ЎЗБЕКИСТОН РЕСПУБЛИКАСИ АЛОҚА, АХБОРОТЛАШТИРИШ ВА ТЕЛЕКОММУНИКАЦИЯ ТЕХНОЛОГИЯЛАРИ ДАВЛАТ ҚЎМИТАСИ

ТОШКЕНТ АХБОРОТТЕХНОЛОГИЯЛАРИ УНИВЕРСИТЕТИ ФАРҒОНА ФИЛИАЛИ

«АХБОРОТ ТЕХНОЛОГИЯЛАРИ» КАФЕДРАСИ

«ҲИМОЯГА»

Кафедра мудири

_____

« ____ »_____2014 й

The place of mobile technologies in development of the society

МАВЗУСИДА

МАЛАКАВИЙ БИТИРУВ ИШИ

БИТИРУВЧИ: Эргашев. Р

612-10 гуруҳ талабаси

Фарғона – 2014 йил

CONTENTS

Intoduction

On June 27, 2011, President Islam Karimov gave a speech praising internet freedom in Uzbekistan. "We fully support the desire of our fellow citizens to freely use the internet," he proclaimed in the public address, given in honor of Uzbekistan‘s Press and Media Day. "I want to repeat once again: we absolutely do not accept the establishment of any walls or restrictions in the information world that lead to isolation."

There was a caveat. "At the same time, taking into account events that transpired in near and distant regions, we must not ignore the fact that destructive forces seeking to mislead young people with immature minds and naive views on life seek to exploit the possibilities of the internet for their own selfish purposes, leading to dire consequences, "Karimov continued. The only way to stop Uzbekistan‘s society from being destroyed was to "educate the youth to distinguish from black and white, friend and foe and increase awareness of the spirit of national and universal values." In other words, the Uzbek government was not opposed to the internet so long as the internet did not tell Uzbeks anything the government did not want them to know.

Mobile phones are growing in popularity all around the world. They are ubiquitous tools and accessories in highly wired societies such as Scandinavia and East Asia; several research studies of phone use have been conducted in such technologically developed places. Mobile technology has also become important in the developing world, for example, by allowing it to “leapfrog” and take advantage of advances in information and communication technologies (ICT) without land lines. Many development projects are currently underway to bring information services via mobile phone to emerging societies.

Studying the use of mobile phones in Uzbekistan with consideration to its broader social context suggests that unique patterns of mobile phone use exist. The mobile users of Uzbekistan seem to be adopting mobile phones in a culturally meaningful way. People are able to use the mobile phones as a technological extension of their natural behavior. For instance, parents worry about their

children’s safety, and they are able to use mobile phones to watch over them even when they are out of sight.

The relatively conservative and minimal use of the mobile phone in public may be a reflection of the political environment. In a society where it may be safest not to attract the attention of the police or to share too much information in a public area, mobile phone owners may be deliberately judicious about their use.

The current vision seems to be that mobile communication converges with the increase in local wireless networks. This will lead to a development that was not foreseen some years ago, when a universal mobile telephone system was on the agenda. The convergence will merge the internet and mobile phones as we know them now. It is unlikely that the strict distinction between mobile and stationary devices will persist, or their classifications or standardized patterns of use. This development will also pose increasing challenges for social studies that may have some essential role in bridging vastly expanding technological potential and human life-worlds that set the final limits to what will be considered adequate, appropriate and desirable features of ubiquitous communication.

1.1 The place of mobile technologies in development of the society

The mobile phone (cell phone, U.S.) has been the most rapidly disseminated technology in world history. The first commercially available mobile telephone networks were developed in the 1980s. In 1994, Finland was leading in the relative number of subscribers, which had just exceeded 10% of the population. In 2004, there were almost two billion mobile subscribers world-wide. Most industrially developed countries have a penetration rate well over 70%, and most developing countries are following this swiftly.

A phenomenon of this kind is a sociological wonder, demonstrating the increasing world networking. However, details of this development also show persistent cultural differentiation. The appropriation of mobile phones by no means leads to uniform cultural and social development. Indeed, mobile communication is indexically tied to local circumstances and ways of life that may somehow be affected, enriched or modified by the potential of mobile communication, but local paths of development have not yet been combined.

The mobile communication technologies are the latest products enabling seamless interaction between people and integrating the global network. They may well turn out to be one of those technologies whose impact on the organization of social action and societies and the resulting cultural and political upheavals we may start to appreciate only afterwards. Further, the impact of innovation is always a two-way process. Innovation is created through its appropriation, through the daily practices of the people implementing it. Following current social studies of science, we may conclude that mobile phones do not have any immediate social impact, but that there is interplay between mobile phones and societies that is open to multifaceted developments that I will try to explore by reference to recent literature.

Mobile communication is integrated with everyday practices and identities. It is extrovert and grounded in mundane routines, workspaces and moments of sociability. This embeddedness into the most trivial, tiny and inconsequential daily

matters is also the key to its success. As Roos had already pointed out in 1993 “The mobile telephone rapidly becomes a normal, self-evident part of one's life ("I don't understand how I managed before")”. Anchoring to daily routines and errands creates a dual nature for mobile media, making them both global and intimately local.

The intimate linkage to local cultures and practices makes it understandable why mobile communication takes different directions in different parts of the world. Ito states in her introduction: “The current variability in wireless deployment is not necessarily on its way to becoming standardized toward universal access but is a symptom of fundamentally heterogeneous and resilient sociotechnical formations that vary across lines such as gender, nation, class, institutional location, and age. Our narrative is not of a single technology disseminated to multiple contexts but of the heterogeneous co-constitution of technology across a transnational stage.” At the global level, we can see roughly four broad lines of sociotechnical development in mobile communication: Europe, Industrialized Asia (e.g., Japan, and Korea), North America and developing countries. They all have appropriated mobile communication in distinctive ways, and evolved in different directions.

Scandinavia was leading mobile development at its inception, as it managed to develop the first automated mobile system covering the whole region. Along with the rest of Europe, Scandinavia lost its momentum in the massive, failed investments in “3G” or “UMTS” at the turn of the millennium. Industrialized Asia is currently the leading edge in mobile development. In all, the Japanese and Korean business models and technologies have proved superior in attracting larger segments of consumers into new forms of communication. In contrast, Northern America has been very slow to adopt mobile technologies. Ling points out three factors that have contributed to the tardiness in Northern America: the lack of a joint standard, the selection of a pricing system, and competitive alternative web-based media. In the United States, interoperability has developed slowly as there have been competing standards. Pricing has been based on sharing the cost

between the caller and the receiver, making mobile calls “unwanted”. Further, the advanced web-based solutions together with increased local wireless networks have lessened interest in mobile communication. In developing countries, for their part, mobile communication is often adopted as the first communication system allowing access to distant others having an impact on its appropriation.

The fact that the failure of “UMTS” and “WAP” is mentioned only in passing, if at all, in the literature is also telling. First, a positive bias reigns in technology studies just as in medical research. Failures are not publishable, only success stories get printed, which distorts the view of history. Further, despite increased efforts at multidisciplinary research, the boundaries between engineering, economics and social and human studies have remained firm. The flop of Universal Mobile Telephone System, including interventionist European regulatory politics is itself an indication of lack of a unified vision, one-sided trust in technologies and lack of sufficient consideration of the social dimension. Unfortunately, the literature reviewed also remains too much within the confines of disciplines to address this historically consequential chain of events that led Europe to lose its advantage in the development of communication networks. Specialist illustrates the crisis of control with pathological symptoms of the American industrial economy. Problems arose first in controlling distribution-related information. The telegraph provided one of the first solutions. Logistical problems caused by growing mass production also amplified the crises of control. What should be produced? How much? When? How should the supply chain be organised? These questions had to be answered on a daily basis and the control mechanisms and techniques at hand could not provide adequate solutions. The first step to resolve these problems was the emergence of a managerial class, who specialised in control related workplace activities. Alongside production and distribution, consumption started to show symptoms of a control crisis. A more intensive flow of information between vendors and consumers was necessary to utilize the capacities of production lines more effectively. This facilitated the

introduction of marketing activities, and later on, in the United States, between the wars, leading to the development of new market research methods.

However, the revolutionary solution to this 20th century crisis came with the emerging information society from the 1960s on: this is the real control revolution. The improvement of production efficiency took place in industrial society, but the problems caused by it are only resolved by revolutionising the control systems of distribution and consumption, and this is what has been happening in information society.

Focusing on the development of computers and the Internet, we can observe the signs of control crisis. These two technologies were originally used by large private and public organizations in desperate need of managing, processing and distributing increased quantities of information, where earlier technologies had become obstacles to the further growth of the organization.

Mobile phones were occasionally seen in use during the public observations, but its use was generally brief and fleeting, only about a minute or two for most observed calls. In contrast, previous work by the researcher showed that mobile phone users in the U.S. sometimes talk on the phone in public for great periods of time, such as while riding the bus. Other research has shown that users in Western societies will use their mobile phones for "meaningless communication" with no explicit goal other than to strengthen relationships. Some explanations for the differences in Uzbekistan include the expense of making a phone call, as well as the culture's general "quietness" outside. Perhaps left over from Soviet times and concern about attracting undue attention, Uzbeks seem concerned with privacy and generally do not talk loudly on buses, subway trains, or other kinds of highly enclosed public space. A natural extension of this behavior would be restricted, conservative use of the mobile phone. Interestingly, the people who were observed using the phone most in outdoor space were men, while indoors the genders of the mobile users were more equally divided. Traditionally, Uzbek women are socially subordinate to men, and they may be conservative in their use of mobile phones outside to deflect attention from themselves. Alternatively, men may simply be

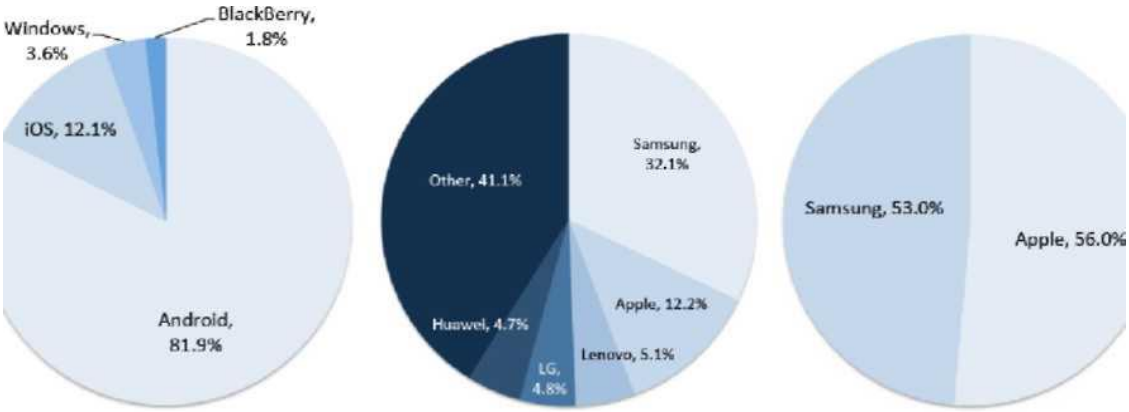
more likely to own a mobile. Although the numbers of people engaging in actual use of the phone in public areas was relatively small as a proportion of the general population, the conspicuous display of phones was much more common. At cafes and restaurants, users would set their mobile phone on the table, perhaps for ease of monitoring calls, convenience, or display. At one popular upscale restaurant, virtually every table had one or two mobile phones on it. Because of the expense of the mobile, the ability to own one is an indication of economic well-being, and there may be a tendency to want to exhibit this prestigious item. It is also possible that these mobile users are not deliberately flaunting their phones, rather they may be visible simply because they are ubiquitous among the well off.

1.2. Mobile operating systems.

Every phone needs some type of operating system to run its services: making calls, texting, taking photos, and so on. The original mobile OSs were fairly simple, since the capabilities of the phones they supported were limited. Modern smartphones have added many of the features of a computer: highspeed Central Processing Units and General Processing Units, large storage space, multitasking, high-resolution screens and cameras, multipurpose communication hardware, and so on. Mobile OSs have had to grow in sophistication to support these features.

Another critical factor is the desire to allow external developers to write software for mobile devices. This represents a huge increment in the effort needed to support the OS, since a full-featured SDK is needed, proper developer support must be provided, and application-facing OS features must be carefully thought out, implemented, and then maintained, often well past the version of the OS where they are introduced.

We briefly review six popular mobile OSs: Android, iOS, Windows Phone, BlackBerry OS, webOS, and Symbian. These OSs, together with Bada, were estimated to control 99.8% of the worldwide smartphone OS market in Q3 2013 (Fig. 1.2.1.a). Broken down by company, worldwide mobile smartphone *shipments* for Q3 2013 are shown in Fig. 1.2.1.b, with Samsung, Apple, Lenovo, LG, and



(a) *(b)* *(c)*

Figure 1.2.1. Smartphone worldwide market share: (a) Q3 2013 share by mobile OS; (b) Q3 2013 shipments by manufacturer; (c) Q3 2013 profit by manufacturer

Huawei occupying the top five spots. Some companies report actual sales numbers as "shipments," while other companies only report the number of devices sent to retailers. For example, Samsung has stated they will not report any numbers for smartphone and tablet sales. Another measure of success is operating profit, shown by company for Q3 2013 in Fig. 1.2.1.c. Apple and Samsung were the only two manufacturers to report positive profits during this quarter.

Smartphones still make up a minority of the total number of mobile phones sold. Current estimates are that 45% of mobile phone sales are for smartphones, and 55% are for non-smart devices.

Android OS

Android OS for mobile devices is developed by the Open Handset Alliance, which is led by Google. Google unveiled the Android distribution in November 2007. Most of the Android core is released under the open-source Apache License. Android uses a Linux kernel with higher-level APIs written in C. Applications are normally programmed in Java and run with the Dalvik virtual machine (DVM) using just-in-time compilation to translate Java bytecode into Dalvik.

Android Inc. was initially founded by Andy Rubin, Rich Miner, Nick Sears, and Chris White in October 2003. Rubin was previously the co-founder of Danger, which developed the Mobile Sidekick. Initial work focused on developing "mobile

phone software." Google acquired Android Inc. in 2005. Rubin stayed to lead development of a mobile device platform using a Linux kernel.

Although the core OS is open-source, other parts are controlled differently. The Open Handset Alliance develops the GPL-licensed parts of Android. Device manufacturers cannot use the Android trademark unless Google certifies their device against the Compatibility Definition Document (CDD). Applications like the Android Market, Google Maps, and Google Docs are not open-source, and are only licensed to devices that pass the CDD.

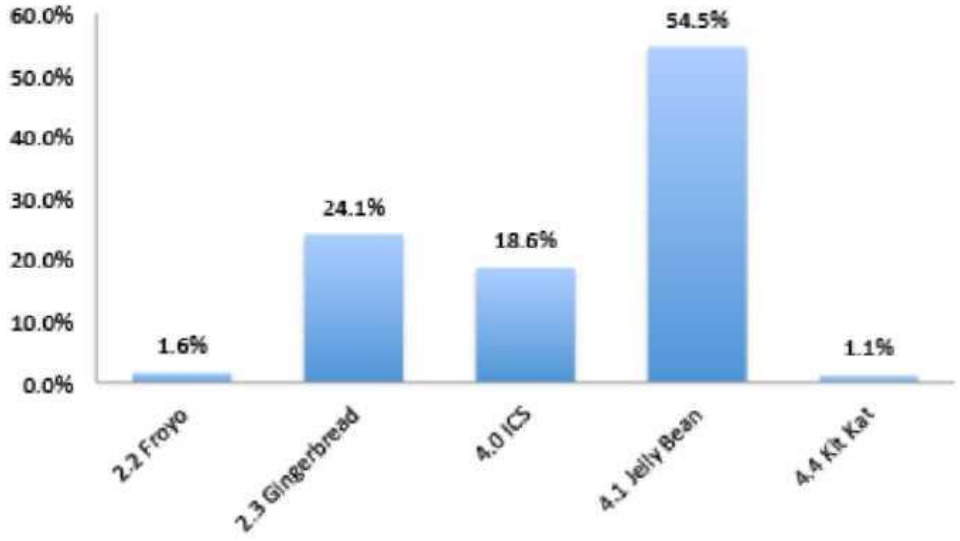


Figure 1.2.2. Percentage of devices running the different versions of Android OS

Android's architecture is split into a Linux kernel with low-level hardware drivers and power management, and SQLite the Android core runtime libraries and the DVM, an Application Framework of application-level APIs, and the applications themselves. Android's usage share by version, as of Dec 2013, is shown in Fig. 1.2.2. Currently, the majority of devices run some variant of Android 4.1, 4.2, or 4.3.

iOS

iOS was originally developed for the iPhone, with the first version released in June 2007. A native application SDK followed in March 2008. iOS is also used

on the iPod Touch-original release September 2007 the iPad original release 2010 and Apple TV. iPhone OS was rebranded iOS in June 2010.

iOS promoted a new style of user interaction for small screen, limited input devices, specifically, direct manipulation. Touch-based gestures like swipe, tap, tap-and-hold, and pinch are used to control on-screen interface elements, and to perform interface operations. Accelerometers support additional physical gestures like shaking and rotating the orientation of the device.

iOS is derived from Mac OS X, and shares its basic Darwin foundation, an open source POSIX-compliant UNIX OS. In this sense iOS can be considered a variant of UNIX.

iOS is made up of four abstraction layers: Core OS, Core Services, Media, and Cocoa Touch:

- **Core OS,** which includes basic low-level features: system support threads, sockets, math, memory general security services certificates, private public keys, encryption external hardware management, Bluetooth, and sound and image processing.
- **Core Services,** which includes basic application services, including accounts, contacts, networking, data management, location, calendar events, store purchasing, SQLite, and XML support.
- **Media Layer,** which includes support for 2D and 3D graphics, audio, and video.
- **Cocoa Touch,** which includes APIs for building applications multitasking, touch input, notifications, interface views, and access to device data.

Windows Phone

Windows Phone is a mobile OS developed by Microsoft as a replacement for their Windows Mobile platform. It was launched in 2010 under the name Windows Phone 7. Various hardware manufacturers including HTC, Samsung, LG, and Nokia are developing Windows Phone devices. In February 2011 Nokia and Microsoft announced that Windows Phone 7 would be the primary OS for all

future Nokia smartphones. Windows Phone 7 received a major upgrade in February 2011, adding features that had been missing in the original release. The second generation Windows Phone 8 was released in October 2012.

Microsoft introduced a new user interface, Metro, with Windows Phone 7. Dynamic tiles populate the home screen, acting as live links to applications, features, and individual items like webpages or media items. Hubs combine local and online content, for example, the Pictures hub can show photos that are on the device and stored on social media networks.

Windows Phone 7's architecture required a hardware layer that meets Microsoft's minimum system requirements: an ARM7 CPU, a DirectX 9-capable GPU, 256MB RAM and 8GB of flash memory, a 5-megapixel camera, a multitouch capacitive display, an A-GPS, an accelerometer, a compass, proximity and light sensors, and six physical buttons: back, start, and search; camera, power/sleep, and volume.

The Windows Phone kernel handles low-level device driver access as well as basic security, networking, and storage. Three libraries: an App Model for application management, a UI model for user-interface management, and a Cloud Integration module for web search via Bing, location services, push notifications, and so on sit above the kernel. Application-facing APIs (Application Programming Interface) include Sliverlight, XNA, HTML/JavaScript and the Common Language Runtime (CLR) that supports C# or VB .Net applications. The kernel itself is a proprietary Windows OS design for embedded devices that combines Windows Embedded and Windows Embedded Compact 7. Windows Phone 8 replaced the Windows CE kernel with one based on Windows NT. This is meant in part to mimic the Windows 8 desktop OS, allowing for easier porting of applications between the two operating systems.

BlackBerry OS

BlackBerry OS is developed by Research In Motion (RIM) for their BlackBerry smartphones and tablet devices. BlackBerry OS 1.0 debuted in January 1999 as part of BlackBerry's pager/email devices. BlackBerry OS 3.6 was released

for the BlackBerry 5810 smartphone in March 2002. The current version, BlackBerry OS 7, was released in August 2011 on the BlackBerry Bold, BlackBerry Torch, and BlackBerry Curve.

One of the main strengths of BlackBerry devices is their ability to handle corporate email. BlackBerry OS supports the Java Mobile Information Device Profile (MIDP) and the Wireless Application Profile (WAP). These protocols are used to synchronize through a BlackBerry Enterprise Server (BES) with push-based calendar, task, contact, email, and note exchange. BES provides the capacity, security, remote wipe, and other features that corporations require for mobile devices that access internal networks and/or corporate data. BlackBerry OS also provides the BlackBerry Internet Service (BIS), a client-specific method to allow Internet access for individual users. This allows consumer customers to access personal email, browse the web, and so on.

BlackBerry OS originally supported applications written in C++. One type of application is a Mobile Data Services (MDS) runtime, which is a container for processing and display data, usually pushed from a user's corporate system. JavaME was also supported, and was used to build applications with access to OS APIs that provide access to standard UI widgets and different OS services.

Both BlackBerry OS 6 and OS 7 were designed to encourage application development. Programming is now done in Java for phones, and in C++ or web-based languages for the PlayBook tablet. OS-supported APIs include browsing, mail, phone, apps, UI, http, math, cryptography, and so on.

A C++ Native Development Kit was recently made available to support development on BlackBerry's PlayBook tablet OS. BlackBerry PlayBook OS 1.0, which is available only on the PlayBook, switched to be QNX-based. QNX is a UNIX microkernel that was originally developed in the 1980s and later re-purposed for embedded devices. RIM (Research in Motion) purchased QNX in April 2010, with plans to transition its upcoming smartphones to OS 10 and QNX. Blackberry 10, together with the Z10 and Q10 Blackberry smartphones, were released in January 2013.

WebOS

Palm and Palm OS dominated the US smartphone market in the early 2000s with their Treo handsets, which represented one of the first examples of a convergent device a combination of a phone and a Palm OS. Hardware and Palm OS development slowed, however, allowing companies like RIM, Microsoft, and eventually Apple and Google to overtake Palm's lead. Palm itself began selling Windows Phone versions of the Treo in the mid-2000s. Palm was sold to Hewlett-Packard in April 2010 for $1.2 billion.

Web OS was developed to replace Palm OS as the next-generation OS for Palm's smartphone devices. Web OS was introduced in January 2009. It initially ran on the Palm Pre and the Palm Pixi , introduced in the second half of 2009. HP updated webOS to version 3.0, including it on the HP Veer and HP Pre 3 phones, and on the HP TouchPad. In August 2011, HP announced it would sell its Personal Systems Group, which included consumer PC products and webOS. Although the decision to sell the Personal Systems Group was later placed under review, HP announced in January it will open-source webOS by the end of September 2012.

The webOS interface revolves around "cards," individual applications that are presented one-at-a-time and can be scrolled through horizontally like a deck of cards to move applications from the foreground to the background. On startup, a launcher screen with a grid of icons is presented, together with a quick-launch bar holding commonly-used applications. The UI supports standard touch and gesture commands like tap, swipe, and pinch.

WebOS's Core OS is built on a Linux 2.6 kernel, with device drivers, an ext3 filesystem, network communication, and bluetooth. Above this sits the UI System Manager, which is responsible for window, UI, and application management. The JavaScript framework provides application-facing APIs, and the webOS Services Manager offers access to location, phone, camera, and so on. webOS applications are programmed in HTML, CSS, and JavaScript, and use Mojo and webOS services for UI and OS support.

Symbian OS

Symbian is a mobile device OS developed by Nokia. It was originally the EPOC graphical operating system for PSION portable devices. In 1998 PSION, Nokia, Ericsson, and Motorola formed Symbian OS. In 2008 plans were announced to form a Symbian platform by merging software assets from Nokia, NTT DoCoMo, and Sony Ericsson. Currently, however, the Symbian Foundation is run and maintained by Nokia alone, providing access to Symbian through standard licensing agreements.

The original Symbian OS was divided into two parts: a core OS that supported a Device Family Reference Design (DFRD) and a UI built on the DFRD. This allowed different UIs to be built for different types of devices, or for different manufacturers' handsets, but with a common OS core. Examples included the Pearl UI used by Nokia and the Quartz UI used by Ericsson. This strategy was later abandoned and different UIs were spun off to different companies.

The latest version of Symbian is Symbian OS 9.5, released in March 2007. Follow on versions include Symbian 1, Symbian 2, and Symbian 3, which was released in 2010 on the Nokia N8. Symbian 3 includes modern mobile OS technology like 2D and 3D graphics acceleration, touch-based interaction, and UI widgets.

Symbian OS follows a familiar architecture. It is built on a nanokernel /microkernel core with basic localization and screen drivers. Base services sit above the kernel, and include low-level libraries, media frameworks, XML, file system management, and hardware abstraction. OS services provide communication, telephony, networking, multimedia, and graphics. These support an Application Services layer with application-facing APIs for development and an interface layer to manages the UI. A JVM (Java ME) is also included above the OS services layer. Nokia provides SDKs for Symbian development that support a variety of languages, including C++ and Java. Nokia announced in 2011 that it would transition its smartphones to Windows Phone 7. This leaves it unclear how, or for how long, Symbian will continue to exist as a major mobile OS.

1.3. Map applications for mobile phones.

There are lots of map applications. They created for many mobile operating systems. I wanted to say some map applications which more popular in the world. They support only Android operating system mobile phones. I can say they are more useful and they have many advantages. Let's introduce those map applications. Our best Android map apps roundup will give you the options needed to let technology lead the way whenever you're away from home.

MapQuest

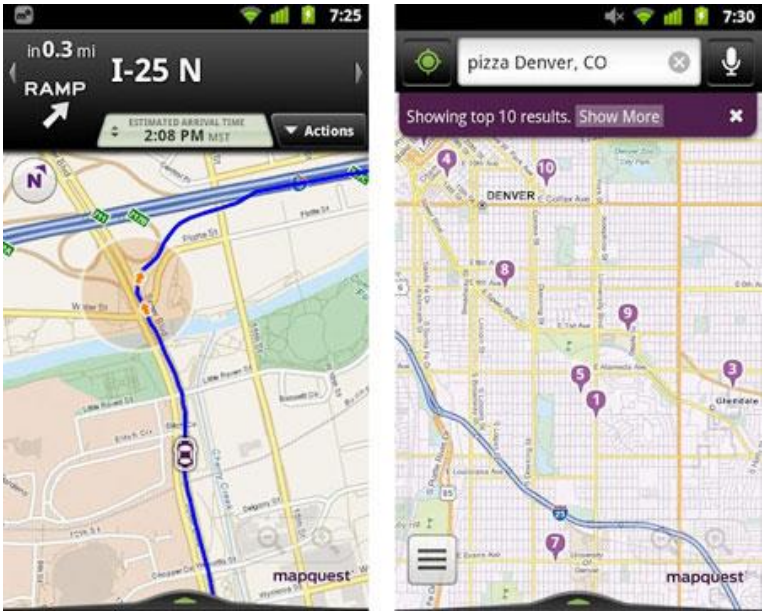


Figure 1.3..1 User interface of Mapquest.

You may have issues with asking strangers for directions or simply prefer having technology put in charge while traveling. Well in that case, the MapQuest app may just be the right choice for you. Amongst its many features, the key offering here is the voice-guided navigation system that assists you with turn-by-turn directions. Furthermore, the application tags along various attributes such as voice controlled search, live traffic flow, satellite maps view as well as driving and walking guidance. If you happen to make a wrong turn when on the go, this offering is said to auto re-route in order to get you to your destination on time. It also features a Place Map toolbar for one-click access to nearby gas stations, restaurants and many more. You can access Google Play, formerly known as the Android Market, to pocket the MapQuest application.

Google Maps

The next candidate on our recommendations list could probably be one of the most accurate Android map apps that you'll turn to every now and then. Google Maps needs no introduction, so we will quickly run through the features it brings along. To begin with, the application lets you find places, rate them and even receive recommendations. The application is complete with voice-guided GPS navigation and it even allows you to check-in at places. Naturally, with this attribute you can also see your friends on the map and easily catch up with them if you happen to be nearby.

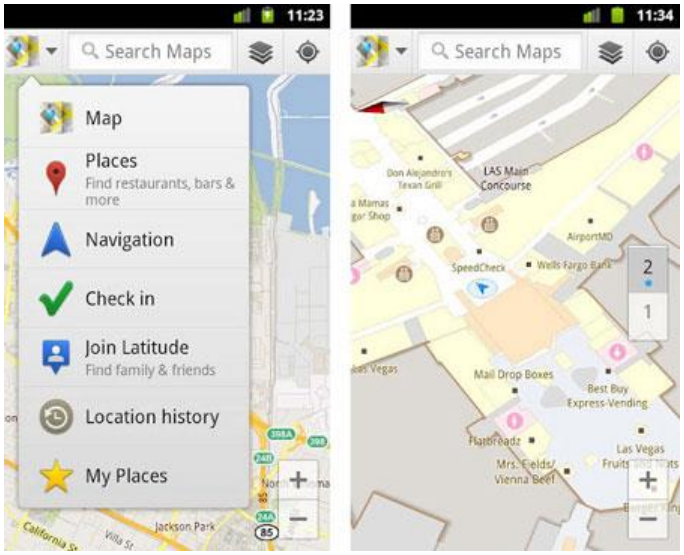


Figure 1.3.2. Desktop Google Maps

Maps (-)

Traveling to places where network coverage is weak may tend to be troublesome. Instead of digging up Google Play for apps that come with the offline access to different worldwide locations, we've got an application that can deliver just the same, Maps (-). It lets you download and cache places to the SD card, which can then be used whenever you decide on traveling to regions with poor

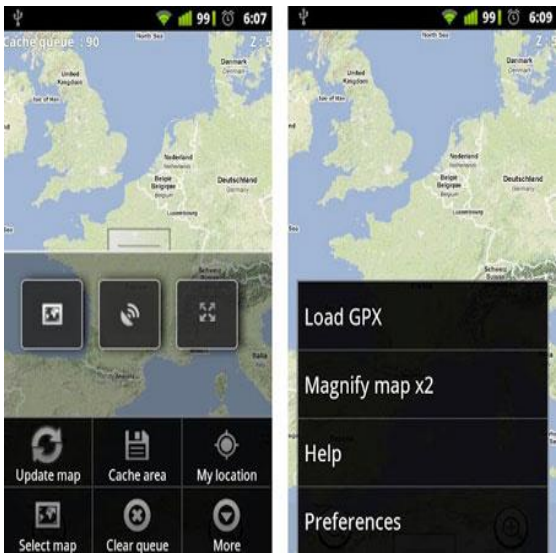


Figure 1.3.3. *Interface of Maps (-)*

network coverage or saving data costs. What's more, the application allows you to magnify the map for better visibility and it comes with an auto-follow GPS feature.

BackCountry Navigator PRO GPS

Packed with galore of features and a neat UI, BackCountry Navigator PRO GPS is next in line on our Android map apps roster. Wearing a $9.99 price tag on Google Play, this entrant gives you the liberty to dabble around in offline topo maps and it comes with too GPS support. So you can rake in the benefits of topo mapping GPS with navigation when hiking through the mountain side.

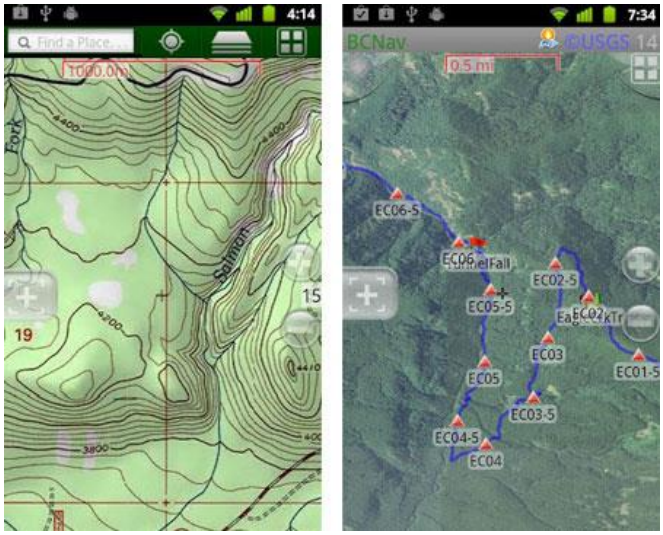


Figure 1.3.4. Using application.

The application allows you to download the required topo maps in advance and also makes accessing them convenient from almost anywhere. You can purchase various add-ons like boundary maps, ATV, Whitewater, Equestrian and other Trail Maps, through the in-app store. Chances are, this tool might just become your favorite companion each time you plan a hiking trip with friends and family.

aMetro

Offering maps of transit systems around the world, the aMetro app is most likely to be quite useful if you take to globetrotting pretty often, especially if you're the type who prefers traveling with a limited budget and getting around through public transport. So whether you're in need of maps for metros, buses, trains or subways, this application is highly capable of delivering it all.

Apart from displaying routes between stations, you can view expected travel times as well. A simple tap on a station selects your destination and just in case you are wondering how much does it all come for? Well, this app doesn't require you to shell out a single penny.

Figure 1.3.5. Desktop of aMetro.

Maps Ruler

The next candidate to feature on our Android map apps compilation is Maps Ruler. If you're a sports enthusiasts, then chances are that you may already have this application downloaded on your device. As the name suggests, this digital ruler lets you calculate distance between two selected points. So you can figure out the shortest way to reach the desired destination. Paths can be saved or loaded whenever you like and it also facilitates carrying out measurement conversions. The app is most likely to place itself as a neat tool for athletes. It can be employed as a golf distance calculator as well. The Maps Ruler tool is available for free through Google Play.

Maps With Me

Maps With Me is one of the many offerings that don't require much time to become a favorite amongst users. Firstly, it comes with offline maps support from almost every country across the globe with the GPS tracking feature in tow. Basically, the application is like virtually carrying the entire world in your pocket. If you want to take a look at a map of any city in the world, all you have to do is simply launch the app and download the appropriate location files for quick access

when offline. Installing maps shouldn’t be a tough job. You can head on to Google Play and download the Maps With Me application for free.

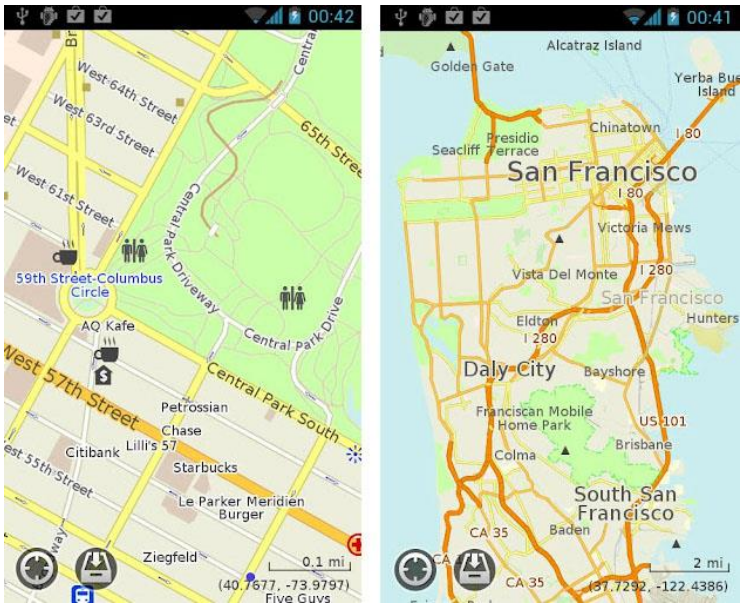


Figure 1.3.6. Desktop of Maps With Me

2.1. Exploring Android, the open source mobile platform

Android is a software environment built for mobile devices. It's not a hardware platform. Android includes a Linux kernel-based OS, a rich UI, end-user applications, code libraries, application frameworks, multimedia support, and much more. And, yes, even telephone functionality is included! Whereas components of the underlying OS are written in C or C++, user applications are built for Android in Java. Even the built-in applications are written in Java.

One feature of the Android platform is that there's no difference between the built-in applications and applications that you create with the SDK. This means that you can write powerful applications to tap into the resources available on the device. The most notable feature of Android might be that it's open source; missing elements can and will be provided by the global developer community. Android's Linux kernel-based OS doesn't come with a sophisticated shell environment, but because the platform is open, you can write and install shells on a device. Likewise, multimedia codecs can be supplied by third-party developers and don't need to rely on Google or anyone else to provide new functionality. That's the power of an open source platform brought to the mobile market.

The term *platform* refers to Android itself the software including all the binaries, code libraries, and tool chains. This work focuses on the Android platform, the Android emulators available in the SDK are simply components of the Android platform.

With all of that as a backdrop, creating a successful mobile platform is clearly a non- trivial task involving numerous players. Android is an ambitious undertaking, even for Google, a company of seemingly boundless resources and moxie and they're getting the job done. Within a span of two years, Android has seen four major software releases and the release of multiple handsets across most major mobile carriers in the global market.

Now that you've got an introduction to what Android is, let's look at the why and where of Android to provide some context and set the perspective for

Android's introduction to the marketplace. After that, it's on to exploring the platform itself!

Android promises to have something for everyone. It aims to support a variety of hardware devices, not just high-end ones typically associated with expensive smartphones. Of course, Android users will enjoy improved performance on a more powerful device, considering that it sports a comprehensive set of computing features. But how well can Android scale up and down to a variety of markets and gain market and mind share? How quickly can the smartphone market become the standard? Some folks are still clinging to phone-only devices, even though smartphones are growing rapidly in virtually every demographic. Let's look at Android from the perspective of a few existing players in the marketplace. When you're talking about the cellular market, the place to start is at the top, with the carriers, or as they're sometimes referred to, the mobile operators.

The majority of cell phones on the market continue to be consumer flip phones and feature phones, phones that aren't smartphones. These phones are the ones consumers get when they walk into the retailer and ask what can be had for free. These consumers are the "I just want a phone" customers. Their primary interest is a phone for voice communications, an address book, and increasingly, texting. They might even want a camera. Many of these phones have additional capabilities such as mobile web browsing, but because of relatively poor user experience, these features aren't employed heavily. The one exception is text messaging, which is a dominant application no matter the classification of device. Another increasingly in-demand category is location-based services, which typically use the Global Positioning System *(GPS).*

Android's challenge is to scale down to this market. Some of the bells and whistles in Android can be left out to fit into lower-end hardware. One of the big functionality gaps on these lower-end phones is the web experience the user gets. Part of the problem is screen size, but equally challenging is the browser technology itself, which often struggles to match the rich web experience of desktop

computers. Android features the market-leading WebKit browser engine, which brings desktop-compatible browsing to the mobile arena. If a rich web experience can be effectively scaled down to feature phone class hardware, it would go a long way toward penetrating this end of the market.

The WebKit browser engine is an open source project that powers the browser found in Macs (Safari) and is the engine behind Mobile Safari, which is the browser on the iPhone. It's not a stretch to say that the browser experience is what makes the iPhone popular, so its inclusion in Android is a strong plus for Android's architecture.

Let's start by naming the major smartphone players: Symbian, BlackBerry from Research in Motion, iPhone from Apple, Windows (Mobile, SmartPhone, and now Phone 7), and of course, the increasingly popular Android platform.

One of the major concerns of the smartphone market is whether a platform can synchronize data and access Enterprise Information Systems for corporate users. Device-management tools are also an important factor in the enterprise market. The browser experience is better than with the lower-end phones, mainly because of larger displays and more intuitive input methods, such as a touch screen, touch pad, slide- out keyboard, or a jog dial.

Android's opportunity in this market is to provide a device and software that people want. For all the applications available for the iPhone, working with Apple can be a challenge; if the core device doesn't suit your needs, there's little room to maneuver because of the limited models available and historical carrier exclusivity. Now that email, calendaring, and contacts can sync with Microsoft Exchange, the corporate environment is more accessible, but Android will continue to fight the battle of scaling the Enterprise walls. Later Android releases have added improved support for the Microsoft Exchange platform, though third-party solutions still out-perform the built- in offerings. BlackBerry is dominant because of its intuitive email capabilities, and the Microsoft platforms are compelling because of tight integration to the desktop experience and overall familiarity for Windows users. iPhone has surprisingly good integration with

Microsoft Exchange for Android to compete in this arena, it must maintain parity with iPhone on Enterprise support.

You've seen how Android stacks up next to feature phones and smartphones. Next, we'll see whether Android, the open source mobile platform, can succeed as an open source project.

Android will likely always be an open source project, but to succeed in the mobile market, it must sell millions of units and stay fresh. Even though Google briefly entered the device fray with its Nexus One phone, it's not a hardware company. From necessity, Android is sold by others such as HTC and Motorola, to name the big players. These manufacturers start with the Android Open Source Platform (AOSP), but extend it to meet their need to differentiate their offerings. Android isn't the first open source phone, but it's the first from a player with the market-moving weight of Google leading the charge. This market leadership position has already translated to impressive unit sales across multiple manufacturers. So, now that there are a respectable number of devices on the market, can Android keep it together and avoid fragmentation?

Open source is a double-edged sword. On one hand, the power of many talented people and companies working around the globe and around the clock to deliver desirable features is a force to be reckoned with, particularly in comparison with a traditional, commercial approach to software development. This topic has become trite because the benefits of open source development are well documented. On the other hand, how far will the competing manufacturers extend and potentially split Android? Depending on your perspective, the variety of Android offerings is a welcome alternative to a more monolithic iPhone device platform where consumers have few choices available.

Another challenge for Android is that the licensing model of open source code used in commercial offerings can be sticky. Some software licenses are more restrictive than others, and some of those restrictions pose a challenge to the open source label. At the same time, Android licensees need to protect their investment, so licensing is an important topic for the commercialization of Android.

2.2. Set up the Android SDK and Emulator

The Android SDK is a freely available download from the Android website. The first thing you should do before going any further in this work is make sure you have the Android SDK installed, along with Eclipse and the Android plug-in for Eclipse, also known as the *Android Development Tools,* or simply as the *ADT.* The Android SDK is required to build Android applications, and Eclipse is the preferred development environment for this work. You can download the Android SDK from http://developer. android.com/sdk/index.html.

As in any development environment, becoming familiar with the class structures is helpful, so having the documentation at hand as a reference is a good idea. The Android SDK includes HTML-based documentation, which primarily consists of Javadoc-formatted pages that describe the available packages and classes. The Android SDK documentation is in the /doc directory under your SDK installation. Because of the rapidly changing nature of this new platform, you might want to keep an eye out for any changes to the SDK.

Android's Java environment can be broken down into a handful of key sections. When you understand the contents in each of these sections, the Javadoc reference material that ships with the SDK becomes a real tool and not just a pile of seemingly unrelated material. You might recall that Android isn't a strictly Java ME software environment, but there's some commonality between the Android platforms and other Java development platforms. The next few sections review some of the Java packages (core and optional) in the Android SDK and where you can use them. The remaining works provide a deeper look into using many of these programming topics.

Core Android packages

If you've ever developed in Java, you'll recognize many familiar Java packages for core functionality. These packages provide basic computational support for things such as string management, input/output controls, math, and more. The following list contains some of the Java packages included in the Android SDK:

- java.lang-Core Java language classes;
- java.i o-Input/output capabilities;
- java.net-Network connections;
- java.text-Text-handling utilities;
- java.math-Math and number-manipulation classes;
- javax.net-Network classes;
- javax. security-Security-related classes;
- javax. xml-DOM-based XML classes;
- org .apache.-HTTP-related classes;
- org .xml-SAX-based XML classes.

Additional Java classes are also included. Generally speaking, this work won't focus much on the core Java packages listed here, because our primary concern is Android development. With that in mind, let's look at the Android-specific functionality found in the Android SDK.

Android-specific packages are easy to identify because they start with android in the package name. Some of the more important packages are:

- android.app-Android application model access;
- android.bluetooth-Android's Bluetooth functionality;
- android.content-Accessing and publishing data in Android;
- android.net-Contains the Uri class, used for accessing content;
- android.gesture-Create, recognize, load, and save gestures;
- android.graphics-Graphics primitives;
- android.location-Location based services (such as GPS);
- androi d.open-OpenGL classes;
- android.os-System-level access to the Android environment;
- android.provider-ContentProvider-related classes;
- android.telephony-Telephony capability access, including support for both Code;
- android.text-Text layout;

- android.util-Collection of utilities for logging and text manipulation, including XML;
- android.view-UI elements;
- androi d.webkit-Browser functionality;
- android.widget-More UI elements.

Some of these packages are core to Android application development, including android.app, android.view, and android.content. Other packages are used to varying degrees, depending on the type of applications that you're constructing.

Optional packages

Not every Android device has the same hardware and mobile connectivity capabilities, so you can consider some elements of the Android SDK as optional. Some devices support these features, and others don't. It's important that an application degrade gracefully if a feature isn't available on a specific handset. Java packages that you should pay special attention to include those that rely on specific, underlying hardware and network characteristics, such as location-based services and wireless technologies such as Bluetooth and Wi-Fi.

This quick introduction to the Android SDK's programming interfaces is just that quick and at a glance. Upcoming works go into the class libraries in further detail, exercising specific classes as you learn about various topics such as UIs, graphics, location-based services, telephony, and more. For now, the focus is on the tools required to compile and run (or build) Android applications. Before you build an Android application, let's examine how the Android SDK and its components fit into the Eclipse environment.

Set up the Emulator

The Android tools include an emulator, a piece of software that pretends to be an Android device. This is very useful for development – not only does it mean you can get started on Android withouta device, but the emulator can help test device configurations that you do not own.

The Android emulator can emulate one or several Android devices. Each configuration you want is stored in an "Android Virtual Device", or AVD. The AVD Manager, which you used to download the SDK components earlier in this work, is where you create these AVDs.

If you do not have the AVD Manager running, you can run it via the android command from your directory, or via Window AVD Manager from Eclipse. It starts up ona screen listing the AVDs you have available – initially, the list will be empty:

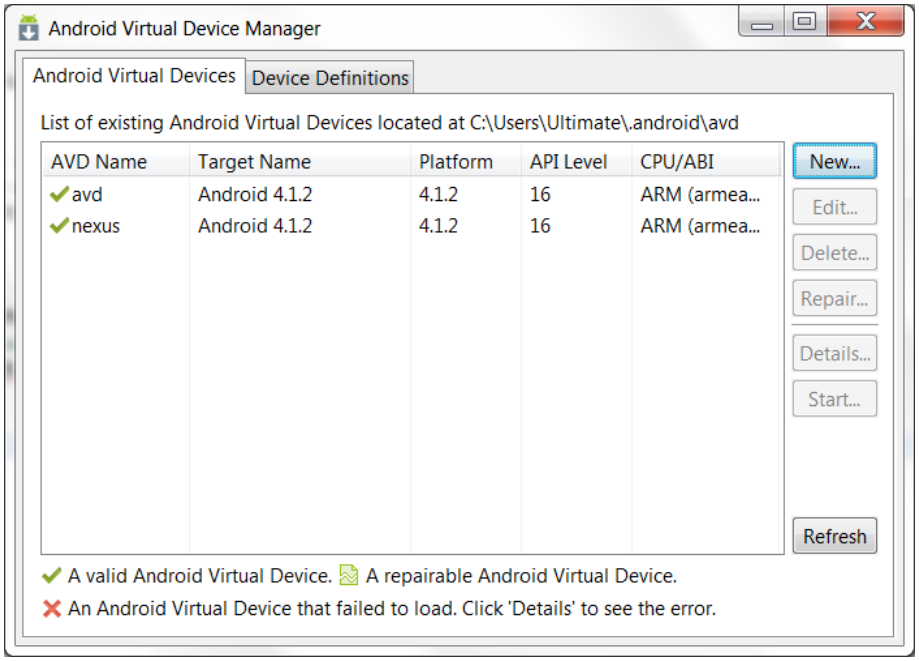


Figure 2.2.1. AVD manager

Click the New... button to createa new AVD file. This brings upa dialog where you can configure what this AVD should look and work like.

You need to provide the following:

• A name for the AVD. Since the name goes into files on your development machine, you will be limited by filename conventions for your operating system.

• The Android version you want the emulator to run. Choose one of the SDKs you installed via the drop-down list. Note that in addition to "pure" Android environments, you will have options based on the third-party add-ons you selected. For example, you probably have some options for setting up AVDs containing the Google APIs, and you will need such an AVD for testing an application that uses Google Maps.

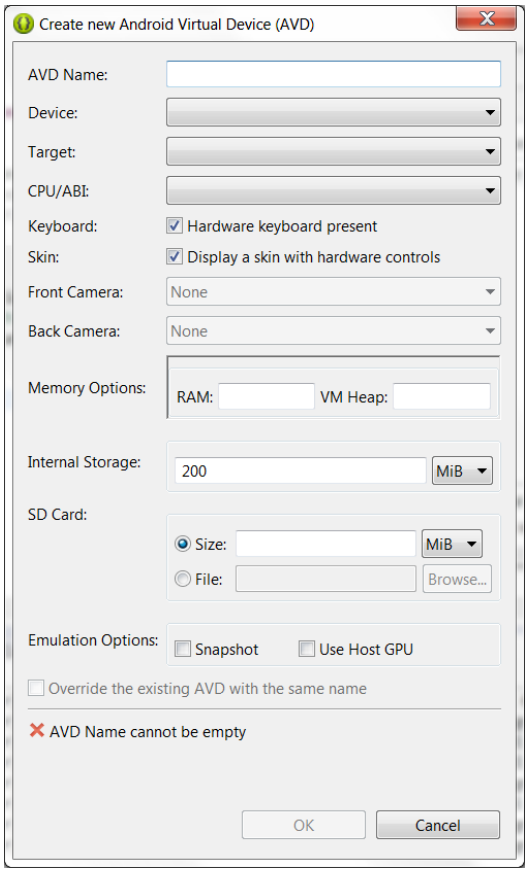


Figure 2.2.2. Creating new Android Virtual Device

• Details about the SD card the emulator should emulate. Since Android devices invariably have some form of "external storage", you probably want to set up an SD card, by supplying a size in the associated field. However, sincea file will be created on your development machine of whatever size you specify for the card, you probably do not want to create a 2GB emulated SD card. 32MB is a nice starting point, though you can go larger if needed.

• The "skin" or resolution the emulator should run in. The skin options you have will depend upon what target you chose. The skins let you choose a typical Android screen resolution. You can also manually specify a resolution when you want to test a non-standard configuration.

You can skip the "Hardware" section for now, as changing those settings is usually only required for advanced configurations.

The resulting dialog might look something like this:

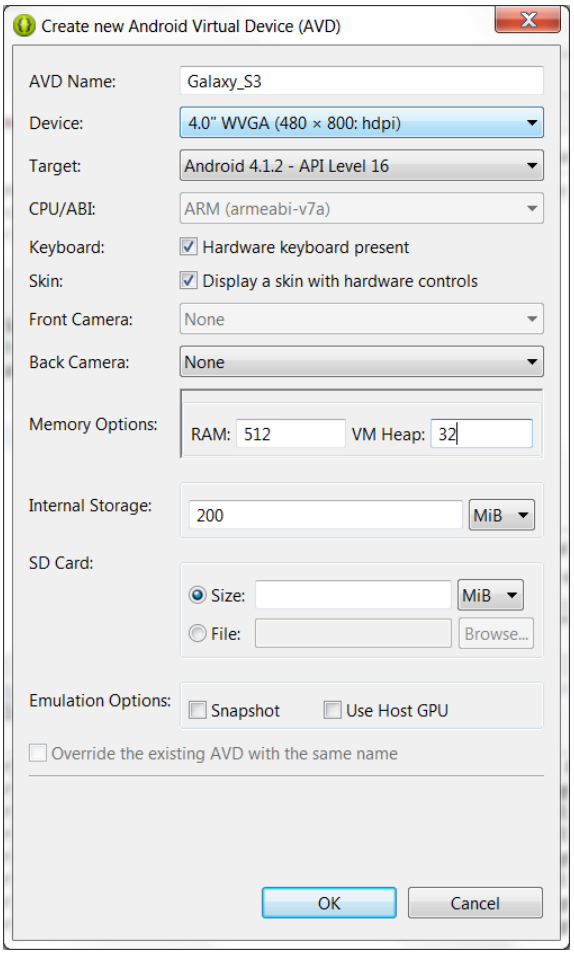


Figure 2.2.3. Creating a new AVD (continued)

Click the Create AVD button, and your AVD stub will be created. To start the emulator, highlight it in the list and click Start... You can skip the launch options for now and just click Launch. The first time you launch a new AVD, it will take a long time to start up. The second and subsequent times you start the AVD, it will come up a bit faster, and usually you only need to start it up once per day (e.g., when you start development). You do not need to stop and restart the emulator every time you want to test your application, in most cases.

The emulator will go througha few startup phases, first with a plain-text "ANDROID" label:

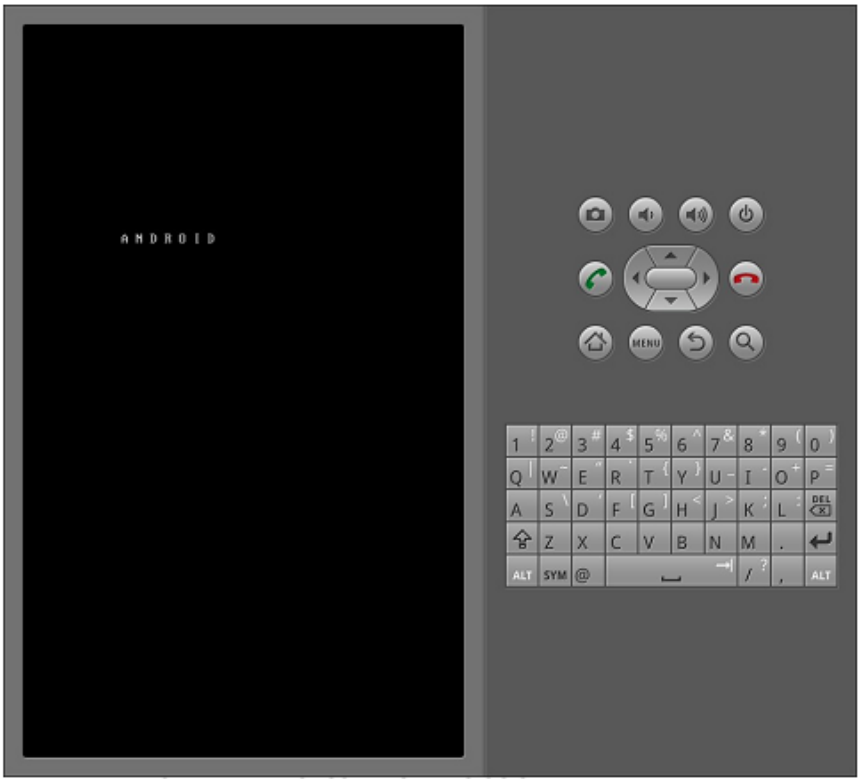


Figure 2.2.4. Starting Android Virtual Device

then a graphical Android logo:

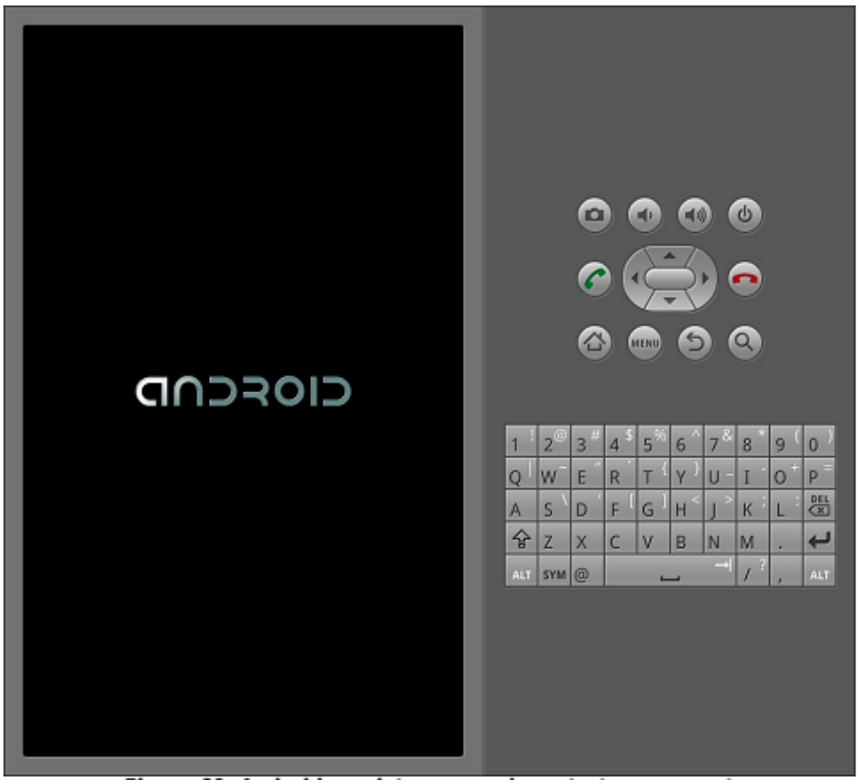


Figure 2.2.5. Staring AVD (continued)

before eventually landing at the home screen (the first time you run the AVD, shown below) or the keyguard:

Figure 2.2.6. Android home screen.

2.3 Building Android application in Eclipse

Eclipse provides a comprehensive environment for Android developers to create applications. In this section, we'll demonstrate how to build a basic Android application, step-by-step. You'll learn how to define a simple UI, provide code logic to support it, and create the deployment file used by all Android applications: AndroidManifest.xml. Our goal in this section is to get a simple application under your belt. We'll leave more complex applications for later works; our focus is on exercising the development tools and providing a concise, yet complete reference. Building an Android application isn't much different from creating other types of Java applications in the Eclipse IDE. It all starts with choosing File > New and selecting an Android application as the build target. Like many development environments, Eclipse provides a wizard interface to ease the task of creating a new application. We'll use the Android Project Wizard to get off to a quick start in building an Android application.

The Android Project Wizard

The most straightforward manner to create an Android application is to use the Android Project Wizard, which is part of the ADT plug-in. The wizard provides a simple means to define the Eclipse project name and location, the

Activity name corresponding to the main UI class, and a name for the application. Also of importance is the Java package name under which the application is created. After you create an application, it's easy to add new classes to the project.

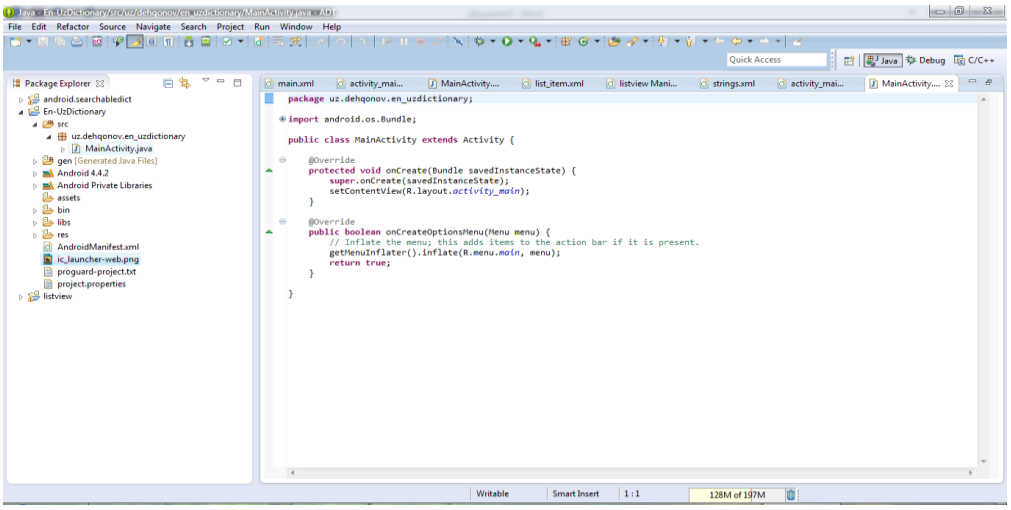


Figure 2.3.1. Creating new Android Application project

Click Finish to create your sample application. At this point, the application compiles and is capable of running on the emulator no further development steps are required. Of course, what fun would an empty project be? Let's flesh out this sample application and create an Android tip calculator.

Android sample application code

The Android Application Wizard takes care of a number of important elements in the Android application structure, including the Java source files, the default resource files, and the AndroidManifest.xml file. Looking at the Package Explorer view in Eclipse, you can see all the elements of this application. Here's a quick description of the elements included in the sample application:

- The src folder contains two Java source files automatically created by the wizard;
- This Twojava contains the main Activity for the application. You'll modify this file to add the sample application's tip calculator functionality;
- R.java contains identifiers for each of the UI resource elements in the application. Never modify this file directly. It automatically regenerates

every time a resource is modified; any manual changes you make will be lost the next time the application is built;

- Android.jar contains the Android runtime Java classes. This reference to the android.jar file found in the Android SDK ensures that the Android runtime classes are accessible to your application;
- The res folder contains all the Android resource folders, including.

Drawables contains image files such as bitmaps and icons. The wizard provides a default Android icon named icon.png. Layout contains an XML file called main.xml. This file contains the UI elements for the primary view of your Activity. In this example, you'll modify this file but you won't make any significant or special changes just enough to accomplish the meager UI goals for your tip calculator. It's not uncommon for an Android application to have multiple XML files in the Layout section of the resources.

Values contains the strings.xml file. This file is used for localizing string values, such as the application name and other strings used by your application.

AndroidManifest.xml contains the deployment information for this project. Although AndroidManifest.xml files can become somewhat complex, this work's manifest file can run without modification because no special permissions are required.

Now that you know what's in the project, let's review how you're going to modify the application. Your goal with the Android tip calculator is to permit your user to enter the price of a meal, then tap a button to calculate the total cost of the meal, tip included. To accomplish this, you need to modify two files: ChapterTwo.java and the UI layout file, main.xml. Let's start with the UI changes by adding a few new elements to the primary View.

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

```
s/android" android:orientation="vertical"
android:layout_width=Mfill_parent"
   android:layout_height=Mfill_parent"
   >
<TextView
android:layout_width=Mfill_pa
rent"
android:layout_height=Mwrap_c
ontent"  android:text="work  2
Android Tip Calculator"
   />
<EditText
android:id=M@+id/mealprice"
android:layout_width=Mfill_pa
rent"
android:layout_height=Mwrap_c
ontent"
android:autoText=Mtrue"
/>
<Button
android:id=M@+id/calculate"
android:layout_width="wrap_co
ntent"
android:layout_height="wrap_c
ontent"
android:text="Calculate Tip"
  />
<TextView
android:id="@+id/answer"
android:layout_width="fill_pa
```

```
rent"
android:layout_height="wrap_c
ontent" android:text="" />

</LinearLayout>
```

The layout for this application is straightforward. The overall layout is a vertical, linear layout with only four elements; all the UI controls, or *widgets,* are going to be in a vertical arrangement.

A static TextView displays the title of the application. An EditText collects the price of the meal for this tip calculator application. The EditText element has an attribute of type android: id, with a value of mealprice. When a UI element contains the android:id attribute, it permits you to manipulate this element from your code. When the project is built, each element defined in the layout file containing the android:id attribute receives a corresponding identifier in the automatically generated R.java class file.

A button named calculate is added to the view. Note that this element also has an android:id attribute because we need to capture click events from this UI element. A TextView named answer is provided for displaying the total cost, including tip. Again, this element has an id because you'll need to update it during runtime.

When you save the file main.xml, it's processed by the ADT plug-in, compiling the resources and generating an updated R.java file. Try it for yourself. Modify one of the id values in the main.xml file, save the file, and open Rjava to have a look at the constants generated there. Remember not to modify the R.java file directly, because if you do, all your changes will be lost! If you conduct this experiment, be sure to change the values back as they make sure the rest of the project will compile as it should. Provided you haven't introduced any syntactical errors into your main.xml file, your UI file is complete. Double-click the main.xml file to launch the layout in a graphical form. At the bottom of the file you can switch between the Layout view and the XML view.

Packaging the application

At this point, your application has compiled and is ready to be run on the device. Let's look more deeply at what happens after the compilation step. You don't need to perform these steps because the ADTs handle these steps for you, but it's helpful to understand what's happening behind the scenes.

Recall that despite the compile-time reliance on Java, Android applications don't run in a Java VM. Instead, the Android SDK employs the Dalvik VM. For this reason, Java bytecodes created by the Eclipse compiler must be converted to the .dex file format for use in the Android runtime. The Android SDK has tools to perform these steps, but thankfully the ADT takes care of all of this for you transparently.

The Android SDK contains tools that convert the project files into a file ready to run on the Android emulator. If you recall from our earlier discussion of Android SDK tools, the tool used at design time is aapt. Application resource XML files are processed by aapt, with the Rjava file created as a result remember that you need to refer to the R class for UI identifiers when you connect your code to the UI. Java source files are first compiled to class files by your Java environment, typically Eclipse and the JDT. After they're compiled, they're then converted to dex files to be ready for use with Android's Dalvik VM. Surprisingly, the project's XML files are converted to a binary representation, not to text as you might expect. But the files retain their .xml extension on the device.

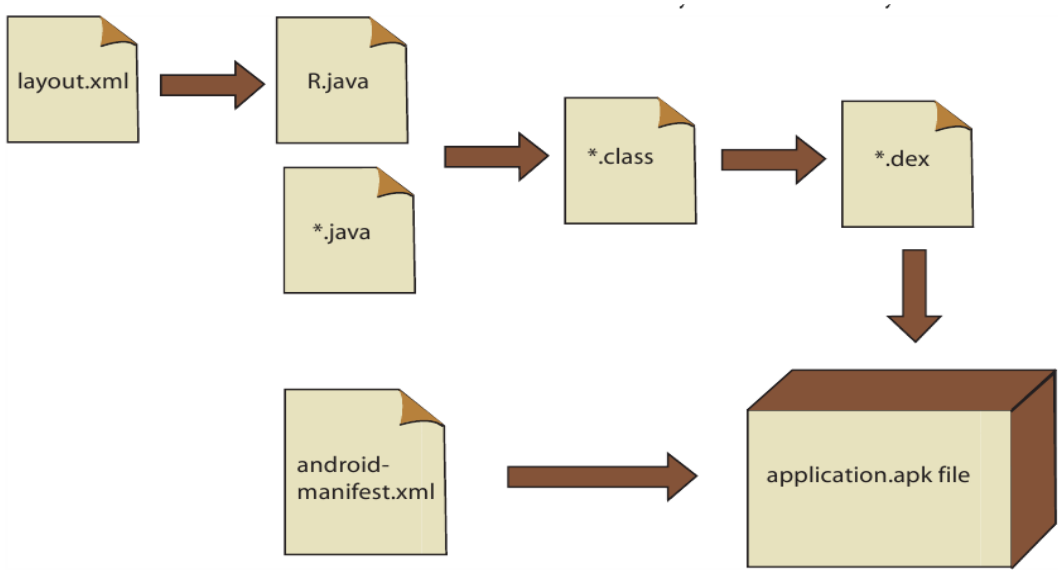


Figure 2.3.2. The ADT employs tools from the Android SDK to convert source files to a package that's ready to run on an Android device or emulator.

The converted XML files, a compiled form of the nonlayout resources including the Drawables and Values, and the dex file (classes.dex) are packaged by the aapt tool into a file with a naming structure of *projectname.apk.* The resulting file can be read with a pkzip-compatible reader, such as WinRAR or WinZip, or the Java archiver, jar.

Now you're finally ready to run your application on the Android emulator. It's important to become comfortable with working in an emulated environment when you're doing any serious mobile software development. There are many good reasons for you to have a quality emulator available for development and testing. One simple reason is that having multiple real devices with requisite data plans is an expensive proposition. A single device alone might cost hundreds of dollars. Android continues to gain momentum and is finding its way to multiple carriers with numerous devices and increasingly sophisticated capabilities. Having one of every device is impractical for all but development shops with the largest of budgets. For the rest of us, a device or two and the Android emulator will have to suffice. Let's focus on the strengths of emulator-based mobile development.

2.4 Working with Google Map API

Google Maps is a desktop and mobile web mapping service application and technology provided by Google, offering satellite imagery, street maps, and Street View perspectives, as well as functions such as a route planner for traveling by foot, car, bicycle, or with public transportation. Also supported are maps embedded on third-party websites via the Google Maps API, and a locator for urban businesses and other organizations in numerous countries around the world. Google Maps satellite images are not updated in real time; however, Google adds data to their Primary Database on a regular basis, and most of the images are no more than 3 years old.

The opt-in redesigned version of the desktop application has been available since 2013, alongside the "classic" version. The redesigned version was met by user criticism regarding hiding some common functions, removing a scale bar, and lack of other features that include My Places and sharable customized links to parametrized split Street View and Map views.

Google Maps uses a close variant of the Mercator projection, and therefore cannot accurately show areas around the poles. A related product is Google Earth, a stand-alone program which offers more globe-viewing features, including showing polar areas.

Google Maps for mobile is the world's most popular app for smartphones, with over 54% of global smartphone owners using it at least once during the month of August 2013.

After the success Google launched the Google Maps API in June 2005 to allow developers to integrate Google Maps into their websites. It is a free service, and currently does not contain ads, but Google states in their terms of use that they reserve the right to display ads in the future.

By using the Google Maps API, it is possible to embed Google Maps site into an external website, on to which site specific data can be overlaid. Although initially only a JavaScript API, the Maps API was expanded to include an API for Adobe Flash applications (but this has been deprecated), a service for retrieving static map images, and web services for performing geocoding, generating driving directions, and obtaining elevation profiles. Over 1,000,000 web sites use the Google Maps API, making it the most heavily used web application development API.

The Google Maps API is free for commercial use, provided that the site on which it is being used is publicly accessible and does not charge for access, and is not generating more than 25 000 map accesses a day. Sites that do not meet these requirements can purchase the Google Maps API for Business.

Google Maps for mobile and other devices

In October 2005, Google introduced a Java application called Google Maps for Mobile, intended to run on any Java-based phone or mobile device. Many of the web-based site's features are provided in the application.

On November 4, 2009, Google Maps Navigation was released in conjunction with Google Android OS 2.0 Eclair on the Motorola Droid, adding voice commands, traffic reports, and street view support. The initial release was limited to the United States. The service was launched in the UK on 20 April 2010 and in large parts of continental western Europe on June 9, 2010.

In March 2011 Google Vice President of Location Service, Marissa Mayer said that Google provided map services to 150 million users.

In June 2012, Apple announced that they would replace Google Maps with their own maps service from iOS 6. However, on December 13, 2012, Google announced the availability of Google Maps in the Apple App Store, starting with the iPhone version. Just hours after the Google Maps iOS app was released, it became the top free app in the App Store.

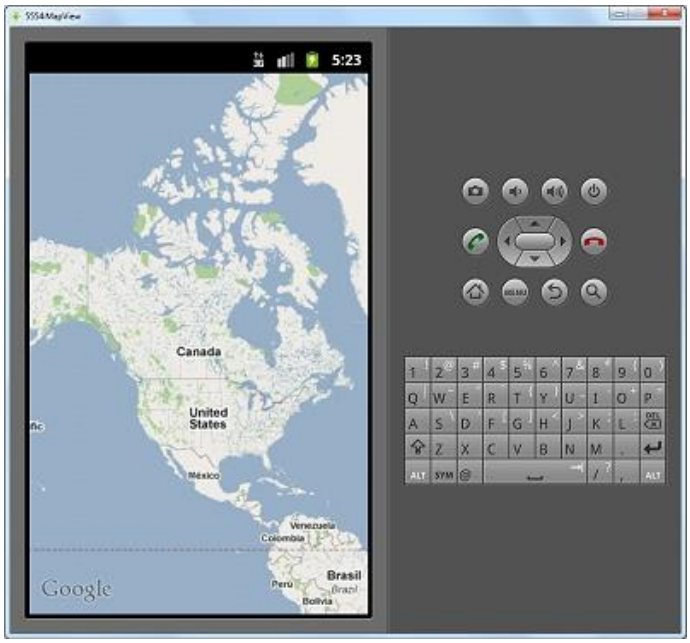


Figure 2.4.1. Google maps API

A MapActivity defines a gateway to the Android Google Maps like API package and other useful map-related utilities. It handles several details behind creating and using a MapView so you don’t to have to worry about them.

The MapView offers the most important features. But a MapActivity provides essential support for the MapView. It manages all the network and filesystem-intensive setup and teardown tasks needed for supporting the map. For example, the MapActivity onResume method automatically sets up network threads for various map-related tasks and caches map section tile data on the filesystem, and the onPause method cleans up these resources. Without this class, all these details would require extra housekeeping that any Activity wishing to include a MapView would have to repeat each time.

Your code won't do much with MapActivity. Extend this class making sure to use only one instance per process, and include a special manifest element to enable the com.google.android.maps package.

The com.google.android.maps package, where MapActivity, MapView, MapController, and other related classes such as GeoPoint and Overlay reside, isn't a standard package in the Android library. This manifest element pulls in support for the Google maps package. Once you include the uses-library element and write a basic Activity that extends MapActivity, you can start writing the main app features with a MapView and related Overlay classes.

Using a MapView

Android offers MapView as a limited version of the Google Maps API in the form of a View for your Android application. A MapView displays tiles of a map, which it obtains over the network as the map moves and zooms, much like the web version of Google Maps.

Android supports many of the concepts from the standard Google Maps API through the MapView. For instance, MapView supports a plain map mode, a satellite mode, a street-view mode, and a traffic mode. When you want to write something on top of the map, draw a straight line between two points, drop a marker, or display full-sized images, you use an Overlay.

3.1 Creating application interface

Nowadays Android operating system is developing highly. There are lots of mobile applications for Android operating system devices. Now I'm going to create a map application for Android operating system. Let's start our creation. for creating we need Eclipse, SDK and Android Virtual Device. Firstly Eclipse is run.

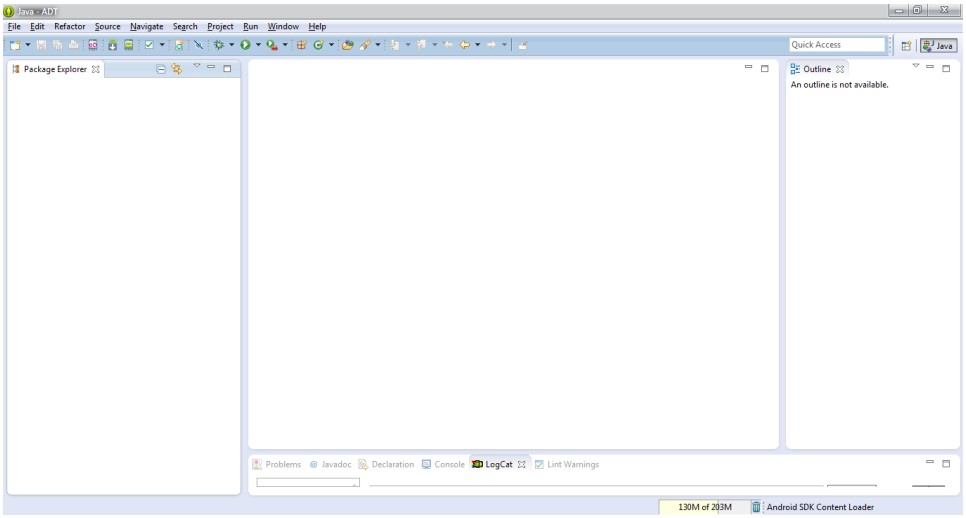


Figure 3.1.1. Main window of Eclipse.

That window is main window of Eclipse. When we use it first time, it will be empty. There is not any project or any library of Android. For creating we must do this:

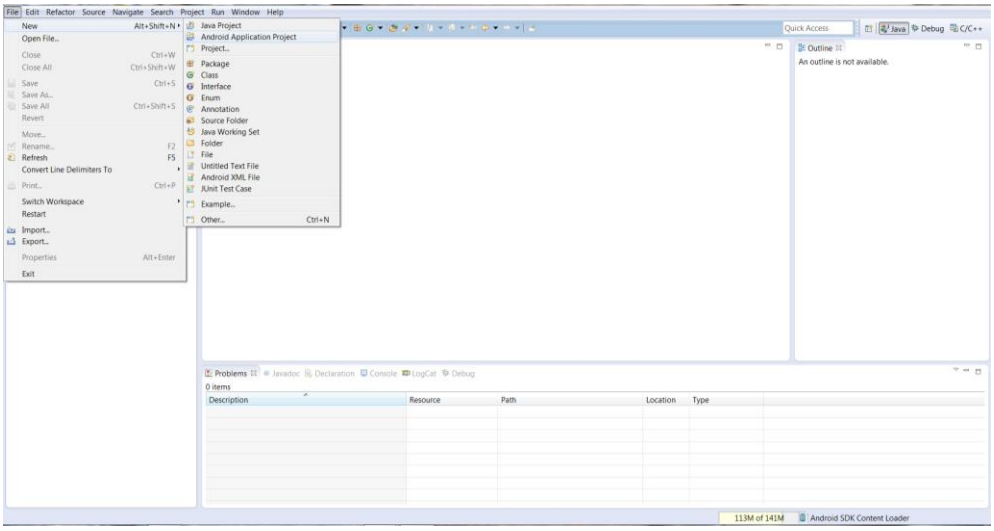


Figure 3.1.2. Creating a new Android project.

We can see above window. There are different parts. They do many functions in process of creating. But now we have to create a new project for our

application. For this we must select menu File then we will select Android Application Project.

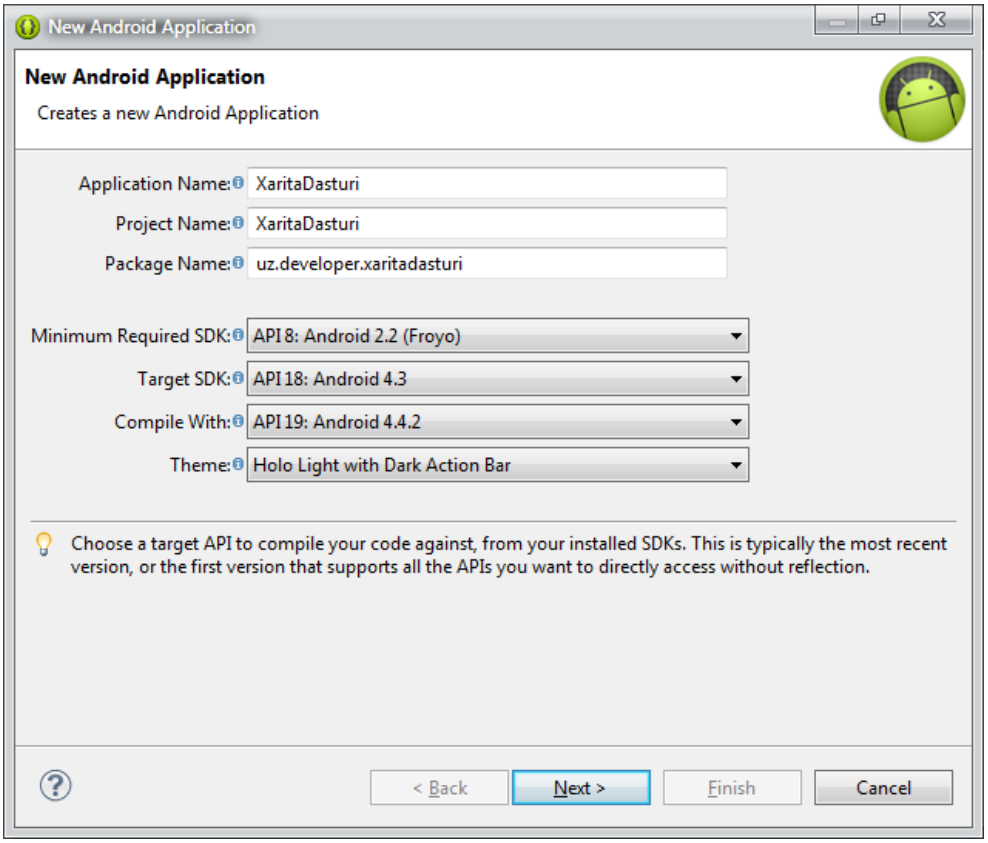


Figure 3.1.3. Naming the project.

After that we have to complete and choose one of the items. Firstly, there is Application Name and we will write something like that, and Project name is similar with Application Name. It is usual. If you want to change that name you can change. It is not harmful. And we must choose Minimum Required SDK maybe it would be like that. If you want to change theme you can change it. Then we can press Next.

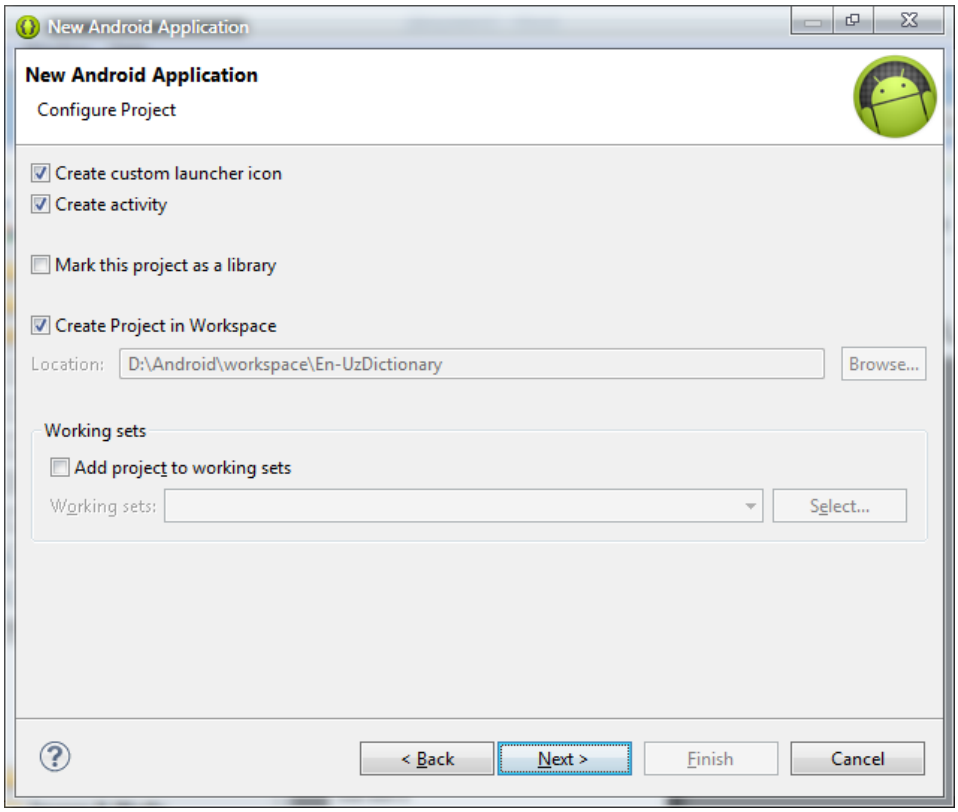


Figure 3.1.4. Configure project.

It is usually like that. Now we will not change those modes. We will press Next.

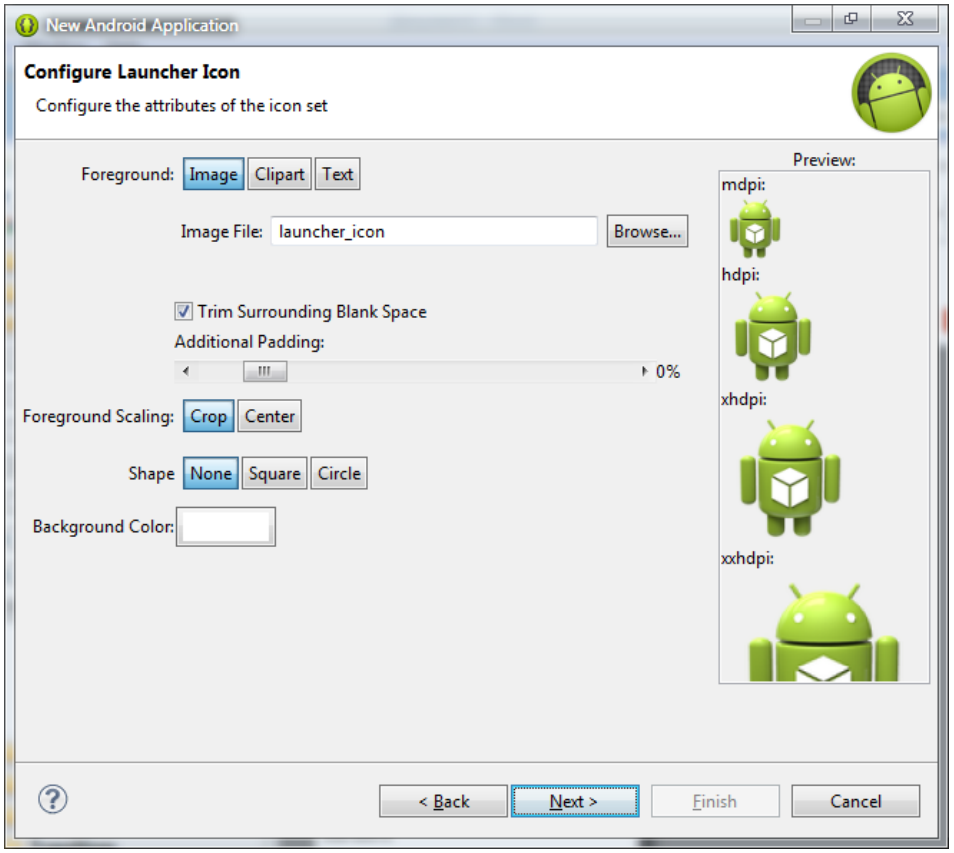


Figure 3.1.5. Configure Launcher Icon.

From here you can modify Configure Launcher Icon if you want to do it. But is not important. Only it serves for Icon of application. It will be see in our mobiles before when we use it.

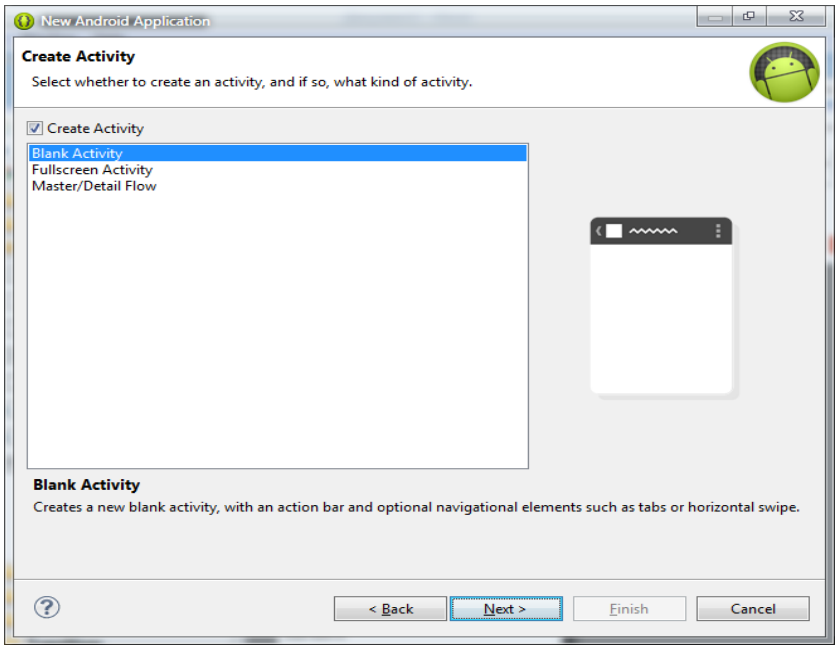


Figure 3.1.6. Create Activity

This is Blank Activity window. From here we can choose activities type. It is usual Blank Activity. And we will click Next.

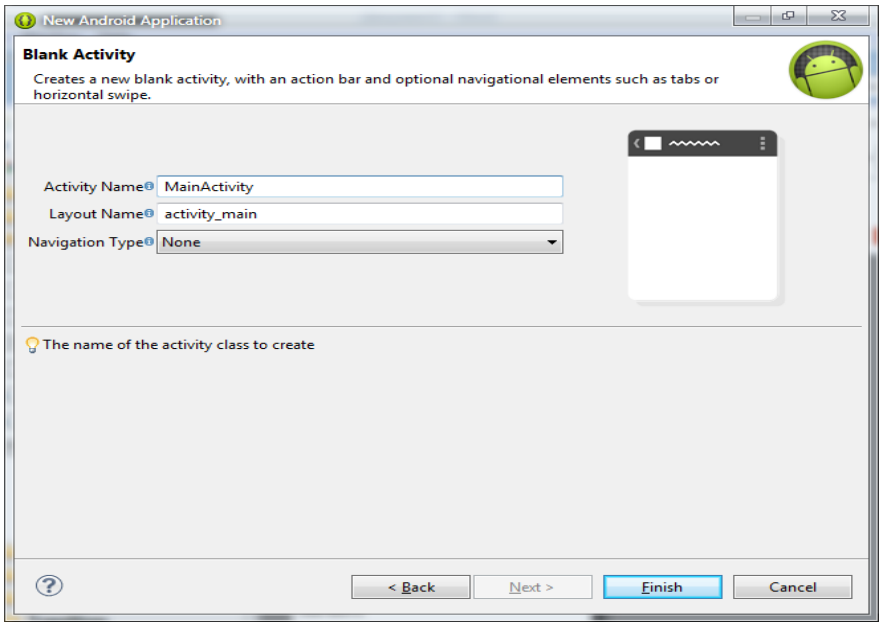


Figure 3.1.7. Blank Activity.

By this window we can change Activity Name, Layout Name and Navigation type. After that we can click Finish.

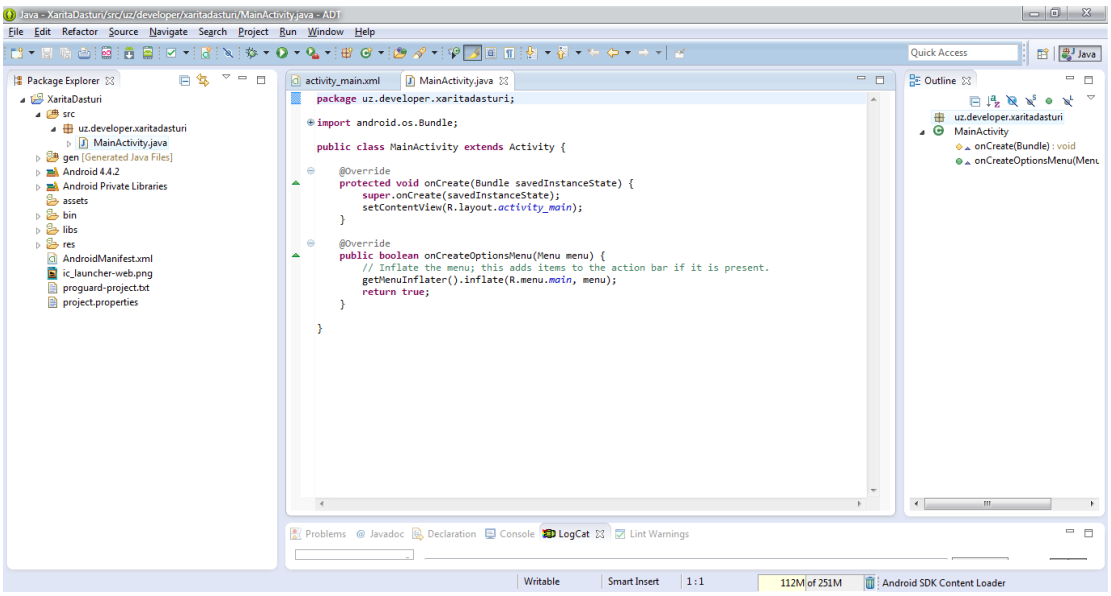


Figure 3.1.8. Desktop.

This is main desktop of our application. There are line of menus, desktop, components and status line. And there are some kind of files. For example MainActivity, MainXML, res and etc.

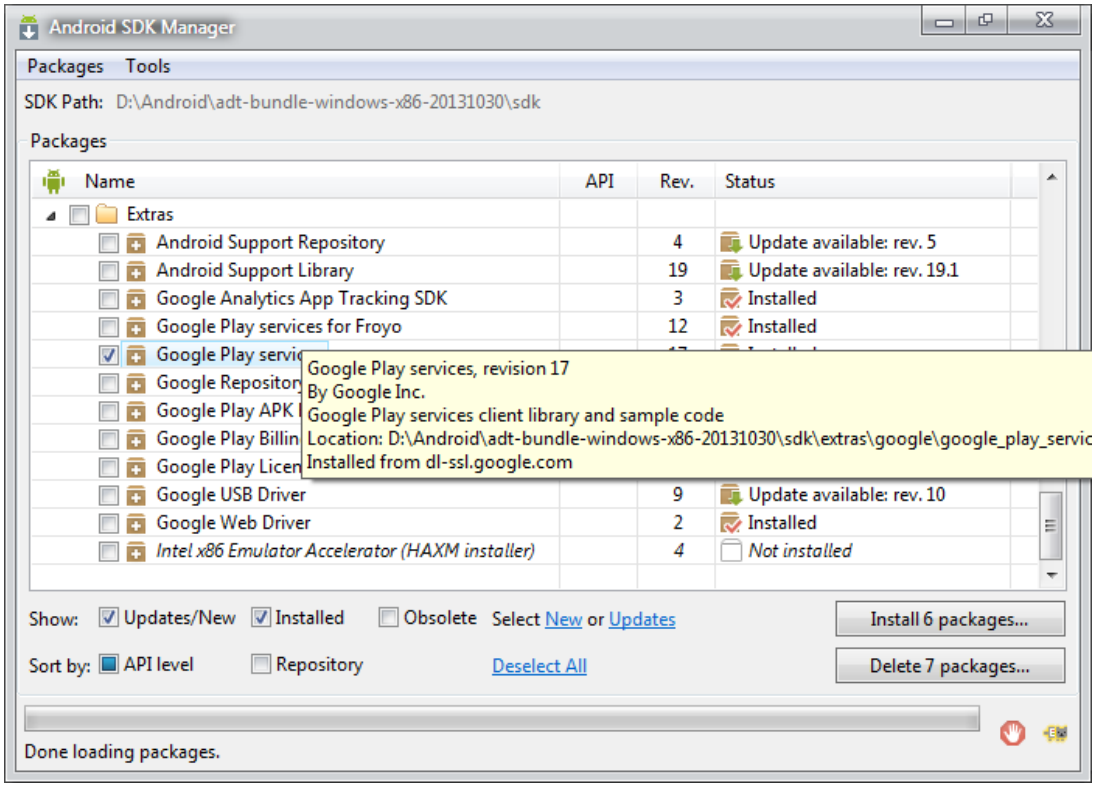


Figure 3.1.9. Android SDK Manager.

By this window we can choose library which we needed. We need Google Play Service and we have to choose this one. After that we will click Install

Packages. After about ten minutes it will be finished. Note, if Google Play Service was installed we have not choose this library.

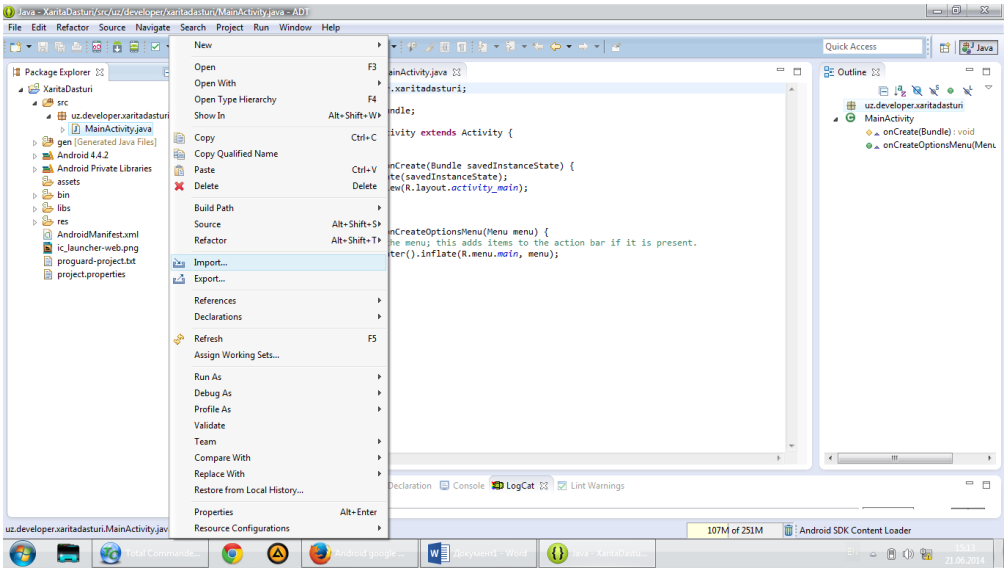


Figure 3.1.10. Importing the library.

After installing we have to import that library to our project for using it. For that we will click on our project. And click Import.

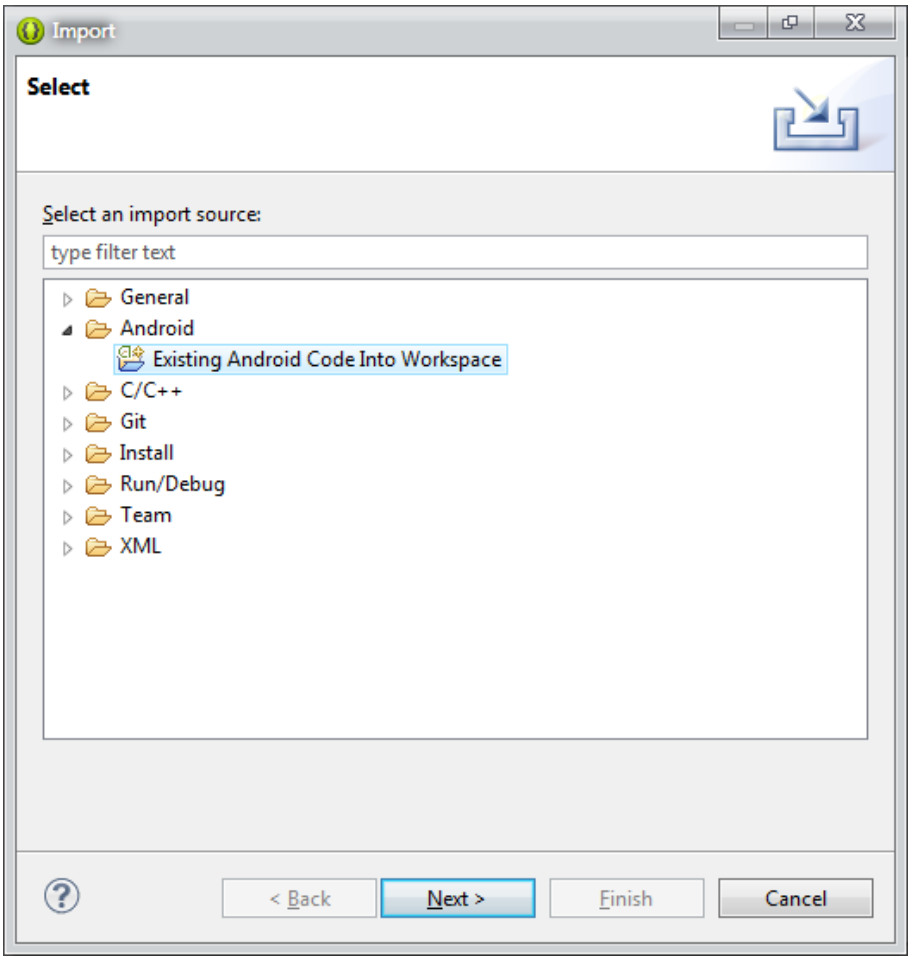


Figure 3.1.11. Importing the library (continued)

There are several folders and we must choose folder Android. There is one file. And we will click on this file.

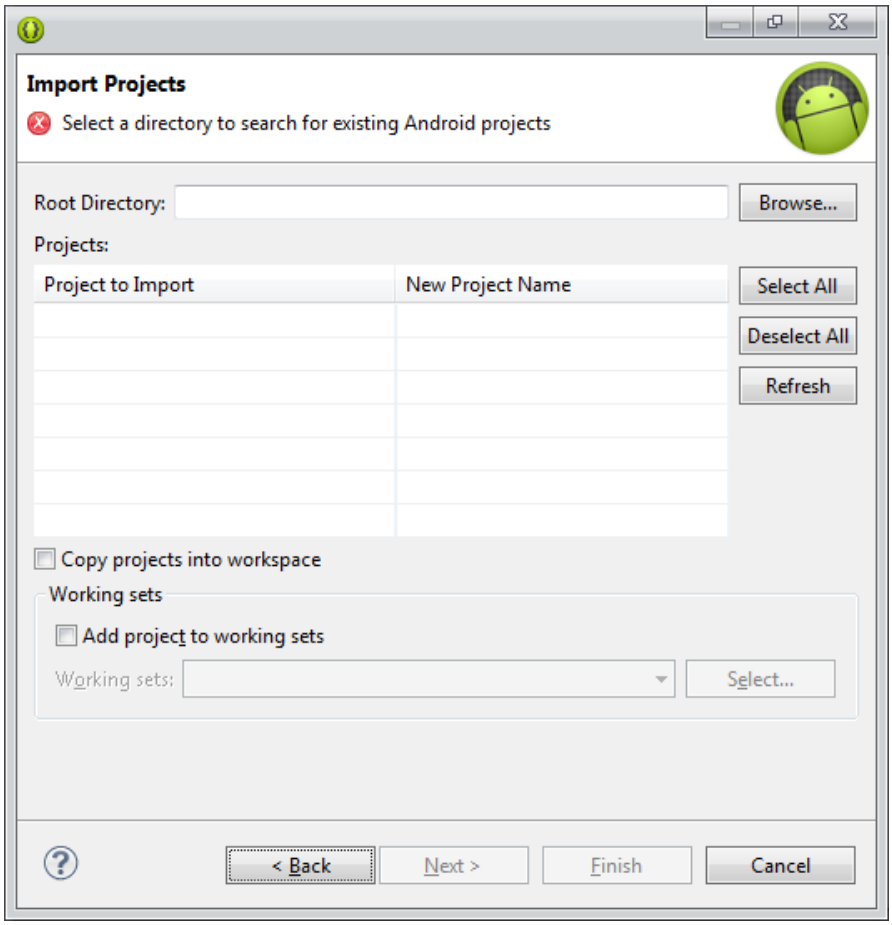


Figure 3.1.12. Window Import Project.

By this window we can do import library to our project. Now we are going to add a new library. It is Google Play Service. It is situated in the folder of Eclipse. We have to click Browse.

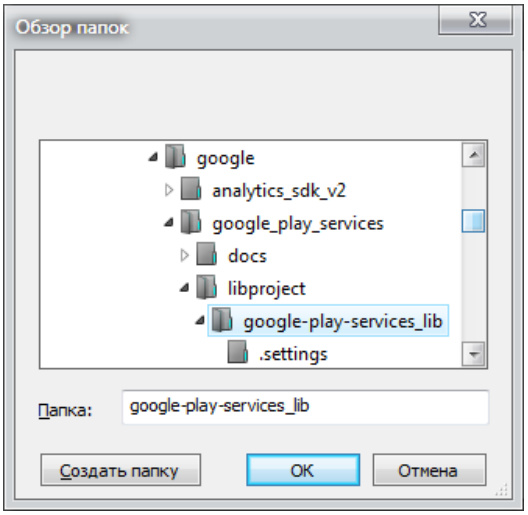


Figure 3.1.13. Choosing google Play Service of folder tree.

From that window we have to find Google Play Service. We will select it and click button OK.

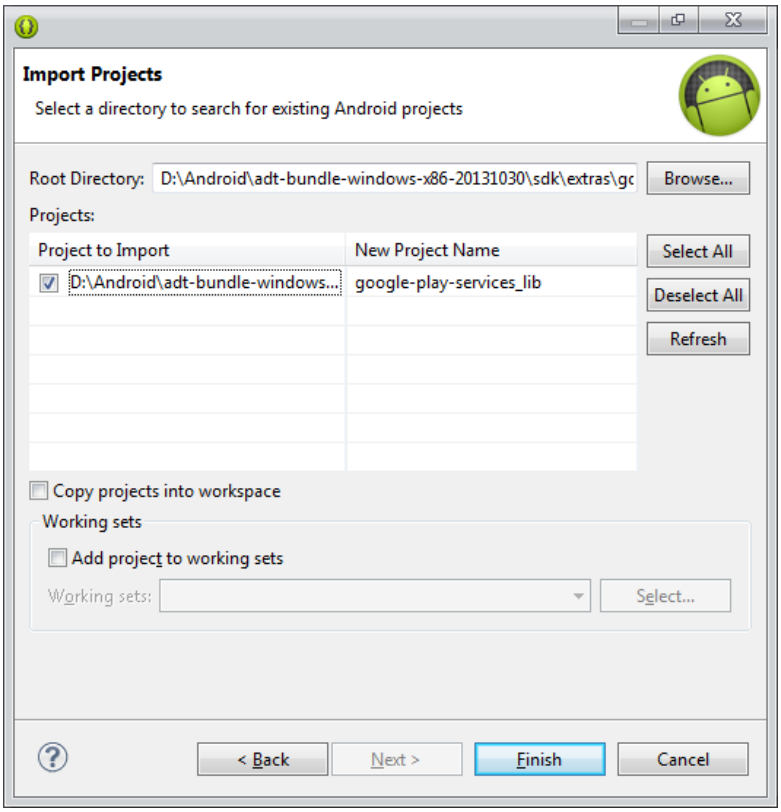


Figure 3.1.14. Import project (continued).

We can see in this window. Our library was added to list of project. We can click button Finish.

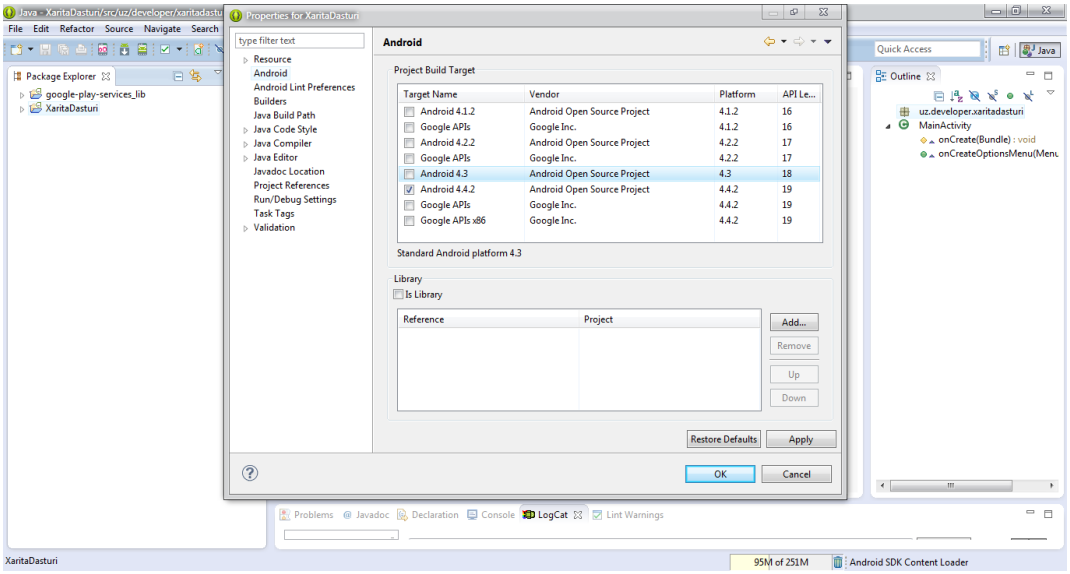


Figure 3.1.15. Propererties.

From that window we have to click button Add and click button OK. If there is nothing in the list of library we have to check the imported project.

After adding the library we can use Google Maps API. But we have one problem. It is not problem. But it takes from us several time. Google Maps API works with key. If we don't have key, we cannot use Google Maps. For getting key we have to enter to https://console.developers.google.com. Before that we must take our Android SHA1 code. For that we must enter to Preferences - > Android - > Build.

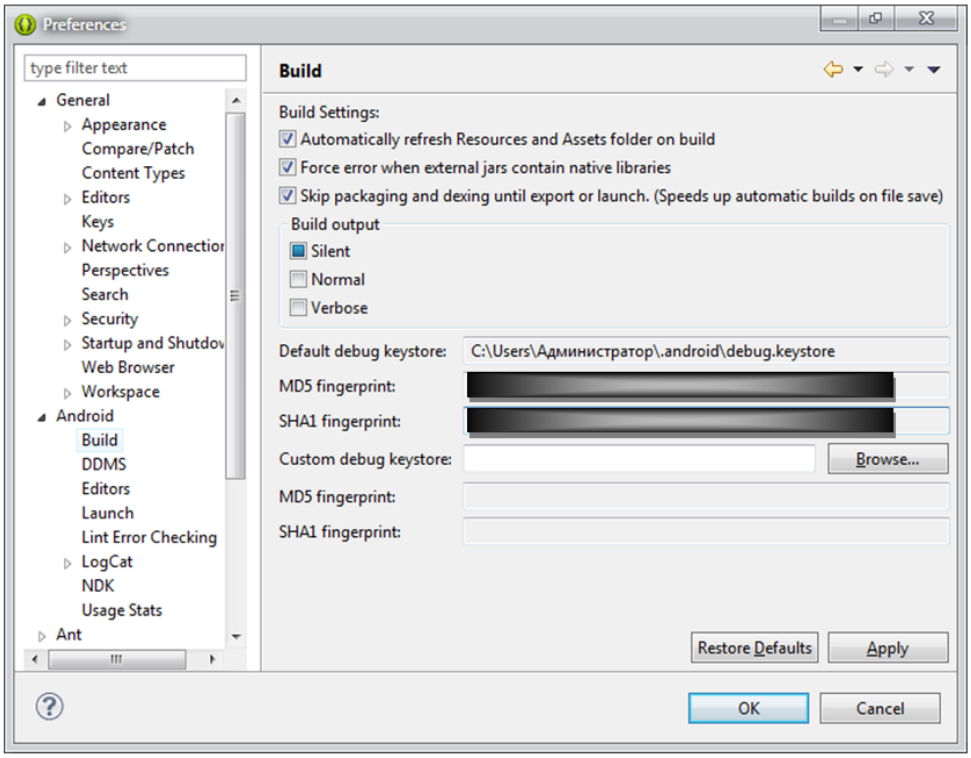


Figure 3.1.16. Preferences window.

From this window we have to copy SHA 1 fingerprint code. And enter the website of Google.

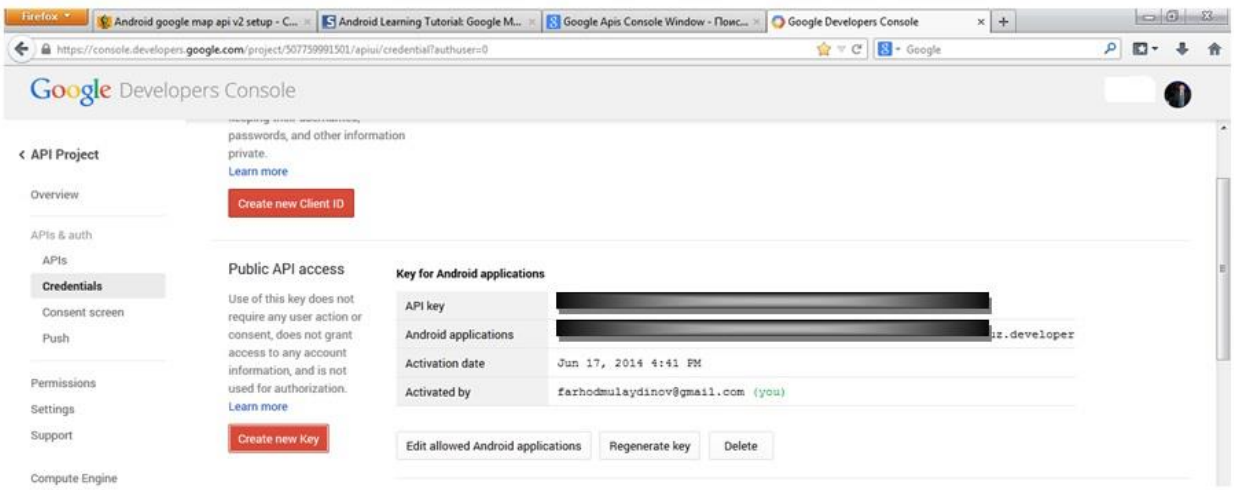


Figure 3.1.17. Getting key of Google Maps API.

After getting key we can write some codes for our application. After that we will have window below.

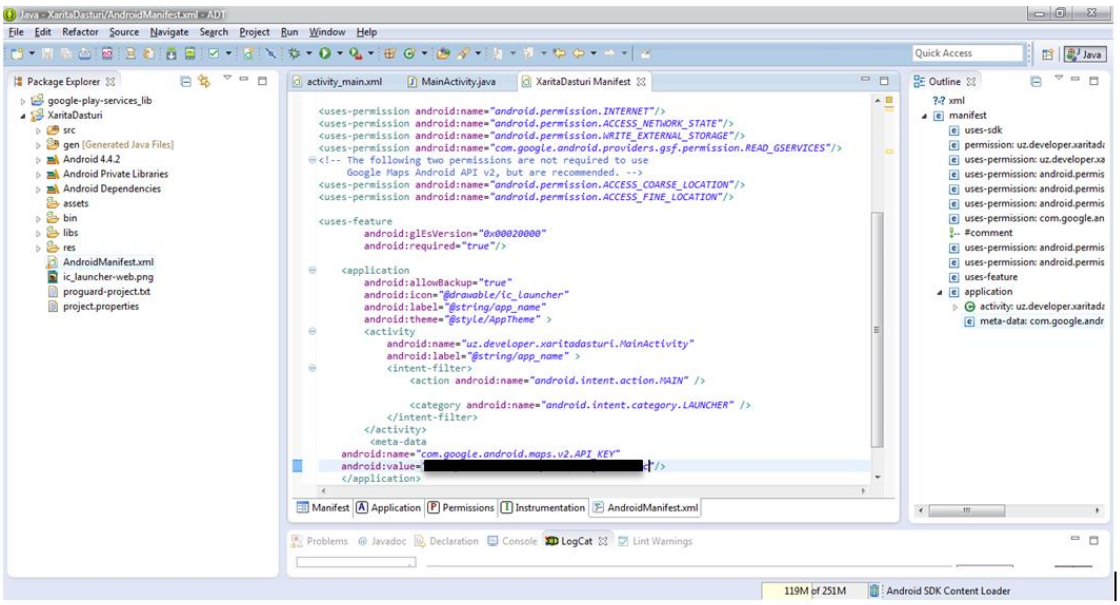


Figure 3.1.18. Process of writing to Android Manifest.xml

There are Permission, application and import codes. Through this we can connect Google Maps with our Android application project. After writing codes, we will have window below.

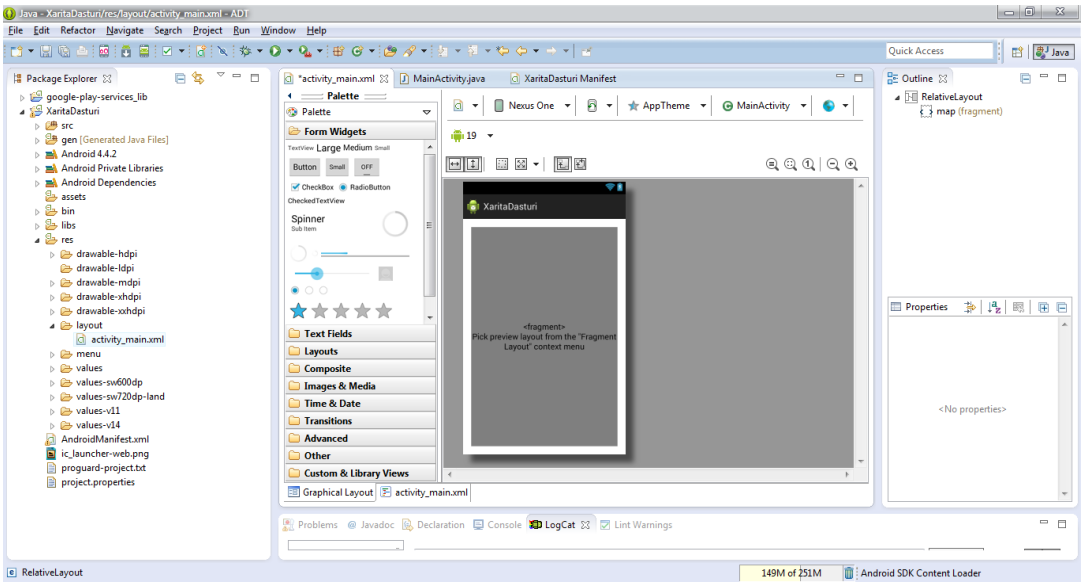


Figure 3.1.19. Graphical layout.

This is the interface of our application. And we have to start Android Virtual Device or Genymotion Emulator for testing the application.

3.2 Writing codes for the application

We imported the library of Google Play Service, got key, and we have emulator. Now we must write code that our application must work. For that we will do some changes. Firstly we will write for AndroidManifest.xml. Open the AndroidManifest.xml and some codes like that.

```
     <permission
         android:name="com.example.osman.permission.MAPS_RECEIVE"
             android:protectionLevel="signature"/>
     <uses-permission
android:name="com.example.osman.permission.MAPS_RECEIVE"/>
      <uses-permission
android:name="android.permission.INTERNET"/>
     <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
     <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
     <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_G
SERVICES"/>
     <!-- The following two permissions are not required to use
          Google Maps Android API v2, but are recommended. -->
     <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
     <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
      <uses-feature
             android:glEsVersion="0x00020000"
             android:required="true"/>
```

Listing 3.2.1 Give permission to the application

After writing codes for AndroidManifest.xml we have to write codes for activating key.

```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
```

```
    android:value="YOUR_KEY"/>
```

Listing 3.2.2. Activating key.

Then for placing the map we have to write codes. In that window you can see those codes which wrote for MainLayout.

```
<span style="color: black; font-family: Consolas, 'Courier New', Courier, mono; font-size: 9pt; white-space: pre;"></span><RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >
 <fragment
          android:id="@+id/map"
          android:layout_width="match_parent"
          android:layout_height="match_parent"

class="com.google.android.gms.maps.SupportMapFragment"/>
 </RelativeLayout>
```

Listing 3.2.3. Codes of MainLayout.xml

And MainActivity.java must be like this:

```
    <span style="color: black; font-family: Consolas, 'Courier New', Courier, mono; font-size: 9pt; white-space: pre;"></span><RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
         xmlns:tools="http://schemas.android.com/tools"
         android:layout_width="match_parent"
         android:layout_height="match_parent"
         android:paddingBottom="@dimen/activity_vertical_margin"
         android:paddingLeft="@dimen/activity_horizontal_margin"
         android:paddingRight="@dimen/activity_horizontal_margin"
```

```
        android:paddingTop="@dimen/activity_vertical_margin"
        tools:context=".MainActivity" >
    <fragment
            android:id="@+id/map"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
class="com.google.android.gms.maps.SupportMapFragment"/>
    </RelativeLayout>
```

Listing 3.2.4. Codes for MainActivity.

Finally, for using completely from a map application we need Zoom components. Because, the map is very large and we can see all places very little. That's why we need Zoom component. For working zoom component we must write codes also like this:

```
    GoogleMap googleMap;
            googleMap                                                =
((SupportMapFragment)(getSupportFragmentManager().findFragmentBy
Id(R.id.map))).getMap();
            LatLng latLng = new LatLng(-33.796923, 150.922433);
            googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
            googleMap.addMarker(new MarkerOptions()
                    .position(latLng)
                    .title("My Spot")
                    .snippet("This is my spot!")
    .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptor
Factory.HUE_AZURE)));
            googleMap.getUiSettings().setCompassEnabled(true);
         googleMap.getUiSettings().setZoomControlsEnabled(true);
    googleMap.animateCamera(CameraUpdateFactory.newLatLngZoom(la
tLng, 10));
```

Listing 3.2.5. Working with ZoomControl

Safety of life activities

Requirements to condition of the work on computer

Requirements to computer premiseses

Microclimate premisesses. The Parameters microclimate and their feature.

A Condition of the air ambience of the premises. The Qualitative composition of the air. The Contents of dust and other polluting material in worker to zone. An Electromagnetic fields and x-ray radiation. The Natural ventilation; The Artificial ventilation; The Air conditioning.

Illumination

The Natural illumination. The Influence to solar radiation on premiseses.

The Color decorating of the interior of the premises as method of protection solar radiation:

- An Artificial illumination. The System of the artificial illumination. The Choice lamp;
- A Standertization of the artificial illumination.

Production noise

- A Standertization of the production noise in computer premises;
- Barriering design and internal sewer of the premises.

Electrical safety

- A Danger of the defeat by electric current;
- An Earth of the computer equipment, requirements;
- A Facility collective and the individual safety when work with equipment;
- A Rendering first help before vrachebnoy at electrical trauma.

Fireman safety

A Reasons fire.

- A Primary facilities fire;

-A Way to evacuations in the event of arising the fire;

-A Fireman water-supply.

On each point, previously than proceed with development action, is conducted analysis existing bad and dangerous production factor, under development actions must be concrete and provide the decision sanitary-hygenic, technical and organizing problems, which prevent the negative factors and create the comfortable of the condition of the work.

The Section is terminated by formation to instructions safety when functioning(working) on computer.

Calculation Intensity Noise

The Noise and vibration present itself professional bad if their intensity exceeds the certain level. For fight with noise are used general and the individual means of protection. Correct planning has Big importance and accomodation enterprise and their separate shop to other noisy enterprise. The Noise is in production premiseses possible vastly to reduce facing a sewer material. When designing and installation different mechanism equipments necessary to provide possibility of the reduction to vibrations and noise to account of the installing the equipment on special shock absorber, reduction revolving details, change the striking interaction unaccented In given problem necessary:

1. Define the total intensity of the noise from three sources on given worker a place;
2. Define the intensity of the noise if wall and ceiling cover material;
3. Formulate the findings.

The Methods of the calculation:

1. The Calculation of the change level to intensities of the noise with change the distance R from the source of the noise is produced on formula:

$$L_R \approx L_1 - 20 \lg R - 8, \partial Б , (4.1)$$

where LR and L1 - a level to intensities of the noise of the source on distance R metre and one metre accordingly.

If between source of the noise and worker revenge there is wall-barrier, level to intensities of the noise falls on N db

$$N = 14,5 \lg G = 15, \partial Б, (4.2)$$

where G - a mass one m2 wall-barriers

Level to intensities of the noise on worker place with account of the influence wall-barriers is defined as

$$L'_R = L_R - N, \partial Б, (4.3)$$

Total intensity of the noise two sources with level LA and LV, is defined as

$$L_\Sigma = L_A + \Delta L, \partial Б, (4.4)$$

where LA - most from two summiruemых level, db;

ΔL - an adjustment, hanging from difference level, is defined on table 4.1.

In table 4.2. consider the level to intensities of the noise, with provision for influences wall-barriers.

At determination of the total power several sources summation follows to conduct consecutively, as from the most intensive.

Follows to take into account that L_Σ it is defined for three sources of the noise and each source is considered with corresponding to wall-barrier.

The Parameters (the type of the material, thickness and mass 1 m2) wall-barriers to take from table 4.3.

Table 4.1

Source level Difference La-Lv, ,db	0	1	2	3	4	5	6	7	8	9	10	15	20
Adjustment, ΔL ,db	3,0	2,5	2,0	1,8	1,5	1,2	1	0,8	0,6	0,5	0,4	0,2	0

Table 4.2

Raw data		Last numeral of the number of the student ticket									
		1	2	3	4	5	6	7	8	9	0
Source of the noise 1	R,м	2,5	2,0	3	3,5	4	4,5	5	5,5	6	6,5
	L_1,dB	80	90	95	100	100	110	100	90	90	100
	№ wall-barriers	1	2	3	4	5	6	7	8	9	10
Source of the noise 2	R,м	7	7,5	8	8,5	9	9,5	8,5	8,5	8	7,5
	L_1,dB	110	100	90	80	80	80	90	90	100	110
	№ wall-barriers	11	12	13	14	15	15	14	13	12	11
Source of the noise 3	R, m	7	6,5	6	5,5	5	4,5	4	3,5	3	2.5
	L_1,dB	95	90	95	100	105	110	105	100	95	90
	№ wall-barriers	10	9	8	7	6	5	4	3	2	1

Table 4.3

№	Material and designs	Thickness to designs, m	Mass 1/m2 barriers, kgs
1	Wall brick	0,12	250
2	Wall brick	0,25	470
3	Wall brick	0,38	690
4	Wall brick	0,52	934
5	Paperboard in several layers	0,02	12

6	Paperboard in several layers	0,04	24
7	Voylok	0,025	8
8	Voylok	0,05	16
9	Reinforced concrete	0,1	240
10	Reinforced concrete	0,2	480
11	Wall from shlakobetona	0,14	150
12	Stena iz shlakobetona	0,28	300
13	Partiton from boards by thickness 0,02 m, with two sides	0,06	70
14	Partiton from boards by thickness 0,01 m, with two sides	0.18	95
15	Gypsum partiton	0,11	117

At determination of the intensities of the noise after covering sewer and ceiling material for simplicity is allowed neglect the action direct sound fine, consider that wall- barriers are found inwardly premiseses and on influences do not render.

Total sewer and ceiling is defined as.

$$M = S_{nm} \cdot \alpha = S_c \cdot \beta = S_{nm} \cdot \gamma, ед.погл., (4.5)$$

where Snm, Ss - accordingly area of the ceiling and sewer of the premises.

Table 4.4

	Penultimate numeral of the number of the student ticket									
	1	2	3	4	5	6	7	8	9	0
S_{nm}, m^2	100	150	200	250	300	350	400	450	500	550
S_c ,m^2	160	180	200	220	250	260	280	300	320	340
α_1,10^{-3}	20	25	30	35	40	45	40	35	30	25
α_2 10^{-2}	95	90	85	80	75	70	75	80	85	90
β_1,10^{-3}	34	33	32	31	30	31	32	33	34	35
β_2,10^{-2}	75	80	85	90	95	90	85	80	75	70

α, β, γ - accordingly factors of the absorption material, which cover ceiling, wall and floor.

Is it Here taken into account that area flap and ceiling of the premises are. Reduction to intensities of the noise will form

$$K = 10 \lg \frac{M_2}{M_1}, \partial Б \ , (4.6)$$

where M1, M2,- accordingly without covering sewer and ceiling special material (M1) and after covering such material (M2), ed. has bent.

Importance M1, is calculated with use factor a1 and M2 - with use a2, and the Floor usually material is not covered and at calculation to take that floor parquet.

Level to intensities of the noise on worker place with account of the covering sewer and ceiling material will form

$$L'_{\Sigma} = L_{\Sigma} - K \ , \partial Б \ , (4.7)$$

Conclusion

I am Ikromov Ulughbek student of The Tashkent University of Information Technologies Ferghana Branch. I am a four years student of University.

At this time in Uzbekistan the Government is pain attention to developing to the youth. The president of our Republic wants our children to be healthy, intellectual, cultural, wise and clever. That's why our president has made a decision on learning a foreign language, especially the English language which was admitted on December 10, 2012. That's why I have decided to create my diploma work in English language. I made my diploma work on this theme: "Creating a map application for Android operating system devices". It consists of three main parts. They are analytical part, practical part and implementing part. The analytical part includes main information of the diploma work. During the creating diploma work I learned Eclipse programming environment , Genymotion virtual emulator and Java programming language.

Nowadays mobile technologies are developing highly. In our country the requirement is growing to mobile technologies. So that I aimed to create map application for mobile phones. And through this process I'm going to make comfort for users. It helps to find places easily.

I used to create this application Java programming language and from environment of Eclipse. And after creation I tested my application on the Genymotion emulator. It can work on many versions of Android. During the creating, firstly I made user interface, after that I pasted necessary components. I used on my application Google Maps API and Zoom Control components. And finally, I connected those components by codes.

I hope in the future my application will be useful for everybody.

List of references:

Literatures:

1. “Uzbekistan on the Threshold of the 21st Century: Threats to the Security, Conditions and Guarantees of Progress”, I.A.Karimov. Tashkent: "Uzbekistan", 1997.

2. “There Is No Future Without Historical Memory”, I.A.Karimov. Tashkent: "Uzbekistan", 1999.

3. “Android in Action 2nd Edition”, Frank Ableson, Robi Sen, Chris King. USA, “Stamford”, 2011;

4. “The Busy Coder’s Guide to Android Development,” Mark L. Murphy. USA, “Manning Publications”, 2011;

5. “Android Programming Tutorials, 3rd Edition” Mark L Murphy., USA, “CommonsWare”, 2011;

6. “Android Design Patterns” Nudelman G., Canada, “John Wiley & Sons, Inc”, 2013;

Internet resources:

1. http://www.w3schools.com/googleAPI/;

2.http://developer.xamarin.com/guides/android/platform_features/maps_and_location/maps/obtaining_a_google_maps_api_key/;

3. https://developer.android.com/sdk/installing/index.html;

4. http://xslab.com/2014/02/10-great-mind-mapping-apps-for-android/;

5. http://eclipsetutorial.sourceforge.net/totalbeginner.html;

Appendix

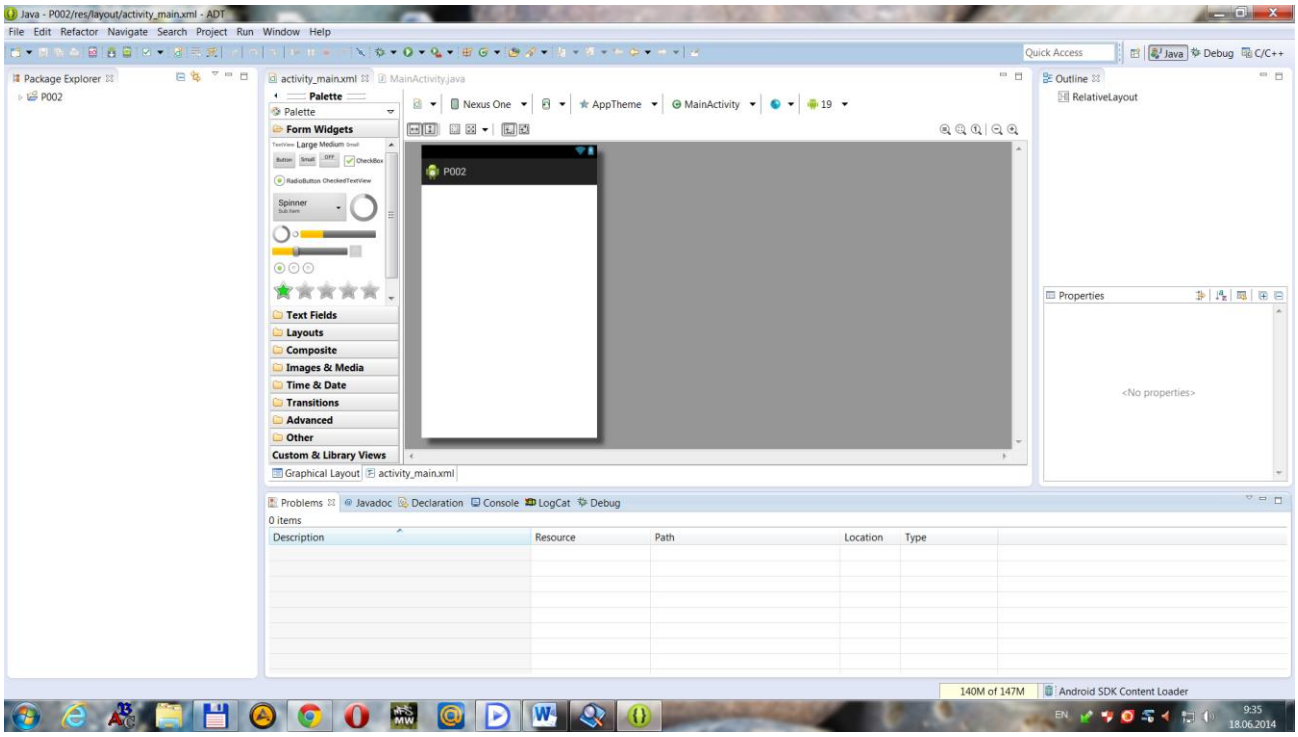


Figure 7.1. The Desktop of Eclipse

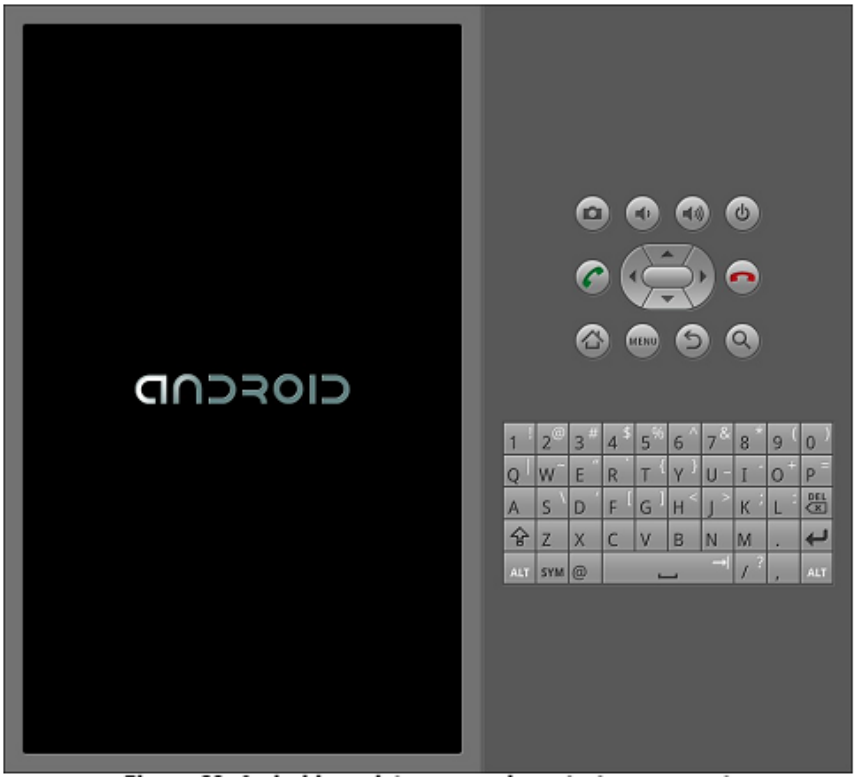


Figure 7.2 Staring Android Virtual Device

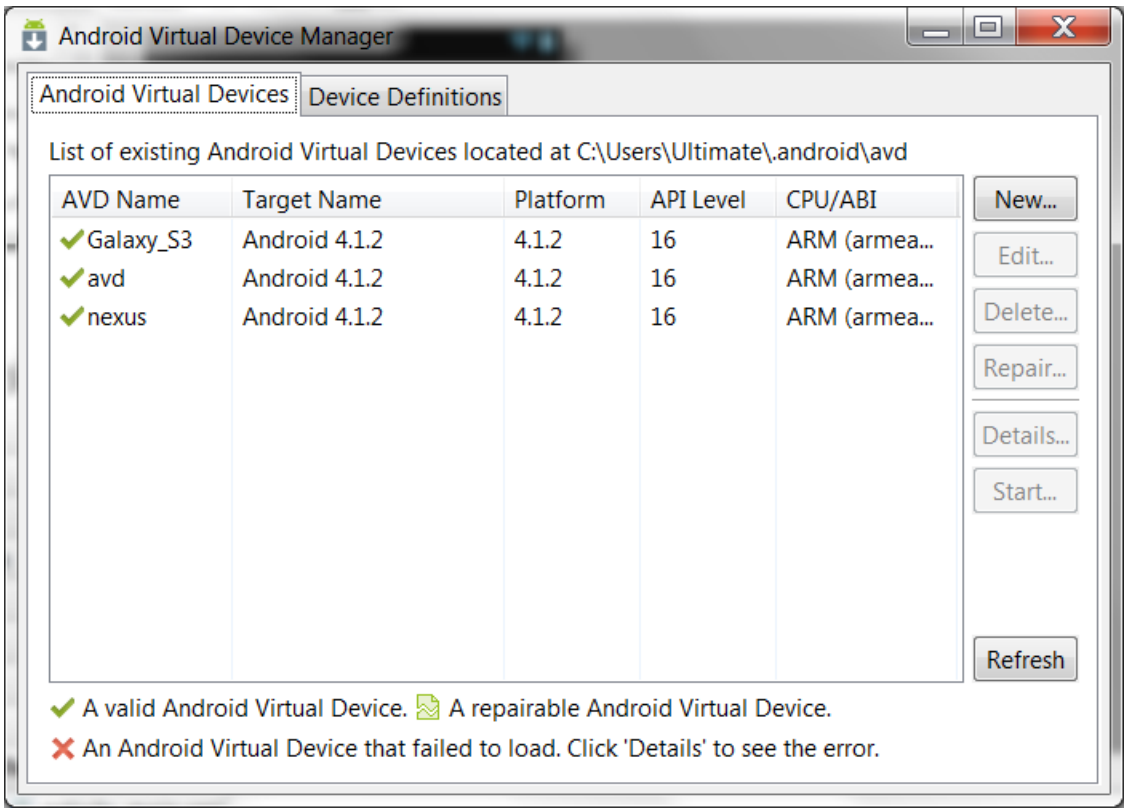


Figure 7.3. The Window of Android Virtual Device Manager

Figure 7.4. MainActivity.java

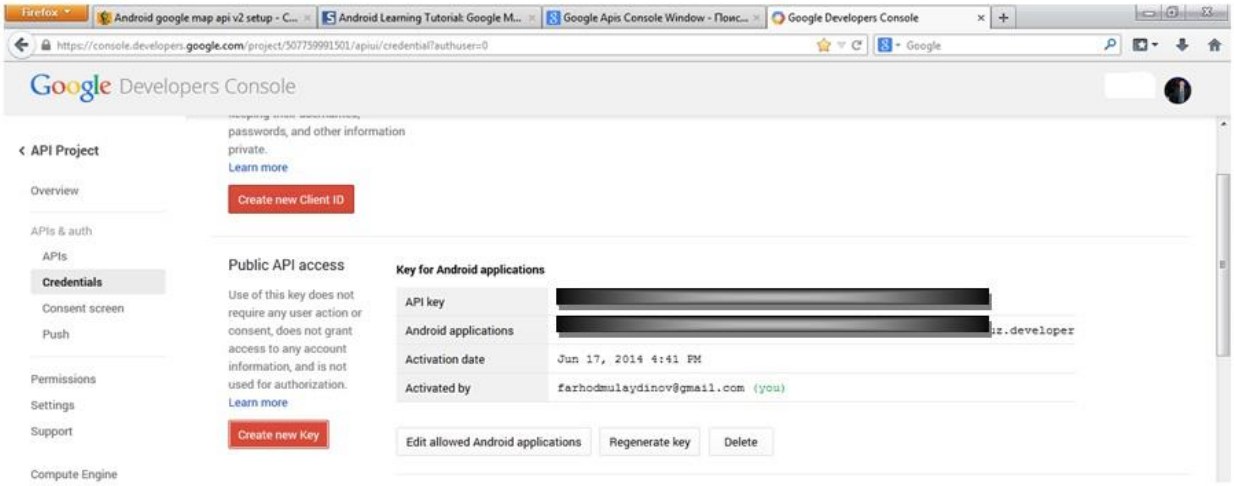


Figure 7.5 Getting key of Google Maps API.

```
     <permission
         android:name="com.example.osman.permission.MAPS_RECEIVE"
             android:protectionLevel="signature"/>
     <uses-permission
android:name="com.example.osman.permission.MAPS_RECEIVE"/>
      <uses-permission
android:name="android.permission.INTERNET"/>
     <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
     <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
     <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_G
SERVICES"/>
     <!-- The following two permissions are not required to use
          Google Maps Android API v2, but are recommended. -->
     <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
     <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
      <uses-feature
             android:glEsVersion="0x00020000"
             android:required="true"/>
```

Listing 7.1 Give permission to the application

After writing codes for AndroidManifest.xml we have to write codes for activating key.

```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="YOUR_KEY"/>
```

Listing 7.2. Activating key.

```
     <permission
         android:name="com.example.osman.permission.MAPS_RECEIVE"
             android:protectionLevel="signature"/>
     <uses-permission
android:name="com.example.osman.permission.MAPS_RECEIVE"/>
      <uses-permission
android:name="android.permission.INTERNET"/>
     <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
     <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
     <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_G
SERVICES"/>
     <!-- The following two permissions are not required to use
          Google Maps Android API v2, but are recommended. -->
     <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
     <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
      <uses-feature
             android:glEsVersion="0x00020000"
             android:required="true"/>
```

Listing 7.3. Give permission to the application