RUNGTA
Group of colleges

INDUSTRIAL TRAINING REPORT

ON ANDROID

PROJECT TITLE : Online Used Goods Selling App.

TRAINING CENTRE : UNIVERSAL INFORMATICS,Indore.

Name : SIMRAN JOTWANI

Roll No. : 3972212026

Semester : 7th Sem

Branch : CSE(B)

RUNGTA
Group of colleges

INDUSTRIAL TRAINING REPORT

ON ANDROID

PROJECT TITLE : **Online Used Goods Selling App.**

TRAINING CENTRE : UNIVERSAL INFORMATICS,Indore.

Name :	**VARTIKA MUKATI**
Roll No. :	**3972212032**
Semester :	**7th Sem**
Branch :	**CSE(B)**

INTRODUCTION OF ANDROID:

- The Android operating system is an *open platform:* Any hardware manufacturer or provider can make or sell Android devices.
- Android is *cross-compatible:* It can run on devices of many different screen sizes and resolutions, including phones and tablets.
- Developing in Android is simple because its default language is Java.

Java: Your Android programming language

Android applications are written in Java — not the full-blown version of Java that's familiar to developers using Java Platform, Enterprise Edition (J2EE), but a subset of the Java libraries that are specific to Android. This smaller subset of Java excludes classes that aren't suitable for mobile devices. If you have experience in Java, you should feel right at home developing apps in Android.

Activities:

An Android application can consist of only a single activity or several. An activity serves as a container for both the user interface and the code that runs it. You can think of activities as *pages* of your app — one page in you app corresponds to one activity.

Intents:

Intents are used to start activities and to communicate among various parts of the Android operating system. An application can either broadcast an intent or receive an intent.

An intent is composed of two elements:

✓ **An action:** The general action to be performed (such as view, edit, or dial) when the intent is received.
✓ **Data:** The information that the action operates on, such as the name of a contact.

Cursorless controls:

Unlike the PC, where you manipulate the mouse to move the cursor, an Android device lets you use your fingers to do nearly anything you can do with a mouse. Rather than right-click in Android, however, you long-press an element until its context menu appears.

As a developer, you can create and manipulate context menus. You can allow users to use two fingers on an Android device, rather than a single mouse cursor, for example. Fingers come in all sizes, so design the user interface in your apps accordingly. Buttons should be large enough (and have sufficient spacing) so that even users with larger fingers can interact with your apps easily, whether they're using your app on a phone or tablet.

Views :

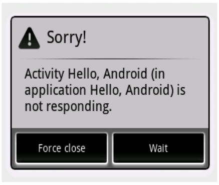
A *view,* which is a basic element of the Android user interface, is a rectangular area of the screen that's responsible for drawing and event handling. Views are a basic building block of Android user interfaces, much like paragraph <p> or anchor <a> tags are building blocks of an HTML page. Some common views you might use in an Android application might be a TextView , ImageView , Layout, and Button, but there are dozens more out there for you to explore. Many more views are ready for you to use.

Asynchronous calls:

You use the AsyncTask class in Android to run multiple operations at the same time without having to manage a separate thread yourself. The AsyncTask class not only lets you start a new process without having to clean up after yourself but also returns the result to the activity that started it — creating a clean programming model for asynchronous processing. In general, we use loaders in this book rather than AsyncTasks, but it's useful to know about AsyncTasks for those occasional cases where a loader won't do what you want. A *thread* is a process that runs separately from, but simultaneously with , everything else that's happening.

You use asynchronous processing for tasks that might take more than a small fraction of a second, such as network (Internet) communication; reading from, or writing to, storage; or media processing. When users have to wait for your task to complete, use an asynchronous call and an element in the user interface to notify them that something is happening.

Downloading the latest Twitter messages via the Internet takes time, for example. If the network slows and you aren't using an asynchronous model, the application will lock up and the user will likely assume that something is wrong because the application isn't responding to her interaction. If the application fails to respond within a reasonable length of time (defined by the Android operating system), the user sees the Application Not Responding (ANR) dialog box, as shown in Figure . The user can then choose whether to close the application or wait for it to recover.


Sorry!

Activity Hello, Android (in application Hello, Android) is not responding.

Background services:

If you're a Windows user, you may already know what a *service* is: an application that runs in the background and doesn't necessarily have a user interface. A classic example is an antivirus application that usually runs in the background as a service. Even though you don't see it, you know that it's running.

Most music players that can be downloaded from the Google Play Store, for example, run as background services. Users can then listen to music while checking e-mail or performing other tasks that require the use of the screen.

Software Tools:

Various Android tools are at your disposal while you're writing Android applications. The following sections outline some of the most popular tools to use in your day-to-day Android development process.

- Internet.
- Audio and Video support.
- Contacts.
- Security.

- Google APIs.

Internet:

Thanks to the Internet capabilities of Android devices, users can find realtime information on the Internet, such as the next showing of a new movie or the next arrival of a commuter train. As a developer, you can have your apps use the Internet to access real-time, up-to-date data, such as weather, news, and sports scores, or (like YouTube) to store your application's icons and graphics.

You can even offload your application's more intense processes to a web server whenever appropriate, to save processing time or to streamline the app. In this well-established software architecture, known as *client–server computing,* the client uses the Internet to make a request to a server that's ready to perform some work for your app. The built-in Maps app is an example of a client that accesses map and GPS data from a web server.

Audio and video support:

Including audio and video in your apps is a breeze in the Android operating system. Many standard audio and video formats are supported, and adding multimedia content to your apps — such as sound effects, instructional videos, background music, and streaming video and audio from the Internet — couldn't be easier. Be as creative as you want to be. The sky's the limit.

Contacts:

Your app can access a user's Contacts list, which is stored on the device, to display the contact information in a new or different way or you can create your own Contacts list. You might even write an app that couples the contact information with the GPS system to alert the user whenever she's near a contact's address. Don't use information from the Contacts list in a malicious way. Use your imagination, but be responsible about it.

Security:

Suppose that someone releases an app that sends a user's entire Contacts list to a server for malicious purposes. For this reason, most functions that modify a user's Android device or access its protected content need specific *permissions*. For example, if you want to download an image from the web, you need permission to use the Internet so that you can download the file to your device, and you need a separate permission to save the image file to an SD card. When your app is being installed, the user is notified of the permissions your app is requesting and can decide whether to proceed. Though asking for permission isn't optional, it's as easy as implementing a single line of code in your application's manifest file.

Google APIs:

Users of the Android operating system aren't limited to making calls, organizing contacts, or installing apps. As a developer, you have great power at your fingertips — you can even integrate maps into your application, for example. To do so, you use the Maps APIs that contain map widgets.

✓ **Pinpointing locations on a map:**

Perhaps you want to write an app that displays a user's current location to friends. You can spend hundreds of hours developing a mapping system, or you can use the Android Maps API, which Google provides for use in your apps. You can embed the API in your application and you don't have to invest hundreds of development hours or even a single cent. Using the Maps API, you can find almost anything that has an address. The possibilities are endless — a friend's location, the nearest grocery store, or your favorite gas station.

Starting a New Project in Eclipse:

First things first: Start Eclipse. You should see a screen that looks similar to the one shown in Figure3-1 . Now you're ready to start cooking with Android. Install the Eclipse Android Development Tools (ADT) plug-in. It gives you the power to generate new Android applications directly from within the Eclipse File menu.

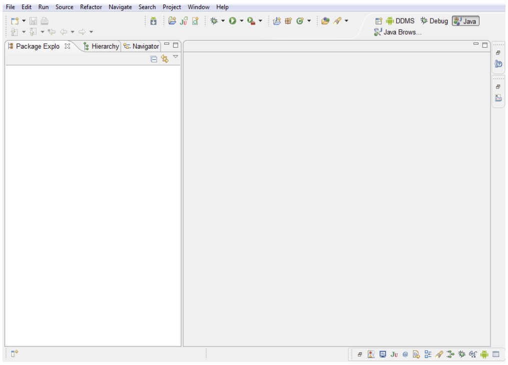

File Edit Run Source Refactor Navigate Search Project Window Help
DDMS
Debug
Java
Java Brows...
Package Explo
Hierarchy
Navigator

Figure 3-1: The Eclipse development environment.

Follow these steps to create your first Android application project:

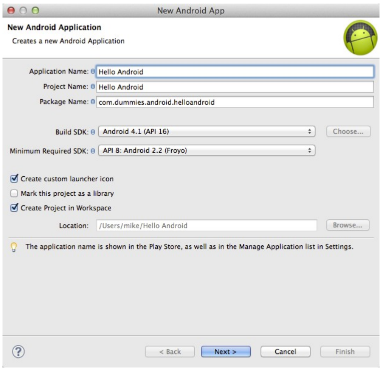
1. **In Eclipse, choose File➪New➪Other . Select Android Application Project.**

   The New Android App Wizard opens, as shown in Figure 3-2.

2. **Enter** Hello Android **as the application name.**

The application name is the name of the application as it pertains to Android. When the application is installed on the emulator or physical device, this name appears in the application launcher. The Project and Package names should auto complete for you. The Project Name field is important. The descriptive name you provide identifies your project in the Eclipse workspace. After your project is created, a folder in the workspace is named with the project name you define.

When you set up Eclipse ,the Eclipse system asks you to set your default workspace. The workspace usually defaults to your *home directory,* where the system places files pertinent to you and where you can find your home directory listed in the Location field. If you would rather store your files in a location other than the default workspace location, deselect the Create Project in Workspace check box, which enables the Location text box. Click the Browse button, and select a location where you want your files to be stored.


New Android App

New Android Application
Creates a new Android Application

Application Name: Hello Android
Project Name: Hello Android
Package Name: com.dummies.android.helloandroid

Build SDK: Android 4.1 (API 16)
Minimum Required SDK: API 8: Android 2.2 (Froyo)

☑ Create custom launcher icon
☐ Mark this project as a library
☑ Create Project in Workspace
Location: /Users/mike/Hello Android

The application name is shown in the Play Store, as well as in the Manage Application list in Settings.

Figure 3-2: The New Android App Wizard.

3. In the Package Name box, type com.dummies.android.helloandroid**.**
This is the name of the Java package.

Understanding Android versioning

Version codes aren't the same as version names. (Huh?) Android has version names and version codes. Each version name has a single version code associated with it. The following table outlines the version names and their respective version code. You can also find this information in the Build Target section of the New Android Project dialog box.

Version Name (Platform Level)	Version Code (API Level)
2.0	5
2.0.1	6
2.1	7
2.2	8
2.3.0–2.3.2	9
2.3.3 - 2.3.4	10
3.0	11
3.1	12
3.2	13
4.0.0 - 4.0.2	14
4.0.3	15
4.1	16

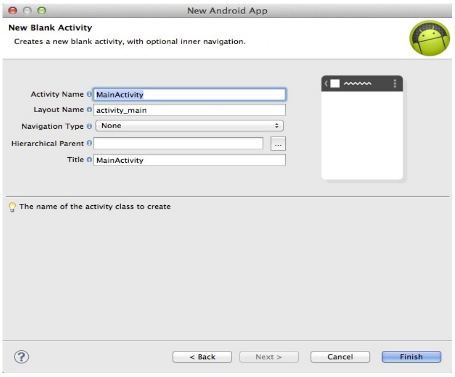
4. Select Android 4.1 from the Build SDK drop-down list and API 8: Android 2.2 from the Minimum Required SDK drop-down list, and then click Next.

The Build SDK drop-down list identifies which application programming interface (API) you want to develop for this project. Always set the Build Target SDK to the latest version that you've tested your app on. When you select Android 4.1, you build and test your app on devices going up to Android 4.1. Doing so allows you to develop with the Android 4.1 APIs, which include new features such as Android Beam. If you had selected Android 2.2 as the target, you wouldn't be able to use any features supported by version 4.1 (or 3.1). Setting the Minimum Required SDK version lower than the Build Target SDK means that your app still runs on older devices, all the way down to Android 2.2 in this case. When you set the build target and minimum required SDKs to different values, you can't use any APIs newer than the minimum required SDK on old devices. For example, you can't use Android Beam on an Android 2.2 device; the app will crash.

For more information, see the section "Understanding the Build Target and Min SDK Version settings," later in this chapter. The Android Application Icon Editor appears.

5. (Optional) Create an application icon for your project and click Next.

6. In the Create Activity box, choose BlankActivity and click Next.
The New Blank Activity screen appears, as shown in Figure 3-3.


New Android App

New Blank Activity

Creates a new blank activity, with optional inner navigation.

Activity Name: MainActivity
Layout Name: activity_main
Navigation Type: None
Hierarchical Parent:
Title: MainActivity

The name of the activity class to create

Figure 3-3: Set up your new activity.

7. Enter MainActivity **in the Activity Name box.**

The New Blank Activity screen defines what the initial activity is called — the entry point to your application. When Android runs your application, this file is the first one to be accessed. A common naming pattern for the first activity in your application is MainActivity.java. (How creative.)

8. Click the Finish button.

You're done! You should see Eclipse with a single project in the Package Explorer. For the purpose of this book, Package Explorer and Project Explorer (which is showing in Figure 3-4) are the same thing. Different names; same function.

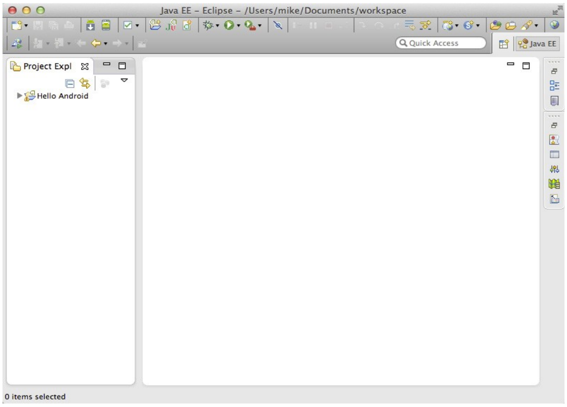


Figure 3-4: The Eclipse development environment with your first Android project , Hello Android.