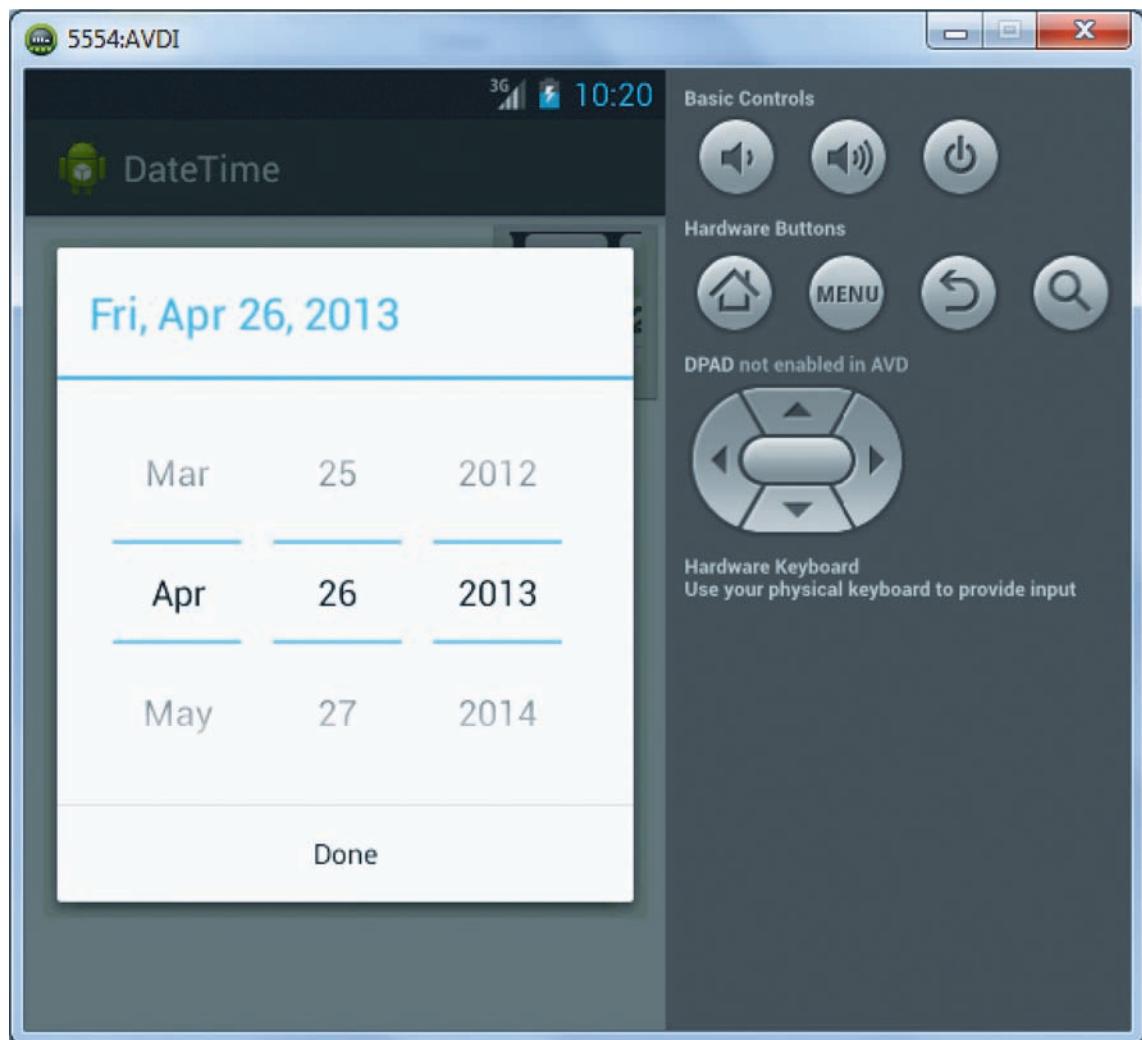


Figure 4.20: Pick the Date

Click **Done** and the output will be as shown in figure 4.21.

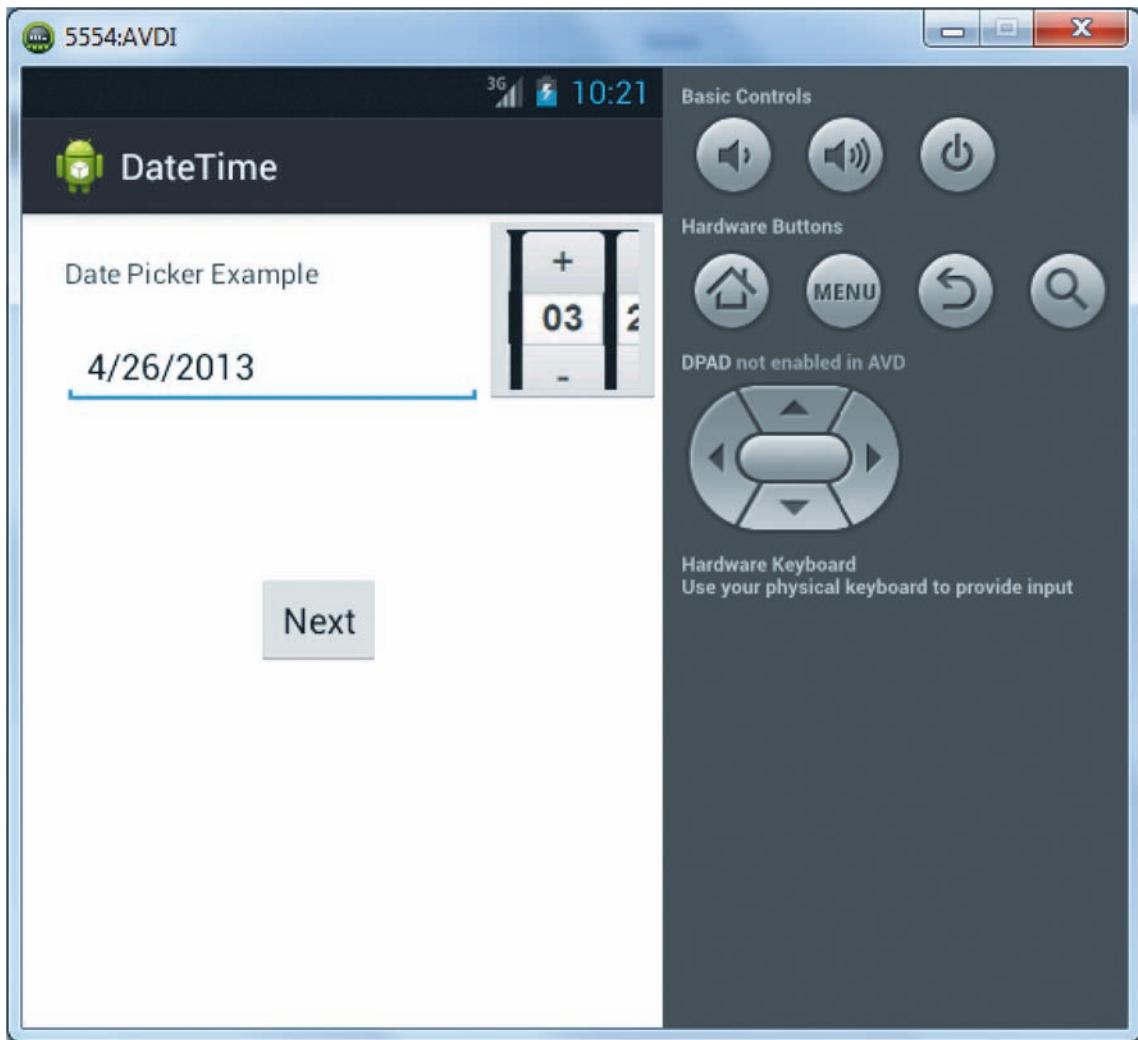


Figure 4.21: Clicking Done

Click **Next** and the output will be as shown in figure 4.22.

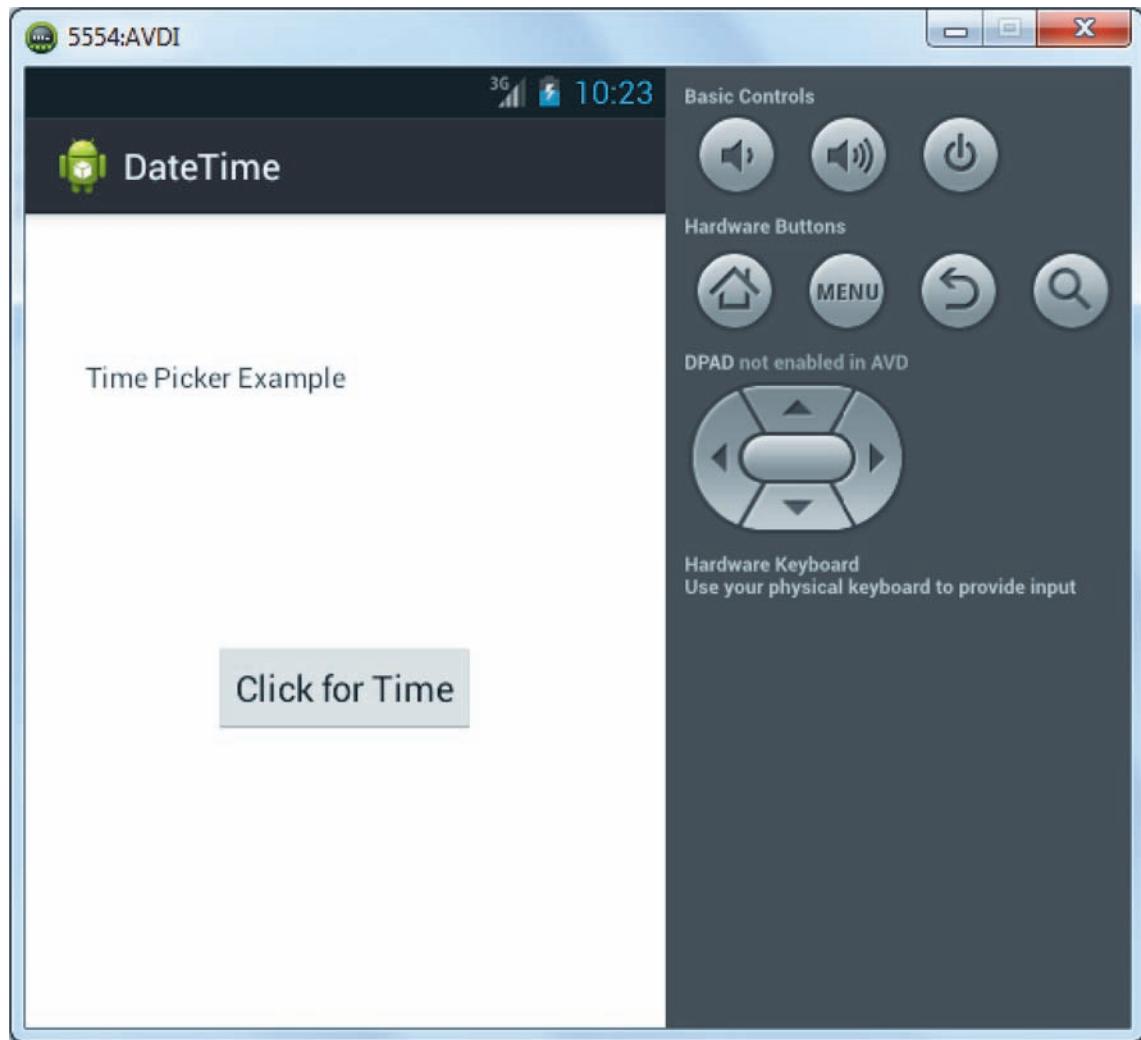


Figure 4.22: Clicking Next

Click the button **Click for Time** and the output will be as shown in figure 4.23.

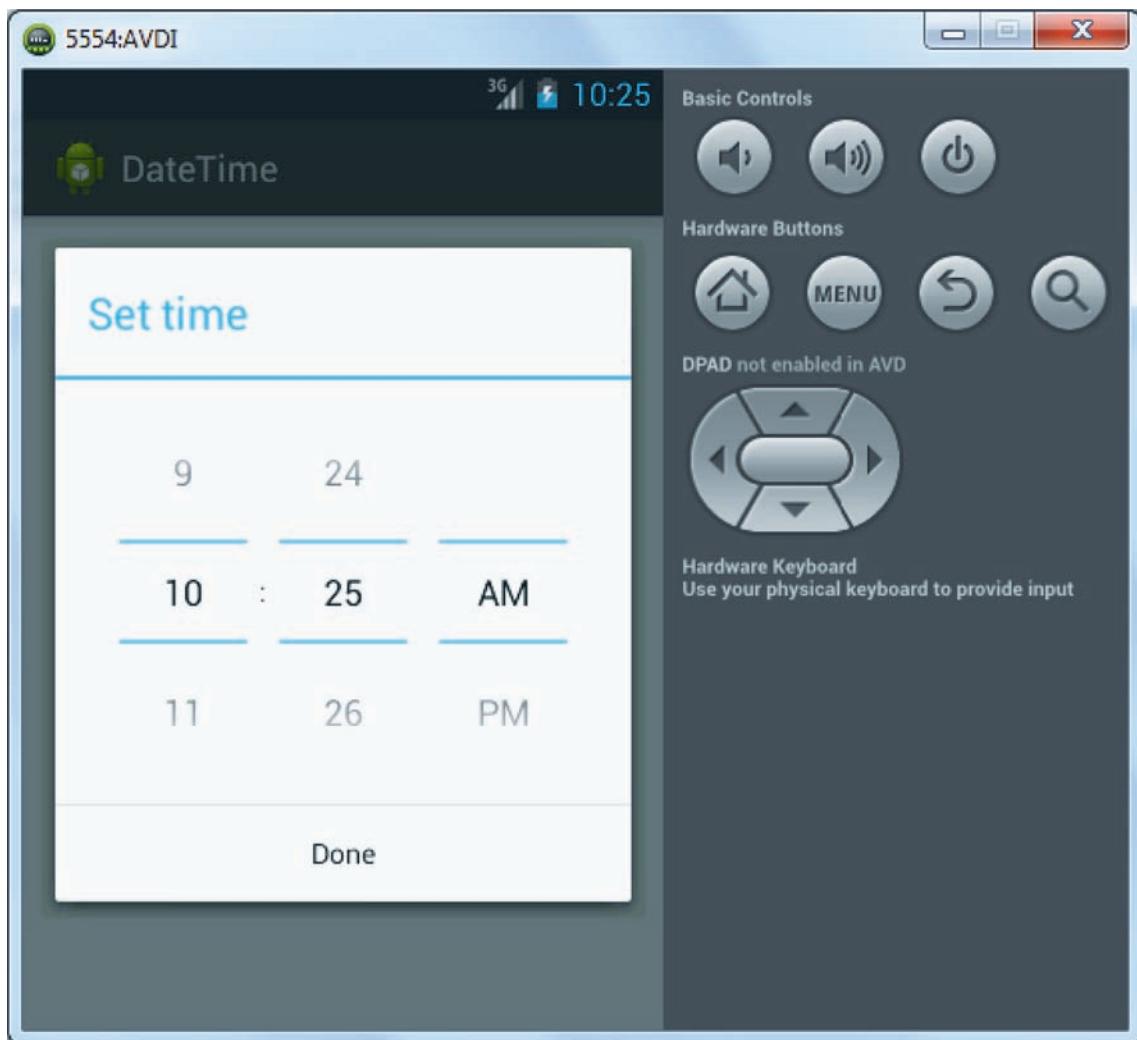


Figure 4.23: Set the time

→ Progress Dialog Notification

It is a dialog with a progress wheel or bar. It is specifically useful for cases where the application fetches some data from the Internet and when done, display it on the device screen. Since it takes 6-7 second to do that, meanwhile, you can use the progress dialog to show the status of download.

This section explains the procedure to create a Progress Dialog application to illustrate the use of a Progress Dialog.

The step to create a **ProgressDialog** application is as follows:

1. Start **Eclipse IDE**.
2. Create a project named **ProgressDialog**.
3. Navigate to **res → layout → activity_main.xml**.
4. Modify the code of **activity_main.xml** file as shown in code snippet 31.

Code Snippet 31:

```
<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/buttonprogressdialog"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="31dp"
        android:layout_marginLeft="50dp"
        android:layout_marginRight="50dp"
        android:text="ProgressDialog" />

</RelativeLayout>
```

5. Navigate to **src → com.example.progressdialog → MainActivity.java** file.
6. Modify the code of **MainActivity.java** file as shown in code snippet 32.

Code Snippet 32:

```
package com.example.progressdialog;

import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.app.Activity;
import android.view.Menu;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.view.View;
import android.app.ProgressDialog;

public class MainActivity extends Activity {
    Button pdring_btn, pbar_btn;
    ProgressDialog myPd_bar;

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

        pdring_btn = (Button) findViewById(R.id.buttonprogressdialog);
        pdring_btn.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                // TODO Auto-generated method stub
                final ProgressDialog myPd_ring = ProgressDialog.show(
                    MainActivity.this, "Please wait",
                    "Loading please wait..", true);
                myPd_ring.setCancelable(true);
                new Thread(new Runnable() {
                    @Override
```

```
public void run() {  
    // TODO Auto-generated method stub  
    try {  
        Thread.sleep(5000);  
    } catch (Exception e) {  
    }  
    myPd_ring.dismiss();  
}  
}).start();  
}  
});  
pabar_btn = (Button) findViewById(R.id.buttonprogressdialog);  
pabar_btn.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        // TODO Auto-generated method stub  
        myPd_bar = new ProgressDialog(MainActivity.this);  
        myPd_bar.setMessage("Loading....");  
        myPd_bar.setTitle("Please Wait...");  
        myPd_bar.setProgressStyle(myPd_bar.STYLE_HORIZONTAL);  
        myPd_bar.setProgress(0);  
        myPd_bar.setMax(30);  
        myPd_bar.show();  
        new Thread(new Runnable() {  
            @Override  
            public void run() {  
                // TODO Auto-generated method stub  
                try {  
                    while (myPd_bar.getProgress() <=
```

```
myPd_bar.getMax() ) {  
    Thread.sleep(1000);  
    handle.sendMessage(handle.  
obtainMessage());  
    if (myPd_bar.getProgress() ==  
myPd_bar.getMax() ) {  
        myPd_bar.dismiss();  
    }  
}  
} catch (Exception e) {  
}  
}  
}  
}).start();  
}  
}  
Handler handle = new Handler() {  
    @Override  
    public void handleMessage(Message msg) {  
        // TODO Auto-generated method stub  
        super.handleMessage(msg);  
        myPd_bar.incrementProgressBy(5);  
    }  
};  
});  
}  
@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;  
}  
}
```

Figure 4.24 displays the output.

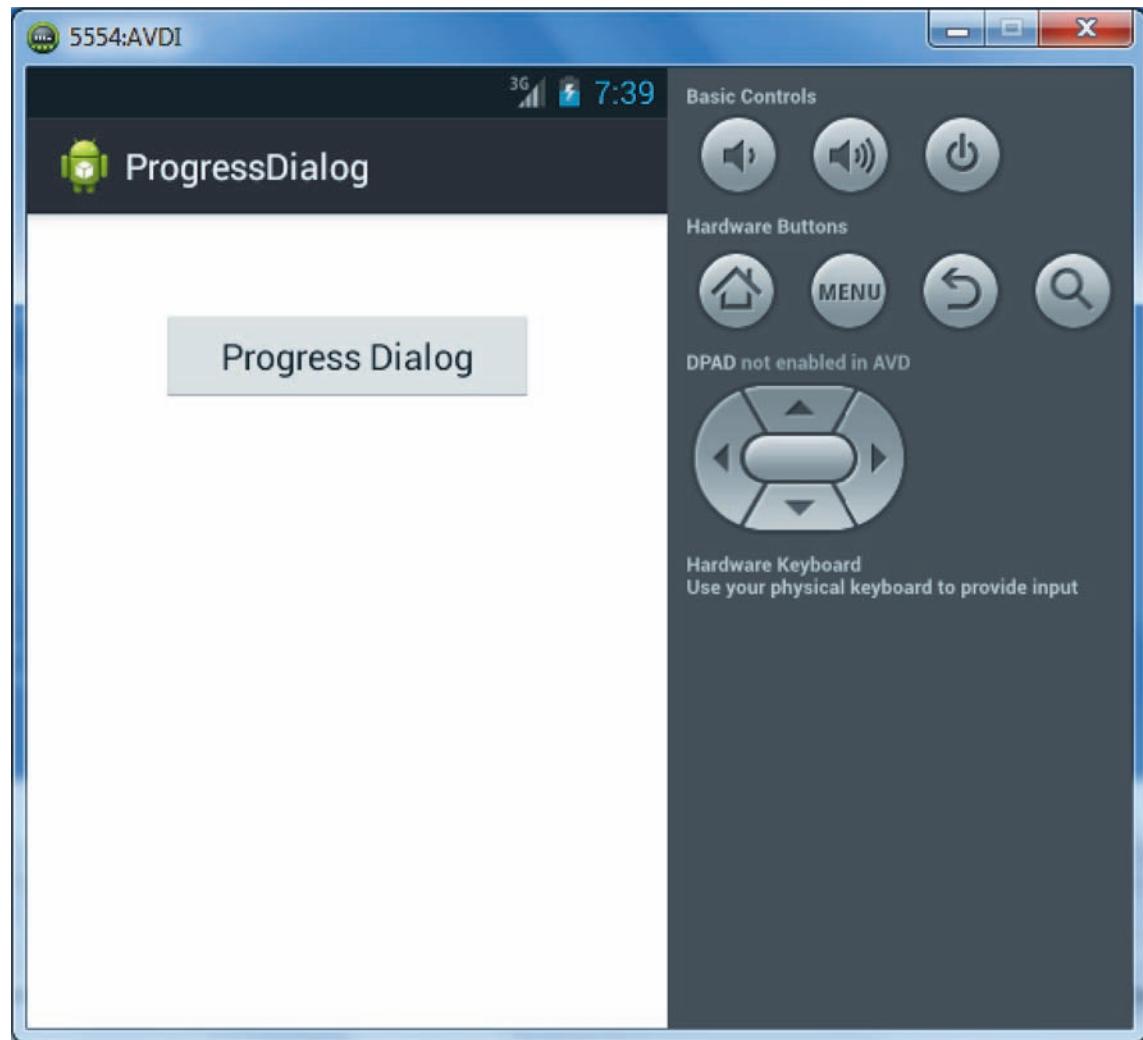


Figure 4.24: Application Displaying the Progress Dialog Button

Clicking the button will display the output as shown in figure 4.25.

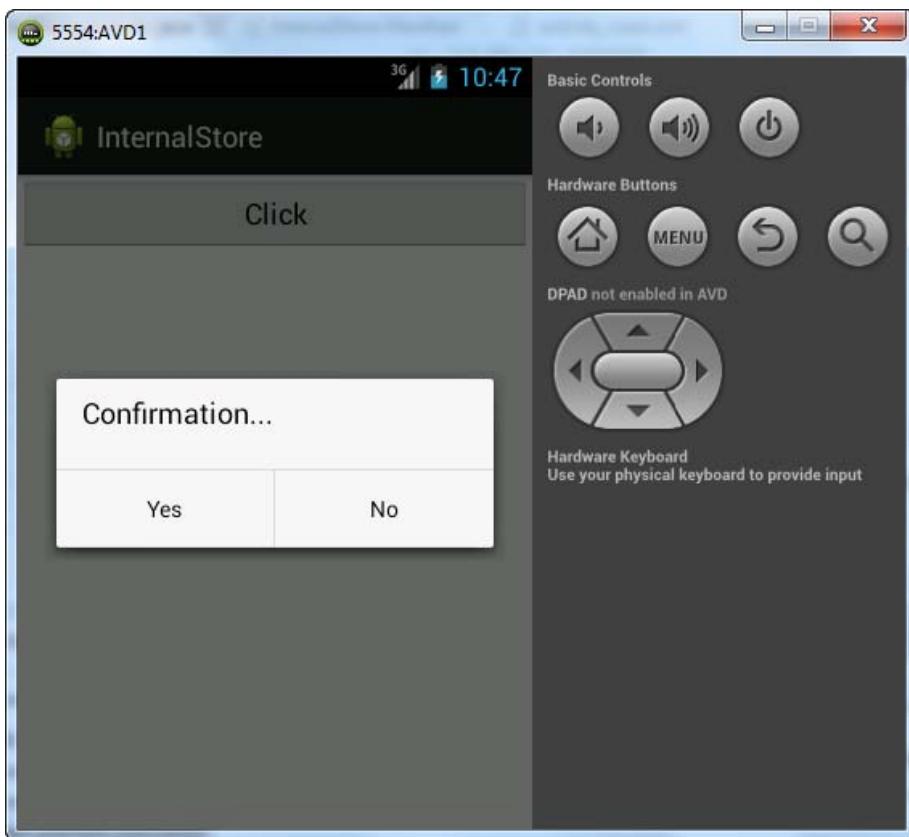


Figure 4.25: Progress Bar

→ Alert Dialog

Like dialogs, alerts also require an action to be performed by the user. Depending on the severity of the alert, the display differs. Most alerts do not require a title. The content can be displayed in a brief sentence that is a question or a statement. The action buttons need to appear below the statement. In rare cases, a title bar is required for an alert. The developer needs to provide these only when the alert is displaying important information, such as, data loss, loss of connectivity, and so on. The alert should again display a short statement or question and the user should be able to quickly make the next decision using the action buttons.

The dialog display includes a title region, content area, and action buttons. The title region displays the dialog content. For instance, it can make a request for an action from the user. The content area differs according to the type of dialog displayed. For instance, it could display a checkbox or an alert that requires the user to decide the next action. Action buttons typically require the user to perform an action, for instance, select **Cancel** or **OK**. There can also be action buttons that require specific actions. In such cases, the dismissive action should be displayed on the right, and the affirmative action should be displayed on the left as shown in figure 4.26.



Figure 4.26: Dismissive and Affirmative Action

The steps to create an application named **AlertDialog** are as follows:

1. Open **Eclipse IDE**.
2. Create a project named **AlertDialog**.
3. Navigate to **res → layout → activity_main.xml**.
4. Modify the code of **activity_main.xml** file as shown in code snippet 33.

Code Snippet 33:

```
<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/buttonAlert"
        android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content"  
        android:layout_alignParentTop="true"  
        android:layout_centerHorizontal="true"  
        android:layout_marginLeft="50dp"  
        android:layout_marginRight="50dp"  
        android:layout_marginTop="31dp"  
        android:text="Alert Dialog"/>  
  
</RelativeLayout>
```

5. Navigate to **src → com.example.alertdialog → MainActivity.xml** file.
6. Modify the code of **MainActivity.xml** file as shown in code snippet 34.

Code Snippet 34:

```
package com.example.alertdialog;  
  
import android.os.Bundle;  
import android.app.Activity;  
import android.app.AlertDialog;  
import android.content.Context;  
import android.content.DialogInterface;  
import android.view.Menu;  
import android.view.View;  
import android.view.View.OnClickListener;  
import android.widget.Button;  
import android.widget.Toast;  
  
public class MainActivity extends Activity {  
  
    final Context context = this;  
    private Button button;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
}
```

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

    // add button listener
    button.setOnClickListener(new OnClickListener() {

        @Override
        public void onClick(View arg0) {
            AlertDialog.Builder alertDialogBuilder = new
            AlertDialog.Builder(
                context);
            // set title
            alertDialogBuilder.setTitle("Confirmation..");
            // set dialog message
            alertDialogBuilder
                .setMessage("Stay in this activity!")
                .setCancelable(false)
                .setPositiveButton("Yes",
                    new DialogInterface.
                    OnClickListener() {
                        public void
                        onClick(DialogInterface dialog,
                            int id) {
                            dialog.cancel();
                        }
                    })
                .setNegativeButton("No",
                    new DialogInterface.
                    OnClickListener() {
                        public void
                        onClick(DialogInterface dialog,
                            int id) {
                            Toast.
                           .makeText(MainActivity.this,
                                "Good
                                Bye..", Toast.LENGTH_LONG)
                        }
                    })
            alertDialogBuilder.show();
        }
    });
}
```

```
.show( );
    MainActivity.this.finish();
}
}
// create alertDialog
AlertDialog alertDialog=alertDialogBuilder.
create();
// show it
alertDialog.show();
}
}
}
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
}
```

Figure 4.27 displays the output of the application.

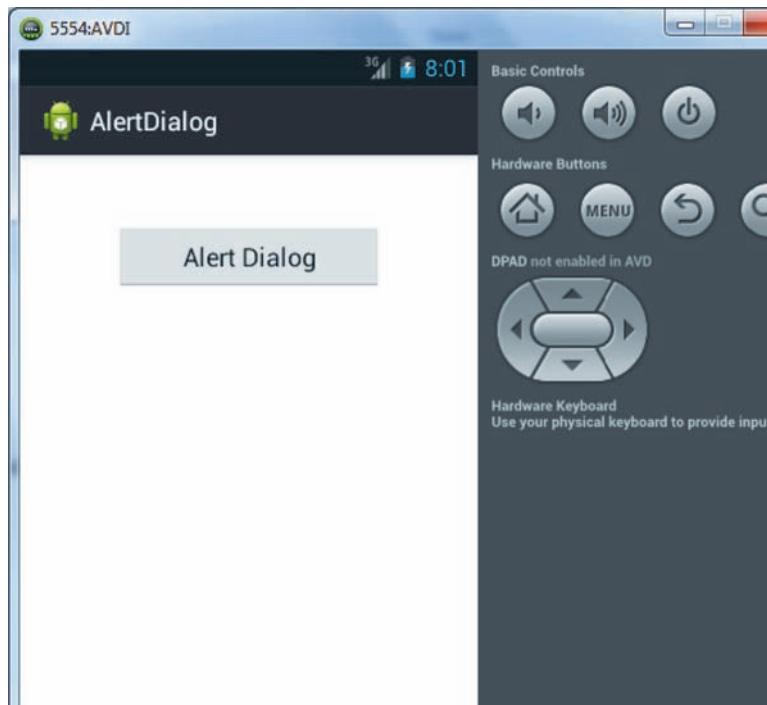


Figure 4.27: AlertDialog

Clicking **Alert Dialog** will display the output as shown in figure 4.28.

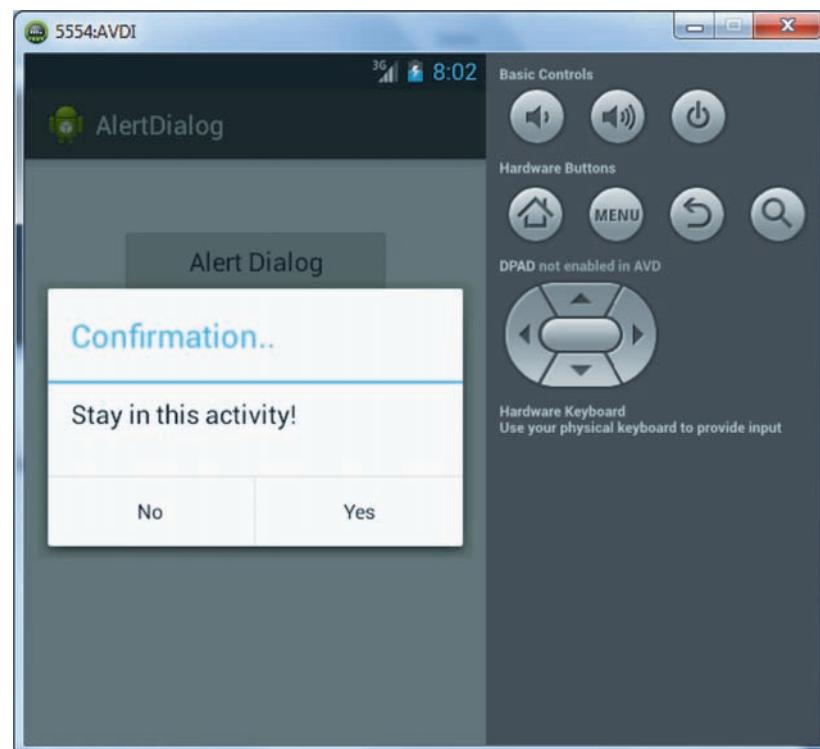


Figure 4.28: Alert Dialog Box

→ Popup Notifications

Popup are smaller versions of dialogs. They need to be designed in such a way that the user only needs to make a selection to move forward, or click outside the popup to exit it as shown in figure 4.28. They do not need to display action buttons or statements.

The steps to create a **PopupNotification** application are as follows:

1. Open **Eclipse IDE**.
2. Create a project named **PopupNotification**.
3. Navigate to **res → layout → activity_main.xml** file.
4. Modify the code of **activity_main.xml** file as shown in code snippet 35.

Code Snippet 35:

```
<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/popup"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="31dp"
        android:layout_marginLeft="50dp"
    />
```

```
        android:layout_marginRight="50dp"
        android:text="Popup Notification"/>

    </RelativeLayout>
```

5. Navigate to **src → com.example.popupnotification → MainActivity.java** file.
6. Modify the code as shown in code snippet 36.

Code Snippet 36:

```
package com.example.popupnotification;

import android.os.Bundle;
import android.app.Activity;
import android.app.AlertDialog;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.view.View.OnClickListener;
import android.content.Context;
import android.content.DialogInterface;

public class MainActivity extends Activity {

    Button back;
    final Context context = this;

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

        back = (Button) findViewById(R.id.popup);

        back.setOnClickListener(new OnClickListener() {
```

```
    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
        AlertDialog.Builder alert_dialog = new
AlertDialog.Builder(
            context);
        alert_dialog.setTitle("Confirmation...");
        alert_dialog.setNegativeButton("Cancel",
            new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface
dialog, int which) {
                    }
                });
        alert_dialog.setPositiveButton("Leave current",
            new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface
dialog, int which) {
                    finish();
                    }
                });
        alert_dialog.show();
    }
}

@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;  
    }  
}
```

7. Click Run → Run.

The output of the application is as shown in figure 4.29.

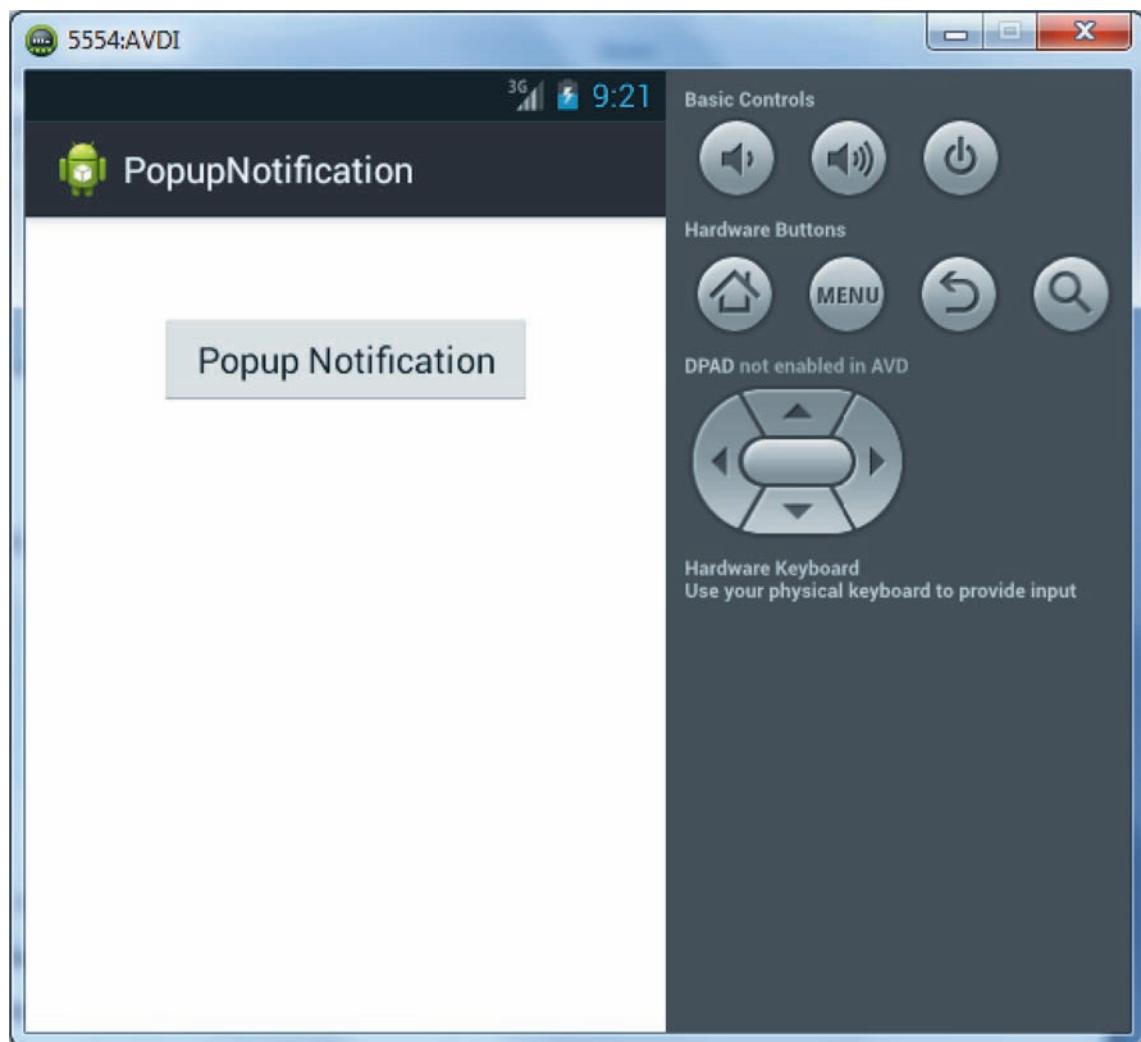


Figure 4.29: Popup Notification Application

On clicking **Popup Notification** the output is as shown in figure 4.30.

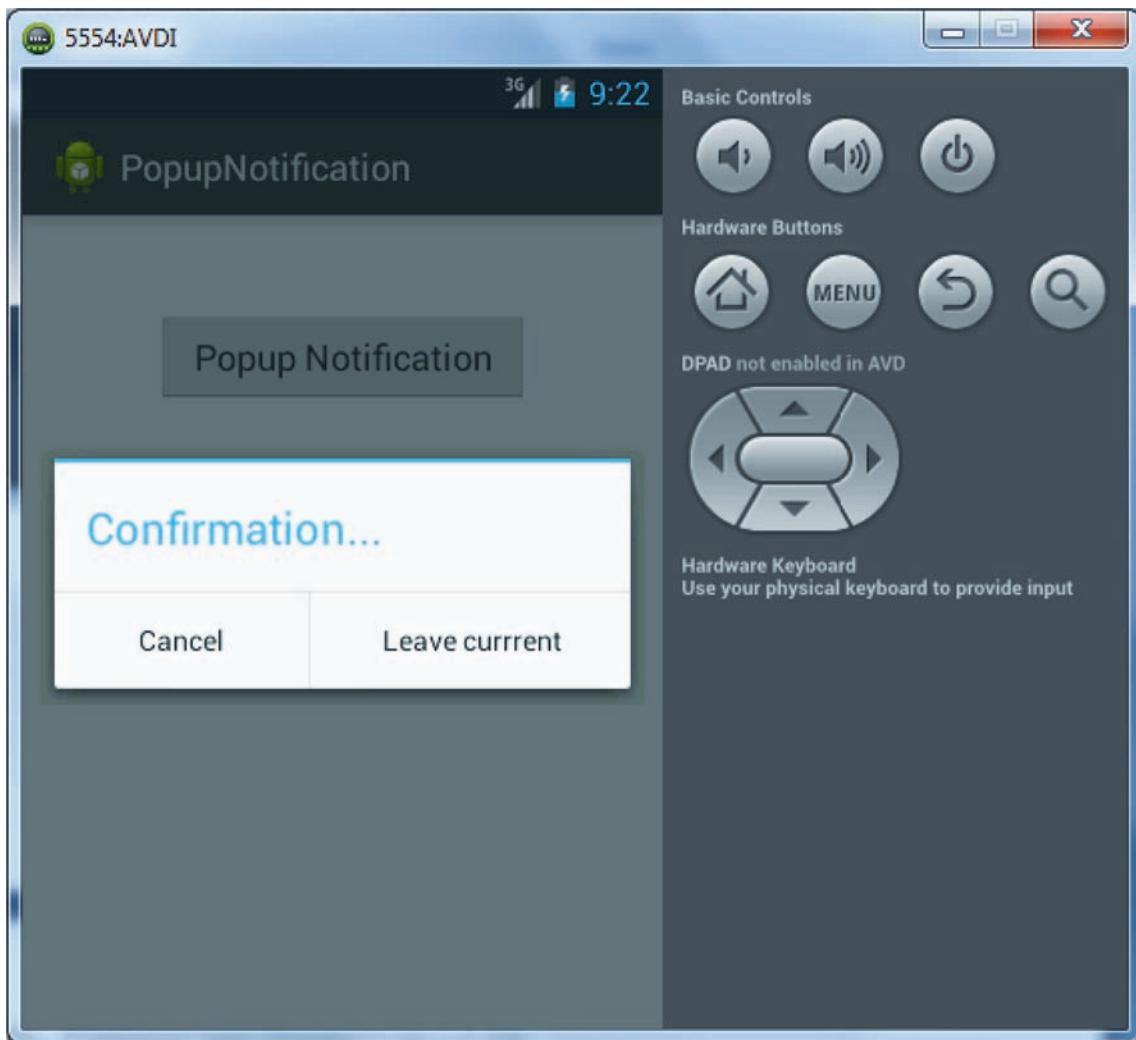


Figure 4.30: Popup Notification

→ Toast

A toast is a popup that displays feedback for an activity. For instance, if you exit an SMS that is half complete, a toast displays the message, “Draft Saved” to inform the user that the message can be edited at a later time. Toasts are timed and exit automatically after the timeout period. The message that pops up on the window surface is referred to as the toast notification. It does not alter user’s current activity. It is visible for a few seconds and automatically disappears. It can be created from a `Service` or an `Activity` class. A `Toast` notification is created by declaring an object of the `Toast` class.

The static method, `makeText()`, is used to create a `Toast` object. The method accepts four arguments.

Syntax for `makeText()` method is:

Syntax:

```
public static Toast.makeText(Context c, CharSequence str, int duration)
```

where,

c: represents the context to be used and can be application of Activity object.

str: represents the message to be displayed.

duration: represents the length of time it will be visible and can either be LENGTH_SHORT or LENGTH_LONG.

- The show() method is used to display the Toast object.
- The setGravity – is used to position the message in screen. By default it appears at bottom centered.

The steps to create an application named **ToastApplication** are as follows:

1. Open **Eclipse IDE**.
2. Create a project named **ToastApplication**.
3. Navigate to **res → layout → activity_main.xml** file.
4. Modify the code in **activity_main.xml** file as shown in code snippet 37.

Code Snippet 37:

```
<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_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="151dp"
        android:text="Click for Toast Notification" />

    </RelativeLayout>

```

5. Navigate to **src → com.example.toastapplication → MainActivity.java** file.
6. Modify the code of **MainActivity.java** file as shown in code snippet 38.

Code Snippet 38:

```

package com.example.toastapplication;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {

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

        Button b = (Button) findViewById(R.id.button1);
        b.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                Toast.makeText(getApplicationContext(),
                        "This is a Toast Notification", Toast.
LENGTH_LONG).show();
            }
        });
    }
}

```

```
    }
```



```
    @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;
    }
}
```

Once you execute the application the output will be as shown in figure 4.31.

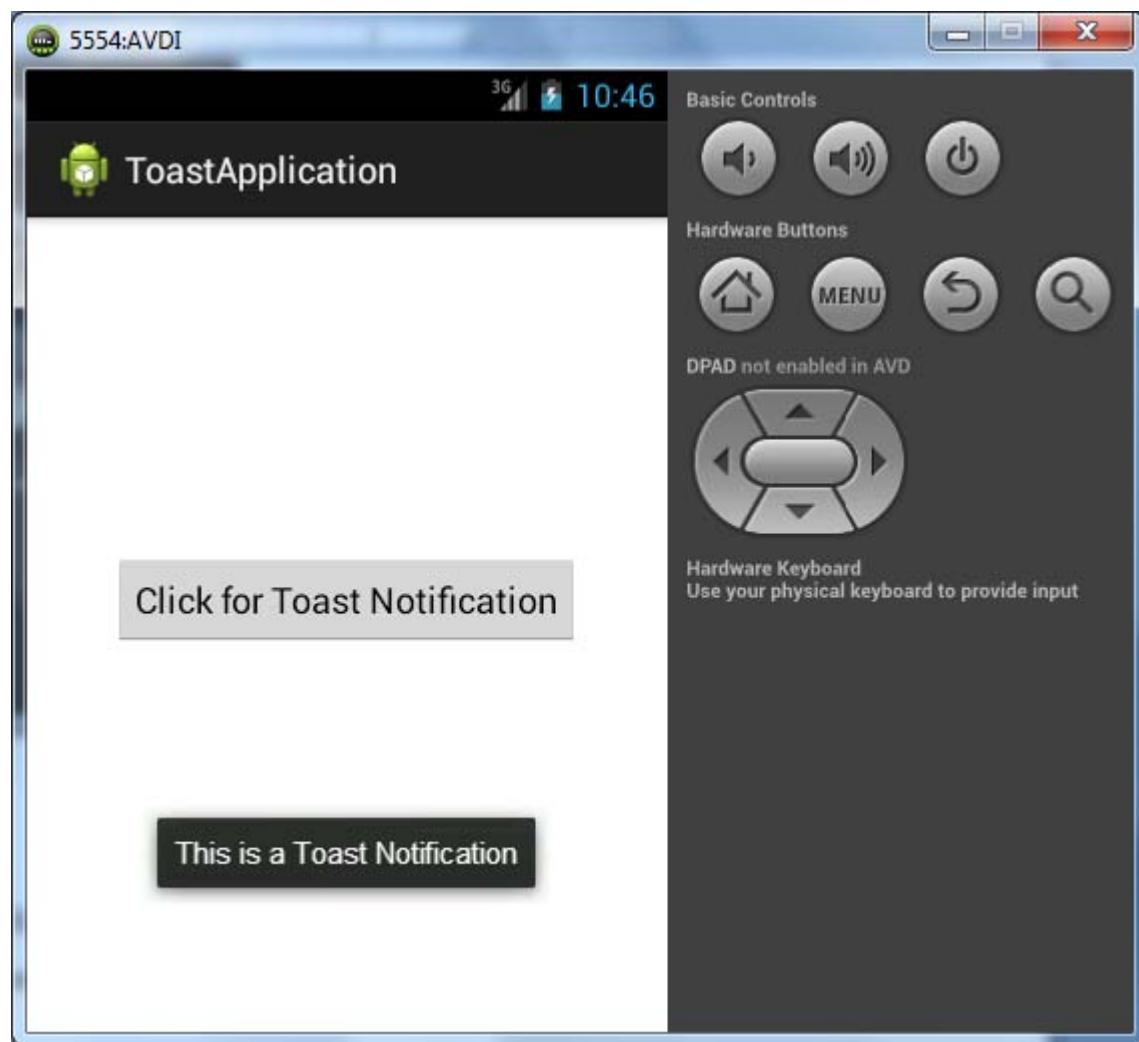


Figure 4.31: Toast Application

→ Status Notifications

Status notifications are alerts that are displayed as icons in the status bar. They typically appear with a pull-down feature. When the user clicks the pull-down, the notification window displays the related alert. For instance, when the user postpones a calendar item, the calendar icon appears on the status bar. When the user clicks the icon, the calendar item is displayed. Notifications are managed by the Notification Manager. The Notification Manager ensures that the status bar icons are updated regularly. A status bar notification displays an icon on the status bar along with a message. When the notification is chosen, an Intent is sent by Android to launch the activity. The status bar notification can be initiated by an Activity or Service class.

A status bar notification is created when the background service wants to inform the user about an event that requires the user to respond.

The steps to create a `StatusNotification` application are as follows:

1. Start **Eclipse IDE**.
2. Create a project named **StatusNotification**.
3. Navigate to **res → layout → activity_main.xml** file.
4. Modify the code of **activity_main.xml** as shown in code snippet 39.

Code Snippet 39:

```
<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/showNotification"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:text="Show Notification"/>

    </RelativeLayout>
```

5. Navigate to **src → com.example.statusnotification → MainActivity.java** file.
6. Modify the code of **MainActivity.java** file as shown in code snippet 40.

Code Snippet 40:

```
package com.example.statusnotification;

import android.os.Bundle;
import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.graphics.Color;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.LinearLayout;
import android.content.Context;

public class MainActivity extends Activity {

    Button showNotification;
    final Context context = this;

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

```
showNotification=(Button)findViewById(R.id.showNotification);  
showNotification.setOnClickListener(newOnClickListener() {  
    @Override  
    public void onClick(View v) {  
        // TODO Auto-generated method stub  
  
        showNotification();  
  
    }  
});  
  
}  
  
@SuppressWarnings("deprecation")  
public void showNotification()  
{  
    NotificationManager notificationManager  
= (NotificationManager) MainActivity.this.getSystemService(Context.  
NOTIFICATION_SERVICE);  
  
    Intent intent = new Intent(context, LinearLayout.class);  
    intent.setFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);  
    long when = System.currentTimeMillis();  
  
    PendingIntent contentIntent = PendingIntent.  
getActivity(context, 0, intent, 0);  
  
    @SuppressWarnings("deprecation")  
    Notification notification = new Notification(R.drawable.ic_  
launcher, "New Message", when);  
    //Notification notification = new Notification.Builder(this)  
    //        .setContentTitle("New Message" + "Content")  
    //        .setSmallIcon(R.drawable.ic_launcher)  
    //        .addAction(R.drawable.ic_launcher, "More",  
contentIntent).build();
```

```
notification.ledARGB=Color.RED;
notification.ledOffMS=300;
notification.ledOnMS=300;

notification.defaults |=Notification.DEFAULT_SOUND;
notification.defaults |=Notification.DEFAULT_LIGHTS;
notification.flags |=Notification.FLAG_AUTO_CANCEL |
Notification.FLAG_SHOW_LIGHTS;
notification.setLatestEventInfo(context,
"Notifcation","This is StatusBarNotification", contentIntent);
notificationManager.notify(10001, notification);
}

@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;
}
}
```

On execution of the application the output is as shown in figure 4.32.

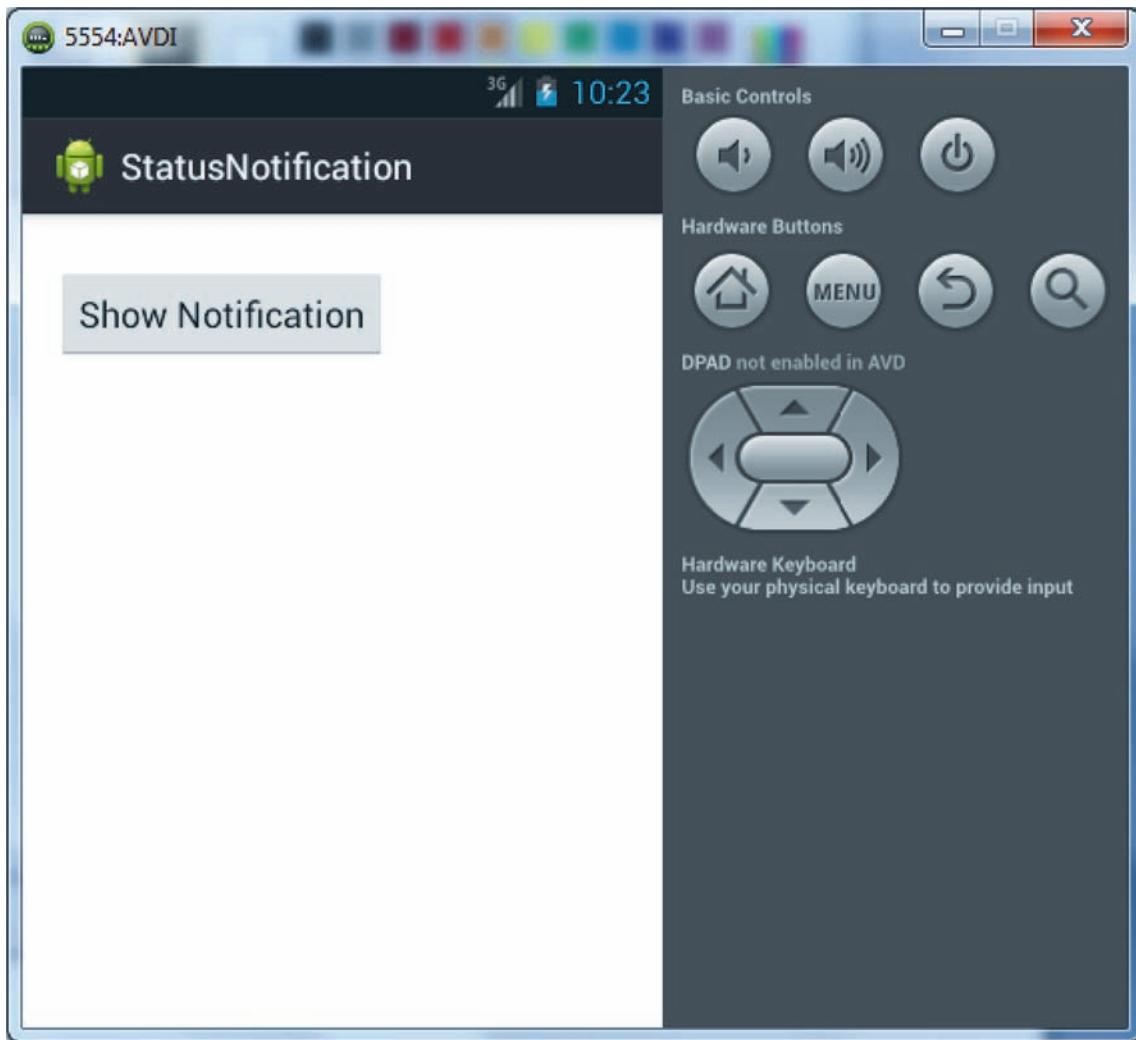


Figure 4.32: StatusBar Notification – Output

On clicking **Show Notification** the output is as shown in figure 4.33.

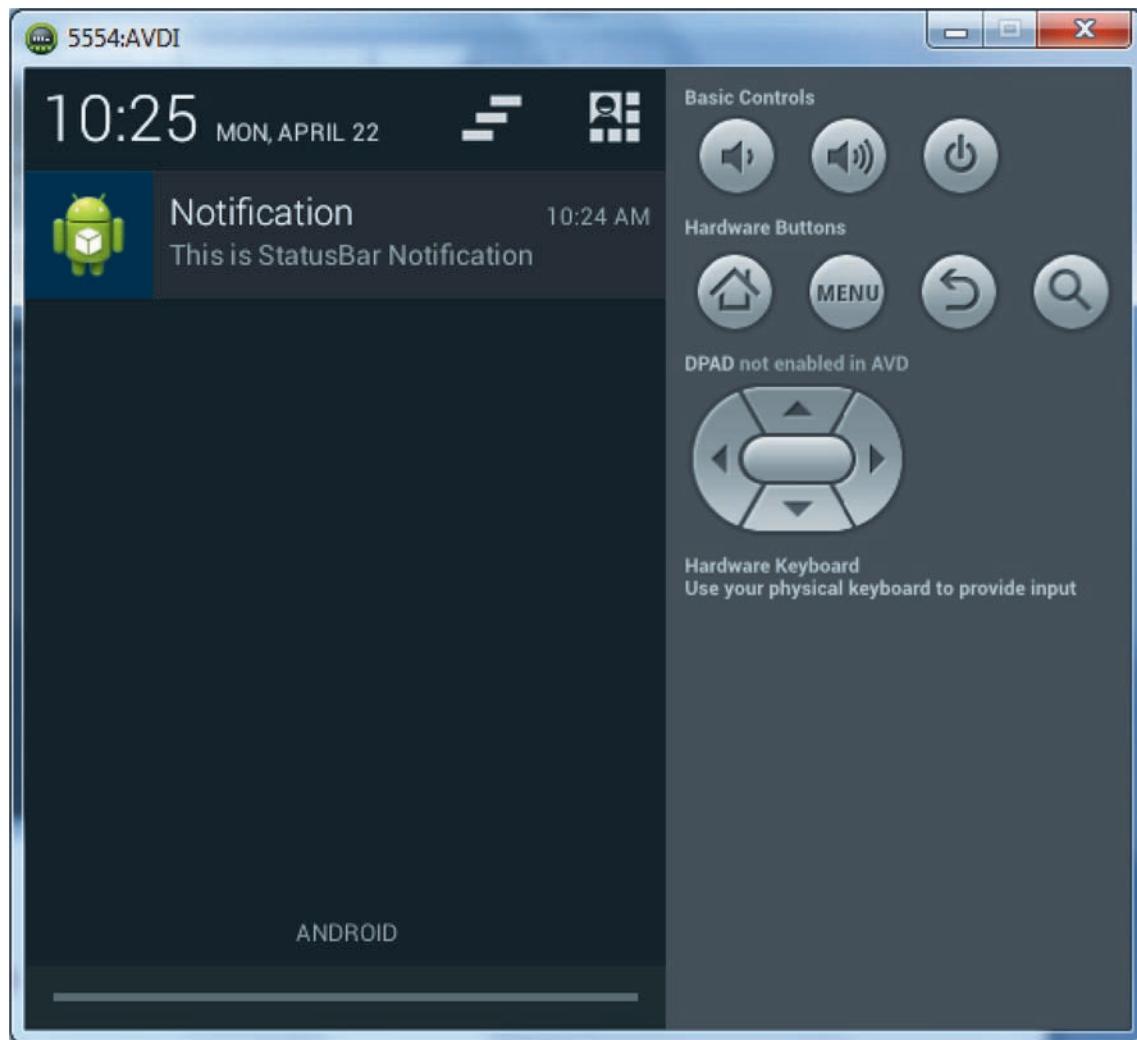


Figure 4.33: StatusBar Notification

4.5 Android Security Model

At the center of the Android security framework is the Linux kernel. The Linux kernel offers a robust and well-tested security system. The main security features that the kernel offers are as follows:

- Permissions tailored to user needs
- Separating processes
- Securing inter-process communication (IPC)
- Eliminating parts of the kernel that are not secure
- Separating resources of one user from another

The Android security model also offers several user security options and protection from third-party applications. It is important to understand permissions, the Application Sandbox, and user security options to comprehend the Android security model.

→ Permissions

Another important aspect of the Android Security framework is the concept of permissions. The basic rule of the security framework is that none of the applications have permissions to execute operations that might severely affect other applications. Also, no application has permissions to perform an operation that could affect the OS or any user information, such as, e-mail or SMS.

Permissions also encompass user permissions for application access, application certification, user identifications, access to files, and root permissions.

When the application is installed, it states the permissions that are additionally required for functions that are not authorized by the default permissions. The user needs to provide consent to approve permissions for these capabilities.

The developer has to sign a certificate with a private key that identifies the author of the application. The certification enables the system to approve or reject access for applications that request permissions at the signature level. Based on the certification, the system can also determine whether or not to approve an application requesting permission for being granted an identity similar to a different application.

When a package is installed on a device, the OS delegates a unique user identification for the package. This ID is valid for the package until it is deleted from the device. When the package is installed on another device, it would have a different user ID. Different packages can only be executed in a single process using the *sharedUserId* attribute in the Android Manifest file. This assigns both packages the same user ID and related permissions thereby considering the two different packages as a single application.

The application's data would also be protected by the user ID and would be inaccessible by other applications. The developer can set the read or write flags when a new file is created to enable other applications to access the file. Although the file belongs to a specific application, setting read/write flags enable all other applications access to the file. File system permissions ensure that one user will not be able to access or change another user's data.

Root permissions set for devices ensure that neither an application, nor a user can alter the OS, another application, or the kernel. The Root user, on the other hand, can access all applications and their data. When a user changes root permissions, the security of the system is compromised. On the other hand, a developer can modify the root permissions on a device to install another OS.

→ Separating Application Processes and Resources

A unique aspect of the Linux kernel is that it separates processes and resources for one application from another, while providing the ability for applications to share resources. This is called the Application Sandbox wherein unique user identification is designated to

each application, and several applications are executed with common user permissions using group IDs. As a rule, applications cannot communicate and have restricted access to the OS. The OS keeps track of actions that applications perform, and restrict them from security breaches. When it comes to resource access, the additional capabilities that the application requires must be defined in the Android Manifest file.

→ User Security Options

Device management, secure password, data encryption, certification based system, and Virtual Private Network (VPN) to ensure secure network connectivity are the features that Android provides to ensure user security.

Device Management: Administrators can perform functions like enforcing security policies using certain applications, such as, e-mail, and can delete data on devices that were lost or were stolen.

Secure Password: While accessing a device, the Android OS will verify a password provided by the user.

Data Encryption: The Android security framework enables all of the user's data to be encrypted by providing an encryption key. This enables the generation of a key from the user's password, which blocks unauthorized access to user data.

Certification Based System: Android comes with an existing list of Certificate Authorities (CAs) for enabling Secure Sockets Layer (SSL) connections. The later Android versions (4.0 onwards) also enable users to add or disable CAs.

VPN: Android comes with a VPN client that enables applications to access the Internet only through the VPN.

Protection Relating to Third-Party Applications: Android has a system of informing users when their devices are communicating with third-party applications. The different kinds of permissions that the third-party application requests are displayed clearly to the user at the time of installation.

Android also informs the user about functions that might create a cost to the user. It does not provide low-level SIM card access to third-party applications. It also prevents access to sensitive devices, such as, camera and microphone. Android protects information about user preferences, device usage, and so on. It offers users the ability to verify applications and provides options for managing digital rights.

4.6 Check Your Progress

1. When setting preferences using an XML file, the XML file should contain a list of _____.

(A)	Preference Objects	(C)	Preference Managers
(B)	PreferenceCheckBoxes	(D)	Preference Subclasses

2. The developer can set a preference to open an activity by adding:

(A)	A Title	(C)	A Subscreen
(B)	An Intent	(D)	A Fragment

3. Features of external storage include:

(A)	Anytime access to the file system	(C)	Used for files without access restrictions
(B)	Upon uninstall all application related files are deleted	(D)	Secure because only the related application can access files

4. A toast is a popup that displays _____ for an activity.

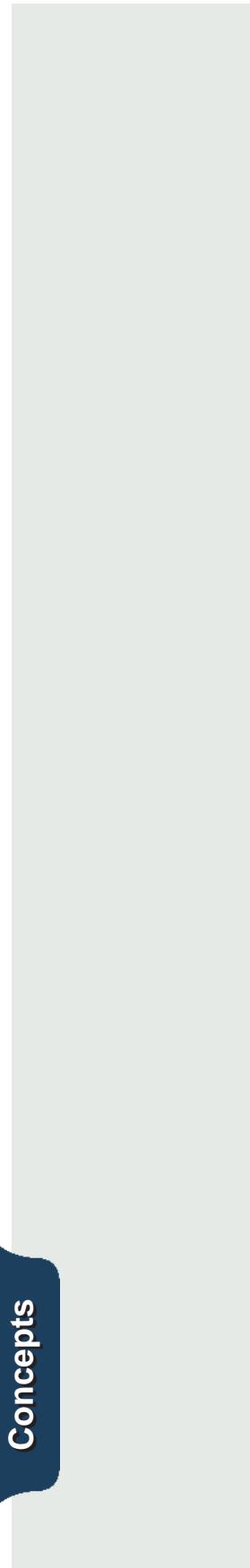
(A)	Status	(C)	Title
(B)	Actions	(D)	Feedback

5. Android offers permissions for which of the following application operations?

(A)	Additional Capabilities Access	(C)	User ID
(B)	Device Administration	(D)	VPN

6. Identify the functions of the Android Security Model.

(A)	Setting Preferences	(C)	Defining Permissions
(B)	Secure Inter-Process Communication (IPC)	(D)	Defining Storage Space



4.6.1 Answers

1.	A, D
2.	B
3.	B
4.	D
5.	A, C
6.	B, C



- Preferences are a subclass that helps users to customize their applications. A developer can specify preferences using an XML file or using code.
- Title, Intents, and sub screens are the different ways that helps to group settings.
- The Android file system offers internal and external storage. It offers several options to write, save, or delete files to either kind of storage.
- Data saved on device's internal storage cannot be accessed by other applications. Data can also be saved to an external storage such as SD card which can be modified and shared by other users.
- Android's notifications enable an app to inform the user about events which can be a calendar event or alerts.
- Notifications include dialogs and alerts, popups, toasts, and status notifications.
- A dialog is displayed in front of an activity as a small window and can be a DatePickerDialog, TimePickerDialog, ProgressDialog, or AlertDialogs.
- Popup are smaller versions of dialogs which are designed in such a way so that users can make a selection to move forward or click outside the popup. Toast is a popup that displays feedback for an activity.
- Status Notifications are alerts that are displayed as icons in the status bar.
- The Android security model provides several security features, such as, permissions, the Application Sandbox, several user security options, and protection from third-party applications.



Try it Yourself

Samantha wants to develop an Android application in which user will input the date of birth and output will be her current age.

She also wants to design and develop a Login application where username and password will be stored in the shared preference.

JELLYBEAN
ICE CREAM SANDWICH

Session - 5

More UI Elements

Welcome to the session, **More UI Elements**.

This session discusses in detail the components of Android User Interface (UI). It describes the advanced (UI) components, Adapters, and dialogs.

In this session, you will learn to:

- ➔ Explain the use of adapters
- ➔ Identify the different types of adapters
- ➔ Describe the advanced and complex UI Components
- ➔ Explain the use of custom dialogs



5.1 Adapters

An Adapter is an object that acts as a bridge between the UI Components such as List view, Grid view, and so on and the underlying data source. The underlying data source can be an Array, database, and so on that fill data to the UI component. Adapter is responsible for providing view for every element of the data source.

Adapters can be of different types. They differ based on their data source and manipulation complexity that they present to the user. The different types of adapters present in Android are as follows:

1. BaseAdapter
2. SimpleAdapter
3. ArrayAdapter
4. SimpleCursorAdapter
5. CursorAdapter
6. ResourceCursorAdapter
7. SpinnerAdapter
8. SimpleCursorTreeAdapter
9. CursorTreeAdapter
10. HeaderViewListAdapter
11. WrapperListAdapter

→ BaseAdapter

BaseAdapter is a common implementation for the adapter which can be used for both ListView and Spinner. It is an abstract base class for the adapter interface. The developer can implement his/her own adapters using BaseAdapter.

→ SimpleAdapter

A SimpleAdapter helps to map static data to the views defined in the XML. The data can be mapped to the view by using ArrayList of Maps and each element in the array will be represented as a separate row in the view.

→ ArrayAdapter

ArrayAdapter is backed by an array of objects to load the data to the UI view. In other words, it is used to bind an array of data to the view. It overrides the `getView()` method to inflate, populate, and return a custom view for the provided array data.

→ **SimpleCursorAdapter:**

An adapter which maps columns from a cursor to TextViews or ImageViews defined in an XML file. The developer can specify which columns to be displayed, in which views the developer wants to display the columns, and the XML file that defines the appearance of these views.

→ **CursorAdapter**

This Adapter that is used for exposing the data from a Cursor to a ListView object using the column named `_id`.

→ **ResourceCursorAdapter**

`ResourceCursorAdapter` is very similar to the `CursorAdapter`, which doesn't have a new `View` method and is used for simple views. It is used to create views defined in an XML file. This Adapter is deprecated in API Level 11.

→ **SpinnerAdapter**

`SpinnerAdapter` acts as a bridge between the spinner component and the data source.

`SpinnerAdapter` allows displaying of data in the following two ways:

1. Displays data in the spinner view.
2. Displays data in the drop-down list when the spinner is pressed.

→ **SimpleCursorTreeAdapter**

This Adapter can be used to map column data from the Cursor to the TextView or ImageView as defined in the XML file. The developer can specify the required columns and separate child views to display the specified content. Binding is done using the `setViewValue()` method of the `SimpleCursorTreeAdapter`. `ViewBinder`. The method returns a boolean value which can be used to determine whether binding has been successful.

→ **CursorTreeAdapter**

This Adapter can be used to display data in an expandable list view by using the data from the cursor. In this, the top-level Cursor exposes the group and the `getChildrenCursor(Cursor)` method returns cursors which exposes the child elements within the particular group.

Note - The cursor must include a column named `_id`, otherwise this class won't work.

→ **HeaderListAdapter:**

`HeaderListAdapter` can be used to implement a `ListView` having a header at the top.

→ **WrapperListAdapter:**

`WrapperListAdapter` can be used to wrap another `ListAdapter`. The method, `getWrappedAdapter()` is invoked for retrieving the wrapped adapter.

5.1.1 Working with `BaseAdapter`

The following section explains the development of an application that displays a list view. The user can make multiple selections by using an object of the `BaseAdapter` class.

To develop the `MultiselectionListView` application, perform the following steps:

1. Start Eclipse.
2. Create a new project named `MultiselectionListView`.
3. Navigate to `res → layout` folder.
4. Create an XML file named `linearlayout`.
5. Modify the code in `linearlayout.xml` file as shown in code snippet 1.

Code Snippet 1:

```
<?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="horizontal"
    android:padding="10dp" >

    <CheckBox
        android:id="@+id/chkEnable"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:text="" />
```

```

<TextView
    android:id="@+id/tvTitle"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center_vertical"
    android:layout_marginLeft="10dp"
    android:textSize="16sp" />

</LinearLayout>

```

6. Navigate to **res → layout** folder.
7. Modify the code in **activity_main.xml** file as shown in code snippet 2.

Code Snippet 2:

```

<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" >

    <ListView
        android:id="@+id/list"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/btnShowCheckedItems"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true" />

    <Button
        android:id="@+id/btnShowCheckedItems"

```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:text="View Selected" />

    </RelativeLayout>
```

8. Navigate to **src → com.example.multiselectionlistview** folder.
9. Create a new class named **Product** that will return the product details.
10. Modify the code in the **Product** class as shown in code snippet 3.

Code Snippet 3:

```
package com.example.multiselectionlistview;

public class Product {

    String name;

    public Product(String name) {

        // TODO Auto-generated constructor stub

        this.name = name;

    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

```

@Override
public String toString() {

    // TODO Auto-generated method stub
    return this.name;

}

}

```

11. Navigate to **src → com.example.multiselectionlistview** folder.
12. Create a new adapter class named **MultiSelectionAdapter** which helps the user to select multiple products from a product list displayed as a checkbox items.
13. Modify the code in **MultiSelectionAdapter** as shown in code snippet 4.

Code Snippet 4:

```

package com.example.multiselectionlistview;

import java.util.ArrayList;

import android.content.Context;
import android.util.SparseBooleanArray;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.TextView;

public class MultiSelectionAdapter<T> extends BaseAdapter {

    Context mContext;
    LayoutInflater mInflater;

```



```
ArrayList<T> mList;
SparseBooleanArray mSparseBooleanArray;

public MultiSelectionAdapter(Context context,
    ArrayList<T> list) {

    // TODO Auto-generated constructor stub
    this.mContext = context;
    mInflater = LayoutInflater.from(mContext);
    mSparseBooleanArray = new SparseBooleanArray();
    mList = new ArrayList<T>();
    this.mList = list;
}

public ArrayList<T> getCheckedItems() {

    ArrayList<T> mTempArry = new ArrayList<T>();
    for (int i = 0; i < mList.size(); i++) {
        if (mSparseBooleanArray.get(i)) {
            mTempArry.add(mList.get(i));
        }
    }
    return mTempArry;
}

@Override
public int getCount() {
    // TODO Auto-generated method stub
    return mList.size();
}

@Override
public Object getItem(int position) {
    // TODO Auto-generated method stub
}
```

```
    return mList.get(position);  
}  
  
@Override  
public long getItemId(int position) {  
    // TODO Auto-generated method stub  
    return position;  
}  
  
@Override  
public View getView(int position, View convertView,  
ViewGroup parent) {  
    // TODO Auto-generated method stub  
    if (convertView == null) {  
        convertView = mInflater.inflate(R.layout.  
        linearlayout, null);  
    }  
  
    TextView tvTitle = (TextView) convertView.  
    findViewById(R.id.tvTitle);  
    tvTitle.setText(mList.get(position).toString());  
  
    CheckBox mCheckBox = (CheckBox) convertView  
        .findViewById(R.id.chkEnable);  
    mCheckBox.setTag(position);  
    mCheckBox.setChecked(mSparseBooleanArray.  
    get(position));  
    mCheckBox.setOnCheckedChangeListener(new  
    OnCheckedChangeListener() {  
  
        @Override  
        public void onCheckedChanged(CompoundButton  
        buttonView,  
            boolean isChecked) {  
            // TODO Auto-generated method stub  
    }  
}
```

```
mSparseBooleanArray.put((Integer) buttonView.  
getTag(), isChecked);  
}  
});  
  
return convertView;  
}  
  
}
```

14. Navigate to **src → com.example.multiselectionlistview** folder.
15. Modify the code in **MainActivity** file as shown in code snippet 5.

Code Snippet 5:

```
package com.example.multiselectionlistview;  
  
import java.util.ArrayList;  
  
import android.app.Activity;  
import android.os.Bundle;  
import android.util.Log;  
import android.view.Menu;  
import android.view.View;  
import android.view.View.OnClickListener;  
import android.widget.Button;  
import android.widget.ListView;  
import android.widget.Toast;  
  
public class MainActivity extends Activity implements  
OnClickListener {  
  
    ListView mListview;  
    Button btnShowCheckedItems;  
    ArrayList<Product> mProducts;  
    MultiSelectionAdapter<Product> mAdapter;
```

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    bindComponents();  
    init();  
    addListeners();  
}  
  
private void bindComponents() {  
    // TODO Auto-generated method stub  
    mListview = (ListView) findViewById(android.R.id.list);  
    btnShowCheckedItems = (Button) findViewById(R.  
        id.btnShowCheckedItems);  
}  
  
private void init() {  
    // TODO Auto-generated method stub  
  
    mProducts = new ArrayList<Product>();  
    mProducts.add(new Product("Samsung"));  
    mProducts.add(new Product("iPhone"));  
    mProducts.add(new Product("HTC"));  
    mProducts.add(new Product("LG"));  
    mProducts.add(new Product("Micromax"));  
    mProducts.add(new Product("Sony"));  
    mProducts.add(new Product("Nexus"));  
    mProducts.add(new Product("Acer"));  
    mProducts.add(new Product("Videocon"));  
    mProducts.add(new Product("Karbonn"));  
    mProducts.add(new Product("Nokia"));  
    mProducts.add(new Product("Motorola"));
```

```
mAdapter = new MultiSelectionAdapter<Product>(this, mProducts);  
mListView.setAdapter(mAdapter);  
}  
  
private void addListeners() {  
    // TODO Auto-generated method stub  
    btnShowCheckedItems.setOnClickListener(this);  
}  
  
@Override  
public void onClick(View v) {  
  
    // TODO Auto-generated method stub  
    if (mAdapter != null) {  
        ArrayList<Product> mArrayProducts = mAdapter.  
getCheckedItems();  
        Log.d(MainActivity.class.getSimpleName(),  
"Selected Items: "  
+ mArrayProducts.toString());  
        Toast.makeText(getApplicationContext(),  
"Selected Items: " + mArrayProducts.toString(),  
Toast.LENGTH_LONG).show();  
    }  
}  
  
@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;  
}  
}
```

Figure 5.1 displays the output when the application is executed.

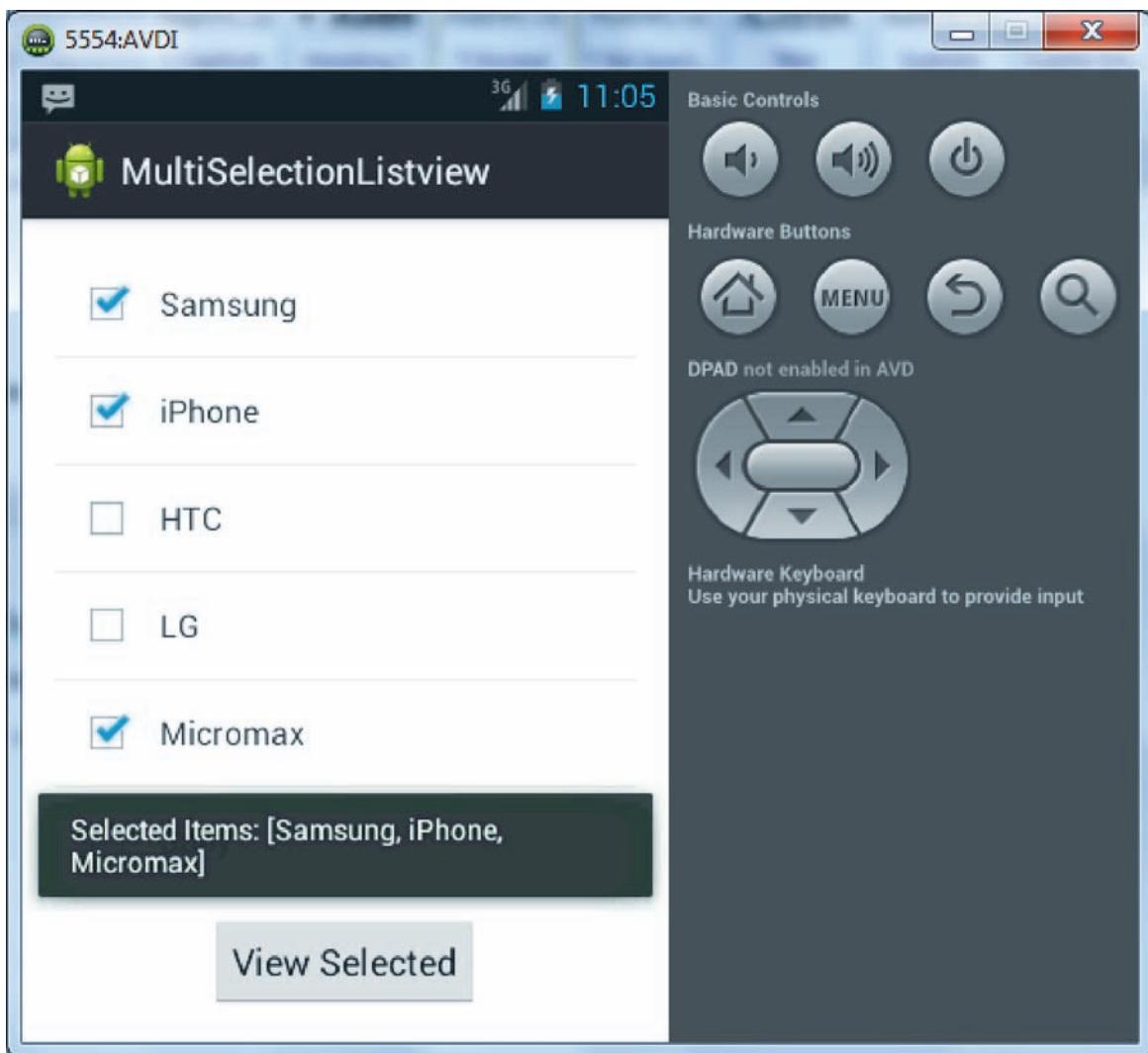


Figure 5.1: Using BaseAdapter class

5.2 Advanced UI Components

UI components act as an interface between the user and the application to input data and to display the expected result. Basic UI components such as buttons, text fields, dialogs, and so on have been already discussed in earlier sessions. In this session, some of the complex and advanced Android UI components that are discussed are as follows:

1. ListView
2. ScrollView
3. TabBar
4. WebView

5. ViewFlipper
6. VideoView

5.2.1 ListView

ListView is used for displaying a list of scrollable data by using an object of the `Adapter` class, which acts as a bridge between view and the data source. The `Adapter` is used for inserting data into the list that is retrieved from an array or a database using a query.

ListView can be implemented in two ways:

Declaring the `ListView` in an XML file as shown in code snippet 6.

Code Snippet 6:

```
<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"
    tools:context=".MainActivity" >

    <ListView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">

    </ListView>

</RelativeLayout>
```

Extending the `ListActivity` class.

The following application named `ListFruit` will display image and text in each row using a `ListView` widget. To develop the application, perform the following steps:

1. Start Eclipse IDE.
2. Create a project named `ListFruit`.
3. Navigate to `res → layout` folder.
4. Modify the code in `main_activity.xml` file as shown in code snippet 7.

Code Snippet 7:

```
<?xml version="1.0" encoding="utf-8"?>  
  
<LinearLayout xmlns:android="http://schemas.android.com/apk/  
res/android"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:padding="5dp" >  
  
    <ImageView  
        android:id="@+id/logo"  
        android:layout_width="50px"  
        android:layout_height="50px"  
        android:layout_marginLeft="5px"  
        android:layout_marginRight="20px"  
        android:layout_marginTop="5px"  
        android:src="@drawable/apple_new" >  
  
    </ImageView>  
  
    <TextView  
        android:id="@+id/label"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="@+id/label"  
        android:textSize="30px" >  
  
    </TextView>  
  
</LinearLayout>
```

The layout file contains an `ImageView` and a `TextView` widget to display the image of the fruit and its name.

5. Navigate to `src → com.example.listfruit` folder.
6. Create a new class named `FruitArrayAdapter`. The class will extend from the `ArrayAdapter` class to read data from the data source.
7. Modify the code in the `FruitArrayAdapter` class as shown in code snippet 8.

Code Snippet 8:

```
package com.example.listfruit;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;

public class FruitArrayAdapter extends ArrayAdapter<String> {

    private final Context context;
    private final String[] values;

    public FruitArrayAdapter(Context context, String[] values) {
        super(context, R.layout.activity_main, values);
        this.context = context;
        this.values = values;
    }

    @Override
    public View getView(int position, View convertView,
                        ViewGroup parent) {
        LayoutInflater inflater = (LayoutInflater) context
                .getSystemService(Context.LAYOUT_INFLATER_
SERVICE);

        View rowView = inflater.inflate(R.layout.activity_main,
                                        parent, false);
        TextView textView = (TextView) rowView.findViewById(
                R.id.label);
        ImageView imageView = (ImageView) rowView.
                findViewById(R.id.logo);
```

```

        textView.setText(values[position]);

        // Change icon based on name
        String s = values[position];

        System.out.println(s);

        if (s.equals("Apple")) {
            imageView.setImageResource(R.drawable.apple_
                new);
        } else if (s.equals("Banana")) {
            imageView.setImageResource(R.drawable.banana_
                new);
        } else if (s.equals("Strawberry")) {
            imageView.setImageResource(R.drawable.
                strawberry_new);
        } else {
            imageView.setImageResource(R.drawable.
                watermelon_new);
        }

        return rowView;
    }
}

```

8. Navigate to **src → com.example.listfruit** folder.
9. Modify the code in **MainActivity** file as shown in code snippet 9. The Adapter class used in this code will be used to load the data.

Code Snippet 9:

```

package com.example.listfruit;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.app.ListActivity;
import android.widget.ArrayAdapter;

```

```
import android.widget.ListView;
import android.widget.Toast;
import android.view.View;

public class MainActivity extends ListActivity {

    static final String[] FRUITS = new String[] { "Apple",
        "Banana",
        "Strawberry", "Watermelon" };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // setContentView(R.layout.activity_main);
        setListAdapter(new FruitArrayAdapter(this, FRUITS));
    }

    @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;
    }

    @Override
    protected void onListItemClick(ListView l, View v, int
        position, long id) {
        // get selected items
        String selectedValue = (String) getListAdapter().getItem(position);
        Toast.makeText(this, selectedValue, Toast.LENGTH_
        SHORT).show();
    }
}
```

Figure 5.2 displays the output after execution of the application.

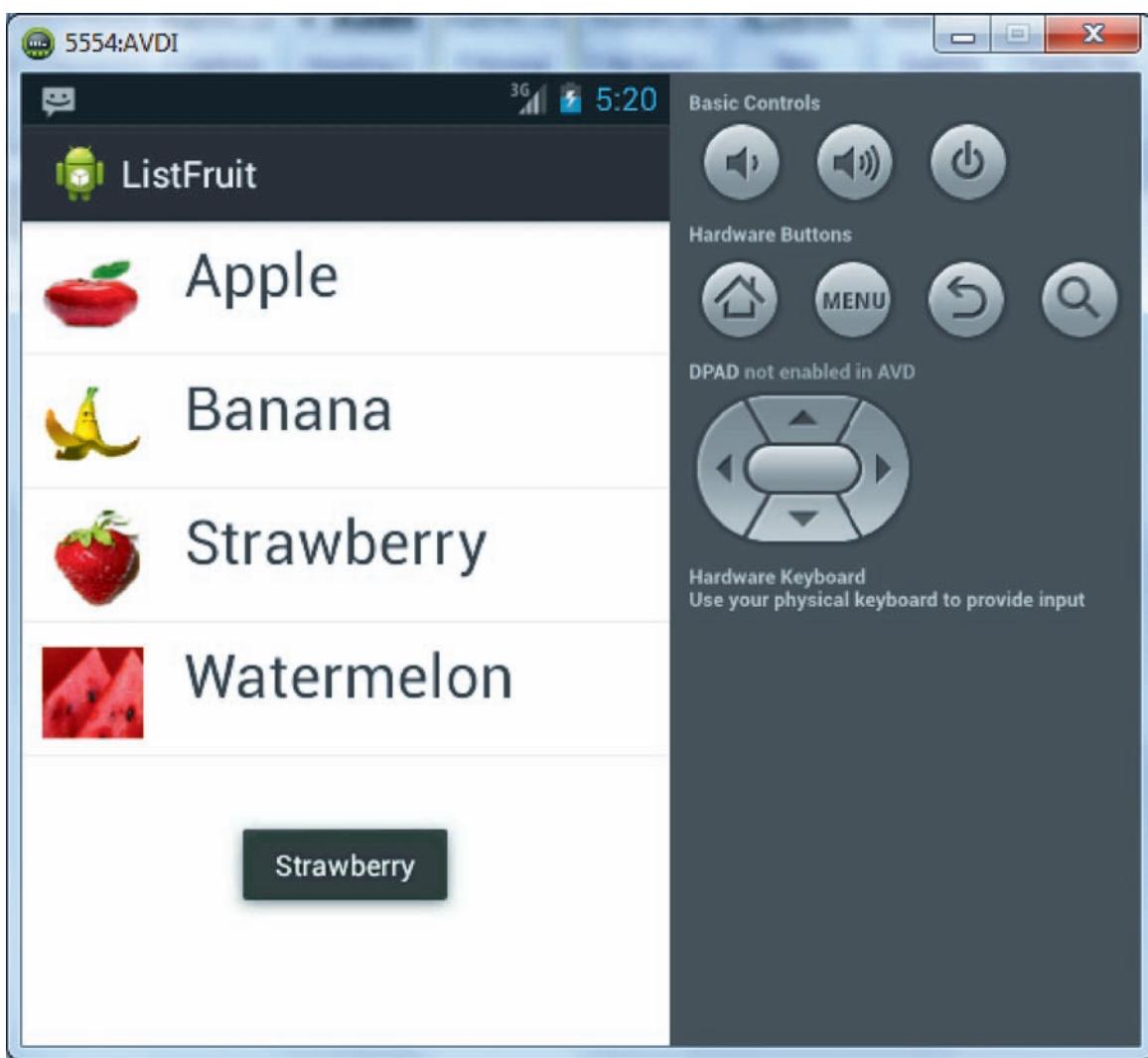


Figure 5.2: ListFruit – Output

5.2.2 ScrollView

`ScrollView` is a container that holds child views and has a property of scrolling when the child items size in the container exceeds the screen size. A `ScrollView` is a `FrameLayout`, meaning that the developer should place one child in it containing the entire contents to scroll. `ScrollView` should not be used with `ListView`, as `ListView` itself takes care of its own scroll.

The following application named `ScrollViewExample` demonstrates the use of `ScrollView`.

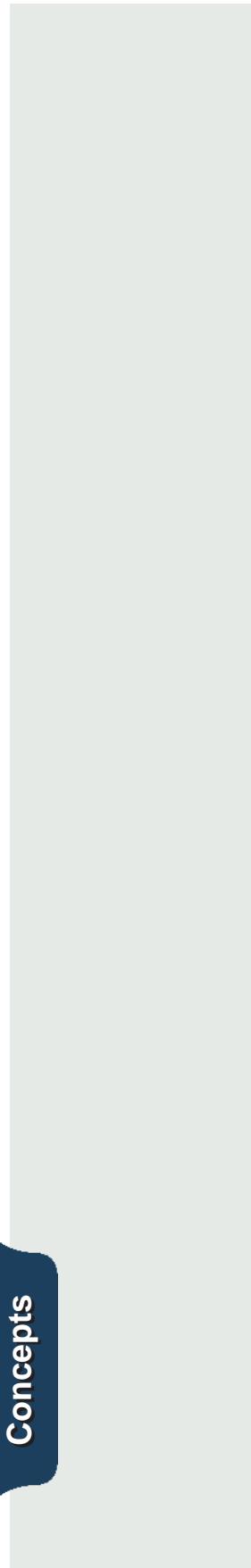
1. Start Eclipse IDE.
2. Create a project named `ScrollViewExample`.

3. Navigate to **res → layout** folder.
4. Modify the code in **activity_main.xml** file as shown in code snippet 10.

Code Snippet 10:

```
<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" >  
  
    <ScrollView  
        android:layout_width="fill_parent"  
        android:layout_height="fill_parent" >  
  
        <LinearLayout  
            android:layout_width="fill_parent"  
            android:layout_height="fill_parent"  
            android:orientation="vertical" >  
  
            <TextView  
                android:layout_width="fill_parent"  
                android:layout_height="50dp"  
                android:text="Red" />  
  
            <TextView  
                android:layout_width="fill_parent"  
                android:layout_height="50dp"  
                android:text="Green" />
```

```
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="50dp"  
    android:text="Yellow" />  
  
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="50dp"  
    android:text="Pink" />  
  
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="50dp"  
    android:text="Violet" />  
  
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="50dp"  
    android:text="Grey" />  
  
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="50dp"  
    android:text="Blue" />  
  
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="50dp"  
    android:text="White" />  
  
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="50dp"  
    android:text="Black" />
```



```
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="50dp"  
    android:text="Brown" />  
  
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="50dp"  
    android:text="Red" />  
  
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="50dp"  
    android:text="Green" />  
  
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="50dp"  
    android:text="Yellow" />  
  
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="50dp"  
    android:text="Pink" />  
  
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="50dp"  
    android:text="Violet" />  
  
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="50dp"  
    android:text="Grey" />
```

```

<TextView
    android:layout_width="fill_parent"
    android:layout_height="50dp"
    android:text="Blue" />

<TextView
    android:layout_width="fill_parent"
    android:layout_height="50dp"
    android:text="White" />

<TextView
    android:layout_width="fill_parent"
    android:layout_height="50dp"
    android:text="Black" />

<TextView
    android:layout_width="fill_parent"
    android:layout_height="50dp"
    android:text="Brown" />

</LinearLayout>
</ScrollView>

</RelativeLayout>
```

5. Navigate to **src → com.example.scrollviewexample** folder.
6. Modify the code in **MainActivity** file as shown in code snippet 11.

Code Snippet 11:

```

package com.example.scrollviewexample;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class MainActivity extends Activity {
    @Override
```

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

Figure 5.3 displays the output after execution of the application.

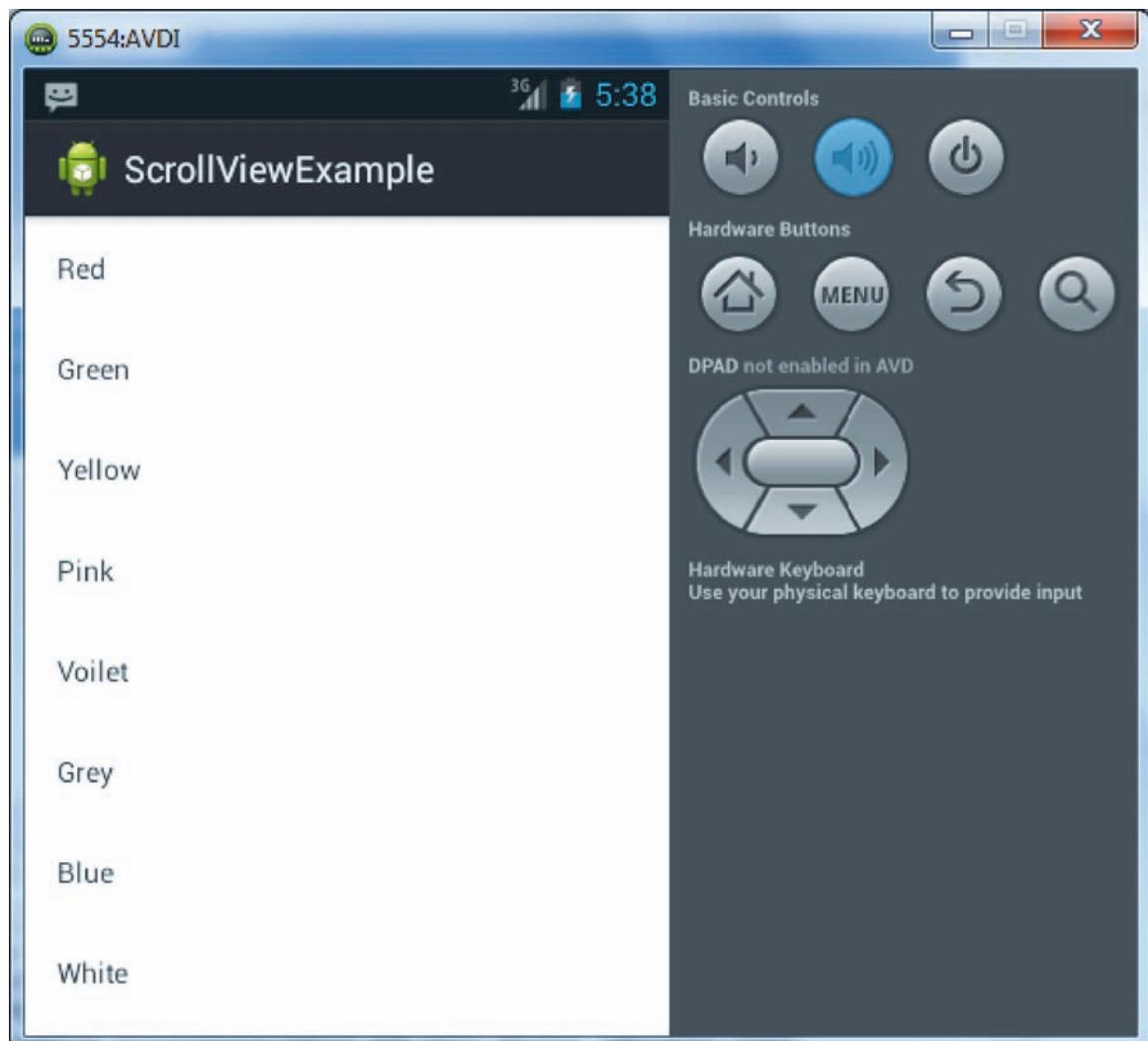


Figure 5.3: ScrollView Display

Note - The content scrolls when the user keeps the mouse button pressed and moves the cursor up and down.

5.2.3 TabBar

TabBar is used to display data in Tab format as is present in android's contacts view. Tabs help to explore and switch between different views and functionalities that are present in the application. It also helps the developer to develop application for users where browsing for categorized data sets are required. It can be scrollable, fixed, or stacked tabs.

TabHost is a container for a tabbed window view. This object holds two children: a set of tab labels that the user clicks to select a specific tab and a FrameLayout object that displays the contents of that page. The following application named **TabBarExample** explains the use of TabView.

To develop the application, perform the following steps:

1. Start Eclipse IDE.
2. Create a project named **TabBarExample**.
3. Navigate to **res → layout** folder.
4. Modify the code in **main_activity.xml** file as shown in code snippet 12. The code displays the use of TabWidget.

Code Snippet 12:

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

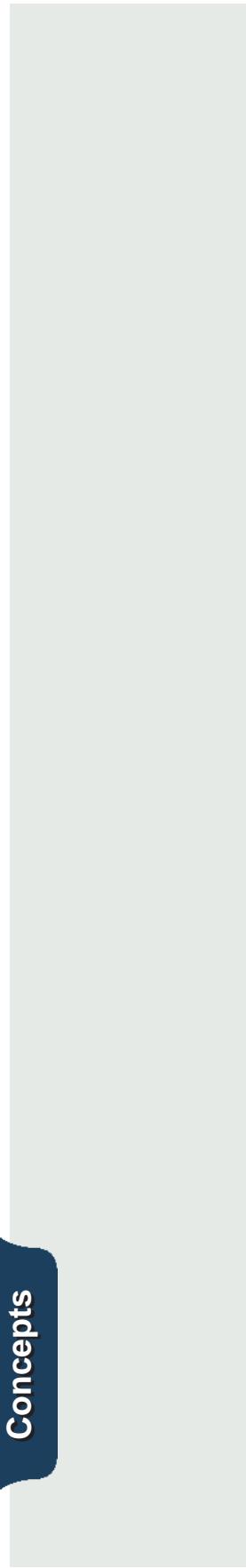
<TabHost xmlns:android="http://schemas.android.com/apk/res/
    android"

        android:id="@+id/tabhost"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >

        <TabWidget
            android:id="@+id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" >

            </TabWidget>
```



```
<FrameLayout
    android:id="@+id/tabcontent"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <LinearLayout
        android:id="@+id/songs_tab"
        android:layout_width="wrap_content"
        android:layout_height="match_parent" >

        <TextView
            android:id="@+id/battery_layout"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:text="Songs"
            android:textAlignment="center" />

        </LinearLayout>

        <LinearLayout
            android:id="@+id/movies_tab"
            android:layout_width="wrap_content"
            android:layout_height="match_parent" >

            <TextView
                android:id="@+id/network_layout"
                android:layout_width="wrap_content"
                android:layout_height="match_parent"
                android:text="Movies"
                android:textAlignment="center" />

            </LinearLayout>

            <LinearLayout
                android:id="@+id/album_tab"
```

```

        android:layout_width="wrap_content"
        android:layout_height="match_parent" >

        <TextView
            android:id="@+id/device_layout"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:text="Albums"
            android:textAlignment="center" />
    </LinearLayout>
</FrameLayout>
</LinearLayout>

</TabHost>
```

5. Navigate to **src → com.example.tabbarexample** folder.
6. Modify the code in **MainActivity** file as shown in code snippet 13. The code shows the declaration and implementation of TabHost and TabSpec.

Code Snippet 13:

```

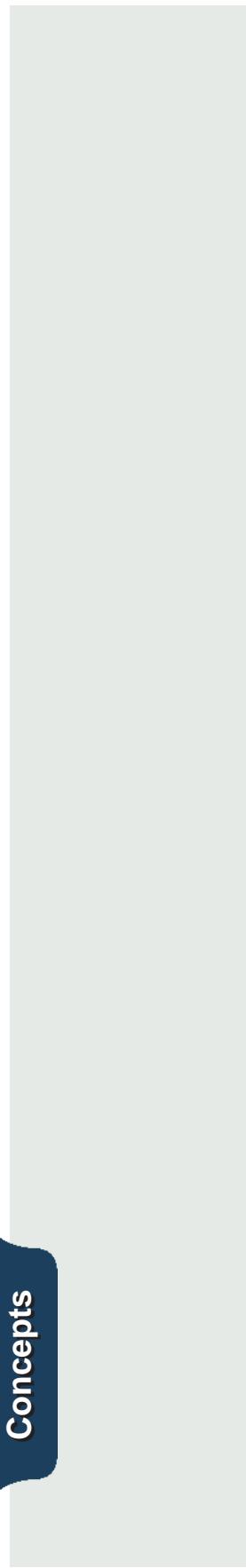
package com.example.tabbarexample;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.app.TabActivity;

import android.view.Menu;
import android.widget.TabHost;
import android.widget.TabHost.TabSpec;

public class MainActivity extends TabActivity {

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



```
setContentView(R.layout.activity_main);

// Access TabHost from Main Layout
TabHost tab_host = (TabHost) findViewById(android.R.id.tabhost);

// Set Tab Specification for Songs Tab

TabSpec songs_tab_spec = tab_host.newTabSpec("songs_tab");

songs_tab_spec.setContent(R.id.songs_tab);
songs_tab_spec.setIndicator("Songs");
tab_host.addTab(songs_tab_spec);

// Set Tab Specification for Movies Tab

TabSpec movies_tab_spec = tab_host.newTabSpec("movies_tab");

movies_tab_spec.setContent(R.id.movies_tab);
movies_tab_spec.setIndicator("Movies");
tab_host.addTab(movies_tab_spec);

// Set Tab Specification for Album Tab

TabSpec album_tab_spec = tab_host.newTabSpec("album_tab");

album_tab_spec.setContent(R.id.album_tab);
album_tab_spec.setIndicator("Albums");
tab_host.addTab(album_tab_spec);

}

@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;
    }
}

```

Figure 5.4 displays the output of the application.

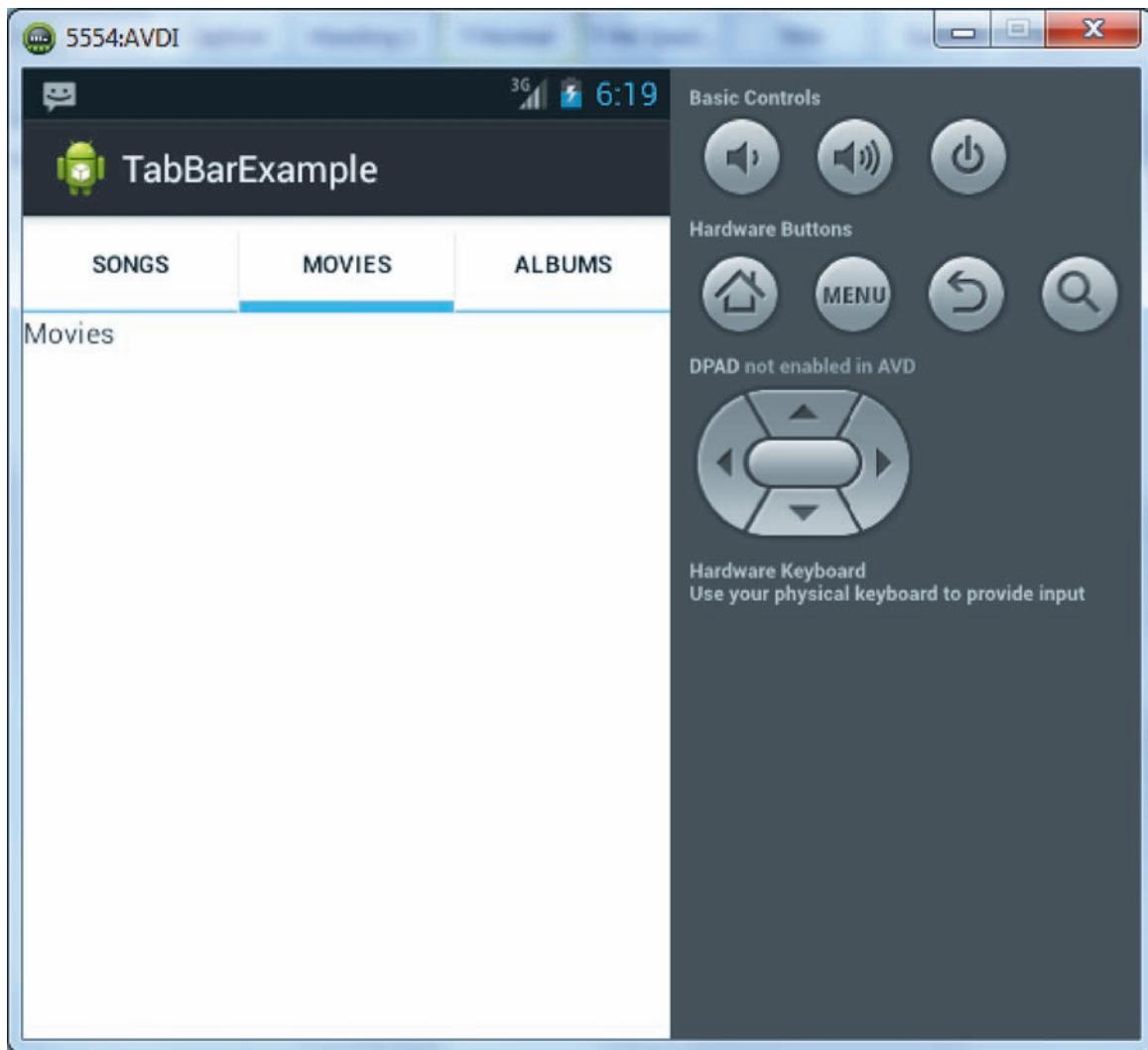


Figure 5.4: TabBarExample – Output

5.2.4 WebView

WebView is used to display Web pages as a part of your activity layout. It does not include any feature of Web browser. WebView loads Web pages and renders raw HTML data.

The following application named **WebViewExample** demonstrates the use of WebView. To develop the application, perform the following steps:

1. Start Eclipse IDE.
2. Create a project named **WebViewExample**.
3. Navigate to **res → layout** folder.
4. Modify the code in **activity_main.xml** file as shown in code snippet 14. The code uses the **WebView** to load a Webpage.

Code Snippet 14:

```
<?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" >

    <WebView

        android:id="@+id/webview01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1" >

    </WebView>

</LinearLayout>
```

5. Navigate to **src → com.example.webviewexample** folder.
6. Modify the code in **MainActivity** file as shown in the code snippet 15. The **WebView** object is used to load a Web page in the Java file.

Code Snippet 15:

```
package com.example.webviewexample;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.graphics.Bitmap;
import android.view.KeyEvent;
import android.webkit.WebView;
```

```
import android.webkit.WebViewClient;

public class MainActivity extends Activity {

    WebView web;

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

        web = (WebView) findViewById(R.id.webview01);
        web.setWebViewClient(new myWebClient());
        web.getSettings().setJavaScriptEnabled(true);
        web.loadUrl("http://www.android.com/");

    }

    public class myWebClient extends WebViewClient {

        @Override
        public void onPageStarted(WebView view, String url,
                Bitmap favicon) {
            // TODO Auto-generated method stub
            super.onPageStarted(view, url, favicon);
        }

        @Override
        public boolean shouldOverrideUrlLoading(WebView view,
                String url) {
            // TODO Auto-generated method stub
            view.loadUrl(url);
            return true;

        }
    }
}
```

```
// To handle "Back" keypress event for WebView to go back to  
previous  
  
// screen.  
  
@Override  
  
public boolean onKeyDown( int keyCode , KeyEvent event ) {  
  
    if ( (keyCode == KeyEvent.KEYCODE_BACK) && web.  
        canGoBack( ) ) {  
  
        web.goBack( );  
  
        return true;  
    }  
  
    return super.onKeyDown(keyCode , event );  
}  
  
}
```

7. Add the following permission in **AndroidManifest.xml** file as shown in code snippet 16.

Code Snippet 16:

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

Figure 5.5 displays the output of the application.

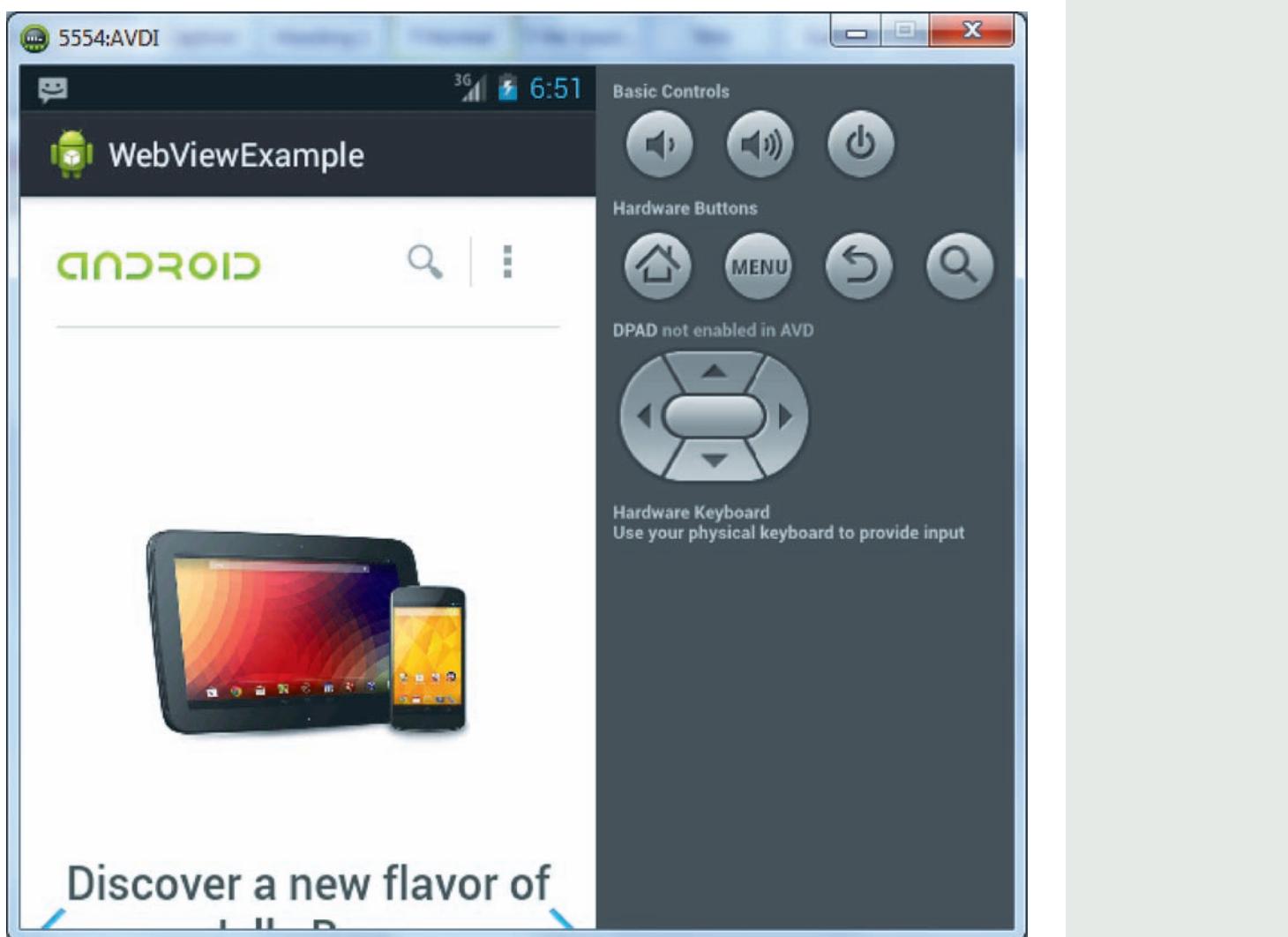


Figure 5.5: WebViewExample – Output

5.2.5 ViewFlipper

The `ViewFlipper` can be used to animate between two or more added views. Here, only one child view can be visible at a time. User can navigate between child views. In `ViewFlipper`, flipin and flipout animations is used while navigating between child views. The developer can also set automatic flipping between each child views at regular interval.

The following application named `ViewFlipperExample` demonstrates the use of `ViewFlipper`. To develop the application, perform the following steps:

1. Start Eclipse IDE.
2. Create a project named `ViewFlipperExample`.
3. Navigate to `res` folder.

4. Create a new folder under it named **anim**.
5. Create a new Android XML file named **flipin**.
6. Modify the code in **flipin.xml** file as shown in code snippet 17.

Code Snippet 17:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/
    android"
        android:interpolator="@android:anim/decelerate_
        interpolator">

    <translate
        android:duration="500"
        android:fromXDelta="-100%"
        android:toXDelta="0%" />

</set>
```

7. Create another new Android XML file named **flipout**.
8. Modify the code in **flipout.xml** file as shown in code snippet 18.

Code Snippet 18:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/
    android"
        android:interpolator="@android:anim/decelerate_
        interpolator">

    <translate
        android:duration="500"
        android:fromXDelta="0%"
        android:toXDelta="100%" />

</set>
```

9. Navigate to **res → layout** folder.
10. Modify the code in **activity_main.xml** file as shown in code snippet 19. The code uses **ViewFlipper** widget.

Code Snippet 19:

```
<?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:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="ViewFlipper" />

    <ViewFlipper

        android:id="@+id/viewflipper"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" >

        <LinearLayout

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

            <TextView

                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="First Screen" />

            <Button

                android:id="@+id/button1"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:text="Flip to Next page" />

        </LinearLayout>
    </ViewFlipper>
</LinearLayout>
```

```

<LinearLayout

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

    <TextView

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Second Screen" />

    <Button

        android:id="@+id/button2"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:text="Flip back" />

</LinearLayout>
</ViewFlipper>

</LinearLayout>

```

11. Navigate to **src → com.example.viewflipperexample** folder.
12. Modify the code in **MainActivity** file as shown in code snippet 20. The ViewFlipper is implemented in Java file.

Code Snippet 20:

```

package com.example.viewflipperexample;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.Button;
import android.widget.ViewFlipper;

```

```
public class MainActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        final ViewFlipper MyViewFlipper = (ViewFlipper)  
findViewByIId(R.id.viewflipper);  
  
        Button button1 = (Button) findViewById(R.id.button1);  
        Button button2 = (Button) findViewById(R.id.button2);  
  
        Animation animationFlipIn = AnimationUtils.  
loadAnimation(this,  
            R.anim.flipin);  
        Animation animationFlipOut = AnimationUtils.  
loadAnimation(this,  
            R.anim.flipout);  
  
        MyViewFlipper.setInAnimation(animationFlipIn);  
        MyViewFlipper.setOutAnimation(animationFlipOut);  
  
        button1.setOnClickListener(new Button.  
OnClickListener() {  
  
            @Override  
            public void onClick(View arg0) {  
                // TODO Auto-generated method stub  
                MyViewFlipper.showNext();  
            }  
        });  
  
        button2.setOnClickListener(new Button.  
OnClickListener() {  
  
            @Override  
            public void onClick(View arg0) {  

```

```
// TODO Auto-generated method stub  
  
        MyViewFlipper.showPrevious();  
    }  
}  
}  
  
@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;  
}  
  
}
```

Figure 5.6 displays the output of the application.

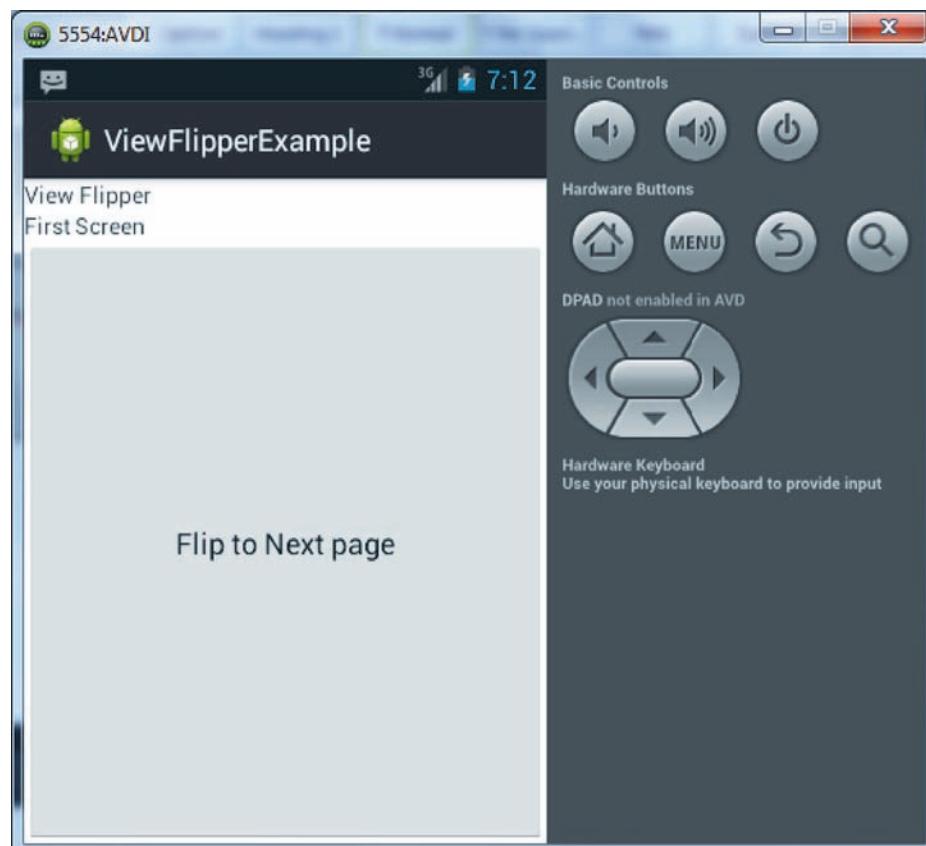


Figure 5.6: ViewFlipper – Output

Figure 5.7 displays the output when the user clicks in the shaded area.

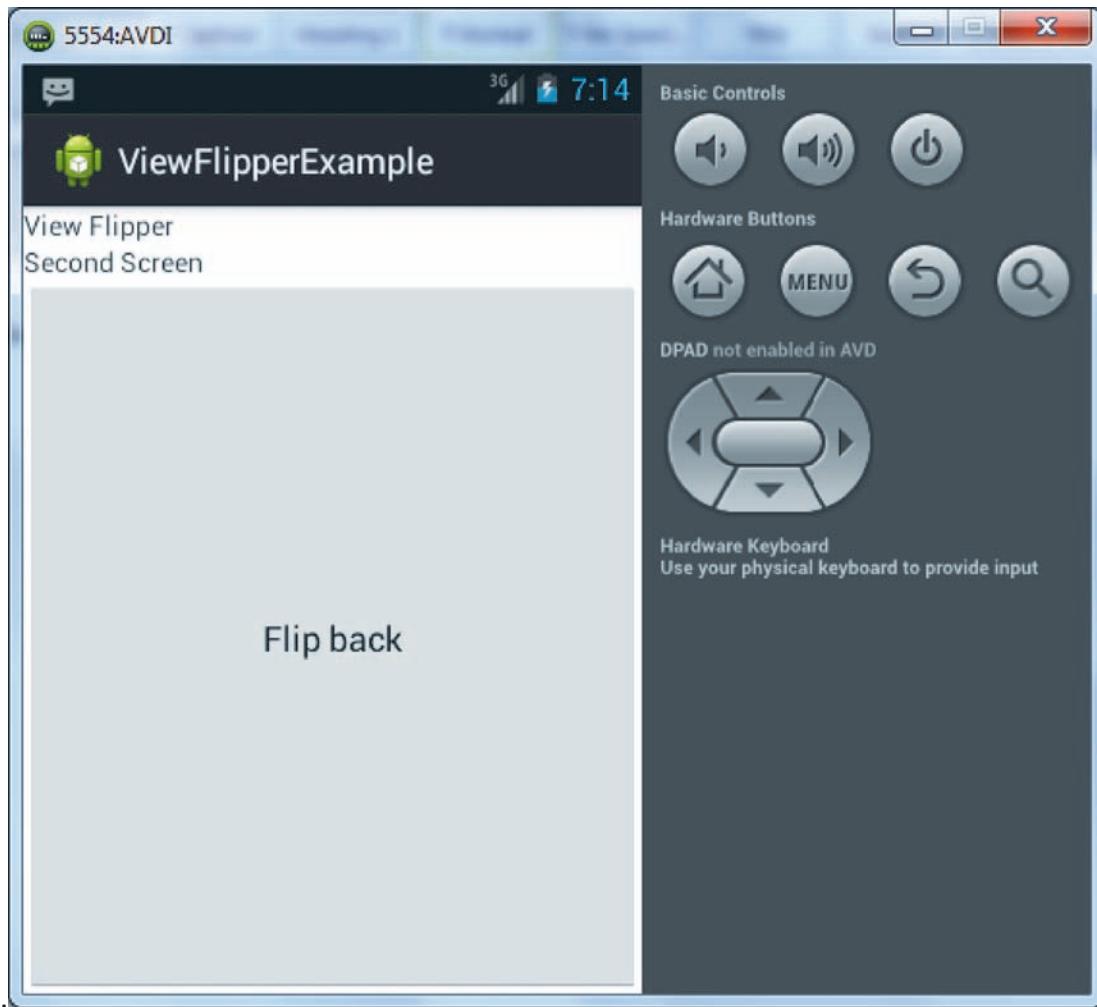


Figure 5.7: Flip Card

5.2.6 VideoView

VideoView is used to play a video file. Android 4.2 can support large extent of video formats including .aac format while lower versions cannot play all the video files of different formats.

5.3 Dialogs

A dialog is used to notify the user with an appropriate message. It is a small window that prompts the user to make a decision or enter additional information.

The different types of dialogs that are available in Android are as follows:

1. Alert Dialog
2. Custom Dialog
3. DatePickerDialog and TimePickerDialog

Time picker and Date picker dialogs have already been discussed in earlier sessions. In this section, a custom dialog is designed.

The following application named **CustomDialogExample** displays the creation of a custom dialog containing an `EditText` view and a `Toast`. To develop the application, perform the following steps:

1. Start Eclipse IDE.
2. Create a project named **CustomDialogExample**.
3. Navigate to **res → layout** folder.
4. Modify the code in the **activity_main.xml** file as shown in code snippet 21.

Code Snippet 21:

```
<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_centerHorizontal="true"  
        android:layout_centerVertical="true"  
        android:text=" Enter Data " />  
  
</RelativeLayout>
```

5. Navigate to **src → com.example.customdialogexample** folder.
6. Modify the code in **MainActivity** file as shown in the code snippet 22.

Code Snippet 22:

```
package com.example.customdialogexample;

import android.os.Bundle;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {

    EditText userData;
    Button signIn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        signIn = (Button) findViewById(R.id.button1);

        signIn.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                alert();
            }
        });
    }

}
```

```
public void alert() {  
    AlertDialog.Builder alert = new AlertDialog.  
        Builder(MainActivity.this);  
  
    userData = new EditText(MainActivity.this);  
  
    alert.setTitle("Enter Data..");  
    alert.setView(userData);  
  
    alert.setPositiveButton("Enter Data", new  
        OnClickListener() {  
  
        @Override  
        public void onClick(DialogInterface dialog, int  
            which) {  
            // TODO Auto-generated method stub  
  
            if (!userData.getText().toString()  
                .equals("")) {  
                Toast.makeText(  
                    getApplicationContext(),  
                    "Data Entered is : "  
                    + userData.getText().  
                    toString(),  
                    Toast.LENGTH_LONG).show();  
  
            } else {  
                Toast.makeText(getApplicationContext(),  
                    "Field should not be empty",  
                    Toast.LENGTH_LONG)  
                    .show();  
            }  
        }  
    } );  
}
```

```
        alert.setNegativeButton("Cancel", new  
        OnClickListener() {  
  
            @Override  
  
            public void onClick(DialogInterface dialog, int  
            which) {  
  
                // TODO Auto-generated method stub  
  
            }  
        });  
  
        alert.show();  
    }  
  
}
```

Figure 5.8 displays the output of the application when the user clicks **Enter Text**.

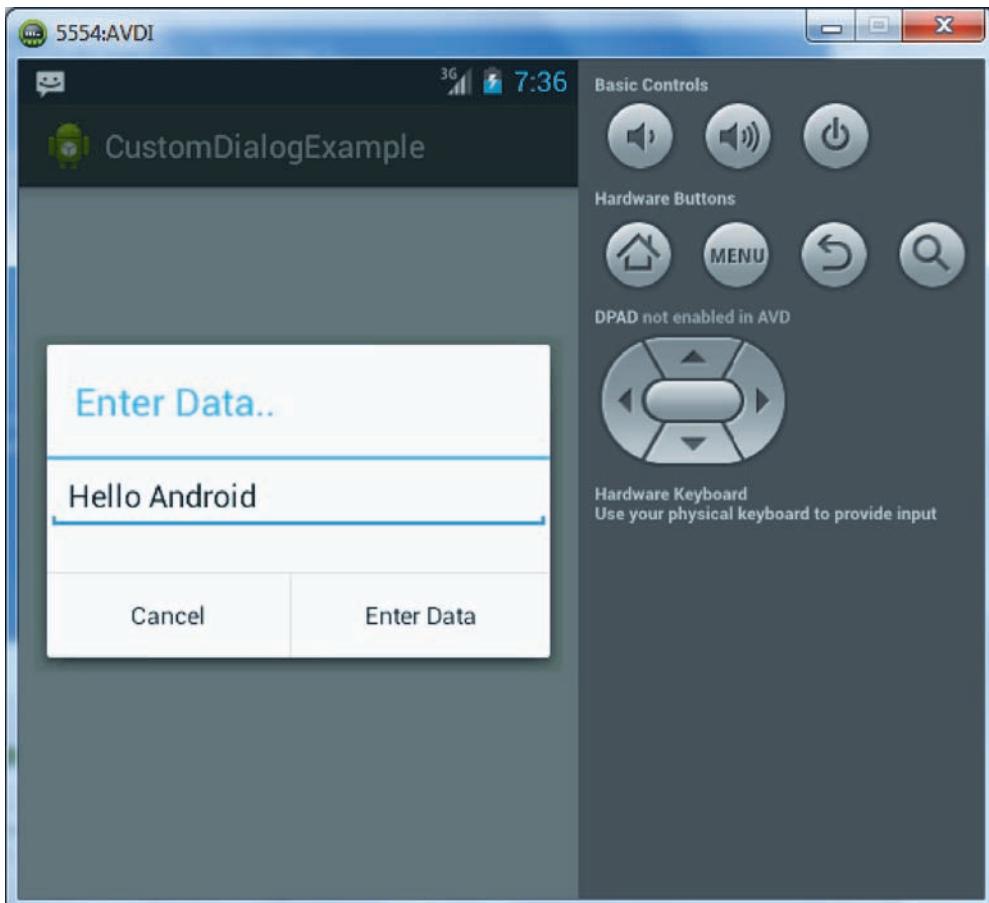


Figure 5.8: Custom Dialog – Output

Figure 5.9 displays the output when the user clicks **Enter Data** present in the dialog box.

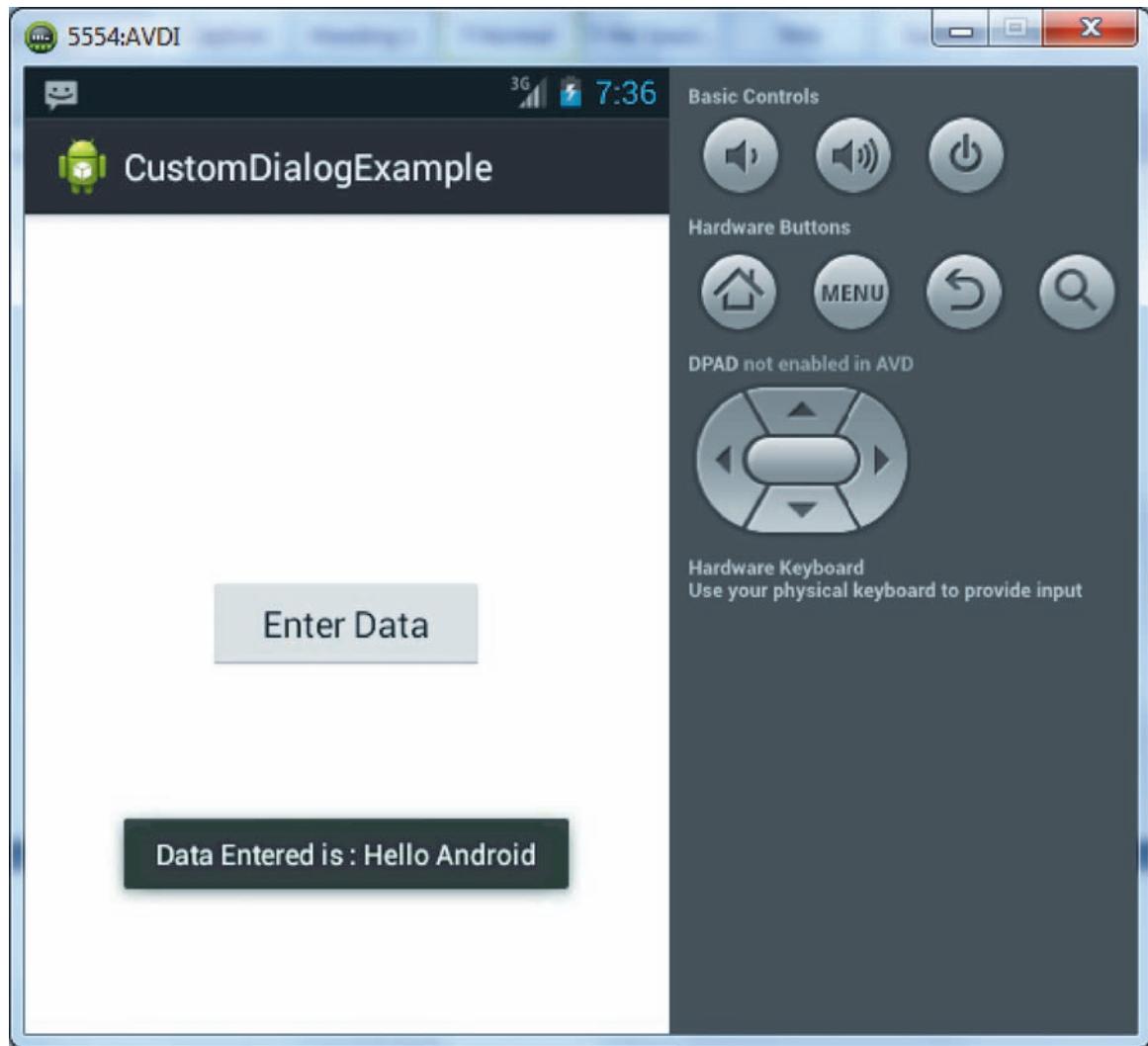


Figure 5.9: Data Saved

5.4 Check Your Progress

1. Which of the following adapter is used for exposing data from a Cursor to a ListView object?

(A)	CursorAdapter	(C)	ResourceCursorAdapter
(B)	SimpleCursorAdapter	(D)	ArrayAdapter

2. Which of the following statement stating the characteristic for an Adapter is true?

(A)	It is an extendable class	(C)	It acts as a bridge between View and data source
(B)	It makes the view visible	(D)	It helps in displaying images

3. Which of the following option defines a Dialog?

(A)	It is light weight process	(C)	It is third party element
(B)	It is one type of thread	(D)	A dialog is used to notify message to the user

4. Which of the following view is used to display a Video in the Android Application?

(A)	SurfaceView	(C)	CheckedTextView
(B)	WebView	(D)	VideoView

5. Which of the following statements are true for SimpleAdapter?

(A)	It is backed by an array of objects	(C)	It can be used for both ListView and Spinner
(B)	It helps to map static data to the views defined in the XML	(D)	It is mapped to the view using an ArrayList of Maps

5.4.1 Answers

1.	A
2.	C
3.	D
4.	D
5.	B, D



- Adapter is an object that acts as a bridge between the UI Components such as List view, Grid view, and so on and the underlying data sources.
- Adapter is responsible for providing view for every element of the data source.
- User Interface components acts as an interface between the user and the application to input data and display the expected result.
- ScrollView is a container that holds child views and has a property of scrolling when the child items size in the container exceeds the screen size.
- TabBar is used to display data in Tab format as is present in android's contacts view.
- TabHost is container for a tabbed window view. This object holds two children: a set of tab labels that the user clicks to select a specific tab and a FrameLayout object that displays the contents of that page.
- WebView is used to display Web pages as a part of your activity layout. It does not include any feature of Web browser.
- The ViewFlipper can be used to animate between two or more added views.
- A Dialog is used to notify appropriate message to the user. It is a small window that prompts the user to make a decision or enter additional information.



Samantha wants to create an application that will perform the following functionalities:

1. Develop an application with two tabs in which the first tab displays a list of audio files where as the second tab displays a list of video files. Clicking any files will display the details of the selected file such as the composer and singer of the music for audio list and the director and producer name for the video files. All these details will be displayed as a Toast message.
2. Develop an application which will display a Webpage using the WebView widget.

“ The real voyage of discovery
consists not seeking in new lands,
but seeing things with new eyes ”

JELLYBEAN
ICE CREAM SANDWICH

Session - 6

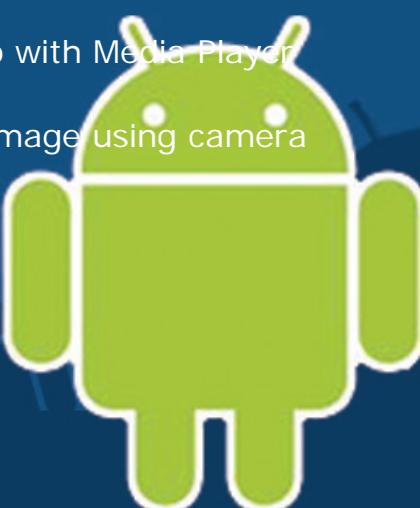
Media Handling

Welcome to the session of **Media Handling**.

This session introduces you to the different ways of handling media in android. This session also discusses the multimedia capabilities of mobile devices and explains the process of creating simple basic applications rich in multimedia. It would drive you through a few features in your mobile devices which most of the users may not be aware of.

In this session, you will learn to:

- ➔ Explain the use of graphics in Android
- ➔ Explain animation
- ➔ Explain playing of audio and video with Media Player
- ➔ Explain the process of capturing image using camera



6.1 Introduction

Multimedia capabilities of mobile devices play a significant role for consumers. Android's open source technology provides support for multimedia capabilities. It ensures that it offers multimedia API for playing and recording of wide range of image, audio, and video formats. Camera API also exposes the capabilities to build applications with comprehensive multimedia functionality.

6.2 Graphics

In media handling, Graphics plays a very important role. Android provides basic level of graphics tools such as canvases, colors filters, points, and rectangles that allows the developer to handle drawing on the screen directly. Android provides different techniques for performing varying graphical tasks. The technique for graphic programming varies depending on graphical demand such as between a 2D and 3D, static and dynamic, and so on. But all of them require a knowledge of the underlying graphics libraries.

The session will look at how to create the simplest form of 2D graphics. However, before doing that a quick look is required on a few important concepts of the 2D graphics such as Views, Canvas, Paint, and `onDraw()` method. They are as follows:

- ➔ Views are known to be the visible elements in an Android UI.
- ➔ Each View is associated with the Canvas.
- ➔ When the View is displayed, its `onDraw()` method is automatically invoked by Android.
- ➔ The developer can be creating their own View with the `onDraw()` method to display basic objects using the Canvas.
- ➔ Canvas has methods for drawing Arcs, Bitmaps, Circles, Lines, Ovals, Paths, Rectangles, and so on.
- ➔ Paint has the method for setting the alpha, color, shade, stroke, and so on.

The following application named **FunWithDrawing** is a simple 2D graphic application. To develop this application, perform the following steps:

1. Start **Eclipse**.
2. Create a new project named **FunWithDrawing** with the activity class name as **FunWithDrawingActivity**.
3. Navigate to **src → com.example.funwithdrawing** folder.
4. Create a custom View class named **DrawableView** by right-clicking the project's

package folder to display the context menu and selecting **New → Class**. The class should extend from **android.view.View** and also implement the necessary constructors.

5. Modify the code in **DrawableView** class as shown in code snippet 1.

Code Snippet 1:

```
package com.example.funwithdrawing;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.view.MotionEvent;
import android.view.View;

public class DrawableView extends View {

    public DrawableView(Context context) {
        super(context);
        // TODO Auto-generated constructor stub
    }

    public DrawableView(Context c, AttributeSet a) {
        super(c, a);
    }

    protected void onDraw(Canvas canvas) {

        // this is the "paintbrush"
        Paint paint = new Paint();

        // set the color randomly
        int whichColor = (int) (Math.random() * 4);

        if (whichColor == 0)
```

```
        paint.setColor(Color.RED);

        else if (whichColor == 1)

            paint.setColor(Color.GREEN);

        else if (whichColor == 2)

            paint.setColor(Color.BLUE);

        else

            paint.setColor(Color.YELLOW);

        // draw Rectangle with corners at (40, 20) and (90, 80)

        canvas.drawRect(40, 20, 90, 80, paint);

    }

    public boolean onTouchEvent(MotionEvent event) {

        // if it's an up ("release") action

        if (event.getAction() == MotionEvent.ACTION_UP) {

            // get the coordinates

            float x = event.getX();

            float y = event.getY();

            // see if they clicked the box

            if (x >= 40 && x <= 90 && y >= 20 && y <= 80) {

                // redraw the View... this calls onDraw again!

                invalidate();

            }

        }

        // indicates that the event was handled

        return true;

    } // end of onTouchEvent

}
```

In the code, the `onDraw()` method is used for drawing primitive shapes such as rectangle, oval, arc by invoking the `drawRect()`, `drawOval()`, and `drawArc()` method respectively. The method also declares an instance of `Paint` class for setting the paintbrush color using the `setColor()` method of the class. Thus, the `Paint` class specifies how to draw.

The `onTouchEvent()` method is invoked when the event takes place and inturn invokes the `invalidate()` method which is used for redrawing the screen.

6. Navigate to **res → layout** folder.
7. Modify the code in **activity_fun_with_drawing.xml** file as shown in code snippet 2.

Code Snippet 2:

```
<?xml version="1.0" encoding="utf-8"?>
<com.example.funwithdrawing.DrawableView
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
```

8. Navigate to **src → com.example.funwithdrawing** folder.
9. Modify the code in **FunWithDrawingActivity** as shown in code snippet 3.

Code Snippet 3:

```
package com.example.funwithdrawing;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class FunWithDrawingActivity extends Activity {

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

    @Override
```

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

Figure 6.1 displays the output of the application.

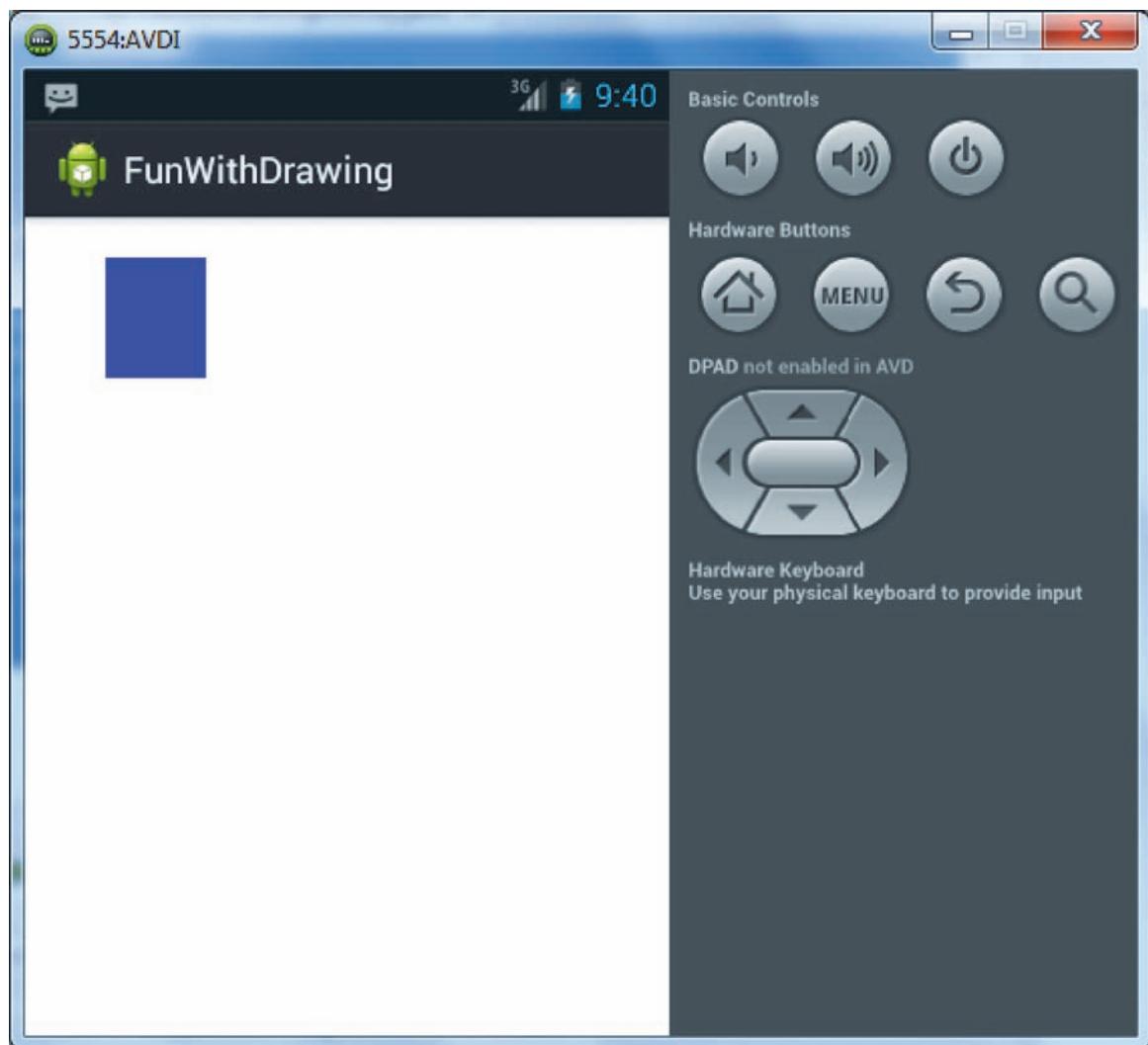


Figure 6.1: FunWithDrawing – Output

6.3 Animation

Animation is very interesting concept which every learner is keen to know and implement it. Therefore, developing animated application plays a very vital role when it comes to apps development in Android. Android provides a set of API for applying animation to UI controls and for drawing 2D and 3D graphics. Android 3.0 introduces the Properties Animation API which allows changing properties of an object over a specific time period.

Android framework provides two animation types namely, property animation and view animation. Property animation has been introduced in Android 3.0 version. Both animation systems are feasible, but the property animation system, is the preferred method to use than view animation. This is because property animation is more flexible and offers more features. In addition to these two systems, the developer can use the traditional Drawable animation. The drawable animation allows the developer to load drawable resources and display them one frame after another. It is created with a series of images which are displayed in a sequence and played in an order.

The API allows you to define arbitrary object properties such as a start and end value and also apply a time-based change to this attribute. This API can also be applied on any Java object and not only on Views.

The super class of the animation API is the `Animator` class. The object of `Animator` class is used to modify the attributes of an object.

You can also add an `AnimatorListener` class to your `Animator` class. This listener is invoked in the different phases of the animation. You can use this listener to perform actions before or after a certain animation, for example, after adding or removing a View from a View Group.

6.3.1 Android Resources

Android resource defines two types of animations. They are as follows:

Property Animation – Creates an animation by modifying an object's property values over a set period of time with an `Animator`. Introduced in Android 3.0 (API level 11), the property animation system lets you animate properties of any object, including ones that are not rendered to the screen. The system is extensible and lets you animate properties of custom types as well. To create a property animation the following points are to be considered:

- **Definition:** An animation defined in XML file that modifies properties of the target object, such as background colors or alpha value, over a set amount of time.
- **File location:** File is located in `res/Animator` folder. The file name will be used as a resource ID.
- **Compiled Resource Data Type:** Resource pointer to a Value Animator, Object Animator, or a Object Set.

- **Resource Reference:** Resource will be referenced in Java file as: **R.animator.filename**, in XML file as: **@[package]animator/filename**.

View Animation – It is the older system and can be used with Views. It is easy to setup and offers enough capabilities to meet the varied needs of an application. There are two types of the animation that you can do with the view animation framework. They are as follows:

- **Tween Animation:** Creates an animation by performing a series of transformations on the contents of a View object. An animation defined in an XML file helps to perform transitions such as rotating, fading, moving, and stretching on a View object. The single root element in the XML file can be either a single `<alpha>`, `<scale>`, `<translate>`, `<rotate>`, interpolator element, or `<set>` element that holds groups of these elements. The points to be considered for tween animations are as follows:
- **File Location:** File is located in **res/anim/** folder. The filename will be used as a resource ID.
 - **Compiled Resource Data Type:** Resource pointer to an Animation.
 - **Resource Reference:** Resource will be referenced in Java file as: **R.anim.filename** and in XML file as: **@[package]anim/filename**.
- **Frame Animation:** Creates animation by showing a sequence of images in an order with a Drawable object. The points to be considered for frame animations are as follows:
- **File Location:** File is located in **res/drawable/** folder. The filename will be used as a resource ID.
 - **Compiled Resource Data Type:** Resource pointer to an AnimationDrawable.
 - **Resource Reference:** Resource will be referenced in Java file as: **R.drawable.filename** and in XML file as: **@[package]drawable/filename**.

Drawable Animation – This involves displaying Drawable resources one after the other, like a roll of film. This method of animation is useful if you want to animate things that are easier to represent with Drawable resources, such as a progression of bitmaps.

The following application named **AnimationExample** is a simple animation application. To create this application, perform the following steps:

1. Start **Eclipse**.
2. Create a new project named **AnimationExample** with the activity class name as **FirstActivity**.
3. Navigate to **res → layout** folder.

4. Modify the code in **activity_first.xml** file as shown in code snippet 4.

Code Snippet 4:

```
<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=".FirstActivity" >

    <Button
        android:id="@+id/TheFirstButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginTop="47dp"
        android:text="Animation1" />

</RelativeLayout>
```

5. Navigate to **src → com.example.animationexample** folder.
 6. Modify the code in **FirstActivity** file as shown in code snippet 5.

Code Snippet 5:

```
package com.example.animationexample;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
```

```
import android.widget.Button;
import android.view.View.OnClickListener;

public class FirstActivity extends Activity {

    Button firstbutton;

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

        firstbutton = (Button) findViewById(R.id.TheFirstButton);
        firstbutton.setOnClickListener(new OnClickListener() {

            public void onClick(View v) {
                Intent intent = new Intent(getApplicationContext(),
                    SecondActivity.class);
                startActivity(intent);
                overridePendingTransition(R.anim.anim_in,
                    R.anim.hold);
            }
        });
    }

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

7. Navigate to **res** → **layout** folder.
8. Create a **New Android XML** file named **second.xml**.
9. Modify the code in **second.xml** file as shown in code snippet 6.

Code Snippet 6:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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/TheSecondButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:layout_marginRight="45dp"
        android:layout_marginTop="97dp"
        android:text="Back" /></RelativeLayout>
```

10. Navigate to **res** folder.
11. Create a new folder named **anim**.
12. Create a **New Android XML** file under **res/anim** named **anim_in.xml**.
13. Modify the code in **anim_in.xml** file as shown in code snippet 7.

Code Snippet 7:

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

    <translate
        android:duration="500"
        android:fromXDelta="100%p"
```

```
        android:fromYDelta="0%p"
        android:toXDelta="0%p"
        android:toYDelta="0" /></set>
```

14. Create another **New Android XML** file under **res/anim** folder named **hold.xml**.
15. Modify the code in **hold.xml** file as shown in code snippet 8.

Code Snippet 8:

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

    <translate
        xmlns:android="http://schemas.android.com/apk/res/
        android"
        android:duration="500"
        android:fromXDelta="0"
        android:interpolator="@android:anim/accelerate_
        interpolator"
        android:toXDelta="0" />
</set>
```

16. Navigate to **src → com.example.animationexample** folder.
17. Create a new class named **SecondActivity**.
18. Modify the code in **SecondActivity** file as shown in code snippet 9.

Code Snippet 9:

```
package com.example.animationexample;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.*;
```

```

public class SecondActivity extends Activity {

    Button secondbutton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);

        secondbutton = (Button) findViewById(R.
                id.TheSecondButton);
        secondbutton.setOnClickListener(newOnClickListener() {

            public void onClick(View v) {
                // TODO Auto-generated method stub

                Intent intent = new Intent(getApplicationContext(),
                        FirstActivity.class);
                startActivity(intent);
                overridePendingTransition(R.anim.anim_in, R.anim.
                hold);
            }
        });
    }
}

```

19. Add the <activity> tag in **AndroidManifest.xml** file as shown in code snippet 10 before the closing </application> tag.

Code Snippet 10:

```

...
<activity android:name=".SecondActivity"></activity>
...

```

Figure 6.2 displays the output after execution of the application.

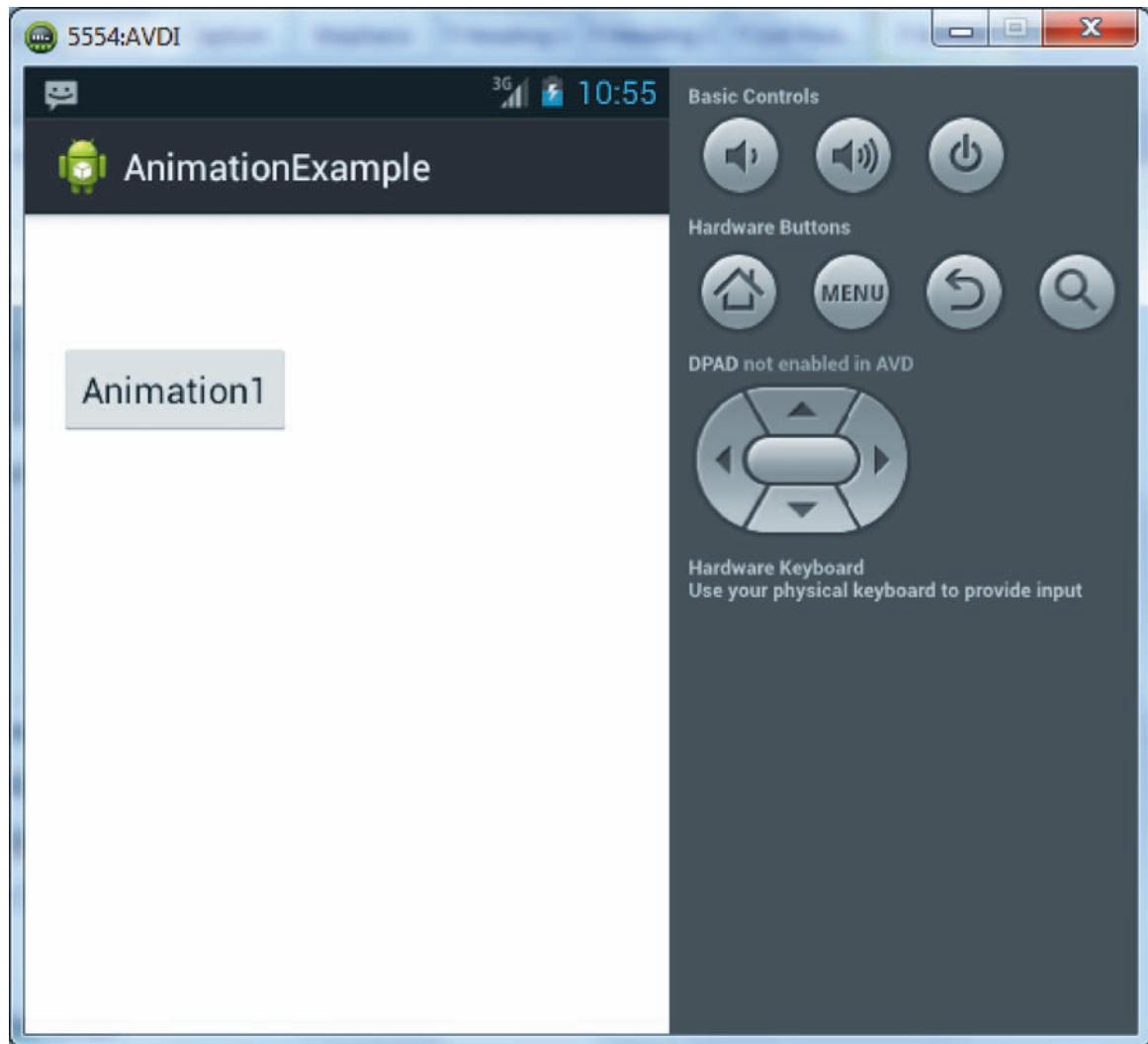


Figure 6.2: AnimationExample

Note - On clicking the button the title animates.

6.4 Audio/Video

Android uses OpenCore as its core component of media framework. OpenCore supports different file formats such as MP3, AAC, AAC+, 3GPP, MPEG-4, JPEG. The following classes are used to play audio and video in Android framework:

- ➔ **MediaPlayer** – This is responsible for playing audio and video and is the primary class.
- ➔ **AudioManager** – This is responsible for managing audio source and audio output on the device.

6.4.1 Media Player Basics

The Android multimedia framework includes support for playing variety of common media types, so that one can easily integrate audio, video, and images into your applications. One can play audio or video from media files stored in your application's resources (raw resources), from standalone files in the filesystem, or from a data stream arriving over a network connection. All of these uses MediaPlayer APIs. MediaPlayer class is the most important component of media framework. The various sources from which media can be played are as follows:

- Resource folder.
- File System path.
- URL.

The different states of a media player are described as follows:

- Creation and initialization of the media player with the media to play using the method `new()` and `setDataSource()`.
- Preparation of the media player for playback using the method `prepare()`.
- Start of the playback using the method `start()`.
- Pause or Stop of the playback using the `pause()` or `stop()` method.
- Termination of the playback using `stop()` method.

In the **AndroidManifest.xml** file the appropriate permission should be provided based on use of related features. The permission that can be granted are as follows:

- **Internet Permission** – This permission is required when the media player is streaming content from network. The application should have the following permission as shown in code snippet 11.

Code Snippet 11:

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

- **Wake Lock Permission** – This permission is provided when the developer does not want the screen to dim or the processor to sleep. The application should have the following permission as shown in code snippet 12.

Code Snippet 12:

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

6.4.2 Playing an Audio Clip

The following application named **MediaDemo** plays audio files. To develop the application, perform the following steps:

1. Start **Eclipse**.
2. Create a new project named **MediaDemo**.
3. Navigate to **res** folder.
4. Create a new folder under **res** folder named **raw**.
5. Copy all the audio files under **res/raw** folder.
6. Navigate to the **res → layout** folder.
7. Modify the code in **activity_main.xml** file as shown in code snippet 13.

Code Snippet 13:

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

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

    <Button

        android:id="@+id/startPlayerBtn"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:onClick="doClick"
        android:text="Start Player" />

    <Button

        android:id="@+id/pausePlayerBtn"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:onClick="doClick"
        android:text="Pause Player" />


```

```

<Button
    android:id="@+id/restartPlayerBtn"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="doClick"
    android:text="Restart Player" />

<Button
    android:id="@+id/stopPlayerBtn"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="doClick"
    android:text="Stop Player" /></LinearLayout>

```

8. Navigate to **src → com.example.mediademo** folder.
9. Modify the source code in **MainActivity** file as shown in code snippet 14.

Code Snippet 14:

```

package com.example.mediademo;

import android.media.AudioManager;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.os.PowerManager;
import android.app.Activity;
import android.content.res.AssetFileDescriptor;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.media.MediaPlayer.OnPreparedListener;

public class MainActivity extends Activity implements
OnPreparedListener {

private static final String TAG = null;

```

```
private MediaPlayer mediaPlayer;
private int playbackPosition = 0;

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

public void doClick(View view) {
    switch (view.getId()) {
        case R.id.startPlayerBtn:
            try {
                // Only have one of these play methods
                // uncommented
                // playAudio(AUDIO_PATH);
                playLocalAudio();
                // playLocalAudio_UsingDescriptor();
            } catch (Exception e) {
                e.printStackTrace();
            }
            break;
        case R.id.pausePlayerBtn:
            if (mediaPlayer != null && mediaPlayer.isPlaying()) {
                playbackPosition = mediaPlayer.
                    getCurrentPosition();
                mediaPlayer.pause();
            }
            break;
        case R.id.restartPlayerBtn:
            if (mediaPlayer != null && !mediaPlayer.isPlaying()) {
                mediaPlayer.seekTo(playbackPosition);
                mediaPlayer.start();
            }
    }
}
```

```
        break;

    case R.id.stopPlayerBtn:
        if (mediaPlayer != null) {
            mediaPlayer.stop();
            playbackPosition = 0;
        }
        break;
    }

private void playAudio(String url) throws Exception {
    killMediaPlayer();

    mediaPlayer = new MediaPlayer();
    mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
    mediaPlayer.setDataSource(url);
    mediaPlayer.setOnPreparedListener(this);
    mediaPlayer.setWakeMode(this, PowerManager.PARTIAL_WAKE_LOCK);
    mediaPlayer.prepareAsync();
}

private void playLocalAudio() throws Exception {
    // add the name of the song to be played in the raw folder
    mediaPlayer = MediaPlayer.create(this, R.raw.geetgatahoomein);
    mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
    mediaPlayer.start();
}

private void playLocalAudio_UsingDescriptor() throws Exception {
```

```
AssetFileDescriptor fileDesc = getResources().  
openRawResourceFd(  
        R.raw.flyaway);  
  
if (fileDesc != null) {  
  
    mediaPlayer = new MediaPlayer();  
    mediaPlayer.setAudioStreamType(AudioManager.  
        STREAM_MUSIC);  
  
    mediaPlayer.setDataSource(fileDesc.  
        getFileDescriptor(),  
        fileDesc.getStartOffset(), fileDesc.  
        getLength());  
  
    fileDesc.close();  
  
    mediaPlayer.prepare();  
    mediaPlayer.start();  
}  
}  
  
@Override  
protected void onDestroy() {  
    super.onDestroy();  
    killMediaPlayer();  
}  
  
private void killMediaPlayer() {  
    if (mediaPlayer != null) {  
        try {  
            mediaPlayer.release();  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```

public void onPrepared(MediaPlayer mp) {
    Log.d(TAG, "MediaPlayer is prepared");
    mp.start();
}

@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;
}
}

```

10. Modify the code in **AndroidManifest.xml** file as shown in code snippet 15 for setting the permission.

Code Snippet 15:

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

Figure 6.3 displays the output when the code is executed in the emulator.

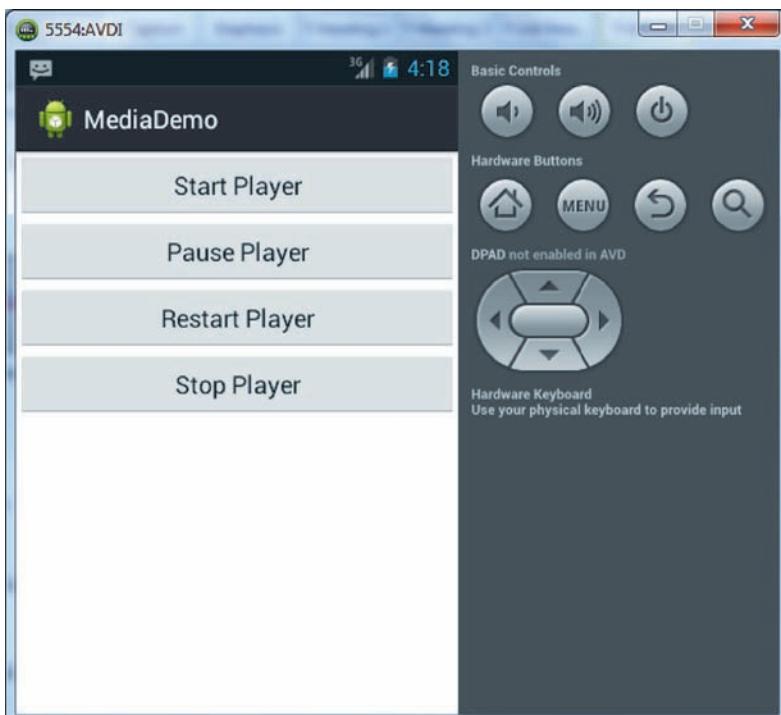


Figure 6.3: MediaDemo – Output

6.4.3 Playing a Video

To play a video, an application named VideoDemo is created. The steps to create the demo are as follows:

1. Start **Eclipse**.
2. Create a project named **ViewDemoApp**.
3. Navigate to **res → layout** folder.
4. Modify the code in **activity_main.xml** file as shown in code snippet 16.

Code Snippet 16:

```
<?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" >

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >

        <VideoView
            android:id="@+id/myvideoview"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent" />
    </LinearLayout></LinearLayout>
```

5. Navigate to **src → com.example.viewdemoapp** folder.
6. Modify the code in **MainActivity** file as shown in code snippet 17.

Code Snippet 17:

```
package com.example.viewdemoapp;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.MediaController;
import android.widget.VideoView;

public class MainActivity extends Activity {

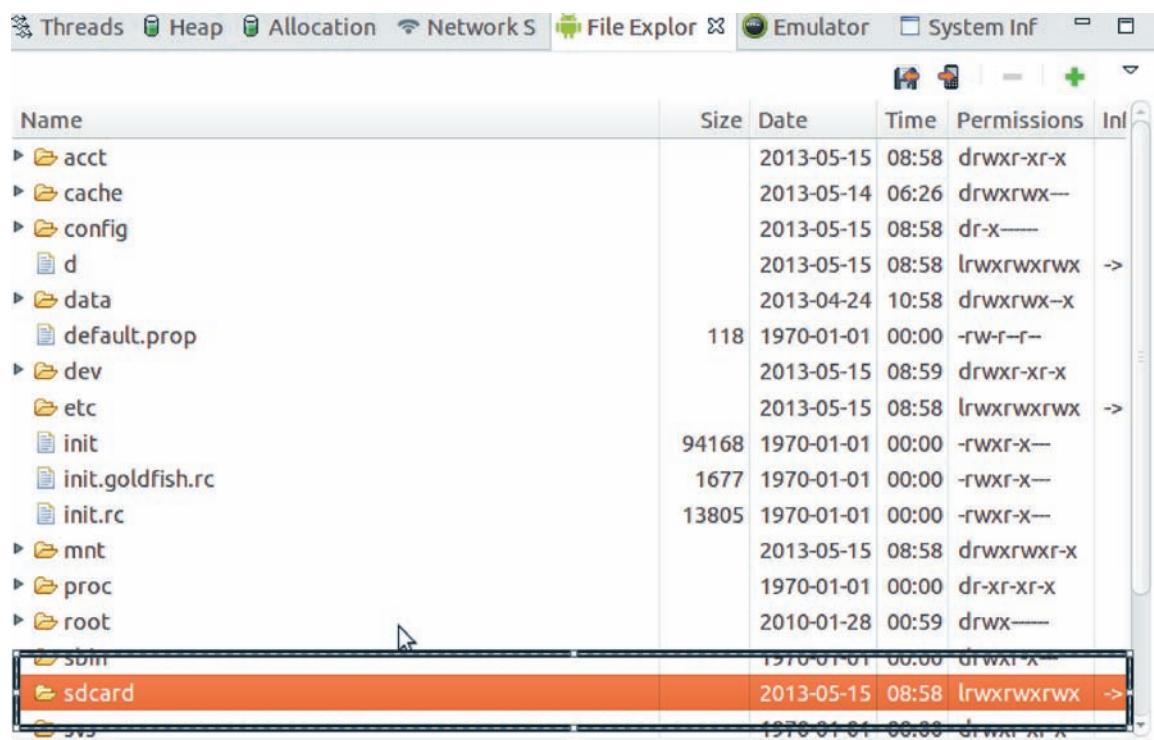
    String SrcPath = "/sdcard/the best line in cartoon history.mp4";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        VideoView myVideoView = (VideoView) findViewById(R.
                id.myvideoview);
        myVideoView.setVideoPath(SrcPath);
        myVideoView.setMediaController(new
                MediaController(this));
        myVideoView.requestFocus();
        myVideoView.start();
    }

    @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;
    }
}
```

Before executing the application one needs to place the video clip the sdcard. This can be done by performing the following steps:

- Change the perspective from Java to DDMS by navigating to the top right corner in your eclipse.
- Once in DDMS perspective navigate to the **File Explorer** tab as shown in the figure 6.4.
- Place the video clip in the sdcard by navigating to the top right corner in the DDMS perspective and then, click the icon push the file onto device.
- Navigate to the desired location of the video clip and click **Open** to place the video clip in the sdcard.



Name	Size	Date	Time	Permissions	In
▶ acct		2013-05-15	08:58	drwxr-xr-x	
▶ cache		2013-05-14	06:26	drwxrwx--	
▶ config		2013-05-15	08:58	dr-x---	
d		2013-05-15	08:58	lrwxrwxrwx	→
▶ data		2013-04-24	10:58	drwxrwx-x	
default.prop	118	1970-01-01	00:00	-rw-r--r-	
▶ dev		2013-05-15	08:59	drwxr-xr-x	
etc		2013-05-15	08:58	lrwxrwxrwx	→
init	94168	1970-01-01	00:00	-rwxr-x--	
init.goldfish.rc	1677	1970-01-01	00:00	-rwxr-x-	
init.rc	13805	1970-01-01	00:00	-rwxr-x--	
▶ mnt		2013-05-15	08:58	drwxrwxr-x	
▶ proc		1970-01-01	00:00	dr-xr-xr-x	
▶ root		2010-01-28	00:59	drwx----	
sbin		1970-01-01	00:00	drwxr-x-	
sdcard		2013-05-15	08:58	lrwxrwxrwx	→
sys		1970-01-01	00:00	drwxr-xr-x	

Figure 6.4: DDMS perspective

Figure 6.5 displays the output once the application is executed.

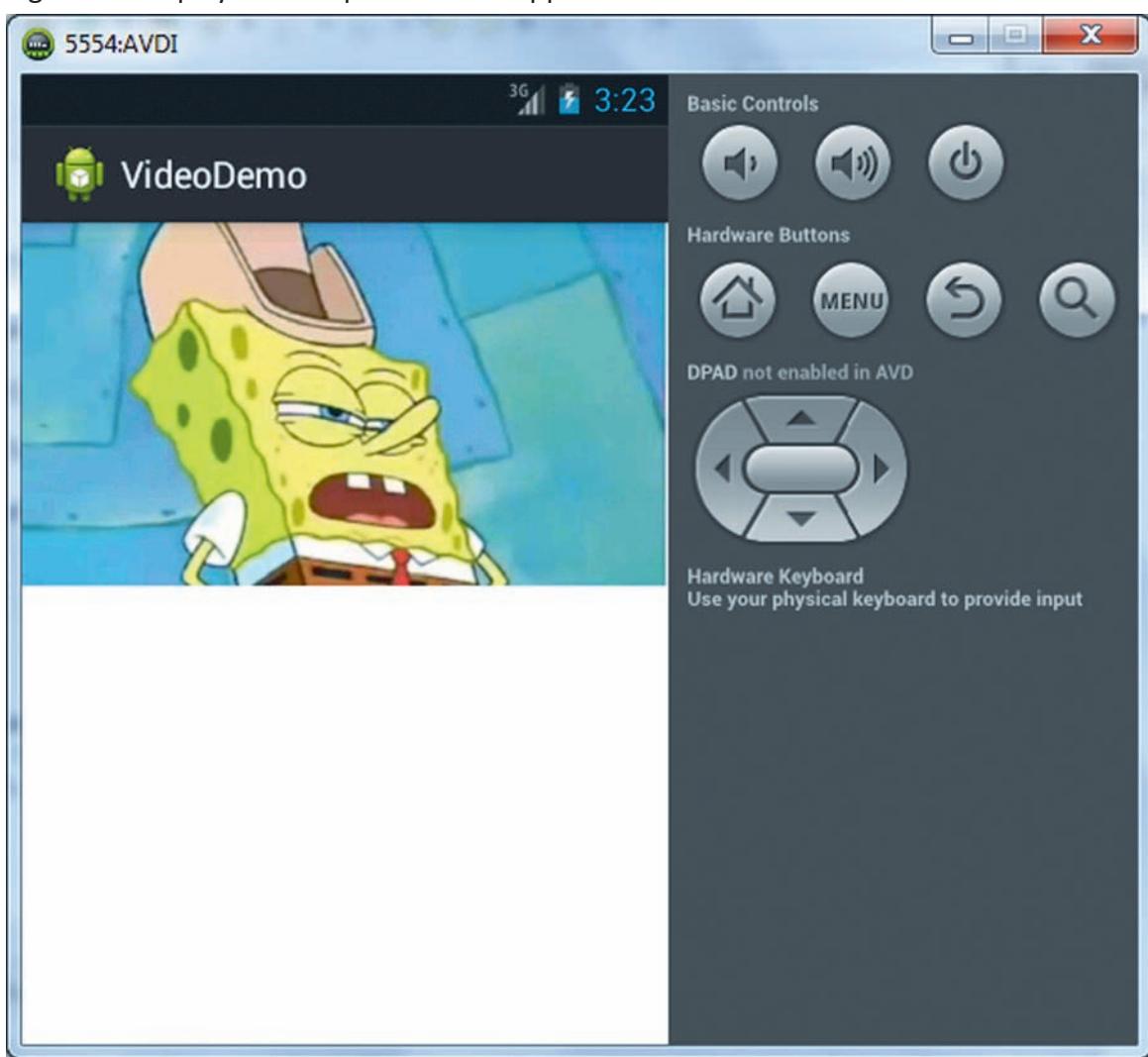


Figure 6.5: VideoDemoApp

6.5 Camera

The Android framework provides support for various cameras and camera features available on devices, allowing you to capture pictures and videos in your application. The Android framework supports capturing images and video through the Camera API or camera Intent.

6.5.1 List of Relevant Classes

The relevant classes are as follows:

1. Camera

The characteristics of this class are as follows:

- This is the main class and is responsible for controlling the device camera.
- The Camera class is used to set image capture settings, start/stop preview, snap pictures, and retrieve frames for encoding the video.
- It is present in android.hardware.camera package.
- It is derived from Object class.

The permission must be granted in **AndroidManifest.xml** file as shown in code snippet 18.

Code Snippet 18:

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```

2. SurfaceView

The characteristics of this class are as follows:

- This class is used to present a live camera preview to the user.
- It is present in android.view.SurfaceView package.
- It is derived from View class.

3. MediaRecorder

The characteristics of this class are as follows:

- This class is used to record video from the camera.
- It is present in android.media.MediaRecorder package.

6.5.2 Working with Camera

1. Start Eclipse.
2. Create a project named **ImageCaptureDemo**.
3. Navigate to **res → layout** folder.
4. Modify the code in **activity_main.xml** file as shown in code snippet 19.

Code Snippet 19:

```
<?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:layout_width="fill_parent"  
        android:layout_height="wrap_content"  
        android:layout_margin="3dp"  
        android:text="Intro"  
        android:textStyle="bold" />  
  
    <Button  
        android:id="@+id/capture_btn"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="Capture" />  
  
    <ImageView  
        android:id="@+id/picture"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_margin="5dp"  
        android:background="@drawable/pic_border"  
        android:contentDescription="Picture" />  
  
    <LinearLayout  
        android:id="@+id/callme"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content" >  
    </LinearLayout></LinearLayout>
```

5. Create another **Android XML File** under **res → layout** folder named **activity_shoot_and_crop.xml**.
6. Modify the code in the file as shown in code snippet 20.

Code Snippet 20:

```
<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"
    tools:context=".ShootAndCropActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world" /></
    RelativeLayout>
```

7. Navigate to **src → com.example.imagecapturedemo** folder.
8. Modify the code in **MainActivity** file as shown in code snippet 21.

Code Snippet 21:

```
package com.example.imagecapturedemo;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View.OnClickListener;
import android.content.ActivityNotFoundException;
import android.content.Intent;
import android.graphics.Bitmap;
import android.net.Uri;
import android.provider.MediaStore;
import android.view.View;
```

```
import android.widget.Button;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.Toast;

public class MainActivity extends Activity implements
OnClickListener {

    Button next;
    //keep track of camera capture intent
    final int CAMERA_CAPTURE = 1;
    //keep track of cropping intent
    final int PIC_CROP = 2;
    //capturedpictureuri
    private Uri picUri;

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

        //retrieve a reference to the UI button
        Button captureBtn = (Button) findViewById(R.id.capture_btn);
        //handle button clicks
        captureBtn.setOnClickListener(this);
    }

    /**
     * Click method to handle user pressing button to launch
     * camera
     */
    public void onClick(View v) {
        if (v.getId() == R.id.capture_btn) {
```

```
try {  
    //use standard intent to capture an image  
  
    Intent captureIntent = new Intent(MediaStore.  
        ACTION_IMAGE_CAPTURE);  
  
    //we will handle the returned data in  
    //onActivityResult  
  
    startActivityForResult(captureIntent,  
        CAMERA_CAPTURE);  
}  
  
catch(ActivityNotFoundException anfe){  
    //display an error message  
  
    String errorMessage = "Whoops – your device  
    doesn't support capturing images!";  
  
    Toast toast = Toast.makeText(this,  
        errorMessage, Toast.LENGTH_SHORT);  
  
    toast.show();  
}  
}  
}  
  
}  
  
}  
  
}/* *  
* Handle user returning from both capturing and cropping the  
* image  
*/  
  
protected void onActivityResult(int requestCode, int  
    resultCode, Intent data) {  
    if (resultCode == RESULT_OK) {  
        //user is returning from capturing an image using  
        //the camera  
  
        if(requestCode == CAMERA_CAPTURE){  
            //get the Uri for the captured image  
            picUri = data.getData();  
  
            //carry out the crop operation  
            performCrop();  
        }  
    }  
}
```

```
Button next =new Button(this);
next.setText("Next");
LinearLayout ll =
(LinearLayout)findViewById(R.id.callme); // It could not be a
LinearLayout, of course
ll.addView(next);
next.setOnClickListener(new
OnClickListener() {

    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
        Intent a=new Intent(getApplicationContext(),
        Sound.class);
        startActivity(a);
    }
});
```

}

```
//user is returning from cropping the image
else if(requestCode == PIC_CROP){
    //get the returned data
    Bundle extras = data.getExtras();
    //get the cropped bitmap
    Bitmap thePic = extras.getParcelable("data");
    //retrieve a reference to the ImageView
    ImageView picView = (ImageView)findViewById(R.
    id.picture);
    //display the returned cropped image
    picView.setImageBitmap(thePic);
}
```

}

```
/*
 * Helper method to carry out crop operation
 */
private void performCrop(){
    //take care of exceptions
    try {
        //call the standard crop action intent (the user
        //device may not support it)
        Intent cropIntent = new Intent("com.android.
        camera.action.CROP");
        //indicate image type and Uri
        cropIntent.setDataAndType(picUri, "image/*");
        //set crop properties
        cropIntent.putExtra("crop", "true");
        //indicate aspect of desired crop
        cropIntent.putExtra("aspectX", 1);
        cropIntent.putExtra("aspectY", 1);
        //indicate output X and Y
        cropIntent.putExtra("outputX", 256);
        cropIntent.putExtra("outputY", 256);
        //retrievedata on return
        cropIntent.putExtra("return-data", true);
        //start the activity - we handle returning in
        //onActivityResult
        startActivityForResult(cropIntent, PIC_CROP);
    }
    //respond to users whose devices do not support the crop
    //action
    catch(ActivityNotFoundException anfe) {
        //display an error message
        String errorMessage = "Whoops - your device doesn't
        support the crop action!";
        Toast toast = Toast.makeText(this, errorMessage,
        Toast.LENGTH_SHORT);
    }
}
```

```

        toast.show();
    }

}

@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;
}

}

```

9. Create a new class named **Sound** under **src → com.example.imagecapturedemo** folder.
10. Modify the code in **Sound** class as shown in code snippet 22.

Code Snippet 22:

```

package com.example.imagecapturedemo;

import android.app.Activity;
import android.media.MediaRecorder;
import android.os.Bundle;

public class Sound extends Activity{
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_shoot_and_crop);
        MediaRecorder recorder;

    }
}

```

11. Add a few permissions and the activity class in the **AndroidManifest.xml** file as shown in code snippet 23.

Code Snippet 23:

```
<activity android:name=".Sound" ></activity></application>

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

Figure 6.6 displays the output once the application is executed.

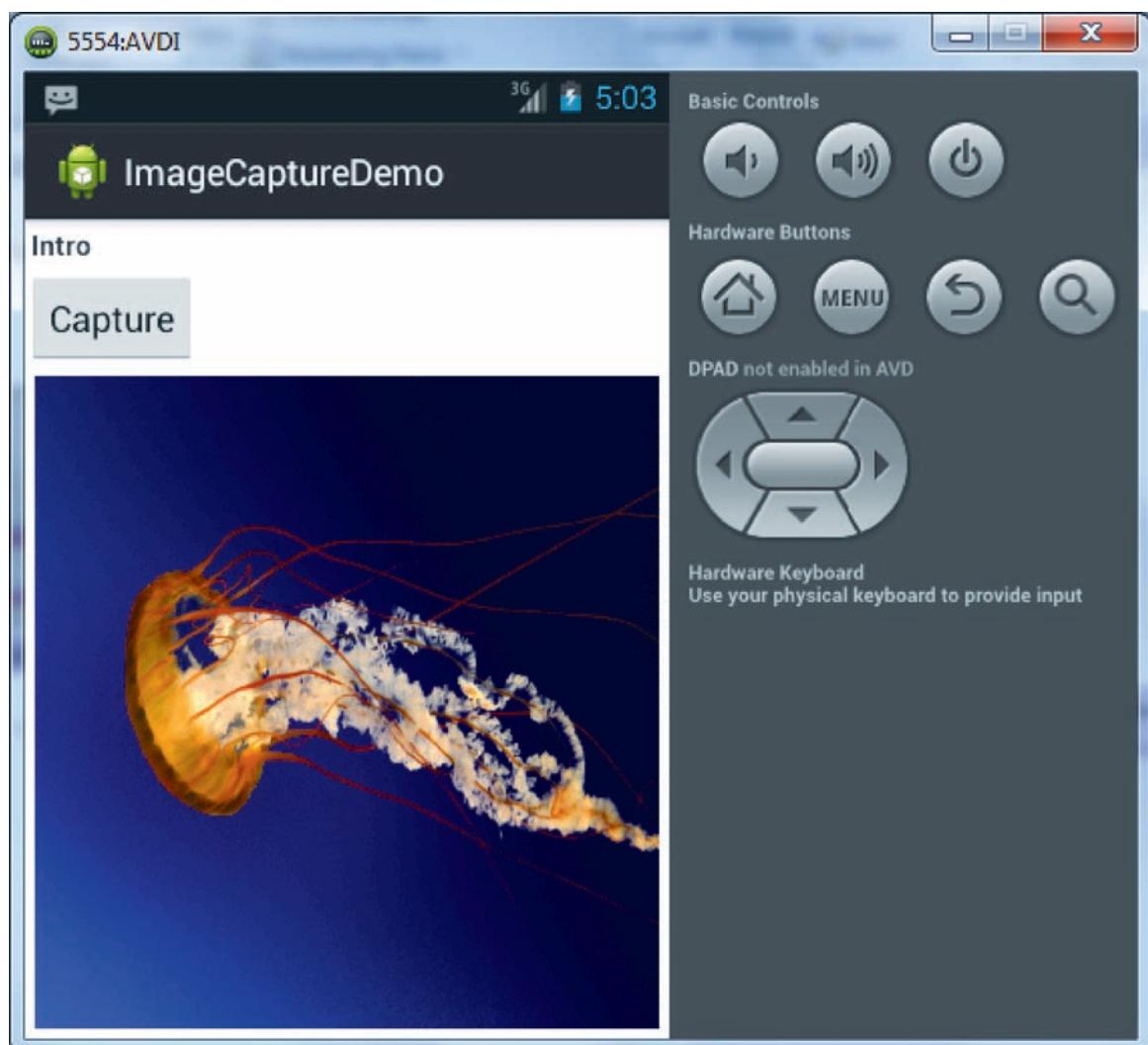


Figure 6.6: ImageCaptureDemo

Note - The capturing of image will take place in the real time Android device.

6.6 Check Your Progress

1. What are the different types of media handling in Android?

(A)	Graphic, Animation, Audio/Video, and Camera	(C)	Sound, Video, and Animation
(B)	Audio, Radio, Speaker, and Application	(D)	Animation

2. Which of the following option represents the version from which Animation was introduced in Android?

(A)	4.3	(C)	3.0
(B)	2.1	(D)	4.0.1

3. Which permission needs to be granted in AndroidManifest.xml file while listening to audio/video?

(A)	android.permission.WAKE_LOCK	(C)	android.permission.INTERNET
(B)	android.permission.SLEEP	(D)	android.permission.READ_DATA

4. Which of the following option represent the class that is of use for incorporating the camera feature?

(A)	android.camera.hardware	(C)	android.show.view.camera
(B)	android.hardware.camera	(D)	android.view.camera

5. Which of the following view object is used in a video application?

(A)	WebView	(C)	Videoview
(B)	Textview	(D)	Surfaceview

6.6.1 Answers

1.	A
2.	C
3.	A
4.	B
5.	C



- Android provides basic level of graphics tools such as canvases, colors filters, points, and rectangles that allows the developer to handle drawing on the screen directly.
- Android framework provides two animation types namely property animation and view animation.
- Property Animation creates an animation by modifying an object's property values over a set period of time with an Animator.
- Tween Animation creates an animation by performing a series of transformations on the contents of a View object.
- Drawable animation allows the developer to load drawable resources and display them one frame after another.
- Android uses OpenCore as its core component of the media framework. OpenCore supports MP3, AAC, AAC+, 3GPP, MPEG-4, and JPEG.
- Android multimedia framework includes support for playing variety of common media types, so that one can easily integrate audio, video and images into your applications.
- Android framework supports capturing images and video through the Camera API or camera Intent.



Samantha is learning to develop Android applications. She now wants to develop an application that will have the following functionalities:

- User will able to view a list of video and audio clips.
- Start and stop the video clips.
- Play, pause, and rewind audios.

“Real generosity towards the future
lies in giving all to the present”

JELLYBEAN
ICE CREAM SANDWICH

Session - 7

Data Handling in Android

Welcome to the session, **Data Handling in Android**.

This session introduces the various ways of handling data in Android. The session proceeds and explains the various ways of storing data using the different storage systems.

In this session, you will learn to:

- ➔ Explain the process of saving and loading preferences
- ➔ Explain persistence of data to file
- ➔ Explain external storage
- ➔ Explain internal storage
- ➔ Explain SQLite Database
- ➔ Explain the process of storing and retrieving data from the database



7.1 Introduction

Android provides different ways of saving persistent application data. The solution of storage that the developer chooses depends on the specific needs, such as whether the data should be private to the application or accessible to other applications (and the user) and how much storage space does the data require. Some of the different data storage options that are provided by Android are as follows:

- **Shared Preference:** Stores all the private primitive data as key-value pairs. This data will be accessed from within the application.
- **Internal Storage:** Stores the private data on the device memory. If the application is uninstalled then the data will be lost.
- **External Storage:** Stores the public data on the shared external storage. If the application is uninstalled then the data will not be lost and will be present in the storage.
- **SQLite Database:** Stores the structured data in a private database.

7.2 Persisting Data to Shared Preferences File

Android provides `SharedPreferences` object to help the developer to save simple application data. Using the `SharedPreferences` object, the developer can save and retrieve the desired data through the use of key/value pairs. The developer needs to specify a key for the data that is required to be saved, and then both the key and its value will be saved automatically to an XML file. It enables the developer to save primitive data types such as boolean, float, long, string, and so on.

The `SharedPreferences` class is present in the `android.content` package and the developer needs to import the class to work with an object of the class.

To use a `SharedPreferences` object, one needs to create a variable of type `SharedPreferences` as shown in code snippet 1.

Code Snippet 1:

```
public class PreferencesActivity extends Activity {
    SharedPreferences prefs;
    String prefName = "MyPref";
    ...
}
```

In the code snippet, a string variable is also created to store the name of the preference file that will be used later by the `SharedPreferences` object.

The following methods can be used to obtain an instance of the SharedPreferences class:

- getSharedPreferences()- Allows the developer to obtain the reference of multiple preference files.
- getPreference()- Allows the developer to obtain the reference of one preference file for the Activity class.

Thus, before using the SharedPreferences object, use the getSharedPreferences() method to obtain an instance of the SharedPreferences object, passing it the name of the preference file, as well as the operating mode that is MODE_PRIVATE as shown in code snippet 2.

Code Snippet 2:

```
//---get the SharedPreferences object---
prefs = getSharedPreferences(prefName, MODE_PRIVATE);
```

Once, an object of the SharedPreferences class is created the developer needs to save some values into the SharedPreferences object. For this, the developer needs to create an instance of the SharedPreferences.Editor class so that the values can be added in batches. This is done by invoking the edit() method on the object of the SharedPreferences class as shown in code snippet 3.

Code Snippet 3:

```
SharedPreferences.Editor editor = prefs.edit();
```

Next, the developer can add key/value pairs to the Editor object by invoking its various methods such as putBoolean(), putString(), putLong(),.putInt(), or putFloat()(which method to use depends on the type of data that is being saved) as shown in code snippet 4.

Code Snippet 4:

```
...
//---save some values using the SharedPreferences object---
editor.putFloat("temperature", 85);
editor.putBoolean("authenticated", true);
editor.putString("username", "Wei-Meng Lee");
...
```

Finally, the commit() method of the Editor class is invoked to commit the changes back to the SharedPreferences object so that the values can be saved to persistent storage. Code snippet 5 displays the use of commit() method.

Code Snippet 5:

```
...
// ---saves the values---
editor.commit();
...
```

Retrieving data from the `SharedPreferences` object is similar. First, one needs to obtain an instance of the `SharedPreferences` object and then use the various methods (depending on the type of data you are retrieving) to retrieve the values based on the keys as shown in code snippet 6.

Code Snippet 6:

```
...
prefs = getSharedPreferences(prefName, MODE_PRIVATE);
float temperature = prefs.getFloat("temperature", 50);
boolean authenticated = prefs.getBoolean("authenticated", false);
String username = prefs.getString("username", "") ;
...
```

The file will be saved under the directory `data/data/<application_package-name>/shared` preference folder. This can be viewed using DDMS perspective.

7.3 External Storage

External storage is another type of storage system in android. The word external signifies that it is outside and all Android-compatible devices support a shared 'external storage' which can be used to save files. This can be a removable storage media (SD card). Files which are saved to the external storage are readable by everyone and can be modified by the user. The modification can only be performed when the developer enables the USB mass storage to transfer files on a computer.

The various advantages of external storage are as follows:

- ➔ External storage will be extending the internal storage space of the android device so that more apps can be stored.
- ➔ External Storage stores all the data in the SD card of a particular android device and the file which is stored in SD card can be retrieved and the data which is stored in that particular SD card can be used in some other device.

The various disadvantages of external storage are as follows:

- If the developer has put a few set of apps on SD card of the android device, then on removal of the SD card, the apps are no longer available for use until and unless the developer replaces back the SD card.
- It takes longer time for your device to boot-up or shut down.
- The apps placed on the SD card require frequent update to support the functionality.

7.4 Internal Storage

As discussed in internal storage, the developer saves the files directly on to the device's internal storage. Also, by default the files which are saved to internal storage are private to that particular application. Any other application cannot access them (nor can the user). Once the user uninstalls the application, the files are removed.

Using an internal file to persist data is simple as the developer can save and retrieve the data using the `Android.content.Context`'s `openFileOutput()` and `openFileInput()` method, which will return an object of `java.io.FileOutputStream` and a `java.io.FileInputStream` class respectively. Using the object of the `FileOutputStream` and `FileInputStream` class, Java file I/O operations can be performed on the data.

To create and write to a private file present in the internal storage, perform the following tasks:

1. Invoke `openFileOutput()` method with the file name and the operating mode. This returns an object of `FileOutputStream` class.
2. Write to the file by invoking the `write()` method.
3. Close the stream invoking the `close()` method.

Code snippet 7 demonstrates the use `FileOutputStream` class.

Code Snippet 7:

```
...
String FILENAME = "android_file_save"; // saving into file
String string = "Hello Internal Storage";

FileOutputStream fos = context.openFileOutput(FILENAME, Context.MODE_PRIVATE);
```

```
// write I/O using write() and close()

fos.write(string.getBytes());
fos.close();
...
```

In the code snippet, the lines `Context.MODE_PRIVATE` helps in creating a file or replacing the file of the same name and also makes it private to the application. It means that our internal files are only visible to us. There are other modes, like `MODE_APPEND`, `MODE_WORLD_READABLE` and `MODE_WORLD_WRITABLE` which are self-explanatory. `MODE_APPEND` is used when data is required to be added at the end of the file, otherwise it will overwrite the file every time something is written to it.

To read a file from internal storage, perform the following tasks:

1. Invoke the `openFileInput()` method and pass the name of the file to read as its argument. This returns an object of `InputStream()` class.
2. Read bytes from the file by invoking the `read()` method.
3. Then, close the stream by invoking the `close()` method.

The absolute path of saved file can be retrieved by invoking the `getFilesDir()` method of the `Context` class. The `fileList()` method will return an array of files saved by the application.

These internal files are deleted once the application itself is removed by the user. In case the developer would like to delete an internal file, in the course of application workflow then the `deleteFile()` method of the `Context` class can be used.

As it can be understood that the internal file storage API is provided by the `Context` class, which basically provides access to our application environment.

The various advantages of internal storage are as follows:

- ➔ In this type of storage, the developer will be storing the apps or the data directly on to your device.
- ➔ All the data and apps saved in the device will be present in the device only.

The various disadvantages of internal storage are as follows:

- ➔ When the device is very low on internal storage space then Android might delete the cache files to recover space.
- ➔ Once the internal memory is full then the device will send an alert informing about memory full and incoming SMS will be rejected.

7.5 SQLite Database

Android uses SQLite database to store relational data. SQLite is a program provided by Android to execute services, such as creating tables and databases, amending rows, executing queries, and so on. The SQLite database created for an Android application is private and cannot be accessed by other applications. For example, if you want to store the results of the students in your school, it is much more efficient to use a database to represent them because you can use the database for querying and retrieve the results according to the specific requirement. SQLite as a database is reliable and is a part of the Android stack. To develop a database application, the following points are to be noted:

- To create a database for your Android App, the developer needs to use the Android API package such as `android.database.sqlite`.
- The developer needs to create a class which is a subclass of `android.database.sqlite.SQLiteOpenHelper` class. This is because this class will enable the developer to create, open, close, and upgrade the database.
- The developer needs to import `android.database.sqliteSQLiteDatabase` class. Using this class instance and its properties the developer can execute the SQL statements to perform the CRUD operations such as `create(insert)`, `retrieve`, `update`, and `delete`.

7.5.1 Storing and Retrieving Data

Storage and retrieval of data is one of the most important tasks that are performed by a database operator. It is the most important characteristics when it comes to database perspective. SQLite queries can be executed by using the `query()` method present in the SQLite database. The `query()` method will return a `Cursor` object that points to all the retrieved rows. In other words, it helps by caching the result of query. It also provides functions with the help of which one can navigate to the desired row and obtain the data from the specified row and columns.

7.5.2 Adding, Modifying, Searching, and Removing SQLite Database

For Android, SQLite is ‘baked into’ the Android runtime, so every Android database based application can create tables in SQLite databases. The databases created in SQLite will be accessible to the classes present in the application and not to the classes present outside the application.

Android provides its own API to deal with database connectivity. This API is provided in android.database and android.database.sqlite packages. In this package the most important classes are as follows:

1. SQLiteOpenHelper
2. SQLiteDatabase
3. Cursor
4. SQLException

→ **SQLiteOpenHelper**

The main functionality of the class is to open the database if it exists, create it if it does not, and upgrade the version as required. It provides a constructor to construct a helper class by subclassing this class and overriding the methods named `onCreate()` and `onUpgrade()`.

The syntax for the constructor is as follows:

Syntax:

```
SQLiteOpenHelper (Context context, String name, SQLiteDatabase.  
CursorFactory factory, int version)
```

where,

`context` – represent the context to create or open the database

`name` – represents the name of the database

`factory` – represents the factory class used for creating the cursor object

`version` – represent the number of the database

Some of the commonly used methods of this class are as follows:

- `onCreate (SQLiteDatabase db)`

The method is invoked when the database is created for the first time. This is where the table is created and populated. The database name is passed as an argument.

- `onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion)`

The method is invoked when the database needs to be upgraded. The implementation should use this method to drop tables, add tables, or perform any other operations such as the need to upgrade to the new schema version.

→ **SQLiteDatabase**

The class has methods to create, delete, execute SQL commands, and perform other common database management tasks. Some of the commonly used methods of this class are as follows:

- `execSQL (String sql)`

This method is used to execute a single SQL statement that is not a SELECT statement or any other SQL statement that returns data. It is used generally for creating table. The sql statement to be executed is passed as an argument.

- `long insert (String table, String nullColumnHack, ContentValues values)`

It is a convenience method used for inserting a row into the database. The method returns the row ID of the newly inserted row, or -1 if an error takes place. The method accepts the table name and the initial column values for the row.

- `int update (String table, ContentValues values, String whereClause, String[] whereArgs)`

It is a convenience method used for updating rows in the database. The method returns the number of rows affected. It accepts the table name, new column values, and an optional where clause.

- `int delete (String table, String whereClause, String[] whereArgs)`

It is a convenience method used for deleting rows in the database. The method returns the number of rows affected if a where clause is passed, otherwise it returns 0.

- `Cursor query (String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)`

The method queries the given table and returns a Cursor over the result set.

→ Cursor

This interface provides random read-write access methods to the result set returned by a database query. Table 7.1 lists some of the methods present in `Cursor` class.

Methods	Description
<code>public int getColumnIndex(String colName)</code>	Returns the index of the indicated column as the result
<code>public String getColumnName(int collIndex)</code>	Returns the name of the specified column as the result
<code>public abstract String[] getColumnNames()</code>	Returns the column names as a string array
<code>public int getColumnIndexOrThrow (String colName)</code>	Returns the index of a column. The concept of throwing an exception arises when there is no existence of the column with the indicated name

Methods	Description
public abstract int getCount()	Returns the count of number of rows
public final int getPosition()	Returns the present cursor position in the result set
public final boolean moveToPosition (int pos)	Moves the cursor towards the desired row in the result set and returns true, if the required record exists
public final boolean moveToFirst()	Moves the cursor to the first row in the queried result set and returns false, if there are no records in the result set
public final boolean moveToNext()	Moves the cursor to the subsequent rows in the result set and returns false, if the cursor is after the last record
public final boolean moveToPrevious()	Moves the cursor to the previous rows in the result set and returns false if the cursor is before the first record

Table 7.1: Methods of Cursor

Besides the methods mentioned, the Cursor interface has `getXXX()` style of methods to retrieve `xxx` type of values. Some of the commonly used methods of this type are as follows:

- `getString(int columnindex)`
- `getLong(int columnindex)`
- `getInt(int columnindex)`
- `getFloat(int columnindex)`
- `getDouble(int columnindex)`

7.6 Working with Content Files

As we know content provider manages access to a central repository of data. This section describes how to create an SQLite database, store the book details, and retrieve it from the database. The steps to create the application are as follows:

1. Start Eclipse.
2. Create a project named `MyAndroidDatabaseApp`.
3. Navigate to `src → com.example.myandroiddatabaseapp` folder.
4. Create a new class named `DBController`.

5. Modify the code in **DBController** file as shown in code snippet 8. This class take care of all the CRUD operations.

Code Snippet 8:

```
package com.example.myandroiddatabaseapp;

import java.util.ArrayList;
import java.util.HashMap;

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

public class DBController extends SQLiteOpenHelper {

    private static final String LOGCAT = null;

    public DBController(Context applicationcontext) {
        super(applicationcontext, "androidsqlite.db", null, 1);
        Log.d(LOGCAT, "Created");
    }

    @Override
    public void onCreate(SQLiteDatabase database) {
        String query;
        query = "CREATE TABLE books ( bookId INTEGER PRIMARY KEY, bookName TEXT )";
        database.execSQL(query);
        Log.d(LOGCAT, "books Created");
    }
}
```

```
@Override  
public void onUpgrade(SQLiteDatabase database, int  
version_old,  
        int current_version) {  
    String query;  
    query = "DROP TABLE IF EXISTS books";  
    database.execSQL(query);  
    onCreate(database);  
}  
  
public void insertBook(HashMap<String, String>  
queryValues) {  
    SQLiteDatabase database = this.getWritableDatabase();  
    ContentValues values = new ContentValues();  
    values.put("bookName", queryValues.get("bookName"));  
    database.insert("books", null, values);  
    database.close();  
}  
  
public int updateBook(HashMap<String, String> queryValues) {  
    SQLiteDatabase database = this.getWritableDatabase();  
    ContentValues values = new ContentValues();  
    values.put("bookName", queryValues.get("bookName"));  
    return database.update("books", values, "bookId" + " = ?" ,  
            new String[] { queryValues.get("bookId") });  
    // String updateQuery =  
    // "Update words set txtWord=' "+word+" ' where  
    // txtWord=' "+oldWord+" ' ";  
    // Log.d(LOGCAT,updateQuery);  
    // database.rawQuery(updateQuery, null);  
    // return database.update("words", values, "txtWord =  
    // ?" , new String[]  
    // { word } );  
}
```

```

public void deleteBook(String id) {
    Log.d(LOGCAT, "delete");
    SQLiteDatabase database = this.getWritableDatabase();
    String deleteQuery = "DELETE FROM books where bookId=" +
        '' + id + "'";
    Log.d("query", deleteQuery);
    database.execSQL(deleteQuery);
}

public ArrayList<HashMap<String, String>> getAllBooks() {
    ArrayList<HashMap<String, String>> wordList;
    wordList = new ArrayList<HashMap<String, String>>();
    String selectQuery = "SELECT * FROM books";
    SQLiteDatabase database = this.getWritableDatabase();
    Cursor cursor = database.rawQuery(selectQuery, null);
    if (cursor.moveToFirst()) {
        do {
            HashMap<String, String> map = new
            HashMap<String, String>();
            map.put("bookId", cursor.getString(0));
            map.put("bookName", cursor.getString(1));
            wordList.add(map);
        } while (cursor.moveToNext());
    }
    // return contact list
    return wordList;
}

public HashMap<String, String> getBookInfo(String id) {
    HashMap<String, String> wordList = new HashMap<String,
    String>();

```

```

        SQLiteDatabase database = this.getReadableDatabase();

        String selectQuery = "SELECT * FROM books where
bookId= '" + id + "'";

        Cursor cursor = database.rawQuery(selectQuery, null);

        if (cursor.moveToFirst()) {

            do {

                // HashMap<String, String> map = new
HashMap<String, String>();

                wordList.put("bookName", cursor.
getString(1));

                // wordList.add(map);

            } while (cursor.moveToNext());
        }

        return wordList;
    }

}

```

6. Navigate to **res → layout** folder.
7. Modify the code in **activity_main.xml** file as shown in code snippet 9.

Code Snippet 9:

```

<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" >

```

```
<RelativeLayout  
    android:id="@+id/relativeLayout1"  
    android:layout_width="fill_parent"  
    android:layout_height="40dp"  
    android:background="#000000"  
    android:orientation="vertical" >  
  
<TextView  
    android:id="@+id/textView1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_marginLeft="5dp"  
    android:text="Books"  
    android:textAppearance="?android:attr/  
textAppearanceLarge"  
    android:textColor="#FFFFFF" />  
  
<Button  
    android:id="@+id/button1"  
    android:layout_width="41dp"  
    android:layout_height="40dp"  
    android:layout_alignParentRight="true"  
    android:layout_alignParentTop="true"  
    android:background="#454545"  
    android:onClick="showAddForm"  
    android:text "+"  
    android:textColor="#FFFFFF"  
    android:textSize="30sp" />  
  
</RelativeLayout>
```

```

<RelativeLayout
    android:id="@+id/relativeLayout1"
    android:layout_width="fill_parent"
    android:layout_height="match_parent"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/relativeLayout1"
    android:orientation="vertical"
    android:layout_marginTop="40dp">

    <ListView
        android:id="@android:id/list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true">
    </ListView>
</RelativeLayout>

```

8. Navigate to **src → com.example.myandroiddatabaseapp**.
9. Modify the code in **MainActivity** file as shown in code snippet 10.

Code Snippet 10:

```

package com.example.myandroiddatabaseapp;

import java.util.ArrayList;
import java.util.HashMap;

import android.os.Bundle;
import android.app.ListActivity;
import android.content.Intent;
import android.view.View;

```

```
import android.widget.AdapterView;
import android.widget.ListAdapter;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.TextView;
import android.widget.AdapterView.OnItemClickListener;

public class MainActivity extends ListActivity {

    Intent intent;
    TextView bookId;
    DBController controller = new DBController(this);
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ArrayList<HashMap<String, String>> bookList =
            controller.getAllBooks();
        if(bookList.size()!=0) {
            ListView lv = getListView();
            lv.setOnItemClickListener(new
                OnItemClickListener() {
                    @Override
                    public void onItemClick(AdapterView<?>
                        parent, View view, int position, long id) {
                        bookId = (TextView) view.findViewById(R.
                            id.bookId);
                        String valBookId = bookId.getText().
                            toString();
                        Intent objIntent = new Intent(getApplicationContext(),
                            EditBook.class);
                        objIntent.putExtra("bookId",
                            valBookId);
                    }
                }
            );
        }
    }
}
```

```

        startActivity(objIntent);

    }

}

ListAdapter adapter = new SimpleAdapter
( MainActivity.this, bookList, R.layout.view_
book_entry, new String[] { "bookId", "bookName" },
new int[] { R.id.bookId, R.id.bookName } );
setListAdapter(adapter);
}
}

public void showAddForm(View view) {
    Intent objIntent = new Intent(getApplicationContext() ,
NewBook.class);
    startActivity(objIntent);
}

}

```

10. Navigate to **res → layout** folder.
11. Create a new android layout file named **view _ book _ entry.xml**.
12. Modify the code in **view _ book _ entry.xml** file as shown in code snippet 11.

Code Snippet 11:

```

<?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="wrap_content"
    android:orientation="vertical" >

    <TextView

        android:id="@+id/bookId"
        android:layout_width="fill_parent"

```

```

        android:layout_height="wrap_content"
        android:visibility="gone" />

<TextView
    android:id="@+id/bookName"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="5dp"
    android:layout_marginTop="5dp"
    android:paddingLeft="6dip"
    android:paddingTop="6dip"
    android:textColor="#A4C739"
    android:textSize="17sp"
    android:textStyle="bold" />

</LinearLayout>

```

13. Navigate to **res → layout** folder.
14. Create a new android layout file named **add_new_book.xml**.
15. Modify the code in **add_new_book.xml** file as shown in code snippet 12.

Code Snippet 12:

```

<?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:background="#CCCCCC"
    android:orientation="vertical"
    android:paddingTop="1dp" >

<TextView
    android:id="@+id/textView1"
    android:layout_width="match_parent"

```

```
        android:layout_height="wrap_content"
        android:background="#000000"
        android:padding="5dp"
        android:text="Add Book"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:textColor="#FFFFFF" />

<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#FFFFFF"
    android:orientation="vertical"
    android:padding="10dp" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="24dp"
        android:layout_marginTop="30dp"
        android:text="Book" />

    <EditText
        android:id="@+id/bookName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView1"
        android:layout_below="@+id/textView1"
        android:ems="10" >
```

```

        <requestFocus />
    </EditText>

    <Button
        android:id="@+id	btnadd"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/bookName"
        android:layout_below="@+id/bookName"
        android:layout_marginTop="32dp"
        android:onClick="addNewBook"
        android:text="Save" />
</RelativeLayout>

</LinearLayout>

```

16. Navigate to **src → com.example.myandroiddatabaseapp.**
17. Create a new class named **NewBook**.
18. Modify the code in **NewBook** file as shown in code snippet 13.

Code Snippet 13:

```

package com.example.myandroiddatabaseapp;

import java.util.HashMap;

import com.example.myandroiddatabaseapp.DBController;

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

```

```
public class NewBook extends Activity {  
    EditText bookName;  
    DBController controller = new DBController(this);  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.add_new_book);  
        bookName = (EditText) findViewById(R.id.bookName);  
    }  
  
    public void addNewBook(View view) {  
        HashMap<String, String> queryValues = new  
        HashMap<String, String>();  
        queryValues.put("bookName", bookName.getText().  
        toString());  
        controller.insertBook(queryValues);  
        this.callHomeActivity(view);  
    }  
  
    public void callHomeActivity(View view) {  
        Intent objIntent = new Intent(getApplicationContext(),  
        MainActivity.class);  
        startActivity(objIntent);  
    }  
}
```

19. Navigate to **res → layout** folder.
20. Create a new android layout file named **activity_edit_book.xml**.
21. Modify the code in **activity_edit_book.xml** file as shown in code snippet 14.

Code Snippet 14:

```
<?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:background="#CCCCCC"
    android:orientation="vertical"
    android:paddingTop="1dp" >

    <TextView

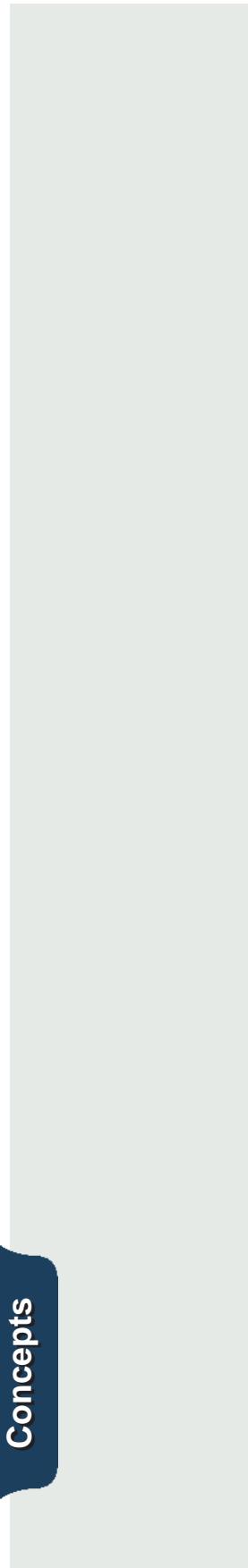
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#000000"
        android:padding="5dp"
        android:text="Edit Book"
        android:textAppearance="?android:attr/
textAppearanceLarge"
        android:textColor="#FFFFFF" />

    <RelativeLayout

        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="#FFFFFF"
        android:orientation="vertical"
        android:padding="10dp" >

        <TextView

            android:id="@+id/textView2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```



```
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="24dp"
        android:layout_marginTop="30dp"
        android:text="Task" />

<EditText
    android:id="@+id/bookName"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView2"
    android:layout_below="@+id/textView2"
    android:ems="10" >

    <requestFocus />
</EditText>

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/bookName"
    android:layout_below="@+id/bookName"
    android:layout_marginTop="19dp"
    android:onClick="editBook"
    android:text="Edit" />

<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```

        android:layout_alignBaseline="@+id/button1"
        android:layout_alignBottom="@+id/button1"
        android:layout_centerHorizontal="true"
        android:onClick="removeBook"
        android:text="Delete" />

    </RelativeLayout>

</LinearLayout>

```

22. Navigate to **src → com.example.myandroiddatabaseapp**.
23. Create a new class named **EditBook**.
24. Modify the code in **EditBook** file as shown in code snippet 15.

Code Snippet 15:

```

package com.example.myandroiddatabaseapp;

import java.util.HashMap;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.EditText;

public class EditBook extends Activity {
    EditText bookName;
    DBController controller = new DBController(this);

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

```

```
setContentView(R.layout.activity_edit_book);

bookName = (EditText) findViewById(R.id.bookName);

Intent objIntent = getIntent();

String bookId = objIntent.getStringExtra("bookId");

Log.d("Reading: ", "Reading all contacts..");

HashMap<String, String> bookList = controller.

getBookInfo(bookId);

Log.d("bookName", bookList.get("bookName"));

if (bookList.size() != 0) {

    bookName.setText(bookList.get("bookName"));

}

}

public void editBook(View view) {

    HashMap<String, String> queryValues = new

    HashMap<String, String>();

    bookName = (EditText) findViewById(R.id.bookName);

    Intent objIntent = getIntent();

    String bookId = objIntent.getStringExtra("bookId");

    queryValues.put("bookId", bookId);

    queryValues.put("bookName", bookName.getText().

    toString());



    controller.updateBook(queryValues);

    this.callHomeActivity(view);

}

public void removeBook(View view) {

    Intent objIntent = getIntent();

    String bookId = objIntent.getStringExtra("bookId");

    controller.deleteBook(bookId);
```

```
this.callHomeActivity(view);

}

public void callHomeActivity(View view) {
    Intent objIntent = new Intent(getApplicationContext() ,
        MainActivity.class);
    startActivity(objIntent);
}

}
```

25. Modify the code in **AndroidManifest.xml** file to add the code before the </application> tag as shown in code snippet 16.

Code Snippet 16:

```
<activity android:name=".NewBook" >
</activity>
<activity android:name=".EditBook" >
</activity>
```

Figure 7.1 displays the output when the application is executed.

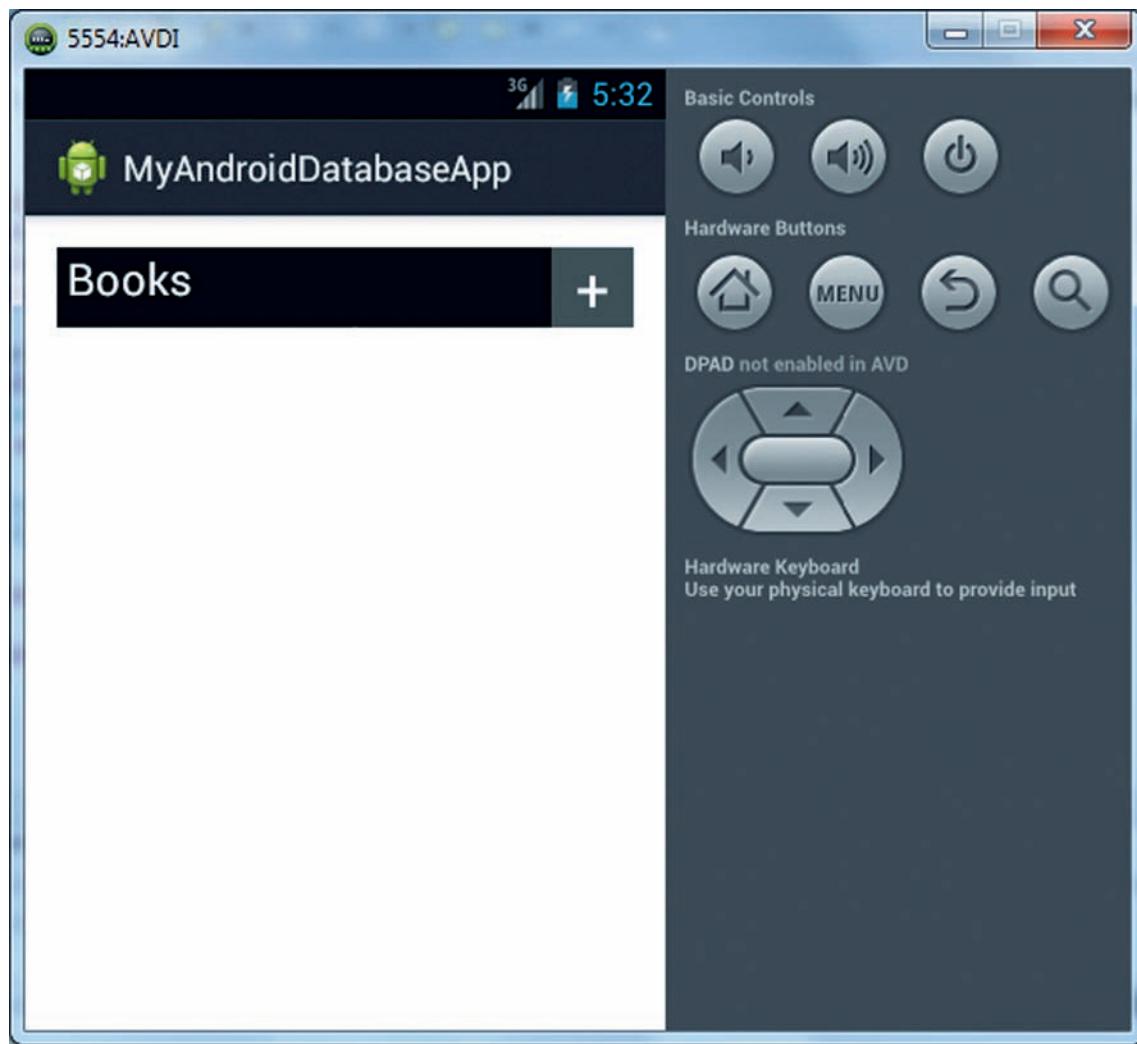


Figure 7.1: Book Database – Output

Figure 7.2 displays the output when the **plus** symbol on the upper right corner is clicked.

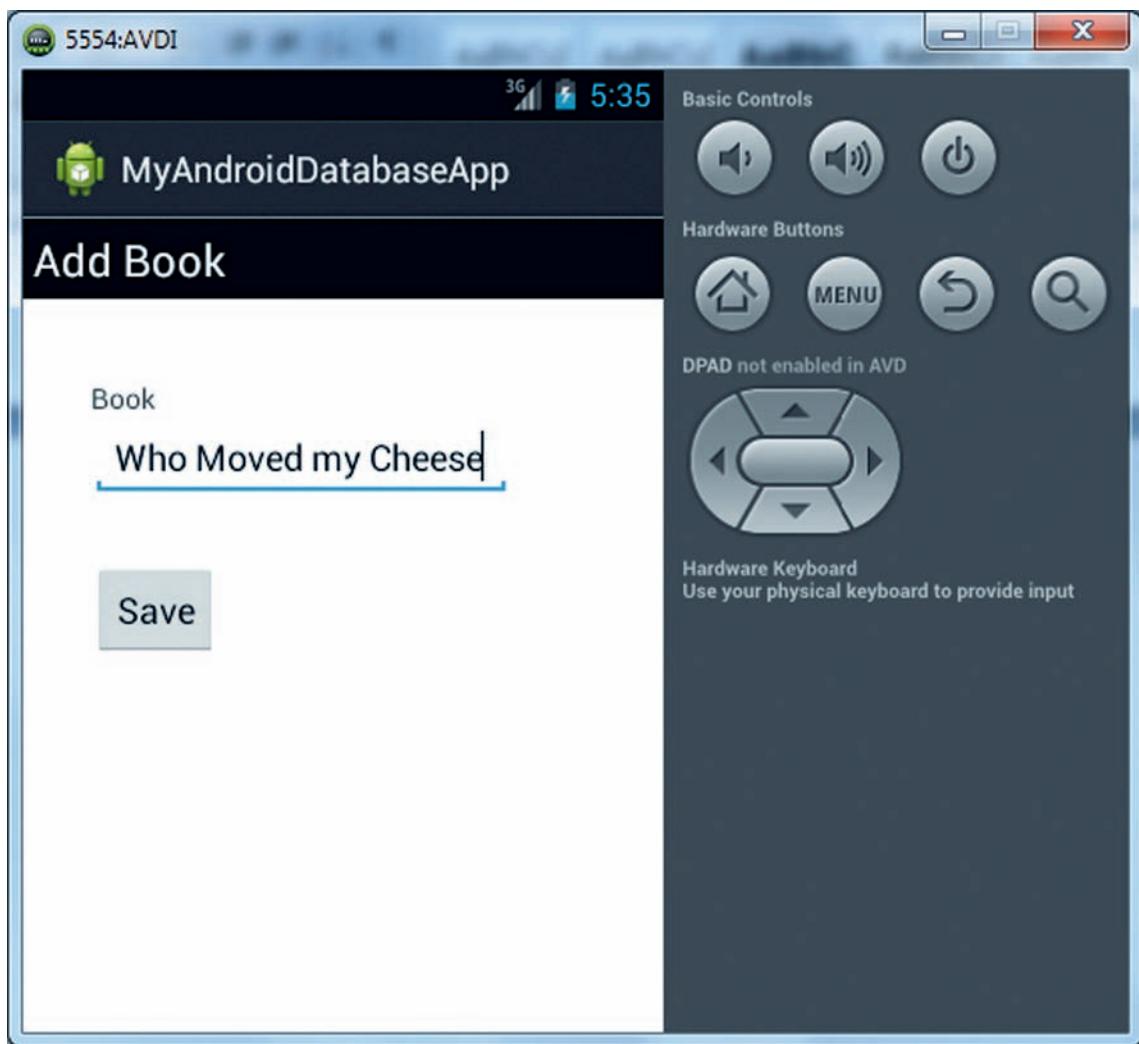


Figure 7.2: Adding Book to the Book Database

Figure 7.3 displays the output when the user clicks **Save** and the book name gets added to the **Book** database.

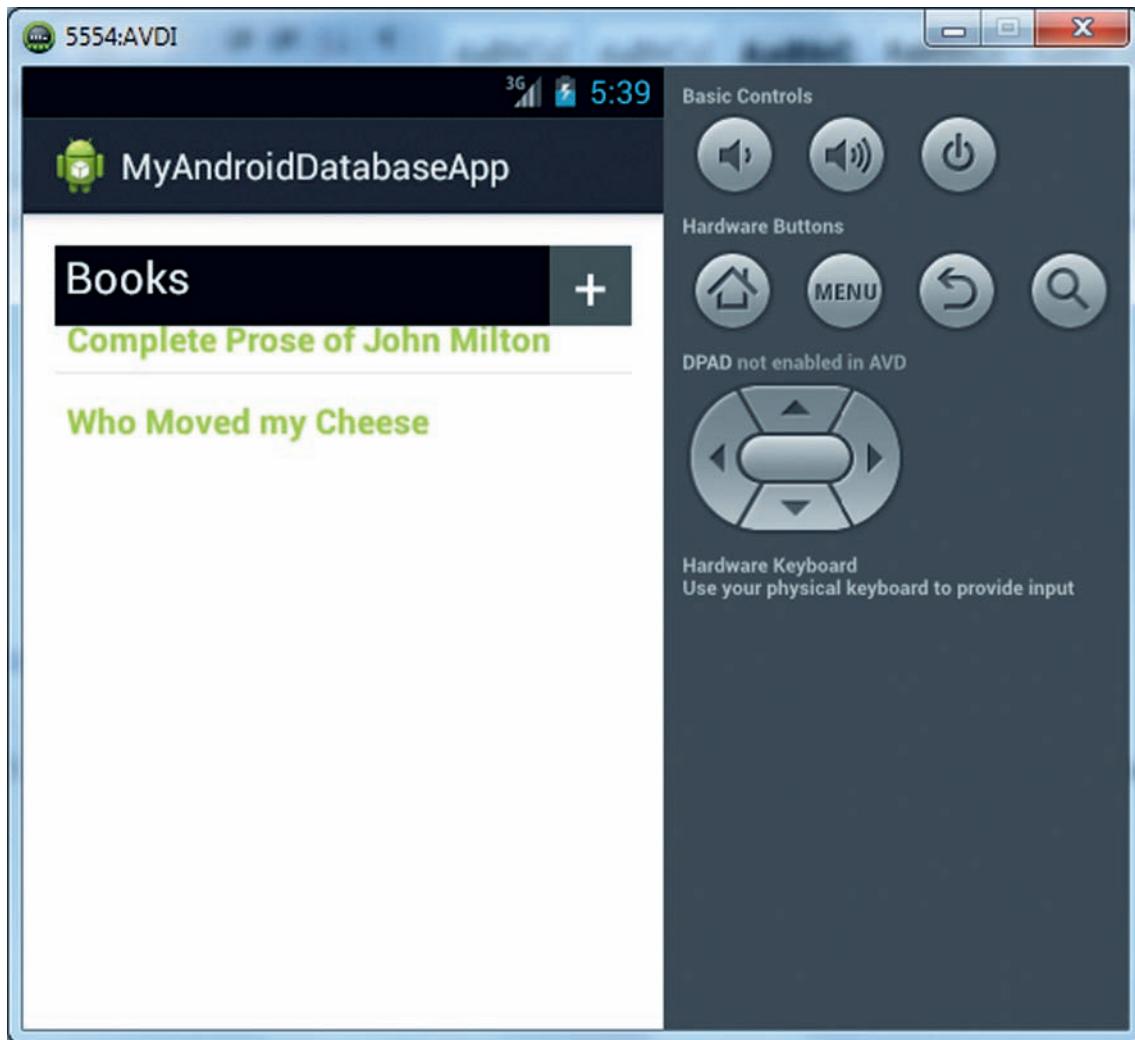


Figure 7.3: Book Added to the Database

Figure 7.4 displays the output when the book is selected to display the **Edit Book** pane.

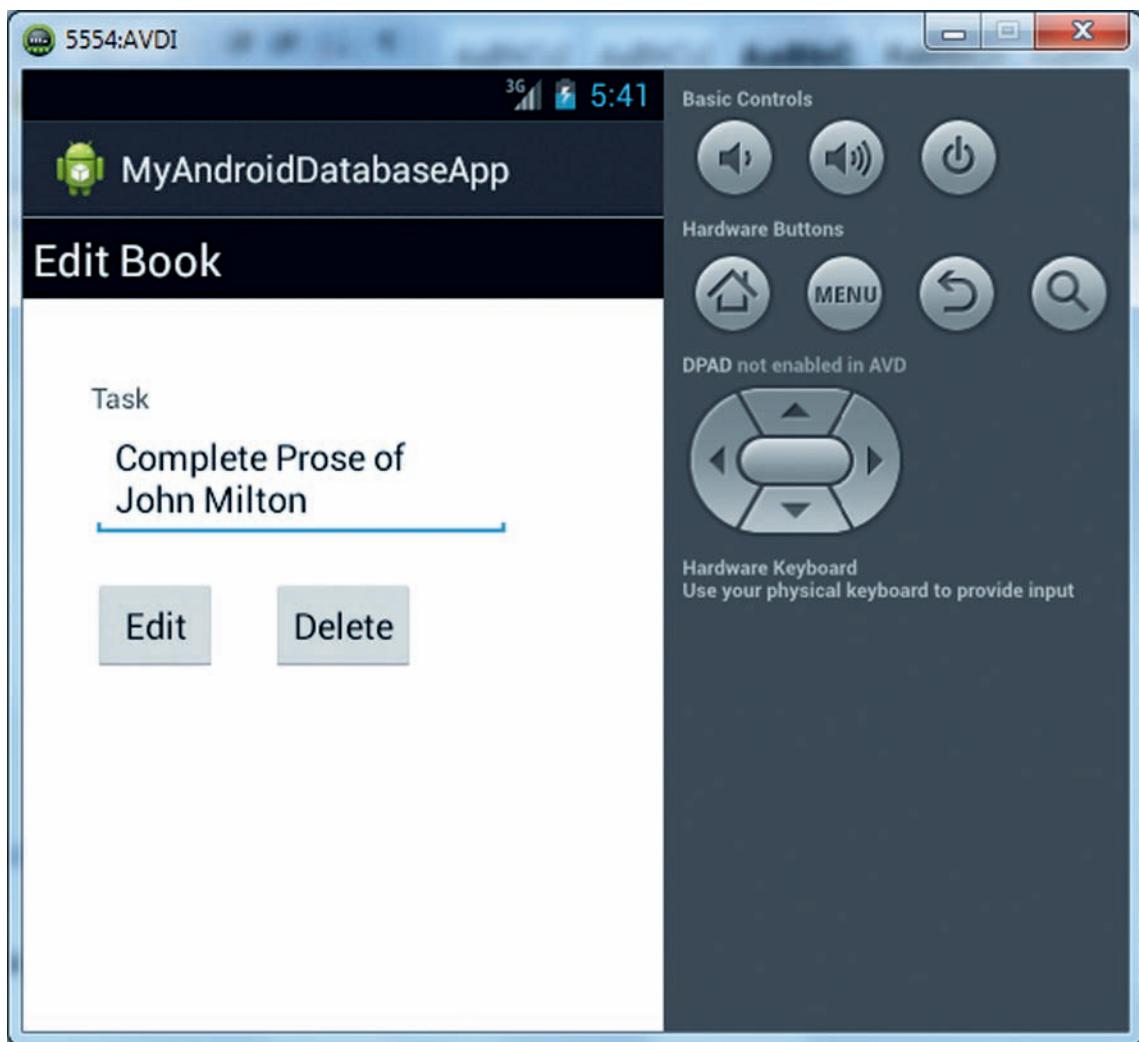


Figure 7.4: Edit Book

Figure 7.5 displays the output after the book name has been edited in the **Edit Book** pane.

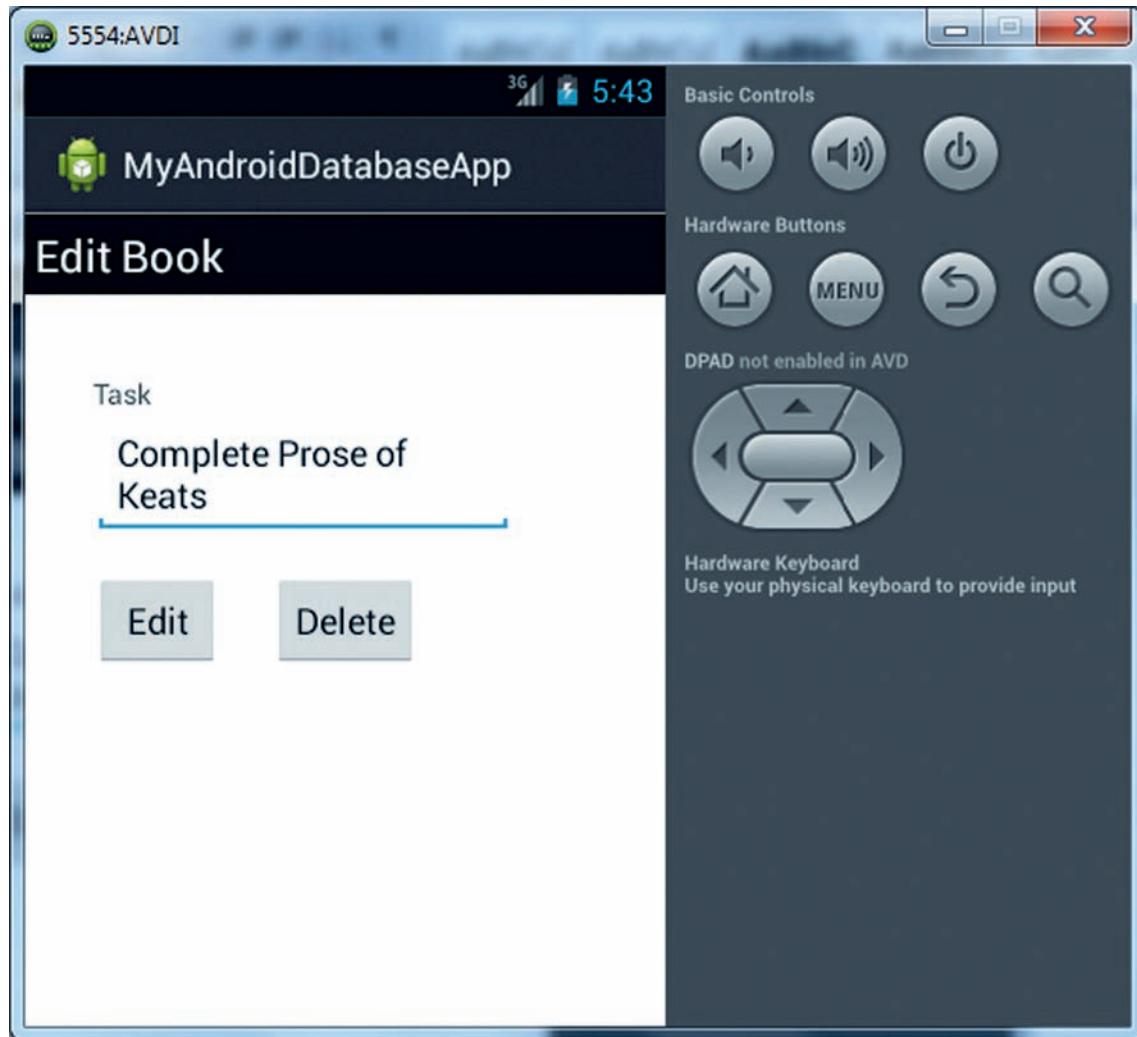


Figure 7.5: Edited Book Name

Figure 7.6 displays the output when the user has clicked **Edit**.

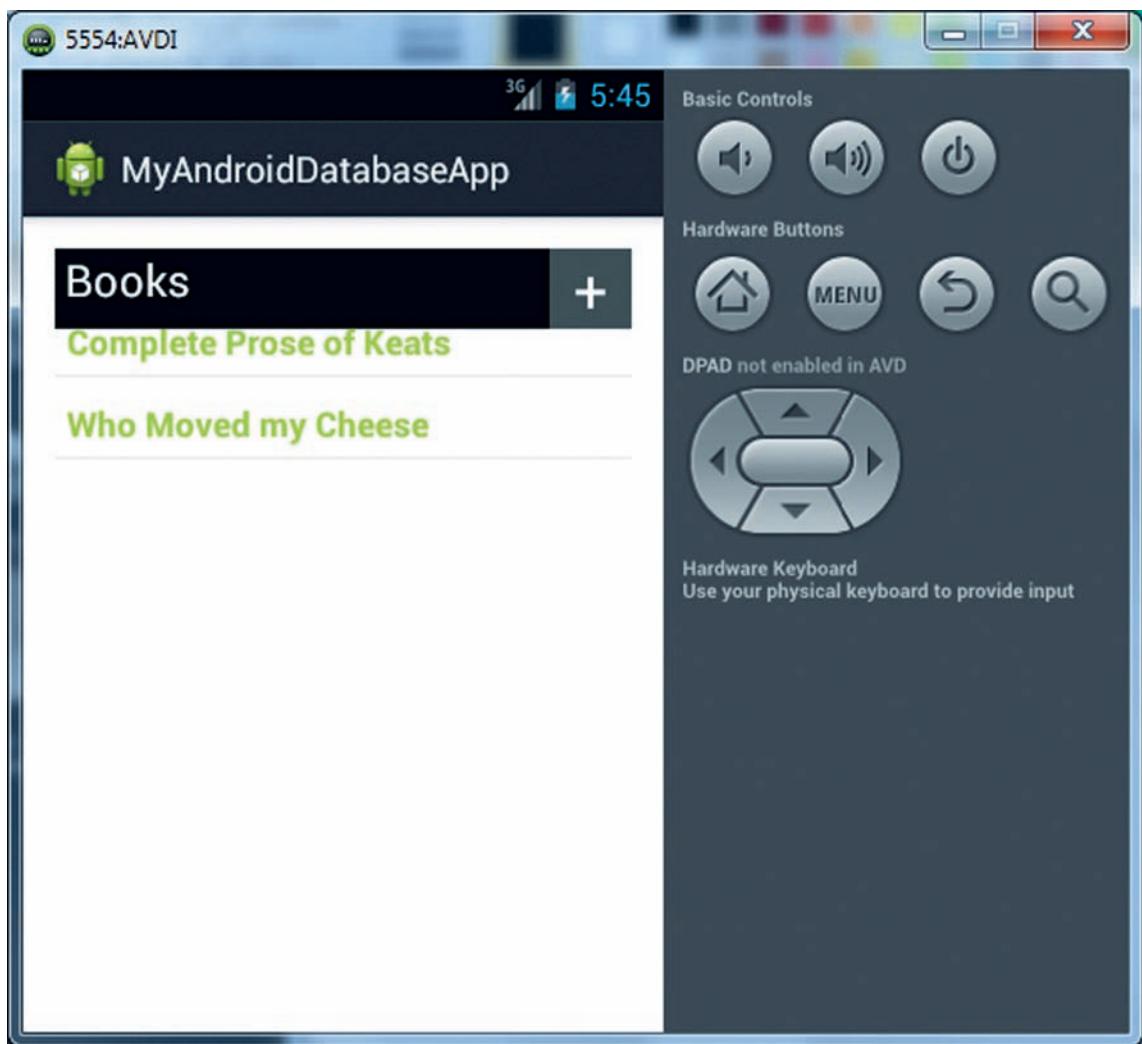


Figure 7.6: Edited BookList

7.7 Check Your Progress

1. Which of the following option is not used for storing data in an Android application?

(A)	Cursor	(C)	SQLiteOpenHelper
(B)	SQLiteDatabase	(D)	SQLException

2. Which of the following option provides random read-write access methods to result set returned by a database query?

(A)	Shared Preferences	(C)	External Storage
(B)	Internal Storage	(D)	Network Storage

3. Which of the following options represents the storage device for External storage?

(A)	Device's Internal Memory	(C)	None of these
(B)	SD card	(D)	Device's Internal Files

4. Which of the following element is required for specifying the name of the additional java files created in your project?

(A)	<uses sdk>	(C)	<uses library>
(B)	<uses permission>	(D)	<application>

5. Which of the following class is used for creating, deleting, and executing SQL commands?

(A)	Cursor	(C)	SQLiteOpenHelper
(B)	SQLiteDatabase	(D)	SQLException

7.7.1 Answers

1.	D
2.	B
3.	B
4.	D
5.	B



- The different types of the storages available in Android are Shared Preferences, Internal Storage, External Storage, and SQLite Database.
- Android provides SharedPreferences object to help the developer to save simple application data.
- The SharedPreferences class is present in the android.content package and the developer needs to import the class to work with an object of the class.
- In Internal storage files are saved directly on to the device's internal storage.
- In External Storage files are saved into the removable storage media (SD card).
- Android uses SQLite database to store relational data.
- SQLite queries can be executed by using the query() method present in the SQLiteDatabase. The query() method will return a Cursor object that points to all the retrieved rows.



Samantha wants to create an admission database system for her college which will enable the admission department of the college to perform the CRUD operation.

“

It is hard to fall,
but it is worse never to have
tried to succeed

”

JELLYBEAN
ICE CREAM SANDWICH

Session - 8

Using Google API

Welcome to the session, **Using Google API**.

This session discusses about the various components that are required to use Google API. It explains the process to create map based activities using Google Maps as a user interface element. The session further proceeds and explains location based services that help the user to find the current location of the device. This session will explain the classes along with their subclasses and examples.

In this session, you will learn to:

- ➔ Explain API and its uses
- ➔ Explain Google API
- ➔ Explain location based service



8.1 Introduction

Every programming language has its own set of API. The question that comes to mind is what is an API? So, it can be explained as follows:

- API stands for Application Programming Interface.
- It is a specification used by software components to communicate with each other.
- It is a library containing specifications for objects, classes, variables, and so on.
- It describes the ways in which a particular task is performed.

The Google API stands for ‘Application Programmable Interface’. It is an interface that helps the programmers in the development of their applications by querying the Google database. Google API’s basically consist of specialized Web services, programs, and specialized scripts. These are used by Internet application developers to find and process information on the Web. In other words, Google APIs are used as an added resource in their applications.

8.2 Using Google API

Application programmers, developers, and integrators usually write software programs that connect remotely to the Google API’s. Google provides its API to assist the developers in easily accessing Google’s Web search database. This will enable and empower the developers in developing software applications that can query billions of Web documents which are constantly being refreshed by Google’s automated crawlers. Programmers can develop applications that will initiate the search queries to Google’s vast index of pages and have results delivered as structured data. The structured data set returned as a result is simple to analyze and work with. Besides this, Google API’s can consistently access data in the Google cache, while at the same time suggests the appropriate words. Google APIs will in fact implement the standardized search syntax used on many of Google’s search properties.

The session will discuss how Google API works with the following:

1. Location Based Services
2. Google Maps

8.2.1 Working with Location Based Services

Location Based Service (LBS) is an information service and has a number of uses in social networking. Location Based Services can be defined as:

- As a set of applications that exploit the knowledge of the geographical position of a

mobile device in order to provide services based on that information.

- LBS provide the mobile clients personalized services according to their current location.

With the incorporation of Global Positioning System (GPS) devices in smartphones, LBS have become important in the past few years. The iPhone was the first to give a huge boost to this kind of applications and now Android continues on the same path.

There are different classes available in the location API to retrieve the location information of the user. Most important classes and interface that are to be used are as follows:

- `LocationManager` class provides access to location service. It includes string constants that return the providers name for two common location providers. The two common location providers are namely, `LocationManager.GPS_PROVIDER` and `LocationManager.NETWORK_PROVIDER`.
- `LocationProvider` class is an abstract class for all the location providers. It provides information on the geographic location of the device by using different technologies.
- `Location` class is a data class that represents the geographic location such as latitude, longitude, and so on.
- `LocationListener` is an interface that provides the callback methods. These methods are invoked when the location changes. The listener object is required to be registered with the location manager.
- `Criteria` is a class that helps the application to select the suitable location provider.

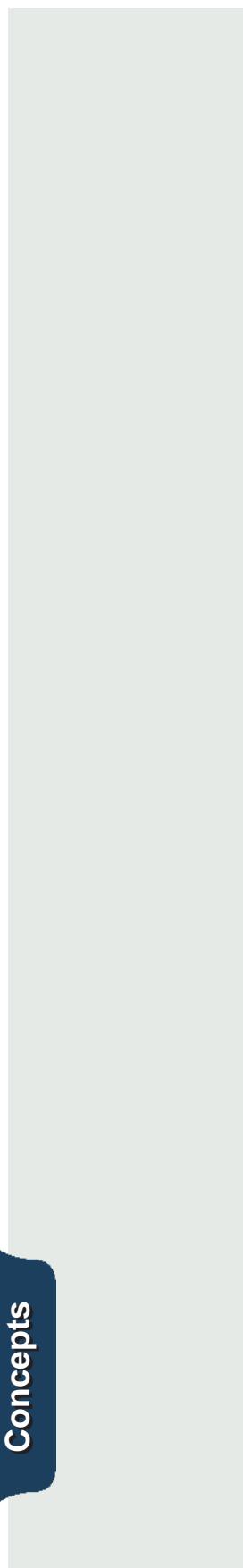
How to build your first LBS application for Android?

The first step in location based service is retrieving the user's current location and then using his/her coordinates to provide data for that location.

In general, the users analytic on the map are feasible in one of the following ways:

- Using the GPS device that comes with the mobile.
- Using the ID of the cell that the user is currently served by.

The first approach is much easier and more accurate (since the second provides only an approximation) as these days a large number of mobile phones have GPS devices incorporated within. The Android SDK's emulator can emulate changes in the user's location and provide dummy data for his coordinates.



To understand the concept, perform the following steps:

1. Start Eclipse.
2. Create a new Android project named **AndroidLbsGeocodingProject1** as shown in figure 8.1.

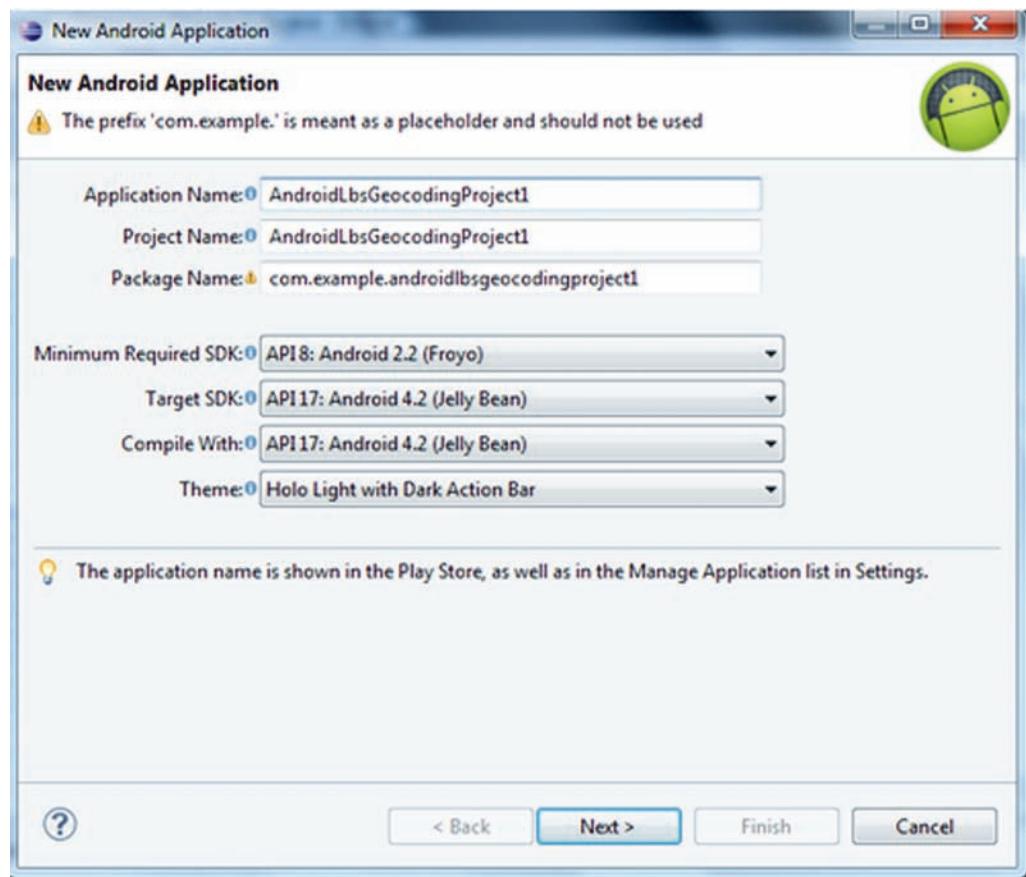


Figure 8.1: AndroidLbsGeoCoding

To start with, a button is required to be added which will result in an event that will trigger the retrieval of the current location's coordinates from a location provider.

3. Navigate to **res → layout** folder.
4. Modify the code in **activity_main.xml** file as shown in code snippet 1.

Code Snippet 1:

```
<?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" >  
  
    <Button  
        android:id="@+id/retrieve_location_button"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="Retrieve Location" />  
  
</LinearLayout>
```

5. Navigate to **src → com.example.androidlbsgeocodingproject1** folder.
6. Modify the code in **MainActivity.java** file as shown in code snippet 2.

Code Snippet 2:

```
package com.example.androidlbsgeocodingproject1;

import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.view.Menu;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {

    private static final long MINIMUM_DISTANCE_CHANGE_FOR_UPDATES = 1; // in Meters
    private static final long MINIMUM_TIME_BETWEEN_UPDATES = 1000; // in Milliseconds

    protected LocationManager locationManager;
```

```
protected Button retrieveLocationButton;

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

    retrieveLocationButton = (Button) findViewById(R.
        id.retrieve_location_button);

    locationManager = (LocationManager)
        getSystemService(Context.LOCATION_SERVICE);

    locationManager.requestLocationUpdates(LocationMana
        ger.GPS_PROVIDER,
        MINIMUM_TIME_BETWEEN_UPDATES,
        MINIMUM_DISTANCE_CHANGE_FOR_UPDATES, new
        MyLocationListener());
}

retrieveLocationButton.setOnClickListener(new
    OnClickListener() {

    @Override
    public void onClick(View v) {
        showCurrentLocation();
    }
}) ;

}

protected void showCurrentLocation() {
    Location location = locationManager
        .getLastKnownLocation(LocationManager.GPS_
        PROVIDER);
```

```
if (location != null) {  
    String message = String.format(  
        "Current Location \n Longitude: %1$s \n  
        Latitude: %2$s",  
        location.getLongitude(), location.  
        getLatitude());  
  
    Toast.makeText(MainActivity.this, message,  
    Toast.LENGTH_LONG)  
        .show();  
}  
}  
  
@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;  
}  
  
private class MyLocationListener implements  
LocationListener {  
  
    public void onLocationChanged(Location location) {  
        String message = String.format(  
            "New Location \n Longitude: %1$s \n  
            Latitude: %2$s",  
            location.getLongitude(), location.  
            getLatitude());  
  
        Toast.makeText(MainActivity.this, message,  
        Toast.LENGTH_LONG)  
            .show();  
    }  
}
```

```
public void onStatusChanged(String s, int i, Bundle b) {  
    Toast.makeText(MainActivity.this, "Provider  
    status changed",  
        Toast.LENGTH_LONG).show();  
}  
  
public void onProviderDisabled(String s) {  
    Toast.makeText(MainActivity.this,  
        "Provider disabled by the user. GPS turned  
        off",  
        Toast.LENGTH_LONG).show();  
}  
  
public void onProviderEnabled(String s) {  
    Toast.makeText(MainActivity.this,  
        "Provider enabled by the user. GPS turned  
        on",  
        Toast.LENGTH_LONG).show();  
}  
}
```

In order to get started with the location capabilities of the Android API, the first step is to obtain a reference of the `LocationManager` class. An instance of this class will provide access to the system location services. This is achieved by invoking the `getSystemService()` method which it inherits it from the `Context` parent class. Then, to obtain the location of the device the developer uses the `requestLocationUpdates()` method. The `requestLocationUpdates()` method accepts the name of the preferred location provider (in our case GPS), the minimum time interval for notifications (in milliseconds), the minimum distance interval for notifications (in meters), and finally a class implementing the `LocationListener` interface.

The interface declares methods for handling changes in the user's location as well as changes in the location provider's status. To implement the `LocationListener` interface, a private inner class named `MyLocationListener` is declared. The class overrides the abstract methods of the interface, `LocationListener`. Toasts messages are used in the method to provide information about the GPS status or any location changes. The only UI element in the application is a button to which a listener is attached which when clicked invokes the `onClick()` method. The `onClick()`

method in turn invokes the user defined `showCurrentLocation()` method. In the `showCurrentLocation()` method, the `getLastKnownLocation()` method of the `LocationManager` instance is executed which returns the last known location and stores it in a `Location` instance. From a `Location` instance, the developer obtains information regarding the user's altitude, latitude, longitude, speed, and so on. In order to execute the application necessary permissions are required to be granted in the **AndroidManifest.xml** file. The permissions to be granted are as follows:

- `ACCESS_FINE_LOCATION`
 - `ACCESS_MOCK_LOCATION`
 - `ACCESS_COARSE_LOCATION`
7. Modify the code in **AndroidManifest.xml** file as shown in code snippet 3.

Code Snippet 3:

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

        package="com.example.androidlbsgeocodingproject1"
        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.
    androidlbsgeocodingproject1.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>  
  
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />  
  
<uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />  
  
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />  
  
</manifest>
```

Next, an AVD Manager is required to be created which will provide GPS support as shown in figure 8.2.

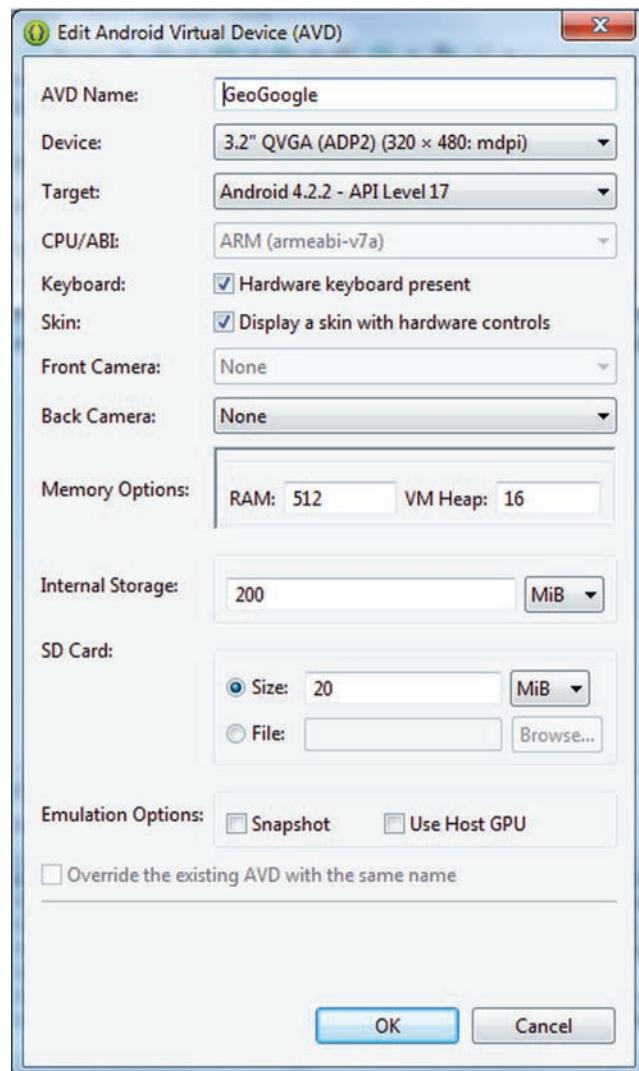
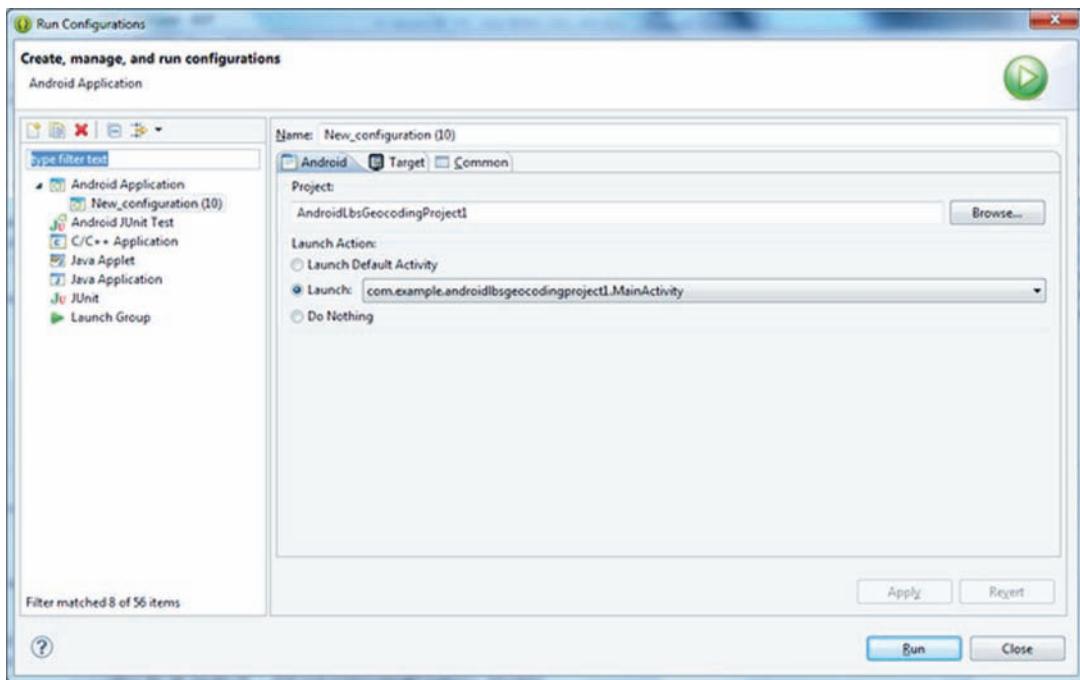


Figure 8.2: Emulator

8. Click **Run → Run Configurations** to create a new configuration for the project as shown in figure 8.3.

**Figure 8.3: Run Configurations**

If the user clicks **Run**, the emulator starts, but nothing will happen because when the emulator starts, there is no last known location to be retrieved (the Location instance that will be returned contains null values). The emulator needs some dummy data. Thus, one has to push the location changes directly into the emulator's GPS location provider. This is performed as follows:

9. Click **DDMS** view of Eclipse.

10. Click **Emulation Control** tab. The **Location Controls** section sends mock location data to the emulator.
11. Click **Send** in the **Manual** tab of the **Location Controls** section as there are already some dummy coordinates as shown in figure 8.4.

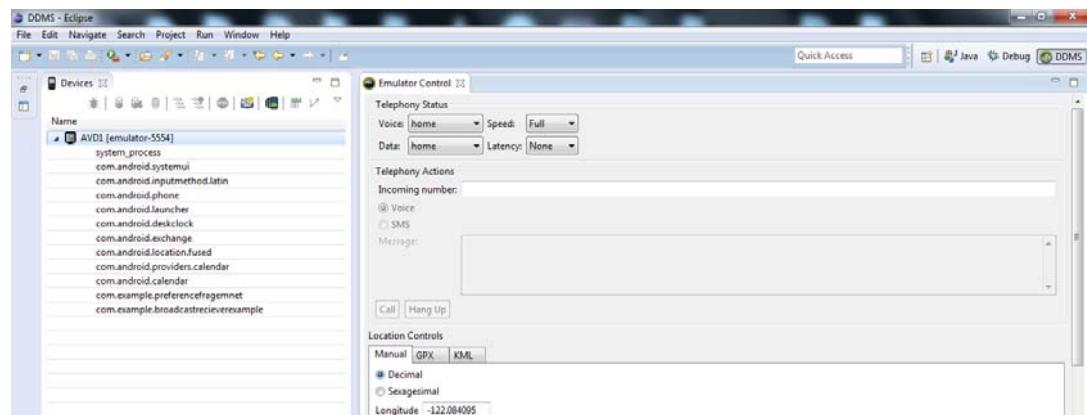


Figure 8.4: DDMS

When the data is sent to the emulator's GPS device by clicking **Send**, the listener which has been added will be triggered and the current location will be printed in the screen in the form of a **Toast** notification as shown in figure 8.5.

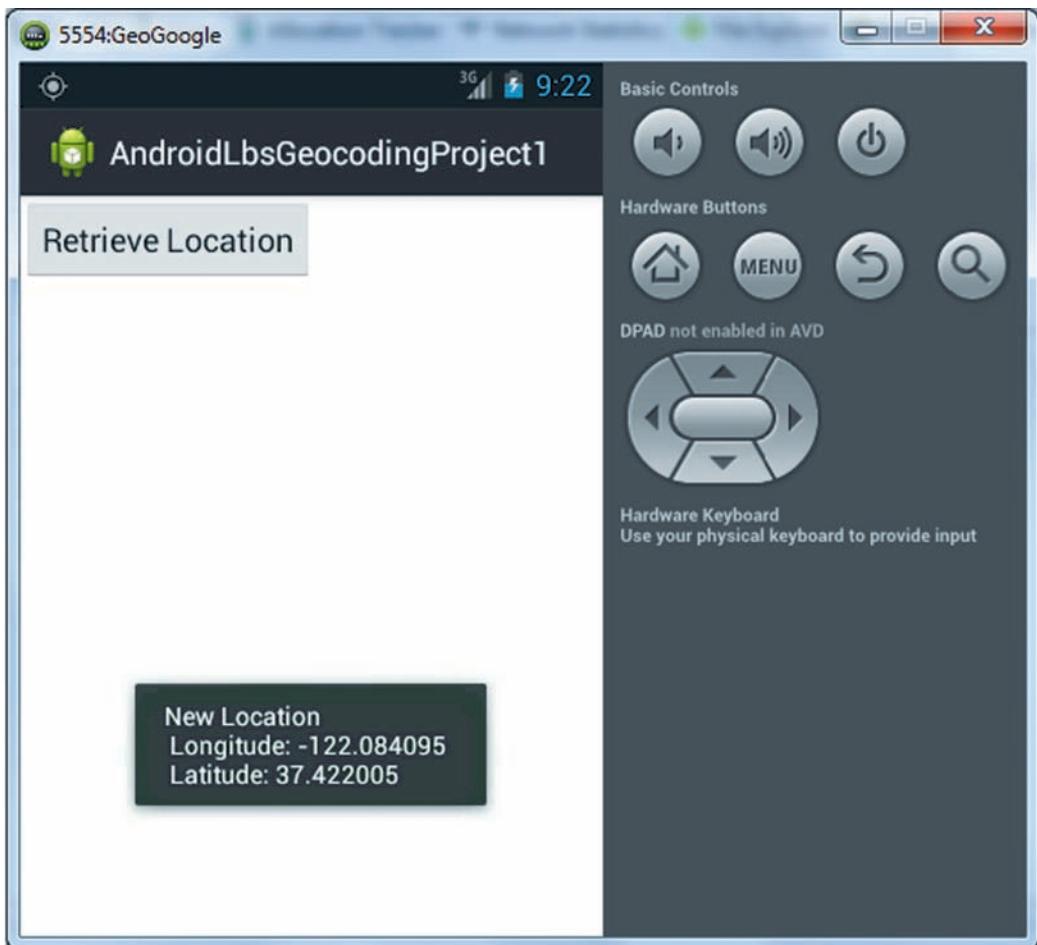


Figure 8.5: Retrieve Location Value

Since, the last known location exists, so when the user clicks **Retrieve Location**, a not null location will be fetched and the same coordinates will again be displayed in the screen as shown in figure 8.6.

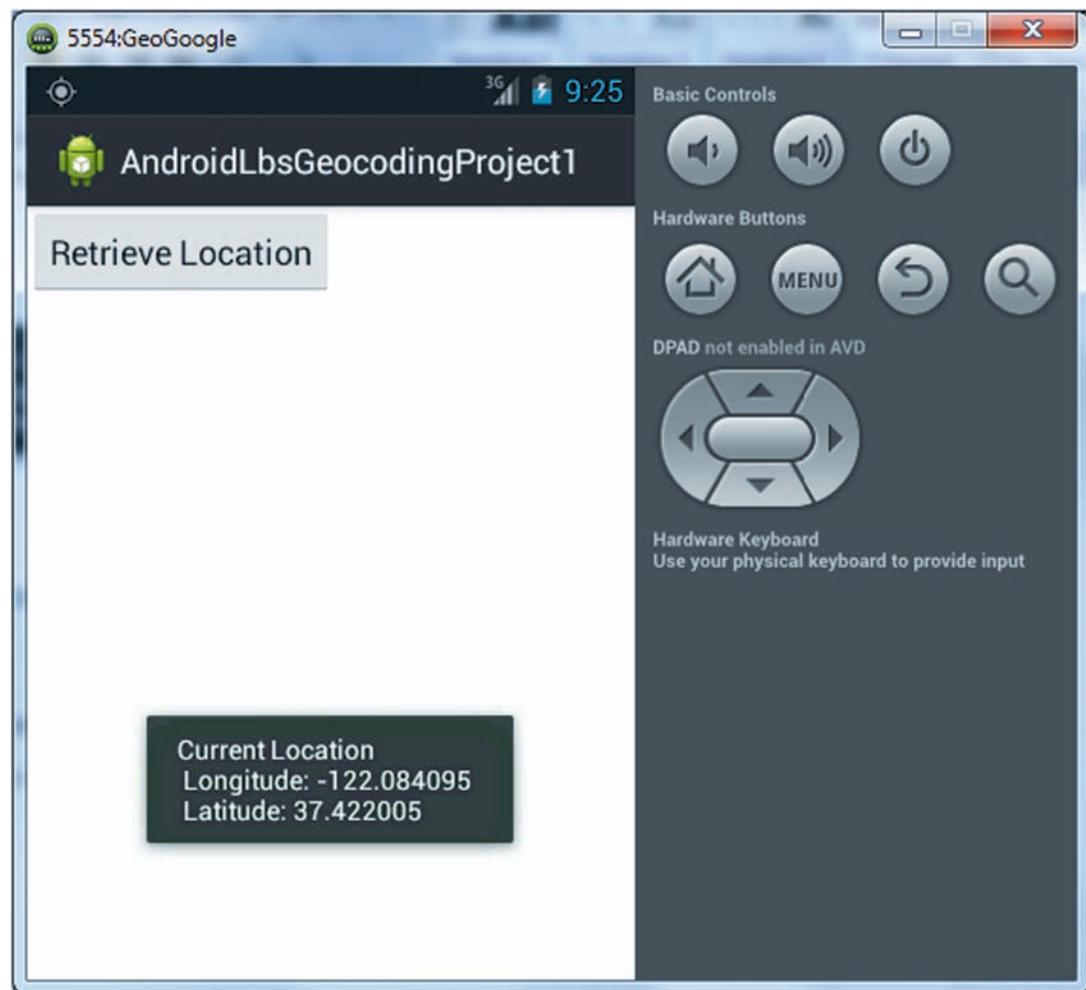
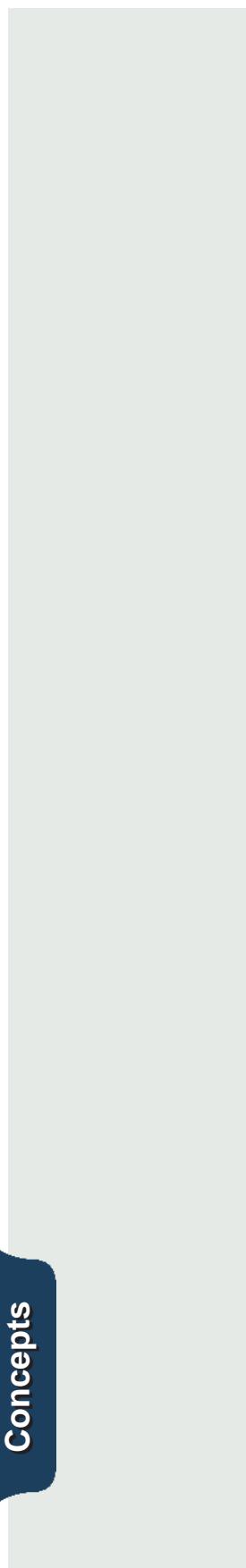


Figure 8.6: Location Show

Thus, the user can change the latitude and longitude in the DDMS and retrieve the updated values.

8.2.2 Working with Google Maps

Google Maps allow the users to explore the world with rich maps provided by Google.

It helps to identify locations with custom markers, augment the map data with image overlays, embed one or more maps as fragments, and much more.

In order to use Google Maps in an Android application, you need to make sure that the application is properly setup to be able to host an Activity Fragment.

Although it only takes a couple of steps to prepare your application for using Google Maps, failing to complete or skipping one of these steps can quickly turn this into an error-prone process.

In short, the following points need to be taken care of to enable the android application to work with Google Maps:

- The Android SDK should be able to setup support for Google Maps.
- The Activity class that will be responsible for showing the map needs to extend from `ActivityFragment` class.
- The application manifest file needs to be setup with the `android.permission.INTERNET` permission.
- The application manifest file is also required to be setup with the Maps key.

Setting up of Android SDK for Google Maps

If the developer wants to work with Google Maps, then the developer has to ensure that the Android SDK has the necessary components installed.

A key component is called the **Google APIs Add-on**, as according to Google : “is an extension to the Android SDK development environment that lets you develop applications for devices that include Google’s set of custom applications, libraries, and services”.

To install Google API, perform the following steps:

1. Navigate to **Window → Android SDK Manager**.
2. Select **Google APIs** under **Android 4.2.2** and **Google Play services** under **Extras**.

3. Click **Install**.

Both the packages will be installed as shown in figure 8.7.

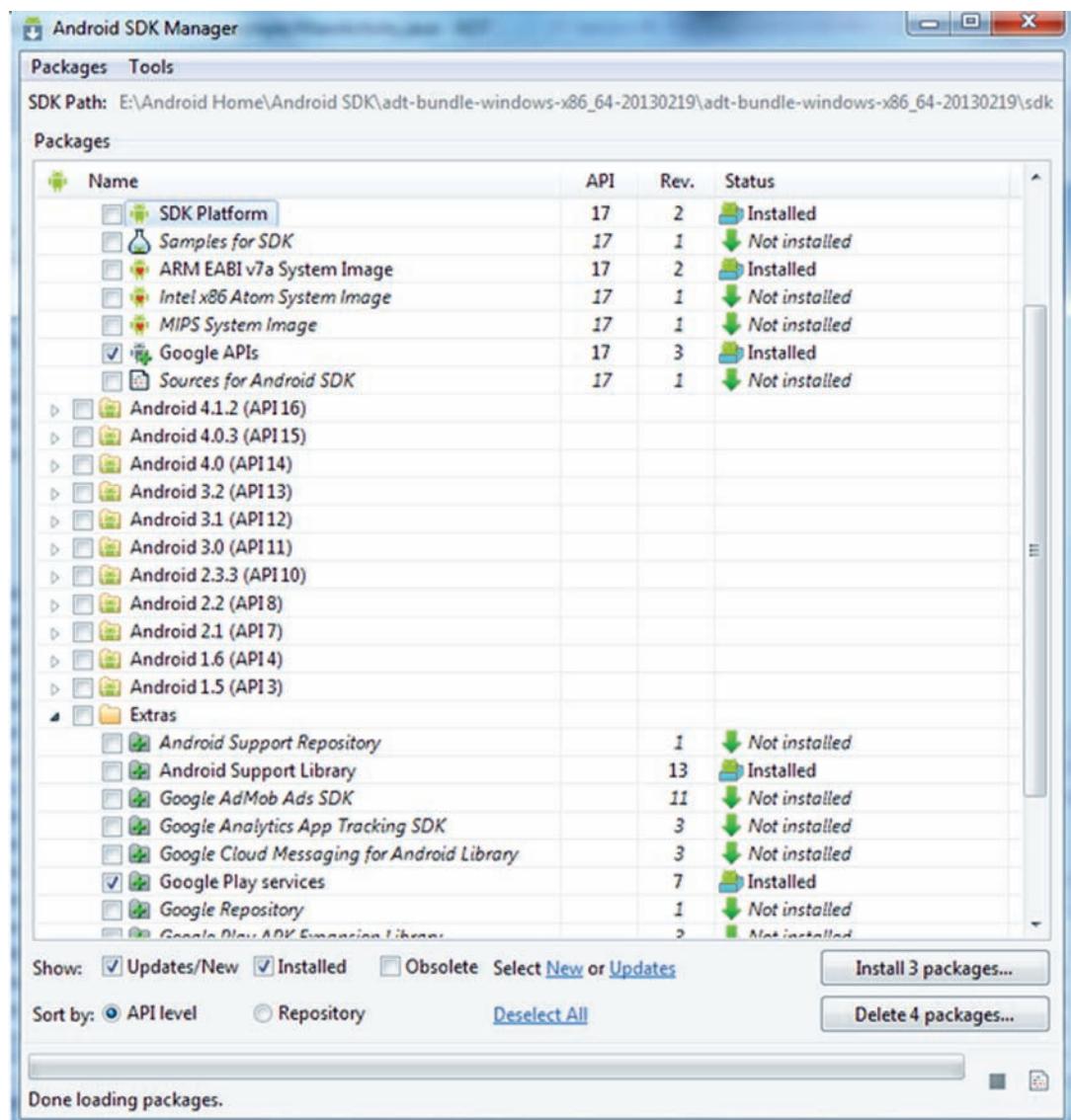


Figure 8.7: Google APIs and Google Play Services

Now that we have setup your Android SDK with the Google APIs and Google Play services, you can create Android Projects in Eclipse that target these APIs. To develop an application, perform the following steps:

1. Start Eclipse.
2. Create a project named **GoogleMaps** as shown in figure 8.8.

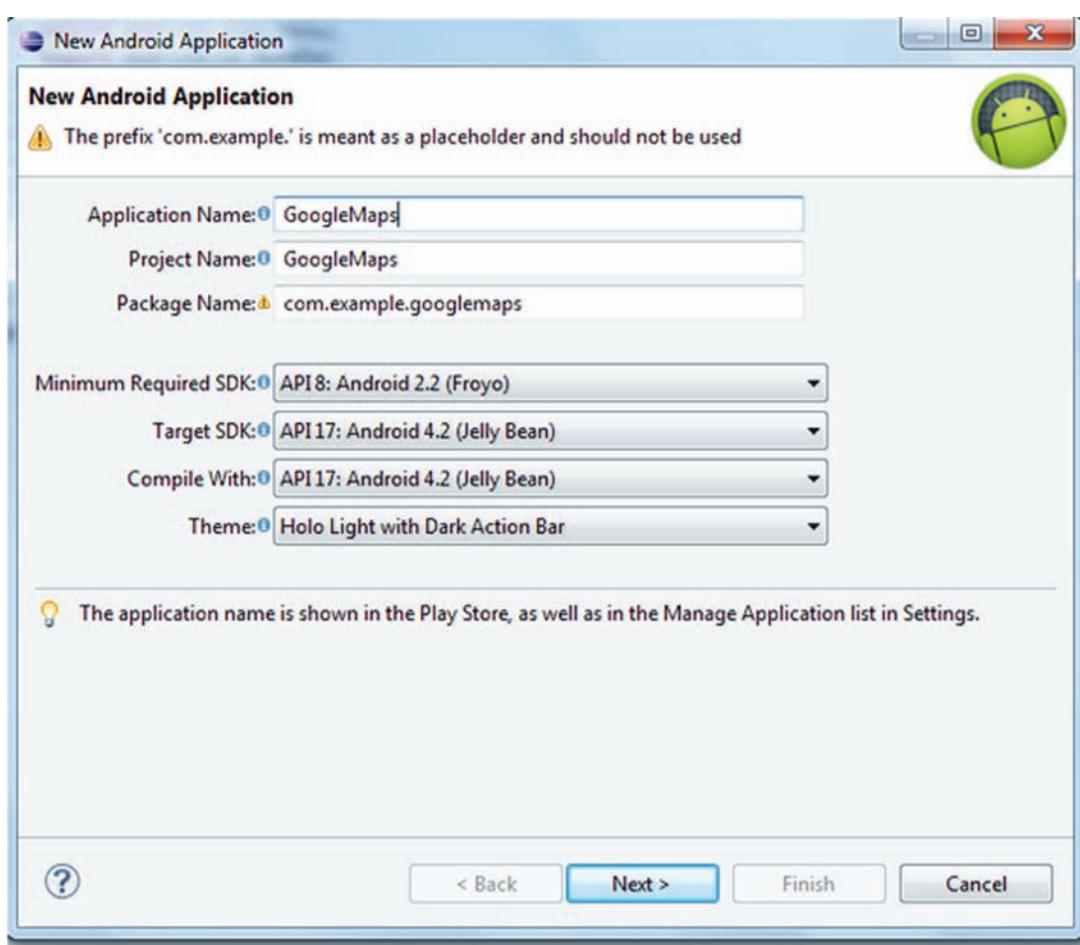


Figure 8.8: GoogleMaps Project in Eclipse

3. Right-click **GoogleMaps** project to display the context menu.
4. Select **File → Import** from the context menu.
5. Select **Android → Existing Android Code Into Workspace** as shown in figure 8.9.

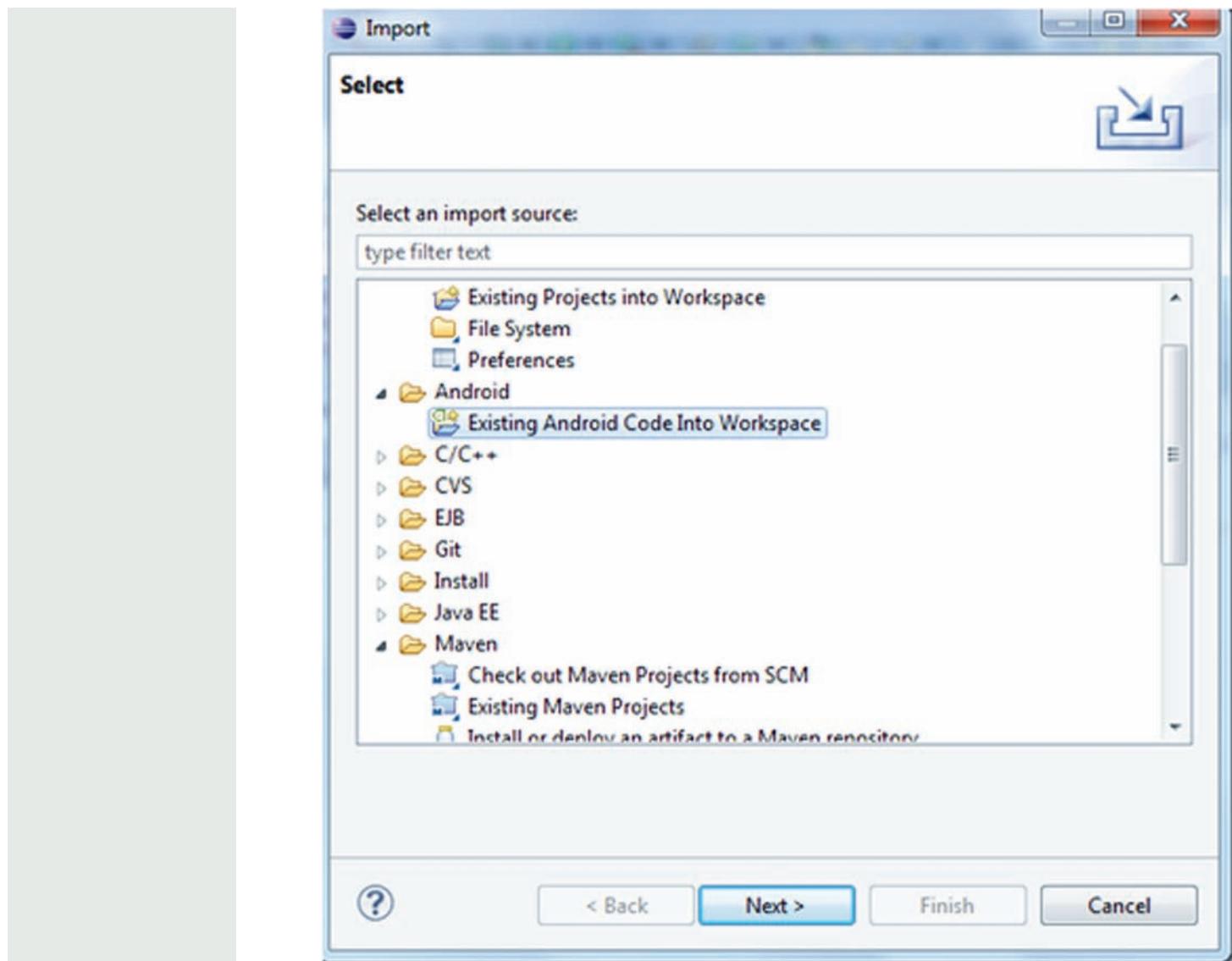


Figure 8.9: Import Existing Project in Workspace

6. Click **Next** to display the **Import Projects** dialog box.
7. Click **Browse** and select **<bundle root> folder → sdk → extras → google → google_play_services → libproject** folder as shown in figure 8.10.

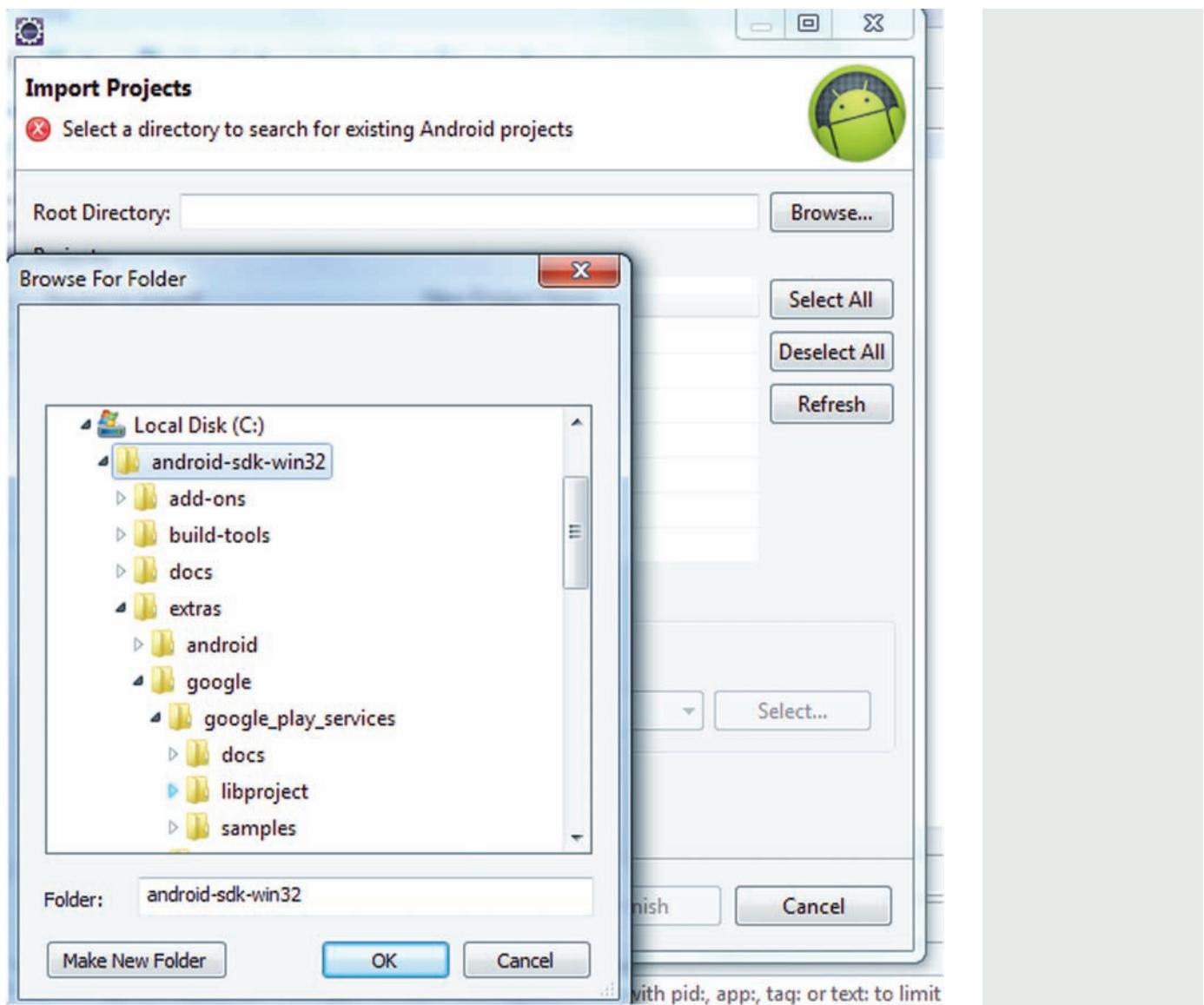


Figure 8.10: Import google plays service

8. Select **google-play-services_lib** under **libproject** folder and click **Finish**.
9. Right-click the project and select **Properties**.
10. Select **Java Build Path** from the left pane and click **Add** under **Project** tab to add the **google_play_service**.
11. **services.**
12. Select **google-play-services_lib** from the list and click **OK** as shown in figure 8.11.

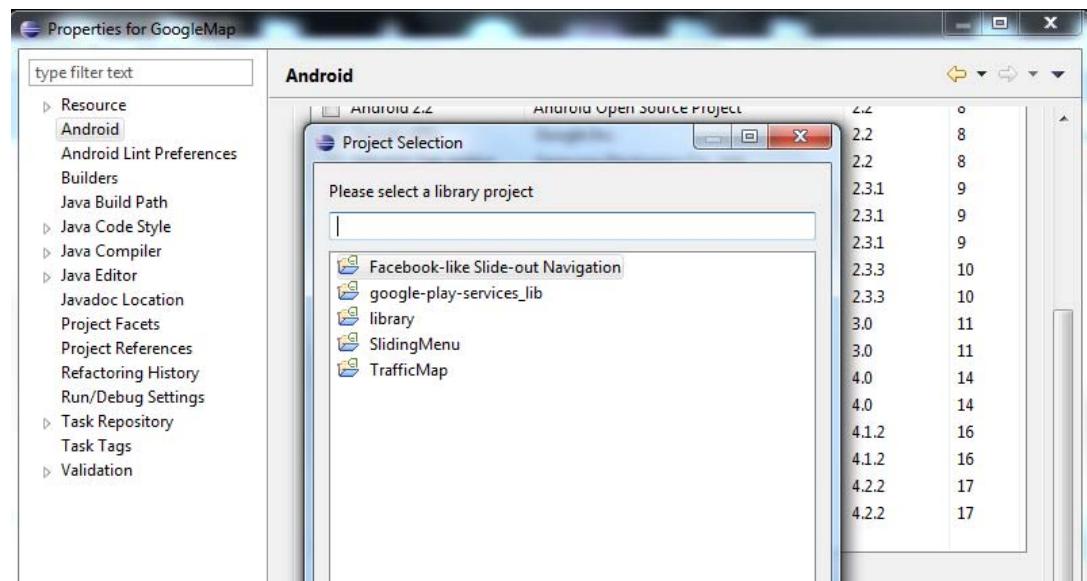


Figure 8.11: Google Play services

13. Click **Apply** and then click **OK** to close the **Properties** dialog box.

Retrieve a Google MAP API Key

Before we can start using maps in our Android application, you need to obtain a Maps API Key. Signing up for such an API key is fairly straightforward, and requires the developer to perform the following:

- Registering the MD5 fingerprint of the certificate that you will use to sign your application. The Maps registration service then provides you a Maps API Key that is associated with your application's signer certificate.

The developer needs to execute a command line tool called **keytool** to display the certificate fingerprint. The **keytool** command is distributed with any Java JDK / JRE and can be located under `%JAVA_HOME%/bin` directory.

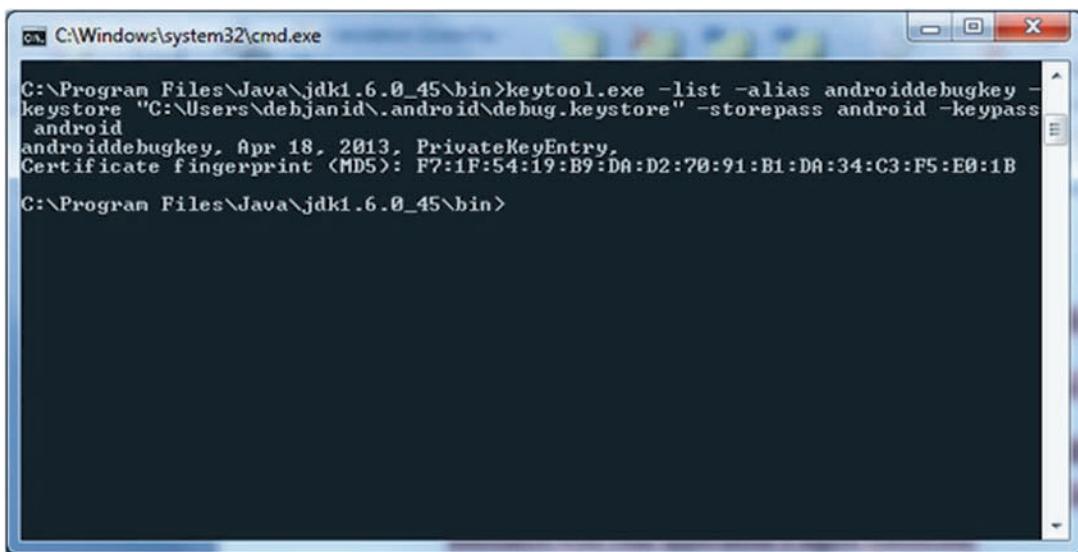
Steps to follow to extract MD5 Fingerprint are as follows:

1. Verify the existence of the debug certificate by going to Eclipse and selecting **Window → Preferences**.
2. Expand **Android** and select **Build**. On the **Build** pane of the preferences dialog box, the path for **Debug default keystore** is specified.
3. Using the debug keystore, the developer needs to extract its MD5 fingerprint. The **keytool.exe** application included with JDK installation is also used.

4. The **keytool.exe** is present in the **C:\Program Files\Java\<JDK_version_number>\bin** folder.
5. Type the following command to extract the MD5 fingerprint:

```
keytool.exe -list -alias androiddebugkey -keystore "C:\Users\<username>\.android\debug.keystore" -storepass android -keypass android
```

Figure 8.12 displays the command in the command prompt and displays the generated MD5 key.

A screenshot of a Windows Command Prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The window contains the following text:

```
C:\Program Files\Java\jdk1.6.0_45\bin>keytool.exe -list -alias androiddebugkey -keystore "C:\Users\debjanid\.android\debug.keystore" -storepass android -keypass android
androiddebugkey, Apr 18, 2013, PrivateKeyEntry,
Certificate fingerprint <MD5>: F7:1F:54:19:B9:DA:D2:70:91:B1:DA:34:C3:F5:E0:1B
C:\Program Files\Java\jdk1.6.0_45\bin>
```

The window has a standard blue title bar and a black body with white text. It is running on a Windows operating system.

Figure 8.12: Google MD5 Keys

6. Enter <https://code.google.com/apis/console/> in the Address bar of a browser. This opens the Google console to register your application for the Maps API.
7. Sign-in with the Gmail id of the developer. If you are using the Google APIs Console for the first time, then click **Create Project** to create a new project named **API Project**.

Figure 8.13 will be displayed once the developer has successfully logged in.

Service	Status
Google Cloud Messaging for Android	No known issues
Google Maps Android API v2	No known issues
Google Maps API v2	No known issues
Google Maps API v3	No known issues
Google Maps Coordinate API	No known issues
Google Maps Geolocation API	No known issues
Google Maps Tracks API	No known issues
Google Play Android Developer API	No known issues
Google+ API	No known issues
Latitude API	No known issues
Places API	No known issues
Search API for Shopping	No known issues
Site Verification API	No known issues
Static Maps API	No known issues

Figure 8.13: Google Console

- Click **Services** from the left pane and activate the **Google Maps Android API v2** as shown in figure 8.14.

Figure 8.14: Activate Google Maps Android API v2

- Click **API Access** from the left pane to request an API key.
- Click **Create new Android Key** to obtain an API key for the registered application as shown in figure 8.15.

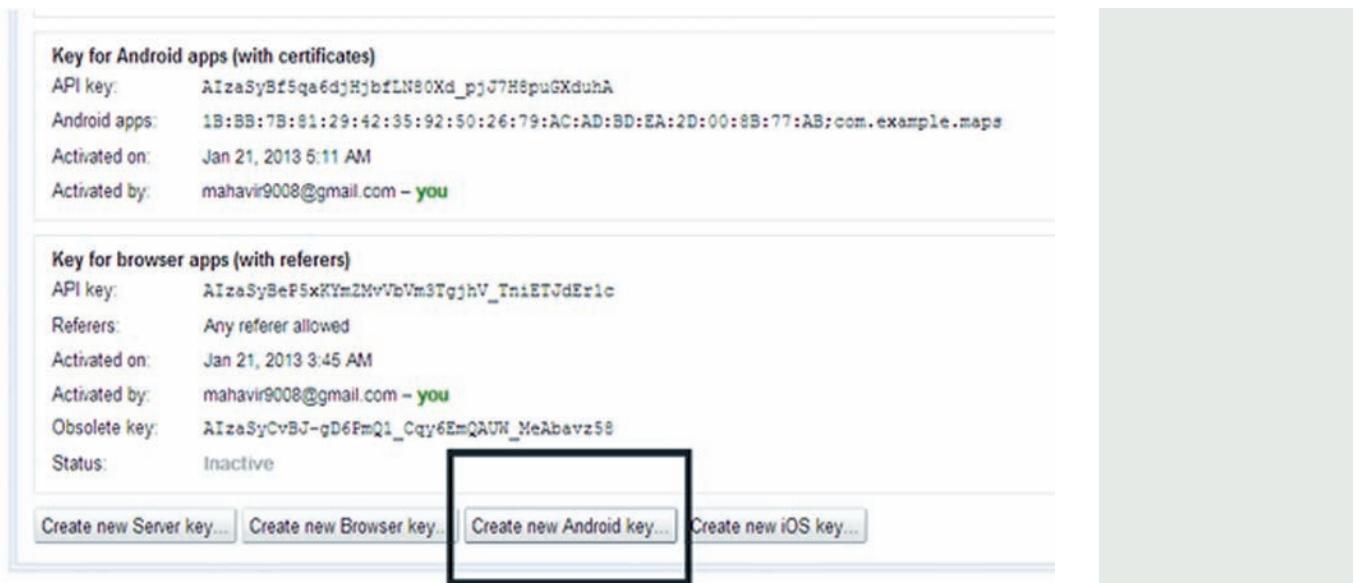


Figure 8.15: Create New Android Key

11. Enter the MD5 Key that was generated on the command prompt followed by semicolon(;) and your application's package name. For example: com.example.googlemaps as shown in figure 8.16.

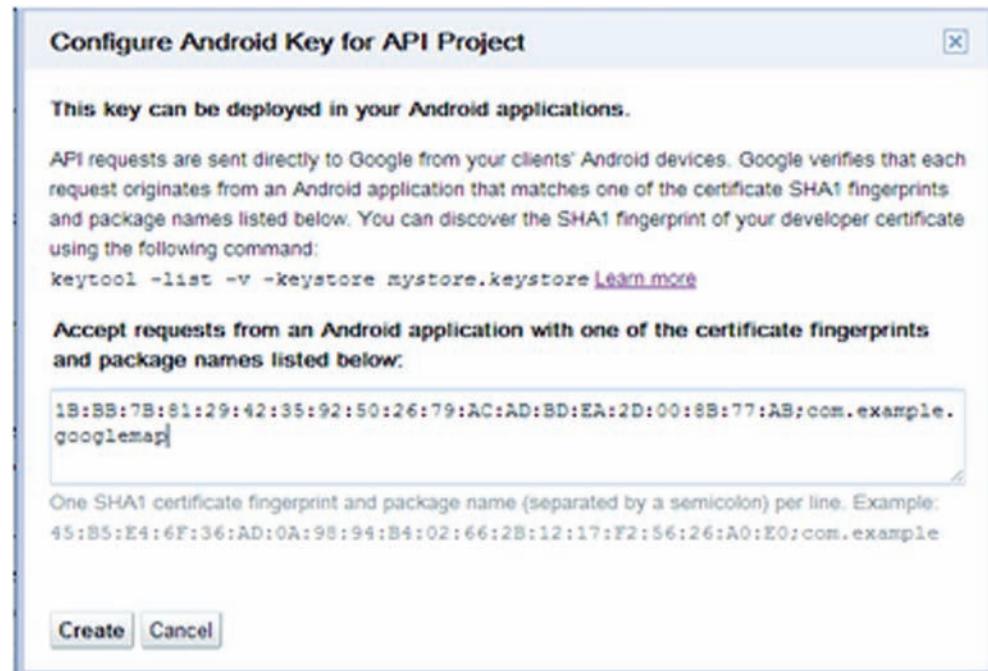


Figure 8.16: Create Key

12. Click **Create** to generate an API key.

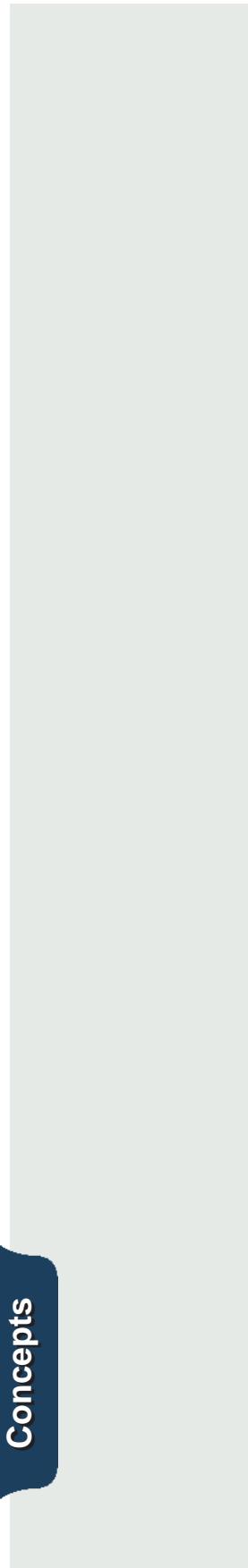
Once the API key has been generated, modify the **AndroidManifest.xml** file with the settings to add the key and set the permissions for your application to access Android features and Google Maps servers as shown in code snippet 4.

Code Snippet 4:

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

        package="com.example.googleproject"
        android:versionCode="1"
        android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />
<permission
        android:name="com.trafficapp.permission.MAPS_
    RECEIVE"
        android:protectionLevel="signature"></permission>
```



```
<uses-permission  
    android:name="com.trafficapp.permission.MAPS_  
RECEIVE"/>  
  
<uses-permission android:name="android.permission.  
INTERNET"/>  
  
<uses-permission android:name="com.google.android.  
providers.gsf.permission.READ_GSERVICES"/>  
  
  
<uses-permission android:name="android.permission.  
ACCESS_NETWORK_STATE"/>  
  
<uses-permission android:name="android.permission.  
ACCESS_COARSE_LOCATION"/>  
  
<uses-permission android:name="android.permission.  
ACCESS_FINE_LOCATION" />  
  
  
  
<uses-feature  
    android:name="android.hardware.location"  
    android:required="false" />  
  
<uses-feature  
    android:name="android.hardware.location.network"  
    android:required="false" />  
  
<uses-feature android:name="android.hardware.location.  
gps" />  
  
<uses-feature  
    android:name="android.hardware.wifi"  
    android:required="false" />  
  
  
<uses-feature  
    android:glEsVersion="0x00020000"  
    android:required="true" />  
  
<application  
    android:allowBackup="true"  
    android:icon="@drawable/ic_launcher"
```

```
    android:label="@string/app_name"

    android:theme="@style/AppTheme" >

    <activity

        android:name="com.example.googleproject.
        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>

    <meta-data

        android:name="com.google.android.maps.v2.API_
        KEY"

        android:value="AIzaSyBIIA9IT9mWABdooEVj53G5-U-
        4uuahMCI" />

</application>

</manifest>
```

As can be seen in the manifest file the `android.permission.INTERNET` permission has been granted and the use of the library, `com.google.android.maps` has been specified.

Note that the `use-library` element is a child of the `application` element. Android won't flag the manifest with an error if the `uses-library` is placed outside the `application`, and it's a mistake that is often made.

Important:

The API key generated is linked to the Android SDK keystore that one is currently using. This means that if you execute your application on another development environment, a different Android SDK keystore is installed and the same API key will not work.

13. Modify the code in **activity_main.xml** file as shown in code snippet 5.

Code Snippet 5:

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

<RelativeLayout xmlns:android="http://schemas.android.com/
apk/res/android"

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

    <LinearLayout

        android:layout_width="fill_parent"
        android:layout_height="fill_parent">

        <fragment

            android:id="@+id/map"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            class="com.google.android.gms.maps.
SupportMapFragment" />

    </LinearLayout>
</RelativeLayout>
```

14. Also, modify the code in the **MainActivity** file as shown in code snippet 6.

Code Snippet 6:

```
package com.example.googleproject;

import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.SupportMapFragment;

import android.os.Bundle;
import android.app.Activity;
import android.support.v4.app.FragmentActivity;
import android.view.Menu;

public class MainActivity extends FragmentActivity {

    private GoogleMap map;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //map = ((SupportMapFragment)
        getSupportFragmentManager()
            .findFragmentById(R.id.map)).getMap();

        map.setMapType(GoogleMap.MAP_TYPE_NORMAL);
    }
}
```

```
@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;  
}  
}
```

15. Build your application and run the application on an Android device to see a map as seen in figure 8.17.

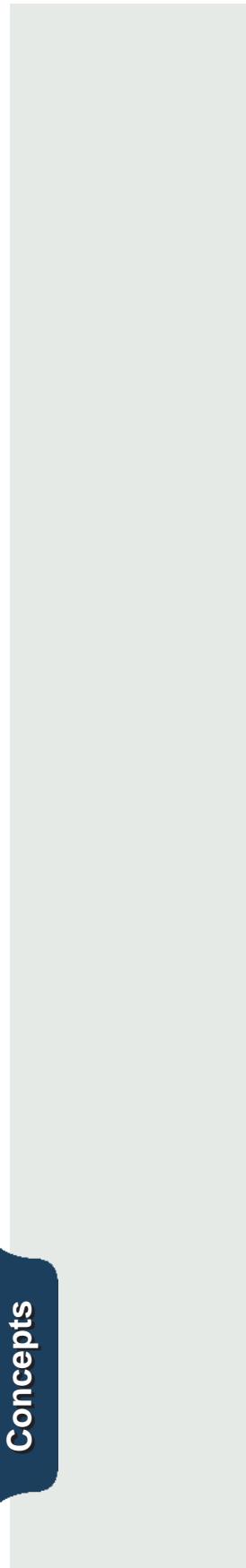
The steps to deploy Android application on an Android-enabled mobile device are as follows:

1. Connect the mobile device through USB cable.
2. Enable USB to connect to storage/device.
3. Copy <application-name>.apk (bin folder of project) to a folder on the mobile device.
4. Disconnect the USB from storage (need not remove the cable, just disable the connection).
5. Click the <application-name>.apk file to install the application on the mobile device.
6. The application is installed and an app icon appears in the application area on the screen.
7. Click the application icon to run it.



Figure 8.17: Google Map on Real Time Device

Note - This application will run only on the real device.



Key developer features

→ Add maps to your app

With version 2 of the Google Maps Android API, the developer can embed maps into an activity as a fragment with a simple XML snippet. The new Maps offer exciting features such as 3D maps, indoor, satellite, terrain, and hybrid maps; and much more.

→ Customize the map

Markers can be added to the map to indicate special locations of interest for the users. Besides these, the developer can also define custom colors or icons for the map markers to match the application's look and feel. The developer can draw polylines and polygons to indicate the path or region, or provide complete image overlays and thus enhance the maps further.

→ Control the user's view

Users can be provided a different view of the world with the ability to control the rotation, tilt, zoom, and pan properties of the 'camera' perspective of the map.

8.3 Check Your Progress

1. Which of the following option correctly represents MD5?

(A)	Media Data 5 logo	(C)	Message Digit 5 algorithm
(B)	Message Digital 5	(D)	Multi Dimensional Identity

2. Which of the following options enables the developer to generate the MD5 fingerprint?

(A)	Console	(C)	Google Map APIs
(B)	Notepad	(D)	automatically generated

3. Which of the following information is displayed by the Location class?

(A)	Integer and Float	(C)	Latitude, Longitude
(B)	Name and key	(D)	Key, Value

4. The Google Map API key is linked to _____.

(A)	Emulator	(C)	Android DDMS
(B)	Android AVD Manager	(D)	Android SDK keystore

5. Which of the following option represents LBS?

(A)	Location Blue Server	(C)	Local Server Boosh
(B)	Location Based Service	(D)	Local Based Service

8.3.1 Answers

1.	C
2.	A
3.	C
4.	A
5.	B



- API stands for Application Programming Interface.
- Location-based services (LBS) refer to ‘a set of applications that exploit the knowledge of the geographical position of a mobile device in order to provide services based on that information’.
- Google Maps allow the developers to develop applications of world with rich maps provided by Google.
- Google API Add on is an extension to Android SDK development environment that lets the developer develop applications for devices that include Google’s set of custom applications, libraries, and services.
- The MD5 fingerprint of the certificate needs to be registered so that it can be used to sign the application.
- The Maps registration service then provides a Maps API Key that is associated with the application’s signer certificate.
- The API key is linked to the Android SDK keystore that is being currently used.



Try it Yourself

Samantha wants to develop an application that will include the following functionalities:

1. Indicate the current location in the map.
2. Display the traffic of your current location.
3. Print the current location in terms of latitude and longitude.

Session - 9

Services, Broadcast Receivers, and Intent Filters

Welcome to the session, **Services, Broadcast Receivers, and Intent Filters**.

This session explains certain Android components that can be used in the Android application. It describes what Android services are, and how they are implemented in the application. It also describes how systems or application events are registered by an Android component called, broadcast receiver.

Further, it also explains how intents allow the application to request functionality from other components of the Android systems.

In this session, you will learn to:

- ➔ Explain services
- ➔ Explain service lifecycle
- ➔ Describe broadcast receiver and its working
- ➔ Explain filters
- ➔ Explain intent matching and its rules
- ➔ Explain filters in manifest file and broadcast receivers



9.1 Introduction

To develop an Android application, the four main components that play an important role are activities, services, content providers , and broadcast receivers. This session discusses services and broadcast receivers in detail. Services are executed in the background to perform long-running operations whereas broadcast receivers responds to announcements. The session also explains Intents which are used for activating components.

9.2 Services

Android application consists of different components which are the building blocks for an application. A system can enter the application through different components which acts as a point of entry. Though all components cannot act as a point of entry to the application but each one has its own entity and plays a specific role. Service is one of the components.

9.2.1 Overview of Services

A Service as discussed earlier is an Android component that can perform background tasks without providing a user interface. It is the base class for all services, which when extended needs to be created in a new thread. As the service uses the main thread by default, it slows down the performance of any running application. Service class creates application components that are specifically suited to handle functions that should run at the background. Any other Android component can start a service and even if the user switches to another application, a service will continue to run in the background. In other words, service is ideal for those cases where a UI need not be present to the user. For example, playing music in the background while working with an application in the foreground. Additionally, a component can communicate with a service by binding itself to it.

Service has higher priority when compared with an inactive Activity. Thus, when there is a need of resources, Service component are less likely to be killed. If a service has been terminated, it can be configured to restart when the resources are available. Using of service ensures that the application will run and respond even when they are not actively used.

9.2.2 Implementing Services

This section will explain how to create each type of service and use it from other application components.

Services are declared in the application's manifest file. To declare a service in the **AndroidManifest.xml** file, add a `<service>` element as a child of the `<application>` element as shown in code snippet 1.

Code Snippet 1:

```
<manifest ... >
    ...
<application ... >
    <service android:name=".SampleService" />
    ...
</application>
</manifest>
```

To define properties, attributes such as permissions to start the service and the process in which the service should run, can be included in the `<service>` element. The `android:name` attribute specifies the class name of the service and is the only required attribute. Once the application is published, the name should not be changed, as it might break some functionality where explicit intents are used to reference the service.

9.2.3 Creating a Service

A component starts a service by invoking the `startService()` method. It results in a call to the Service class `onStartCommand()` method. The lifecycle of this started service is not dependent on the component it starts, and can run in the background indefinitely. The service either stops itself by invoking the `stopSelf()` method, or is stopped by another component when `stopService()` method is invoked.

While starting a service, a component can also pass an Intent that specifies the service and include any data for the service to use. The Intent is received by the Service class in the `onStartCommand()` method. For example, some data needs to be uploaded on an online database. The service can be started by an activity and provide the data to be saved by passing an intent to `startService()` method. The service will then receive the intent in the method `onStartCommand()`, connect to the Web, and perform the database transfer. When the transfer is done, the service stops itself, and is destroyed.

Services are launched on the main application thread, meaning that any processing done in the `onStartCommand()` method will happen on the main GUI thread.

To create a started service, following are the two classes from which your service class can extend from:

- ➔ **IntentService** – `IntentService` is a subclass of `Service` class that handles all start requests one at a time. It uses a worker thread and is the best option to avoid handling of multiple requests simultaneously. The implementation of `onHandleIntent()` method will result in receiving the intent for each start request, enabling the user to perform the background work.

- **Service** – Service is the base class for all services, which when extended needs to be created in a new thread.

The following section will describe how to implement the service by using either IntentService or Service.

→ **Creating a Service extending the IntentService class**

IntentService class is implemented to avoid multi-threading, that is handling of multiple requests simultaneously. It handles asynchronous requests on demand. It is started as a normal service, performs the tasks within a worker thread, and terminates when the task is performed.

The functions of IntentService class are as follows:

- Creates a default worker thread and separates all intents given to onStartCommand() from the main thread.
- Avoids multi-threading by creating a work queue where it passes one intent at a time to the onHandleIntent() method.
- Stops the service once all requests have been completed.
- Provides default onBind() method to return null.
- Provides default onStartCommand() method to send the intent to the work queue and then to the method onHandleIntent().

The section explains in detail the implementation of the IntentService class.

The steps to create a IntentService project are as follows:

1. Start Eclipse IDE.
2. Create a project named **IntentServiceExample** and leave the rest as it is.
3. Navigate to **src → com.example.intentserviceexample** folder.
4. Right-click the folder to display the context menu.
5. Select **New → Class** to create a new class as shown in figure 9.1.

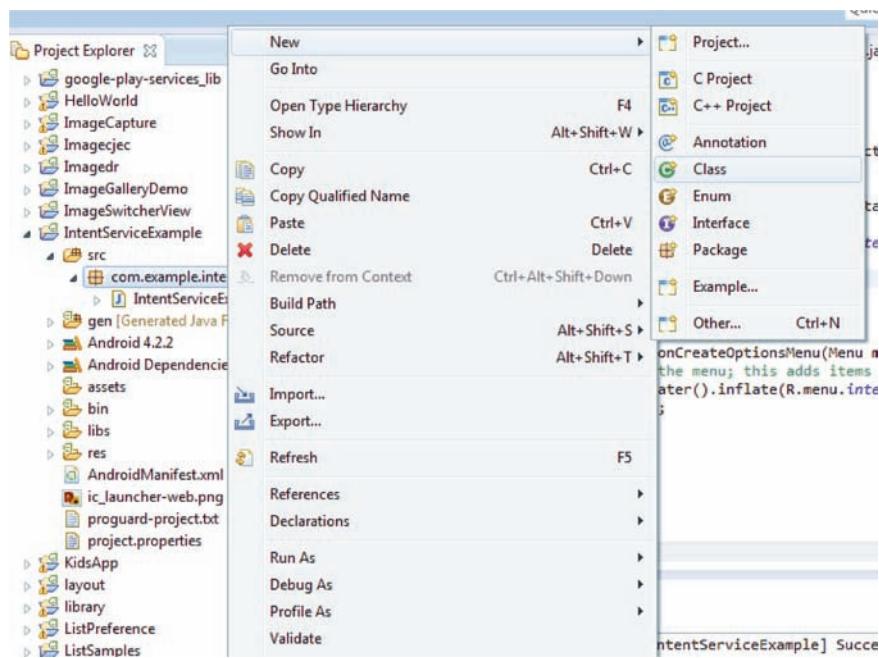


Figure 9.1: Create Class

This will display the **New Java Class** dialog box as shown in figure 9.2.

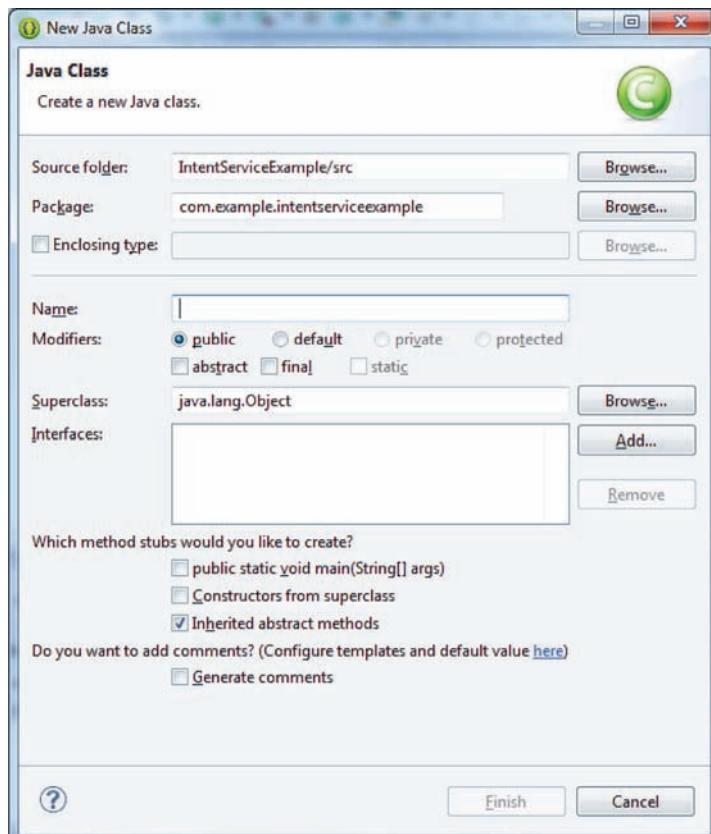


Figure 9.2: New Java Class

6. Type **MyIntentService** in the **Name** box.
7. Click **Finish**.
8. Modify the code in the newly created class as shown in code snippet 2.

Code Snippet 2:

```
package com.example.intentserviceexample;

import android.app.IntentService;
import android.content.Intent;

import android.widget.Toast;

public class MyIntentService extends IntentService {

    public MyIntentService() {
        super("MyIntentService");
        // TODO Auto-generated constructor stub
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        Toast.makeText(this, "My IntentService started",
        Toast.LENGTH_LONG)
            .show();
    }

    @Override
    public void onCreate() {
        super.onCreate();
        Toast.makeText(this, "Creating a Service ...",
        Toast.LENGTH_LONG).show();
    }
}
```

Note - The class must override `onHandleIntent()` method.

9. Navigate to **src → com.example.intentserviceexample → MainActivity.java** file.
10. Modify the code as shown in code snippet 3.

Code Snippet 3:

```

package com.example.intentserviceexample;

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

public class MainActivity extends Activity {

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

        Button start = (Button) findViewById(R.id.button1);
        start.setOnClickListener(new OnClickListener() {
            public void onClick(View V) {
                Intent intent = new Intent(MainActivity.
                        this,
                        MyIntentService.class);
                startService(intent);
            }
        });
    }

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

```

11. Navigate to **res → values → strings.xml** file to change the string appearing as a text.
12. Modify the code in the **strings.xml** file as shown in code snippet 4.

Code Snippet 4:

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

<string name="app_name">IntentServiceExample</string>
<string name="action_settings">Settings</string>
<string name="hello_world">Hello Service Example!</string>

</resources>
```

13. Navigate to **res → layout → activity_main.xml** file.
14. Modify the code in the **activity_main.xml** file as shown in code snippet 5.

Code Snippet 5:

```
<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">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />
```

```
<Button  
    android:id="@+id/button1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_below="@+id/textView1"  
    android:layout_marginTop="33dp"  
    android:text="Start Service"/>  
  
</RelativeLayout>
```

15. Navigate and open **AndroidManifest.xml** file to add the `<service>` element.
16. Modify the code in **AndroidManifest.xml** file as shown in code snippet 6.

Code Snippet 6:

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/  
    android"  
        package="com.example.intentsericeexample"  
        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.intentsericeexample.  
                MainActivity"  
            android:label="@string/app_name">  
            <intent-filter>
```

```
<actionandroid:name="android.intent.action.MAIN"/>

<categoryandroid:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
<serviceandroid:name=".MyIntentService"></service>
</application>

</manifest>
```

On execution of the application and clicking the **Start Service**, the output will be as shown in figure 9.3.

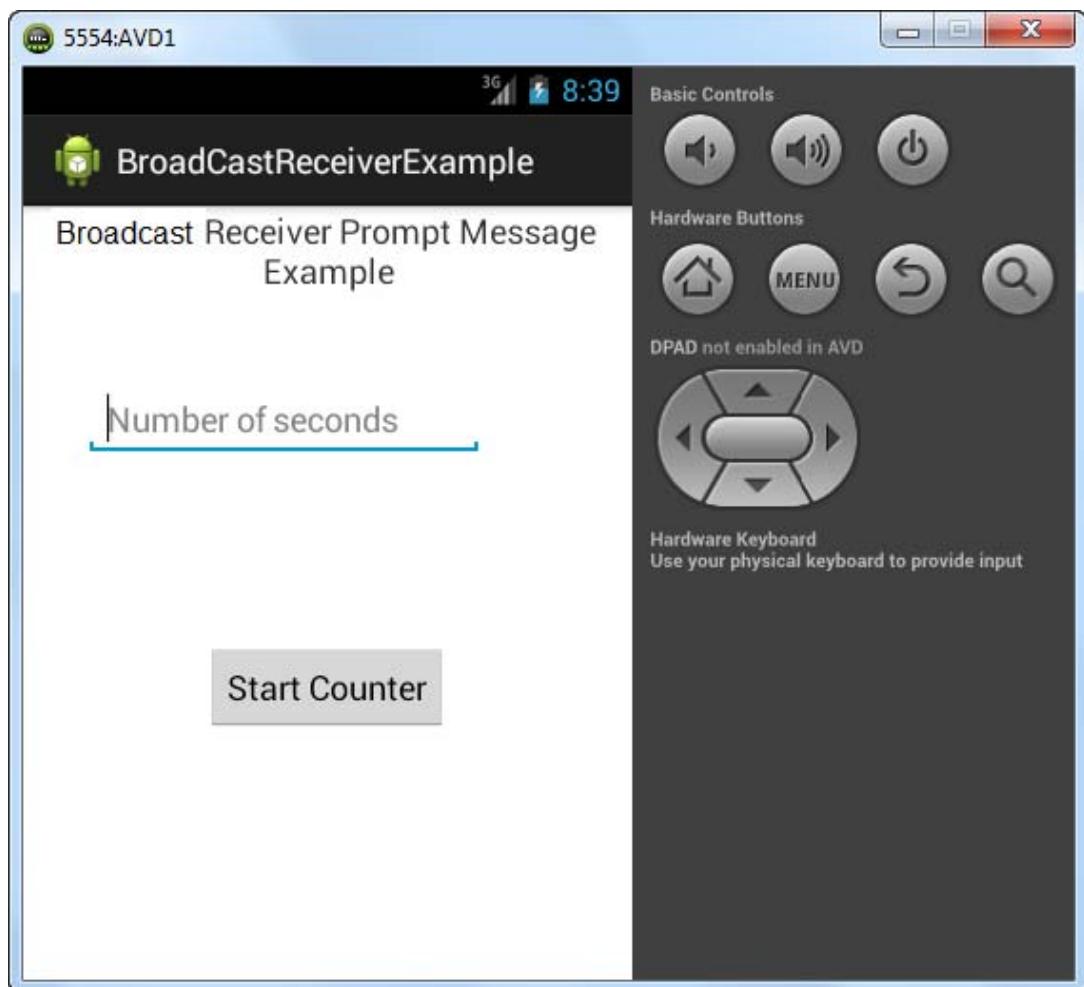


Figure 9.3: MyIntentService Output

→ Creating a Service extending the Service class

Using `IntentService` class makes the implementation of a started service easy, however, if the user requires the service to perform multi-threading, then the developer can extend from the `Service` class to handle each intent.

The section explains in detail the implementation of the `Service` class where the user will start and stop the music using service.

The steps to create a `ServiceDemoExample` project are as follows:

1. Start Eclipse IDE.
2. Create a new project named `ServiceDemoExample`.
3. Navigate to `src → com.example.servicedemoexample` package.
4. Right-click the folder to display the context menu.
5. Select **New → Class** to create a class.
6. Type `ServiceExample`.
7. Click **Finish**.
8. Modify the code in `ServiceExample` class as shown in code snippet 7.

Code Snippet 7:

```
package com.example.servicedemoexample;

import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;
import android.widget.Toast;

public class ServiceExample extends Service {

    MediaPlayermp;

    @Override
```

```

public IBinder onBind(Intent intent) {
    // TODO Auto-generated method stub
    return null;
}

@Override
public void onCreate() {
    Toast.makeText(this, "Service call oncreate()",
        event once",
        Toast.LENGTH_SHORT).show();

    mp = MediaPlayer.create(this, R.raw.flyaway);
    mp.setLooping(false); // Set looping
}

@Override
public void onDestroy() {
    Toast.makeText(this, "Service Stop",
        Toast.LENGTH_SHORT).show();

    mp.stop();
}

@Override
public void onStart(Intent intent, int startid) {
    Toast.makeText(this, "Service started",
        Toast.LENGTH_SHORT).show();
    mp.start();
}
}

```

9. Navigate to **src → com.example.servicedemoexample → MainActivity.java** file.
10. Modify the code in **MainActivity.java** file as shown in code snippet 8.

Code Snippet 8:

```
package com.example.servicedemoexample;

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

public class MainActivity extends Activity {

    Button startservice, stop_service;

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

        startservice = (Button) findViewById(R.id.button1);
        stop_service = (Button) findViewById(R.id.button2);
        startservice.setOnClickListener
            (new OnClickListener() {

                @Override
                public void onClick(View v) {
                    // TODO Auto-generated method stub
                    Toast.makeText(getApplicationContext(),
                        "Service call to start",
                        Toast.LENGTH_SHORT).show();
                }
            });
    }
}
```

```
        startService(new Intent(MainActivity.  
this, ServiceExample.class));  
  
    }  
});  
  
stop_service.setOnClickListener(newOnClickListener() {  
  
    @Override  
    public void onClick(View v) {  
        // TODO Auto-generated method stub  
        Toast.makeText(MainActivity.this,  
        "Service call to stop",  
        Toast.LENGTH_SHORT).show();  
        stopService(new Intent(MainActivity.  
this, ServiceExample.class));  
    }  
});  
  
}  
  
@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;  
}  
}
```

11. Create a new folder named **raw** inside the **/res** directory by right-clicking the **/res** folder to display the context menu.
12. Select **New → Folder** and type the name as **raw**.

13. Copy any song inside the folder. For example, it is **flysong.mp3** as shown in figure 9.4.

Note - The song name should be in lowercase and without space.

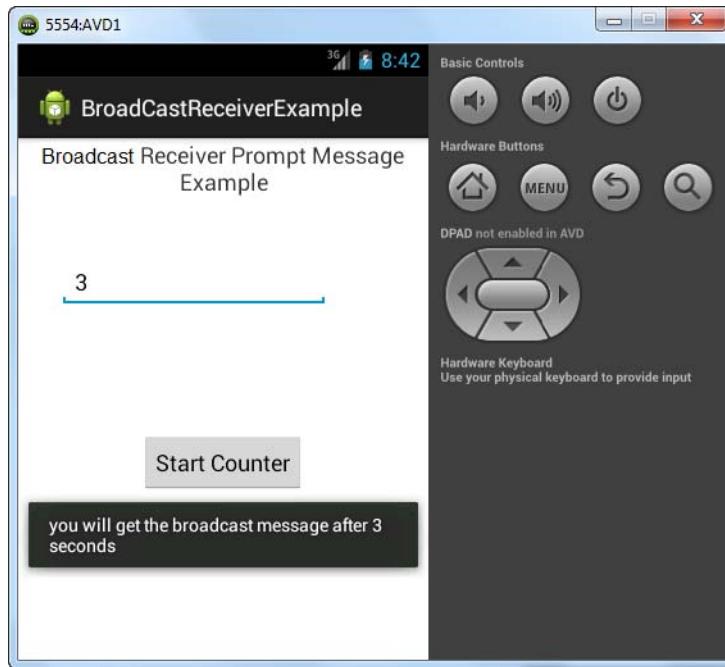


Figure 9.4: Song

14. Click **res → layout → activity_main.xml** file to display the file in Graphical Layout mode and add two buttons by dragging the button as shown in figure 9.5.

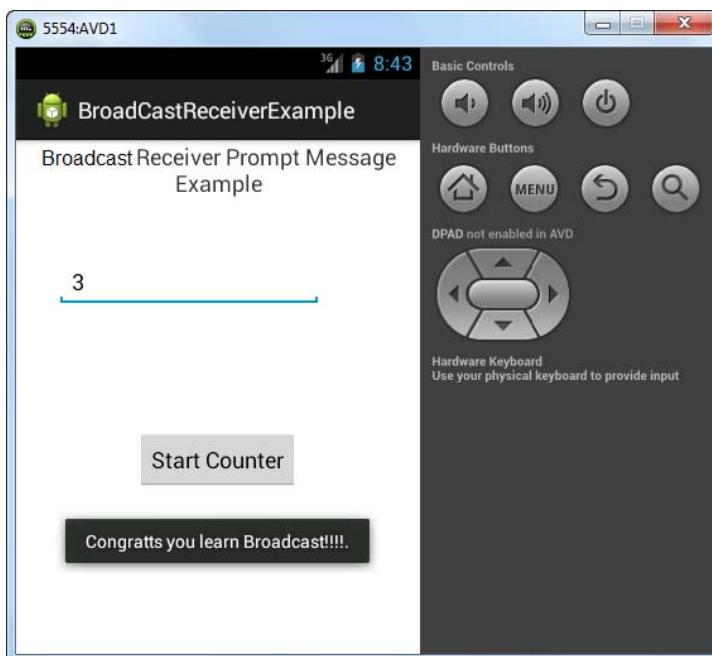


Figure 9.5: Create a Button

15. Click **activity_main.xml** tab to open the file.
16. Modify the code in **activity_main.xml** file as shown in code snippet 9.

Code Snippet 9:

```
<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">  
  
    <TextView  
        android:id="@+id/textView1"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="@string/hello_world" />  
  
    <Button  
        android:id="@+id/button2"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_alignRight="@+id/button1"  
        android:layout_centerVertical="true"  
        android:text="Stop Music by Service" />  
  
    <Button  
        android:id="@+id/button1"  
        android:layout_width="wrap_content"
```

```

    android:layout_height="wrap_content"
    android:layout_below="@+id/textView1"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="52dp"
    android:text="Start Music by Service" />

</RelativeLayout>

```

17. Navigate to **res → values → strings.xml** file.
18. Modify the code in **strings.xml** file as shown in code snippet 10.

Code Snippet 10:

```

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

    <string name="app_name">ServiceDemoExample</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello using Service class!
    </string>

</resources>

```

19. Open **AndroidManifest.xml** file and add the **<service>** element as shown in code snippet 11.

Code Snippet 11:

```

    ...
<service android:name=".ServiceExample"></service>
    ...

```

On execution of the program the output will be as shown in figure 9.6.

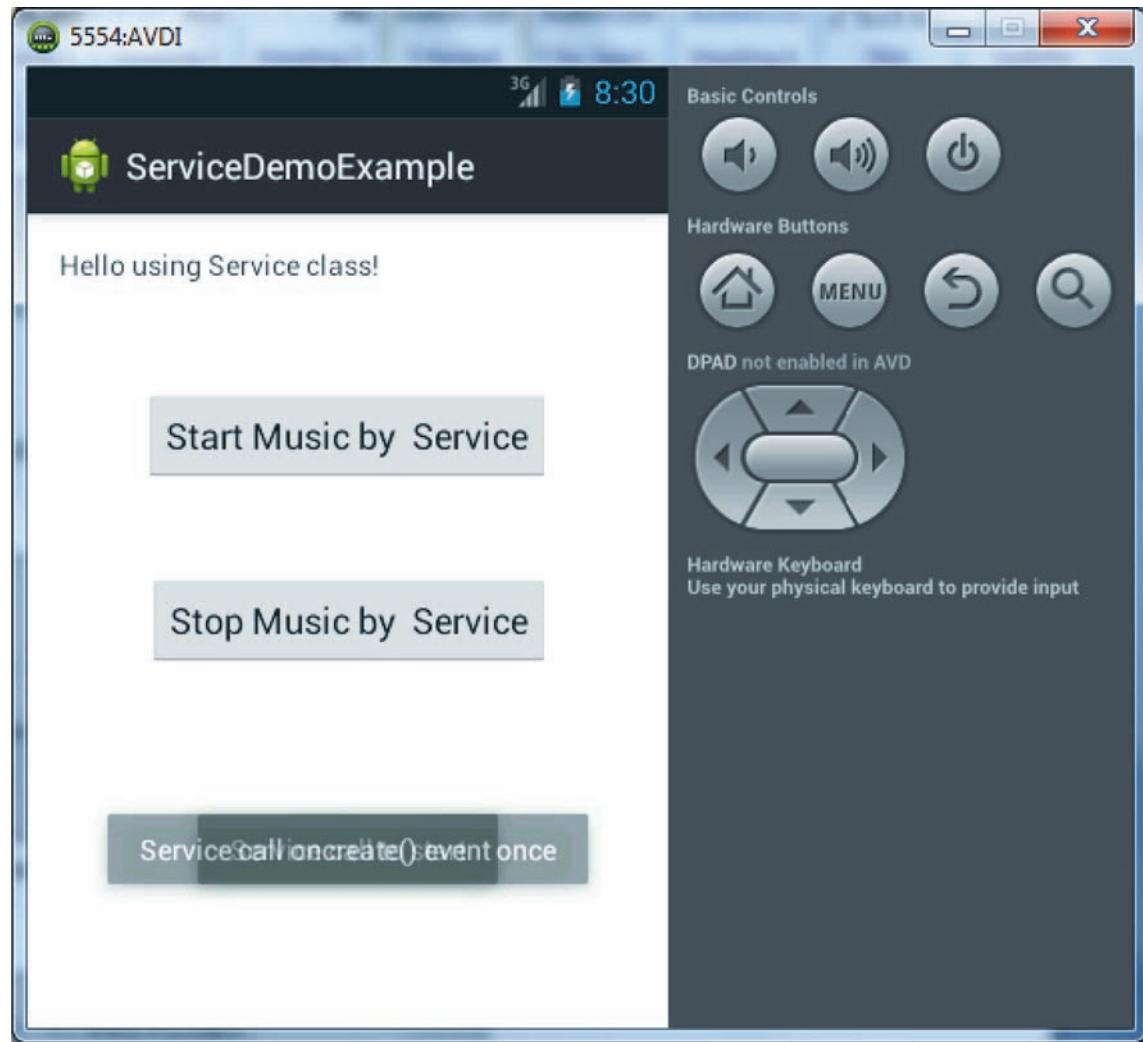


Figure 9.6: Start Music by Service

Figure 9.7 displays the output when the **Stop Music by Service** is clicked.

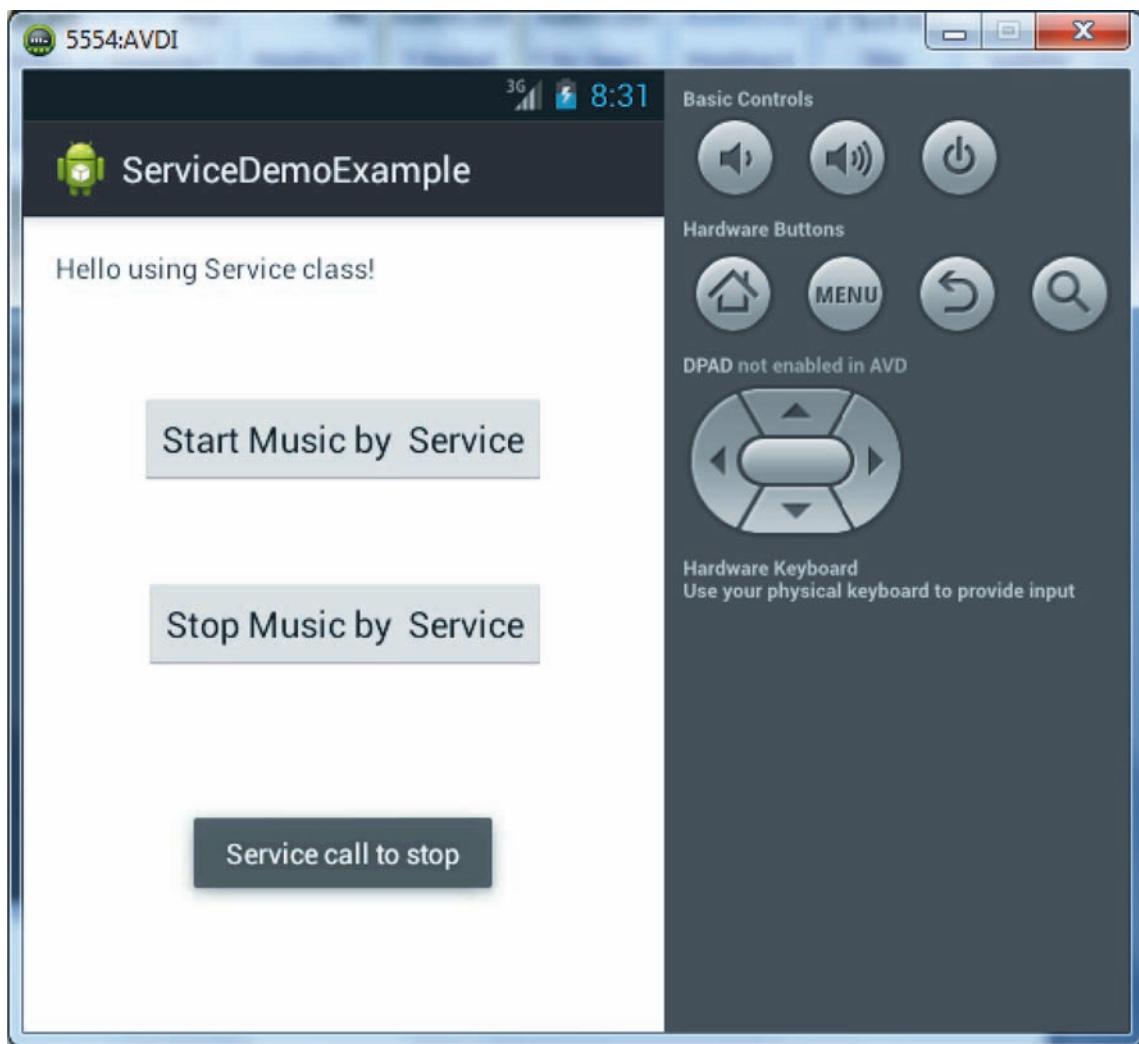


Figure 9.7: Stop Music by Service

9.2.4 Starting a Service

A service can be started from an activity or a component by passing an Intent to `startService()` method. The `onStartCommand()` method of the Service class is then invoked by the Android system and the Intent is passed.

Note - `onStartCommand()` method should never be invoked directly.

For example, in code snippet 12, an activity can start the example service using an explicit intent with `startService()` method.

Code Snippet 12:

```
startService(new Intent(MainActivity.this, ServiceExample.class));
...
```

The `startService()` method returns instantly and the Android system calls the service's `onStartCommand()` method. In case, the service is not running already, the system will first invoke `onCreate()` method followed by the invocation of the `onStartCommand()` method.

When multiple requests are sent to start a service, it results in multiple calls to the service's `onStartCommand()` method but it requires the invocation of the method `stopService()` method only once to stop the service.

9.2.5 Stopping a Service

A started service must stop either by invoking itself the `stopSelf()` method or by invoking the `stopService()` method by another component. The moment it is requested to stop using either ways, the system destroys the service.

However, if the service is handling multiple requests by using `onStartCommand()`, then it should not be stopped as a new start request might have been received. To avoid this issue, `stopSelf(int)` method can be used to ensure that the request to stop the service is based on the most recent start request.

Code snippet 13 demonstrates how to stop a started service using Intents.

Code Snippet 13:

```
...
stopService(new Intent(MainActivity.this, ServiceExample.class));
...
```

Note - To avoid wastage of system resources and consumption of battery power, it is important that the application stops its services when the work is complete. If required, other components can stop the service by invoking the `stopService()` method.

9.2.6 Creating a Bound Service

When components of an application bind to a service by calling `bindService()`, it is called a bound service. Binding to a service helps in creating long-standing connection. Developer can bind the service with Activity, Service, and Content Provider. The `bindService()` method is invoked when the developer interact with activity, service, or to expose some of the application functionality to other process using Android Interface Definition Language (AIDL).

To used a bound service, the `onBind()` callback method must be implemented, it returns an `IBinder` that specifies the interface for communication with the service. The service exists only till it serves the application component that it is bound to, however if there are no components bound to the service, the system destroys it.

While creating a bound service, the interface that specifies how a client can communicate with the service must be defined. This interface should be an implementation of `IBinder` and must extend the `Service` class from the `onBind()` callback method.

Several clients can bind to the service at once. When the interaction is complete, the client invokes the method `unbindService()` to unbind. The moment there are no clients bound to the service, the system destroys the service.

When using a bound service, developer must use an `IBinder` which provide the programming layer in which client can used to interact with the service. There are three ways where developer can define layer and they are as follows:

- By Using a Messenger.
- By Extending a Binder Class.
- By Using a AIDL.

In the following example, the developer can start, stop, bind, and unbind the service with button click.

The steps to create a `BoundServiceExample` project are as follows:

1. Start Eclipse IDE.
2. Create a project named `BoundServiceExample`.
3. Navigate to `src → com.example.boundserviceexample` package.
4. Right-click the folder to display the context menu.
5. Select **New → Class** to create a class.
6. Type `MyBoundService`.
7. Click **Finish**.
8. Modify the code in `MyBoundService` class as shown in code snippet 14.

Code Snippet 14:

```
package com.example.boundserviceexample;

import java.util.ArrayList;
import java.util.Timer;
import java.util.TimerTask;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.util.Log;

public class MyBoundService extends Service {

    private Timer timer = new Timer();
    private int counter = 0, incrementby = 100;
    private static boolean isRunning = false;

    ArrayList<Messenger>mClients =
    newArrayList<Messenger>();

    int mValue = 0; // Holds last value set by a client.

    static final int MSG_REGISTER_CLIENT = 1;
    static final int MSG_UNREGISTER_CLIENT = 2;
    static final int MSG_SET_INT_VALUE = 3;
    static final int MSG_SET_STRING_VALUE = 4;
```

```
final Messenger mMessenger = new Messenger  
(new IncomingHandler()); // Target  
  
@Override  
public IBinder onBind(Intent intent) {  
    return mMessenger.getBinder();  
}  
  
class IncomingHandler extends Handler { // Handler of incoming  
messages from  
// clients.  
  
    @Override  
    public void handleMessage(Message msg) {  
        switch (msg.what) {  
            case MSG_REGISTER_CLIENT:  
                mClients.add(msg.replyTo);  
                break;  
            case MSG_UNREGISTER_CLIENT:  
                mClients.remove(msg.replyTo);  
                break;  
            case MSG_SET_INT_VALUE:  
                incrementBy = msg.arg1;  
                break;  
            default:  
                super.handleMessage(msg);  
        }  
    }  
}  
  
private void sendMessageToUI(int intValueToSend) {  
    for (int i = mClients.size() - 1; i >= 0; i--) {  
        try {
```

```
// Send data as an Integer  
mClients.get(i).send(  
    Message.obtain(null, MSG_SET_INT_  
        VALUE, intvaluetosend,  
        0));  
  
// Send data as a String  
Bundle b = new Bundle();  
b.putString("str1", "ab" + intvaluetosend +  
    "cd");  
Message msg = Message.obtain(null, MSG_SET_  
    STRING_VALUE);  
msg.setData(b);  
mClients.get(i).send(msg);  
  
} catch (RemoteException e) {  
    // The client is dead. Remove it from the list;  
    // we are going  
    // through the list from back to front so this is  
    // safe to do  
    // inside the loop.  
    mClients.remove(i);  
}  
}  
}  
}  
  
@Override  
public void onCreate() {  
    super.onCreate();  
  
    timer.scheduleAtFixedRate(new TimerTask() {  
        public void run() {  
            onTimerTick();  
        }  
    });  
}
```

```
    } , 0 , 100L);

    isRunning = true;

}

@Override

public int onStartCommand(Intent intent, int flags,
int startId) {

    Log.i("MyService", "Received start id " + startId + " : " +
intent);

    return START_STICKY; // run until explicitly stopped.

}

public static boolean isRunning() {

    return isRunning;

}

private void onTimerTick() {

    try {

        counter += incrementBy;

        sendMessageToUI(counter);

    } catch (Throwable t) { // you should always ultimately
// catch all

        // exceptions in timer tasks.

    }
}

@Override

public void onDestroy() {
    super.onDestroy();
    if (timer != null) {
```

```

        timer.cancel();
    }

    counter = 0;

    isRunning = false;
}

}

```

9. Navigate to **src → com.example.boundserviceexample → MainActivity.java** file.
10. Modify the code in **MainActivity.java** file as shown in code snippet 15.

Code Snippet 15:

```

package com.example.boundserviceexample;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {

```

```

Button Servicestart, Servicestop, btnBindservice,
btnUnbindservice,
        btnaddby100;

TextView textStatus, textIntValue, textStrValue;
Messenger mservicecall=null;
boolean mIsBound;
final Messenger mMessenger = new Messenger
(new IncomingHandler());

class IncomingHandler extends Handler {
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
            case MyBoundService.MSG_SET_INT_VALUE:
                textIntValue.setText("Status : " + msg.arg1);
                break;
            case MyBoundService.MSG_SET_STRING_VALUE:
                String str1 = msg.getData().getString("str1");
                textStrValue.setVisibility(View.GONE);
                textStrValue.setText("Str Message: " + str1);
                break;
            default:
                super.handleMessage(msg);
        }
    }
}

private ServiceConnection mConnection =
new ServiceConnection() {
    public void onServiceConnected(ComponentName className,
        IBinder service) {
        mservicecall = new Messenger(service);
    }
}

```

```

        textStatus.setText("Attached.");
    try {
        Message msg = Message.obtain(null,
            MyBoundService.MSG_REGISTER_CLIENT);
        msg.replyTo = mMessenger;
        mservicecall.send(msg);
    } catch (RemoteException e) {
        // In this case the service has crashed before we
        // could even do
        // anything with it
    }
}

public void onServiceDisconnected(ComponentName className) {
    // This is called when the connection with the
    // service has been
    // unexpectedly disconnected - process crashed.
    mservicecall = null;
    textStatus.setText("Disconnected.");
}
}

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

    Servicestart = (Button) findViewById(R.id.Servicestart);
    Servicestop = (Button) findViewById(R.id.Servicestop);
    btnBindservice = (Button) findViewById(R.id.btnBindservice);
}

```

```
btnUnbindservice = (Button) findViewById (R.id.  
btnUnbindservice);  
  
textStatus = (TextView) findViewById(R.id.statusText);  
  
textIntValue = (TextView) findViewById  
(R.id.textIntValue);  
  
textStrValue = (TextView) findViewById(R.id.textStrValue);  
btndaddby100 = (Button) findViewById(R.id.btndaddby100);  
  
Servicestart.setOnClickListener(newOnClickListener()  
{  
  
    @Override  
  
    public void onClick(View v) {  
  
        // TODO Auto-generated method stub  
  
        startService(new Intent(MainActivity.this,  
MyBoundService.class));  
  
    }  
  
});  
  
Servicestop.setOnClickListener(newOnClickListener()  
{  
  
    @Override  
  
    public void onClick(View v) {  
  
        doUnbindService();  
  
        stopService(new Intent(MainActivity.this,  
MyBoundService.class));  
  
    }  
  
});  
  
btnBindservice.setOnClickListener  
(newOnClickListener() {  
  
    @Override  
  
    public void onClick(View v) {  
  
        doBindService();  
    }  
});
```

```
        }

    } );

btnUnbindservice.setOnClickListerner
(newOnClickListerner() {

    @Override
    publicvoidonClick(Viewv) {
        doUnbindService();
    }
} );

btnaddby100.setOnClickListerner(newOnClickListerner()
{

    @Override
    publicvoidonClick(Viewv) {
        sendMessageToService(100);
    }
} );

restoreMe(savedInstanceState);

CheckIfServiceIsRunning();

}

@Override
protectedvoidonSaveInstanceState(BundleoutState) {
    super.onSaveInstanceState(outState);
}
```

```

        outState.putString("textStatus", "Method called:"
            + textStatus.getText().toString()));

        outState.putString("textIntValue", textView.
            getText().toString());

        outState.putString("textStrValue", textView.
            getText().toString());
    }

private void restoreMe(Bundle state) {
    if (state != null) {
        textStatus.setText(state.getString("textStatus"));
        textView.setText(state.
            getString("textIntValue"));
        textView.setText(state.
            getString("textStrValue"));
    }
}

private void checkIfServiceIsRunning() {
    // If the service is running when the activity starts, we
    // want to
    // automatically bind to it.
    if (MyBoundService.isRunning()) {
        doBindService();
    }
}

private void sendMessageToService(int intValueToSend) {
    if (mIsBound) {
        if (mService != null) {
            try {
                Message msg = Message
                    .obtain(null, MyBoundService.
                        MSG_SET_INT_VALUE,

```

```

        intvaluetosend, 0);

        msg.replyTo = mMessenger;

        mservicecall.send(msg);

    } catch (RemoteException e) {
    }

}

}

void doBindService() {
    bindService(new Intent(this, MyBoundService.class),
    mConnection,
    Context.BIND_AUTO_CREATE);
    mIsBound = true;
    textStatus.setText("Binding.");
}

void doUnbindService() {
    if (mIsBound) {
        // If we have received the service, and hence
        // registered with it,
        // then now is the time to unregister.
        if (mservicecall != null) {
            try {
                Message msg = Message.obtain(null,
                    MyBoundService.MSG_UNREGISTER_
                    CLIENT);
                msg.replyTo = mMessenger;
                mservicecall.send(msg);
            } catch (RemoteException e) {
                // There is nothing special we need to do if
                // the service has
                // crashed.
            }
        }
    }
}

```

```
        }

    }

    // Detach our existing connection.

    unbindService(mConnection);

    mIsBound = false;

    textStatus.setText("Unbinding..");

}

}

@Override

protected void onDestroy() {

    super.onDestroy();

    try {

        doUnbindService();

    } catch (Throwable e) {

    }

}

@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;

}

}
```

11. Click **res → layout → activity_main.xml** file to display the file.
12. Modify the code as show in code snippet 16.

Code Snippet 16:

```
<?xmlversion="1.0"encoding="utf-8"?>  
  
<LinearLayoutxmlns:android="http://schemas.android.com/apk/  
res/android"  
  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:orientation="vertical">  
  
    <RelativeLayout  
  
        android:layout_width="fill_parent"  
        android:layout_height="wrap_content">  
  
        <TextView  
  
            android:id="@+id/comment"  
            android:layout_width="fill_parent"  
            android:layout_height="wrap_content"  
            android:text="Demo of Bound Service Example"  
            android:textSize="20sp"  
            android:textStyle="bold"/>  
        </RelativeLayout>  
  
        <RelativeLayout  
  
            android:id="@+id/RelativeLayout01"  
            android:layout_width="fill_parent"  
            android:layout_height="wrap_content">  
  
            <Button  
  
                android:id="@+id/Servicestart"  
                android:layout_width="wrap_content"  
                android:layout_height="wrap_content"  
                android:text="Run Service">  
            </Button>
```

```
<Button  
    android:id="@+id/Servicestop"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignParentRight="true"  
    android:text="Stop Service">  
</Button>  
</RelativeLayout>  
  
<TextView  
    android:id="@+id/textStatus"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:gravity="center_vertical"  
    android:text="CallBack Used "  
    android:textSize="24sp" />  
  
<TextView  
    android:id="@+id/textIntValue"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="Output"  
    android:textSize="24sp" />  
  
<TextView  
    android:id="@+id/textStrValue"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="String Value Goes Here"  
    android:textSize="24sp"
```

```
        android:visibility="gone" />

    <RelativeLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">

        <TextView
            android:id="@+id/showvalue"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="See the onBound( ) effect"
            android:textSize="24sp"
            android:textStyle="bold" />
        </RelativeLayout>

        <RelativeLayout
            android:id="@+id/RelativeLayout03"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content">

            <Button
                android:id="@+id/btnaddby100"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Add value by 100">
            </Button>
        </RelativeLayout>

        <RelativeLayout
            android:id="@+id/RelativeLayout02"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content">
```

```
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">

    <Button
        android:id="@+id/btnBindservice"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bind to Service">
    </Button>

    <Button
        android:id="@+id/btnUnbindservice"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:text="Unbind from Service">
    </Button>
</RelativeLayout>

</LinearLayout>
```

13. Navigate and open **AndroidManifest.xml** file.
14. Modify the code to add the `<service>` element as shown in code snippet 17.

Code Snippet 17:

```
<?xmlversion="1.0"encoding="utf-8"?>

<manifestxmlns:android="http://schemas.android.com/apk/res/
android"

    package="com.example.boundserviceexample"
    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.boundserviceexample.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>  
    <service android:name=".MyBoundService"></service>  
    </application>  
  
</manifest>
```

Once the application is executed the output will be as shown in figure 9.8.

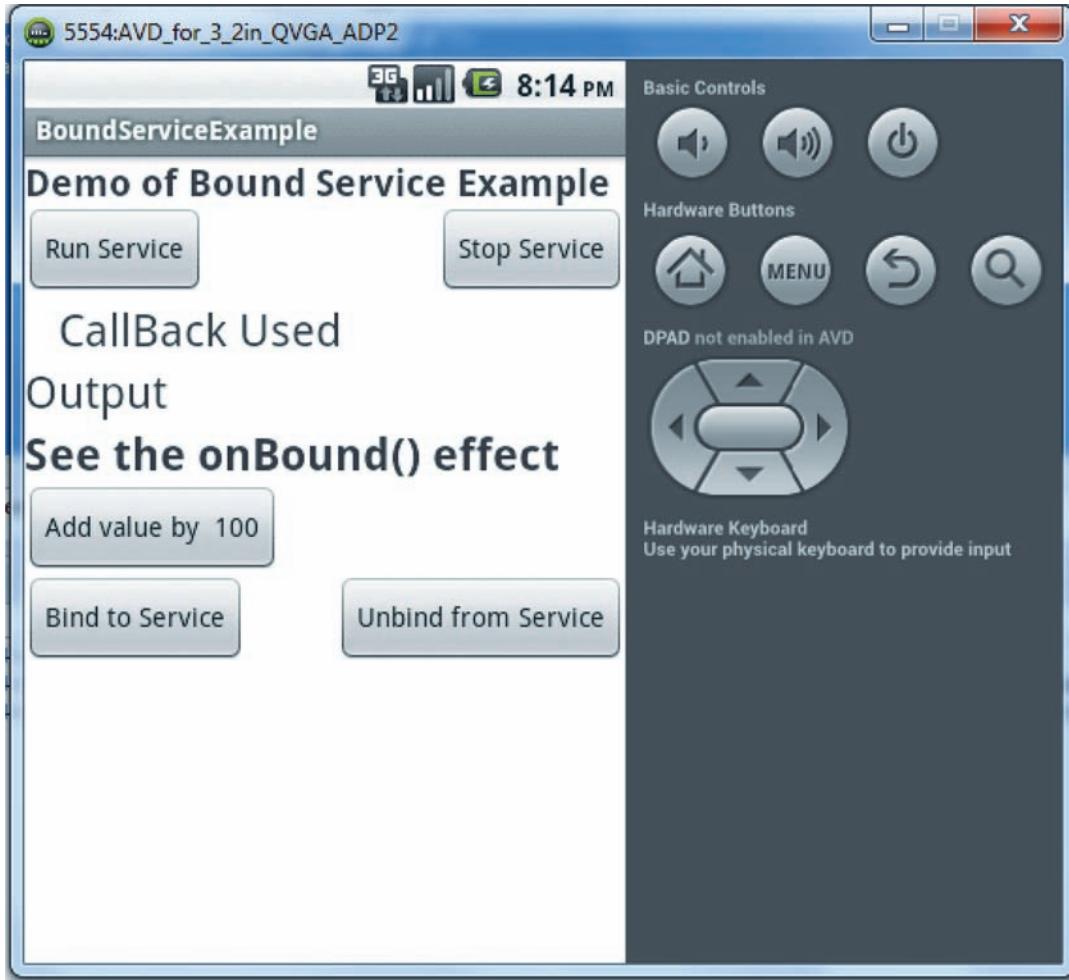


Figure 9.8: Bound Service Example

Click **Run Service**, then click **Add value by 100**, and finally click **Bind to Service** to get the output as shown in figure 9.9.

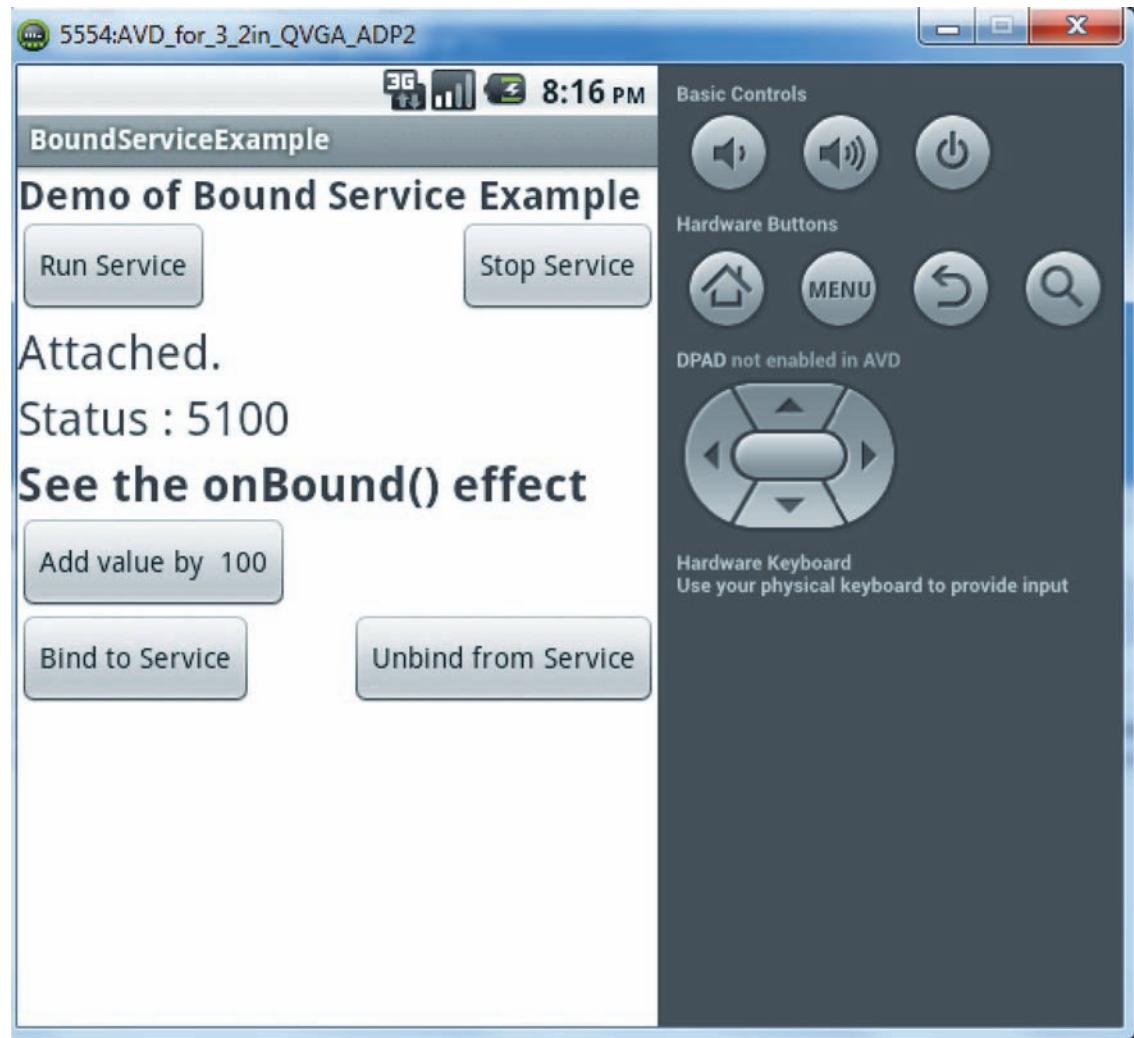


Figure 9.9: Service Bounded

9.3 Broadcast Receiver and Intent filters

Broadcast Receivers is also a type of component and plays an important part in building an Android application. It plays a specific role in the application development and is basically responsible for responding to system-wide broadcast announcements. On the other hand, Intents are responsible for binding the different individual component to each other at runtime. The section explains both of them in details.

9.3.1 Overview of Broadcast Receiver

A broadcast receiver is an android component which allows the application to register for system or application events. The operating system and other applications broadcast

announcements about the things that are happening, for example, a broadcast may be sent about the screen turning off, the change in battery status, Wi-Fi turning on, or a picture that was captured. Broadcast receivers do not display a user interface; they might create a status bar notification to inform the user about a broadcast event. Each broadcast is delivered as Intent. In other words, the messages are sometimes called events or intents.

Applications can generate broadcasts to send events not knowing who will receive them. Receivers that require the information, subscribe to specific messages using filters. If the message matches a filter, the receiver is activated and notified of the message.

Broadcast receivers are great for performing small amounts of work, in response to an external stimulus, such as posting an announcement to the user after being sent a new GPS location report. They are active for a fixed amount of time; fairly strong guarantees can be made about not killing their process while running. However, they are not suitable for anything of indeterminate length, such as networking.

9.3.2 Implementing Broadcast Receivers

Following are the two steps to implement a broadcast receiver:

1. You have to create a subclass of Android's Broadcast Receiver.
2. You have to implement the `onReceive()` method.

Note - Broadcast receivers are registered in **AndroidManifest.xml** file using the `<receiver>` element or `context.registerReceiver()` method in Java class.

→ Implementing the `onReceive()` method

Android calls the `onReceive()` method on all registered broadcast receivers, when a matching broadcast intent is identified. For example, a user wants to be notified whenever the battery level is low. In this case, the receiver can be registered to the event `Intent.ACTION_BATTERY_LOW`. So, the moment battery level falls below a defined threshold, the `onReceive()` method will be called.

`onReceive()` method accepts two arguments:

- Context – This object is used to access additional information or to start services or activities.
- Intent – The Intent object is the action you used to register your receiver. This object contains additional information that can be used in the implementation of `onReceive()` method.

The Intent object passed to the `onReceive()` method often contains additional information that can be used to determine more about the event. On the other hand, the Context object is required if the user wants to start an Activity or a Service.

The section explains **BroadCastReceiverExample** in detail where user will get the message from Broadcast Receiver whatever the user types in the text box.

The steps to create a **BroadCastReceiverExample** project are as follows:

1. Start Eclipse IDE.
2. Create a project named **BroadCastReceiverExample**.
3. Navigate to **src → com.example.broadcastreceiverexample** package.
4. Right-click the folder to display the context menu.
5. Select **New → Class** to create a class.
6. Type **MyBroadcast**.
7. Click **Finish**.
8. Modify the code in **MyBroadcast** class as shown in code snippet 18.

Code Snippet 18:

```
package com.example.broadcastreceiverexample;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Vibrator;
import android.widget.Toast;

public class MyBroadcast extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "Congrats you learnt sending Broadcast!!!!.", Toast.LENGTH_LONG).show();
    }
}
```

9. Navigate to **src → com.example.broadcastreceiverexample → MainActivity.java** file.
10. Modify the code in **MainActivity.java** file as shown in code snippet 19.

Code Snippet 19:

```
package com.example.broadcastreceivexample;

import android.os.Bundle;
import android.app.Activity;
import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {

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

    public void startAlert(View view) {
        EditText text = (EditText) findViewById(R.id.time);
        int i = Integer.parseInt(text.getText().
        toString());
        Intent intent = new Intent(this, MyBroadcast.
        class);
        PendingIntent pendingIntent = PendingIntent.
        getBroadcast(
            this.getApplicationContext(), 234324243, intent,
            0);
        AlarmManager alarmManager = (AlarmManager)
        getSystemService(ALARM_SERVICE);
        alarmManager.set(AlarmManager.RTC_WAKEUP,
        System.currentTimeMillis())
    }
}
```

```

        + ( i * 1000 ) , pendingIntent ) ;

        Toast.makeText( this , "you will get the broadcast
message after " + i + " seconds" ,
        Toast.LENGTH_LONG ) . show ( ) ;
    }

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

}

```

11. Navigate and open **AndroidManifest.xml** file.
12. Modify the code to add the <receiver> element as shown in code snippet 20.

Code Snippet 20:

```

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

    package="com.example.broadcastreceivereexample"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

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

```

```

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.example.broadcastreceiverexample.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>

    <receiver
        android:name=".MyBroadcast"
        android:enabled="true" >
    </receiver>
</application>

</manifest>

```

13. Navigate to **res → layout → activity_main.xml** file.
14. Modify the code as given in code snippet 21.

Code Snippet 21:

```

?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"

```

```
        android:orientation="vertical" >

        <EditText
            android:id="@+id/time"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentTop="true"
            android:layout_marginLeft="32dp"
            android:layout_marginTop="92dp"
            android:ems="10"
            android:hint="Number of seconds"
            android:inputType="numberDecimal" >

            <requestFocus />
        </EditText>

        <Button
            android:id="@+id/ok"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@+id/time"
            android:layout_centerHorizontal="true"
            android:layout_marginTop="97dp"
            android:onClick="startAlert"
            android:text="Start Counter" />

        <TextView
            android:id="@+id/abcv"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:text="Broadcast Receiver Prompt Message Example"
            android:textSize="18sp" />

    </RelativeLayout>
```

Once you execute the application, the output will be as shown in figure 9.10.

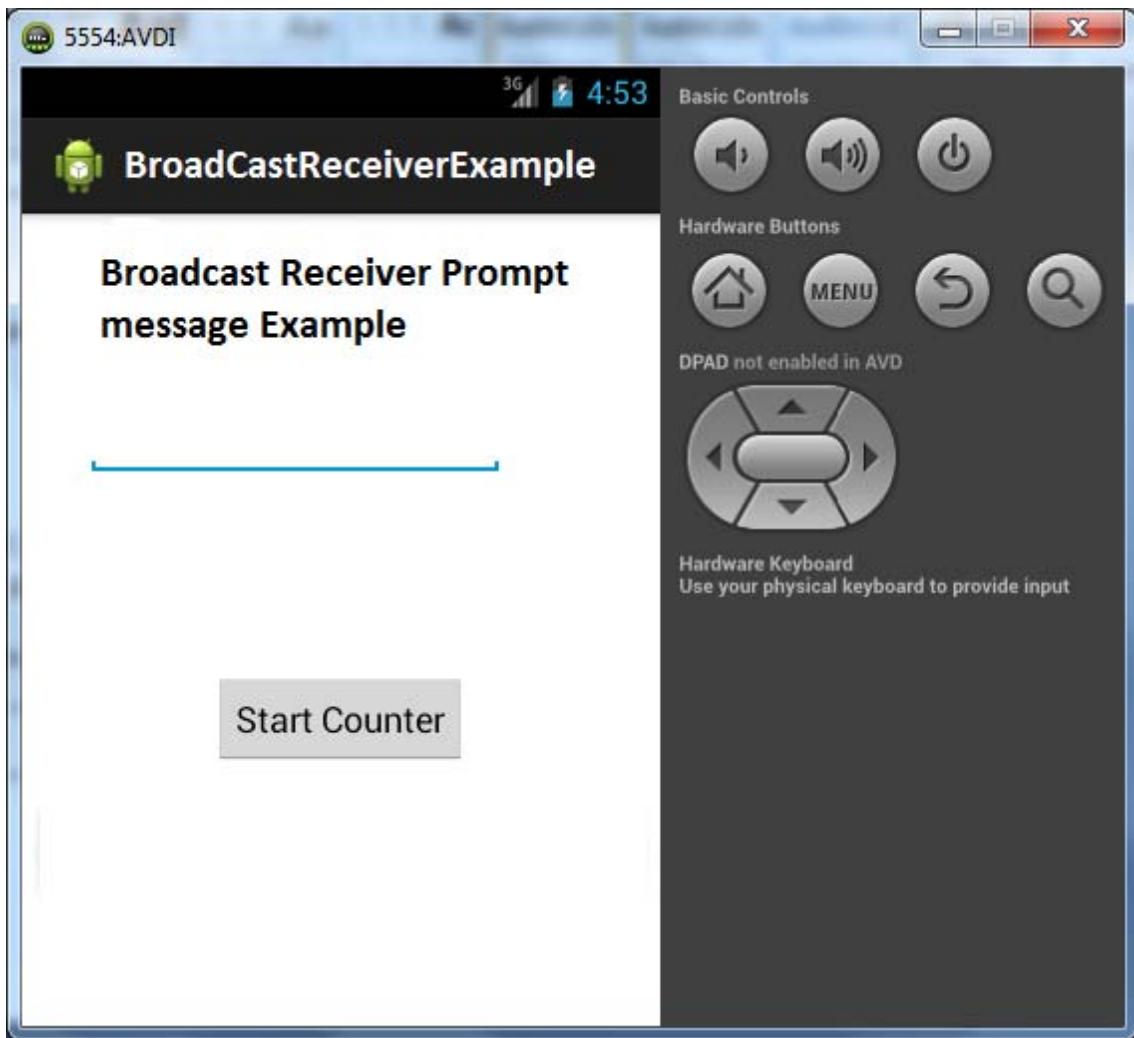


Figure 9.10: Broadcast Message Example

Enter **3** in the Edit box and the output will be as shown in figure 9.11.

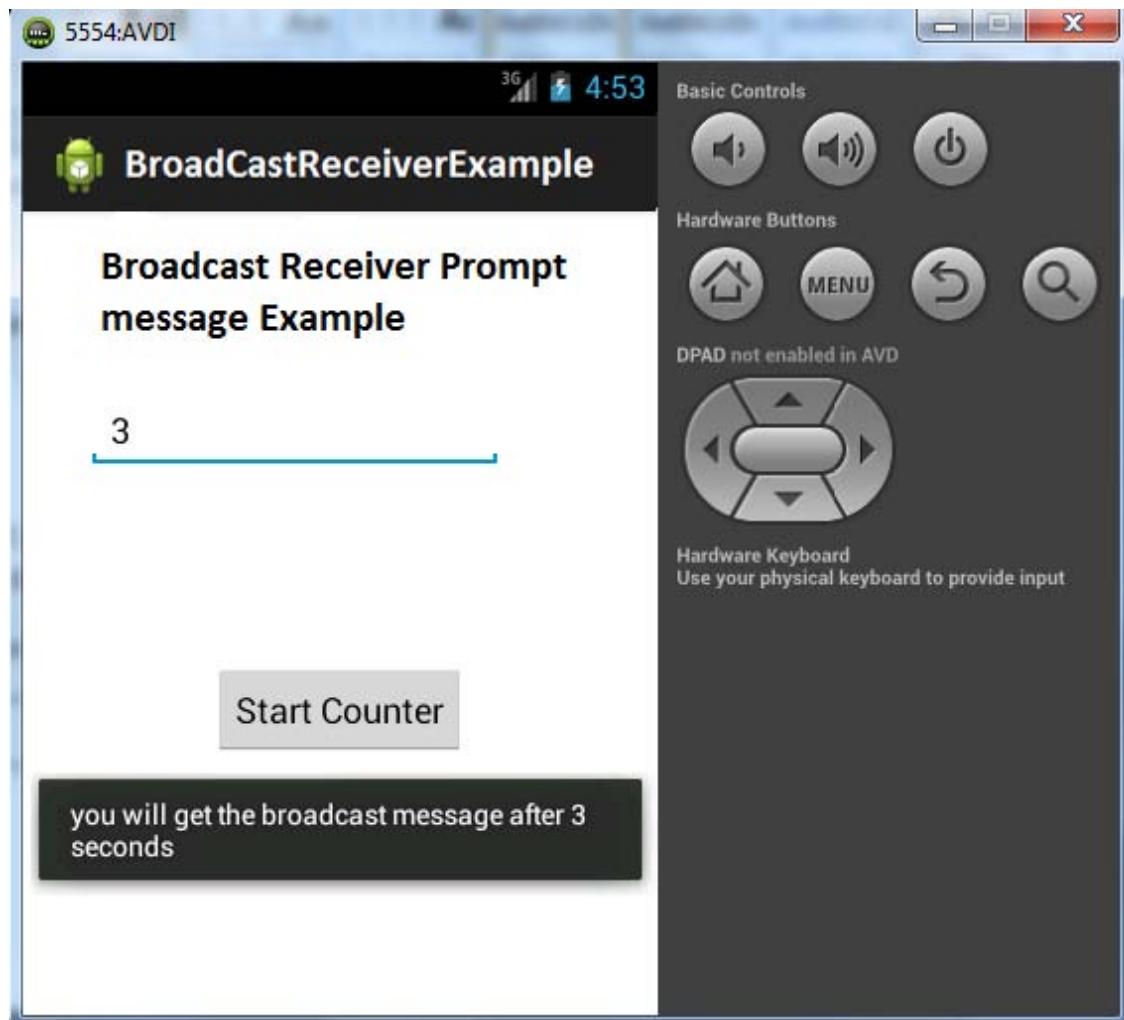


Figure 9.11: Broadcast Pending Request

After 3 seconds have passed the message will be displayed as shown in figure 9.12.

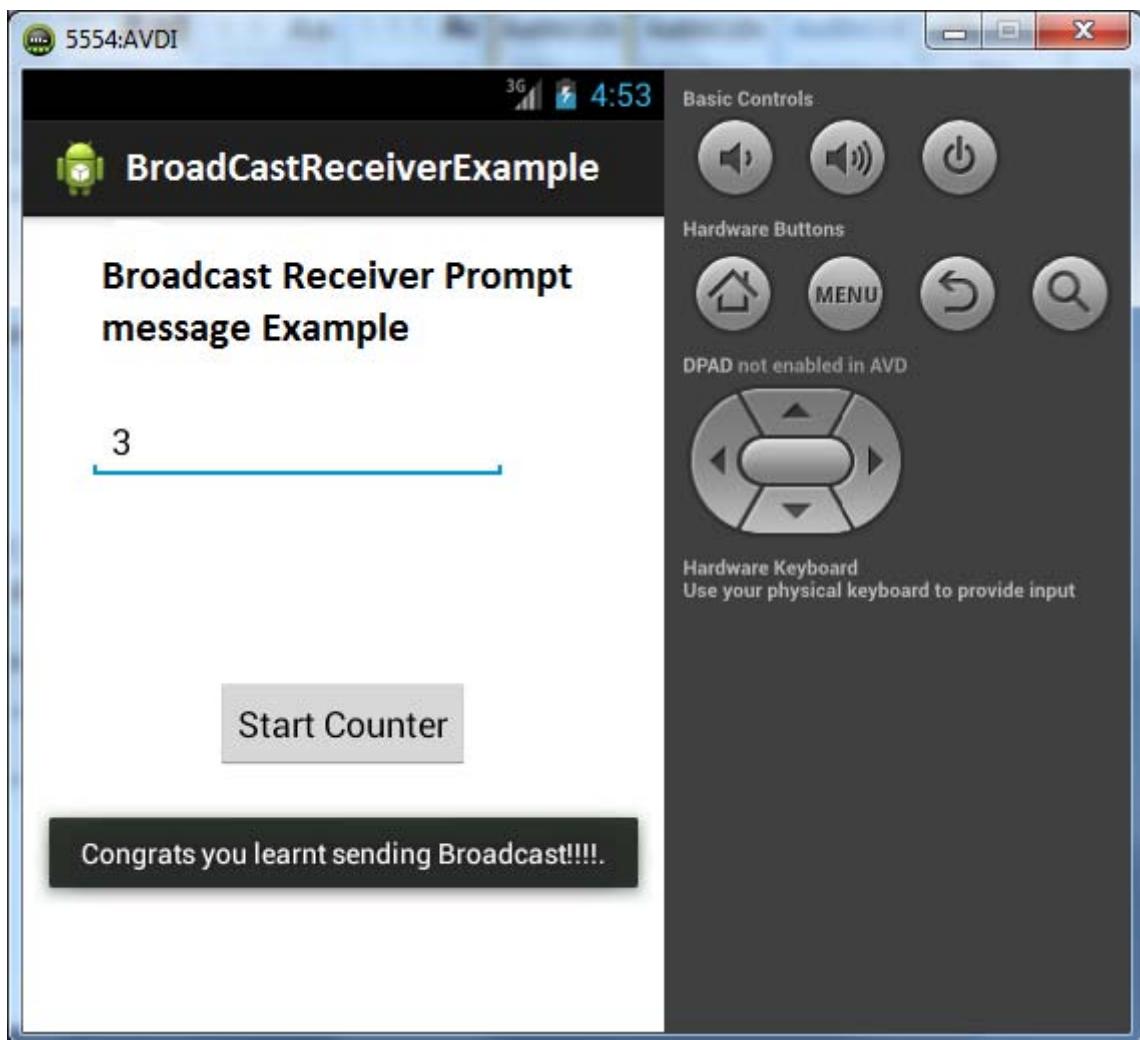


Figure 9.12: Broadcast Message

9.3.3 System Broadcast

Several system events are defined as final static fields in the Intent class. In the Application Program Interface (API), there are many classes that have specific broadcast events.

Table 9.1 lists some of the available events.

Event	Usage
Intent.ACTION_BATTERY_LOW	The battery level has dropped below a threshold
Intent.ACTION_BATTERY_OKAY	The battery level has increased again
Intent.ACTION_BOOT_COMPLETED	Android is up and running
Intent.ACTION_DEVICE_STORAGE_LOW	Storage space on the device is becoming less
Intent.ACTION_DEVICE_STORAGE_OK	The storage situation has improved again
Intent.ACTION_HEADSET_PLUG	A headset was plugged in or a previously plugged headset was removed
Intent.ACTION_LOCALE_CHANGED	The language of the device has been changed by the user
Intent.ACTION_MY_PACKAGE_REPLACED	The application has been updated
Intent.ACTION_PACKAGE_ADDED	A new application has been installed
Intent.ACTION_POWER_CONNECTED	The device has been plugged in
Intent.ACTION_POWER_DISCONNECTED	The device has been disconnected again
KeyChain.ACTION_STORAGE_CHANGED	The keystore has changed
BluetoothDevice.ACTION_ACL_CONNECTED	A Bluetooth ACL connection has been established
AudioManager.ACTION_AUDIO_BECOMING_NOISY	The internal audio speaker is about to be used instead of other output means (like a headset)

Table 9.1: Intent Constants

9.3.4 Role of Filters

When Intent is sent to the Android system, it determines suitable applications for this Intent. In case several components have been registered for this type of Intent, Android offers the user the choice to open one of them.

This decision is based on `IntentFilters`. To inform the system which implicit intents they can handle, activities, services, and broadcast receivers can have more than one intent filter. Each filter declares the capability of a component and a set of intents that a component can handle. It filters in the intents of a desired type, while filtering out the unwanted intents. However, only unwanted implicit intents that do not name a target class are filtered out. An explicit intent is always delivered to its target without consulting the filter.

Let's take an example of NotePad Sample Application which contains two filters, where a component has separate filter for each jobs it can do, one for starting up with a specific note that a user can view or edit, and second for starting a new, blank note that a user can fill in and save.

An intent filter is an instance of the `IntentFilter` class. However, the Android system requires pre-defined knowledge about the capabilities of a component before launching it, therefore, intent filters are usually not set up in Java code, but defined in the `AndroidManifest.xml` file as `<intent-filter>` elements.

Code snippet 22 shows an example of registering an activity. The following code will register an `Activity` for the `Intent` which is triggered when a user wants to view.

Code Snippet 22:

```
<activity android:name=".BrowserActivitiy"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http" />
    </intent-filter>
</activity>
```

A filter has fields that parallel the action, data, and category fields of an `Intent` object. An implicit intent is tested against the filter in all three categories. In order to be delivered to a component that owns the filter, an implicit intent must pass all three tests. In case it fails even one of them, the Android system will not deliver it to the component. However, since a component can have multiple intent filters, an intent that could not pass through one of the component's filters might pass through another.

Caution: An intent filter is not meant for security. While it opens a component to receive only a certain kinds of implicit intents, it does not prevent explicit intents from attacking the component.

9.3.5 Intent Matching

Matching of intents with intent filters is performed to discover a target component that is to be activated, and also to get information about the set of components available on

the device. For instance, the Android system populates the application launcher that is available for the user to launch, by finding all the activities with intent filters that specify the `android.intent.action.MAIN` action and `android.intent.category.LAUNCHER` category. In the `AndroidManifest.xml` file, it also shows the icons and labels of the activities in the launcher. In the same way, it discovers the home screen by looking for the activity with `android.intent.category.HOME` in its filter.

An intent filter can match against the three characteristics – actions, data, and categories in an `Intent`. To order multiple matching filters, intent filters include a ‘priority’ value. Intent filter objects are generally created in XML as part of a package’s `AndroidManifest.xml` file, using intent filter tags.

→ Intent-Matching Rules

An intent matching is based on three rules. To match an `IntentFilter` with an `Intent`, three conditions must and should be fulfilled—the action, the category, and the data (both the data type and data scheme+authority+path, if specified) must match.

Action matches if any of the given values match the Intent action or if no action were specified in the filter.

The **Data** characteristic is divided into four attributes—types, schemes, authority, and path. **Data Type** matches if any of the given values match the Intent type. The Intent type is determined by invoking `resolveType(ContentResolver)`. A wildcard can be used for the MIME sub-type, in both the Intent and IntentFilter, so that the type ‘`audio/*`’ will match ‘`audio/mpeg`’, ‘`audio/aiff`’, ‘`audio/*`’, and so on. MIME type matching here is case sensitive. Therefore, only lower case letters should be used for the MIME types.

Data Scheme matches when any of the given values match the Intent data’s scheme. The Intent scheme is determined by calling `getData()` and `getScheme()` on that URI. Scheme matching is case sensitive. Therefore, only lower case letters should be used.

Data Authority matches the given values with Intent data’s authority and any one of the data’s schemes in the filter matches the Intent, or no authorities are supplied to the filter. The Intent authority is determined by calling `getData()` and `getAuthority()` on that URI. Authority matching is case sensitive and only lower case letters can be used.

Data Path matches any given values with the Intent’s data path and when both a scheme and an authority in the filter matches the Intent, or no paths are supplied in the filter. The Intent authority is determined by calling `getData()` and `getPath()` on the URI.

9.3.6 Filters in Manifest

This section explains how filters in Android are reflected in the `AndroidManifest.xml` file.

Intents activate core components of an application, which are its activities, services, and broadcast receivers. Intent is a collection of information that describes a desired action—including the data that needs to be acted upon, the category of component that performs the action, and other important instructions. Android searches for appropriate component

to reply to the intent, launches a new instance of the component if required, and passes it to the Intent object.

By using intent filters, components specify their capability—that is the kinds of intents they can respond to. The Android system need to know which intents can be handled by a component before it launches the component. Intent filters are specified in the manifest as `<intent-filter>` elements. In other words, using Intent Filters an application component tells Android that they can service action requests from others. A component can have any number of filters, every filter describing a diverse capability. To register an application component as an intent handler, the `<intent-filter>` tag is used within the components node in the **AndroidManifest.xml** file.

When an intent explicitly names a target component, it activates that component; the filter does not play any role. But when intent does not specify a target by name, it activates a component only if it can pass through one of the component's filters.

The default intent filter declaration in **AndroidManifest.xml** file is as shown in code snippet 23.

Code Snippet 23:

```
<intent-filter android:icon="resource path from drawables"
    android:label="resource path of string"
    android:priority="integer">

</intent-filter>
```

9.3.7 Filters in Broadcast Receivers

When the user registers a `BroadcastReceiver` to be executed in the main activity thread, the receiver is called with any broadcast Intent that matches the filter, in the main application thread.

The system can broadcast Intents that are ‘sticky’—the intents that stay around even after the broadcast is complete—to be sent to any later registrations. If the `IntentFilter` matches any of these sticky intents, that Intent will be returned and sent to the receiver as if it had just been broadcast.

There might be times when multiple sticky intents may match the filter, and then in that case each of the intents will be sent to the receiver. Only one of these intents can be returned directly by the function and that is randomly decided by the system.

When the user knows that the registered Intent is sticky, then null can be supplied for the receiver. In case when no receiver is registered, the function simply returns the sticky Intent that matches the filter. The same rules apply when there are multiple matches.

Code snippet 24 explains the usage of filters in broadcast receivers.

Code Snippet 24:

```
<receiver android:name=".BCastReceiver">  
    <intent-filter>  
        <action android:name="android.intent.action.MAIN" />  
  
        <category android:name="android.intent.category.LAUNCHER" />  
        <action android:name="android.intent.action.ACTION_POWER_CONNECTED" />  
    </intent-filter>  
    </receiver>  
    ...
```

The intent-filter here is used to specify the action the receiver has to perform. Here in this code snippet, ACTION_POWER_CONNECTED, tells the broadcast receiver to notify when power cable connected to device and is handled by the `onReceive()` method in the BCastReceiver class.

Note - This method cannot be called from a `BroadcastReceiver` component, that is, from a `BroadcastReceiver` that is declared in an application's manifest. However, this method can be called from another `BroadcastReceiver` that has itself been registered at run time with `registerReceiver(BroadcastReceiver, IntentFilter)`, since the duration of such a registered `BroadcastReceiver` depends on the object that registered it.

9.4 Check Your Progress

1. Which of the following method is called to start a service?

(A)	stopService()	(C)	beginService()
(B)	startService()	(D)	receiveService()

2. Which of the following method does not support multi-threading?

(A)	onStartCommand()	(C)	onCreate()
(B)	IntentService	(D)	onDestory()

3. A service can be started from an activity or a component by passing an _____ to startService() method.

(A)	Method	(C)	Class
(B)	Intent	(D)	Filter

4. When the service is handling multiple requests by using onStartCommand(), then which method needs to be called to stop the service based on the most recent start request?

(A)	stopService()	(C)	onstopService()
(B)	stopSelf()	(D)	stopSelf(int)

5. Match the method with its function.

(A)	onBind()	(1)	Components can use this to stop service
(B)	stopSelf()	(2)	System initiates a component
(C)	stopService()	(3)	To create a bound service
(D)	onCreate()	(4)	A started service stops itself

6. _____ method is invoked when the class extends Service.

(A)	onBind()	(C)	onCreate()
(B)	onStart()	(D)	bindService()

7. Broadcast Receiver invokes which method in the following option?

(A)	onReceive()	(C)	Context.registerReceiver()
(B)	onDestroy()	(D)	onNew()

8. Match the events with their correct usage.

(A)	Intent.ACTION_BATTERY_OKAY	(1)	The application has been updated
(B)	Intent.ACTION_MY_PACKAGE_REPLACED	(2)	The battery level has increased again
(C)	Intent.ACTION_PACKAGE_ADDED	(3)	The keystore has changed
(D)	KeyChain.ACTION_STORAGE_CHANGED	(4)	A new application has been installed

9. Where to include the service class name in Android?

(A)	Assets Folder	(C)	Manifest file
(B)	AIDL	(D)	Raw Folder

10. The **Data** characteristic is divided into four attributes—types, _____, authority, and path.

(A)	Service	(C)	Intents
(B)	Broadcast Receivers	(D)	Schemes

9.4.1 Answers

1.	B
2.	C
3.	B
4.	D
5.	A-3, B-4, C-1, D-2
6.	C
7	A
8.	A-2, B-1, C-4, D-3
9.	C
10.	D



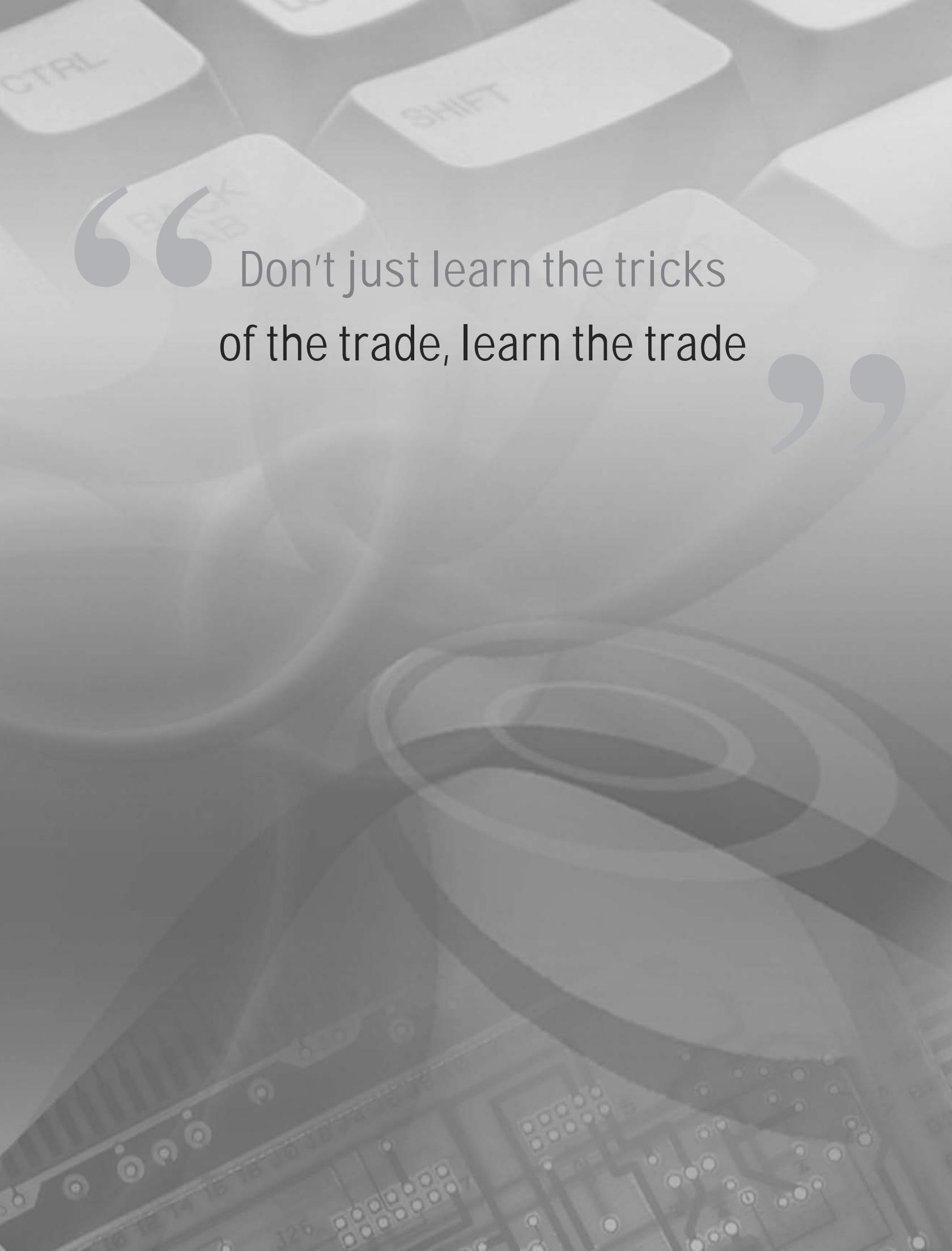
Summary

- ➔ A Service is an application component that is able to carry out long-running operations in the background; it does not provide a user interface.
- ➔ In order to interact with the components, a service can let other components bind to it and perform Inter Process Communication (IPC).
- ➔ A service runs in the main thread of the application that hosts it.
- ➔ A Broadcast Receiver is a component that responds to system-wide broadcast announcements.
- ➔ A Broadcast Receiver (short receiver) is an Android component which allows you to register for system or application events.
- ➔ Activities, Services, and Broadcast Receivers have intent filters to inform the system which intents they can handle.
- ➔ Intent filters can handle two types of intents, implicit and explicit.
- ➔ Matching of intents with intent filters is done to discover a target component to be activated, and also to get information about the set of components available on the device.
- ➔ To match an intent filter with an Intent, three conditions must be fulfilled—the action, category, and the data.



Samantha who is learning Android programming wants to create an application where a message stating the battery status will be notified to user with the help of Broadcast Receiver. You as a developer is required to help her to achieve this.

“ Don't just learn the tricks
of the trade, learn the trade ”



JELLYBEAN ICE CREAM SANDWICH

Session - 10

Bluetooth, Network, Wi-Fi, and Sensors

Welcome to the session of **Bluetooth, Network, Wi-Fi, and Sensors**.

This session explains certain Android features that can be used in an Android application. It explains the low level communication APIs by checking the Bluetooth, network, and Wi-Fi packages. Bluetooth APIs help to manage and monitor Bluetooth devices whereas Wi-Fi APIs helps to create hotspots and modify the configuration settings.

In this session, you will learn to:

- ➔ Explain Bluetooth, Network, Wi-Fi, and Sensors
- ➔ Explain the process of using Bluetooth
- ➔ Explain network management and Internet connectivity
- ➔ Explain working with Wi-Fi
- ➔ Explain working with Sensors



10.1 Introduction

This session deals with the Bluetooth, which allows the user to transfer data between bluetooth connected devices through wireless medium within a short range of distance. Network, is used for connecting to Internet and accessing data. The session proceeds to explain about WifiManager which represents the connectivity with the Wi-Fi network and Sensors which helps to measure motion, environmental conditions, position, and so on.

10.2 Bluetooth

Bluetooth presents a network stack that allows the user to transfer data between Bluetooth connected devices within a short range of distance. These devices wirelessly exchange data.

Android provides Bluetooth APIs, which can perform the following functionalities:

- ➔ Scan for Bluetooth devices which are present nearby.
- ➔ Query for paired Bluetooth devices using the local Bluetooth adapter.
- ➔ Establishing the RFCOMM channels.
- ➔ Connect through service discovery to other devices.
- ➔ Exchange data to and from other devices.
- ➔ Manage multiple connections.

Android provides all the Bluetooth API's under `android.bluetooth` package.

The main classes and interfaces required to create Bluetooth connections are as follows:

- ➔ `BluetoothAdapter`
- ➔ `BluetoothDevice`
- ➔ `BluetoothSocket`
- ➔ `BluetoothServerSocket`
- ➔ `BluetoothClass`
- ➔ `BluetoothProfile`

- BluetoothHandset
- BluetoothA2dp
- BluetoothHealth
- BluetoothHealthCallback
- BluetoothHealthAppConfiguration
- BluetoothProfile.ServiceListener

The main functionalities of the classes are explained in detail.

→ **BluetoothAdapter:**

Is a representation of the local Bluetooth adapter and helps to perform the following functionalities:

- Discover all Bluetooth devices.
- Query a list of devices.
- Create a Bluetooth device instance using the MAC address.
- Create a `BluetoothServerSocket` which will enable it to listen for communications from other Bluetooth connected devices.

→ **BluetoothDevice:**

Is a representation of a remote Bluetooth device and performs the following functionalities:

- Request a connection with remote device using `BluetoothSocket`.
- Retrieve information about the device such as name, address, and state of the device.

→ **BluetoothSocket:**

Is a representation of the interface like TCP socket and performs the following functionalities:

- Allows data exchange between the bluetooth connected devices via `InputStream` and `OutputStream` can occur through `BluetoothSocket`.

→ BluetoothServerSocket:

Is a representation of an open server socket and performs the following functionalities:

- Listens for the incoming requests.
- Responds to the received requests by returning a Bluetooth socket when the connection is accepted.

→ Bluetooth Class:

Is a read only set of properties that describe the characteristics and capabilities of a Bluetooth device and all Bluetooth profiles and services supported by the particular device.

→ Bluetooth Profile:

Is a wireless interface specification for communication between Bluetooth based devices.

→ Bluetooth Handset:

It provides support for Bluetooth headsets that are connected with the mobile device.

→ BluetoothA2dp:

It stands for Advanced Audio Distribution Profile and performs the following functionality:

- Defines the way of streaming a high quality audio from one Bluetooth connected device to another.

→ Bluetooth Health:

It represents a Health Device profile proxy and is used for controlling the Bluetooth service.

→ BluetoothHealthCallback:

Is an abstract class that the callback methods must be implemented for the following functionality:

- Receive updates about the changes in the Bluetooth channel state and application registration.

→ BluetoothHealthAppConfiguration:

It represents a configuration of the Bluetooth Health third party application that is used for communicating with a remote Bluetooth health device.

→ **BluetoothProfile.Servicelistener:**

Is an interface that performs the following functionality:

- Notifies the IPC clients whenever they are connected or disconnected from the service.

The following application turns on and off the Bluetooth, searches for the nearest Bluetooth connected devices, and displays them in list view.

1. Start Eclipse.
2. Create a project named **BlueToothExample**.
3. Navigate to **res → layout** folder.
4. Modify the code in **activity_main.xml** file as shown in code snippet 1.

Code Snippet 1:

```
<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" >

    <ToggleButton
        android:id="@+id/tb"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:text="ToggleButton"
```

```

        android:textOff="Bluetooth is Off"
        android:textOn="Bluetooth is On" />

    <Button
        android:id="@+id/buttonSearch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/tb"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="18dp"
        android:text="Search Near Devices" />

    <ListView
        android:id="@+id/listView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/buttonSearch"
        android:layout_centerHorizontal="true" />
    </ListView>

</RelativeLayout>
```

5. Navigate to **src → com.example.bluetoothexample** folder.
6. Modify the code in **MainActivity** file as shown in code snippet 2.

Code Snippet 2:

```

package com.example.bluetoothexample;

import java.util.ArrayList;
import java.util.Set;

import android.os.Bundle;
import android.app.Activity;
```

```
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.ToggleButton;

public class MainActivity extends Activity implements
OnClickListener {

    private ToggleButton tb;
    private BluetoothAdapter bl;
    Button search;
    ListView listView;

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

        tb = (ToggleButton) findViewById(R.id.tb);
        tb.setOnClickListener(this);
        bl = BluetoothAdapter.getDefaultAdapter();
        search = (Button) findViewById(R.id.buttonSearch);
        listView = (ListView) findViewById(R.id.listView1);

        search.setOnClickListener(new OnClickListener()
    }

}
```

```
    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
        searchDevices();

    }

}

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

@Override
public void onClick(View v) {
    if (v == tb) {
        if ((tb.isChecked())) {
            try {
                Intent discoverableIntent = new Intent(
                    BluetoothAdapter.ACTION_
REQUEST_DISCOVERABLE);

                discoverableIntent.putExtra(
                    BluetoothAdapter.EXTRA_
DISCOVERABLE_DURATION, 300);

                startActivity(discoverableIntent);
            // b1.enable();
        } catch (Exception e) {
            }
        }
    }
}
```

```
        } else {
            try {
                bl.disable();
            } catch (Exception ee) {

            }
        }

    }

public void searchDevices() {

    BluetoothAdapter mBluetoothAdapter =
    BluetoothAdapter
        .getDefaultAdapter();
    Set<BluetoothDevice> pairedDevices =
    mBluetoothAdapter
        .getBondedDevices();

    ArrayList<String> s = new ArrayList<String>();
    for (BluetoothDevice bt : pairedDevices)
        s.add(bt.getName());
    listView.setAdapter(new
    ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1,
        s));
}

}
```

7. Modify the code in **AndroidManifest** file and grant the permissions for Bluetooth as shown in code snippet 3.

Code Snippet 3:

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

<manifest xmlns:android="http://schemas.android.com/apk/
res/android"

    package="com.example.bluetoothexample"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

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

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

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="com.example.
            bluetoothexample.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>
```

To deploy the Android application on an Android-enabled mobile device, perform the following steps:

8. Connect the mobile device through USB cable.
9. Enable USB to connect to storage/device.
10. Copy **<application-name>.apk** (bin folder of project) to a folder on the mobile device.
11. Disconnect the USB from storage (need not remove the cable, just disable the connection).
12. Click the **<application-name>.apk** file to install the application on the mobile device.
13. The application is installed and an app icon appears in the application area on the screen.
14. Click the application icon to run it.

Figure 10.1 displays the output when connected with an actual android device.

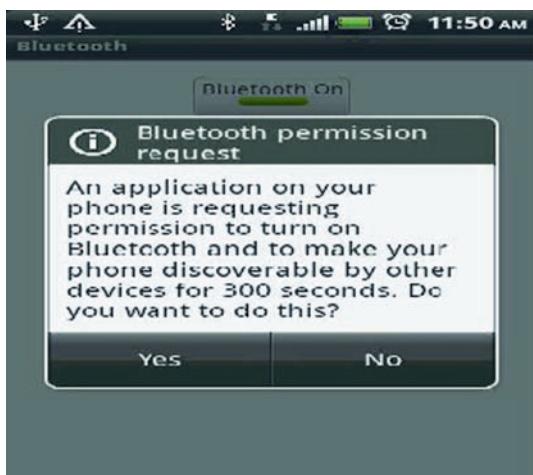


Figure 10.1: Bluetooth

Figure 10.2 is displayed when the **Bluetooth On** is clicked.



Figure 10.2: Bluetooth On

Figure 10.3 is displayed when **Search Near Devices** is clicked. It searches for nearest Bluetooth paired connected devices and displays them.



Figure 10.3: Bluetooth Searching

Note - This particular application can be run only in real time Android devices since emulator cannot check the availability of Bluetooth devices.

10.3 Network Operations

This section explains how to establish a Network connection and how to communicate with the network, download an image, and so on. Most of Android applications performing network operations use HTTP to send and receive data. Android uses following two HTTP clients:

→ **HttpClient:**

`HttpClient` encapsulates a smorgasbord objects which are required to execute HTTP requests to send and receive the data over the network. `HttpClient` simplifies the handling of Http Requests. The thread safety of `HttpClient` depends on the way the configuration of a specific client is being made.

→ **HttpURLConnection:**

`HttpURLConnection` is used to make a single request to the Http Server. Connection for the particular URL can be established by invoking the `openConnection()` method and the data from the URL can be read by invoking the method `getInputStream()` of the `InputStreamReader` class. It is used for sending and receiving data over the Web.

The class uses the `GET` and `POST` method to send and receive data. By default it uses the `GET` method. The `POST` method is used if the method `setDoOutput(true)` method has been invoked. The other HTTP methods such as `HEAD`, `PUT`, `DELETE`, and so on are used with `setRequest()` method.

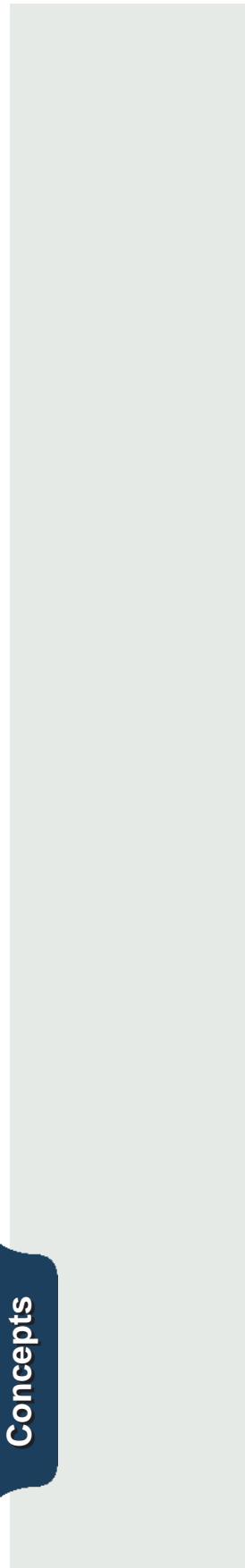
The following application checks whether the network connection is available and downloads the JSON data.

To develop the application, perform the following steps:

1. Start Eclipse.
2. Create a project named `NetworkConnection`.
3. Navigate to `res → layout` folder.
4. Modify the code in `activity_main.xml` file as shown in code snippet 4.

Code Snippet 4:

```
<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/buttonCheck"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="26dp"
        android:text="Check Network Availability" />

    <Button
        android:id="@+id/buttonDownload"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/buttonCheck"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="28dp"
        android:text="Download data" />

    <ScrollView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/buttonDownload" >
```

```
<TextView  
    android:id="@+id/textView1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_marginTop="16dp"  
    android:text=""  
    android:textAppearance="?android:attr/  
    textAppearanceMedium" />  
  
</ScrollView>  
  
</RelativeLayout>
```

5. Navigate to **src → com.example.networkconnection** folder.
6. Modify the code in **MainActivity** file as shown in code snippet 5.

Code Snippet 5:

```
package com.example.networkconnection;  
  
import android.os.Bundle;  
import android.app.Activity;  
import android.view.Menu;  
import java.io.BufferedReader;  
import java.io.InputStreamReader;  
import java.io.OutputStreamWriter;  
import java.net.HttpURLConnection;  
import java.net.URL;  
  
import org.json.JSONArray;  
import org.json.JSONException;  
import org.json.JSONObject;  
  
import android.content.Context;  
import android.net.ConnectivityManager;
```

```
import android.net.NetworkInfo;
import android.os.AsyncTask;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {

    Button checkNetwork;
    Button downloadData;
    TextView showResult;

    URL url;
    HttpURLConnection connection = null;
    OutputStreamWriter request = null;

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

        checkNetwork = (Button) findViewById
            (R.id.buttonCheck);
        downloadData = (Button) findViewById
            (R.id.buttonDownload);
        showResult = (TextView) findViewById
            (R.id.textView1);

        checkNetwork.setOnClickListener
            (new OnClickListener() {
```

```
@Override  
public void onClick(View v) {  
    // TODO Auto-generated method stub  
  
    if (checkNetwork()) {  
        Toast.makeText(MainActivity.this,  
                "Network connection available",  
                Toast.LENGTH_LONG)  
                .show();  
    } else {  
        Toast.makeText(MainActivity.this,  
                "Network connection not  
                available", Toast.LENGTH_LONG).  
                show();  
    }  
}  
});  
  
downloadData.setOnClickListener(new  
OnClickListener() {  
  
    @Override  
    public void onClick(View v) {  
        // TODO Auto-generated method stub  
  
        if (checkNetwork()) {  
            new DownloadTask().execute();  
        } else {  
            Toast.makeText  
(getApplicationContext(),
```

```
        "Network connection not available",
        Toast.LENGTH_LONG).show();
    }

}

}

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

public boolean checkNetwork() {

    boolean haveConnectedWifi=false;
    boolean haveConnectedMobile=false;
    boolean haveConnectedActive=false;
    ConnectivityManager connMgr;
    android.net.NetworkInfo wifi;
    android.net.NetworkInfo mobile;

    /*
     * Checking the availability of network
     */

    connMgr = (ConnectivityManager) MainActivity.this
        .getSystemService(Context.
CONNECTIVITY_SERVICE);
```

```
wifi = connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);

mobile = connMgr.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);

NetworkInfo activeNetworkInfo = connMgr.getActiveNetworkInfo();

if (activeNetworkInfo != null) {

    haveConnectedActive = true;

}

if (wifi.isAvailable()) {

    haveConnectedWifi = true;

}

if (mobile.isAvailable()) {

    haveConnectedMobile = true;

}

// if network is connected and either Wi-Fi or mobile
available means

// return true , else false

if ((haveConnectedWifi == true || haveConnectedMobile == true)

    && haveConnectedActive == true) {

    return true;

} else {

    return false;

}

}

public class DownloadTask extends AsyncTask<Context, String, String> {
```

```
String result;

@Override
protected String doInBackground(Context... arg0)
{
    // TODO Auto-generated method stub

    try {
        url = new URL(
            "http://free.worldweatheronline.
            com/feed/weather.ashx?q="
            + "bangalore"
            + "&format=json"
            + "_of_days=5&key=fc137
            2a610062229130602");

        connection = (HttpURLConnection) url.
        openConnection();
        connection.setDoOutput(true);
        connection.
        setRequestProperty("Content-Type",
            "application/x-www-form-
            urlencoded");
        connection.setRequestMethod("POST");
        request = new OutputStreamWriter
        (connection.getOutputStream());
        request.flush();
        request.close();
        StringBuilder sb = new StringBuilder();
        InputStreamReader isr = new
        InputStreamReader(
            connection.getInputStream());
        BufferedReader reader = new
        BufferedReader(isr);
        String line;
        while ((line = reader.readLine()) != null) {
```

```
        sb.append(line + "\n");
        result = sb.toString();

    }

} catch (Exception e) {
    e.printStackTrace();
}

return result;
}

@Override
protected void onProgressUpdate(String... values)
{
    // TODO Auto-generated method stub
    super.onProgressUpdate(values);

}

@Override
protected void onPreExecute() {
    // TODO Auto-generated method stub
    super.onPreExecute();

    Toast.makeText(getApplicationContext(),
    "Fetching data....",
    Toast.LENGTH_LONG).show();

}

@Override
protected void onPostExecute(String result) {
```

```
// TODO Auto-generated method stub
super.onPostExecute(result);

System.out.println("result is " + result);

// response[0] = result;
try {
    JSONObject rootObj = new
    JSONObject(result);
    JSONObject rootObj1 = rootObj.
    getJSONObject("data");
    if (!rootObj1.has("error")) {

        JSONArray rooms_array = rootObj1
            .getJSONArray("current_
            condition");

        JSONObject jsonObj = rooms_array.
        getJSONObject(0);

        String humidity = jsonObj.
        getString("humidity");
        String temperature = jsonObj.
        getString("temp_C");
        String pressure = jsonObj.
        getString("pressure");

        showResult.setText("\n\n" +
        "Temperature : " + temperature
        + "\n\n" + "Humidity : " +
        humidity + "\n\n"
        + "Pressure : " + pressure);

    }
}
```

```
        } catch (JSONException e) {
            Toast.makeText(getApplicationContext(),
                    "Unable to reach Server", Toast.LENGTH_SHORT).show();
        }

        e.printStackTrace();
        finish();
    } catch (NullPointerException e) {
        Toast.makeText(
                getApplicationContext(),
                "Unable to reach Server", Toast.LENGTH_SHORT).show();
    }

    e.printStackTrace();
    finish();
}

}

}
```

The source code checks network connections, establishes network connections, downloads data, and parses data using JSON.

7. Modify the **AndroidManifest.xml** file as shown in code snippet 6. It is modified to grant the permissions required for accessing the network.

Code Snippet 6:

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/  
res/android"  
    package="com.example.networkconnection"  
    android:versionCode="1"  
    android:versionName="1.0" >
```

```
<uses-sdk  
    android:minSdkVersion="8"  
    android:targetSdkVersion="17" />  
  
<uses-permission android:name="android.permission.  
INTERNET" />  
<uses-permission android:name="android.permission.  
ACCESS_NETWORK_STATE" />  
  
<application  
    android:allowBackup="true"  
    android:icon="@drawable/ic_launcher"  
    android:label="@string/app_name"  
    android:theme="@style/AppTheme" >  
    <activity  
        android:name="com.example.  
networkconnection.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>
```

Figure 10.4 displays the output when the application is executed.

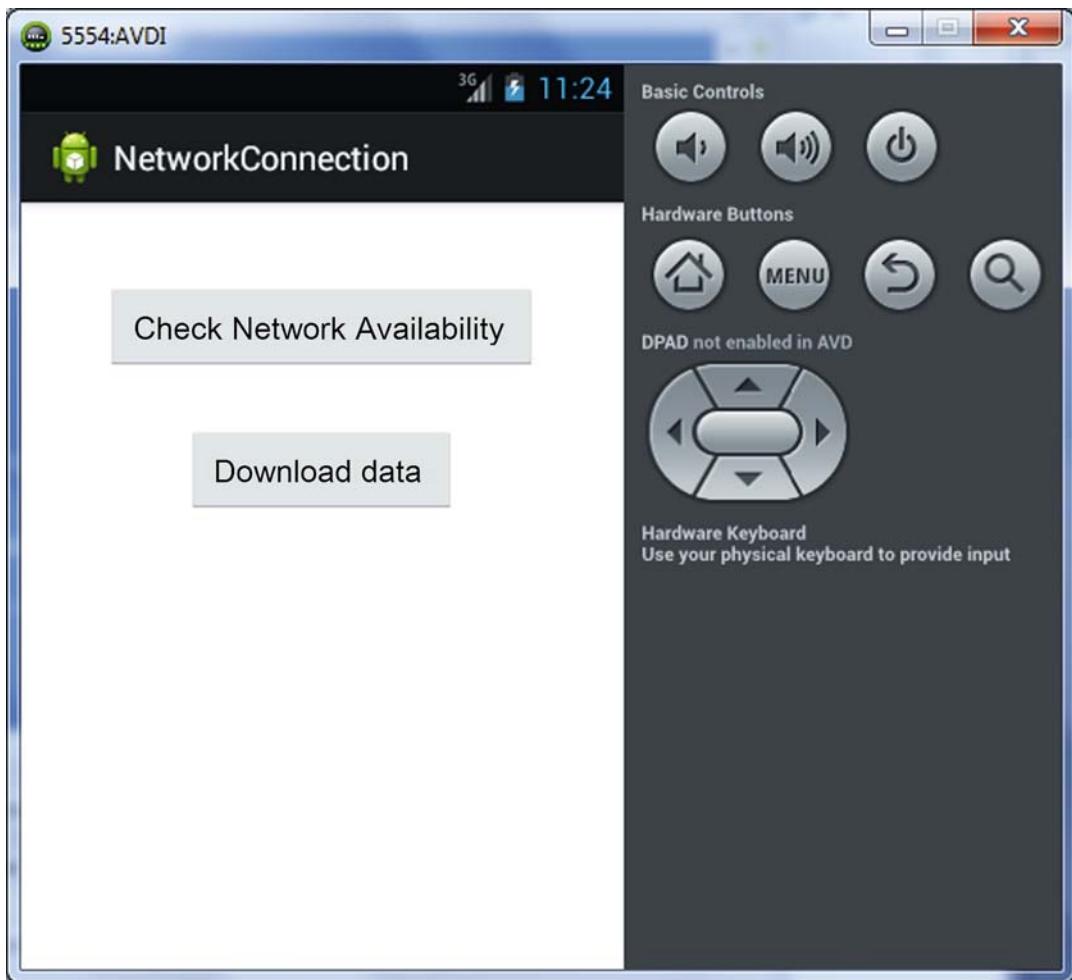


Figure 10.4: Output - Network Application

Click **Check Network Availability** to display the output as shown in figure 10.5.

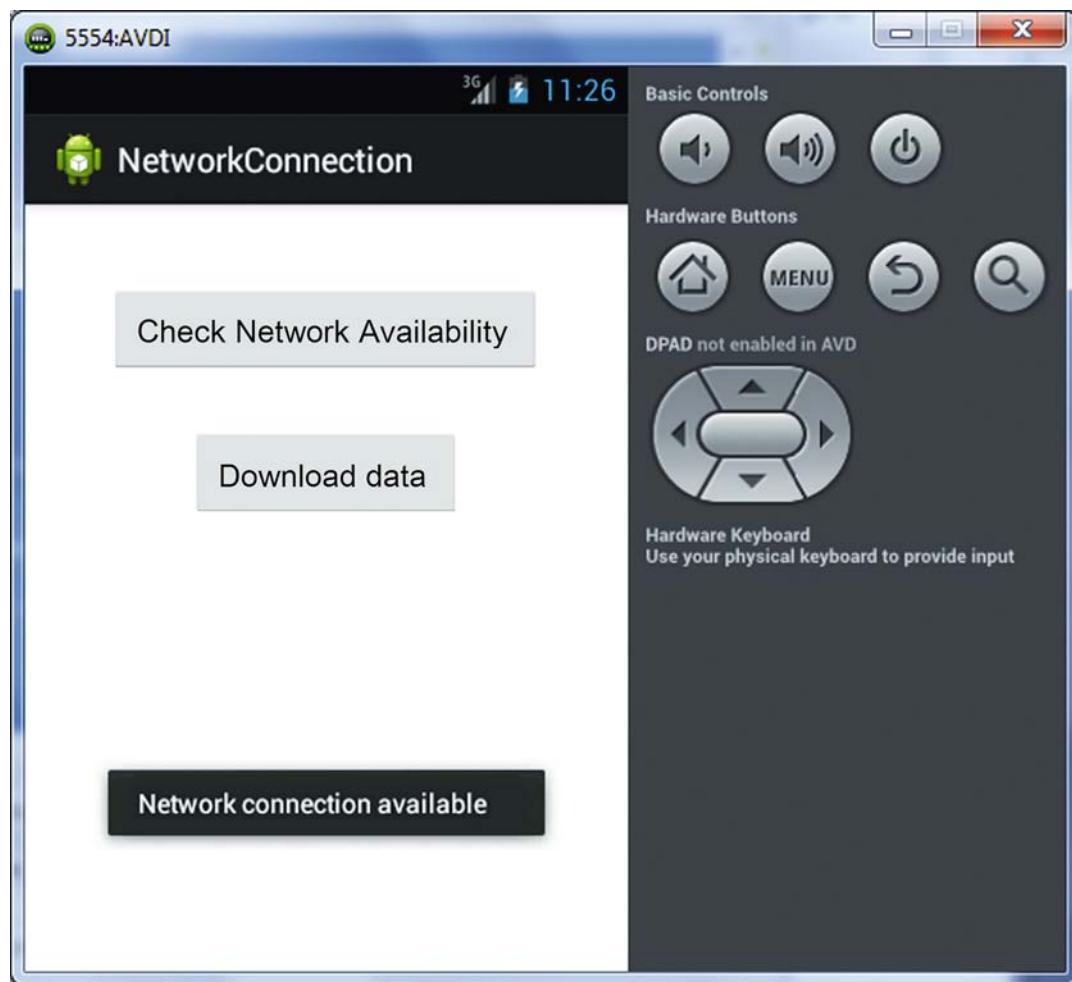


Figure 10.5: Check Network Availability

Figure 10.6 displays the response when **Download Data** is clicked.

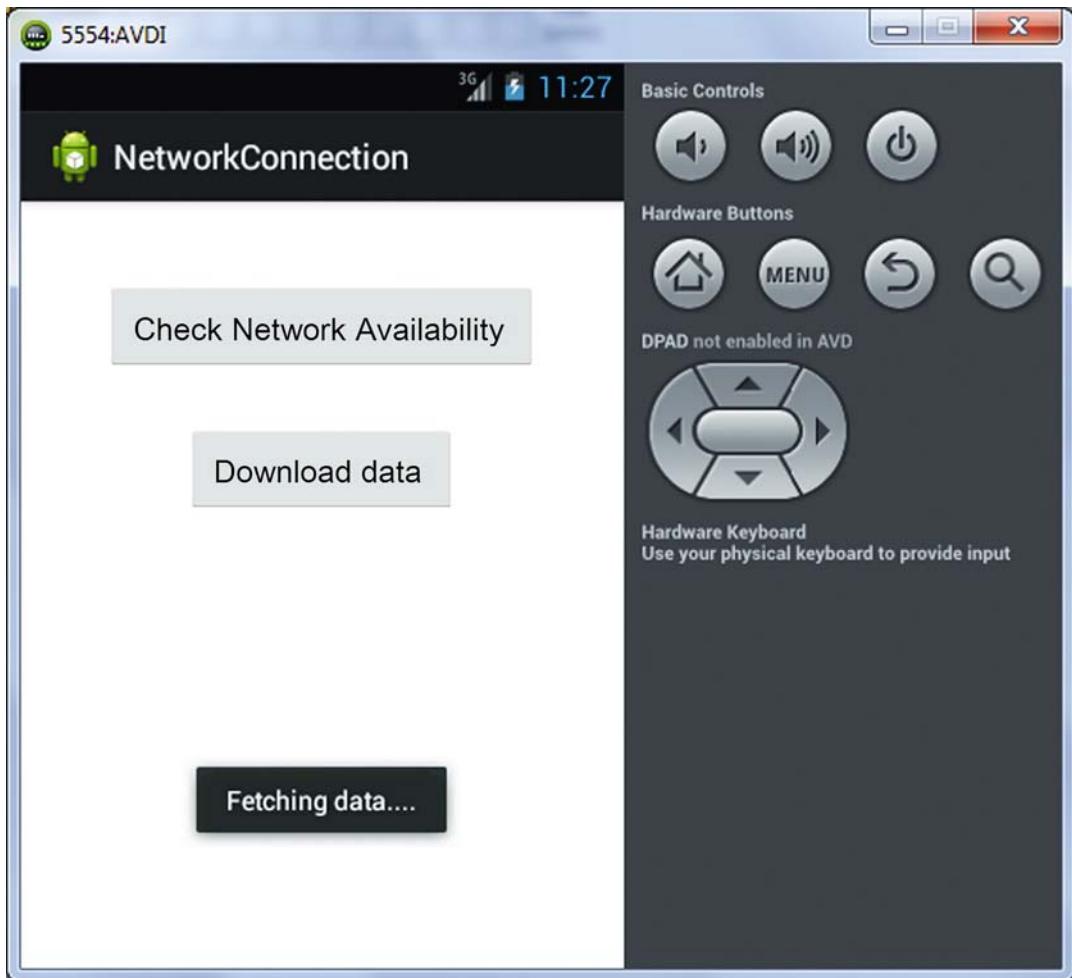


Figure 10.6: Response from Network Connection

Figure 10.7 displays the final output.

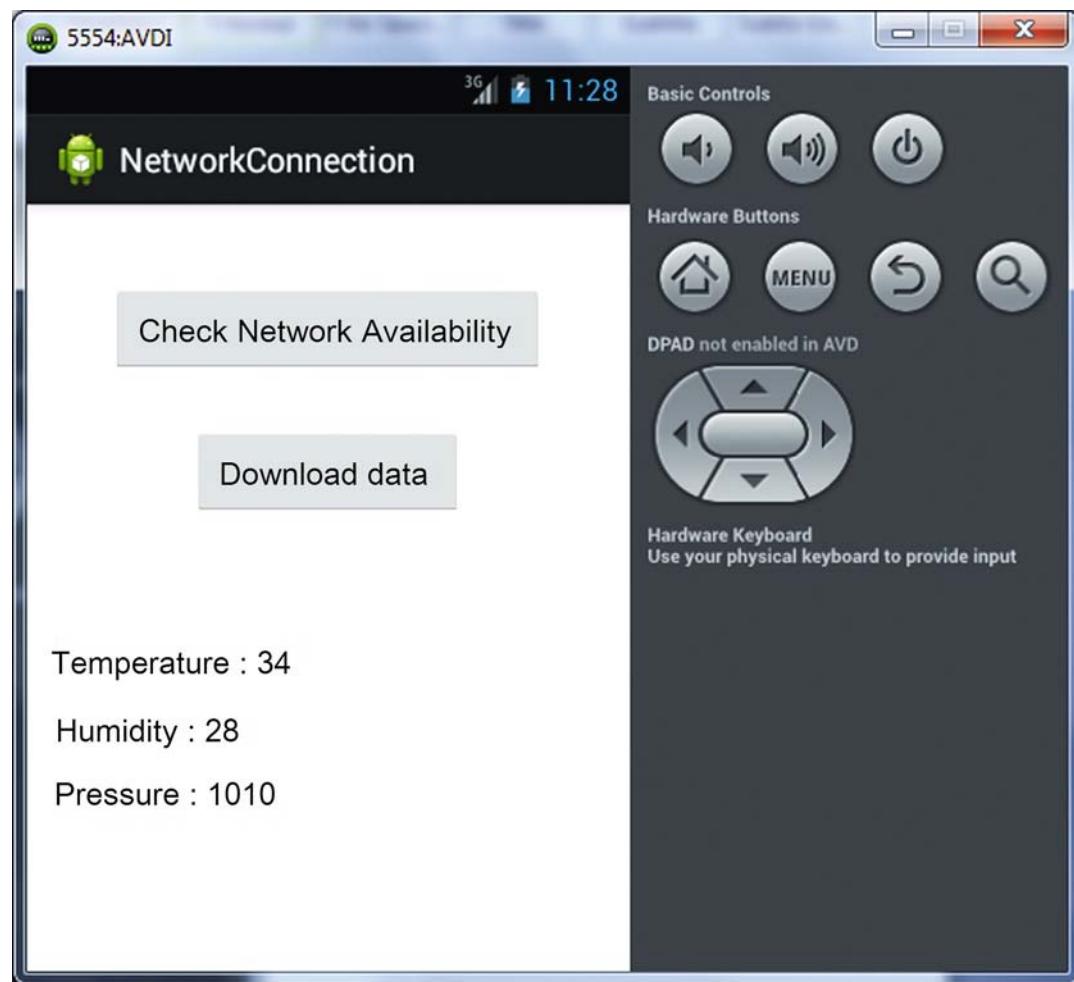


Figure 10.7: Downloaded Data

10.4 Wi-Fi

Wi-Fi, is a technology that allows devices to share network over a computer network.

Android provides Wi-Fi APIs, through which applications can communicate with the wireless stack that provides Wi-Fi network access. These APIs also have the ability to scan, add, save, terminate, and initiate connections. Some of the classes of the APIs are as follows:

→ **WifiManager:**

This class is used to manage the Wi-Fi connectivity by invoking the method, `Context.getSystemService(Context.WIFI_SERVICE)`. In other words, this class is used specifically when one is making Wi-Fi specific operations.

The Wi-Fi Manager can deal with the following:

- List of configured networks.
- Currently available active networks.
- Provide enough information to the developer regarding which network one has to choose and connect to.

→ **ConnectivityManager:**

ConnectivityManager notifies the application whenever any network changes happen. An instance of ConnectivityManager can be created by invoking the method, Context.getSystemService(Context.CONNECTIVITY_SERVICE).

Some of the tasks performed by the ConnectivityManager are as follows:

- Scans Wi-Fi, GPRS, and UMTS and other network connections.
- Sends broadcast intents whenever the connectivity changes
- Queries the state of available network using an API

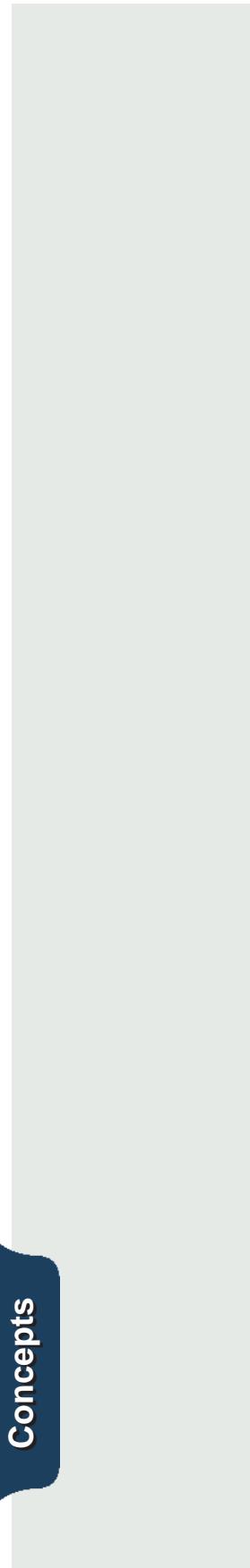
The following example demonstrates how to enable and disable the Wi-Fi using the WifiManager. To develop the application, perform the following steps:

1. Start Eclipse.
2. Create a project named **WifiExample**.
3. Navigate to **res → layout** folder.
4. Modify the code in **activity_main.xml** file as shown in code snippet 7.

Code Snippet 7:

```
<?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:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal" >
```



```
        android:text="WifiManager"  
        android:textSize="40dp" />  
  
<Button  
        android:id="@+id/onwifi"  
        android:layout_width="200dp"  
        android:layout_height="wrap_content"  
        android:layout_gravity="center_horizontal"  
        android:layout_marginTop="20dp"  
        android:text="Turn Wi-Fi On" />  
  
<Button  
        android:id="@+id/offwifi"  
        android:layout_width="200dp"  
        android:layout_height="wrap_content"  
        android:layout_gravity="center_horizontal"  
        android:text="Turn Wi-Fi Off" />  
  
<TextView  
        android:id="@+id/wifistate"  
        android:layout_width="fill_parent"  
        android:layout_height="wrap_content"  
        android:layout_marginTop="50dp"  
        android:textSize="30dp"  
        android:textStyle="bold" />  
  
</LinearLayout>
```

5. Navigate to **src → com.example.wifiexample** folder.
6. Modify the code in **MainActivity** as shown in code snippet 8.

Code Snippet 8:

```
package com.example.wifiexample;

import android.os.Bundle;
import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.net.wifi.WifiManager;

public class MainActivity extends Activity {

    TextView WifiState;

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

        WifiState = (TextView) findViewById(R.id.wifistate);
        Button OnWifi = (Button) findViewById(R.id.onwifi);
        Button OffWifi = (Button) findViewById(R.id.offwifi);

        this.registerReceiver(this.
            WifiStateChangedReceiver, new IntentFilter(
                WifiManager.WIFI_STATE_CHANGED_
                ACTION));
    }
}
```

```
OnWifi.setOnClickListener(new Button.OnClickListener() {

    @Override
    public void onClick(View arg0) {
        // TODO Auto-generated method stub
        WifiManager wifiManager = (WifiManager)
        getBaseContext()
            .getSystemService(Context.WIFI_SERVICE);
        wifiManager.setWifiEnabled(true);
    }
});
```



```
OffWifi.setOnClickListener(new Button.OnClickListener() {

    @Override
    public void onClick(View arg0) {
        // TODO Auto-generated method stub
        WifiManager wifiManager = (WifiManager)
        getBaseContext()
            .getSystemService(Context.WIFI_SERVICE);
        wifiManager.setWifiEnabled(false);
    }
});
```



```
}
```



```
private BroadcastReceiver WifiStateChangedReceiver =
new BroadcastReceiver() {
```



```
    @Override
```

```
public void onReceive(Context context, Intent intent) {  
    // TODO Auto-generated method stub  
  
    int extraWifiState = intent.getIntExtra(  
        WifiManager.EXTRA_WIFI_STATE,  
        WifiManager.WIFI_STATE_UNKNOWN);  
  
    switch (extraWifiState) {  
        case WifiManager.WIFI_STATE_DISABLED:  
            WifiState.setText("WIFI STATE DISABLED");  
            break;  
        case WifiManager.WIFI_STATE_DISABLING:  
            WifiState.setText("WIFI STATE  
DISABLING");  
            break;  
        case WifiManager.WIFI_STATE_ENABLED:  
            WifiState.setText("WIFI STATE ENABLED");  
            break;  
        case WifiManager.WIFI_STATE_ENABLING:  
            WifiState.setText("WIFI STATE ENABLING");  
            break;  
        case WifiManager.WIFI_STATE_UNKNOWN:  
            WifiState.setText("WIFI STATE UNKNOWN");  
            break;  
    }  
}  
};  
}
```

7. Modify the code in **AndroidManifest.xml** file and add the permission as shown in code snippet 9.

Code Snippet 9:

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

<manifest xmlns:android="http://schemas.android.com/apk/
res/android"

    package="com.example.wifiexample"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk

        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

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

    <application

        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity

            android:name="com.example.wifiexample.
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>
```

Figure 10.8 displays the output after executing the code.

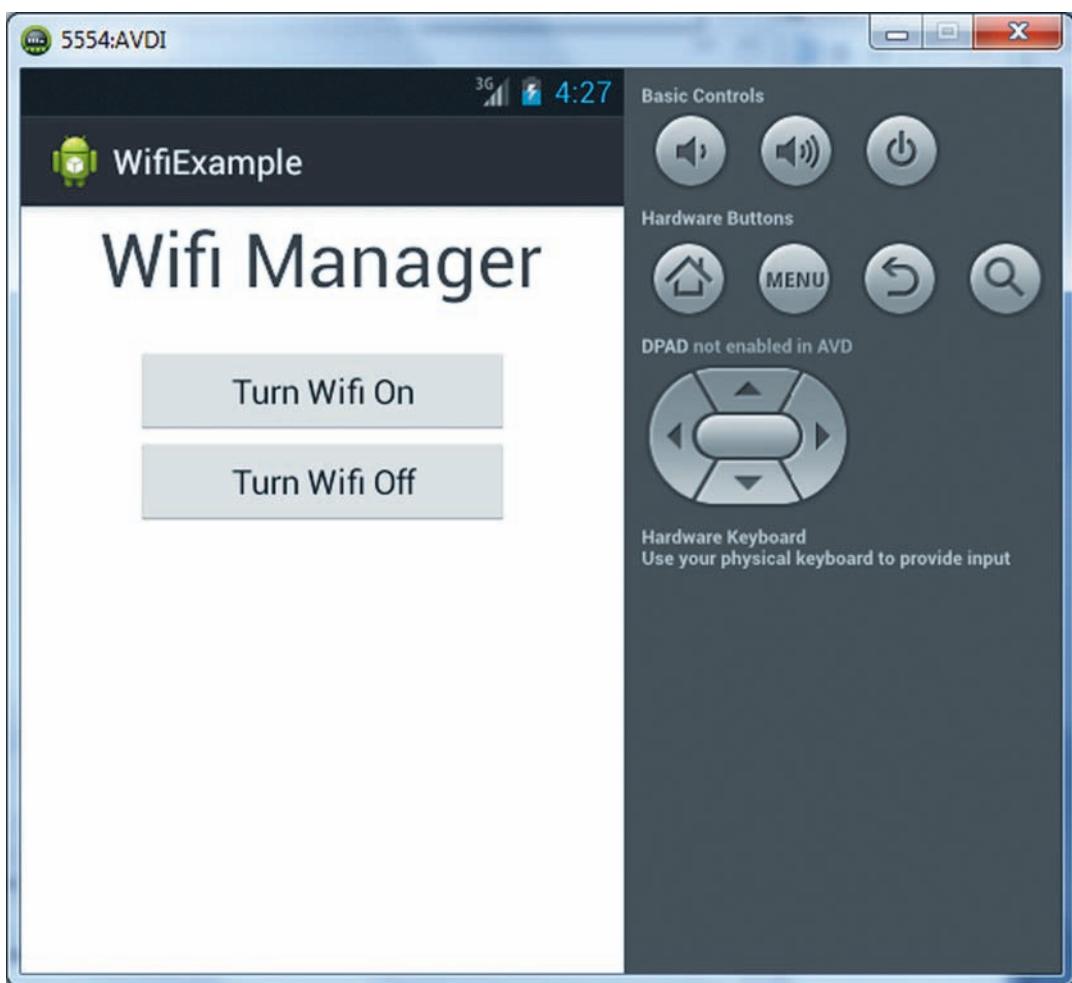


Figure 10.8: WiFiExample – Output

Figure 10.9 displays the output when **Turn Wi-Fi On** is clicked.

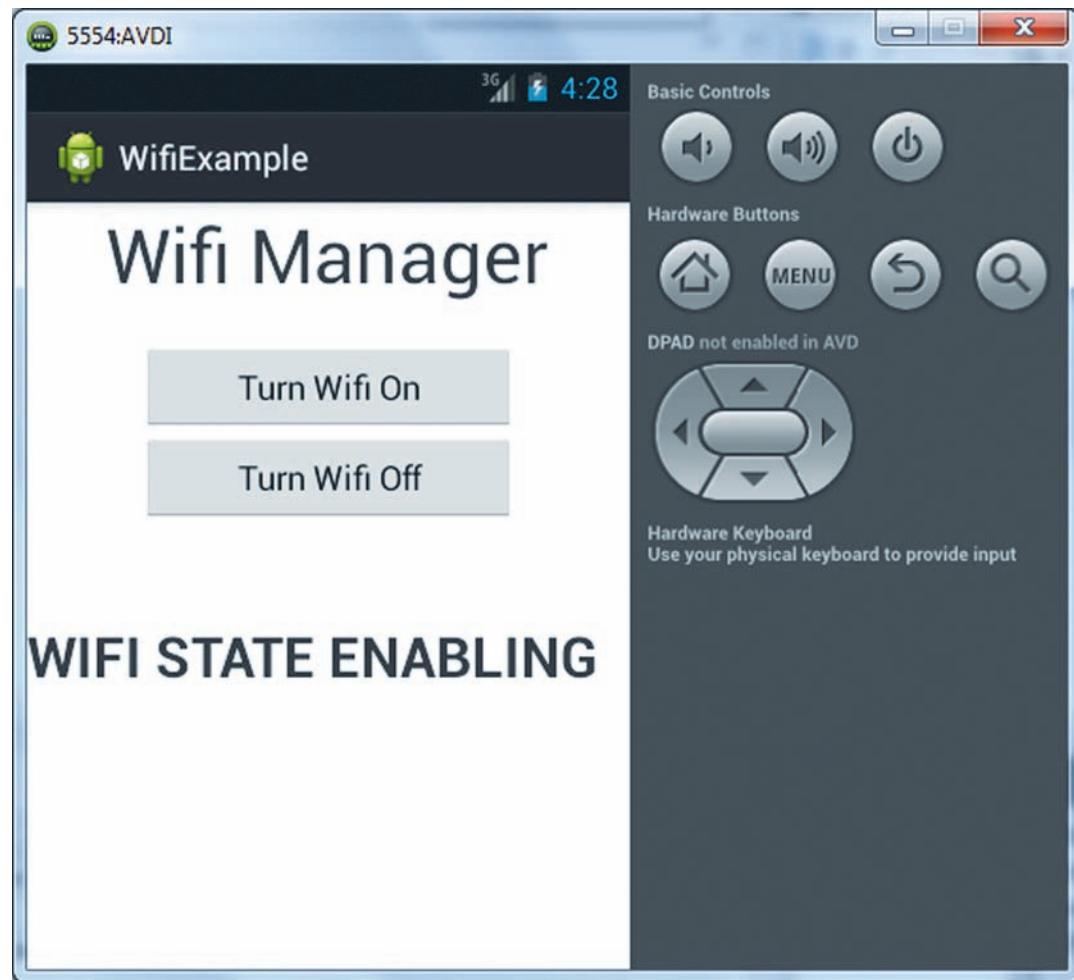


Figure 10.9: Wi-Fi State Enabled

10.4.1 Sensors

Most of the Android powered devices come with the default sensors such as:

- Motion Sensors
- Environmental Sensors
- Position Sensors

These sensors are known to measure the motion, environmental, and orientation changes and provide the accurate data.

One can not only access the sensors which are available on the Android devices but also can retrieve the raw sensor data by using the available Android sensor framework. This framework is a collection of classes and the interfaces which help the developer to perform a variety of sensor-related tasks. The tasks that can be performed by using the sensor framework are as follows:

- Checks the availability of sensors on the device.
- Checks the individual sensor's capabilities such as the maximum range, resolution, power requirements, and so on.
- Defines the minimum rate at which to acquire the raw sensor data.
- Monitors the sensor changes by registering and unregistering the sensor event listeners.

Some of the different types of sensor supported by Android are as follows:

- TYPE_ACCELEROMETER
- TYPE_GRAVITY
- TYPE_LIGHT
- TYPE_ORIENTATION
- TYPE_PRESSURE

The three broad classifications of sensors are as follows:

→ **Motion Sensors:**

These sensors measure acceleration forces along the three axis. This includes accelerometers, gravity sensors, relational vector sensors, and gyroscopes.

→ **Environmental Sensors:**

These sensors are used to measure environmental parameters such as temperature, pressure, humidity and so on.

→ **Position Sensors:**

These sensors measure the physical position of the device which includes orientation changes and magnetometers.

Many of the Android sensors have been introduced from Android version 1.5. The sensor classes and interface are present in the android.hardware package. Some of the classes of Sensor framework are as follows:

- **SensorManager**

The SensorManager class handles the usage of sensors and can be invoked by the method, `Context.getSystemService()`. Code snippet 10 displays the creation of an instance of SensorManager class.

Code Snippet 10:

```
SensorManager sManager=(SensorManager)
getSystemService(Context.SENSOR_SERVICE);
```

- **Sensor**

This class is used to retrieve a list of Sensors available in the devices.

- **SensorEvent**

This class stores information about the sensor type, sensor data, and so on.

Sensor Coordinate System

Sensor coordinate system uses 3-axis coordinate system to read the data values. When the device is in default orientation, X-axis is horizontal, Y-axis is vertical, and Z-axis points towards the outside of the screen, means it is facing to user.

- **SensorEventListener**

Once you acquired a sensor, you can register a SensorEventListener object on it. This listener will get informed, if the sensor data changes.

To avoid the unnecessary usage of battery you register your listener in the `onResume()` method and de-register it in the `onPause()` method.

10.5 Check Your Progress

1. Which of the following option represents the package where Bluetooth classes and interface are present?

(A)	android.bluetooth.app	(C)	android.bluetooth
(B)	android.io.bluetooth	(D)	android.vi.bluetooth

2. _____ represents a Health Device profile proxy for controlling the Bluetooth Service.

(A)	BluetoothAdapter	(C)	BluetoothSocket
(B)	BluetoothDevice	(D)	BluetoothHealth

3. Which of the following option is true for HttpURLConnection?

(A)	Used to connect to wifi	(C)	Used to connect to bluetooth
(B)	Its used to make a single request to the Http Server	(D)	Used to sense the sensor availability

4. Which of the following technology allows devices to share network over a computer network?

(A)	Wi-Fi	(C)	BluetoothSocket
(B)	Sensor	(D)	DeviceManager

5. _____ notifies the application whenever any network changes happen.

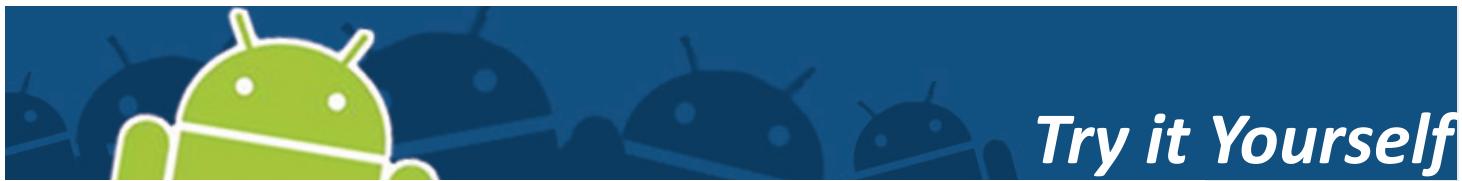
(A)	WifiManager	(C)	ConnectivityManager
(B)	WifiNetwork	(D)	NetworkManager

10.5.1 Answers

1.	C
2.	D
3.	B
4.	A
5.	C

Summary

- Bluetooth is a network stack that transfers data between devices wirelessly.
- Android provides all the Bluetooth API's under `android.bluetooth` package.
- Bluetooth Adapter plays the role of discovering all the Bluetooth enabled devices and querying for request.
- Android applications performing network operations uses HTTP to send and receive data.
- HttpClient can be used to send and receive data from the server by creating a `DefaultHttpClient()` which would help in sending and receiving of the data.
- HttpURLConnection is used to make a single request to the Http Server.
- Android provides Wi-Fi APIs, through which applications can communicate with the wireless stack that provides Wi-Fi network access.
- Sensors can be used to measure motion, orientation, and environmental changes.



Samantha is an enthusiastic learner and now wants to check the bluetooth connection and share a few files present on her device through Bluetooth. You as a developer help her to achieve this?

“

To avoid criticism, do nothing,
say nothing, be nothing

”

JELLYBEAN ICE CREAM SANDWICH

Session - 11

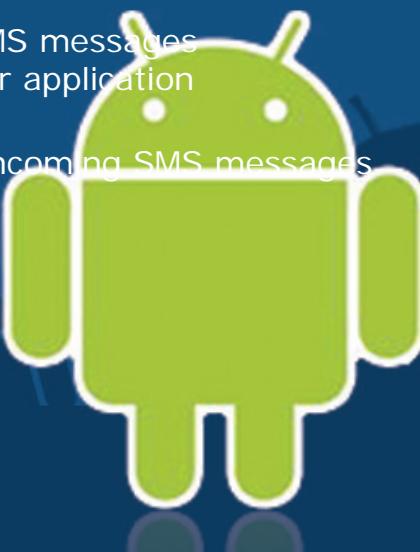
SMS and Telephony

Welcome to the Session, **SMS and Telephony**.

This session introduces the concept of SMS and Telephony where in you would learn how to send and receive SMS and also the key importance of the Telephony.

In this session, you will learn to:

- ➔ Explain Telephony
- ➔ Explain Telephony Manager
- ➔ Explain SMS
- ➔ Explain the process of sending SMS messages programmatically from within your application
- ➔ Explain the process of receiving incoming SMS messages



11.1 Introduction

The quest for developing or inventing something new does not end and increases as days pass. As soon as the basic concept of developing Android application is clear and all applications are running, the developer might think of developing something interesting such as adding the capability of communication with the outside world. A developer might wish to develop an application that will have the capability of sending an SMS message to another phone. In this session, you will be learning how to send and receive SMS messages programmatically from within your Android application.

11.2 Telephony

Telephony provides access to information about the telephony services on the devices. Applications can be using the methods present in this class to determine the telephony services and states, as well as have access to some types of subscriber information. Applications can also register to a listener to receive notification of telephony state changes. Apart from this, the telephony API is also used for activities such as monitoring phone information including the current states of the phone, connections, and network. Android's telephony APIs help to monitor mobile voice and data connections. It helps the applications to access the underlying hardware stack so that the developer can create their own dialer and integrate them within the applications for call handling and phone state monitoring.

11.2.1 Telephony Manager

The `Android TelephonyManager` provides information about the android telephony system. In other words, it provides information about the available telephony services and states on the device. To use `TelephonyManager`, first you need to obtain an instance of the `Telephony Service` by invoking the method `Context .getSystemService(TELEPHONY_SERVICE)`. This because as an instance of this class cannot be directly instantiated. This particular telephony service can be used to retrieve call state, cell location, and operator name.

Access to information in telephony is permission protected and the application should have the required permission specified in the **AndroidManifest.xml** file to access the information.

A change to the state of the phone is monitored using the `PhoneStateListener` class. The listener can listen for, respond to call state, phone service change, and so on. To react to the change in events of the phone state, the `PhoneStateListener` is created and the respective event handler of the event is overridden.

The following example displays the use of monitoring call state such as ringing, off-hook, and so on.

To develop the application, perform the following steps:

1. Start Eclipse.
2. Create a new Android application project named **TelephonyExample**.
3. Navigate to **res → layout** folder
4. Modify the code in **activity_main.xml** file as shown in code snippet 1.

Code Snippet 1:

```
<?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:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Telephony Demo"
        android:textSize="22sp" />

    <TextView
        android:id="@+id/textViewOut"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Telephony Output" >
    </TextView>

</LinearLayout>
```

5. Navigate to **AndroidManifest.xml** file.
6. Modify and add the code under **<uses-sdk>** tag as shown in code snippet 2.

Code Snippet 2:

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

7. Navigate to **src → com.example.telephonyexample** folder.
8. Add the code in **MainActivity.java** file as shown in code snippet 3.

Code Snippet 3:

```
package com.example.telephonyexample;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.content.Context;
import android.telephony.PhoneStateListener;
import android.telephony.TelephonyManager;
import android.widget.TextView;

public class MainActivity extends Activity {

    TextView Tv;
    TelephonyManager telephonyManager;
    PhoneStateListener phoneStateListener;

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

        // Get the UI
        Tv = (TextView) findViewById(R.id.textViewOut);

        // Get the telephony manager
        telephonyManager = (TelephonyManager)
            getSystemService(Context.TELEPHONY_SERVICE);

        // Create a new PhoneStateListener
        phoneStateListener = new PhoneStateListener() {
```

```
@Override  
public void onCallStateChanged(int state, String  
incomingNumber) {  
    String stateString = "N/A";  
    switch(state) {  
        case TelephonyManager.CALL_STATE_IDLE:  
            stateString = "Idle";  
            break;  
        case TelephonyManager.CALL_STATE_OFFHOOK:  
            stateString = "Off State";  
            break;  
        case TelephonyManager.CALL_STATE_RINGING:  
            stateString = "Ringing";  
            break;  
    }  
    Tv.append(String  
            .format("\nonCallStateChanged: %s",  
            stateString));  
}  
};  
// Register the listener with the telephony manager  
telephonyManager.listen(phoneStateListener,  
    PhoneStateListener.LISTEN_CALL_STATE);  
}  
}
```

Figure 11.1 displays the output in the emulator when the application is executed.

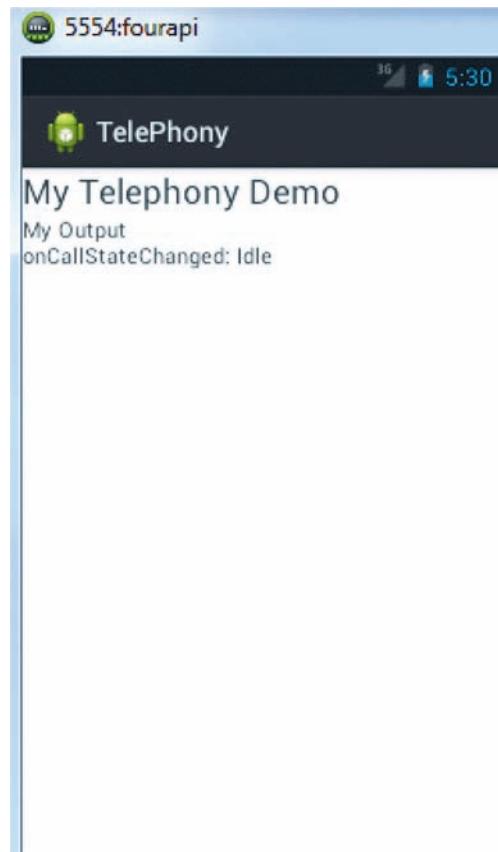


Figure 11.1: TelephonyExample – Output

11.3 SMS

SMS stands for Short messaging service. SMS messaging is one of the most important and frequently used application that is executed on mobile devices. Today, any mobile phone that users buy have SMS messaging capability and nearly all the mobile users of any age group knows how to send and receive messages. Android comes with the built-in SMS application that enables you to send and receive SMS messages.

This section explains how developers can programmatically send and receive SMS messages in an Android application. The good news for the Android developers is that you do not need a real time device to test the SMS messaging as the emulator is more than enough for testing since it has that capability built-in.

The `SmsManager` manages the SMS operations, such as sending text and data messages. Similarly, `ConnectivityManager` manages the Internet connectivity in a mobile phone. This class supervises network connections and also deals with sending broadcast intents in case network connectivity changes.

11.3.1 Sending SMS Messages from Application

The application can send SMS to another phone, when an event takes place. In an Android application, the developer can access the SMS application using the `SmsManager`. To obtain the default instance of `SmsManager` class, the developer needs to invoke the `SmsManager.getDefault()` method. The class cannot be directly instantiated, unlike the other classes. The `sendTextMessage()` function is used to send an SMS message.

The syntax of `sendTextMessage()` method is as follows:

Syntax:

```
public void sendTextMessage (String destinationAddress, String
scAddress, String text, PendingIntent sentIntent, PendingIntent
deliveryIntent)
```

where,

`destinationAddress` – denotes the recipient address.

`scAddress` – denotes the service center address.

`text` – represents the body of the message to be sent.

`sentIntent` – broadcasted when the message is sent successfully.

`deliveryIntent` – broadcasted when the message is delivered successfully.

Code snippet 4 displays the use of `SmsManager` class.

Code Snippet 4:

```
...
SmsManager smsMgr = SmsManager.getDefault();
smsMgr.sendTextMessage("phone_num", null, "msg_text", null, null);
...
```

To develop an application that sends an sms, perform the following steps:

1. Start **Eclipse**.
2. Create a project named **SMSApp**.
3. Navigate to **res → layout** folder.
4. Modify the code in **activity_main.xml** file as shown in code snippet 5.

Code Snippet 5:

```
<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/buttonSendSMS"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Send SMS" />

</LinearLayout>
```

5. Navigate to **AndroidManifest.xml** file and add the code under `<uses-sdk>` tag as shown in code snippet 6.

Code Snippet 6:

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

6. Navigate to **src → com.example.smsapp** folder.
7. Modify the code in **MainActivity.java** file as shown in code snippet 7.

Code Snippet 7:

```
package com.example.smsapp;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.telephony.SmsManager;
```

```
public class MainActivity extends Activity {  
  
    Button button;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        button = (Button) findViewById(R.id.buttonSendSMS);  
        button.setOnClickListener(new View.OnClickListener() {  
  
            public void onClick(View v) {  
  
                sendSMS("5556", "HelloWorld!!");  
            }  
            private void sendSMS(String phoneNumber, String message) {  
                SmsManager smsmanager = SmsManager.getDefault();  
                smsmanager.sendTextMessage(phoneNumber,  
                    null, message, null,  
                    null);  
            }  
        } );  
    }  
  
    @Override  
    public boolean onCreateOptionsMenu(Menu menu) {  
        // Inflate the menu; this adds items to the actionbar if it  
        // is present.  
        getMenuInflater().inflate(R.menu.main, menu);  
        return true;  
    }  
}
```

In the code `sendSMS("5556", "Hello World!!")` method has been used. In this method the number of the emulator to which the developer is sending the SMS message along with the message has been passed as an argument. Every emulator is identified by a number which appears on top left corner.

Launch two different emulator devices and notice the emulator numbers appearing on top left corner. For example, consider you have 2 emulators named emulator A and emulator B. The application will be executed on one of the emulator for example on emulator A. Then in the application code the developer needs to write emulator B's number. Then once you execute the application in your emulator A and click **Send SMS**, emulator B's notification bar will display a message, **Hello world** which has been received from emulator A.

Figure 11.2 displays the output when the application is executed.

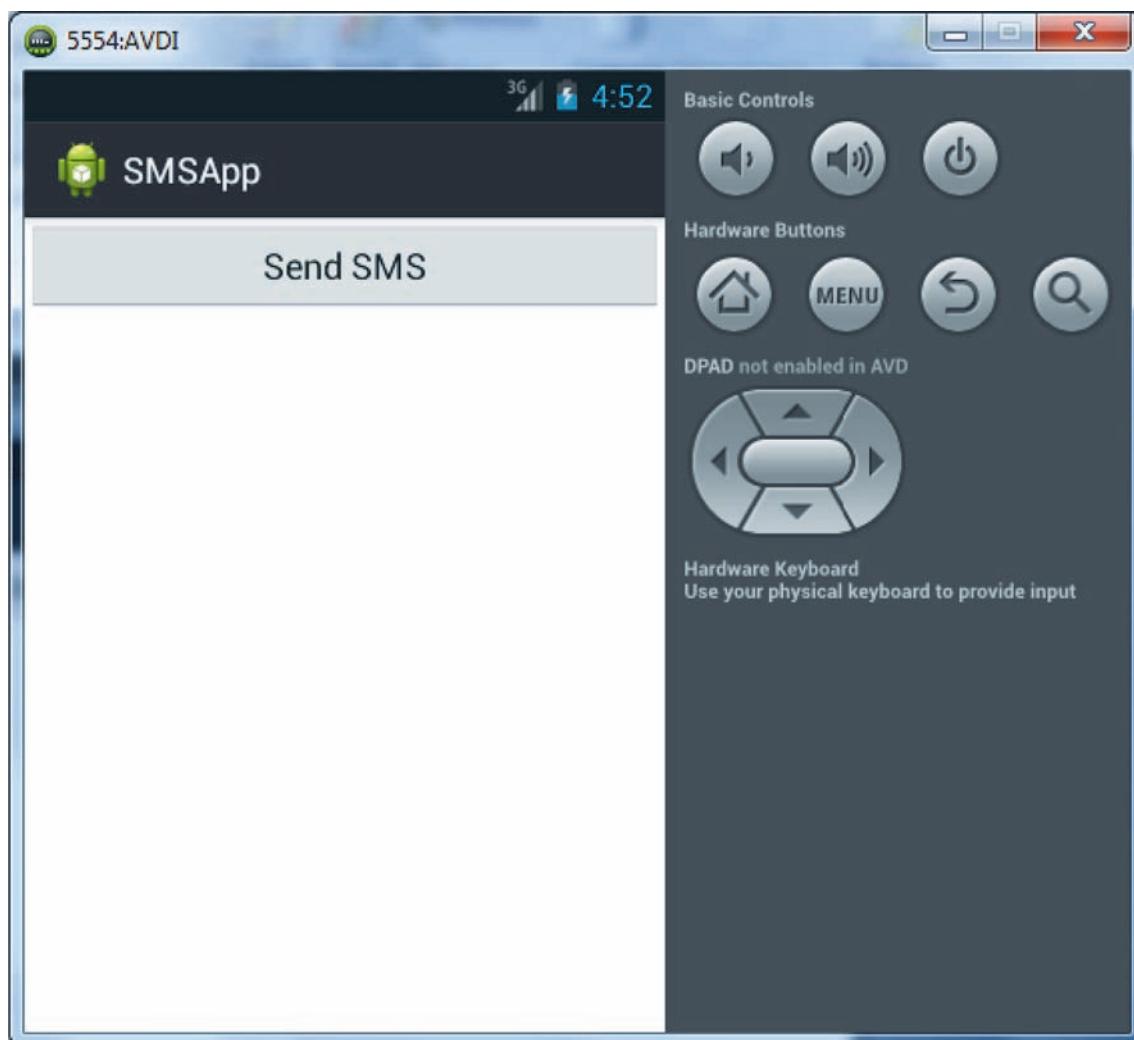


Figure 11.2: SMSApp Application – Output

Figure 11.3 displays the output in the second emulator when the **Send SMS** is clicked.

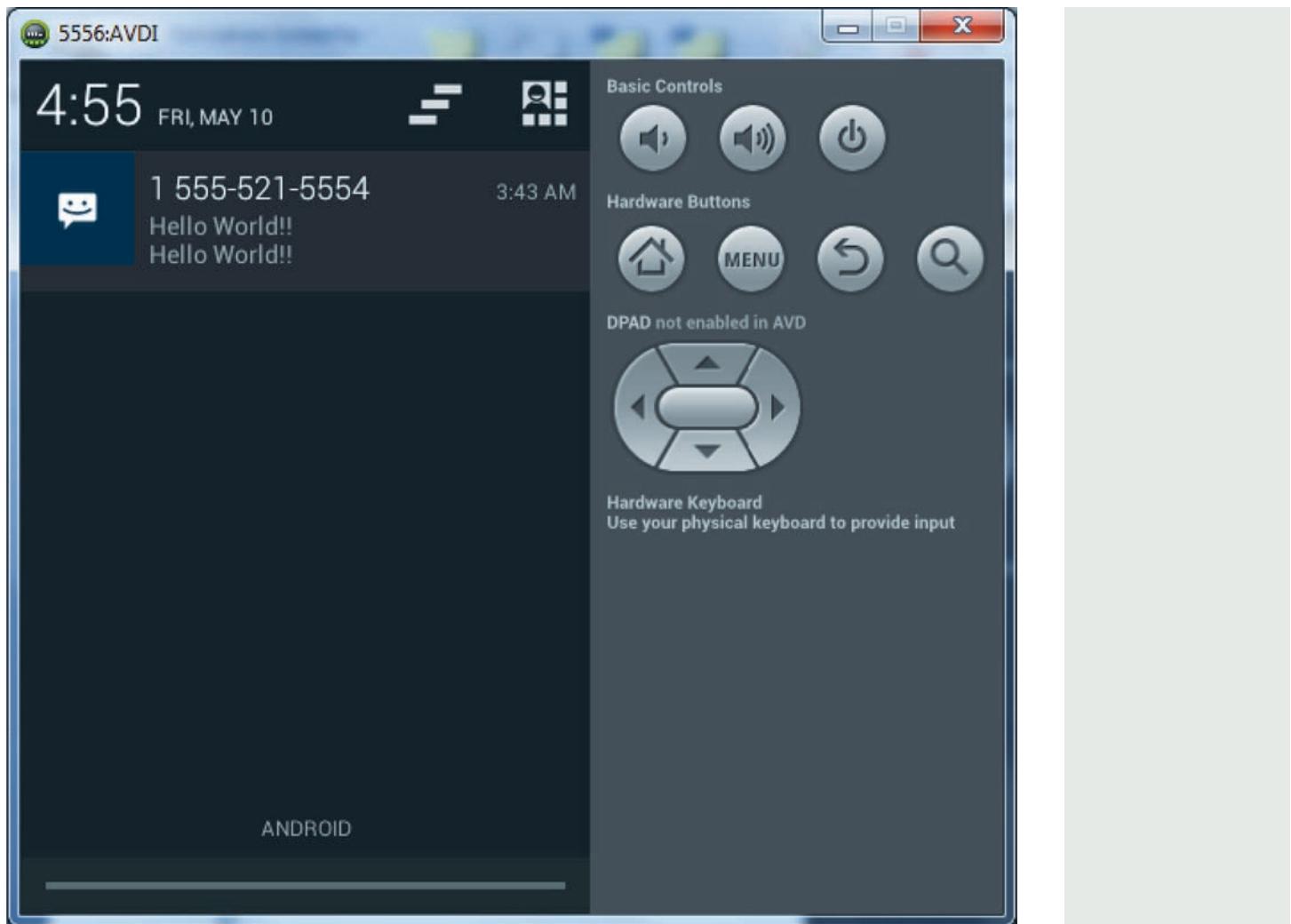


Figure 11.3: Output in the Second Emulator

In this application the message which has been written in the code is delivered if and only if the developer specifies the exact emulator number to which the message is being sent. If the application is tried in the AVD (android virtual device) emulator then the SMS size is very restricted and it should not be more than 160 characters.

11.3.2 Receiving Incoming SMS Messages

Apart from sending SMS messages from within your Android applications, the developer can also receive incoming SMS messages from within the application using BroadCastReceiver object. This is of help when the application wants to perform an action on receipt of SMS.

To receive an SMS, perform the following steps:

1. Start **Eclipse**.
2. Create a project named **SMSReceiverApp**.

3. Navigate to **res** → **layout** folder.
4. Modify the code in **activity_main.xml** file as shown in code snippet 8.

Code Snippet 8:

```
<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/buttonSendSMS"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Receive SMS" />

</LinearLayout>
```

5. Navigate to project's **AndroidManifest.xml** file and add the code under **<uses-sdk>** tag as shown in code snippet 9.

Code Snippet 9:

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

6. Add the following code before the **</application>** tag as shown in code snippet 10.

Code Snippet 10:

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

7. Navigate to **src → com.example.smsreceiverapp** folder.
8. Modify the code in **MainActivity.java** file as shown in code snippet 11.

Code Snippet 11:

```
package com.example.smsreceiverapp;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.telephony.SmsManager;

public class MainActivity extends Activity {

    Button button;

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

        button = (Button) findViewById(R.id.buttonSendSMS);
        button.setOnClickListener(new View.OnClickListener() {

            public void onClick(View v) {
                // Here write the emulator's number to which
                // your sending the
                // SMS
                sendSMS("5556", "Hello World!!");
            }
        });
    }
}
```

```

private void sendSMS(String phoneNumber, String
message)

{
    SmsManager smsmanager =
    SmsManager.getDefault();

    smsmanager.sendTextMessage(phoneNumber,
    null, message, null,
    null);

}

}

@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;
}

}

```

9. Create a new class named **SMSReceiver** in **src → com.example.smsreceiverapp** folder.
10. Add the code as shown in code snippet 12.

Code Snippet 12:

```

package com.example.smsreceiverapp;

import android.os.Bundle;
import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.telephony.SmsMessage;

```

```
import android.view.Menu;
import android.widget.Toast;

public class SMSReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Bundle bundle = intent.getExtras(); // --get the SMS
        message passed in
        SmsMessage[] msgs = null;
        String str = " ";

        if (bundle != null) {
            // --retrieve the SMS message received--
            Object[] obj = (Object[]) bundle.get("obj");
            msgs = new SmsMessage[obj.length];
            for (int i = 0; i < msgs.length; i++) {
                msgs[i] = SmsMessage.createFromPdu((byte[])
                obj[i]);
                str += "SMS from " + msgs[i].
                getOriginatingAddress();
                str += ":";
                str += msgs[i].getMessageBody().toString();
                str += "\n";
            }
            // ---display the new SMS message
            Toast.makeText(context, str, Toast.LENGTH_
            SHORT).show();
        }
    }
}
```

11. Start an emulator from **Windows → Android Virtual Device Manager**.
12. Select the **DDMS** perspective and check the running emulator.
13. Select that emulator from the **Device** pane on the left.
14. Select the **Emulator Control** tab from the right pane.
15. Select **SMS**.
16. Type the number, **12980097** in the **Incoming number** box. It indicates the ddms number.
17. Type the message **Hello SMS Receiver** in the **Message** box.
18. Click **Send** as shown in figure 11.4.

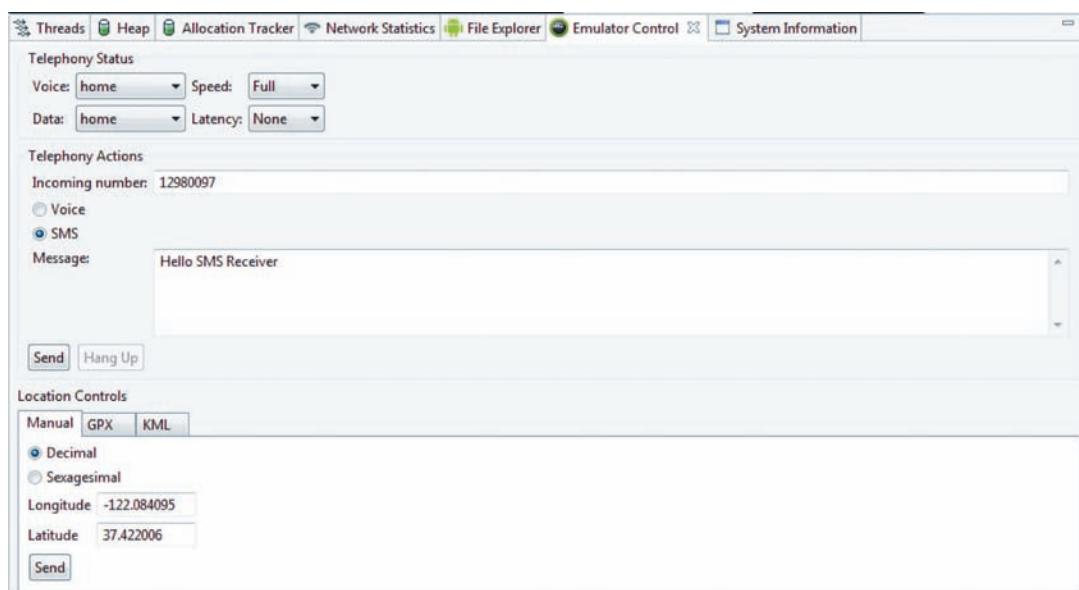


Figure 11.4: Setting the SMS Message

Switch over to Java perspective and then execute the application in the respective emulator and you will receive a message from the ddms in the emulator's notification bar. After executing the application the output will be as shown in figure 11.5.

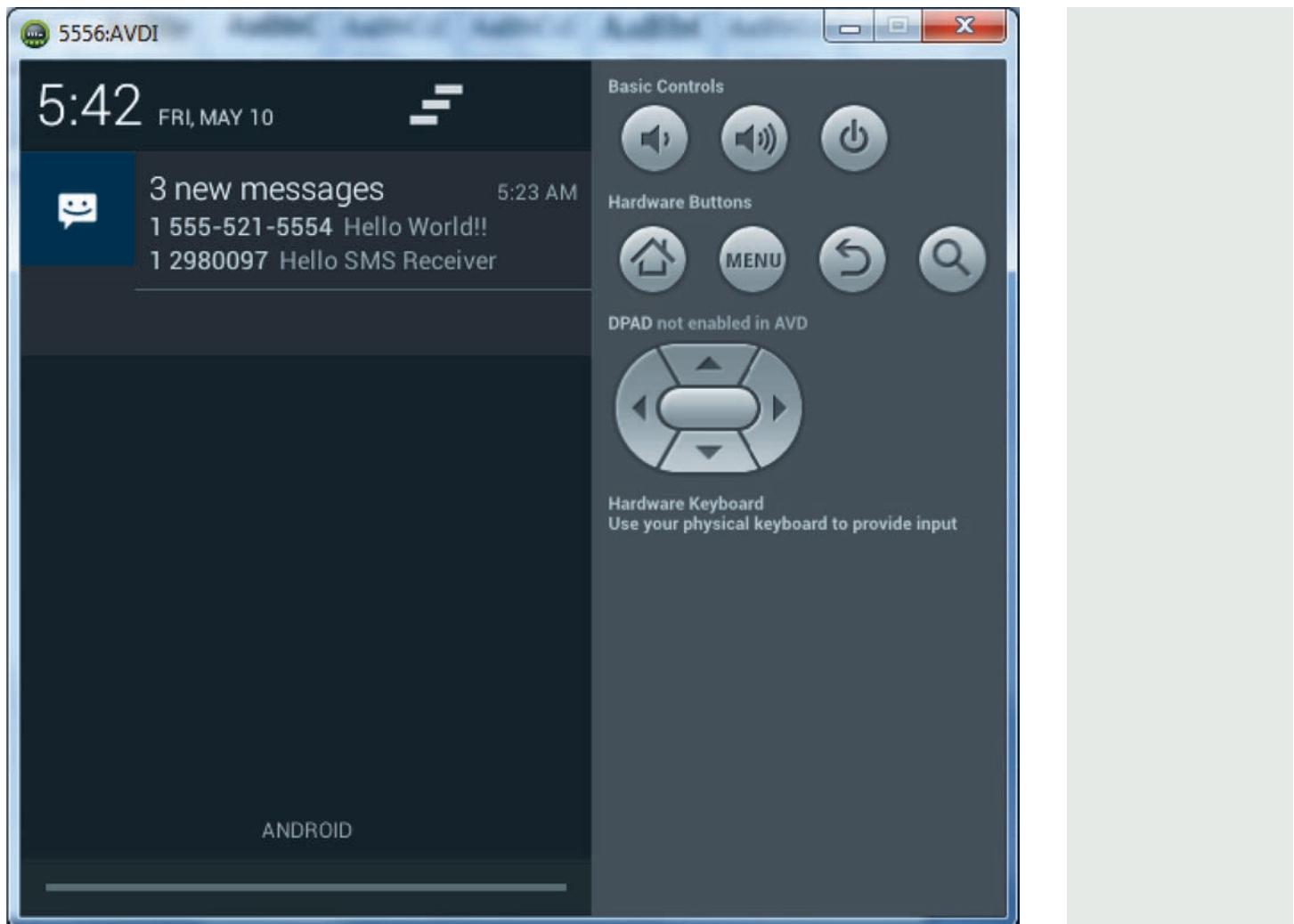


Figure 11.5: SMS Received

11.3.3 Sending SMS and Tracking It

There are several ways in which you can track SMS. Some of these are as follows:

- For tracking SMS you have to use broadcast receiver which checks the status of the message. You have to develop your own `BroadcastReceiver` and pass the string using an object of the `PendingIntent` class.
- You can also use `ContentObserver` listening on the content://sms to track SMS. This class will invoke the `onChange()` method when the sms database changed.

11.4 Check Your Progress

1. The telephony API is also used for other activities such as monitor phone information including the current states of the phone, connections, and _____.

(A)	SME	(C)	SMS
(B)	MME	(D)	Network

2. Which of the following method should be invoked by the TelephonyManager to obtain an instance of the Telephony Service?

(A)	Context. getSystemService(TELEPHONY_SERVICE)	(C)	Context. getSystemService(TELEPHONY_SERVICE)
(B)	getSystemService(TELEPHONY_SERVICE)	(D)	getSystemService(TELEPHONY_SERVICE)

3. SMS stands for _____.

(A)	Short Messaging Services	(C)	Short Message System
(B)	Short Multimedia Service	(D)	Short Multimedia Messaging Service

4. Which of the following permission needs to be granted in the Android Manifest file to send an SMS?

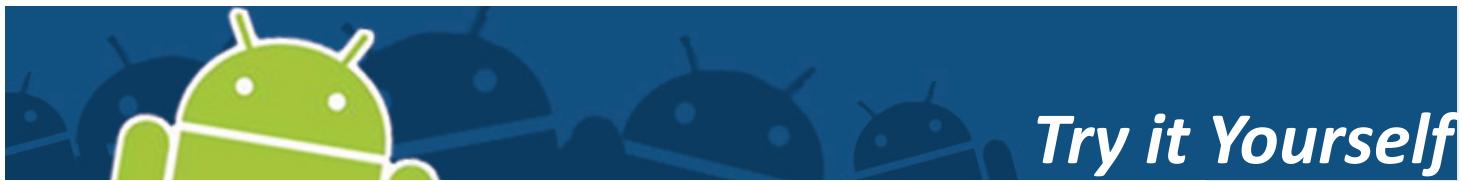
(A)	<uses-permission android:name="android. permission.SEND_SMS"/>	(C)	<uses-permission android:name="android. permission.RECEIVE_SMS"/>
(B)	<uses-permission android:name="android. permission.GO_SMS"/>	(D)	<uses-permission android:name="android. permission.PENDING_SMS"/>

11.4.1 Answers

1.	D
2.	A
3.	A
4.	A



- Telephony provides access to information about the telephony services on the devices.
- The Android TelephonyManager provides information about the android telephony system. In other words, it provides information about the available telephony services and states on the device.
- SMS stands for Short Messaging Service. SMS messaging is one of the most important and frequently used applications that is executed on the mobile devices.
- The SmsManager manages the SMS operations, such as sending text and data messages. Similarly, ConnectivityManager manages the Internet connectivity in a mobile phone.
- Apart from sending SMS messages from your Android applications, the developer can also receive incoming SMS messages from within the application using BroadCastReceiver object.



Samantha who has learnt the basics of Android programming wants to develop an application where user will be able to send and receive message.

“

Democritus said, words are
but the shadows of actions

”

JELLYBEAN
ICE CREAM SANDWICH

Session - 12

Development for Android Market

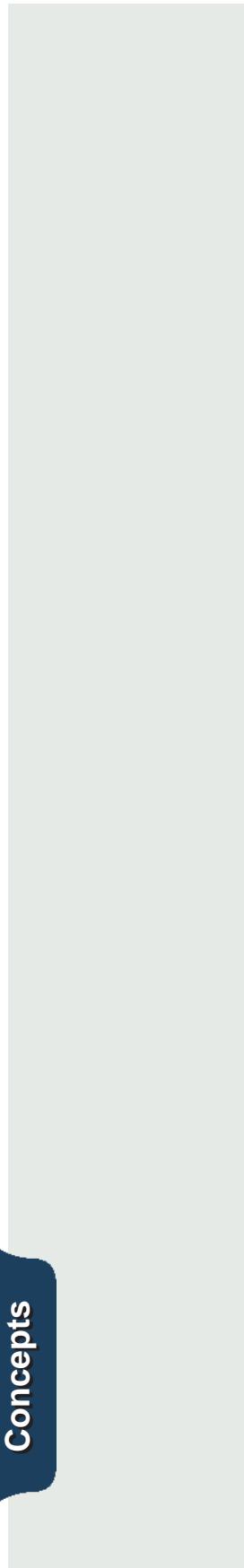
Welcome to the session, **Development for Android Market**.

This session introduces the developers to Android market. It also walks the developer through the basic requirement needed for the Android app development and publishing of app in the android market.

In this session, you will learn to:

- ➔ Explain the requirement for Android market
- ➔ Explain the different versions and feature set
- ➔ Explain the share of android in the market
- ➔ Explain different kinds of devices available in the market
- ➔ Describes the making of an .apk file
- ➔ Explain the process of publishing the .apk file in Android market





12.1 Introduction

The Android market or the Android central is known to be the very own repository of Google for Android applications. The Android market offers lots of applications for Android devices or the handsets (devices those have android operating system). There are many categories of apps available in Android market such as education, entertainment, and so on. These apps can be free or paid. The free apps available on the Android market can be downloaded and installed in your android phones and used by the user immediately. This session discusses about the process of publishing android applications in the Android market.

12.2 What is Android Market?

Android Market is recently renamed as Google play which is developed and maintained by Google. It is an online electronic store or digital application distribution platform for android powered devices. It is a robust publishing platform. The services offered by PlayStore (formerly known as Android Market) to the users includes downloading free and paid applications, books, movies, music, magazines, and so on. Users can also purchase mobile devices like Google-Nexus, Chromebook through playstore. The initial release of android market was on October 23, 2008. At the initial beta release android market provided only free applications. And later on by February 2009 Google implemented the purchase system, which enabled the users to download applications only if the user made the payment while downloading the app from the android market. At that time Android market contained only 2300 applications in it.

The advantage of distributing applications over the Android Market is tremendous, since, the developer has access to global users. Also, the developer will have access to revenue generating tools, such as in-app-billing and application-licensing. The developer will also know the sale trends and understand the end users.

12.3 Application

Before uploading the application in android market, the basic points to be noted are as follows:

1. Testing the application.
2. Checking the application performance.
3. SDK Compatibility.

1. Testing the application:

The application developed by any developer has to be tested well before uploading it to the Android market so that the application that is uploaded is not only error and bug free

but also does not crash at users end. If these conditions are fulfilled then only the number of users downloading the app will increase.

The output of the application when tested on an emulator may vary when tested on real device. For this reason, the application is required to be tested on real devices of varying types before being uploaded to the market for the users to download. The Android devices are manufactured by various manufacturers with different screen sizes.

The screen sizes are divided into 4 different types. The standard screen sizes suggested by the Android specifications are as follows:

- Small Screens having a size of 426 dp by 320 dp.
- Normal Screens having a size of 470 dp by 320 dp.
- Large Screens having a size of 640 dp by 480 dp.
- Extra large Screens having a size of 960 dp by 720 dp.

Note - dp stands for dots per pixel.

To augment the support for different screen sizes, Android provides different drawable folder for each application to place different resolution of images. Android will choose the images from the respective folder depending on the screen size.

To prepare your application for releasing the developer typically needs to perform five tasks which are as follows:

- a. **Collect Material for Release:** This includes preparation of End User License Agreement (EULA) for the application. This will help to protect not only the developer and the organization but also the intellectual property rights. It also includes the process of obtaining an encrypted key for digitally signing the application and creation of an icon for the application.
- b. **Arrange Application for Resource:** This includes gathering of all the materials required for configuring the applications which includes configuration changes made to the source code, resource files, and the manifest file of the application. Besides this the developer has to clean up the project, update the manifest file setting, update the URLs for servers and services, and so on.
- c. **Build Application for Resource:** This includes signing the application and building the application for release.
- d. **Prepare Remote Server:** This includes the process of ensuring that if a remote server

is used it is secure and configured for use.

- e. **Test Application for Resource:** This includes testing the application to ensure that the application works properly under real device and varying network conditions.

2. Application performance:

The performance of the application is one of the most important concerns to the developer. Depending on the performance of the application, the number of downloads will vary. While developing the gaming applications or interactive application with server, the code has to be optimized so as to know the response rate of the application.

3. SDK Compatibility:

There are many android powered devices manufactured by various manufacturers which uses different android versions on their devices ranging from version as low as 2.3 to highest version of 4.2. For this, Android's aim is to provide an open platform for developers to build creative apps. The apps thus need to be SDK compatible for the following reasons:

1. **Requirement for Customizable Device by the User:** Mobile phone is a gateway to Internet and users want to customize it by adding functionality. Due to this requirement, Android was designed as a robust platform for executing after-market application.
2. **Large Requirement for Developer:** Users has different needs and requirements and no device manufacturer can cater to all these user requirements. Thus, third party developers are required for developing user specific apps. To meet all these requirements, Android open source project aims to make it as easy and open as possible for developer to builds apps.
3. **Requirement for a Common Ecosystem:** The more compatible the phones are the more number of applications will be developed. Due to this, Android provides an open source and allows a third party to integrate their app as well as into the system.

In 2013, the number of devices executing a particular version of Android version is as follows:

- ➔ Version 2.3 is having 38.4% of distribution shares in market.
- ➔ Version 4.0 is having 27.4% of distribution shares in market.
- ➔ Version 4.1 is having 26.1% of distribution shares in market.
- ➔ Version 4.2 is having 2.3% of distribution shares in market.

Thus, it can be summarized that 94.2% devices are powered by 2.3+ versions and 5.8% devices are powered by 2.2 and lower versions. Depending on the mentioned share percentage, the developer decides for which version the developed application will be targeted for.

Table 12.1 displays the distribution percentage of the different versions available in the Android market.

Code Name	Version	Distribution %
Donut	1.6	0.1
Eclair	2.1	1.7
Froyo	2.2	3.7
GingerBread	2.3–2.3.7	38.5
Honeycomb	3.0–3.2	0.1
Ice Cream Sandwich	4.0–4.0.4	27.5
JellyBean	4.1–4.2	28.4

Table 12.1: Market Share of Different Platforms

The following are some of the tasks that needs to be performed before uploading the package.

1. Set permissions.
2. Establish version, set icon, and application label.
3. Set compatibility options.
4. Remove log data.
5. Export the project and create the key using Eclipse.

1. Permissions:

The following details in the **AndroidManifest.xml** file should be set correctly:

- <uses-permission> element

In this tag, the developer should specify only those permissions that are relevant and required for executing the application.

- ➔ android:icon and android:label attributes

In the `<application>` element the developer should specify values for these attributes as these are displayed to the user.

- ➔ android:versionCode and android:versionName attributes

In the `<manifest>` element the developer should specify values for these attributes.

There are several additional manifest elements that can be set by the developer if the application is being released on Google Play. For example, the `android:minSdkVersion` and `android:targetSdkVersion` attributes, which are located in the `<uses-sdk>` element. Code snippet 1 demonstrates the use of some of these attributes.

Code Snippet 1:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

2. Version:

The version names have to be specified in the **AndroidManifest** file that is written within the `<manifest>` tag. Code snippet 2 demonstrates the use of the tag.

Code Snippet 2:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.helloandroid"
    android:versionCode="1"
    android:versionName="1.0" >
```

In the code snippet, the version code starts from 1 and is required to be incremented by 1 for each upload. Version name specifies the release version and always should be ascending order.

3. Specifying Icon, Application Label, and SDK Version:

The icon to be set for the application and the name of the application has to be specified inside the `<application>` tag in **AndroidManifest** file as shown in code snippet 3.

Code Snippet 3:

```

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.helloworld.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>

```

Code snippet 4 demonstrates the process of specifying the SDK version.

Code snippet 4:

```

<uses-sdk android:minSdkVersion="8"
    android:targetSdkVersion="10"
    android:maxSdkVersion="16" />

```

The minSdkVersion, specifies the minimum sdk version the application will support.

The targetSdkVersion, specifies the version to which the application is built for.

The maxSdkVersion, specifies the maximum sdk version the application can support.

4. Compatibility Options:

The Android system will decide on which platform the application will be installed depending on the target specified in the manifest file. There is an android compatibility program which works for the benefits of the users, developers, and device manufacturers. The aim of the compatibility program is to provide the following features:

- It provides a consistent application and hardware environment to application developers.
- It enables a consistent application experience for consumers.

- It enables device manufacturers to differentiate while being compatible.
- It minimizes costs and overhead associated with compatibility.

The Android compatibility program consists of three key components. They are as follows:

- The source code to the Android software stack.
- The Compatibility Definition Document (CDD) which specifies the ‘policy’ aspect of compatibility.
- The Compatibility Test Suite (CTS) which specifies the ‘mechanism’ of compatibility.

Each version of the Android platform exists as a separate branch in the source code tree and there is a separate CTS and CDD defined for each version. You need to create a compatible device according to the specifications present in the CDD, CTS, and source code along with the specified hardware and software customizations.

5. Remove Log data:

Remove the calls to logs, as it will display output data in LogCat always. This will not only increase the size of the log file but also increase the time required to be spent on to print huge data each time. Unnecessary resource files are required to be removed as it will lead to increase in size of the application.

6. Exporting project and key creation using Eclipse:

Before exporting a project a key needs to be created. To create a key you have to use Eclipse. The process of exporting package in Eclipse is as follows:

1. Right-click project to display the context menu.
2. Select **Export** to display the **Export** dialog box.
3. Expand **Android → Export Android Application** as shown in figure 12.1.

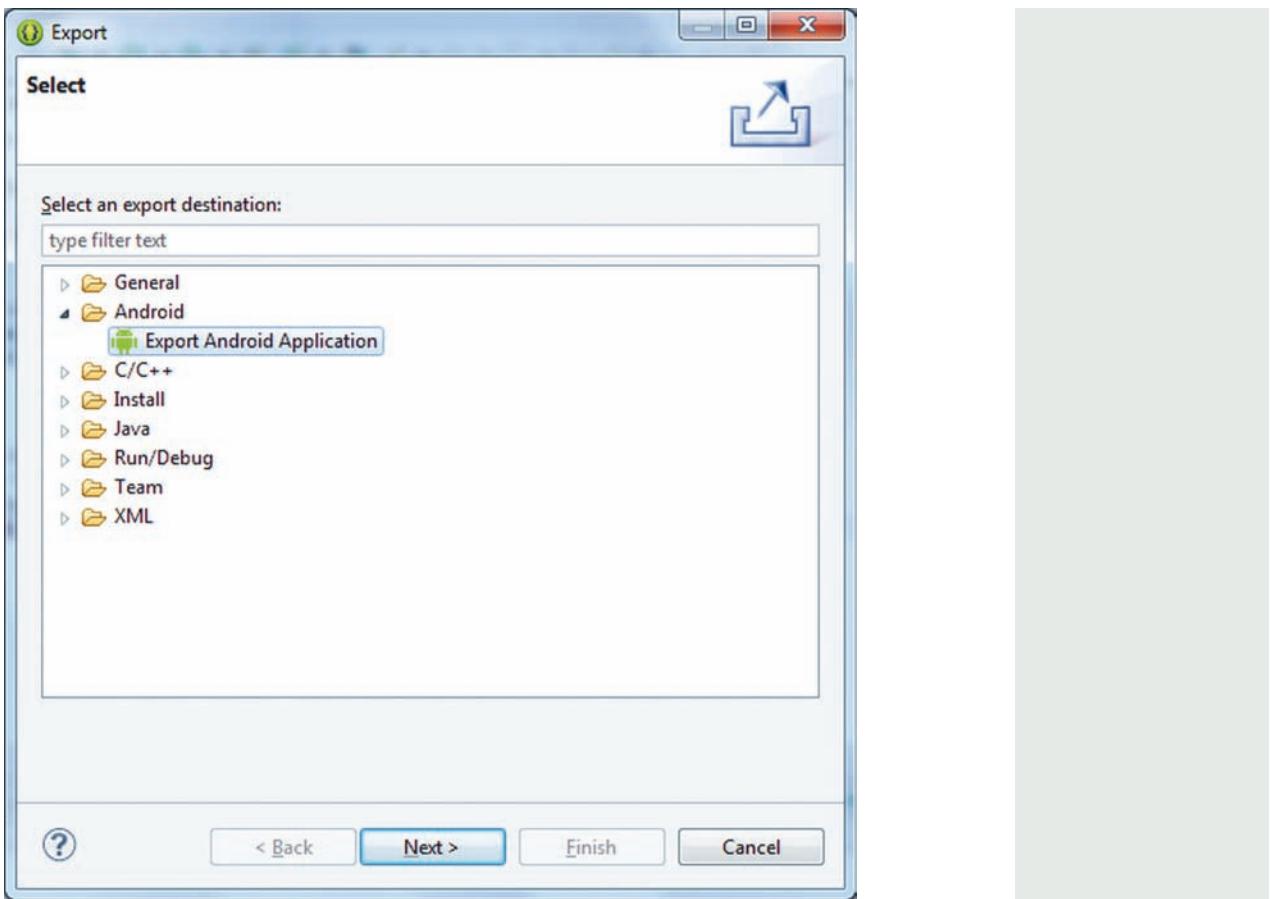


Figure 12.1: Export Dialog Box

4. Click **Next** to display the **Project Checks** pane of the **Export Android Application** dialog box as shown in figure 12.2. This will help the developer to choose the project to be exported.

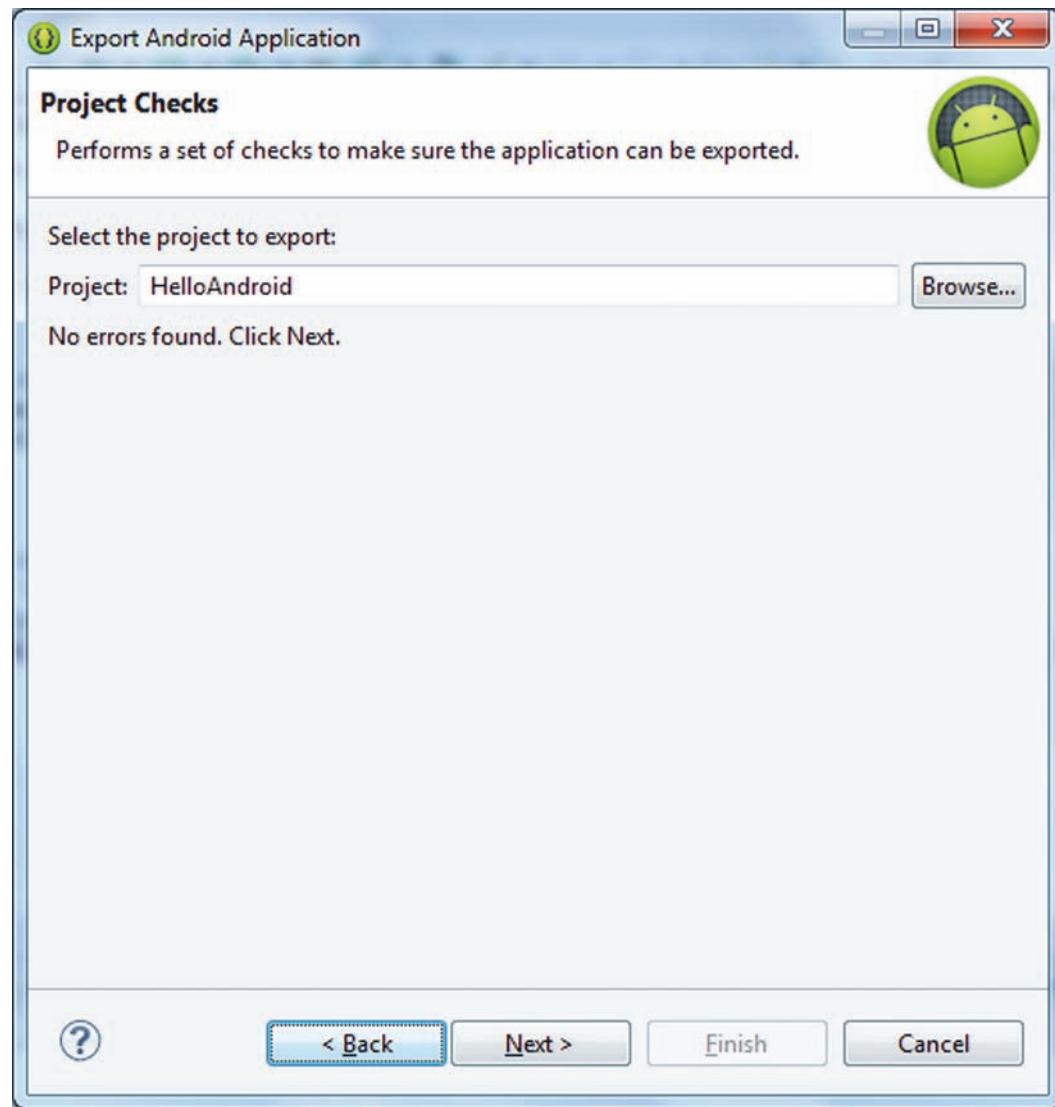
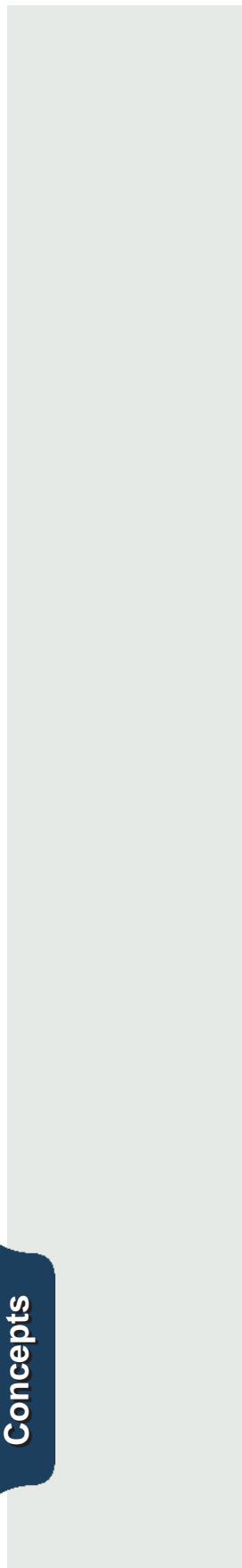


Figure 12.2: Export Android Application

5. Click **Next** to display the **Keystore selection** pane of **Export Android Application** dialog box.

6. Select **Create new keystore**. **Create new keystore** is selected if the developer is uploading the application for the first time. The developer can select **Use existing keystore**, if the application has been already uploaded to Playstore.
7. Browse and select the location in the **Location** box and type the password in the **Password** and **Confirm** box as shown in figure 12.3.



Figure 12.3: Keystore Selection Pane

8. Click **Next** to display the **Key Creation** pane of the **Export Android Application** dialog box.
9. The **Key Creation** pane will appear and fill in the details as shown in figure 12.4.



Figure 12.4: Key Creation Pane

Alias: Provide an alias name to the application.

Password/Confirm Password: Enter the password and confirm the same.

Validity (years): The validation certificate is created and it has to be valid for minimum of 50 years. Otherwise, the application will be rejected by the Playstore.

First and Last Name: Enter the first and last name (as preferred) and the other details such as Organizational unit, Organization, City or Locality, State or province, and Country code which are optional.

10. Click **Next** to display the **Destination and key/certificate checks** pane of **Export Android Application** dialog box.

11. Browse and select the destination of the .apk file as shown in figure 12.5.

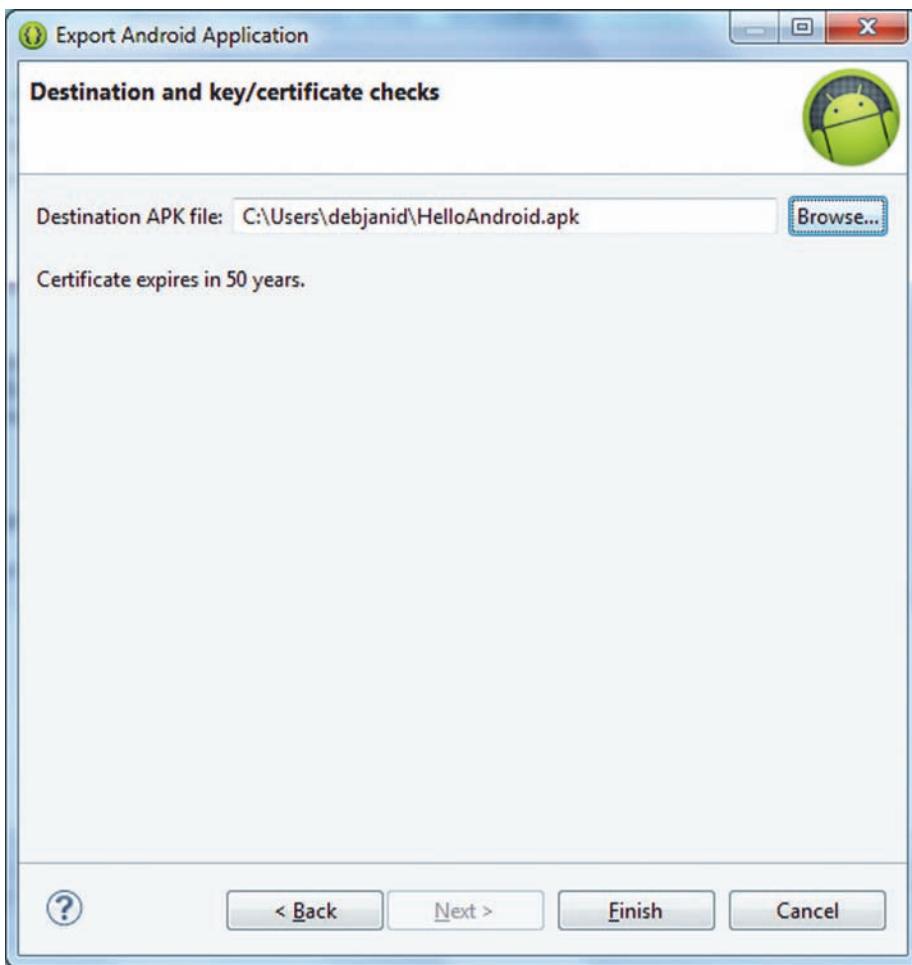


Figure 12.5: Destination and key/certificate checks Pane

Note - The application will expire in 50 years as validity specified as 50 years.

12. Click **Finish**. Eclipse will generate a new **HelloAndroid.apk** file in the specified target directory. This .apk file can be uploaded to Playstore.

12.4 Publishing Android Application to Playstore

The steps for publishing an android application to Playstore are as follows:

1. Create a signed .apk file.
2. Sign in to the playstore account.
3. Add a new application.

4. Store Listing.
5. Upload the application screenshots.
6. Categorize details.
7. Enter contact information.
8. Upload the .apk file.
9. Pricing and distribution.
10. Publish the application.

The section explains in details.

1. Create a signed .apk File

As discussed, the developer needs to create an **.apk** file with 50 years validity. The playstore will only allow application which is at least valid for 50 years.

2. Sign in to the playstore account

The developer needs to register for a publisher account by visiting the **Google Play Developer** console at <https://play.google.com/apps/publish/>. The basic details of the developer such as name, email address, and so on are accepted. The developer distribution agreement related to the developer's country is accepted. If the developer is registering for the first time then US\$25 is paid as registration fee using Google wallet.

The **Sign in** page for Android Market is as shown in figure 12.6.

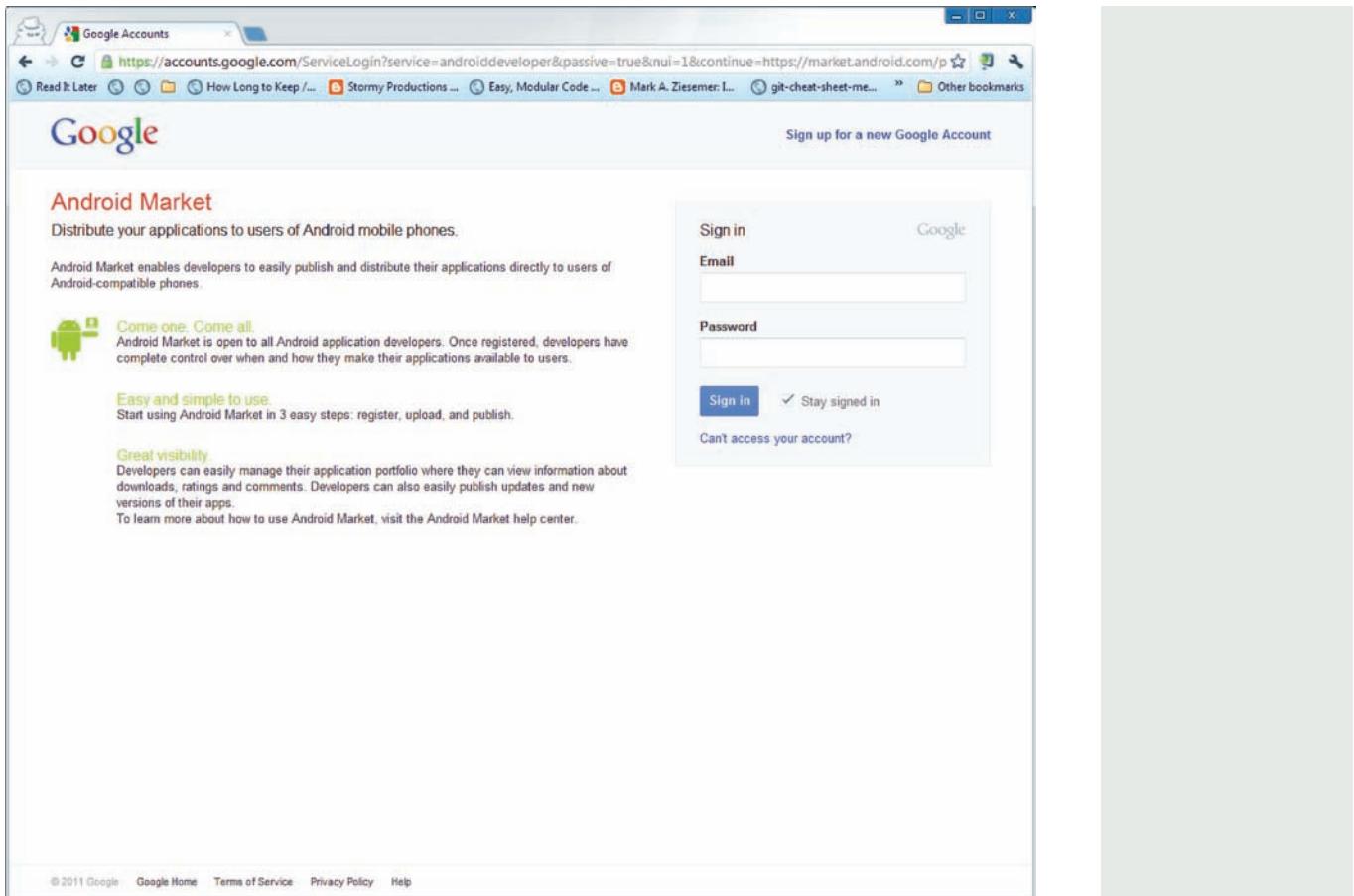


Figure 12.6: Sign in Page

3. Add a New Application

Add your application to playstore by clicking **Add new application** as shown in figure 12.7.

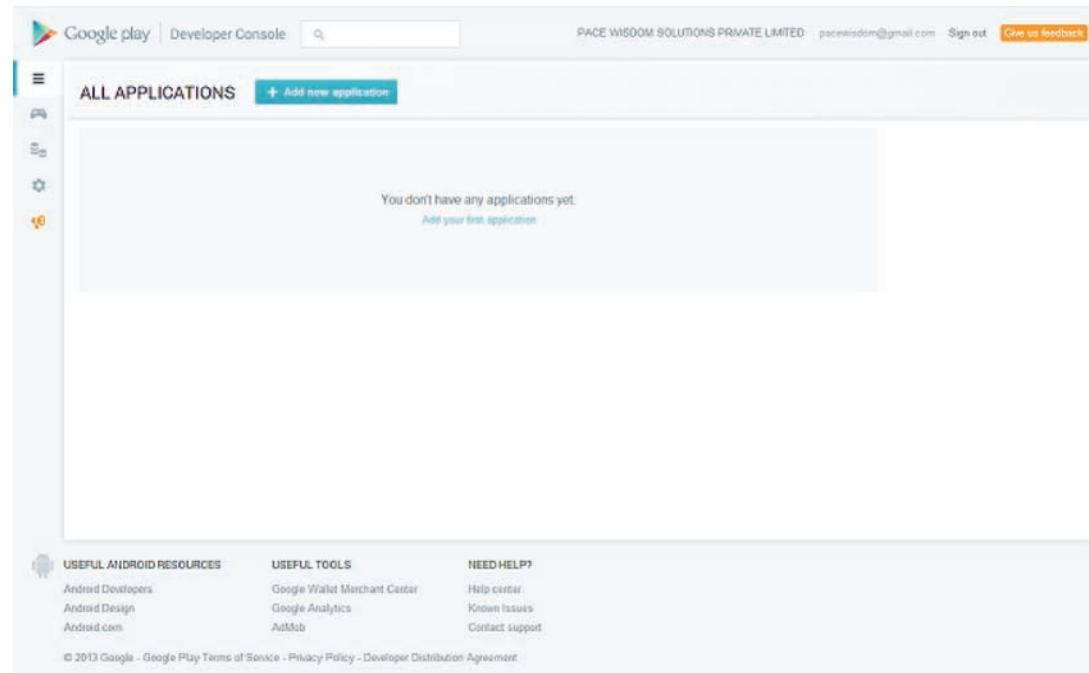


Figure 12.7: Adding a New Application

Clicking **Add new application** displays the **ADD NEW APPLICATION** screen as shown in figure 12.8.

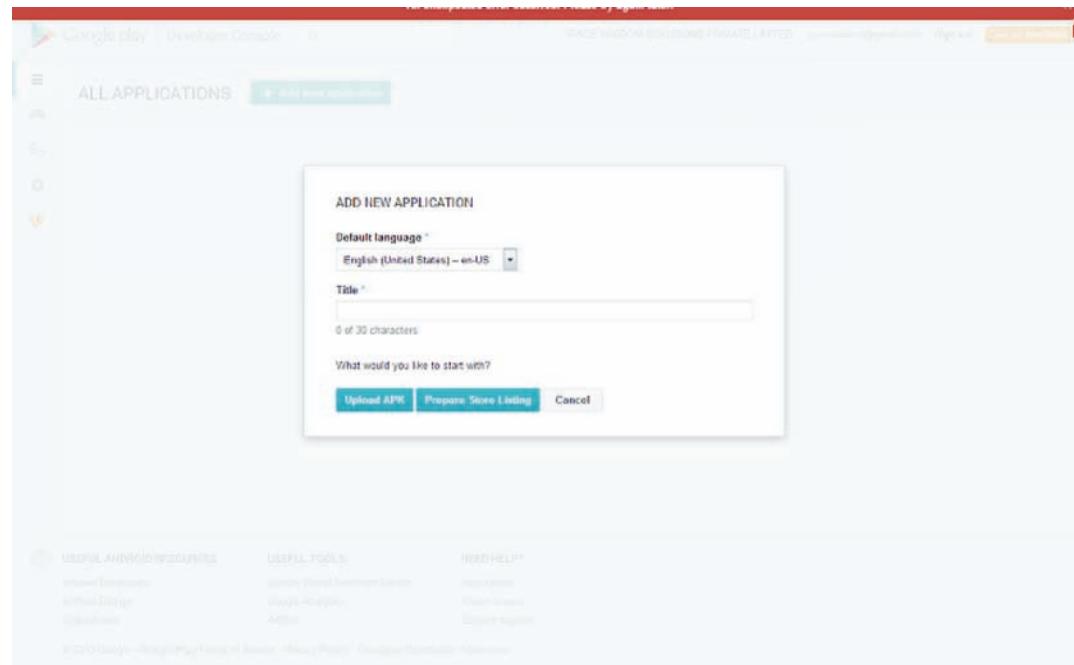


Figure 12.8: ADD NEW APPLICATION

Following activities are performed:

- Language is selected from the **Default language** list as shown in figure 12.9.

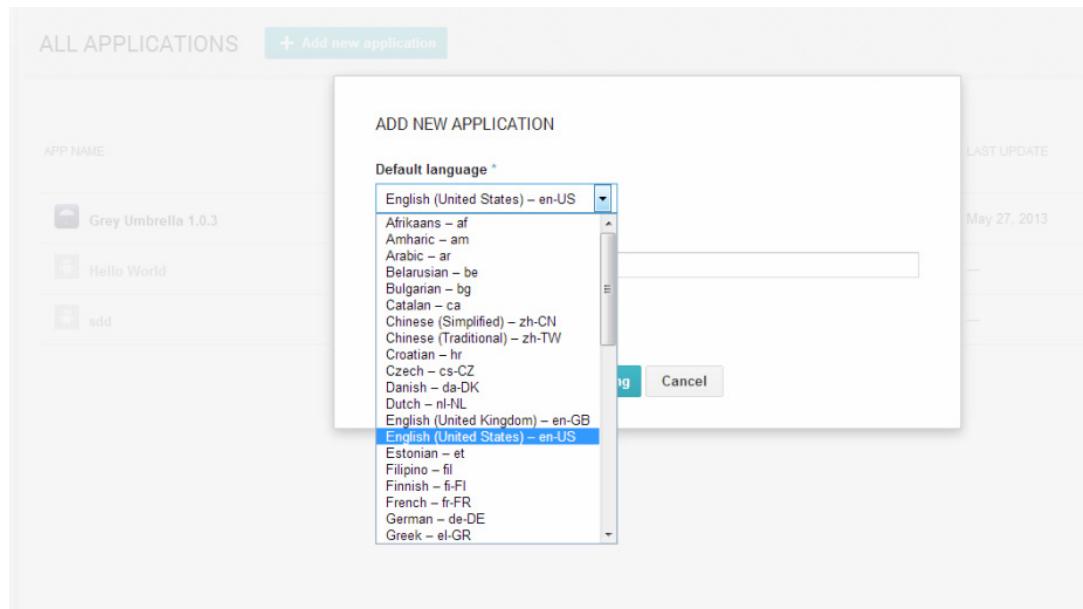


Figure 12.9: Language Selection

- Title of the application is specified in the **Title** box.
- Next, **Store Listing** is clicked.

4. Prepare Store Listing

Store listing displays a page which allows the developer to enter the **Description** of the application, **Promo text**, **Recent changes**, and so on as shown in figure 12.10.

The screenshot shows the Google Play Developer Console interface for the 'HELLO WORLD' app. The left sidebar includes links for API, Store Listing, In-app Products, Services & APIs, and Optimization Tools. The main content area is titled 'STORE LISTING' and shows the following details:

- Title:** English (United States) - en-US (Hello World)
- Description:** English (United States) - en-US (Hello World)
- Promo text:** English (United States) - en-US (Hello World)
- Recent changes:** English (United States) - en-US (Hello World)

GRAPHIC ASSETS:

If you haven't added localized graphics for each language, graphics for your default language will be used. Learn more

Phone: Only upload phone screenshots if your app is designed for phones.

7 inch tablet: Only upload tablet screenshots if your app is designed for tablets. Learn more

10 inch tablet: Only upload tablet screenshots if your app is designed for tablets. Learn more

High-res icon: Default - English (United States) - en-US (96x96), 128x128, 256x256 (both alpha)

Feature Graphic: Default - English (United States) - en-US (96x96 or 128x128), 256x256 (both alpha)

Promo Graphic: Default - English (United States) - en-US (640x128 or 256x480), 480x960 (both alpha)

Promo Video: Default - English (United States) - en-US (Please enter a URL). Please enter a URL.

CATEGORIZATION:

Application type: Select an application type

Category: Select a category

Content rating: Select a content rating

CONTACT DETAILS:

Please provide either a website or an email address.

Website: <http://paceworld.com>

Email: mduvula@paceworld.com

Phone: Phone

PRIVACY POLICY:

If you wish to provide a privacy policy URL for this application, please enter it below.

Link to policy: [View URL](#) Not submitting a privacy policy URL at this time. Learn more

USEFUL ANDROID RESOURCES:

- Android Developers
- Android Design
- Android.com

USEFUL TOOLS:

- Google Wallet Merchant Center
- Google Analytics
- AdMob

NEED HELP?

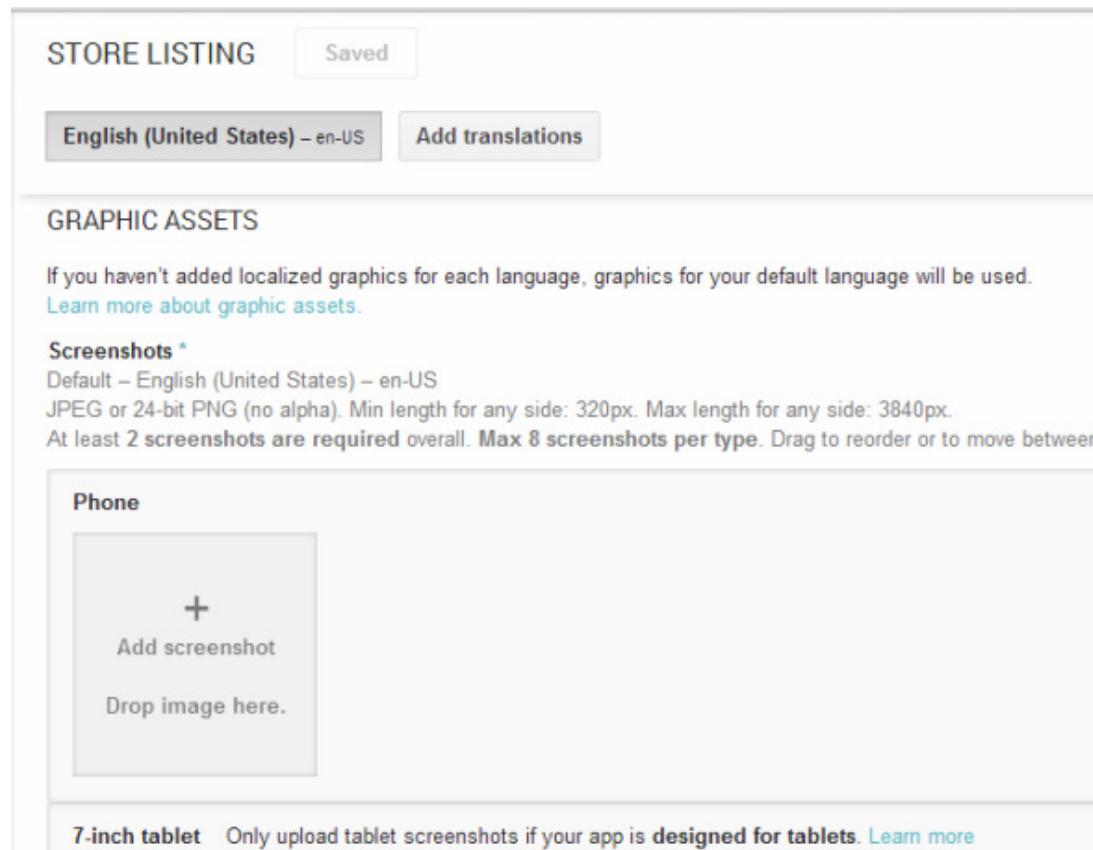
- Help center
- Translators
- Contact support

© 2013 Google - Google Play Terms of Service | Privacy Policy | Developer Distribution Agreement

Figure 12.10: Store Listing

5. Upload of the Application Screenshots

Playstore requires screenshots of the application which is going to be uploaded as shown in figure 12.11.

**Figure 12.11: Application Screenshots**

Screen shots of the application is taken using either Emulator or Real device and uploaded as shown in figure 12.12.

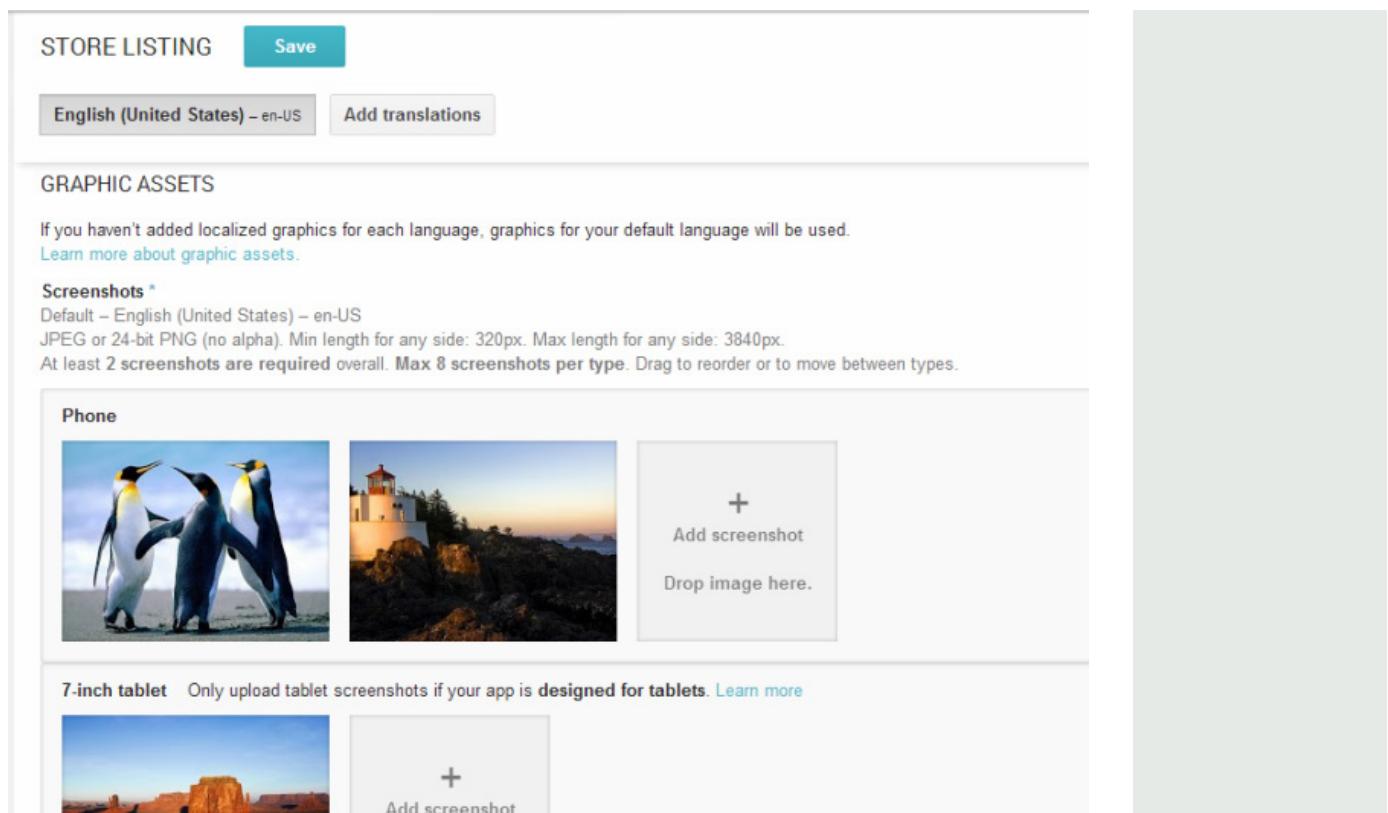


Figure 12.12: Image Upload

Different screenshots can be uploaded for different screen sizes as specified.

In the same pane the developer needs to fill in the **Categorization Details**.

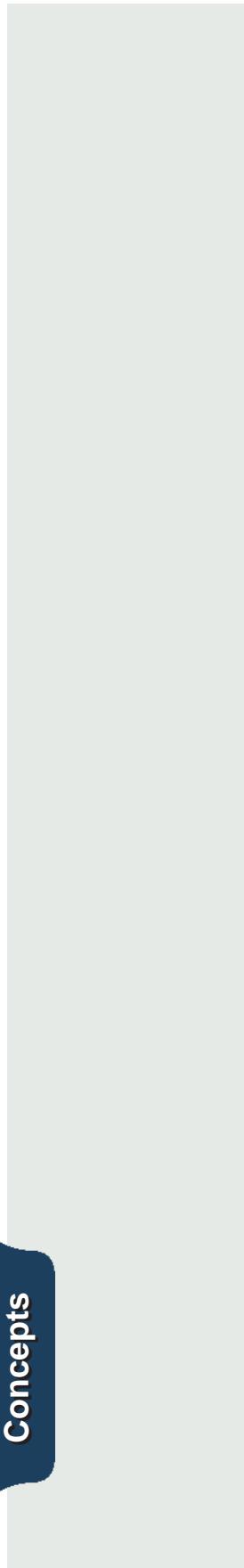
6. Categorization Details

Categorization details mainly specify the application type and category. The **Application type** and **Category** helps the developer to specify the type of application that is getting uploaded as shown in figure 12.13.

Note - Application type can be Social, Entertainment, Puzzle, and so on.

7. Contact Information

Playstore allows the developer to enter the contact information as shown in figure 12.13. The contact information includes details such as **Website**, **Email**, and **Phone**.



STORE LISTING Saved

CATEGORIZATION

Application type * Select an application type

Category * Select a category

Content rating * Select a content rating
Learn more about content rating.

CONTACT DETAILS
Please provide either a website or an email address.

Website * http://pacewisdom.com

Email * madhukarah@pacewisdom.com

Phone

PRIVACY POLICY *
If you wish to provide a privacy policy URL for this application, please enter it below.

Figure 12.13: Category and Contact Information

Once the developer completes all the information, he/she has to click **Save**.

8. Upload of the .apk File

Finally, the developer needs to click **Upload you first APK** once the details have been completed as shown in figure 12.14.

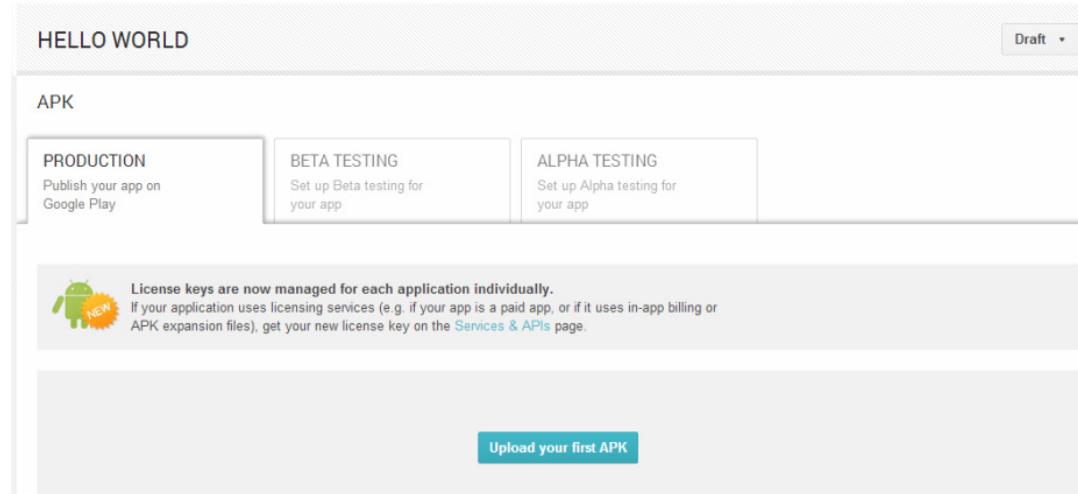


Figure 12.14: Upload of .apk File

The developer needs to navigate to the desired folder and select the **.apk** file and the uploading process starts as shown in figure 12.15.

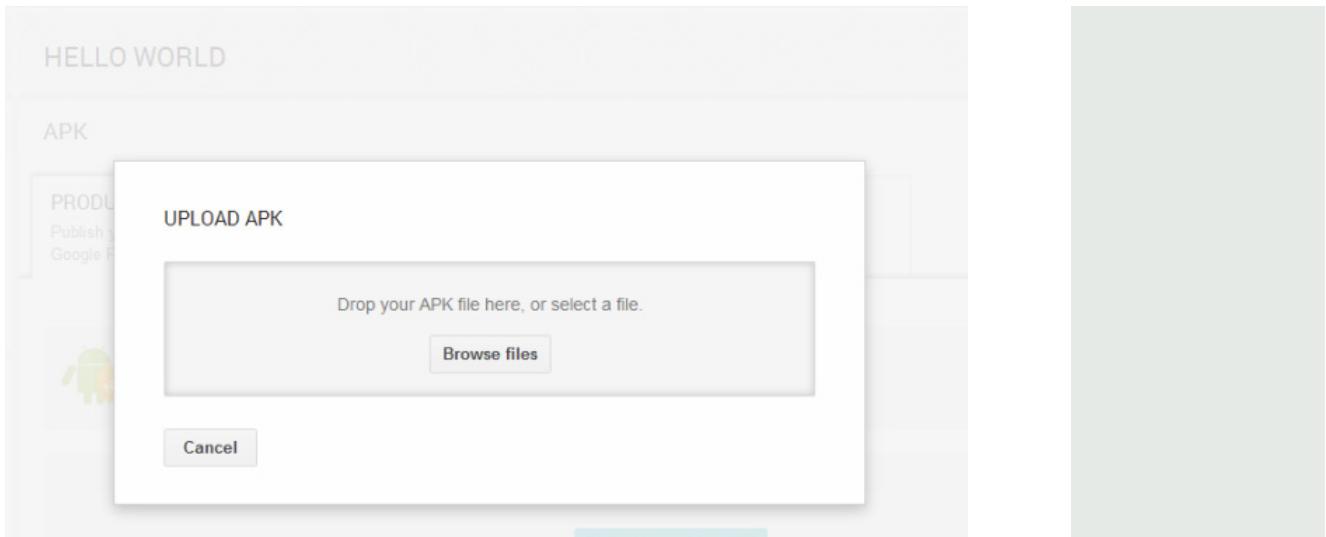


Figure 12.15: Upload of APK File

9. Pricing and Distribution

Click **Pricing and Distribution** tab which is displayed in the left pane of the screen as shown in figure 12.16.

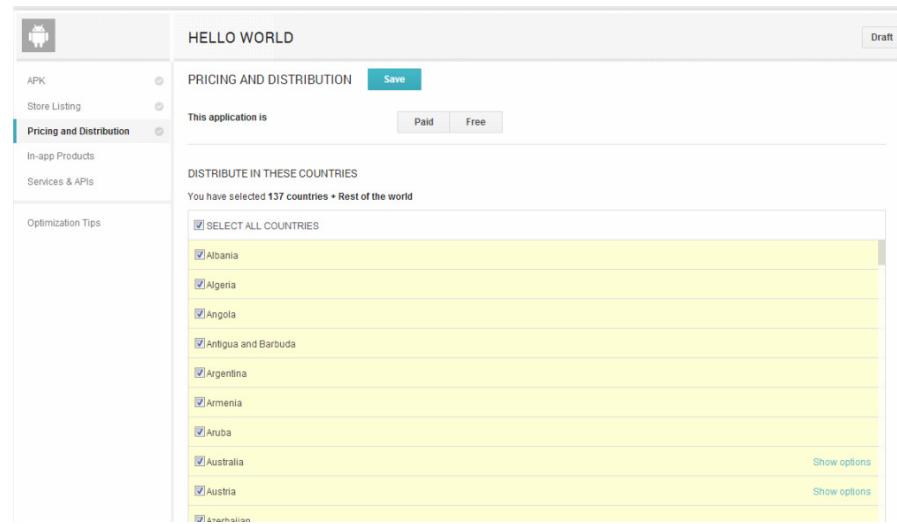
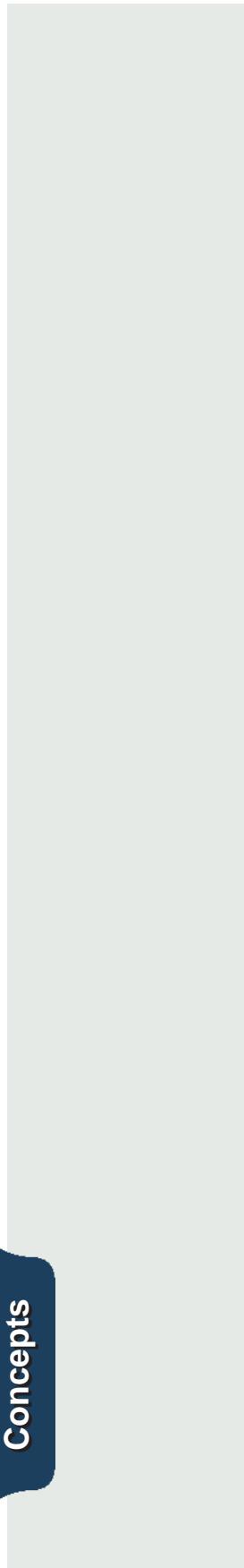


Figure 12.16: Pricing and Distribution Tab

This tab helps to set the price and name of the countries where the application can be distributed.

10. Publish the Application

Once all these steps are completed, the developer is ready to publish the application.

12.5 Check Your Progress

1. Which of the following options displays the year when Google was first acquired by Android?

(A)	2005	(C)	2008
(B)	2007	(D)	2006

2. The Android market is known to be the very own repository of _____.

(A)	Apple	(C)	None of these
(B)	Samsung	(D)	Google

3. Which of the following term correctly expands the file extension .apk?

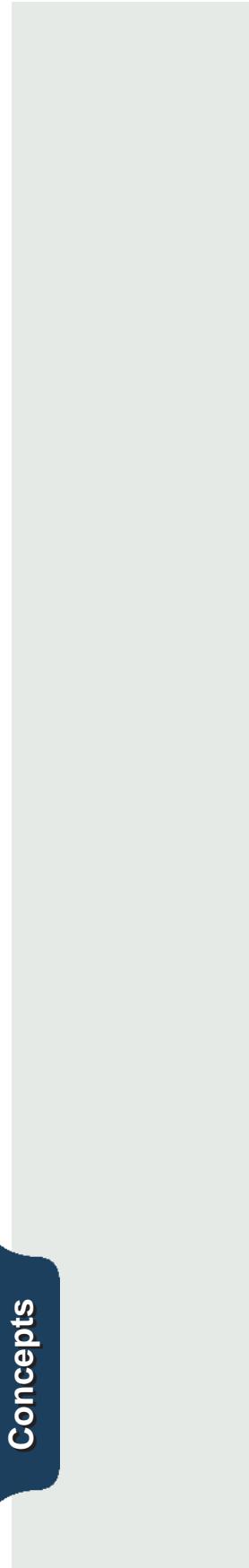
(A)	Android Programmer Kit	(C)	Android Package Key
(B)	Android Packaging Kit	(D)	Android Programmer Key Interface

4. Which of the following option represents the resolution for large screen size in landscape mode of Android?

(A)	780 * 876 mb	(C)	1200 * 800 px
(B)	180 dp * 48 sp	(D)	640 dp * 480 dp

5. Which of the following option represents the % share of Ice Cream Sandwich in Android market?

(A)	27.5	(C)	23
(B)	89	(D)	46



12.5.1 Answers

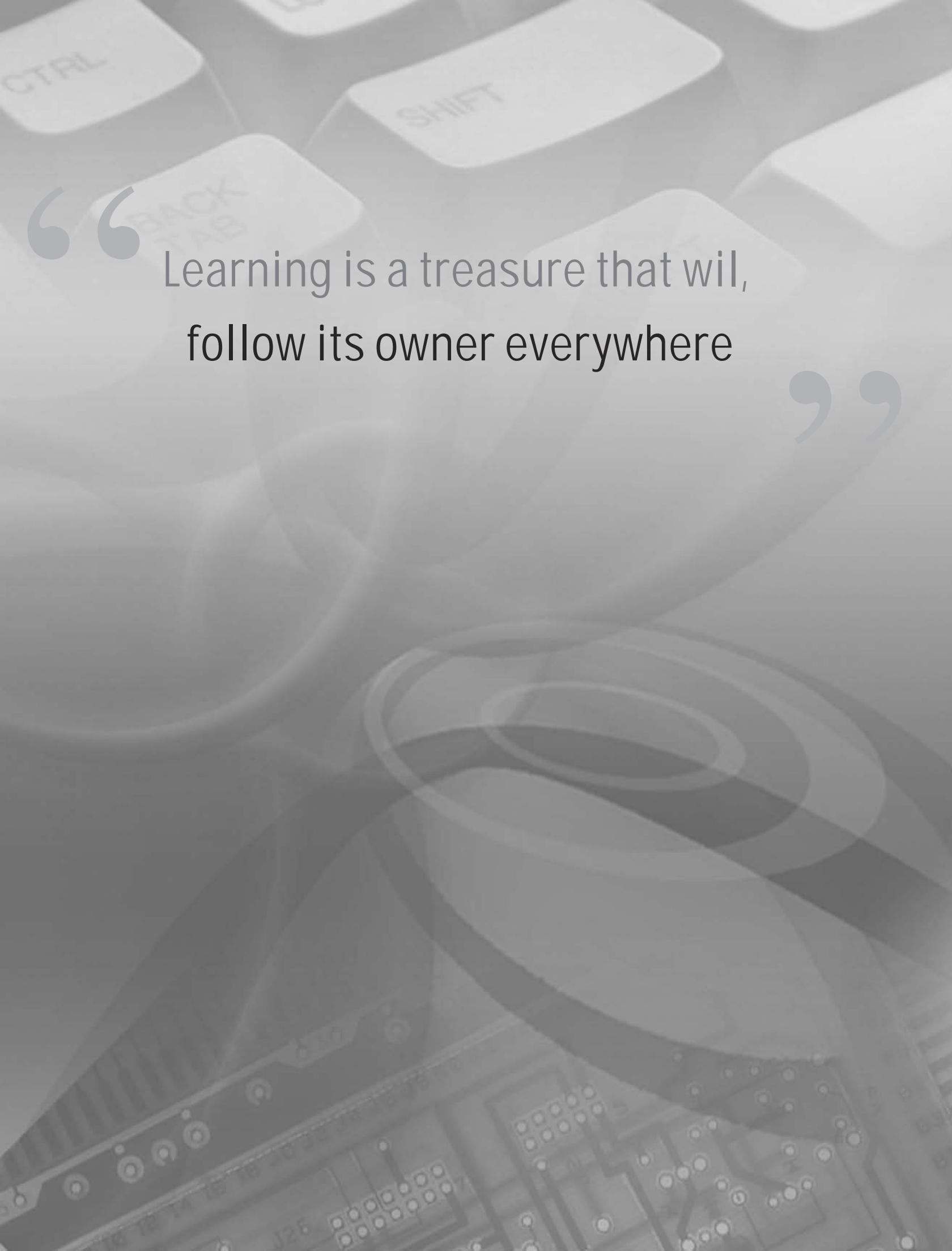
1.	A
2.	D
3.	B
4.	C
5.	A



- The Android Market or the Android central is known to be the very own repository of Google for Android applications.
- Android Market recently renamed as Google play is developed and maintained by Google. It is an online electronic store or digital application distribution platform for android powered devices.
- The application developed by any developer has to be tested before it is uploaded to the Android market to ensure that your app is error and bug free.
- The developer needs to register before publishing the app with a publisher account by visiting the Google Play Developer console at <https://play.google.com/apps/publish/>.

 Try it Yourself

Samantha wants to create a Student Management System application and publish it on the Android market. You as a developer help her to publish the application.



“ Learning is a treasure that wil,
follow its owner everywhere ”