Computer science (CS)

Department

Control it

Under Supervision of

Prof. Dr: Wael Saied

2015

Team Work

- Manar Magdy Mohamed Rabiaa El khashab
- Aya Abd El Gleel Ibrahim Abd El Gleel
- Ahmed Atia Abd Elshafi Mohamed
- Ahmed Ismael Ibrahim Ismael

Acknowledgment

"Praise is to Allah firstly and finally. Who gives us patience brain, respect for science"

"Who taught us letters, we become slaves for him"

We would like to express our gratitude for everyone who helped us during the graduation project, starting with endless thanks for our supervisor Dr. Wael Saied who didn't keep any effort in encouraging us to do a great job, providing our group with valuable information and advices to be better each time. Thanks for the continuous support and kind communication which had a great effect regarding to feel interesting about what we are working on.

Learned skills

*we learn how to work with in a team before starting our career.

*this project helps us to identify priorities, become motivated, avoid temporization, and schedule time effectively.

*learn practical tools and ideas to manage projects, meet deadlines, and achieve both long-term and short-term goals.

*challenge every problem face us and don't raise white flag.

*team work helps us to develop effective ways to manage stress in academia, private life, and the workplace.

*no pain, no gain

Abstract

With the recent and fast developments in technology, smart phones offer many advanced services rather than making calls, so the number of smart phone users increases day by day. User can install and run many smart phone applications that make life easier or enjoyable.

The main objective of this application

The Remote Desktop Manager platform is composed of multiple applications designed to manage all of your connections and credentials.

Remote Desktop Manager acts like a wrapper and a manager around existing technologies, and uses no proprietary protocols. All communications are established using either an external library or third party software.

Remote Desktop Manager for Android is a window on your system that is actually managed by your desktop software, although the RDP protocol is supported, the main objective of the mobile editions is to allow you to have access to the credentials stored in your data source.

Table of contents

4.2 NetBeans

Chapter 5: how this project run

References

Chapter 1

Introduction to android OS

1.1 Android (operating system)

Android is a mobile operating system (OS) based on the Linux kernel and currently developed by Google. With a user interface based on direct manipulation, Android is designed primarily for touchscreen mobile devices such as smartphones and tablet computers, with specialized user interfaces for televisions (Android TV), cars (Android Auto), and wrist watches (Android Wear). The OS uses touch inputs that loosely correspond to real-world actions, like swiping, tapping, pinching, and reverse pinching to manipulate on-screen objects, and a virtual keyboard. Despite being primarily designed for touchscreen input, it has also been used in game consoles, digital cameras, regular PCs, and other electronics. As of 2015, Android has the largest installed base of all operating systems.

As of July 2013, the Google Play store has had over one million Android applications ("apps") published, and over 50 billion applications downloaded.

An April–May 2013 survey of mobile application developers found that 71% of them create applications for Android.

Another 2015 survey found that 40% of full-time professional developers see Android as the "priority" target platform, which is more than iOS (37%) or other platforms.

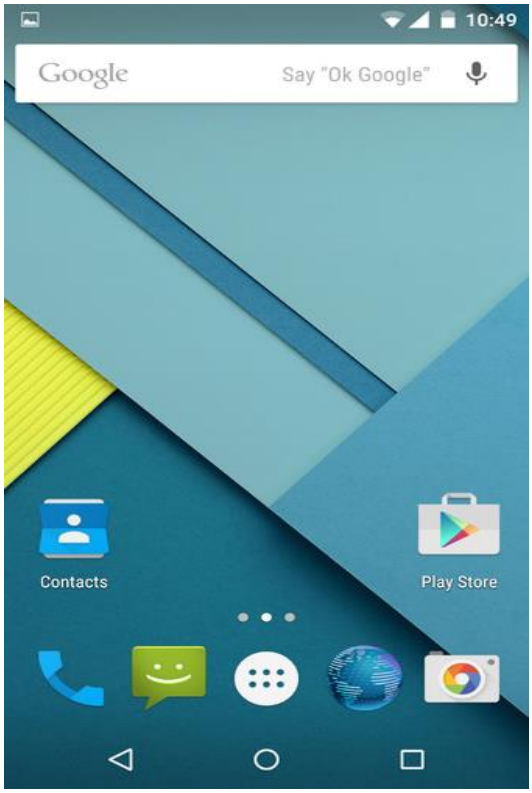
At Google I/O 2014, the company revealed that there were over one billion active monthly Android users, up from 538 million in June 2013

Android's source code is released by Google under open source licenses, although most Android devices ultimately ship with a combination of

open source and proprietary software, including proprietary software developed and licensed by Google. Initially developed by Android, Inc., which Google bought in 2005.

Android was unveiled in 2007, along with the founding of the Open Handset Alliance – a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices.

Android is popular with technology companies which require a ready-made, low-cost and customizable operating system for high-tech devices Android's open nature has encouraged a large community of developers and enthusiasts to use the open-source code as a foundation for community-driven projects, which add new features for advanced users or bring Android to devices which were officially, released running other operating systems. The operating system's success has made it a target for patent litigation as part of the so-called "smartphone wars" between technology companies.



1.2 History

Android, Inc. was founded in Palo Alto, California in October 2003 by Andy Rubin (co-founder of Danger), Rich Miner (co-founder of Wildfire Communications, Inc.), Nick Sears (once VP at T-Mobile), and Chris White (headed design and interface development at WebTV) to develop, in Rubin's words, "smarter mobile devices that are more aware of its owner's location and preferences". The early intentions of the company were to develop an advanced operating system for digital cameras. Though, when it was realized that the market for the devices was not large enough, the company diverted its efforts toward producing a smartphone operating system that would rival Symbian and Microsoft Windows Mobile. Despite the past accomplishments of the founders and early employees, Android Inc. operated secretly, revealing only that it was working on software for mobile phones. That same year, Rubin ran out of money. Steve Perlman, a close friend of Rubin, brought him $10,000 in cash in an envelope and refused a stake in the company.

In July 2005, Google acquired Android Inc. for at least $50 million, whose key employees, including Rubin, Miner and White, stayed at the company after the acquisition. Not much was known about Android Inc. at the time, but many assumed that Google was planning to enter the mobile phone market with this move. At Google, the team led by Rubin developed a mobile device platform powered by the Linux kernel. Google marketed the platform to handset makers and carriers on the promise of providing a flexible, upgradable system. Google had lined up a series of hardware component and software partners and signaled to carriers that it was open to various degrees of cooperation on their part Speculation about Google's intention to enter the mobile communications market continued to build through December 2006. An earlier prototype codenamed "Sooner" had a closer resemblance to a BlackBerry phone, with no touchscreen, and a physical, QWERTY keyboard, but was later re-engineered to support a touchscreen, to compete with other announced devices such as the 2006 LG Prada and 2007 Apple iPhone. In September 2007, InformationWeek covered an

Evalueserve study reporting that Google had filed several patent applications in the area of mobile telephony.

On November 5, 2007, the Open Handset Alliance, a consortium of technology companies including Google, device manufacturers such as HTC, Sony and Samsung, wireless carriers such as Sprint Nextel and T-Mobile, and chipset makers such as Qualcomm and Texas Instruments, unveiled itself, with a goal to develop open standards for mobile devices. That day, Android was unveiled as its first product, a mobile device platform built on the Linux kernel version 2.6.25. The first commercially available smartphone running Android was the HTC Dream, released on October 22, 2008.

In 2010, Google launched its Nexus series of devices – a line of smartphones and tablets running the Android operating system, and built by manufacturing partners. HTC collaborated with Google to release the first Nexus smartphone, the Nexus One. Google has since updated the series with newer devices, such as the Nexus 5 phone (made by LG) and the Nexus 7 tablet (made by Asus). Google releases the Nexus phones and tablets to act as their flagship Android devices, demonstrating Android's latest software and hardware features. Until January 2015, Google offered several Google Play Edition devices over Google Play, as Google-customized Android phones and tablets that, while not carrying the Google Nexus branding, run an unmodified version of Android.

On March 13, 2013, Larry Page announced in a blog post that Andy Rubin had moved from the Android division to take on new projects at Google. He was replaced by Sunder Pichai, who also continues his role as the head of Google's Chrome division, which develops Chrome OS.

Since 2008, Android has seen numerous updates which have incrementally improved the operating system, adding new features and fixing bugs in previous releases. Each major release is named in alphabetical order after a dessert or sugary treat; for example, version 1.5 "Cupcake" was followed by 1.6 "Donut". Version 4.4.4 "KitKat" appeared as a security-only update; it was released on June 19, 2014,

shortly after 4.4.3 was released. Android 5.0 "Lollipop" was released on November 14, 2014, introducing "material design" as a new design language and one of its key new features; it was followed by two bug fix releases (5.0.1 and 5.0.2).

In 2014, Google launched Android One, a standardized smartphone, mainly targeting customers in the developing world. Android One smartphones running the latest version of Android (e.g. the latest Android 5.1) close to the stock version of the operating system. As of March 3, 2015, the newest version of the Android operating system, 5.1, is available for selected devices including the Android One series, the Nexus 6 phablet, and the Nexus 9 tablet.

From 2010 to 2013, Hugo Barra served as product spokesperson, representing Android at both press conferences and Google I/O, Google’s annual developer-focused conference. Barra's product involvement included the entire Android ecosystem of software and hardware, including Honeycomb, Ice Cream Sandwich, Jelly Bean and KitKat operating system launches, the Nexus 4 and Nexus 5 smartphones, the Nexus 7 and Nexus 10 tablets, and other related products such as Google Now and Google Voice Search, Google’s speech recognition product comparable to Apple’s Siri. In 2013, Barra left the Android team for Chinese smartphone maker Xiaomi.

In May 2015, Google announced Project Braille as a cut-down version of Android that uses its lower levels (excluding the user interface), intended for the "Internet of Things" (IoT) embedded systems.

1.3 Features

Interface

Notifications are accessed by sliding from the top of the display; individual notifications can be dismissed by sliding them away, and may contain additional functions (such as on the "missed call" notification)

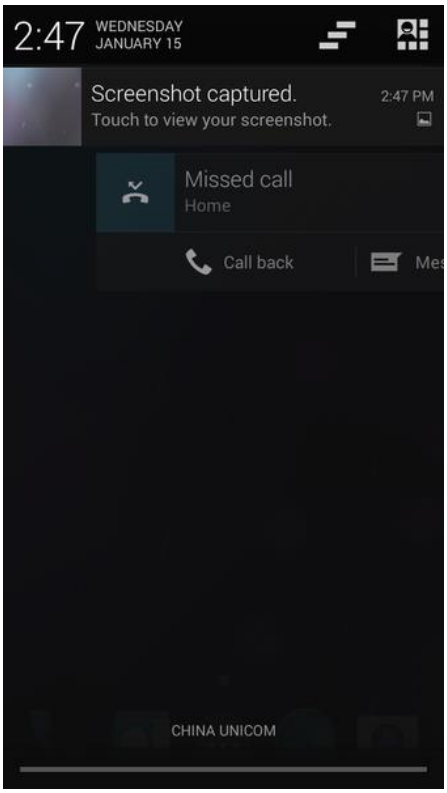
Android's default user interface is based on direct manipulation, using touch inputs, that loosely correspond to real-world actions, like swiping, tapping, pinching, and reverse pinching to manipulate on-screen objects, and a virtual keyboard. The response to user input is designed to be immediate and provides a fluid touch interface, often using the vibration capabilities of the device to provide haptic feedback to the user. Internal hardware such as accelerometers, gyroscopes and proximity sensors are used by some applications to respond to additional user actions, for example adjusting the screen from portrait to landscape depending on how the device is oriented, or allowing the user to steer a vehicle in a racing game by rotating the device, simulating control of a steering wheel.

Android devices boot to the home screen, the primary navigation and information "hub" on Android devices that is analogous to the desktop found on PCs (Android also runs on regular PCs, as described below). Android home screens are typically made up of app icons and widgets; app icons launch the associated app, whereas widgets display live, auto-updating content such as the weather forecast, the user's email inbox, or a news ticker directly on the home screen. A home screen may be made up of several pages that the user can swipe back and forth between, though Android's home screen interface is heavily customizable, allowing the user to adjust the look and feel of the device to their tastes. Third-party apps available on Google Play and other app stores can extensively re-theme the home screen, and even mimic the look of other operating systems, such as Windows Phone. Most manufacturers, and

some wireless carriers, customize the look and feel of their Android devices to differentiate themselves from their competitors.

Present along the top of the screen is a status bar, showing information about the device and its connectivity. This status bar can be "pulled" down to reveal a notification screen where apps display important information or updates, such as a newly received email or SMS text, in a way that does not immediately interrupt or inconvenience the user Notifications are persistent until read (by tapping, which opens the relevant app) or dismissed by sliding it off the screen. Beginning on Android 4.1, "expanded notifications" can display expanded details or additional functionality; for instance, a music player can display playback controls, and a "missed call" notification provides buttons for calling back or sending the caller an SMS message.

Android provides the ability to run applications which change the default launcher and hence the appearance and externally visible behavior of Android. These appearance changes include a multi-page dock or no dock, and many more changes to fundamental features of the user interface.



Applications

Applications ("apps"), which extend the functionality of devices, are written using the Android software development kit (SDK) and, often, the Java programming language that has complete access to the Android APIs. Java may be combined with C/C++, together with a choice of non-default runtimes that allow better C++ support; the Go programming language is also supported since its version 1.4, which can also be used exclusively although with a restricted set of Android APIs. The SDK includes a comprehensive set of development tools,[63] including a debugger, software libraries, a handset emulator based on QEMU, documentation, sample code, and tutorials. Initially, Google's supported integrated development environment (IDE) was Eclipse using the Android Development Tools (ADT) plugin; in December 2014, Google released Android Studio, based on IntelliJ IDEA, as its primary IDE for Android application development. Other development tools are available, including a native development kit (NDK) for applications or extensions in C or C++, Google App Inventor, a visual environment for novice programmers, and various cross platform mobile web applications frameworks. In January 2014, Google unveiled a framework based on Apache Cordova for porting Chrome HTML 5 web applications to Android, wrapped in a native application shell.

Android has a growing selection of third-party applications, which can be acquired by users by downloading and installing the application's APK (Android application package) file, or by downloading them using an application store program that allows users to install, update, and remove applications from their devices. Google Play Store is the primary application store installed on Android devices that comply with Google's compatibility requirements and license the Google Mobile Services software. Google Play Store allows users to browse, download and update applications published by Google and third-party developers; As of July 2013, there are more than one million applications available for Android in Play Store. As of May 2013, 48 billion applications have been installed from Google Play Store and in July 2013, 50 billion applications were installed. Some carriers offer direct carrier billing for Google Play

application purchases, where the cost of the application is added to the user's monthly bill.

Due to the open nature of Android, a number of third-party application marketplaces also exist for Android, either to provide a substitute for devices that are not allowed to ship with Google Play Store, provide applications that cannot be offered on Google Play Store due to policy violations, or for other reasons. Examples of these third-party stores have included the Amazon App store, Get Jar, and Slide Me. F-Droid, another alternative marketplace, seeks to only provide applications that are distributed under free and open source licenses.

Memory management

Since Android devices are usually battery-powered, Android is designed to manage memory (RAM) to keep power consumption at a minimum, in contrast to desktop operating systems which generally assume they are connected to unlimited mains electricity. When an Android application is no longer in use, the system will automatically suspend it in memory; while the application is still technically "open", suspended applications consume no resources (for example, battery power or processing power) and sit idly in the background until needed again. This brings a dual benefit by increasing the general responsiveness of Android devices, since applications do not need to be closed and reopened from scratch each time, and by ensuring that background applications do not consume power needlessly.

Android manages the applications stored in memory automatically: when memory is low, the system will begin killing applications and processes that have been inactive for a while, in reverse order since they were last used (oldest first). This process is designed to be invisible to the user, so that users do not need to manage memory or the killing of applications themselves. As of 2011, third-party task killers were reported by Life hacker as doing more harm than good.

1.4 Hardware

The main hardware platform for Android is the ARM architecture (ARMv7 and ARMv8-A architectures), with x86 and MIPS architectures also officially supported (the latter two became officially supported in later Android versions). Since Android 5.0 "Lollipop", 64-bit variants of all platforms are supported in addition to the 32-bit variants. Unofficial Android-x86 project used to provide support for the x86 and MIPS architectures ahead of the official support.[5][80] Since 2012, Android devices with Intel processors began to appear, including phones[81] and tablets. While gaining support for 64-bit platforms, Android was first made to run on 64-bit x86 and then on ARM64.

As of November 2013, Android 4.4 recommends at least 512 MB of RAM, while for "low RAM" devices 340 MB is the required minimum amount that does not include memory dedicated to various hardware components such as the baseband processor. Android 4.4 requires a 32-bit ARMv7, MIPS or x86 architecture processor (latter two through unofficial ports), together with an OpenGL ES 2.0 compatible graphics processing unit (GPU). Android supports OpenGL ES 1.1, 2.0, 3.0 and 3.1. Some applications may explicitly require a certain version of the OpenGL ES, and suitable GPU hardware is required to run such applications.

Android devices incorporate many optional hardware components, including still or video cameras, GPS, orientation sensors, dedicated gaming controls, accelerometers, gyroscopes, barometers, magnetometers, proximity sensors, pressure sensors, thermometers, and touchscreens. Some hardware components are not required, but became standard in certain classes of devices, such as smartphones, and additional requirements apply if they are present. Some other hardware was initially required, but those requirements have been relaxed or eliminated altogether. For example, as Android was developed initially as a phone OS, hardware such as microphones were required, while over time the phone function became optional. Android used to require an autofocus camera, which was relaxed to a fixed-focus camera if it is even

present at all, since the camera was dropped as a requirement entirely when Android started to be used on set-top boxes.

In addition to running on smartphones and tablets, several vendors run Android natively on regular PC hardware with a keyboard and mouse. In addition to their availability on commercially available hardware, similar PC hardware–friendly versions of Android are freely available from the Android-x 86 projects, including customized Android 4.4. Using the Android emulator that is part of the Android SDK, or by using Blue Stacks or Andy, Android can also run non-natively on x86. Chinese companies are building a PC and mobile operating system, based on Android, to "compete directly with Microsoft Windows and Google Android". The Chinese Academy of Engineering noted that "more than a dozen" companies were customizing Android following a Chinese ban on the use of Windows 8 on government PCs.

1.5 Development

Android is developed in private by Google until the latest changes and updates are ready to be released, at which point the source code is made available publicly. This source code will only run without modification on select devices, usually the Nexus series of devices. The source code is, in turn, adapted by OEMs to run on their hardware. Android's source code does not contain the often proprietary device drivers that are needed for certain hardware components.

The green Android logo was designed for Google in 2007 by graphic designer Irina Blok. The design team was tasked with a project to create a universally identifiable icon with the specific inclusion of a robot in the final design. After numerous design developments based on science-fiction and space movies, the team eventually sought inspiration from the human symbol on restroom doors and modified the figure into a robot shape. As Android is open-sourced, it was agreed that the logo should be likewise, and since its launch the green logo has been reinterpreted into countless variations on the original design.

Update schedule

Google provides major incremental upgrades to Android every six to nine months, with confectionery-themed names, which most devices are capable of receiving over the air. The latest major release is Android 5.0 "Lollipop".

Compared to its chief rival mobile operating system, namely iOS, Android updates are typically slow to reach actual devices. For devices not under the Nexus brand, updates often arrive months from the time the given version is officially released. This is partly due to the extensive variation in hardware of Android devices, to which each upgrade must be specifically tailored, as the official Google source code only runs on their flagship Nexus devices. Porting Android to specific hardware is a time- and resource-consuming process for device manufacturers, who prioritize their newest devices and often leave older ones behind. Hence, older smartphones are frequently not updated if the manufacturer decides it is not worth their time, regardless of whether the phone is capable of running the update. This problem is compounded when manufacturers customize Android with their own interface and apps, which must be reapplied to each new release. Additional delays can be introduced by wireless carriers who, after receiving updates from manufacturers, further customize and brand Android to their needs and conduct extensive testing on their networks before sending the upgrade out to users.

The lack of after-sale support from manufacturers and carriers has been widely criticized by consumer groups and the technology media. Some commentators have noted that the industry has a financial incentive not to upgrade their devices, as the lack of updates for existing devices fuels the purchase of newer ones, an attitude described as "insulting". The Guardian has complained that the method of distribution for updates is complicated only because manufacturers and carriers have designed it that way. In 2011, Google partnered with a number of industry players to announce an "Android Update Alliance", pledging to deliver timely updates for every device for 18 months after its release; however, there has not been another official word about that alliance.

In 2012, Google began decoupling certain aspects of the operating system (particularly core applications) so they could be updated through Google Play Store, independently of Android itself. One of these components, Google Play Services, is a closed-source system-level process providing APIs for Google services, installed automatically on nearly all devices running Android version 2.2 and higher. With these changes, Google can add new operating system functionality through Play Services and application updates without having to distribute an upgrade to the operating system itself. As a result, Android 4.2 and 4.3 contained relatively fewer user-facing changes, focusing more on minor changes and platform improvements.

Linux kernel

Android's kernel is based on one of the Linux kernel's long-term support (LTS) branches. Since April 2014, Android devices mainly use versions 3.4 or 3.10 of the Linux kernel. The specific kernel version depends on the actual Android device and its hardware platform; Android has used various kernel versions since the version 2.6.25 that was used in Android 1.0.

Android's variant of the Linux kernel has further architectural changes that are implemented by Google outside the typical Linux kernel development cycle, such as the inclusion of components like Binder, ashmem, pmem, logger, wakelocks, and different out-of-memory (OOM) handling. Certain features that Google contributed back to the Linux kernel, notably a power management feature called "wakelocks", were rejected by mainline kernel developers partly because they felt that Google did not show any intent to maintain its own code. Google announced in April 2010 that they would hire two employees to work with the Linux kernel community, but Greg Kroah-Hartman, the current Linux kernel maintainer for the stable branch, said in December 2010 that he was concerned that Google was no longer trying to get their code changes included in mainstream Linux. Some Google Android developers hinted that "the Android team was getting fed up with the process," because they were a small team and had more urgent work to do on Android.

In August 2011, Linus Torvalds said that "eventually Android and Linux would come back to a common kernel, but it will probably not be for four to five years". In December 2011, Greg Kroah-Hartman announced the start of Android Mainlining Project, which aims to put some Android drivers, patches and features back into the Linux kernel, starting in Linux 3.3. Linux included the auto sleep and wakelocks capabilities in the 3.5 kernel, after many previous attempts at merger. The interfaces are the same but the upstream Linux implementation allows for two different suspend modes: to memory (the traditional suspend that Android uses), and to disk (hibernate, as it is known on the desktop). Google maintains a public code repository that contains their experimental work to re-base Android off the latest stable Linux versions.

The flash storage on Android devices is split into several partitions, such as /system for the operating system itself, and /data for user data and application installations. In contrast to desktop Linux distributions, Android device owners are not given root access to the operating system and sensitive partitions such as /system are read-only. However, root access can be obtained by exploiting security flaws in Android, which is

used frequently by the open-source community to enhance the capabilities of their devices, but also by malicious parties to install viruses and malware.

Android is a Linux distribution according to the Linux Foundation, Google's open-source chief Chris DiBona, and several journalists. Others, such as Google engineer Patrick Brady, say that Android is not Linux in the traditional Unix-like Linux distribution sense; Android does not include the GNU C Library (it uses Bionic as an alternative C library) and some of other components typically found in Linux distributions.

Software stack

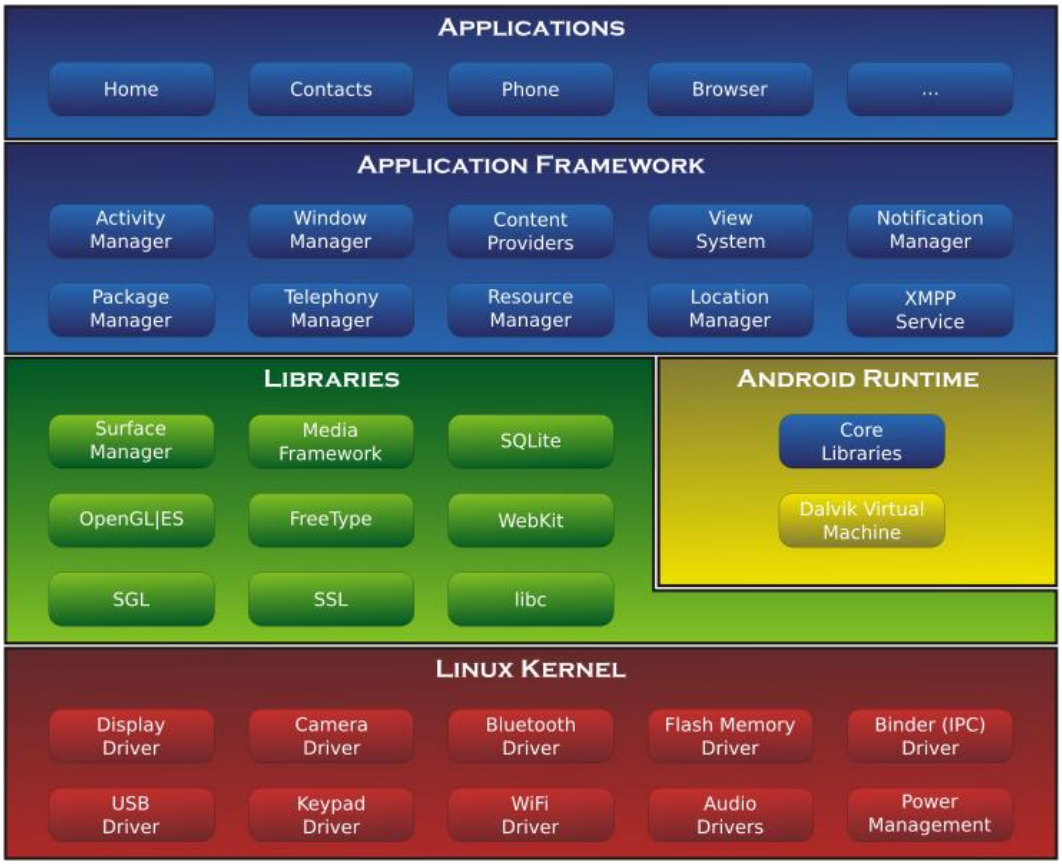
On top of the Linux kernel, there are the middleware, libraries and APIs written in C, and application software running on an application framework which includes Java-compatible libraries based on Apache Harmony. Development of the Linux kernel continues independently of other Android's source code bases.

Until version 5.0, Android used Dalvik as a process virtual machine with trace-based just-in-time (JIT) compilation to run Dalvik "dex-code" (Dalvik Executable), which is usually translated from the Java bytecode. Following the trace-based JIT principle, in addition to interpreting the majority of application code, Dalvik performs the compilation and native execution of select frequently executed code segments ("traces") each time an application is launched. Android 4.4 introduced Android Runtime (ART) as a new runtime environment, which uses ahead-of-time (AOT) compilation to entirely compile the application bytecode into machine code upon the installation of an application. In Android 4.4, ART was an experimental feature and not enabled by default; it became the only runtime option in the next major version of Android, 5.0.

Android's standard C library, Bionic, was developed by Google specifically for Android, as a derivation of the BSD's standard C library code. Bionic itself has been designed with several major features specific to the Linux kernel. The main benefits of using Bionic instead of the GNU C Library (glibc) or uClibc are its smaller runtime footprint, and optimization for low-frequency CPUs. At the same time, Bionic is licensed under the terms of the BSD license, which Google finds more suitable for the Android's overall licensing model.

Aiming for a different licensing model, toward the end of 2012 Google switched the Bluetooth stack in Android from the GPL-licensed BlueZ to the Apache-licensed Blue Droid.

Android does not have a native X Window System by default, nor does it support the full set of standard GNU libraries. This made it difficult to port existing Linux applications or libraries to Android, until version r5 of the Android Native Development Kit brought support for applications written completely in C or C++.Libraries written in C may also be used in applications by injection of a small shim and usage of the JNI.



Open-source community

Android has an active community of developers and enthusiasts who use the Android Open Source Project (AOSP) source code to develop and distribute their own modified versions of the operating system. These community-developed releases often bring new features and updates to devices faster than through the official manufacturer/carrier channels, albeit without as extensive testing or quality assurance; provide continued support for older devices that no longer receive official updates; or bring Android to devices that were officially released running other operating systems, such as the HP Touchpad. Community releases often come pre-rooted and contain modifications unsuitable for non-technical users, such as the ability to overclock or over/undervolted the device's processor. Cyanogen Mod is the most widely used community firmware, and acts as a foundation for numerous others. There have also been attempts with varying degrees of success to port Android to iPhones, notably the I Droid Project.

Historically, device manufacturers and mobile carriers have typically been unsupportive of third-party firmware development. Manufacturers express concern about improper functioning of devices running unofficial software and the support costs resulting from this. Moreover, modified firmware's such as Cyanogen Mod sometimes offer features, such as tethering, for which carriers would otherwise charge a premium. As a result, technical obstacles including locked bootloaders and restricted access to root permissions are common in many devices. However, as community-developed software has grown more popular, and following a statement by the Librarian of Congress in the United States that permits the "jailbreaking" of mobile devices, manufacturers and carriers have softened their position regarding third party development, with some, including HTC, Motorola, Samsung and Sony, providing support and encouraging development. As a result of this, over time the need to circumvent hardware restrictions to install unofficial firmware has lessened as an increasing number of devices are shipped with unlocked or unlockable bootloaders, similar to Nexus series of phones, although usually requiring that users waive their devices'

warranties to do so. However, despite manufacturer acceptance, some carriers in the US still require that phones are locked down, frustrating developers and customers.

1.6 Security and privacy

Android applications run in a sandbox, an isolated area of the system that does not have access to the rest of the system's resources, unless access permissions are explicitly granted by the user when the application is installed. Before installing an application, Play Store displays all required permissions: a game may need to enable vibration or save data to an SD card, for example, but should not need to read SMS messages or access the phonebook. After reviewing these permissions, the user can choose to accept or refuse them, installing the application only if they accept. The sandboxing and permissions system lessens the impact of vulnerabilities and bugs in applications, but developer confusion and limited documentation has resulted in applications routinely requesting unnecessary permissions, reducing its effectiveness. Google has now pushed an update to Android Verify Apps feature, which will now run in background to detect malicious processes and crack them down.

The "App Ops" privacy and application permissions control system, used for internal development and testing by Google, was introduced in Google's Android 4.3 release for the Nexus devices. Initially hidden, the feature was discovered publicly; it allowed users to install a management application and approve or deny permission requests individually for each of the applications installed on a device. Access to the App Ops was later restricted by Google starting with Android 4.4.2 with an explanation that the feature was accidentally enabled and not intended for end-users; for such a decision, Google received criticism from the Electronic Frontier Foundation. Individual application permissions management, through the App Ops or third-party tools, is currently only possible with root access to the device.

Research from Security Company Trend Micro lists premium service abuse as the most common type of Android malware, where text messages are sent from infected phones to premium-rate telephone numbers without the consent or even knowledge of the user. Other malware displays unwanted and intrusive adverts on the device, or sends personal information to unauthorized third parties. Security threats on Android are reportedly growing exponentially; however, Google engineers have argued that the malware and virus threat on Android is being exaggerated by security companies for commercial reasons, and have accused the security industry of playing on fears to sell virus protection software to users. Google maintains that dangerous malware is actually extremely rare, and a survey conducted by F-Secure showed that only 0.5% of Android malware reported had come from the Google Play store.

Google currently uses Google Bouncer malware scanner to watch over and scan the Google Play store apps. It is intended to flag up suspicious apps and warn users of any potential threat with an application before they download it. Android version 4.2 Jelly Bean was released in 2012 with enhanced security features, including a malware scanner built into the system, which works in combination with Google Play but can scan apps installed from third party sources as well, and an alert system which notifies the user when an app tries to send a premium-rate text message, blocking the message unless the user explicitly authorizes it. Several security firms, such as Lookout Mobile Security, AVG Technologies, and McAfee, have released antivirus software for Android devices. This software is ineffective as sandboxing also applies to such applications, limiting their ability to scan the deeper system for threats.

Android smartphones have the ability to report the location of Wi-Fi access points, encountered as phone users move around, to build databases containing the physical locations of hundreds of millions of such access points. These databases form electronic maps to locate smartphones, allowing them to run apps like Foursquare, Google Latitude, Facebook Places, and to deliver location-based ads. Third party monitoring software such as Taint Droid, an academic research-funded

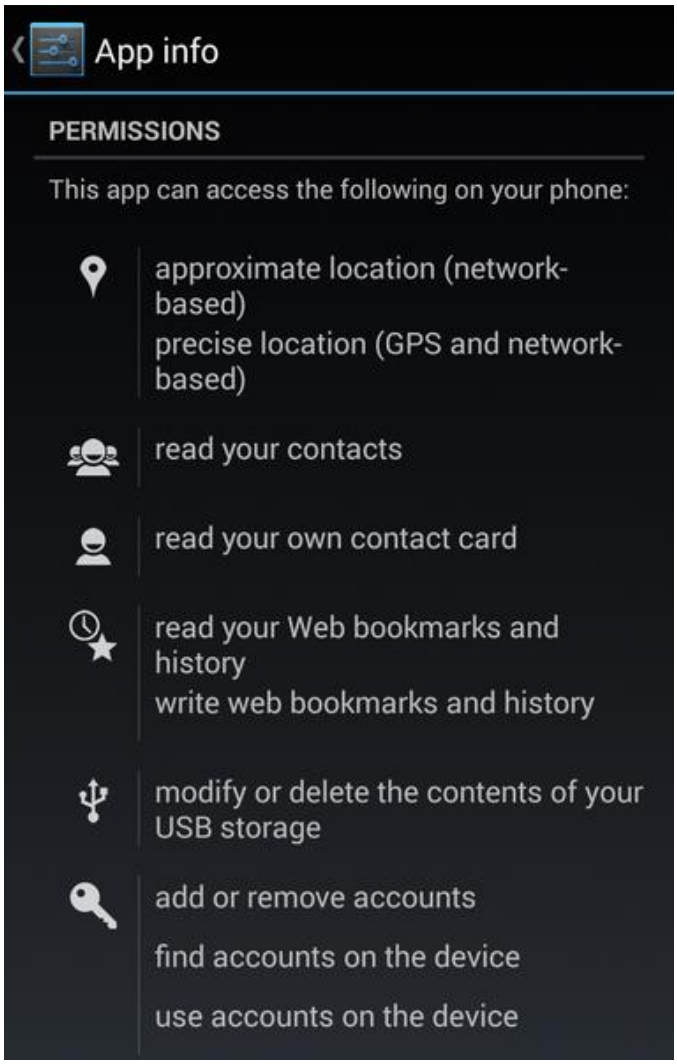
project, can, in some cases, detect when personal information is being sent from applications to remote servers. In August 2013, Google released Android Device Manager (ADM), a component that allows users to remotely track, locate, and wipe their Android device through a web interface. In December 2013, Google released ADM as an Android application on the Google Play store, where it is available to devices running Android version 2.2 and higher.

The open-source nature of Android allows security contractors to take existing devices and adapt them for highly secure uses. For example Samsung has worked with General Dynamics through their Open Kernel Labs acquisition to rebuild Jelly Bean on top of their hardened micro visor for the "Knox" project.

As part of the broader 2013 mass surveillance disclosures it was revealed in September 2013 that the American and British intelligence agencies, the National Security Agency (NSA) and Government Communications Headquarters (GCHQ), respectively, have access to the user data on iPhone, BlackBerry, and Android devices. They are reportedly able to read almost all smartphone information, including SMS, location, emails, and notes. In January 2014, further reports revealed the intelligence agencies' capabilities to intercept the personal information transmitted across the Internet by social networks and other popular applications such as Angry Birds, which collect personal information of their users for advertising and other commercial reasons. GCHQ has, according to The Guardian, a wiki-style guide of different apps and advertising networks, and the different data that can be siphoned from each. Later that week, the Finnish Angry Birds developer Rovio announced that it was reconsidering its relationships with its advertising platforms in the light of these revelations, and called upon the wider industry to do the same.

The documents revealed a further effort by the intelligence agencies to intercept Google Maps searches and queries submitted from Android and other smartphones to collect location information in bulk. The NSA and GCHQ insist their activities are in compliance with all relevant domestic and international laws, although the Guardian stated "the latest disclosures could also add to mounting public concern about how

the technology sector collects and uses information, especially for those outside the US, who enjoy fewer privacy protections than Americans.



Chapter 2

Introduction to socket programming

2.1 socket

Definition: Sockets are the fundamental technology for programming software to communicate on TCP/IP networks. A socket provides a bidirectional communication endpoint for sending and receiving data with another socket. Socket connections normally run between two different computers on a LAN or across the Internet, but they can also be used for interprocess communication on a single computer.

Socket Libraries

Network programmers typically use socket libraries rather than coding directly to lower level socket APIs. The two most commonly use socket libraries are Berkeley Sockets (for Linux/Unix systems) and WinSock (for Windows systems).

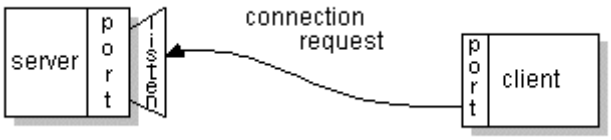
A socket library provides a set of API functions similar to those programmers use for working with files, such as open (), read (), write () and close ().

Sockets and Addresses

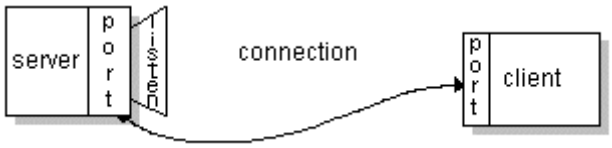
Socket endpoints on TCP/IP networks each have a unique address that is the combination of an IP address and a TCP/IP port number. When creating a new socket, the socket library automatically generates a unique port number on that device, and the programmer can also specify their own port numbers in specific situations.

Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.



If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.

The client and server can now communicate by writing to or reading from their sockets.

An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints. That way you can have multiple connections between your host and the server.

The java.net package in the Java platform provides a class, Socket, that implements one side of a two-way connection between your Java program and another program on the network. The Socket class sits on top of a platform-dependent implementation, hiding the details of any particular system from your Java program. By using the java.net.Socket class instead of relying on native code, your Java programs can communicate over the network in a platform-independent fashion.

Additionally, java.net includes the Server Socket class, which implements a socket that servers can use to listen for and accept connections to clients. This lesson shows you how to use the Socket and Server Socket classes.

If you are trying to connect to the Web, the URL class and related classes (URLConnection, URLEncoder) are probably more appropriate than the socket classes. In fact, URLs are a relatively high-level connection to the Web and use sockets as part of the underlying implementation. See Working with URLs for information about connecting to the Web via URLs.

2.2 Reading from and Writing to a Socket

Let's look at a simple example that illustrates how a program can establish a connection to a server program using the Socket class and then, how the client can send data to and receive data from the server through the socket.

The example program implements a client, EchoClient that connects to an echo server. The echo server receives data from its client and echoes it back. The example EchoServer implements an echo server.

(Alternatively, the client can connect to any host that supports the Echo Protocol).

The EchoClient example creates a socket, thereby getting a connection to the echo server. It reads input from the user on the standard input stream, and then forwards that text to the echo server by writing the text to the socket. The server echoes the input back through the socket to the client. The client program reads and displays the data passed back to it from the server.

Note that the EchoClient example both writes to and reads from its socket, thereby sending data to and receiving data from the echo server.

Let's walk through the program and investigate the interesting parts. The following statements in the try-with-resources statement in the EchoClient example are critical. These lines establish the socket connection between the client and the server and open a PrintWriter and a BufferedReader on the socket:

```
String hostName = args[0];
int portNumber = Integer.parseInt(args[1]);

try (
    Socket echoSocket = new Socket(hostName, portNumber);
    PrintWriter out =
        new PrintWriter(echoSocket.getOutputStream(), true);
    BufferedReader in =
        new BufferedReader(
            new InputStreamReader(echoSocket.getInputStream()));
    BufferedReader stdIn =
        new BufferedReader(
            new InputStreamReader(System.in))
)
```

The first statement in the try-with resources statement creates a new Socket object and names it echoSocket. The Socket constructor used here requires the name of the computer and the port number to which you want to connect. The example program uses the first command-line argument as the name of the computer (the host name) and the second command line argument as the port number. When you run this

program on your computer, make sure that the host name you use is the fully qualified IP name of the computer to which you want to connect. For example, if your echo server is running on the computer echoserver.example.com and it is listening on port number 7, first run the following command from the computer echoserver.example.com if you want to use the EchoServer example as your echo server:

Java EchoServer 7

Afterward, run the EchoClient example with the following command:

Java EchoClient echoserver.example.com 7

The second statement in the try-with resources statement gets the socket's output stream and opens a PrintWriter on it. Similarly, the third statement gets the socket's input stream and opens a BufferedReader on it. The example uses readers and writers so that it can write Unicode characters over the socket.

To send data through the socket to the server, the EchoClient example needs to write to the PrintWriter. To get the server's response, EchoClient reads from the BufferedReader object stdIn, which is created in the fourth statement in the try-with resources statement. If you are not yet familiar with the Java platform's I/O classes, you may wish to read Basic I/O.

The next interesting part of the program is the while loop. The loop reads a line at a time from the standard input stream and immediately sends it to the server by writing it to the PrintWriter connected to the socket:

```
String userInput;
while ((userInput = stdIn.readLine()) != null) {
    out.println(userInput);
    System.out.println("echo: " + in.readLine());
}
```

The last statement in the while loop reads a line of information from the BufferedReader connected to the socket. The readLine method waits until the server echoes the information back to EchoClient. When

readline returns, EchoClient prints the information to the standard output.

The while loop continues until the user types an end-of-input character. That is, the EchoClient example reads input from the user, sends it to the Echo server, gets a response from the server, and displays it, until it reaches the end-of-input. (You can type an end-of-input character by pressing Ctrl-C.) The while loops then terminates, and the Java runtime automatically closes the readers and writers connected to the socket and to the standard input stream, and it closes the socket connection to the server. The Java runtime closes these resources automatically because they were created in the try-with-resources statement. The Java runtime closes these resources in reverse order that they were created. (This is good because streams connected to a socket should be closed before the socket itself is closed).

This client program is straightforward and simple because the echo server implements a simple protocol. The client sends text to the server, and the server echoes it back. When your client programs are talking to a more complicated server such as an HTTP server, your client program will also be more complicated. However, the basics are much the same as they are in this program:

.Open a socket

.Open an input stream and output stream to the socket

.Read from and write to the stream according to the server's protocol

.Close the streams

.Close the socket

Only step 3 differs from client to client, depending on the server. The other steps remain largely the same

2.3 Writing the Server Side of a Socket

This section shows you how to write a server and the client that goes with it. The server in the client/server pair serves up Knock Knock jokes. Knock Knock jokes are favored by children and are usually vehicles for bad puns. They go like this:

Server: "Knock knock!"

Client: "Who's there?"

Server: "Dexter."

Client: "Dexter who?"

Server: "Dexter halls with boughs of holly."

Client: "Groan."

The example consists of two independently running Java programs: the client program and the server program. The client program is implemented by a single class, KnockKnockClient, and is very similar to the EchoClient example from the previous section. The server program is implemented by two classes: KnockKnockServer and KnockKnockProtocol. KnockKnockServer, which is similar to EchoServer, contains the main method for the server program and performs the work of listening to the port, establishing connections, and reading from and writing to the socket. The class KnockKnockProtocol serves up the jokes. It keeps track of the current joke, the current state (sent knock knock, sent clue, and so on), and returns the various text pieces of the joke depending on the current state. This object implements the protocol—the language that the client and server have agreed to use to communicate.

The following section looks in detail at each class in both the client and the server and then shows you how to run them.

The Knock Knock Server

This section walks through the code that implements the Knock Knock server program, KnockKnockServer.

The server program begins by creating a new Server Socket object to listen on a specific port (see the statement in bold in the following code segment). When running this server, choose a port that is not already dedicated to some other service. For example, this command starts the server program KnockKnockServer so that it listens on port 4444:

Java KnockKnockServer 4444

The server program creates the Server Socket object in a try-with-resources statement:

```
int portNumber = Integer.parseInt(args[0]);

try (
    ServerSocket serverSocket = new ServerSocket(portNumber);
    Socket clientSocket = serverSocket.accept();
    PrintWriter out =
        new PrintWriter(clientSocket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
        new InputStreamReader(clientSocket.getInputStream()));
) {
```

Server Socket is a java.net class that provides a system-independent implementation of the server side of a client/server socket connection. The constructor for Server Socket throws an exception if it can't listen on the specified port (for example, the port is already being used). In this case, the KnockKnockServer has no choice but to exit.

If the server successfully binds to its port, then the Server Socket object is successfully created and the server continues to the next step—accepting a connection from a client (the next statement in the try-with-resources statement):

ClientSocket = serverSocket.accept ();

The accept method waits until a client starts up and requests a connection on the host and port of this server. (Let's assume that you ran the server program KnockKnockServer on the computer named knockknockserver.example.com.) In this example, the server is running on the port number specified by the first command-line argument. When a connection is requested and successfully established, the accept method returns a new Socket object which is bound to the same local port and has its remote address and remote port set to that of the client. The server can communicate with the client over this new Socket and continue to listen for client connection requests on the original Server Socket This particular version of the program doesn't listen for more client connection requests. However, a modified version of the program is provided in Supporting Multiple Clients.

After the server successfully establishes a connection with a client, it communicates with the client using this code:

```
try (
    // ...
    PrintWriter out =
        new PrintWriter(clientSocket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
        new InputStreamReader(clientSocket.getInputStream()));
) {
    String inputLine, outputLine;

    // Initiate conversation with client
    KnockKnockProtocol kkp = new KnockKnockProtocol();
    outputLine = kkp.processInput(null);
    out.println(outputLine);

    while ((inputLine = in.readLine()) != null) {
        outputLine = kkp.processInput(inputLine);
        out.println(outputLine);
        if (outputLine.equals("Bye."))
            break;
    }
```

This code does the following:

Gets the socket's input and output stream and opens readers and writers on them.

Initiates communication with the client by writing to the socket (shown in bold).

Communicates with the client by reading from and writing to the socket the while loop.

Step 1 is already familiar. Step 2 is shown in bold and is worth a few comments. The bold statements in the code segment above initiate the conversation with the client. The code creates a KnockKnockProtocol object—the object that keeps tracks of the current joke, the current state within the joke, and so on.

After the KnockKnockProtocol is created, the code calls KnockKnockProtocol's process Input method to get the first message that the server sends to the client. For this example, the first thing that the server says is "Knock! Knock!" Next, the server writes the information to the PrintWriter connected to the client socket, thereby sending the message to the client.

Step 3 is encoded in the while loop. As long as the client and server still have something to say to each other, the server reads from and writes to the socket, sending messages back and forth between the client and the server.

The server initiated the conversation with a "Knock! Knock!" so afterwards the server must wait for the client to say "Who's there?" As a result, the while loop iterates on a read from the input stream. The readLine method waits until the client responds by writing something to its output stream (the server's input stream). When the client responds, the server passes the client's response to the KnockKnockProtocol object and asks the KnockKnockProtocol object for a suitable reply. The server immediately sends the reply to the client via the output stream

connected to the socket, using a call to println. If the server's response generated from the KnockKnockServer object is "Bye." this indicates that the client doesn't want any more jokes and the loop quits.

The Java runtime automatically closes the input and output streams, the client socket, and the server socket because they have been created in the try-with-resources statement.

The Knock Knock Protocol

The KnockKnockProtocol class implements the protocol that the client and server use to communicate. This class keeps track of where the client and the server are in their conversation and serves up the server's response to the client's statements. The KnockKnockProtocol object contains the text of all the jokes and makes sure that the client gives the proper response to the server's statements. It wouldn't do to have the client say "Dexter who?" when the server says "Knock! Knock!"

All client/server pairs must have some protocol by which they speak to each other; otherwise, the data that passes back and forth would be meaningless. The protocol that your own clients and servers use depends entirely on the communication required by them to accomplish the task.

The Knock Knock Client

The KnockKnockClient class implements the client program that speaks to the KnockKnockServer. KnockKnockClient is based on the EchoClient program in the previous section, Reading from and Writing to a Socket and should be somewhat familiar to you. But we'll go over the program anyway and look at what's happening in the client in the context of what's going on in the server.

When you start the client program, the server should already be running and listening to the port, waiting for a client to request a connection. So, the first thing the client program does is to open a socket that is connected to the server running on the specified host name and port:

```
String hostName = args[0];
int portNumber = Integer.parseInt(args[1]);

try (
    Socket kkSocket = new Socket(hostName, portNumber);
    PrintWriter out = new PrintWriter(kkSocket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
        new InputStreamReader(kkSocket.getInputStream()));
)
```

When creating its socket, the KnockKnockClient example uses the host name of the first command-line argument, the name of the computer on your network that is running the server program KnockKnockServer.

The KnockKnockClient example uses the second command-line argument as the port number when creating its socket. This is a remote port number—the number of a port on the server computer—and is the port to which KnockKnockServer is listening. For example, the following command runs the KnockKnockClient example with knockknockserver.example.com as the name of the computer that is running the server program KnockKnockServer and 4444 as the remote port number:

Java KnockKnockClient knockknockserver.example.com 4444

The client's socket is bound to any available local port—a port on the client computer. Remember that the server gets a new socket as well. If you run the KnockKnockClient example with the command-line arguments in the previous example, then this socket is bound to local port number 4444 on the computer from which you ran the KnockKnockClient example. The server's socket and the client's socket are connected.

Next comes the while loop that implements the communication between the client and the server. The server speaks first, so the client must listen first. The client does this by reading from the input stream attached to the socket. If the server does speak, it says "Bye." and the client exits the loop. Otherwise, the client displays the text to the standard output and then reads the response from the user, who types into the standard input. After the user types a carriage return, the client sends the text to the server through the output stream attached to the socket.

```
while ((fromServer = in.readLine()) != null) {
    System.out.println("Server: " + fromServer);
    if (fromServer.equals("Bye."))
        break;

    fromUser = stdIn.readLine();
    if (fromUser != null) {
        System.out.println("Client: " + fromUser);
        out.println(fromUser);
    }
}
```

The communication ends when the server asks if the client wishes to hear another joke, the client says no, and the server says "Bye".

The client automatically closes its input and output streams and the socket because they were created in the try-with-resources statement.

Running the Programs

You must start the server program first. To do this, run the server program using the Java interpreter, just as you would any other Java application. Specify as a command-line argument the port number on which the server program listens:

Java KnockKnockServer 4444

Next, run the client program. Note that you can run the client on any computer on your network; it does not have to run on the same computer as the server. Specify as command-line arguments the host

name and the port number of the computer running the KnockKnockServer server program:

Java KnockKnockClient knockknockserver.example.com 4444

If you are too quick, you might start the client before the server has a chance to initialize itself and begin listening on the port. If this happens, you will see a stack trace from the client. If this happens, just restart the client.

If you try to start a second client while the first client is connected to the server, the second client just hangs. The next section, Supporting Multiple Clients, talks about supporting multiple clients.

When you successfully get a connection between the client and server, you will see the following text displayed on your screen:

Server: Knock! Knock

Now, you must respond with:

Who's there?

The client echoes what you type and sends the text to the server. The server responds with the first line of one of the many Knock Knock jokes in its repertoire. Now your screen should contain this (the text you typed is in bold):

Server: Knock! Knock

Who's there?

Client: Who's there?

Server: Turnip

Now, you respond with:

Turnip who?

Again, the client echoes what you type and sends the text to the server. The server responds with the punch line. Now your screen should contain this:

Server: Knock! Knock!

Who's there?

Client: Who's there?

Server: Turnip

Turnip who?

Client: Turnip who?

(Server: Turnip the heat, its cold in here! Want another? (Y/n)

If you want to hear another joke, type y; if not, type n. If you type y, the server begins again with "Knock! Knock!" If you type n, the server says "Bye." thus causing both the client and the server to exit.

If at any point you make a typing mistake, the KnockKnockServer object catches it and the server responds with a message similar to this:

Server: You're supposed to say "Who's there!"

The server then starts the joke over again:

Server: Try again. Knock! Knock!

Note that the KnockKnockProtocol object is particular about spelling and punctuation but not about capitalization.

Supporting Multiple Clients

To keep the KnockKnockServer example simple, we designed it to listen for and handle a single connection request. However, multiple client requests can come into the same port and, consequently, into the same Server Socket. Client connection requests are queued at the port, so the server must accept the connections sequentially. However, the server can service them simultaneously through the use of threads—one thread per each client connection.

The basic flow of logic in such a server is this:

```
while (true) {
    accept a connection;
    create a thread to deal with the client;
}
```

The thread reads from and writes to the client connection as necessary.

Chapter 3

Introduction to Threads in java

3.1 Defining and Starting a Thread

An application that creates an instance of Thread must provide the code that will run in that thread. There are two ways to do this:

Provide a Runnable object. The Runnable interface defines a single method, run, meant to contain the code executed in the thread. The Runnable object is passed to the Thread constructor, as in the HelloRunnable example:

```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}
```

Subclass Thread. The Thread class itself implements Runnable, though its run method does nothing. An application can subclass Thread, providing its own implementation of run, as in the HelloThread example:

```
public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new HelloThread()).start();
    }

}
```

Notice that both examples invoke Thread. Start in order to start the new thread.

Which of these idioms should you use? The first idiom, which employs a Runnable object, is more general, because the Runnable object cans subclass a class other than Thread. The second idiom is easier to use in simple applications, but is limited by the fact that your task class must be a descendant of Thread. This lesson focuses on the first approach, which separates the Runnable task from the Thread object that executes the task. Not only is this approach more flexible, but it is applicable to the high-level thread management APIs covered later.

The Thread class defines a number of methods useful for thread management. These include static methods, which provide information about, or affect the status of, the thread invoking the method. The other methods are invoked from other threads involved in managing the thread and Thread object. We'll examine some of these methods in the following sections.

3.2 Pausing Execution with Sleep

Thread. Sleep causes the current thread to suspend execution for a specified period. This is an efficient means of making processor time available to the other threads of an application or other applications that might be running on a computer system. The sleep method can also be used for pacing, as shown in the example that follows, and waiting for another thread with duties that are understood to have time requirements, as with the Simple Threads example in a later section.

Two overloaded versions of sleep are provided: one that specifies the sleep time to the millisecond and one that specifies the sleep time to the nanosecond. However, these sleep times are not guaranteed to be precise, because they are limited by the facilities provided by the underlying OS. Also, the sleep period can be terminated by interrupts, as we'll see in a later section. In any case, you cannot assume that invoking sleep will suspend the thread for precisely the time period specified.

The Sleep Messages example uses sleep to print messages at four-second intervals:

```
public class SleepMessages {
    public static void main(String args[])
        throws InterruptedException {
        String importantInfo[] = {
            "Mares eat oats",
            "Does eat oats",
            "Little lambs eat ivy",
            "A kid will eat ivy too"
        };

        for (int i = 0;
             i < importantInfo.length;
             i++) {
            //Pause for 4 seconds
            Thread.sleep(4000);
            //Print a message
            System.out.println(importantInfo[i]);
        }
    }
}
```

Notice that main declares that it throws Interrupted Exception. This is an exception that sleep throws when another thread interrupts the current thread while sleep is active. Since this application has not defined another thread to cause the interrupt, it doesn't bother to catch Interrupted Exception.

3.3 Interrupts

An interrupt is an indication to a thread that it should stop what it is doing and do something else. It's up to the programmer to decide exactly how a thread responds to an interrupt, but it is very common for the thread to terminate. This is the usage emphasized in this lesson.

A thread sends an interrupt by invoking interrupt on the Thread object for the thread to be interrupted. For the interrupt mechanism to work correctly, the interrupted thread must support its own interruption.

Supporting Interruption

How does a thread support its own interruption? This depends on what it's currently doing. If the thread is frequently invoking methods that throw Interrupted Exception, it simply returns from the run method after it catches that exception. For example, suppose the central message loop in the SleepMessages example were in the run method of a thread's Runnable object. Then it might be modified as follows to support interrupts:

```
for (int i = 0; i < importantInfo.length; i++) {
    // Pause for 4 seconds
    try {
        Thread.sleep(4000);
    } catch (InterruptedException e) {
        // We've been interrupted: no more messages.
        return;
    }
    // Print a message
    System.out.println(importantInfo[i]);
}
```

Many methods that throw Interrupted Exception, such as sleep, are designed to cancel their current operation and return immediately when an interrupt is received.

What if a thread goes a long time without invoking a method that throws Interrupted Exception? Then it must periodically invoke Thread. Interrupted, which returns true if an interrupt has been received. For example:

```
for (int i = 0; i < inputs.length; i++) {
    heavyCrunch(inputs[i]);
    if (Thread.interrupted()) {
        // We've been interrupted: no more crunching.
        return;
    }
}
```

In this simple example, the code simply tests for the interrupt and exits the thread if one has been received. In more complex applications, it might make more sense to throw an Interrupted Exception:

```
if (Thread.interrupted()) {
    throw new InterruptedException();
}
```

This allows interrupt handling code to be centralized in a catch clause.

The Interrupt Status Flag

The interrupt mechanism is implemented using an internal flag known as the interrupt status. Invoking Thread. Interrupt sets this flag. When a thread checks for an interrupt by invoking the static method Thread. Interrupted, interrupt status is cleared. The non-static is Interrupted method, which is used by one thread to query the interrupt status of another, does not change the interrupt status flag.

By convention, any method that exits by throwing an Interrupted Exception clears interrupt status when it does so. However, it's always possible that interrupt status will immediately be set again, by another thread invoking interrupt.

3.4 Joins

The join method allows one thread to wait for the completion of another. If t is a Thread object whose thread is currently executing,

t.join();

Causes the current thread to pause execution until t's thread terminates. Overloads of join allow the programmer to specify a waiting period. However, as with sleep, join is dependent on the OS for timing, so you should not assume that join will wait exactly as long as you specify.

Like sleep, join responds to an interrupt by exiting with an Interrupted Exception.

3.5 The Simple Threads Example

The following example brings together some of the concepts of this section. Simple Threads consists of two threads. The first is the main thread that every Java application has. The main thread creates a new thread from the Runnable object, MessageLoop, and waits for it to finish. If the MessageLoop thread takes too long to finish, the main thread interrupts it.

The MessageLoop thread prints out a series of messages. If interrupted before it has printed all its messages, the MessageLoop thread prints a message and exits.

```
public class SimpleThreads {

    // Display a message, preceded by
    // the name of the current thread
    static void threadMessage(String message) {
        String threadName =
            Thread.currentThread().getName();
        System.out.format("%s: %s%n",
                          threadName,
                          message);
    }

    private static class MessageLoop
        implements Runnable {
        public void run() {
            String importantInfo[] = {
                "Mares eat oats",
                "Does eat oats",
                "Little lambs eat ivy",
                "A kid will eat ivy too"
            };
            try {
                for (int i = 0;
                     i < importantInfo.length;
                     i++) {
                    // Pause for 4 seconds
                    Thread.sleep(4000);
                    // Print a message
                    threadMessage(importantInfo[i]);
                }
            } catch (InterruptedException e) {
                threadMessage("I wasn't done!");
            }
        }
    }
```

```
public static void main(String args[])
    throws InterruptedException {

    // Delay, in milliseconds before
    // we interrupt MessageLoop
    // thread (default one hour).
    long patience = 1000 * 60 * 60;

    // If command line argument
    // present, gives patience
    // in seconds.
    if (args.length > 0) {
        try {
            patience = Long.parseLong(args[0]) * 1000;
        } catch (NumberFormatException e) {
            System.err.println("Argument must be an integer.");
            System.exit(1);
        }
    }

    threadMessage("Starting MessageLoop thread");
    long startTime = System.currentTimeMillis();
    Thread t = new Thread(new MessageLoop());
    t.start();

    threadMessage("Waiting for MessageLoop thread to finish");
    // loop until MessageLoop
    // thread exits
    while (t.isAlive()) {
        threadMessage("Still waiting...");
        // Wait maximum of 1 second
        // for MessageLoop thread
        // to finish.
        t.join(1000);
        if (((System.currentTimeMillis() - startTime) > patience)
              && t.isAlive()) {
            threadMessage("Tired of waiting!");
                  t.interrupt();
                  // Shouldn't be long now
                  // -- wait indefinitely
                  t.join();
              }
           }
           threadMessage("Finally!");
       }
    }
```

3.6 Synchronization

Threads communicate primarily by sharing access to fields and the objects reference fields refer to. This form of communication is extremely efficient, but makes two kinds of errors possible: thread interference and memory consistency errors. The tool needed to prevent these errors is synchronization.

However, synchronization can introduce thread contention, which occurs when two or more threads try to access the same resource simultaneously and cause the Java runtime to execute one or more threads more slowly, or even suspend their execution. Starvation and livelock are forms of thread contention. See the section Liveness for more information.

This section covers the following topics:

* Thread Interference describes how errors are introduced when multiple threads access shared data.

*Memory Consistency Errors describes errors that result from inconsistent views of shared memory.

*Synchronized Methods describes a simple idiom that can effectively prevent thread interference and memory consistency errors.

*Implicit Locks and Synchronization describes a more general synchronization idiom, and describes how synchronization is based on implicit locks.

*Atomic Access talks about the general idea of operations that can't be interfered with by other threads.

3.7 Thread Interference

Consider a simple class called Counter

```
class Counter {
    private int c = 0;

    public void increment() {
        c++;
    }

    public void decrement() {
        c--;
    }

    public int value() {
        return c;
    }

}
```

Counter is designed so that each invocation of increment will add 1 to c, and each invocation of decrement will subtract 1 from c. However, if a Counter object is referenced from multiple threads, interference between threads may prevent this from happening as expected.

Interference happens when two operations, running in different threads, but acting on the same data, interleave. This means that the two operations consist of multiple steps, and the sequences of steps overlap.

It might not seem possible for operations on instances of Counter to interleave, since both operations on c are single, simple statements. However, even simple statements can translate to multiple steps by the virtual machine. We won't examine the specific steps the virtual machine takes — it is enough to know that the single expression c++ can be decomposed into three steps:

1. Retrieve the current value of c.

2.Increment the retrieved value by 1.

3.Store the incremented value back in c.

The expression c-- can be decomposed the same way, except that the second step decrements instead of increments.

Suppose Thread A invokes increment at about the same time Thread B invokes decrement. If the initial value of c is 0, their interleaved actions might follow this sequence:

1.Thread A: Retrieve c

2.Thread B: Retrieve c

3.Thread A: Increment retrieved value; result is 1

4.Thread B: Decrement retrieved value; result is -1

5.Thread A: Store result in c; c is now 1

6.Thread B: Store result in c; c is now -1

Thread A's result is lost, overwritten by Thread B. This particular interleaving is only one possibility. Under different circumstances it might be Thread B's result that gets lost, or there could be no error at all. Because they are unpredictable, thread interference bugs can be difficult to detect and fix.

3.8 Memory Consistency Errors

Memory consistency errors occur when different threads have inconsistent views of what should be the same data. The causes of memory consistency errors are complex and beyond the scope of this tutorial. Fortunately, the programmer does not need a detailed understanding of these causes. All that is needed is a strategy for avoiding them.

The key to avoiding memory consistency errors is understanding the happens-before relationship. This relationship is simply a guarantee that memory writes by one specific statement are visible to another specific

statement. To see this, consider the following example. Suppose a simple int field is defined and initialized:

int counter = 0;

The counter field is shared between two threads, A and B. Suppose thread A increments counter:

Counter++;

Then, shortly afterwards, thread B prints out counter:

System.out.println (counter);

If the two statements had been executed in the same thread, it would be safe to assume that the value printed out would be "1". But if the two statements are executed in separate threads, the value printed out might well be "0", because there's no guarantee that thread A's change to counter will be visible to thread B — unless the programmer has established a happens-before relationship between these two statements.

There are several actions that create happens-before relationships. One of them is synchronization, as we will see in the following section.

We've already seen two actions that create happens-before relationships.

When a statement invokes Thread. Start, every statement that has a happens-before relationship with that statement also has a happens-before relationship with every statement executed by the new thread. The effects of the code that led up to the creation of the new thread are visible to the new thread.

When a thread terminates and causes a Thread. Join in another thread to return, then all the statements executed by the terminated thread have a happens-before relationship with all the statements following the successful join. The effects of the code in the thread are now visible to the thread that performed the join.

3.9 Deadlock

Deadlock describes a situation where two or more threads are blocked forever, waiting for each other. Here's an example.

Alphonse and Gaston are friends, and great believers in courtesy. A strict rule of courtesy is that when you bow to a friend, you must remain bowed until your friend has a chance to return the bow. Unfortunately, this rule does not account for the possibility that two friends might bow to each other at the same time. This example application, Deadlock, models this possibility:

```
public class Deadlock {
    static class Friend {
        private final String name;
        public Friend(String name) {
            this.name = name;
        }
        public String getName() {
            return this.name;
        }
        public synchronized void bow(Friend bower) {
            System.out.format("%s: %s"
                + "  has bowed to me!%n",
                this.name, bower.getName());
            bower.bowBack(this);
        }
        public synchronized void bowBack(Friend bower) {
            System.out.format("%s: %s"
                + " has bowed back to me!%n",
                this.name, bower.getName());
        }
    }

    public static void main(String[] args) {
        final Friend alphonse =
            new Friend("Alphonse");
        final Friend gaston =
            new Friend("Gaston");
        new Thread(new Runnable() {
            public void run() { alphonse.bow(gaston); }
        }).start();
        new Thread(new Runnable() {
            public void run() { gaston.bow(alphonse); }
        }).start();
    }
}
```

3.10 Starvation and Livelock

Starvation and livelock are much less common a problem than deadlock, but are still problems that every designer of concurrent software is likely to encounter.

Starvation

Starvation describes a situation where a thread is unable to gain regular access to shared resources and is unable to make progress. This happens when shared resources are made unavailable for long periods by "greedy" threads. For example, suppose an object provides a synchronized method that often takes a long time to return. If one thread invokes this method frequently, other threads that also need frequent synchronized access to the same object will often be blocked.

Livelock

A thread often acts in response to the action of another thread. If the other thread's action is also a response to the action of another thread, then livelock may result. As with deadlock, livelocked threads are unable to make further progress. However, the threads are not blocked — they are simply too busy responding to each other to resume work. This is comparable to two people attempting to pass each other in a corridor: Alphonse moves to his left to let Gaston pass, while Gaston moves to his right to let Alphonse pass. Seeing that they are still blocking each other, Alphone moves to his right, while Gaston moves to his left. They're still blocking each other, so..

Chapter 4

Tools used

FOR CLIENT

4.1 Eclipse

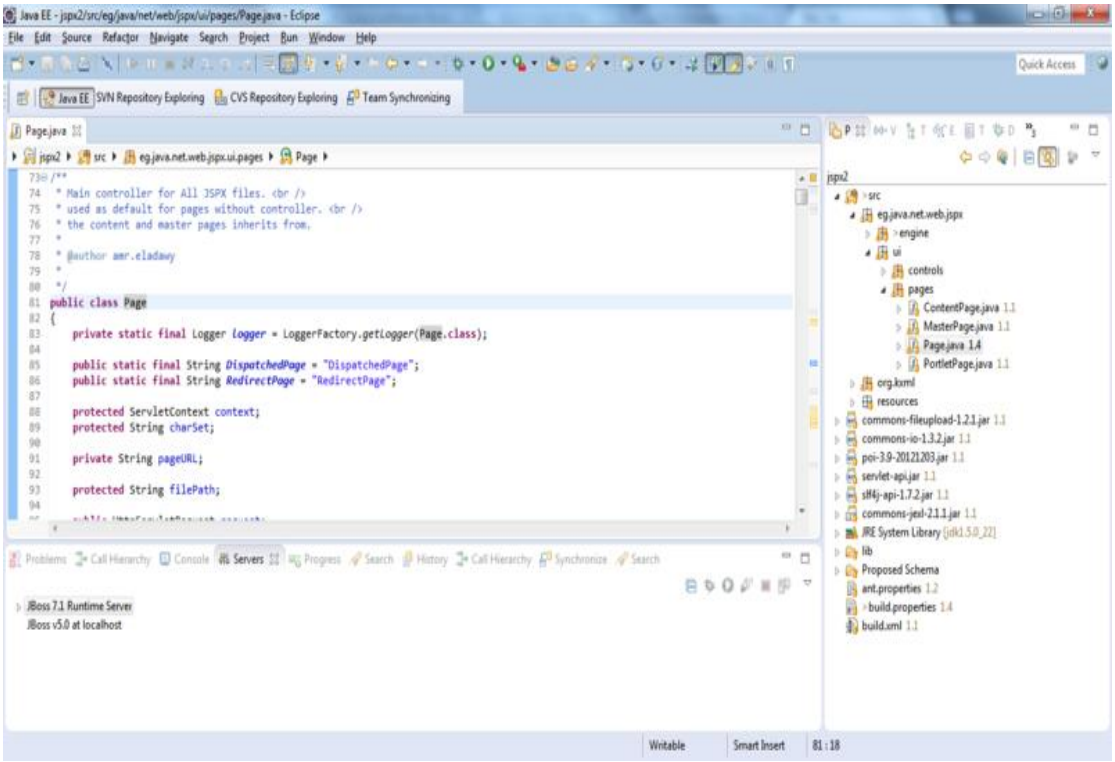
4.1.1 Eclipse (software)

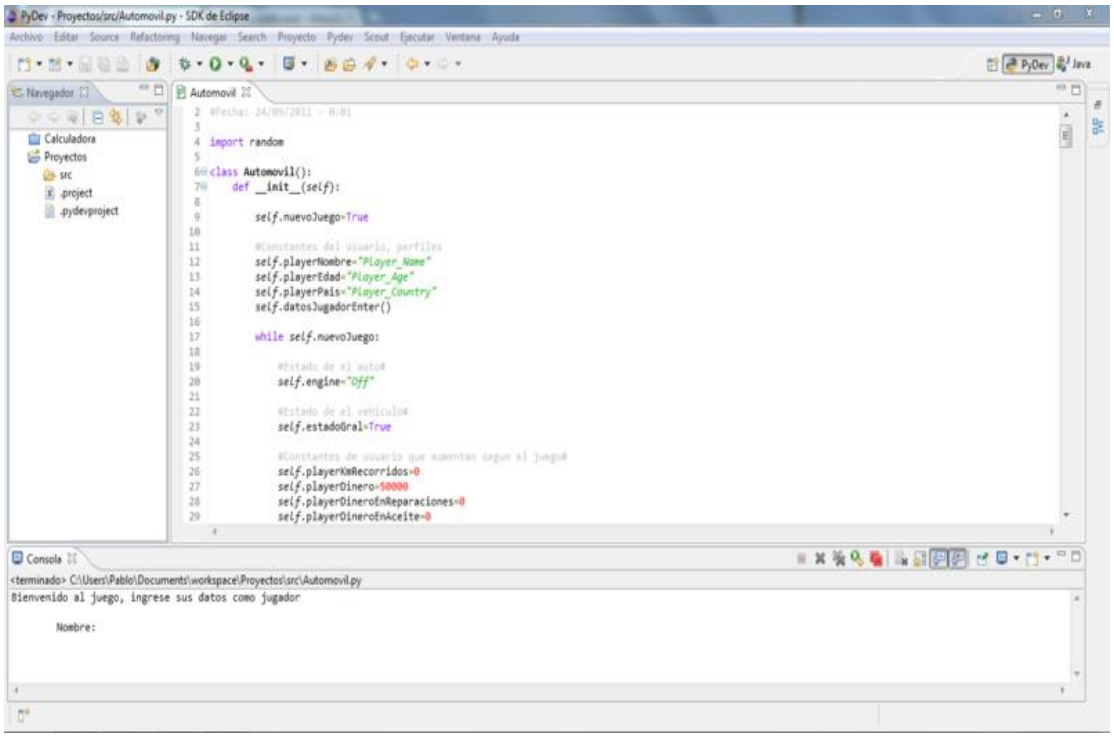
In computer programming, Eclipse is an integrated development environment (IDE). It contains a base workspace and an extensible plug-in system for customizing the environment. Written mostly in Java,

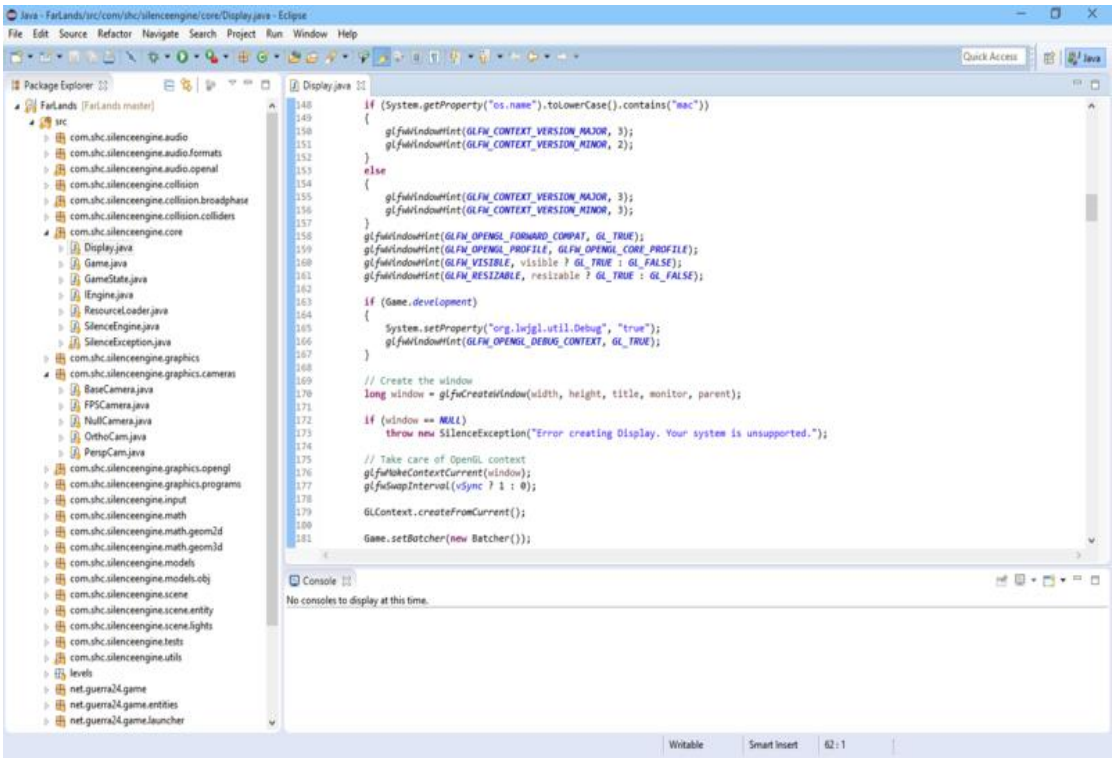
Eclipse can be used to develop applications. By means of various plug-ins, Eclipse may also be used to develop applications in other programming languages: Ada, ABAP, C, C++, COBOL, Fortran, Haskell, JavaScript, Lasso, Lua, Natural, Perl, PHP, Prolog, Python, R, Ruby (including Ruby on Rails framework), Scala, Clojure, Groovy, Scheme, and Erlang. It can also be used to develop packages for the software Mathematica. Development environments include the Eclipse Java development tools (JDT) for Java and Scala, Eclipse CDT for C/C++ and Eclipse PDT for PHP, among others.

The initial codebase originated from IBM VisualAge. The Eclipse software development kit (SDK), which includes the Java development tools, is meant for Java developers. Users can extend its abilities by installing plug-ins written for the Eclipse Platform, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules.

Released under the terms of the Eclipse Public License, Eclipse SDK is free and open source software (although it is incompatible with the GNU General Public License). It was one of the first IDEs to run under GNU Classpath and it runs without problems under IcedTea.




PyDev - Proyectos/src/Automovil.py - SDK de Eclipse
Archivo Editar Source Refactoring Navegar Search Proyecto Pydev Scout Ejecutar Ventana Ayuda
Navegador
Calculadora
Proyectos
src
.project
.pydevproject
Automovil
#Fecha: 24/09/2011 - 0:01
import random
class Automovil():
def __init__(self):
self.nuevoJuego=True
#Constantes del usuario, perfiles
self.playerNombre="Player_Name"
self.playerEdad="Player_Age"
self.playerPais="Player_Country"
self.datosJugadorEnter()
while self.nuevoJuego:
#Estado de el auto#
self.engine="Off"
#Estado de el vehiculo#
self.estadoGral=True
#Constantes de usuario que aumentan segun el juego#
self.playerKmRecorridos=0
self.playerDinero=50000
self.playerDineroEnReparaciones=0
self.playerDineroEnAceite=0
Consola
<terminado> C:\Users\Pablo\Documents\workspace\Proyectos\src\Automovil.py
Bienvenido al juego, ingrese sus datos como jugador
Nombre:


Java - FarLands/src/com/shc/silenceengine/core/Display.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
FarLands [FarLands master]
src
com.shc.silenceengine.audio
com.shc.silenceengine.audio.formats
com.shc.silenceengine.audio.openal
com.shc.silenceengine.collision
com.shc.silenceengine.collision.broadphase
com.shc.silenceengine.collision.colliders
com.shc.silenceengine.core
Display.java
Game.java
GameState.java
IEngine.java
ResourceLoader.java
SilenceEngine.java
SilenceException.java
com.shc.silenceengine.graphics
com.shc.silenceengine.graphics.cameras
BaseCamera.java
FPSCamera.java
NullCamera.java
OrthoCam.java
PerspCam.java
com.shc.silenceengine.graphics.opengl
com.shc.silenceengine.graphics.programs
com.shc.silenceengine.input
com.shc.silenceengine.math
com.shc.silenceengine.math.geom2d
com.shc.silenceengine.math.geom3d
com.shc.silenceengine.models
com.shc.silenceengine.models.obj
com.shc.silenceengine.scene
com.shc.silenceengine.scene.entity
com.shc.silenceengine.scene.lights
com.shc.silenceengine.tests
com.shc.silenceengine.utils
levels
net.guerra24.game
net.guerra24.game.entities
net.guerra24.game.launcher
Display.java
if (System.getProperty("os.name").toLowerCase().contains("mac"))
{
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 2);
}
else
{
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
}
glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
glfwWindowHint(GLFW_VISIBLE, visible ? GL_TRUE : GL_FALSE);
glfwWindowHint(GLFW_RESIZABLE, resizable ? GL_TRUE : GL_FALSE);
if (Game.development)
{
System.setProperty("org.lwjgl.util.Debug", "true");
glfwWindowHint(GLFW_OPENGL_DEBUG_CONTEXT, GL_TRUE);
}
// Create the window
long window = glfwCreateWindow(width, height, title, monitor, parent);
if (window == NULL)
throw new SilenceException("Error creating Display. Your system is unsupported.");
// Take care of OpenGL context
glfwMakeContextCurrent(window);
glfwSwapInterval(vSync ? 1 : 0);
GLContext.createFromCurrent();
Game.setBatcher(new Batcher());
Console
No consoles to display at this time.
Writable
Smart Insert
62 : 1

4.1.2 History

Eclipse was inspired by the Smalltalk-based VisualAge family of integrated development environment (IDE) products,. Although fairly successful, a major drawback of the VisualAge products was that developed code was not in a component model; instead, all code for a project was held in a compressed lump (somewhat like a zip file but in a proprietary format called .dat); individual classes could not be easily accessed, certainly not outside the tool. A team primarily at the IBM Cary NC lab developed the new product as a Java-based replacement. In November 2001, a consortium was formed with a board of stewards to further the development of Eclipse as open-source software. It is estimated that IBM had already invested close to $40 million by that time. The original members were Borland, IBM, Merant, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft and WebGain. The number of stewards increased to over 80 by the end of 2003. In January 2004, the Eclipse Foundation was created.

Eclipse 3.0 (released on 21 June 2004) selected the OSGi Service Platform specifications as the runtime architecture.

The Association for Computing Machinery recognized Eclipse with the 2011 ACM Software Systems Award on 26 April 2012.

Licensing

The Eclipse Public License (EPL) is the fundamental license under which Eclipse projects are released. Some projects require dual licensing, for which the Eclipse Distribution License (EDL) is available, although use of this license must be applied for and is considered on a case-by-case basis.

Eclipse was originally released under the Common Public License, but was later re-licensed under the Eclipse Public License. The Free Software Foundation has said that both licenses are free software licenses, but are incompatible with the GNU General Public License (GPL). Mike Milinkovich, of the Eclipse Foundation commented that moving to the GPL would be considered when version 3 of the GPL was released.

Name

According to Lee Nackman, Chief Technology Officer of IBM's Rational division (originating in 2003) at that time, the name "Eclipse" (dating from at least 2001) was not a wordplay on Sun Microsystems, as the product's primary competition at the time of naming was Microsoft Visual Studio (which it, Eclipse, was to eclipse).[13] Different versions of Eclipse have been named after different celestial bodies, more specifically planets or planets' natural satellites. Examples are: Europa, Ganymede, Callisto, Galileo and Luna. The latest version coming in 2015 has been named Mars.

Releases

Since 2006 the Foundation has coordinated an annual Simultaneous Release. Each release includes the Eclipse Platform as well as a number of other Eclipse projects.

As of 2008 each Simultaneous Release has occurred on the 4th Wednesday of June.

Version Name	Date	Platform Version	Projects	Main Changes
N/A	21 June 2004	3.0[15]		
N/A	28 June 2005	3.1		
Callisto	30 June 2006	3.2	Callisto projects[16]	
Europa	29 June 2007	3.3	Europa projects[17]	
Ganymede	25 June 2008	3.4	Ganymede projects[18]	
Galileo	24 June 2009	3.5	Galileo projects[19]	
Helios	23 June 2010	3.6	Helios projects[20]	
Indigo	22 June 2011	3.7	Indigo projects[21]	
Juno	27 June 2012	3.8 and 4.2[22] [Notes 1]	Juno projects[25]	
Kepler	26 June 2013	4.3	Kepler projects[26]	
Luna	25 June 2014	**4.4**	Luna projects[27]	Integrated Java 8 support (in the previous version this was possible via a "Java 8 patch" plugin)
Mars	24 June 2015 (planned)	4.5	Mars projects[28]	
Neon	June 2016 (planned)	4.6	Neon projects[29]	

Old version Older version, still supported **Latest version** Future release

4.1.3 Architecture

Eclipse uses plug-ins to provide all the functionality within and on top of the runtime system. Its runtime system is based on Equinox, an implementation of the OSGi core framework specification.

In addition to allowing the Eclipse Platform to be extended using other programming languages, such as C and Python, the plug-in framework allows the Eclipse Platform to work with typesetting languages like LaTeX and networking applications such as telnet and database management systems. The plug-in architecture supports writing any desired extension to the environment, such as for configuration management. Java and CVS support is provided in the Eclipse SDK, with support for other version control systems provided by third-party plug-ins.

With the exception of a small run-time kernel, everything in Eclipse is a plug-in. This means that every plug-in developed integrates with Eclipse in exactly the same way as other plug-ins; in this respect, all features are "created equal". Eclipse provides plug-ins for a wide variety of features, some of which are through third parties using both free and commercial models. Examples of plug-ins include for UML, for Sequence and other UML diagrams, a plug-in for DB Explorer, and many others.

The Eclipse SDK includes the Eclipse Java development tools (JDT), offering an IDE with a built-in incremental Java compiler and a full model of the Java source files. This allows for advanced refactoring techniques and code analysis. The IDE also makes use of a workspace, in this case a set of metadata over a flat file space allowing external file modifications as long as the corresponding workspace "resource" is refreshed afterwards.

Eclipse implements the graphical control elements of the Java toolkit called SWT, whereas most Java applications use the Java standard Abstract Window Toolkit (AWT) or Swing. Eclipse's user interface also uses an intermediate graphical user interface layer called JFace, which simplifies the construction of applications based on SWT. Eclipse was made to run on Wayland during a GSoC-Project in 2014.

Language packs being developed by the "Babel project" provide translations into over a dozen natural languages.

4.1.4 Rich Client Platform

Eclipse provides the Rich Client Platform (RCP) for developing general purpose applications. The following components constitute the rich client platform:

.Equinox OSGi – a standard bundling framework

.Core platform – boot Eclipse, run plug-ins

.Standard Widget Toolkit (SWT) – a portable widget toolkit

.JFace – viewer classes to bring model view controller programming to .SWT, file buffers, text handling, text editors

.Eclipse Workbench – views, editors, perspectives, wizards

Examples of rich client applications based on Eclipse are:

IBM Notes 8 and 9

Novell/NetIQ Designer for Identity Manager

Apache Directory Studio

Remote Component Environment

4.1.5 Server platform

Eclipse supports development for Tomcat, GlassFish and many other servers and is often capable of installing the required server (for development) directly from the IDE. It supports remote debugging, allowing the user to watch variables and step through the code of an application that is running on the attached server.

4.1.6 Web Tools Platform

The Eclipse Web Tools Platform (WTP) project is an extension of the Eclipse platform with tools for developing Web and Java EE applications. It includes source and graphical editors for a variety of languages, wizards and built-in applications to simplify development, and tools and APIs to support deploying, running, and testing apps.

4.1.7 Modeling platform

The Modeling project contains all the official projects of the Eclipse Foundation focusing on model-based development technologies. They are all compatible with the Eclipse Modeling Framework created by IBM. Those projects are separated in several categories: Model Transformation, Model Development Tools, Concrete Syntax Development, Abstract Syntax Development, Technology and Research, and Amalgam.

Model Transformation

Model Transformation projects uses EMF based models as an input and produces either a model or text as an output. Model to model transformation projects includes ATL, an open source transformation language and toolkit used to transform a given model or to generate a new model from a given EMF model. Model to text transformation projects contains Acceleo, an implementation of MOFM2T, a standard model to text language from the OMG. Acceleo is an open source code generator that can generate any textual language (Java, PHP, Python, etc.) from EMF based models defined with any metamodel (UML, SysML, etc.).

Model Development Tools

Model Development Tools projects are implementations of modeling standard used in the industry like UML or OCL and their toolkit. Among those projects can be found implementations of the following standards:

*UML * SysML * OCL * BPMN * IMM * SBVR * XSD *NEDA

Concrete Syntax Development

The Concrete Syntax Development project contains the Graphical Modeling Framework, an Eclipse based framework dedicated to the graphical representation of EMF based models.

Abstract Syntax Development

The Abstract Syntax Development project hosts the Eclipse Modeling Framework, core of most of the modeling project of the Eclipse Foundation and the framework available for EMF like CDO, EMF query or EMF validation.

Technology and research

Technology and Research projects are prototypes of modeling project; this project is used to host all the modeling projects of the Eclipse Foundation during their incubation phase.

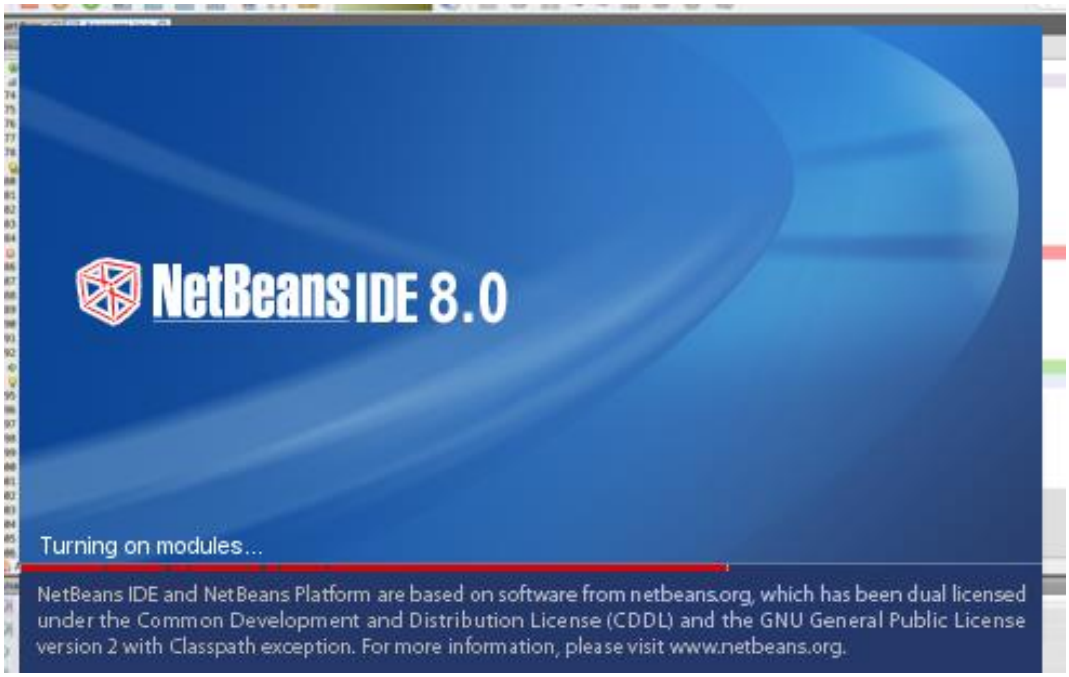
Amalgam

Amalgam provides the packaging and integration between all the available modeling tools for the Eclipse package dedicated to modeling tools.

FOR SERVER

4.2 NetBeans

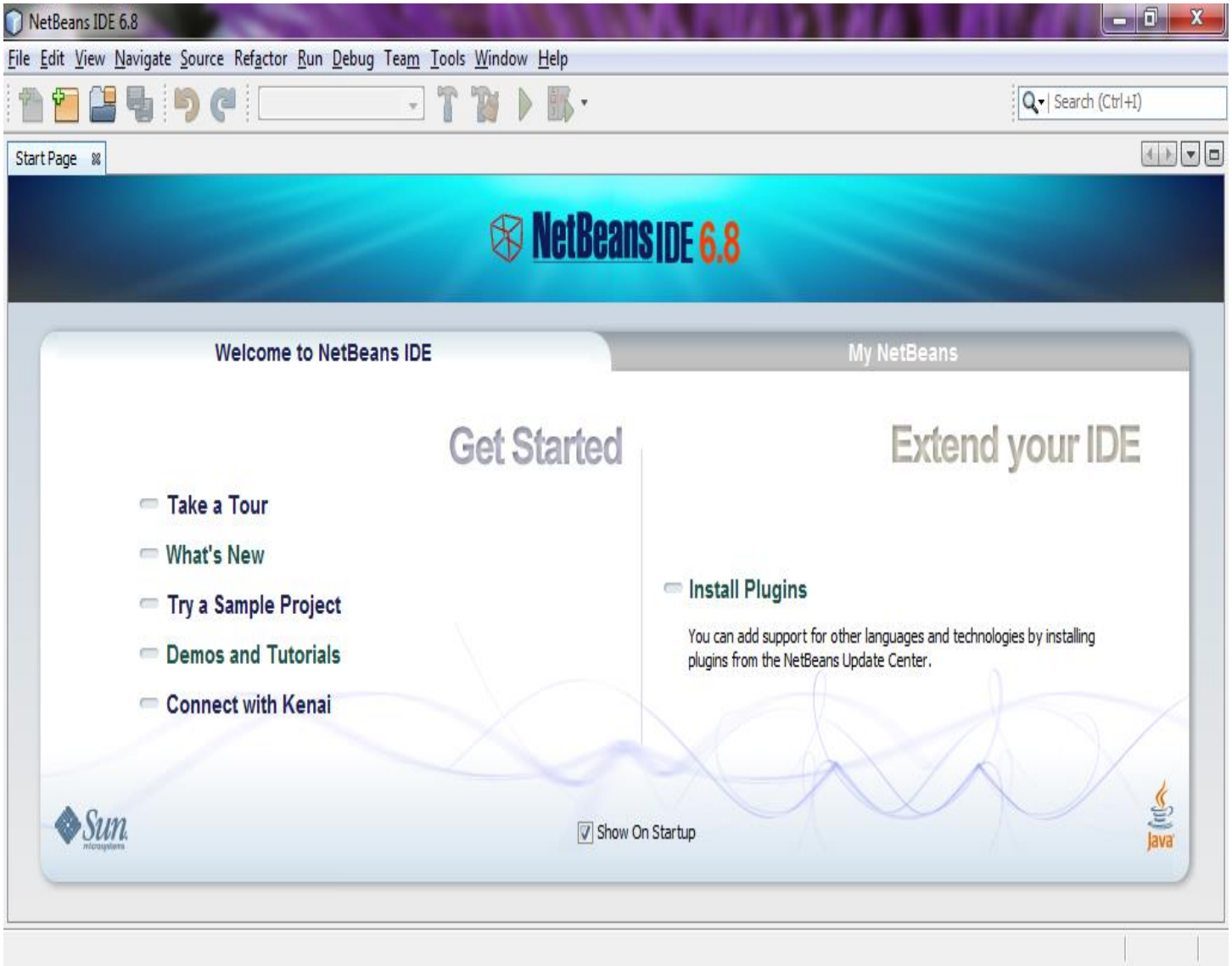
4.2.1 NetBeans (software)

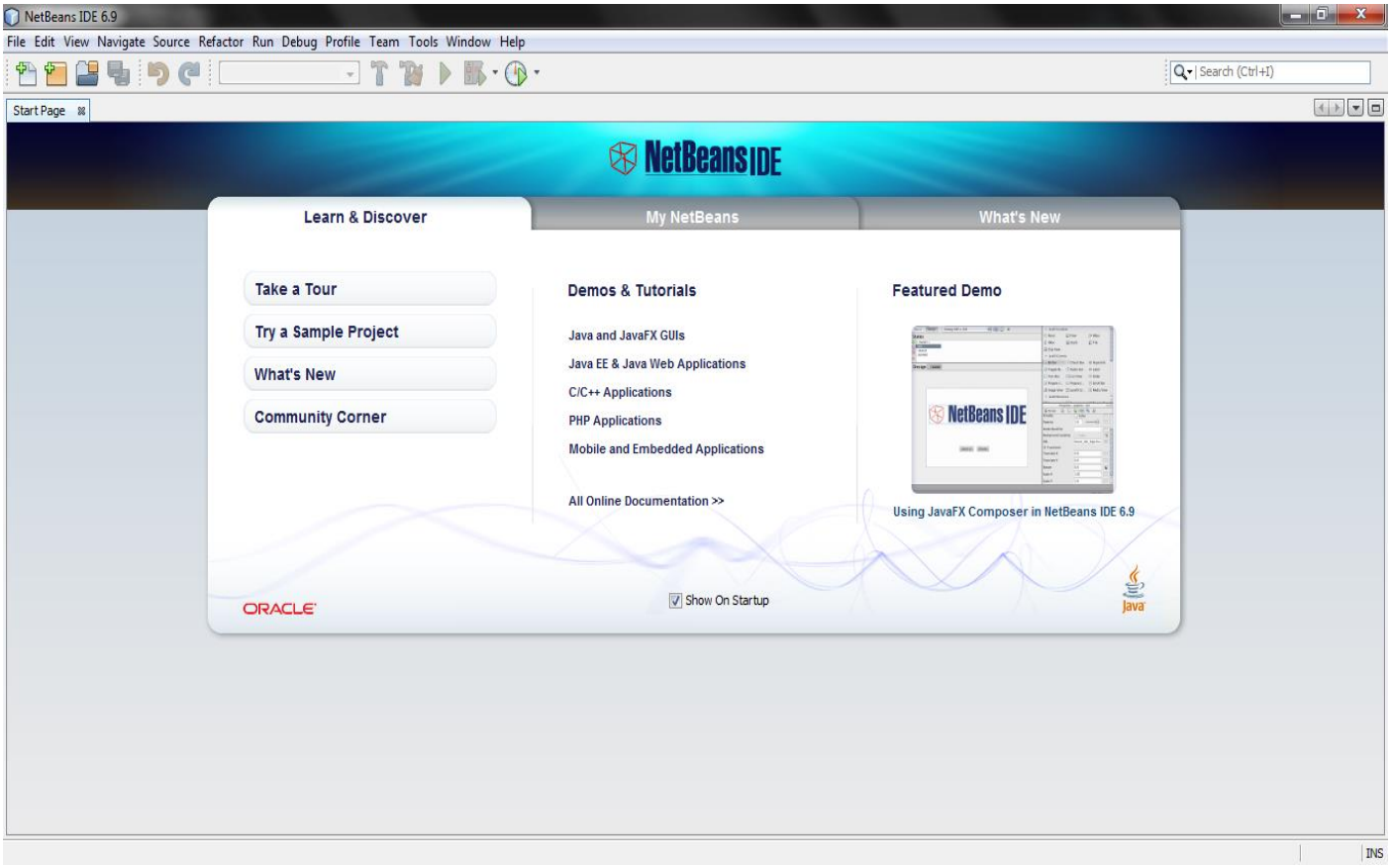


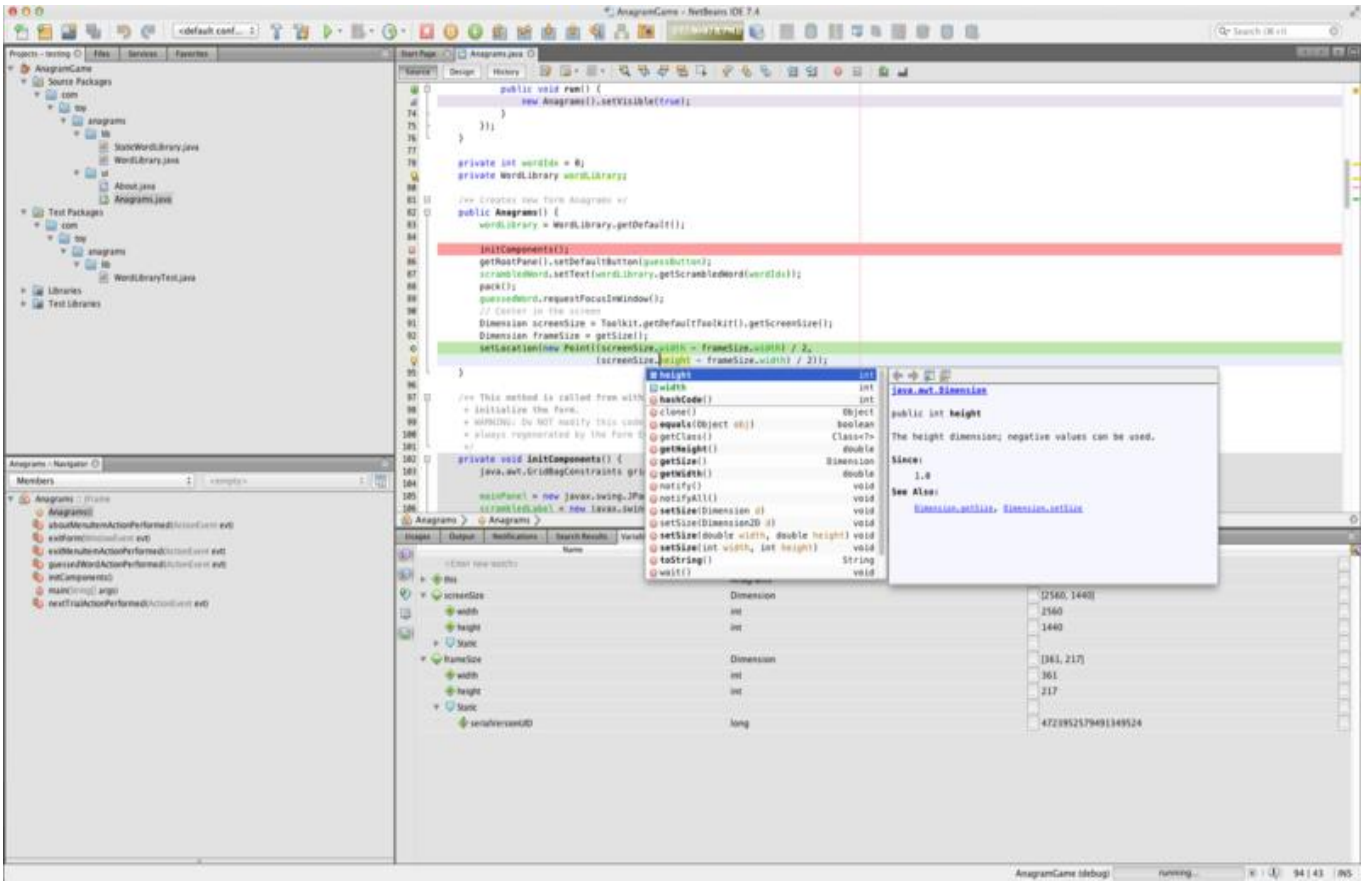
NetBeans is a software development platform written in Java. The NetBeans Platform allows applications to be developed from a set of modular software components called modules. Applications based on the NetBeans Platform, including the NetBeans integrated development The [environment (IDE), can be extended by third party developers. NetBeans IDE is primarily intended for development in Java, but also supports other languages, in particular PHP, C/C++ and HTML5.

NetBeans is cross-platform and runs on Microsoft Windows, Mac OS X, Linux, Solaris and other platforms supporting a compatible JVM.

The NetBeans Team actively support the product and seek feature suggestions from the wider community. Every release is preceded by a time for Community testing and feedback.




NetBeans IDE 6.9
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Start Page
NetBeans IDE
Learn & Discover
My NetBeans
What's New
Take a Tour
Try a Sample Project
What's New
Community Corner
Demos & Tutorials
Java and JavaFX GUIs
Java EE & Java Web Applications
C/C++ Applications
PHP Applications
Mobile and Embedded Applications
All Online Documentation >>
Featured Demo
Using JavaFX Composer in NetBeans IDE 6.9
ORACLE
Show On Startup
Java
INS



4.2.2 History

NetBeans began in 1996 as Xelfi (word play on Delphi),[7][8] a Java IDE student project under the guidance of the Faculty of Mathematics and Physics at Charles University in Prague. In 1997 Roman Staněk formed a company around the project and produced commercial versions of the NetBeans IDE until it was bought by Sun Microsystems in 1999. Sun open-sourced the NetBeans IDE in June of the following year. Since then, the NetBeans community has continued to grow.[9] In 2010, Sun (and thus NetBeans) was acquired by Oracle.

Current versions

NetBeans IDE 6.0 introduced support for developing IDE modules and rich client applications based on the NetBeans platform, a Java Swing GUI builder (formerly known as "Project Matisse"), improved CVS support, WebLogic 9 and JBoss 4 support, and many editor enhancements. NetBeans 6 is available in official repositories of major Linux distributions.

NetBeans IDE 6.5, released in November 2008, extended the existing Java EE features (including Java Persistence support, EJB 3 and JAX-WS). Additionally, the NetBeans Enterprise Pack supports development of Java EE 5 enterprise applications, including SOA visual design tools, XML schema tools, web services orchestration (for BPEL), and UML modeling. The NetBeans IDE Bundle for C/C++ supports C/C++ and FORTRAN development.

NetBeans IDE 6.8 is the first IDE to provide complete support of Java EE 6 and the GlassFish Enterprise Server v3. Developers hosting their open-source projects on kenai.com additionally benefit from instant messaging and issue tracking integration and navigation right in the IDE, support for web application development with PHP 5.3 and the Symfony framework, and improved code completion, layouting, hints and navigation in JavaFX projects.

NetBeans IDE 6.9, released in June 2010, added support for OSGi, Spring Framework 3.0, Java EE dependency injection (JSR-299), Zend Framework for PHP, and easier code navigation (such as "Is Overridden/Implemented" annotations), formatting, hints, and refactoring across several languages.

NetBeans IDE 7.0 was released in April 2011. On August 1, 2011, the NetBeans Team released NetBeans IDE 7.0.1, which has full support for the official release of the Java SE 7 platform.

NetBeans IDE 7.3 was released in February 2013 which added support for HTML5 and web technologies.

.NetBeans IDE 7.4 was released on October 15, 2013

.NetBeans IDE 8.0 was released on March 18, 2014

.NetBeans has a roadmap document for release plans

4.2.3 NetBeans Platform

Framework for simplifying the development of Java Swing desktop applications. The NetBeans IDE bundle for Java SE contains what is needed to start developing NetBeans plugins and NetBeans Platform based applications; no additional SDK is required.

Applications can install modules dynamically. Any application can include the Update Center module to allow users of the application to download digitally signed upgrades and new features directly into the running application. Reinstalling an upgrade or a new release does not force users to download the entire application again.

The platform offers reusable services common to desktop applications, allowing developers to focus on the logic specific to their application. Among the features of the platform are:

User interface management (e.g. menus and toolbars)

User settings management

Storage management (saving and loading any kind of data)

Window management

Wizard framework (supports step-by-step dialogs)

NetBeans Visual Library

Integrated development tools

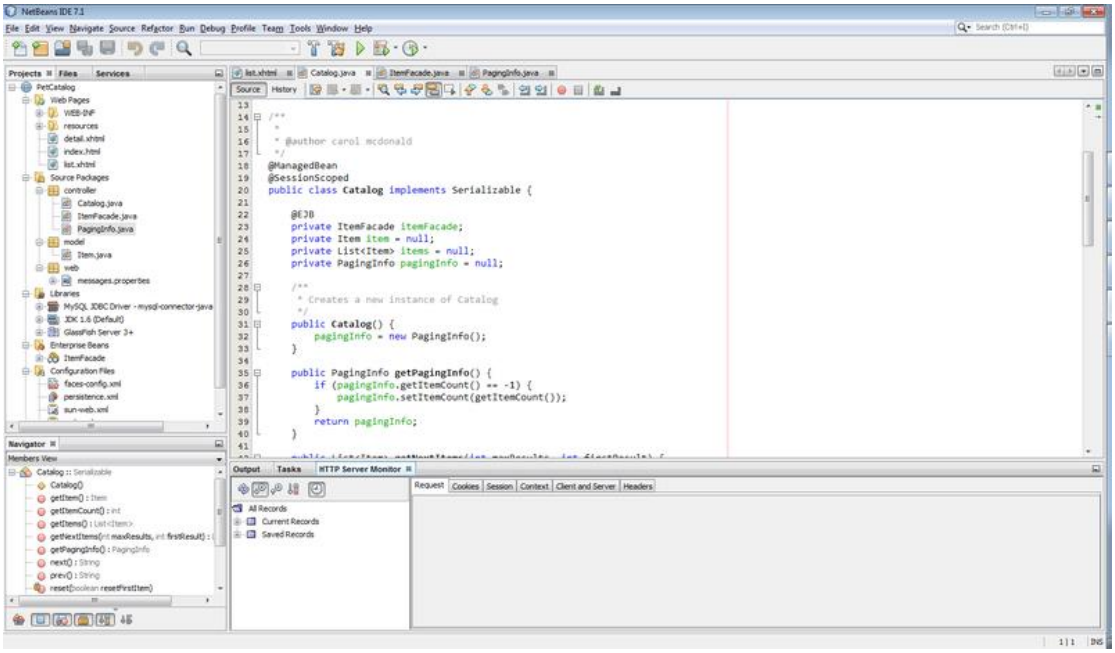
NetBeans IDE is a free, open-source, cross-platform IDE with built-in-support for Java Programming Language.

4.2.4 NetBeans IDE

NetBeans IDE is an open-source integrated development environment. NetBeans IDE supports development of all Java application types (Java SE (including JavaFX), Java ME, web, EJB and mobile applications) out of the box. Among other features are an Ant-based project system, Maven support, refactoring's, version control (supporting CVS, Subversion, Git, Mercurial and Clearcase).

Modularity: All the functions of the IDE are provided by modules. Each module provides a well-defined function, such as support for the Java language, editing, or support for the CVS versioning system, and SVN. NetBeans contains all the modules needed for Java development in a single download, allowing the user to start working immediately. Modules also allow NetBeans to be extended. New features, such as support for other programming languages, can be added by installing additional modules. For instance, Sun Studio, Sun Java Studio Enterprise, and Sun Java Studio Creator from Sun Microsystems are all based on the NetBeans IDE.

License: From July 2006 through 2007, NetBeans IDE was licensed under Sun's Common Development and Distribution License (CDDL), a license based on the Mozilla Public License (MPL). In October 2007, Sun announced that NetBeans would henceforth be offered under a dual license of the CDDL and the GPL version 2 licenses, with the GPL linking exception for GNU Classpath.



4.2.5 Integrated modules

NetBeans Profiler

The NetBeans Profiler is a tool for the monitoring of Java applications:

It helps developers find memory leaks and optimize speed. Formerly downloaded separately, it is integrated into the core IDE since version 6.0.

The Profiler is based on a Sun Laboratories research project that was named JFluid. That research uncovered specific techniques that can be used to lower the overhead of profiling a Java application. One of those techniques is dynamic bytecode instrumentation, which is particularly useful for profiling large Java applications. Using dynamic bytecode instrumentation and additional algorithms, the NetBeans Profiler is able

to obtain runtime information on applications that are too large or complex for other profilers. NetBeans also support Profiling Points that let you profile precise points of execution and measure execution time.

GUI design tool

Formerly known as project Matisse, the GUI design-tool enables developers to prototype and design Swing GUIs by dragging and positioning GUI components.

The GUI builder has built-in support for JSR 295 (Beans Binding technology), but the support for JSR 296 (Swing Application Framework) was removed in 7.1.

NetBeans JavaScript editor

The NetBeans JavaScript editor provides extended support for JavaScript, Ajax, and CSS.

JavaScript editor features comprise syntax highlighting, refactoring, code completion for native objects and functions, generation of JavaScript class skeletons, generation of Ajax callbacks from a template; and automatic browser compatibility checks.

CSS editor features comprise code completion for styles names, quick navigation through the navigator panel, displaying the CSS rule declaration in a List View and file structure in a Tree View, sorting the outline view by name, type or declaration order (List & Tree), creating rule declarations (Tree only), refactoring a part of a rule name (Tree only).

The NetBeans 7.4 and later uses the new JavaScript engine developed by Oracle.

NetBeans IDE Download Bundles

Users can choose to download NetBeans IDE bundles tailored to specific development needs. Users can also download and install all other features at a later date directly through the NetBeans IDE.

NetBeans IDE Bundle for Web and Java EE

The NetBeans IDE Bundle for Web & Java EE provides complete tools for all the latest Java EE 6 standards, including the new Java EE 6 Web Profile, Enterprise Java Beans (EJBs), servlets, Java Persistence API, web services, and annotations. NetBeans also supports the JSF 2.0 (Facelets), JavaServer Pages (JSP), Hibernate, Spring, and Struts frameworks, and the Java EE 5 and J2EE 1.4 platforms. It includes GlassFish and Apache Tomcat. Some of its features with javaEE includes

*Improved support for CDI, REST services and Java Persistence

*New support for Bean Validation

*Support for JSF component libraries, including bundled Prime Faces library

*Improved editing for Expression Language in JSF, including code completion, refactoring and hints

NetBeans IDE Bundle for Java ME

The NetBeans IDE Bundle for Java ME is a tool for developing applications that run on mobile devices; generally mobile phones, but this also includes entry-level PDAs, and Java Card, among others.

The NetBeans IDE comes bundled with the latest Java ME SDK 3.0 which supports both CLDC and CDC development. One can easily integrate third-party emulators for a robust testing environment. You can download other Java platforms, including the Java Card Platform 3.0, and register them in the IDE.

NetBeans IDE Bundle for PH

NetBeans supports PHP since version 6.5. The bundle for PHP includes:

*syntax highlighting, code completion, occurrence highlighting, error highlighting, CVS version control

*semantic analysis with highlighting of parameters and unused local variables

*PHP code debugging with xdebug

*PHP Unit testing with PHPUnit and Selenium

*Code coverage

*Symfony framework support (since version 6.8)

*Zend Framework support (since version 6.9)

*Yii Framework support (since version 7.3)

*PHP 5.3 namespace and closure support (since version 6.8)

*Code Folding for Control Structures (since version 7.2 dev)

NetBeans IDE Complete Bundle

Oracle also releases a version of NetBeans that includes all of the features of the above bundles. This bundle includes:

*NetBeans Base IDE

*Java SE, JavaFX

*Web and Java EE

*Java ME

*++C/C

*PHP (Version 6.5 and later)

*Groovy

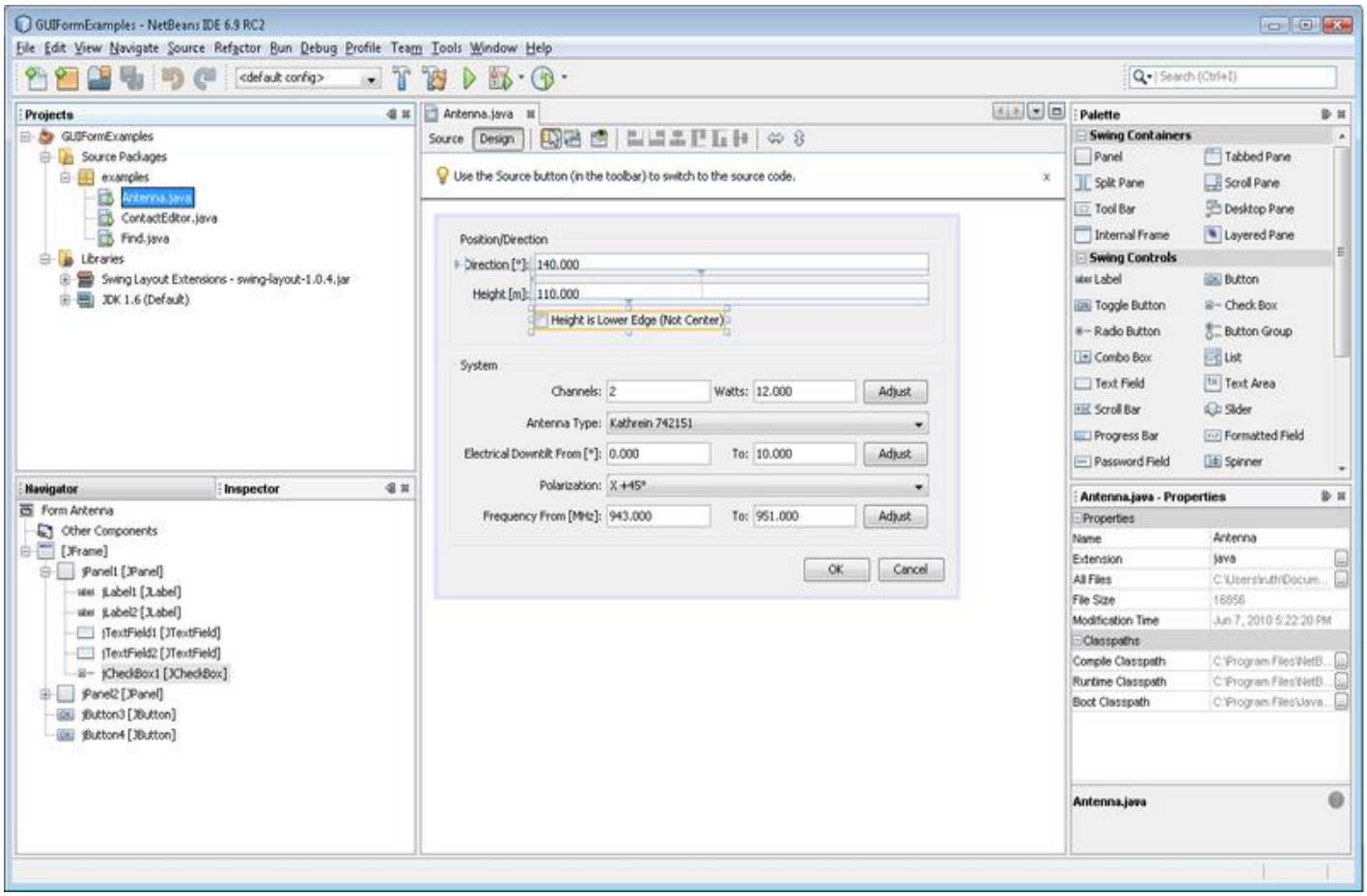
*GlassFish

*Apache Tomcat

*Official Ruby support was removed with the release of 7.0

Other NetBeans IDE Bundles

Apart from the above-mentioned, a NetBeans IDE Bundle is also available for Python. NetBeans for Python is available as an "early access" downloads in an IDE bundle form and as a plugin for any NetBeans 6.5 IDE bundle as well. The Python bundle has been discontinued as of NetBeans 7.0.



Chapter 5

How this project runs

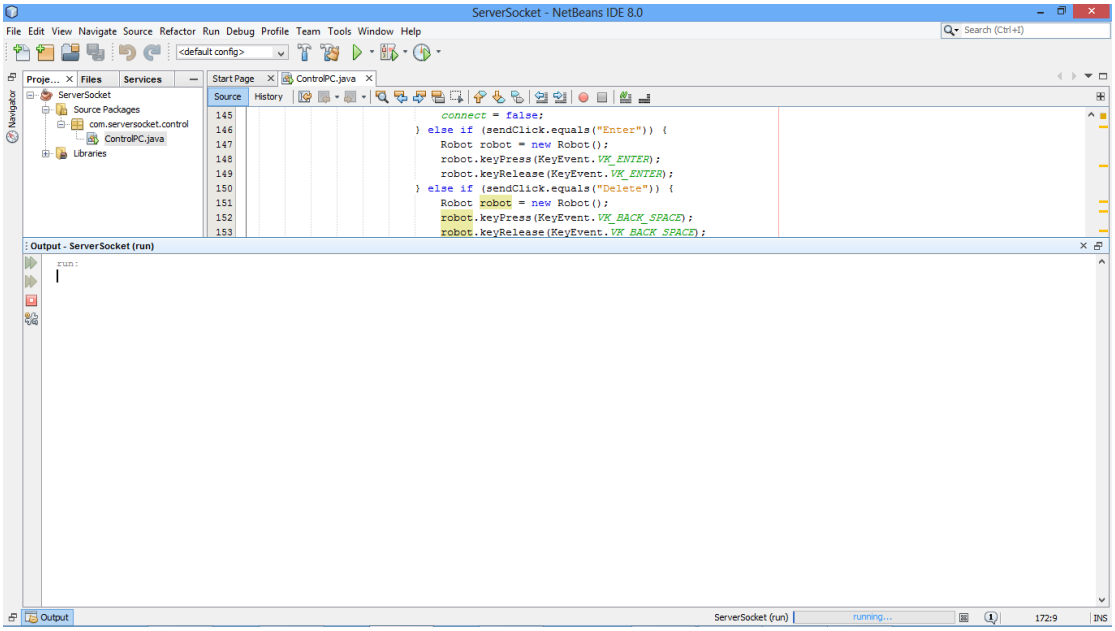
5.1 introductions

How it works

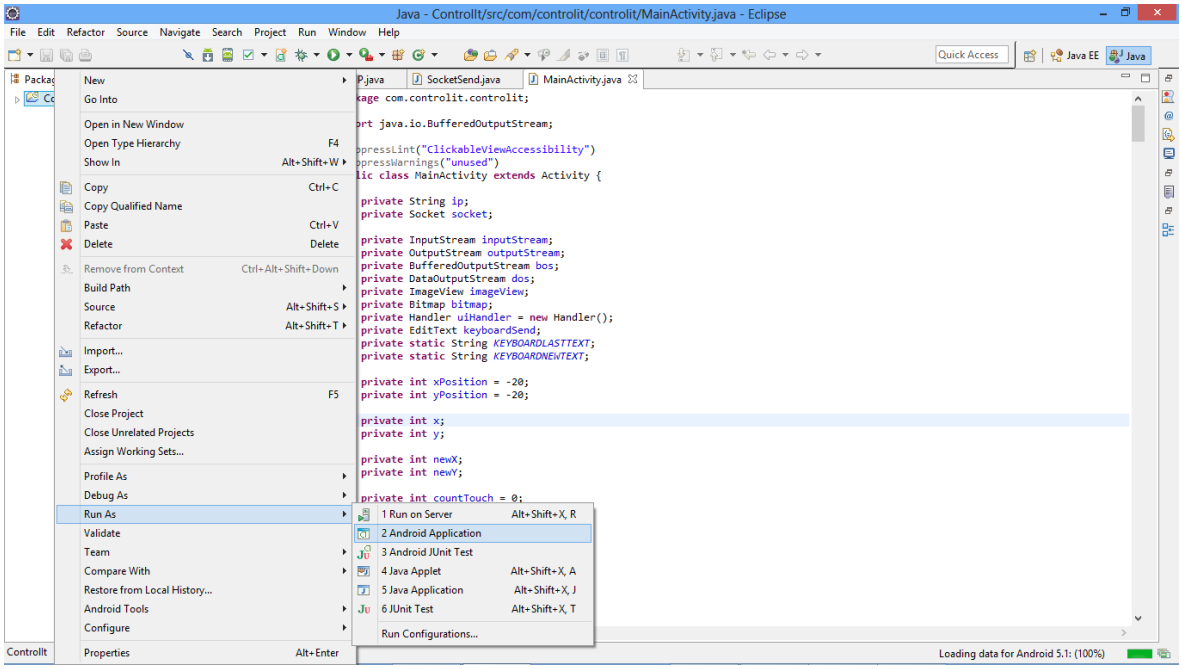
Remote Desktop Connection is a technology that allows you to sit at a computer (sometimes called the client computer) and connect to a remote computer (sometimes called the host computer) in a different location. For example, you can connect to your work computer from your home computer and have access to all of your programs, files, and network resources as though you were in front of your computer at work. You can leave programs running at work and then, when you get home, you can see your work computer's desktop displayed on your home computer, with the same programs running.

5.2 First step

NetBeans for server

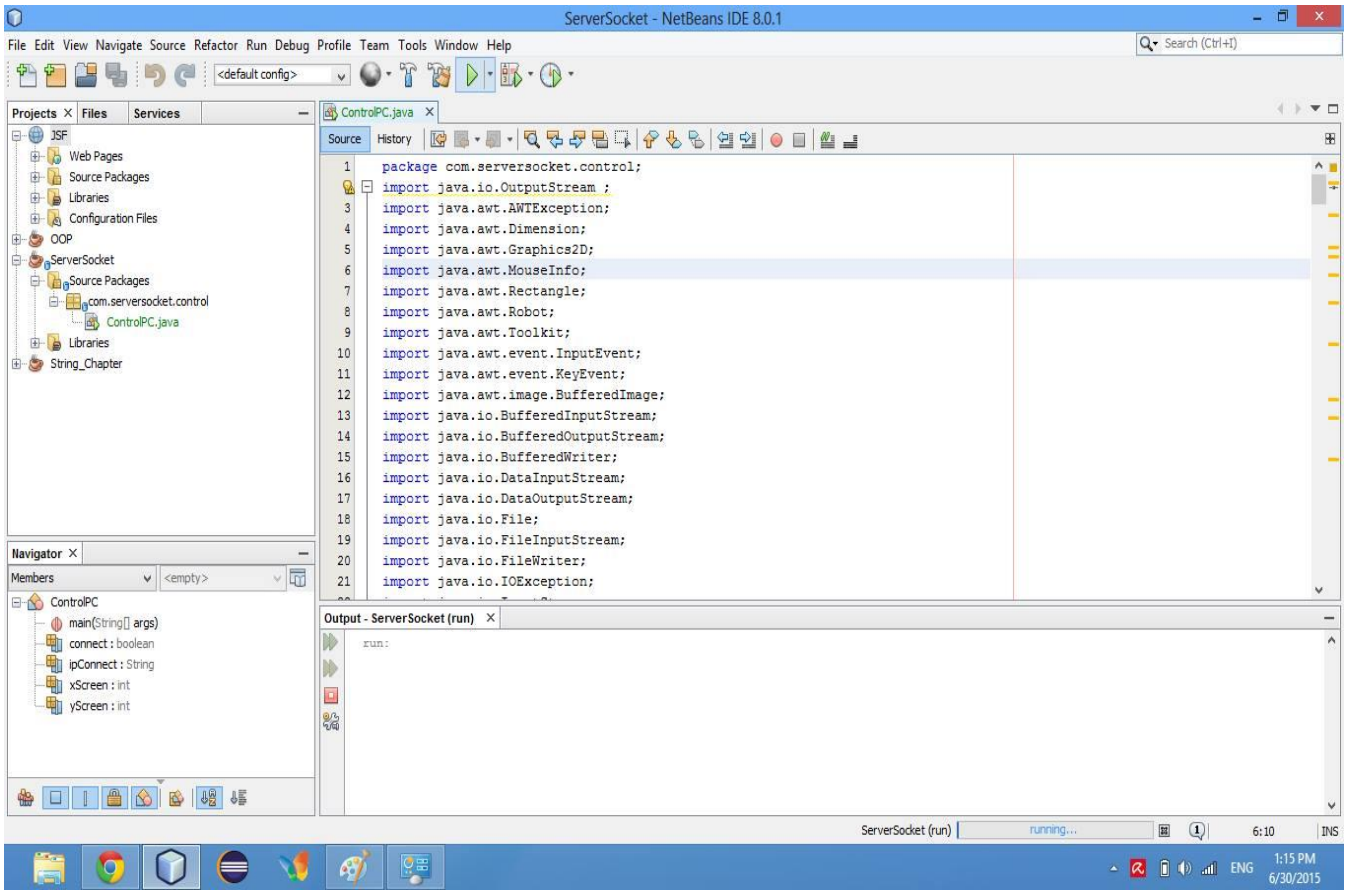


Eclipse for client

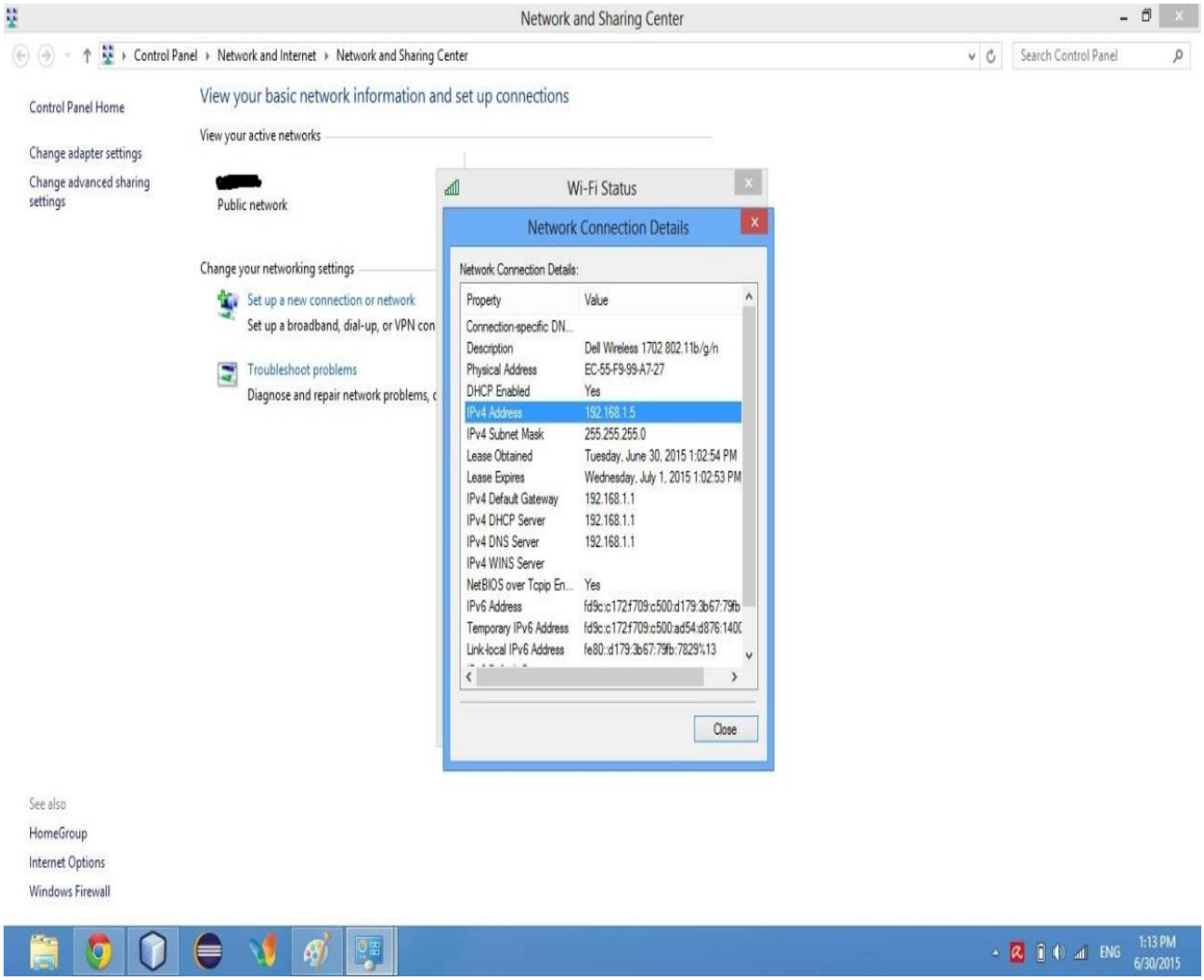


5.3 run

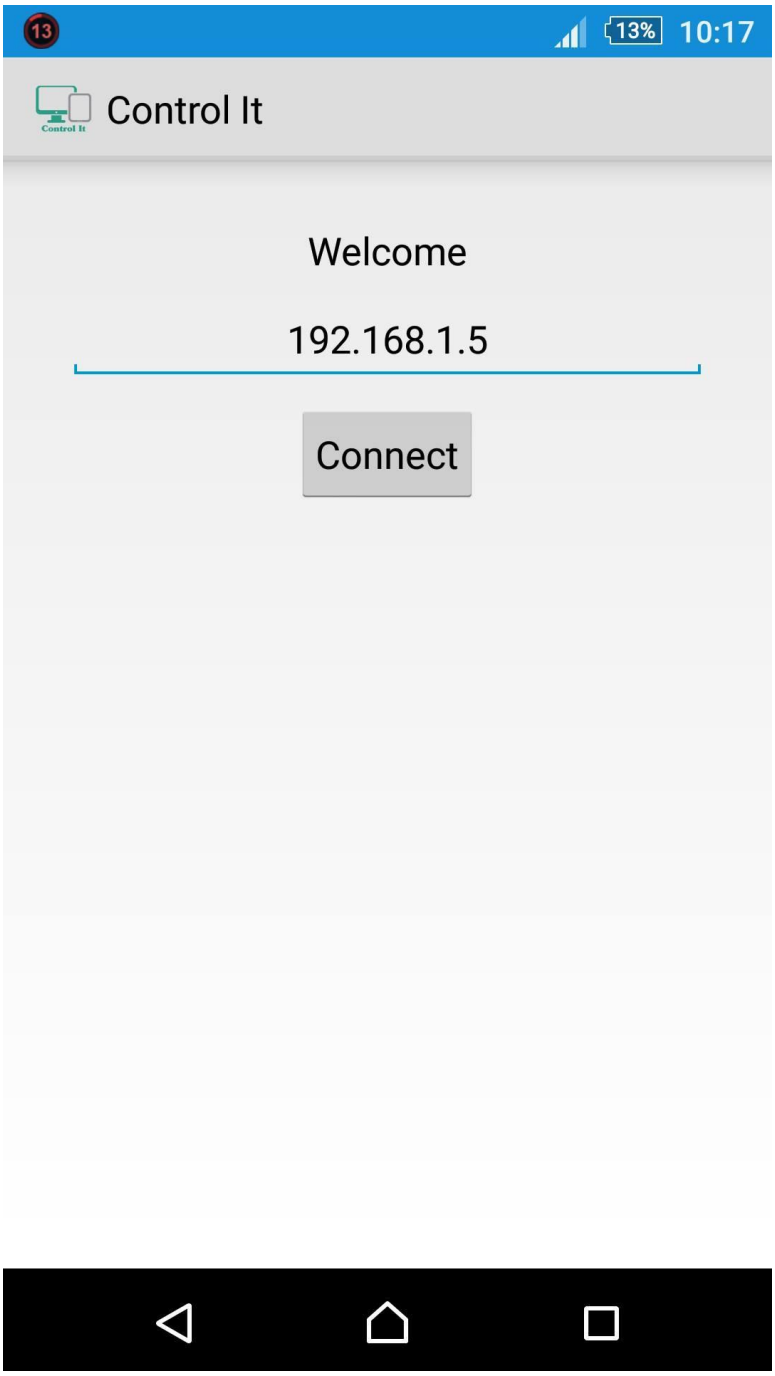
First step: you must run the server from NetBeans ide



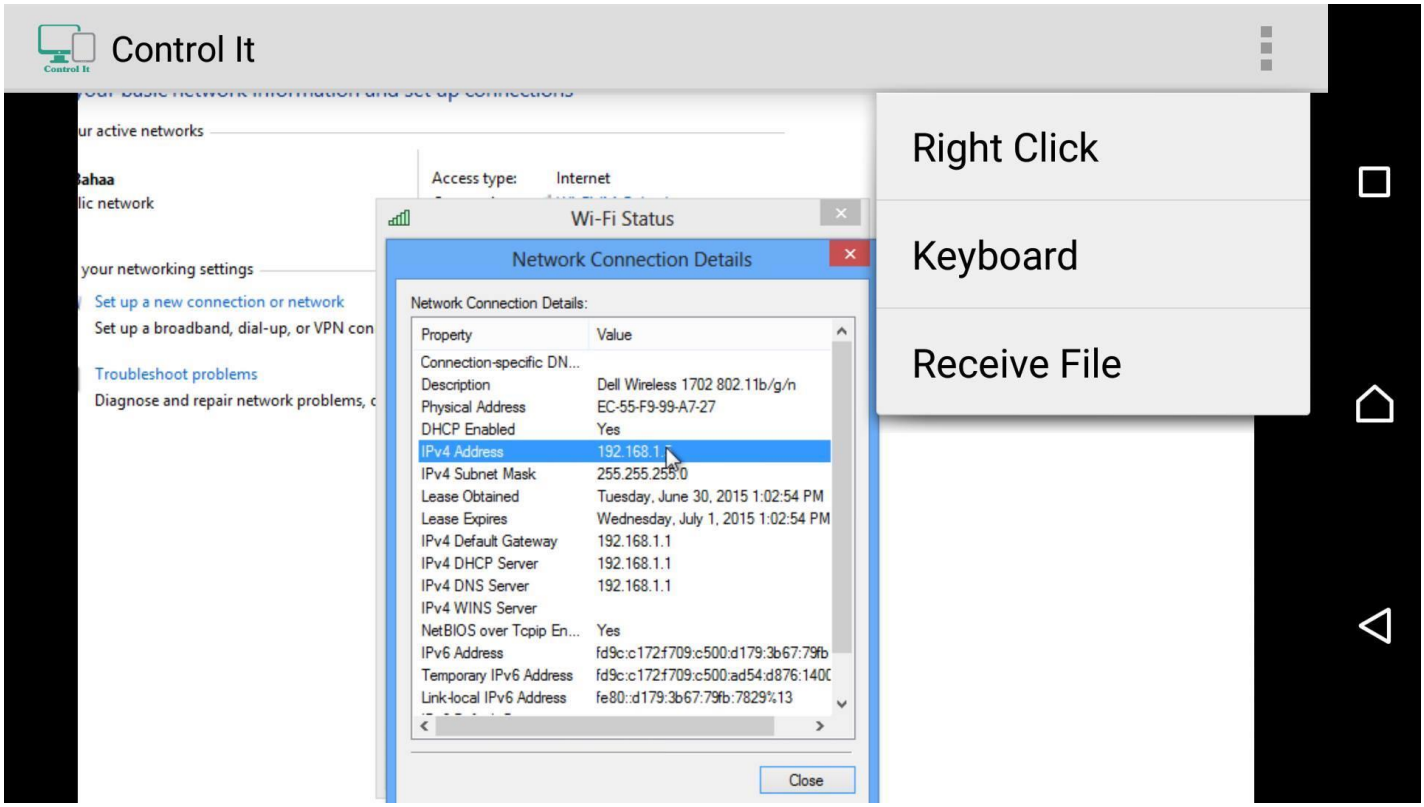
Second step: you must get the IP address of the local server as shown:



Third step: open the app from your mobile and write the IP then press connects



Fourth step: Now the computer screen appear on your mobile, you can control it, and you have options such as: keyboard that you can write a code or write any thin, you can play a game, etc...



5.4 conclusions

The Remote Desktop feature in Windows allows you to control your computer from another office or from home. Remote Desktop allows you to use the applications on your office computer, and access your data without being in your office.

Caution! There is a risk that technologies which allow you to access your desktop may be exploited by others to attempt to do the same. Be careful to properly secure your desktop to minimize this risk.

Remote Desktop enables you to connect to your computer across the Internet from virtually any computer or device. Once connected, Remote Desktop gives you mouse and keyboard control over your computer while showing you everything that's happening on the screen. With Remote Desktop, you can leave your computer at the office without losing access to your files, applications, and e-mail.

5.5 future works

We provide fully managed desktop computing solutions in the Cloud.

Fully Managed Solution

Remote Desktop makes managing your desktop infrastructure easy by centralizing your architecture and avoiding the need to manage and maintain large physical desktop deployments or complex VDI solutions (Virtual Desktop Infrastructures).

Support Multiple Devices & BYOD

Remote Desktop users can access their desktops from their choice of device, desktop and laptop computers (Microsoft Windows and Mac OS), low cost thin clients, iPhone/iPad, or Android tablet. BYOD (Bring Your Own Device) policies have never been easier to implement

High Security, High Availability

Remote Desktop provides dedicated high-speed, high-availability storage within our own private cloud environment. Corporate data is never stored on devices, helping to keep data secure and corporate data protection policies intact.

References

_http://compnetworking.about.com/od/networkprogramming/g/what-is-a-socket.htm
_https://docs.oracle.com/javase/tutorial/networking/sockets/index.html
_https://docs.oracle.com/javase/tutorial/essential/concurrency/threads.html