**Android Practical Experiments**

1. **Introduction to android:**
2. **JAVA( JDK) INSTALLATION.**
3. **Eclipse IDE.**
4. **Android SDK.**
5. **Create Custom Android AVD.**
6. **Locate Path for Android SDK.**
7. **How to install ADT plug-ins.**
8. **FIRST REQUISITE TO INSTALL JAVA:**

**JAVA JDK**

* **Search “Java JDK”**
* **Mostly click the first link.**
* **There should be four options to download(Java platform JDK, JDK+JavaFX Bundle, and JDK+Java EE Bundle).**
* **Install just like any other program, with all default settings.**
* **Click the Strat Button and Go to My Computer🡪C: Drive🡪Program Files Floder🡪Open up the Floder called Java🡪Open the Java version which you have downloaded🡪Open the bin folder🡪Inside the bin folder look for the javac,and RIGHT-CLICK on it and go to Properties.**
* **A new window should pop up giving the properties of the javac, there should be an attribute called Location just copy the path C:\Program Files\Java\jdk1.6.0\_12\bin”.**
* **Now click on the Start Button again and right-click on the “My Computer” and open up the “Properties”🡪Look for “Advanced System Settings” and open that up(Click Continue)🡪Towards the bottom of the popup, Click “Environment Variables”🡪Set up new User Variables by clicking the “New” Button🡪 for the Variable Name –Path🡪for the Variable Value –Paste the location of javac(C:\Program Files\Java\jdk1.7.0\bin) 🡪 Click OK on all three popups and close your System settings panel as well.**
* **Open up the command prompt and type cmd and then after click enter🡪type “javac” in the command Prompt and your problem should be solved.**

1. **ECLIPSE IDE:**

**The first step towards developing any applications is obtaining the integrated development environment (IDE). In the case of Android, the recommended IDE is Eclipse, a multi-language software development environment featuring an extensible plug-in system. It can be used to develop various types of applications, using languages such as Java, Ada, C, C++, COBOL, Python, etc.**

**For Android development, you should download the Eclipse IDE for Java EE Developers (www.eclipse .org/downloads/packages/eclipse-ide-java-ee developers/ heliossr1). Six editions are available: Windows (32 and 64-bit),Mac OS X (Cocoa 32 and 64), and Linux (32 and 64-bit). Simply select the relevant one for your operating system. All the examples in this book were test educing the 32-bit version of Eclipse for Windows.**

**Once the Eclipse IDE is downloaded, unzip its content (the eclipse folder) into**

**a folder, say C:\Android\. Figure 1-7 shows the content of the eclipse folder.**

1. **DOWNLOAD ANDROID SDK:**

**The next important piece of software you need to download is, of course, the Android SDK. The Android SDK contains a debugger, libraries, an emulator, documentation, sample code, and tutorials.You can download the Android SDK from** [**http://developer.android.com/sdk/**](http://developer.android.com/sdk/)**index.html.**

**Once the SDK is downloaded, unzip its content (the android-sdk-windows folder)into the C:\Android\ folder, or whatever name you have given to the folder you just created.**

1. **HOW TO CREATE & SETUP CUSTOM ANDROID EMULATOR?**

**OPEN ECLIPSE=>Select Window ➪Android SDK and AVD Manager. In the resulting dialog, select Virtual Devices from the left panel and click the New *. . .* button.**

**🡪Enter a name for your device, and choose an SDK target and screen resolution. Set the SD Card size to larger than 8 MB: enter 12 into the text-entry box as shown in**

**figure**

**Android Virtual Devices:**

**Android Virtual Devices are used to simulate the software builds and hardware specifications available on different devices. This lets you test your application on a variety of hardware platforms without needing to buy a variety of phones. *The Android SDK doesn’t include any pre-built virtual devices, so you will need to create at least one device before you can run your applications within an emulator.***

**Each virtual device is configured with a name, a target build of Android (based on the SDK version it supports), an SD Card capacity, and screen resolution, as shown in the ‘‘Create new AVD’’ dialog. Each virtual device also supports a number of specific hardware settings and**

**figure**

**restrictions that can be added in the form of NVPs in the hardware table. These additional settings include:**

**➤Maximum virtual machine heap size**

**➤Screen pixel density**

**➤SD Card support**

**➤The existence of DPad, touchscreen, keyboard, and trackball hardware**

**➤Accelerometer and GPS support**

**➤Available device memory**

**➤Camera hardware (and resolution)**

**➤Support for audio recording**

**Different hardware settings and screen resolutions will present alternative user-interface skins to represent the different hardware configurations. This simulates a variety of mobile device types. To complete the illusion, you can create a custom skin for each virtual device to make it look like the device it is emulating.**

1. **Locate Path for Android SDK:**

**OPEN ECLIPSE=>Select Window ➪Preferences🡺 Select Android From Left Hand Side 🡺Click on Browse Button 🡺Browse the Android SDK-Windows from the Drive.**

**F igure**

1. **INSTALLING THE ADT PLUG-IN:**

**Install the developer tools plug-in by following these steps:**

**1. Open Eclipse =>Select Help ➪Install New Software*. . .* from within Eclipse.**

**2. In the resulting dialog box enter the following address into the Work With text entry box and press Enter: https://dl-ssl.google.com/android/eclipse/**

**3. Eclipse will now search for the ADT plug-in. When finished it will display the available plug-in, as shown in Figure. Select it by clicking the checkbox next to the Developer Tools root node, and click Next.**

**figre**

**4. Eclipse will now download the plug-in. When it’s finished, ensure both the Android DDMSand Android Developer Tools plug-ins are selected and click Next.**

**5. Read and then Accept the terms of the license agreement, and click Next and then Finish. As the ADT plug-in is not signed, you’ll be prompted before the installation continues.**

**6. When installation is complete you’ll have to restart Eclipse and update the ADT preferences. Restart and select Window ➪Preferences (or Eclipse ➪Preferences for MacOS).**

**7. Then select Android from the left panel.**

**8. Click Browse *. . .* and navigate to the folder into which you unzipped the Android SDK; then click Apply. The list will then update to display each of the available SDK targets. Click OK to complete the SDK installation.**

**Updating the Plug-In As the Android SDK matures, there are likely to be frequent updates to the ADT plug-in. In most cases, to update your plug-in you simply:**

**1. Navigate to Help ➪Check for Updates.**

**2. If there are any ADT updates available, they will be presented. Simply select them and choose Install.**

**Dalvik Debug Monitor Service (DDMS)**

**The emulator lets you see how your application will look, behave, and interact, but to really see what’s happening under the surface you need the Dalvik Debug Monitoring Service. The DDMS is a powerful debugging tool that lets you interrogate active processes, view the stack and heap, watch and pause active threads, and explore the file system of any connected Android device. The DDMS perspective in Eclipse also provides simplified access to screen captures of the emulator and the logs generated by Log Cat. If you’re using the ADT plug-in, the DDMS is fully integrated into Eclipse and is available from the DDMS perspective. If you aren’t using the plug-in or Eclipse, you can run DDMS from the command line and it will automatically connect to any running device or emulator.**

**The Android Debug Bridge (ADB)**

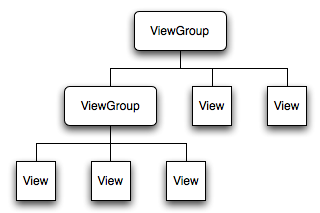
**The *Android debug bridge (ADB)* is a client-service application that lets you connect with an Android Emulator or device. It’s made up of three components: a daemon running on the emulator, a service that runs on your development hardware, and client applications (like the DDMS) that communicate with the daemon through the service.**

**As a communications conduit between your development hardware and the Android device/emulator, the ADB lets you install applications, push and pull files, and run shell commands on the target device. Using the device shell you can change logging settings, and query or modify SQLite databases available on the device. The ADT tool automates and simplifies a lot of the usual interaction with the ADB, including application installation and updating, file logging, and file transfer (through the DDMS perspective**

1. **Program to show use of UI elements:**

**USER INTERFACE:**

**In an Android application, the user interface is built using View and View-Group objects. There are many types of views and view groups, each of which is a descendant of the View class. View objects are the basic units of user interface expression on the Android platform. The View class serves as the base for subclasses called "widgets," which offer fully implemented UI objects, like text fields and buttons. The View-Group class serves as the base for subclasses called "layouts," which offer different kinds of layout architecture, like linear, tabular and relative.**

****

**On the Android platform, we define an Activity's UI using a hierarchy of View and View Group nodes, as shown in the diagram above. This hierarchy tree can be as simple or complex as we need it to be, and we can build it up using Android's set of predefined widgets and layouts, or with custom Views that we create.**

**FORM WIDGETS VIEWS:**

**A widget is a View object that serves as an interface for interaction with the user. Android provides a set of fully implemented widgets, like buttons, checkboxes, and text-entry fields, so we can quickly build our UI. Some widgets provided by Android are more complex, like a date picker, a clock, and zoom controls. But we're not limited to the kinds of widgets provided by the Android platform. If we want to do something more customized and create our own actionable elements, we can, by defining our own View object or by extending and combining existing widgets. The wrap\_content constant will set the size of a View to the minimum required to contain the contents it displays (such as the height required to display a wrapped text string). The fill\_parent constant expands the View to fill the available space within the parent View (or screen).Our layout is set to fill the entire screen, while both text-based Views are restricted to that required by the text being displayed.**

**Different Form-Widget views which we are going to use in our application are:**

* **Text Views**
* **Button Views**
* **Check box Views**
* **Spinner Views**
* **Progress Bar Views**
* **Seek Bar Views**
* **Rating Bar Views**
* **Checked Text Views**
* **Switch Views**
* **Quick Contact Badge**

**Text Views:**

**A standard read-only text label. It supports multiline display, string formatting, and automatic word wrapping. To display a text label to an user we can view it in three different ways Large, Medium, & Small. .**

**Button Views:**

**A standard push-button. We can have some buttons such as:**

**a) Toggle Buttons: this will set the text on the view either ON or OFF. When it is in ON it will radiate a light on the button.**

**b) Radio Button: A two-state grouped button. A group of these presents the user with a number of binary options of which only one can be enabled at a time.**

**CheckBox: Views: A two-state button represented by a checked or unchecked box.**

**QuickContactBadge Views:Displays a badge showing the image icon assigned to a contact you specify using a phone number, name, e-mail address, or URI. Clicking the image will display the quick contact bar, which provides shortcuts for contacting the selected contact—including calling, sending an SMS, e-mail, and IM.**

**Spinner Views: A composite control that displays a Text View and an associated List View that lets you select an item from a list to display in the textbox. It’s made from a Text View displaying the current selection, combined with a button that displays a selection dialog when pressed.**

**Seek Bar Views: By using this view we can set the progress of the text by just dragging the view.**

**Progress Bar Views:The Progress-Bar view provides visual feedback of some ongoing tasks, such as when you are performing a task in the background. For example, you might be downloading some data from the Web and need to update the user about the status of the download**

**Rating Bar view: we can use this particular view when we are rating for some thing.**

**figr**

**DIFFERENT TEXT FIELDS VIEWS:**

**A text field allows the user to type text into your app. It can be either single line or multi-line. Touching a text field places the cursor and automatically displays the keyboard. In addition to typing, text fields allow for a variety of other activities, such as text selection (cut, copy, paste) and data look-up via auto-completion.**

**You can add a text field to you layout with the** [**EditText**](http://developer.android.com/reference/android/widget/EditText.html) **object. You should usually do so in your XML layout with a <EditText> element.**

**Text fields can have different input types, such as number, date, password, or email address. The type determines what kind of characters are allowed inside the field, and may prompt the virtual keyboard to optimize its layout for frequently used characters.**

**"text"**

**Normal text keyboard.**

**"textEmailAddress"**

**Normal text keyboard with the @ character.**

**"textUri"**

**Normal text keyboard with the / character.**

**"number"**

**Basic number keypad.**

**TIME AND DATE PICKER’S VIEWS:**

**Selecting the date & Time is one of the common tasks you need to perform in a mobile application. Android supports this functionality through the Time Picker & Date Picker Views.**

**Time Picker:**

**The Time-picker View enables users to select a time of the day, in either 24-hour mode or AM/PM mode.**

**Date Picker:**

**Using a Date Picker View You can enable the user to select the day on the Activity**

**F igr**

**DIFFERENT IMAGES AND MEDIA VIEWS:**

**The different images &Media Views are**

* **Image View**
* **Image Button**
* **Gallery View**
* **Media Controller**
* **Video View**

**Image View is used to view an image on the screen.**

**Gallery View: Gallery is a view that shows items in an Center locked Horizontal Scrolling list.**

**figr**

**COMPOSITE VIEWS:**

* **List View**
* **Expandable List**
* **Grid View**
* **Scroll View**
* **Search View**
* **Horizontal Scroll View**
* **Sliding Drawer**
* **Web View**

**figr**

**SELECTION WIDGETS:**

**Android adapters provide a common interface to the data model behind the selection widgets. They are responsible for providing the data for a selection widget, as well as for converting individual elements of data into specific views to be displayed inside the widgets.**

**LISTVIEW:**

**Displaying lists is a very common. The user gets a list of items that can be scrolled. Then, the user select one item. Android provides the view ListView for this.**

**ListView is a ViewGroup that creates a list of scrollable items. The list items are automatically inserted to the list using a ListAdapter, and we use attached listener via setOnItemSelectedListener() to find out when the selection has changed.**

**Analog Clock and Digital Clock Views:**

**The AnalogClock view displays an analog clock with two hands-one for minutesand one for hours. Its counter Part, the Digital Clock view, displays the time digitally. Both display the system time, and do not allow you to display a particular time.**

**figr**

**Web View:**

**The web-view enables you to embed a web browser in your activity. This is very useful if your application needs to embed some web content, such as maps from some other providers, and so on.**

**figr**

**Grid View:**

**The GridView shows items in a two-dimensional scrolling grid. You can use the GridView togetherwith an ImageView to display a series of images.**

**figr**

1. **Program to show demo of layouts:**

**Layouts are extensions of the ViewGroup class used to position child controls for UI. Layouts can be nested, letting us create arbitrarily complicated interfaces using a combination of layouts.**

**The following includes some of the more versatile layout classes available.**

1. **LinearLayout**

**A) Horizontal**

**B) Vertical**

1. **RelativeLayout**
2. **TableLayout**
3. **Frame Layout**
4. **Absolute Layout**

**Linear Layout:**

**A Linear Layout aligns each child View in either a vertical or a horizontal line. A vertical layout has a column of Views, while a horizontal layout has a row of Views. The Linear Layout enables us to specify a "widget" for each child View that controls the relative size of each within the available space.**

**Linear Horizontal layout Linear Vertical Layout**

**RELATIVE LAYOUT:**

**The most flexible of the native layouts, the Relative Layout lets us define the positions of each child View relative to the others and to the screen boundaries. Relative Layout is a Layout where the positions of the children can be described in relation to each other or to the parent. To make our Relative Layout work, we need ways to reference other widgets within an XML layout file, plus to indicate the relative position of those widgets.**

**Positions Relative to Container:**

**android:layout\_alignParentTop**

**Aligns the widget's top with the top of the container.**

**android:layout\_alignParentBottom**

**Aligns the widget's bottom with the bottom of the container.**

**android:layout\_alignParentLeft**

**Aligns the widget's left side with the left side of the container.**

**android:layout\_alignParentRight**

**Aligns the widget's right side with the right side of the container.**

**android:layout\_centerParentHorizontal**

**Positions the widget horizontally at the center of the container.**

**android:layout\_centerParentVertical**

**Positions the widget's vertically at the center of the container.**

**android:layout\_centerInParent**

**Positions the widget both horizontally and vertically at the center of the container.**

**TABLELAYOUT**

**The Table Layout lets us lay out Views using a grid of rows and columns. Tables can span multiple rows and columns, and columns can be set to shrink or grow. A layout that arranges its children into rows and columns. A TableLayout consists of a number of TableRow objects, each defining a row (actually, we can have other children, which will be explained below). TableLayout containers do not display border lines for their rows, columns, or cells. Each row has zero or more cells; each cell can hold one View object. The table has as many columns as the row with the most cells.**

**FRAMELAYOUT**

**The simplest of the layout, the Frame Layout simply pins each child view to the top left corner. Adding multiple children stacks each new child on top of the one before, with each new View obscuring the last.**

**Absolute Layout:**

**The Absolute Layout enables you to specify the exact location of its children. However, there is a problem with the Absolute Layout when the activity is viewed on a high-resolution screen . For this reason, the Absolute Layout has been deprecated since Android. You should avoid using the Absolute Layout in your UI, as it is not guaranteed to be supported in future versions of Android.**

1. **Program to create Menus and Dialog box.**

**Menus offer a way to expose application functions without sacrificing valuable screen space. Each Activity can specify its own menu that's displayed when the device's menu button is pressed. Application menus are another important part of an application's UI. Menus offer a reliable interface that reveals application functions and settings. The most common application menu is revealed by pressing the MENU key on the device. However, we can also add Context Menus, which may be revealed when the user presses and holds down on an item.**

**Menus are also structured using a View hierarchy, but we don't define this structure. Instead, we define the onCreateOptionsMenu() or onCreateContextMenu() callback methods for our Activity and declare the items that we want to include in our menu. At the appropriate time, Android will automatically create the necessary View hierarchy for the menu and draw each of our menu items in it.**

**Menus also handle their own events, so there's no need to register event listeners on the items in our menu. When an item in our menu is selected, the onOptionsItemSelected() or onContextItemSelected() method will be called by the framework. And just like our application layout, we have the option to declare the items for we menu in an XML file. Menus are an important part of any application. They provide familiar interfaces that reveal application functions and settings. Android offers an easy programming interface for developers to provide standardized application menus for various situations.**

**Android offers three fundamental types of application menus:**

1. **Options Menu :**

**This is the primary set of menu items for an Activity. It is revealed by pressing the device MENU key. Within the Options Menu are two groups of menu items:**

1. **Icon Menu**

**This is the collection of items initially visible at the bottom of the screen at the press of the MENU key. It supports a maximum of six menu items. These are the only menu items that support icons and the only menu items that do not support checkboxes or radio buttons.**

1. **Expanded Menu**

**This is a vertical list of items exposed by the "More" menu item from the Icon Menu. It exists only when the Icon Menu becomes over-loaded and is comprised of the sixth Option Menu item and the rest.**

1. **Context Menu :**

**This is a floating list of menu items that may appear when you perform a long-press on a View (such as a list item).**

1. **Submenu :**

**This is a floating list of menu items that is revealed by an item in the Options Menu or a Context Menu. A Submenu item cannot support nested Submenus.**

**OPTIONS MENU:**

**The Options Menu is an Android user interface component that provides standardized menus. It is opened by pressing the device MENU key. When opened, the menus appear at the bottom of the screen. The Icon Menu is displayed holding the first six menu items. If more than six items are added to the Options Menu, then those that can't fit in the Icon Menu are revealed in the Expanded Menu, via the "More" menu item. The Expanded Menu is automatically added when there are more than six items.**

**The Options Menu is where we should include basic application functions and any necessary navigation items (e.g., to a home screen or application settings). We can also add Submenus for organizing topics and including extra menu functionality.**

**When this menu is opened for the first time, the Android system will call the Activity onCreateOptionsMenu() callback method. Override this method in our Activity and populate the Menu object given to us. We can populate the menu by inflating a menu resource that was defined in XML, or by calling add() for each item we'd like in the menu. This method adds a MenuItem, and returns the newly created object to you. We can use the returned MenuItem to set additional properties like an icon, a keyboard shortcut, an intent, and other settings for the item.**

**There are multiple add() methods. Usually, we'll want to use one that accepts an itemId argument. This is a unique integer that allows us to identify the item during a callback.**

**When a menu item is selected from the Options Menu, we'll receive a callback to the onOptionsItemSelected() method of our Activity. This callback passes us the MenuItem that has been selected. We can identify the item by requesting the itemId, with getItemId(), which returns the integer that was assigned with the add() method. Once we identify the menu item, we can take the appropriate action.**

**CONTEXT MENU:**

**The Android context menu is similar, in concept, to the menu revealed with a "right-click" on a PC. When a view is registered to a context menu, performing a "long-press" (press and hold for about two seconds) on the object will reveal a floating menu that provides functions relating to that item. Context menus can be registered to any View object, however, they are most often used for items in a ListView, which helpfully indicates the presence of the context menu by transforming the background color of the ListView item when pressed.**

**To create a context menu, you must override the Activity's context menu callback methods: onCreateContextMenu() and onContextItemSelected().**

**Inside the onCreateContextMenu() callback method, you can add menu items using one of the add() methods, or by inflating a menu resource that was defined in XML. Then, register a ContextMenu for the View, with registerForContextMenu().**

**In onCreateContextMenu(), we are given not only the ContextMenu to which we will add menu, but also the View that was selected and a ContextMenuInfo object, which provides additional information about the object that was selected. In this example, nothing special is done in onCreateContextMenu(), just a couple items are added as usual.**

**In the onContextItemSelected() callback, we request the getItemId() from the MenuItem, which provides information about the currently selected item. All we need from this is the list ID for the selected item, so whether editing a note or deleting it. This ID can passed to the "edit()" or "delete" methods though they are not implemented in the example. Here, we are just writing a text to the footer to tell the user which menu has been selected.**

**ALERT DIALOG BOX:**

**The** [**AlertDialog**](http://developer.android.com/reference/android/app/AlertDialog.html) **class allows you to build a variety of dialog designs and is often the only dialog class you'll need. As shown in figure 2, there are three regions of an alert dialog:**

**Figure 2. The layout of a dialog.**

1. **Title**

**This is optional and should be used only when the content area is occupied by a detailed message, a list, or custom layout. If you need to state a simple message or question (such as the dialog in figure 1), you don't need a title.**

1. **Content area**

**This can display a message, a list, or other custom layout.**

1. **Action buttons**

**There should be no more than three action buttons in a dialog.**

**The** [**AlertDialog.Builder**](http://developer.android.com/reference/android/app/AlertDialog.Builder.html) **class provides APIs that allow you to create an** [**AlertDialog**](http://developer.android.com/reference/android/app/AlertDialog.html) **with these kinds of content, including a custom layout.**

**// 1. Instantiate an** [**AlertDialog.Builder**](http://developer.android.com/reference/android/app/AlertDialog.Builder.html) **with its constructor   
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());   
   
// 2. Chain together various setter methods to set the dialog characteristics   
builder.setMessage(R.string.dialog\_message)   
       .setTitle(R.string.dialog\_title);   
   
// 3. Get the** [**AlertDialog**](http://developer.android.com/reference/android/app/AlertDialog.html) **from** [**create()**](http://developer.android.com/reference/android/app/AlertDialog.Builder.html#create())**AlertDialog dialog = builder.create();**

**The set...Button() methods require a title for the button (supplied by a** [**string resource**](http://developer.android.com/guide/topics/resources/string-resource.html)**) and a** [**DialogInterface.OnClickListener**](http://developer.android.com/reference/android/content/DialogInterface.OnClickListener.html) **that defines the action to take when the user presses the button.**

**There are three different action buttons you can add:**

**Positive**

**You should use this to accept and continue with the action (the "OK" action).**

**Negative**

**You should use this to cancel the action.**

**Neutral**

**You should use this when the user may not want to proceed with the action, but doesn't necessarily want to cancel. It appears between the positive and negative buttons. For example, the action might be "Remind me later."**

1. **Program to show how to use intents (implicit and explicit)**

**Intent** **is used to launch an appropriate activity. It tells OS an** **action** **to take and the** **data** **on which that action is to be acted upon. For example, when we touch a search button, application creates a url that contains the query string. Then, we load the url into a web browser by creating a new** **intent** **for viewing the url, then passing that** **intent** **to the** **startActivity** **method.**

**Intent is basically a message that is passed between components (such as Activities, Services, Broadcast Receivers, and Content Providers). So, it is almost equivalent to parameters passed to API calls. The fundamental differences between API calls and intents' way of invoking components are:**

* **API calls are synchronous while intent-based invocations are asynchronous.**
* **API calls are compile time binding while intent-based calls are run-time binding.**

**Of course, Intents can be made to work exactly like API calls by using what are called explicit intents, which will be explained later. But more often than not, implicit intents are the way to go and that is what is explained here. When we make an Android application, we start by sub classing Activity class. Activities provide the reusable/interchangeable parts of the flow of UI across the applications. Then, how does one activity invoke another, and pass information about what the user wants to do? The unit of communication is the Intent class.**

**Intent represents an abstraction of a function that one activity requires another activity to perform. Intent form the basis of a system of loose coupling that allows activities to launch another. When an application dispatches intent, it's possible that several activities might be registered to provide the desired operations. One component that wants to invoke another has to only express its' intent to do a job. And any other component that exists and has claimed that it can do such a job through intent-filters is invoked by the android platform to accomplish the job. This means, both the components are not aware of each other's existence and can still work together to give the desired result for the end-user.**

**This invisible connection between components is achieved through the combination of intents, intent-filters and the android platform.**

**This leads to huge possibilities like:**

* **Mix and match or rather plug and play of components at runtime.**
* **Replacing the inbuilt android applications with custom developed applications.**
* **Component level reuse within and across applications.**
* **Service orientation to the most granular level, if I may say**

**Here is additional description about intent, almost formal.**

**An intent is an abstract description of an operation to be performed. It can be used with startActivity to launch an Activity,broadcastIntent to send it to any interested BroadcastReceiver components, and startService (Intent) or bindService (Intent, ServiceConnection, int) to communicate with a background Service.**

**An Intent provides a facility for performing late runtime binding between the code in different applications. Its most significant use is in the launching of activities, where it can be thought of as the glue between activities. It is basically a passive data structure holding an abstract description of an action to be performed. The primary pieces of information in an intent are:**

**Once we have our intent, we need to pass it to Android and get the child activity to launch. Here, we have two options:**

* **Call startActivity() with the Intent. This will cause Android to find the best matching activity and pass the intent to the activity for handling. The activity will not be informed when the child activity is complete.**
* **Call startActivityForResult(), passing it the intent and a number which is unique to the calling activity. Android will find the best matching activity and pass the intent over to the activity. The activity will be notified when the child activity is complete via onActivityResult() callback.**

**EXPLICIT INTENT**

**In an** **explicit intent, we actually specify the activity that is required to respond to the intent. In other words, we explicitly designate the target component. This is typically used for application internal messages. In an** **implicit intent, the main power of the android design, we just declare an intent and leave it to the platform to find an activity that can respond to the intent. Here, we do not declare the target component and hence is typically used for activating components of other applications seamlessly.**

**This example has 2 activities:**

** InvokingActivity**

** InvokedActivity**

**The** **Invoking Activity** **has a button "Invoke Next Activity" which when clicked explicitly calls the** **Invoked Activity** **class. The relevant part of the code is here:**

**Button invokingButton = (Button)findViewById(R.id.invokebutton);**

**invokingButton.setOnClickListener(new OnClickListener() {**

**public void onClick(View v) {**

**Intent explicitIntent = new Intent(InvokingActivity.this,InvokedActivity.class);**

**startActivity(explicitIntent);**

**}**

**});**

**To explicitly select an Activity class, we need to create a new Intent. Then, we should pass our intent to startActivity(Intent) method by specifying the current application Context and Activity class. The startActivity(Intent) method is used to start a new activity. It takes a single argument, an Intent which describes the activity to be executed.**

**After startActivity(Intent) is called, the new Activity (in our case, it is InvokedActivity) will be created and become visible and active, and it will be placed at the top of the activity stack.**

**IMPLICIT INTENT**

**As described earlier, an implicit intent does not name a target component that should act upon the intent. Android resolves as to which component is best suited to respond to an Implicit Intent. In other words, we can ask the Android to launch an Activity that can perform a given action without knowing which application/Activity will perform the task.**

**Basically, an Intent object has the following information (among other things like Component name, extras and flags) which is of interest for implicit intents:**

* **Action**
* **Category**
* **Data**

**So, Android compares the three (action, category and data) to something called Intent Filters that are declared by probable target components who are willing to accept Implicit Intent calls. Intent Filters are the way of any component to advertise its own capabilities to the Android system. This is done declaratively in the AndroidManifest.xml file.**

**So here are some important points to remember:**

1. **Implicit Intents do not specify a target component.**
2. **Components willing to receive implicit intents have to declare their ability to handle a specific intent by declaring intent filters.**
3. **A component can declare any number of Intent Filters.**
4. **There can be more than one component that declares the same Intent Filters and hence can respond to the same implicit intent. In that case, the user is presented both the component options and he can choose which one he wants to continue with.**
5. **We can set priorities for the intent filters to ensure the order of responses.**

**The Implicit Intent Activity creates an implicit intent object contacts. This intent object's component is not set. However, the action is set to android.content.intent.ACTION\_VIEW and the data's URI is set to People.CONTENT\_URI. Such an intent matches with the intent filter declared by the view contacts native activity.**

**Here is the relevant piece of code for the same:**

**Button b=(Button)findViewById(R.id.ViewContacts);**

**b.setOnClickListener(new OnClickListener() {**

**public void onClick(View v) {**

**Intent i= new Intent(android.content.Intent.ACTION\_VIEW, Uri.parse("http://www.facebook.com"));**

**startActivity(i);**

**}**

**});**

**In this manner many of the native applications can be seamlessly invoked as one of the activities in our applications through implicit intents.**

**6. Program to work with database (create, insert ,delete ,update ,select operations)**

**A) INTRODUCING SQLite**

***SQLite* is a well-regarded relational database management system (RDBMS). It is:**

**➤Open-source**

**➤Standards-compliant**

**➤Lightweight**

**➤Single-tier**

**It has been implemented as a compact C library that’s included as part of the Android software stack.By being implemented as a library, rather than running as a separate ongoing process, each SQLitedatabase is an integrated part of the application that created it. This reduces external dependencies, minimizes latency, and simplifies transaction locking and synchronization.**

**SQLite has a reputation for being extremely reliable and is the database system of choice for many consumer electronic devices, including several MP3 players, the iPhone, and the iPod Touch. Lightweight and powerful, SQLite differs from many conventional database engines by loosely typing each column, meaning that column values are not required to conform to a single type. Instead, each value is typed individually for each row. As a result, type checking isn’t necessary when assigning or extracting values from each column within a row.**

**B) Introducing the SQLiteOpenHelper**

**SQLiteOpenHelper is an abstract class used to implement the best practice pattern for creating, opening, and upgrading databases. By implementing an SQLite Open Helper you hide the logic used to decide if a database needs to be created or upgraded before it’s opened. the SQLiteOpenHelper class by overriding the constructor, onCreate, and onUpgrade methods to handle the creation of a new database and upgrading to a new version, respectively.**

**To use an implementation of the helper class, create a new instance, passing in the context, database name, and current version, and a CursorFactory (if you’re using one). Call getReadableDatabase or getWritableDatabase to open and return a readable/writable instance of the underlying database. A call to getWritableDatabase can fail because of disk space or permission issues, so it’s good practice to provide fallback to the getReadableDatabase method.**

**In particular, the DATABASE\_CREATE constant contains the SQL statement for creating the contacts table within the MyDB database.**

**Within the DBAdapter class, you also extend the SQLiteOpenHelper class, which is a helper class in Android to manage database creation and version management. In particular, you override the onCreate() and onUpgrade() methods:**

**The onCreate() method creates a new database if the required database is not present.**

**The onUpgrade() method is called when the database needs to be upgraded. This is achieved by checking the value defined in the DATABASE\_VERSION constant. For this implementation of the onUpgrade() method, you simply drop the table and create it again.**

**C) Introducing SQLite Databases**

**Using SQLite you can create independent relational databases for your applications. Use them to store and manage complex, structured application data. Android databases are stored in the /data/data/<package\_name>/databases folder on your device (or emulator). By default all databases are private, accessible only by the application that created them. Database design is a big topic that deserves more thorough coverage than is possible within this book. It is worth highlighting that standard database best practices still apply in Android. In particular, when you’re creating databases for resource-constrained devices (such as mobile phones), it’s important to normalize your data to reduce redundancy.**

**SQLite Database**

**For more complex data structures, a database provides a quicker and more flexible access method than flat files or shared preferences. Android provides a built-in database called SQLite that provides full relational database capability utilizing SQL commands. Each application that uses SQLite has its own instance of the database, which is by default accessible only from the application itself. The database is stored in the /data/data/ <package\_name>/databases folder of an Android device. A Content Provider can be used to share the database information between applications. The different steps for utilizing**

**SQLite are**

**1. Create a database.**

**2. Open the database.**

**3. Create a table.**

**4. Create an insert interface for datasets.**

**5. Create a query interface for datasets.**

**6. Close the database.**

**The next recipe provides a general method to accomplish these steps.**

**Recipe: Creating a Separate Database Package**

**D) CURSORS AND CONTENT VALUES**

**ContentValues are used to insert new rows into tables. Each Content Values object represents a single table row as a map of column names to values. Queries in Android are returned as Cursor objects. Rather than extracting and returning a copy of the result values, Cursors are pointers to the result set within the underlying data. Cursors provide a managed way of controlling your position (row) in the result set of a database query. The Cursor class includes a number of navigation functions including, but not limited to, the following:**

**➤moveToFirst Moves the cursor to the first row in the query result**

**➤moveToNext Moves the cursor to the next row**

**➤moveToPrevious Moves the cursor to the previous row**

**➤getCount Returns the number of rows in the result set**

**➤getColumnName Returns the name of the specified column index**

**➤getColumnNames Returns a string array of all the column names in the current Cursor**

**➤moveToPosition Moves the Cursor to the specified row**

**➤getPosition Returns the current Cursor position**

**Android provides a convenient mechanism for simplifying the management of Cursors within your Activities. The startManagingCursor method integrates the Cursor’s lifetime into the calling Activity’s. When you’ve finished with the Cursor, call stopManagingCursor to do just that.**

**Later in this chapter you’ll learn how to query a database and how to extract specific row/column**

**values from the resulting Cursors.**

**Pre-Creating the Database**

**In real-life applications, sometimes it would be more efficient to pre-create the database at design time rather than run time. To pre-create a SQLite database, you can use many of the free tools available on the Internet. One such tool is the SQLite Database Browser, which is available free for the different platforms (**<http://sourceforge.net/projects/sqlitebrowser/>**). Once you have installed the SQLite Database Browser, you can create a database visually. Figure 6-13 shows that I have created a contacts table with the fields indicated.**

**7. Program to show how to use notifications**

**INTRODUCING NOTIFICATIONS**

**Your applications can use Notifications to alert users without using an Activity. Notifications are**

**handled by the Notification Manager, and currently have the ability to:**

**➤ Create new status bar icons**

**➤ Display additional information (and launch an Intent) in the extended status bar window**

**➤ Flash the lights/LEDs**

**➤ Vibrate the phone**

**➤ Sound audible alerts (ringtones, Media Store audio)**

**Using Notifications is the preferred way for invisible application components (Broadcast Receivers, Services, and inactive Activities) to alert users that events have occurred that may require attention. They are also used to indicate ongoing background Services — particularly Services that have been set to foreground priority.**

**As a user interface metaphor, Notifications are particularly well suited to mobile devices. It’s likely that your users will have their phones with them at all times but quite unlikely that they will be paying attention to them, or your application, at any given time. Generally users will have several applications open in the background, and they won’t be paying attention to any of them. In this environment it’s important that your applications be able to alert users when specific events occur that require their attention. Notifications can be persisted through insistent repetition, being marked ongoing, or simply by displaying an icon on the status bar. Status bar icons can be updated regularly or expanded to show additional information using the expanded status bar window .**

**Introducing the Notification Manager**

**The *Notification Manager* is a system Service used to handle Notifications. Get a reference to it using the getSystemService method.**

**NotificationManager nm = (NotificationManager) getSystemService (NOTIFICATION\_ SERVICE);**

**Using the Notification Manager you can trigger new Notifications, modify existing ones, or remove those that are no longer required.**

**Creating Notifications**

**Android offers a number of ways to convey information to users using Notifications.**

**1. The status bar icon**

**2. The extended notification status drawer**

**3. Additional phone effects such as sound and vibration**

**This section will examine the first two while later in this chapter you’ll learn how to enhance Notifications using various properties on the Notification object to flash the device LEDs, vibrate the phone, and play audio.**

**Creating a Notification and Configuring the Status Bar Icon**

**Start by creating a new Notification object, passing in the icon to display in the status bar, along with the status bar ticker text and the time of this Notification.**

**Creating a Notification**

**// Choose a drawable to display as the status bar icon**

**int icon = R.drawable.icon;**

**// Text to display in the status bar when the notification is launched**

**String tickerText = "Notification";**

**// The extended status bar orders notification in time order**

**long when = System.currentTimeMillis();**

**Notification notification = new Notification(icon, tickerText, when);**

**The ticker text will scroll along the status bar when the Notification is fired.**

**Configuring the Extended Status Notification Display**

**You can configure the appearance of the Notification within the extended status window in two ways:**

**1. Use the setLatestEventInfo method to update the details displayed in the standard extended**

**status Notification display.**

**2. Set the contentView and contentIntent properties to assign a custom UI for the extended**

**status display using a Remote View.**

**The simplest technique is to use the setLatestEventInfo method to populate the default status window layout. The standard extended status window layout shows the icon and time defined in the constructor,along with a title and a details string.**

**Notifications often represent a request for action or attention, so you can specify a PendingIntent that will be fired if a user clicks the Notification item. In most cases that Intent should open your application and navigate to the Activity that provides context for the notification (e.g., showing an unread SMS or e-mail message).**

**Setting Notification values**

**Context context = getApplicationContext();**

**// Text to display in the extended status window**

**String expandedText = "Extended status text";**

**// Title for the expanded status**

**String expandedTitle = "Notification Title";**

**// Intent to launch an activity when the extended text is clicked**

**Intent intent = new Intent(this, MyActivity.class);**

**PendingIntent launchIntent = PendingIntent.getActivity(context, 0, intent, 0);**

**notification.setLatestEventInfo(context, expandedTitle, expandedText, launchIntent);**

**It’s good form to use one Notification icon to represent multiple instances of the same event (e.g., receiving multiple SMS messages). To do this, update the values set by setLatestEventInfo to reflect the most recent message (or a summary of multiple messages) and re-trigger the Notification to update the display values.**

1. [Android Design: Notifications](http://developer.android.com/design/patterns/notifications.html)

**A notification is a message you can display to the user outside of your application's normal UI. When you tell the system to issue a notification, it first appears as an icon in the notification area. To see the details of the notification, the user opens the notification drawer. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.**

**Figure 1. Notifications in the notification area.**

**Figure 2. Notifications in the notification drawer.**

**Figure 3. Notification in normal view.**

**The callouts in the illustration refer to the following:**

1. **Content title**
2. **Large icon**
3. **Content text**
4. **Content info**
5. **Small icon**
6. **Time that the notification was issued. You can set an explicit value with**[setWhen()](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html" \l "setWhen(long))**; if you don't it defaults to the time that the system received the notification.**

**Big view**

**A notification's big view appears only when the notification is expanded, which happens when the notification is at the top of the notification drawer, or when the user expands the notification with a gesture. Expanded notifications are available starting with Android 4.1.**

**The following screenshot shows an inbox-style notification:**

**Figure 4. Big view notification.**

**Notice that the big view shares most of its visual elements with the normal view. The only difference is callout number 7, the details area. Each big view style sets this area in a different way. The available styles are:**

**Big picture style The details area contains a bitmap up to 256 dp tall in its detail section.**

**Big text style**

**Displays a large text block in the details section.**

**Inbox style**

**Displays lines of text in the details section.**

**All of the big view styles also have the following content options that aren't available in normal view:**

**Big content title**

**Allows you to override the normal view's content title with a title that appears only in the expanded view.**

**Summary text**

**Allows you to add a line of text below the details area.**

**Applying a big view style to a notification is described in the section**[Applying a big view style to a notification](http://developer.android.com/guide/topics/ui/notifiers/notifications.html#ApplyStyle)**.**

**Creating a Notification**

**You specify the UI information and actions for a notification in a**[NotificationCompat.Builder](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html)**object. To create the notification itself, you call**[NotificationCompat.Builder.build()](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html" \l "build())**, which returns a**[Notification](http://developer.android.com/reference/android/app/Notification.html)**object containing your specifications. To issue the notification, you pass the**[Notification](http://developer.android.com/reference/android/app/Notification.html)**object to the system by calling**[NotificationManager.notify()](http://developer.android.com/reference/java/lang/Object.html" \l "notify())**.**

**Required notification contents**

**Main.xml:**

**<?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=*"@string/hello"* />**

**<Button**

**android:id=*"@+id/button1"***

**android:layout\_width=*"match\_parent"***

**android:layout\_height=*"wrap\_content"***

**android:text=*"Display Notification"* />**

**</LinearLayout>**

**Notificationview.xml:**

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

**<LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"***

**android:layout\_width=*"match\_parent"***

**android:layout\_height=*"match\_parent"***

**android:orientation=*"vertical"* >**

**<TextView**

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

**android:layout\_width=*"wrap\_content"***

**android:layout\_height=*"wrap\_content"***

**android:text=*"This is the Checked notification layout"* />**

**</LinearLayout>**

**NotificationActivity.java:**

**package com.notification;**

**import android.app.Activity;**

**import android.app.Notification;**

**import android.app.NotificationManager;**

**import android.app.PendingIntent;**

**import android.content.Intent;**

**import android.os.Bundle;**

**import android.view.View;**

**import android.view.View.OnClickListener;**

**import android.widget.Button;**

**public class NotificationActivity extends Activity {**

**/\*\* Called when the activity is first created. \*/**

**int notificationid=1;**

**@Override**

**public void onCreate(Bundle savedInstanceState) {**

**super.onCreate(savedInstanceState);**

**setContentView(R.layout.main);**

**Button b=(Button)findViewById(R.id.button1);**

**b.setOnClickListener(new OnClickListener() {**

**@Override**

**public void onClick(View v) {**

**// TODO Auto-generated method stub**

**displaynotification();**

**}**

**});**

**}**

**private void displaynotification() {**

**// TODO Auto-generated method stub**

**Intent i=new Intent (this, notificationview.class);**

**i.putExtra("notificationid", notificationid);**

**PendingIntent pen=PendingIntent.getActivity**

**(this, 0, i,0);**

**NotificationManager nm=(NotificationManager)**

**getSystemService(NOTIFICATION\_SERVICE);**

**Notification n=new Notification(R.drawable.ic\_launcher,**

**"Reminder:Ur meeting stars within 5 min",**

**System.currentTimeMillis());**

**CharSequence from="MEETING";**

**CharSequence message="UR Meeting is on 4th floor at 3:00 clock";**

**n.setLatestEventInfo(this, from, message, pen);**

**nm.notify(notificationid,n);**

**}**

**}**

**NotificationView.java:**

**package com.notification;**

**import android.app.Activity;**

**import android.app.NotificationManager;**

**import android.os.Bundle;**

**public class notificationview extends Activity**

**{**

**protected void onCreate(Bundle savedInstanceState) {**

**super.onCreate(savedInstanceState);**

**setContentView(R.layout.notificationview);**

**NotificationManager nm=(NotificationManager)**

**getSystemService(NOTIFICATION\_SERVICE);**

**nm.cancel(getIntent().getExtras().getInt("notificationid"));**

**}**

**}**

**Manifest.xml:**

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

**<manifest xmlns:android=*"http://schemas.android.com/apk/res/android"***

**package=*"com.notification"***

**android:versionCode=*"1"***

**android:versionName=*"1.0"* >**

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

**<application**

**android:icon=*"@drawable/ic\_launcher"***

**android:label=*"@string/app\_name"* >**

**<activity**

**android:name=*".NotificationActivity"***

**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=*".notificationview"***

**android:label=*"@string/app\_name"* >**

**<intent-filter>**

**<action android:name=*"android.intent.action.MAIN"* />**

**<category android:name=*"android.intent.category.DEFAULT"* />**

**</intent-filter>**

**</activity>**

**</application>**

**</manifest>**