**Chapter 3. DISCRIPTION OF TOOLS**

**3.1 Android Studio**

Android Studio provides the fastest tools for building apps on every type of Android device. World-class code editing, debugging, performance tooling, a flexible build system, and an instant build/deploy system all allow you to focus on building unique and high quality apps.

Instant Run

Android Studio's Instant Run feature pushes code and resource changes to your running app. It intelligently understands the changes and often delivers them without restarting your app or rebuilding your APK, so you can see the effects immediately.

Intelligent code editor

The code editor helps you write better code, work faster, and be more productive by offering advanced code completion, refactoring, and code analysis. As you type, Android Studio provides suggestions in a dropdown list. Simply press Tab to insert the code.

Fast and feature-rich emulator

The Android Emulator installs and starts your apps faster than a real device and allows you to prototype and test your app on various Android device configurations: phones, tablets, Android Wear, and Android TV devices. You can also simulate a variety of hardware features such as GPS location, network latency, motion sensors, and multi-touch input.

Configure Builds Without Limits

Android Studio's project structure and gradle -based builds provide the flexibility you need to generate APKs for all device types.

Robust and flexible build system

Android Studio offers build automation, dependency management, and customizable build configurations. You can configure your project to include local and hosted libraries, and define build variants that include different code and resources, and apply different code shrinking and app signing configurations.

Designed for teams

Android Studio integrates with version control tools, such as GitHub and Subversion, so you can keep your team in sync with project and build changes. The open source Gradle build system allows you to tailor the build to your environment and run on a continuous integration server such as Jenkins.

Optimized for all Android devices

Android Studio provides a unified environment where you can build apps for Android phones, tablets, Android Wear, Android TV, and Android Auto. Structured code modules allow you to divide your project into units of functionality that you can independently build, test, and debug.

Code with Confidence

At every step, Android Studio helps ensure that you're creating the best code possible.

Code templates and sample apps

Android Studio includes project and code templates that make it easy to add well-established patterns such as a navigation drawer and view pager. You can start with a code template or even right-click an API in the editor and select Find Sample Code to search for examples. Moreover, you can import fully functional apps from GitHub, right from the Create Project screen.

Lintelligence

Android Studio provides a robust static analysis framework and includes over 280 different lint checks across the entirety of your app. Additionally, it provides several quick fixes that help you address issues in various categories, such as performance, security, and correctness, with a single click.

Testing tools & frameworks

Android Studio provides extensive tools to help you test your Android apps with JUnit 4 and functional UI test frameworks. With Espresso Test Recorder, you can generate UI test code by recording your interactions with the app on a device or emulator. You can run your tests on a device, an emulator, a continuous integration environment, or in Firebase Test Lab.

Create Rich and Connected Apps

Android Studio knows not all code is written in Java and not all code runs on the user's device.

C++ and NDK support

Android Studio fully supports editing C/C++ project files so you can quickly build JNI components in your app. The IDE provides syntax highlighting and refactoring for C/C++, and an LLDB-based debugger that allows you to simultaneously debug your Java and C/C++ code. The build tools can also execute your CMake and ndk-build scripts without any modification and then add the shared objects to your APK.

Firebase and Cloud integration

The Firebase Assistant helps you connect your app to Firebase and add services such as Analytics, Authentication, Notifications and more with step- by-step procedures right inside Android Studio. Built-in tools for Google Cloud Platform also help you integrate your Android app with services such as Google Cloud Endpoints and project modules specially-designed for Google App Engine.

Eliminate Tiresome Tasks

Android Studio provides GUI tools that simplify the less interesting parts of app development.

Layout Editor

When working with XML layout files, Android Studio provides a drag-and-drop visual editor that makes it easier than ever to create a new layout. The Layout Editor was built in unison with the Constraint Layout API, so you can quickly build a layout that adapts to different screen sizes by dragging views into place and then adding layout constraints with just a few clicks.

APK Analyzer

You can use the APK Analyzer to easily inspect the contents of your APK. It reveals the size of each component so you can identify ways to reduce the overall APK size. It also allows you preview packaged assets, inspect the DEX files to troubleshoot multidex issues, and compare the differences between two APKs.

Vector Asset Studio

Android Studio makes it easy to create a new image asset for every density size. With Vector Asset Studio, you can select from Google-provided material design icons or import an SVG or PSD file. Vector Asset Studio can also generate bitmap files for each screen density to support older versions of Android that don't support the Android vector drawable format.

Translations Editor

The Translations Editor gives you a single view of all of your translated resources, making it easy to change or add translations, and to find missing translations without opening each version of the strings.xml file. It even provides a link to order translation services.

**3.1.1 JDK**

The Java Development Kit (JDK) is an implementation of either one of the Java Platform, Standard Edition, Java Platform, Enterprise Edition, or Java Platform, Micro Edition platforms released by Oracle Corporation in the form of a binary product aimed at Java developers on Solaris, Linux, macOS or Windows. The JDK includes a private JVM and a few other resources to finish the development of a Java Application. Since the introduction of the Java platform, it has been by far the most widely used Software Development Kit (SDK). On 17 November 2006, Sun announced that they would release it under the GNU General Public License (GPL), thus making it free software. This happened in large part on 8 May 2007, when Sun contributed the source code to the OpenJDK

The JDK has as its primary components a collection of programming tools, including:

appletviewer – this tool can be used to run and debug Java applets without a web browser

apt – the annotation-processing tool

extcheck – a utility that detects JAR file conflicts

idlj – the IDL-to-Java compiler. This utility generates Java bindings from a given Java IDL file.

jabswitch – the Java Access Bridge. Exposes assistive technologies on Microsoft Windows systems.

java – the loader for Java applications. This tool is an interpreter and can interpret the class files generated by the javac compiler. Now a single launcher is used for both development and deployment. The old deployment launcher, jre, no longer comes with Sun JDK, and instead it has been replaced by this new java loader.

javac – the Java compiler, which converts source code into Java bytecode

javadoc – the documentation generator, which automatically generates documentation from source code comments

jar – the archiver, which packages related class libraries into a single JAR file. This tool also helps manage JAR files.

javafxpackager – tool to package and sign JavaFX applications

jarsigner – the jar signing and verification tool

javah – the C header and stub generator, used to write native methods

javap – the class file disassembler

javaws – the Java Web Start launcher for JNLP applications

JConsole – Java Monitoring and Management Console

jdb – the debugger

jhat – Java Heap Analysis Tool (experimental)

jinfo – This utility gets configuration information from a running Java process or crash dump. (experimental)

jmap Oracle jmap - Memory Map– This utility outputs the memory map for Java and can print shared object memory maps or heap memory details of a given process or core dump. (experimental)

jmc – Java Mission Control

jps – Java Virtual Machine Process Status Tool lists the instrumented HotSpot Java Virtual Machines (JVMs) on the target system. (experimental)

jrunscript – Java command-line script shell.

jstack – utility that prints Java stack traces of Java threads (experimental)

jstat – Java Virtual Machine statistics monitoring tool (experimental)

jstatd – jstat daemon (experimental)

keytool – tool for manipulating the keystore

pack200 – JAR compression tool

policytool – the policy creation and management tool, which can determine policy for a Java runtime, specifying which permissions are available for code from various sources.

VisualVM – visual tool integrating several command-line JDK tools and lightweight[clarification needed] performance and memory profiling capabilities

wsimport – generates portable JAX-WS artifacts for invoking a web service.

xjc – Part of the Java API for XML Binding (JAXB) API. It accepts an XML schema and generates Java classes.

Experimental tools may not be available in future versions of the JDK.

The JDK also comes with a complete Java Runtime Environment, usually called a private runtime, due to the fact that it is separated from the "regular" JRE and has extra contents. It consists of a Java Virtual Machine and all of the class libraries present in the production environment, as well as additional libraries only useful to developers, such as the internationalization libraries and the IDL libraries.

Copies of the JDK also include a wide selection of example programs demonstrating the use of almost all portions of the Java API.

**3.2. Database**

SQLite is a opensource SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation.

SQLite supports all the relational database features. In order to access this database, you don't need to establish any kind of connections for it like JDBC,ODBC e.t.c

**Database - Package**

The main package is android.database.sqlite that contains the classes to manage your own databases

**Database - Creation**

In order to create a database you just need to call this method openOrCreateDatabase with your database name and mode as a parameter. It returns an instance of SQLite database which you have to receive in your own object.Its syntax is given below

SQLiteDatabase mydatabase = openOrCreateDatabase("your database name",MODE\_PRIVATE,null);

Apart from this , there are other functions available in the database package , that does this job. They are listed below

1. openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorHandler)

This method only opens the existing database with the appropriate flag mode. The common flags mode could be OPEN\_READWRITE OPEN\_READONLY

1. openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)

It is similar to the above method as it also opens the existing database but it does not define any handler to handle the errors of databases

1. openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)

It not only opens but create the database if it not exists. This method is equivalent to openDatabase method.

1. openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)

This method is similar to above method but it takes the File object as a path rather then a string. It is equivalent to file.getPath()

**Database - Insertion**

we can create table or insert data into table using execSQL method defined in SQLiteDatabase class. Its syntax is given below

mydatabase.execSQL("CREATE TABLE IF NOT EXISTS TutorialsPoint(Username VARCHAR,Password VARCHAR);");

mydatabase.execSQL("INSERT INTO TutorialsPoint VALUES('admin','admin');");

This will insert some values into our table in our database. Another method that also does the same job but take some additional parameter is given below

1. execSQL(String sql, Object[] bindArgs)

This method not only insert data , but also used to update or modify already existing data in database using bind arguments

**Database - Fetching**

We can retrieve anything from database using an object of the Cursor class. We will call a method of this class called rawQuery and it will return a resultset with the cursor pointing to the table. We can move the cursor forward and retrieve the data.

Cursor resultSet = mydatbase.rawQuery("Select \* from TutorialsPoint",null);

resultSet.moveToFirst();

String username = resultSet.getString(0);

String password = resultSet.getString(1);

There are other functions available in the Cursor class that allows us to effectively retrieve the data. That includes

1. getColumnCount()

This method return the total number of columns of the table.

1. getColumnIndex(String columnName)

This method returns the index number of a column by specifying the name of the column

1. getColumnName(int columnIndex)

This method returns the name of the column by specifying the index of the column

1. getColumnNames()

This method returns the array of all the column names of the table.

1. getCount()

This method returns the total number of rows in the cursor

1. getPosition()

This method returns the current position of the cursor in the table

1. isClosed()

This method returns true if the cursor is closed and return false otherwise

**Database - Helper class**

For managing all the operations related to the database , an helper class has been given and is called SQLiteOpenHelper. It automatically manages the creation and update of the database. Its syntax is given below

public class DBHelper extends SQLiteOpenHelper {

public DBHelper(){

super(context,DATABASE\_NAME,null,1);

}

public void onCreate(SQLiteDatabase db) {}

public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {}

}

Example

Here is an example demonstrating the use of SQLite Database. It creates a basic contacts applications that allows insertion, deletion and modification of contacts.

To experiment with this example, you need to run this on an actual device on which camera is supported.

1 You will use Android studio to create an Android application under a package com.example.sairamkrishna.myapplication.

2 Modify src/MainActivity.java file to get references of all the XML components and populate the contacts on listView.

3 Create new src/DBHelper.java that will manage the database work

4 Create a new Activity as DisplayContact.java that will display the contact on the screen

5 Modify the res/layout/activity\_main to add respective XML components

6 Modify the res/layout/activity\_display\_contact.xml to add respective XML components

7 Modify the res/values/string.xml to add necessary string components

8 Modify the res/menu/display\_contact.xml to add necessary menu components

9 Create a new menu as res/menu/mainmenu.xml to add the insert contact option

10 Run the application and choose a running android device and install the application on it and verify the results.