Chapter 3: DESCRIPTION OF TOOLS

3.1 Android Studio

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on Jet Brains' IntelliJ IDEA software and designed specifically for development.[1] It is available for download on Windows, machos and Linux based operating systems.[2] It is a replacement for the Eclipse Android Development Tools (ADT) as primary IDE for native Android application development. Android Studio was announced on May 16, 2013 at the Google I/O conference. [3] It was in early access preview stage starting from version 0.1 in May 2013, then entered beta stage starting from version 0.8 which was released in June 2014. The first stable build was released in December 2014, starting from version 1.0. The current stable version is 3.1 released in March 2018.[5] The Android Studio IDE is free to download and use. It has a rich UI development environment with templates to give new developers a launching pad into Android development. Developers will find that Studio gives them the tools to build phone and tablet solutions as well as emerging technology solutions for Android TV, Android Wear, Android Auto, Glass and additional contextual models.[7]

Android Studio is intended to be used by development teams as small as one person or as large as global teams. The Android Studio IDE can be linked to larger teams with GIT or similar version control services for larger teams.[7] Mature Android developers will find tools that are necessary for large teams to deliver solutions rapidly to their customers. Android solutions can be developed using either Java or C++ in Android Studio. The workflow for Android Studio is built around the concept of continuous integration. Continuous Integration allows for teams to test their code each and every time a developer checks in their work. Issues can be captured and reported to the team immediately. The concept of continuously checking code provides actionable feedback to the developers with the goal of releasing versions of a mobile solution faster to the Google Play App Store.[8] To this end, there is rigorous support for LINT tools, Pro-Guard and App Signing tools. Performance tools provide access to view how well an Android application package file (APK) is going.[10] the performance and profiling tools display a color-coded image.

The GPU rendering shows how well your app does in maintaining Google's 16-ms-perframe benchmark.[1] Memory tools visualize where and when your app will use too much system RAM and when Garbage collection occurs, Battery Analysis tools present how much drain you're placing on a device.[2] Android Studio supports Google App Engine for quick cloud integration of new APIs and features. You will find support for many APIs directly in Android Studio such as Google Play, Android Pay and Health. There is support for all platforms of android starting with Android 1.6 and later. There are variants of Android that are significantly different to the Google Android version. The most popular is Amazon's Fire OS. Android Studio can be used to build Amazon Fire OS APKs using these guidelines. Android Studio is replacing Google's support for Eclipse ADT.

Features

The following features are provided in the current stable version

- Cradle-based build support.
- Android-specific refactoring and quick fixes.
- Lint tools to catch performance, usability, version compatibility and other problems
- ProGuard integration and app-signing capabilities.
- Template-based wizards to create common Android designs and components.
- A rich layout editor that allows users to drag-and-drop UI components, option to preview layouts on multiple screen configurations.
- Support for building Android Wear apps.[5]
- Built-in support for Google Cloud Platform, enabling integration with Firebase Cloud Messaging (Earlier 'Google Cloud Messaging') and Google App Engine
- Android Virtual Device (Emulator) to run and debug apps in the Android studio.

Android Studio supports all the same programming languages of IntelliJ, and PyCharm e.g. Python, and Kotlin; and Android Studio 3.0 supports "Java 7 language features and a subset of Java 8 language features that vary by platform version." External projects backport some Java 9 features.

Android Resources Organizing & Accessing

There are many more items which you use to build a good Android application. Apart from coding for the application, you take care of various other resources like static content that your code uses, such as bitmaps, colors, layout definitions, user interface strings, animation instructions, and more.These resources are always maintained separately in various subdirectories under res/ directory of the project. This tutorial will explain you how you can organize your application resources, specify alternative resources and access them in your applications.

Alternative Resources

Your application should provide alternative resources to support specific device configurations. For example, you should include alternative drawable resources (i.e. Images) for different screen resolution and alternative string resources for different languages.[4] At runtime, Android detects the current device configuration and loads the appropriate resources for your application. To specify configuration-specific alternatives for a set of resources, follow the following Create a new directory in res/ named in the form <resources name>-<config_qualifier>. Here resources name will be any of the resources mentioned in the above table, like layout, drawable etc. The qualifier will specify an individual configuration for which these resources are to be used. You can check official documentation for a complete list of qualifiers for different type of resources.Save the respective alternative resources in this new directory. The resource files must be named exactly the same as the default resource files as shown in the below example, but these files will have content specific to the alternative. For example though image file name will be same but for high resolution screen, its resolution will be high.

Android - Environment Setup

You will be glad to know that you can start your Android application development on either of the following operating systems −

- Microsoft Windows XP or later version.
- Mac OS X 10.5.8 or later version with Intel chip.
- Linux including GNU C Library 2.7 or later.

Second point is that all the required tools to develop Android applications are freely available and can be downloaded from the Web. Following is the list of software's you will need before you start your Android application programming. Java JDK5 or later version. Android Studio Here last two components are optional and if you are working on Windows machine then these components make your life easy while doing Java based application development. So let us have a look how to proceed to set required environment.

3.1.1 Java Development Kit (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[1] 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 Opened. Java Development Kit (JDK) is a bundle of software components that is used to develop Java based applications. JDK is an implementation of either of Java SE, Java EE or Java ME. Usually, learners start from JDK implementation of Java SE to learn core Java features, which is also known as Java SDK. JDK includes the JRE, set of API classes, Java compiler, Web start and additional files needed to write Java applets and applications. Java Development Kit is a bundle of the following software components that are needed to develop Java based applications.

Java Compiler

Java compiler is **java** tool located in /bin folder of the JDK installation directory. The **javac** tool (accessed using javac command) reads class and interface definitions, written in the Java programming language, and compiles them into bytecode class files. It can also process annotations in Java source files and classes. There are two ways to pass source code file names to javac:

- For a small number of source files, simply list the file names on the command line separated by blank space. For example: D:\JavaPrograms>javac SelectionSortDemo.java SequentialSearchDemo.java

- For a large number of source files, list the file names in a file, separated by blanks or line breaks. Then use the list file name on the javac command line, preceded by and character. For an example, store three source file names SelectionSortDemo.java, SequentialSearchDemo.java, SystemOutPrintlnDemo.java in a file named source-file-list and then supply following command in order to compile the source code files stored in source-file list.

```
D:\JavaPrograms>javac @source-file-list
```

Java Interpreter

Java interpreter is used to interpret the .class Java files that have been compiled by Java compiler (javac). Java interpreter is accessed using java command. The java command starts a Java application. It does this by starting a Java runtime environment, loading a specified class, and calling that class's main method. The method must be declared public and static, it must not return any value, and it must accept a String array as a parameter. The method declaration has the following form:

public static void main(String[] args)

By default, the first argument without an option is the name of the class to be called. A fully qualified class name should be used. If the -jar option is specified, then the first non-option argument is the name of a JAR file containing class and resource files for the application, with the startup class indicated by the Main-Class manifest header. The Java runtime searches for the startup class, and other classes used, in three sets of locations: the bootstrap class path, the installed extensions, and the user class path. Non-option arguments after the class name or JAR file name are passed to the main function.

Java Disassembler

The javap command is the disassembly tool of JDK that disassembles one or more class files. Its output depends on the options used. If no options are used, javap prints out the package, protected, and public fields and methods of the classes passed to it. The javap prints its output to stdout.

Java Header File Generator

Java Header File Generator (javah command-line tool) generates C header and source files that are needed to implement native methods. The generated header and source files are used by C programs to reference an object's instance variables from native source code. The .h file contains a struct definition whose layout parallels the layout of the corresponding class.[1] The fields in the struct correspond to instance variables in the class. The name of the header file and the structure declared within it are derived from the name of the class.[3] If the class passed to javah is inside a package, the package name is prepended to both the header file name and the structure name. Underscores (_) are used as name delimiters. By default javah creates a header file for each class listed on the command line and puts the files in the current directory. Use the -stubs option to create source files. Use the -o option to concatenate the results for all listed classes into a single file. The new native method interface, Java Native Interface (JNI), does not require header information or stub files. The javah tool can still be used to generate native method function prototypes needed for JNI-style native methods. The javah tool produces JNI-style output by default, and places the result in the .h file.

Java Documentation

Java Documentation helps to maintain code. The javadoc tool comes as part of Java Development Kit that parses the declarations and documentation comments in a set of Java source files and produces a corresponding set of HTML pages describing (by default) the public and protected classes, nested classes (but not anonymous inner classes), interfaces, constructors, methods, and fields. You can use it to generate the API (Application Programming Interface) documentation or the implementation documentation for a set of source files.

You can run the javadoc tool on entire packages, individual source files, or both. When documenting entire packages, you can either use -subpackages for traversing recursively down from a top-level directory, or pass in an explicit list of package names. When documenting individual source files, you pass in a list of source (.java) file names.

Java Debugger

The Java Debugger, jdb, is a simple command-line debugger for Java classes. It is a demonstration of the Java Platform Debugger Architecture that provides inspection and debugging of a local or remote Java Virtual Machine.

Java Applet Viewer

This is used to view the Java applets. The appletviewer command connects to the documents or resources designated by urls and displays each applet referenced by the documents in its own window.

3.2 Firebase Database

Firebase is a backend platform for building Web, Android and IOS applications. It offers real time database, different APIs, multiple authentication types and hosting platform. This is an introductory tutorial, which covers the basics of the Firebase platform and explains how to deal with its various components and sub-components. This tutorial is directed towards developers in need for a simple, user-friendly backend platform. After you finish this tutorial, you will be familiar with the Firebase Web Platform. You can also use this as a reference in your future development. [5] This tutorial is intended to make you comfortable in getting started with the Firebase backend platform and its various functions. You will need some JavaScript knowledge to be able to follow this tutorial. Knowledge about some backend platform is not necessary, but it could help you to understand the various Firebase concepts.

All Firebase Realtime Database data is stored as JSON objects. You can think of the database as a cloud-hosted JSON tree. Unlike a SQL database, there are no tables or records. When you add data to the JSON tree, it becomes a node in the existing JSON structure with an associated key. You can provide your own keys, such as user IDs or semantic names, or they can be provided for you using childByAutoId.

Because the Firebase Realtime Database allows nesting data up to 32 levels deep, you might be tempted to think that this should be the default structure. However, when you fetch data at a location in your database, you also retrieve all of its child nodes. In addition, when you grant someone read or write access at a node in your database, you also grant them access to all data under that node. Therefore, in practice, it's best to keep your data structure as flat as possible.