Mobile Computing

Unit - I

What is computing?

The utilization of computers to complete a task. It involves both hardware & software functions performing some sort of task with a computer.

Examples of computing being used in everyday life: Swiping a debit card, sending an email, or using a cell phone can all be considered forms of computing.

What is the mobility?

The capability to change location while communicating to invoke computing service at some remote computers.

What is mobile computing?

Ability to compute remotely while on the move. It is possible to access information from anywhere and at anytime.

Definition: What is mobile computing?

- Computing that is not obstructed while the location of it changes.
- Mobile Computing is using a computer (of one kind or another) while on the move.
- Computing on the go!!
- Mobile Computing is a technology that allows transmission of data, voice and video via a computer or any other wireless enabled device without having to be connected to a fixed physical link.
- The process of computation on a mobile device.
- Facilitates a large number of applications on a single device.

The Mobile Computing Structure:

1. Mobile communication
2. Mobile hardware
3. Mobile software

Mobile communication

The mobile communication in this case, refers to the infrastructure put in place to ensure that seamless and reliable communication goes on. These would include devices such as protocols, services, bandwidth, and portals necessary to facilitate and support the stated services. The data format is also defined at this stage. This ensures that there is no collision with other existing systems which offer the same service.

Since the media is unguided / unbounded, the overlaying infrastructure is basically radio wave-oriented. That is, the signals are carried over the air to intended devices that are capable of receiving and sending similar kinds of signals.

Mobile Hardware

Mobile hardware includes mobile devices or device components that receive or access the service of mobility. They would range from portable laptops, smartphones, tablet Pc's, Personal Digital Assistants.

These devices will have a receptor medium that is capable of sensing and receiving signals. These devices are configured to operate in full- duplex, whereby they are capable of sending and receiving signals at the same time. They don't have to wait until one device has finished communicating for the other device to initiate communications.

Above mentioned devices use an existing and established network to operate on. In most cases, it would be a wireless network.

Mobile software

Mobile software is the actual program that runs on the mobile hardware. It deals with the characteristics and requirements of mobile applications. This is the engine of the mobile device. In other terms, it is the operating system of the appliance. It's the essential component that operates the mobile device.

Since portability is the main factor, this type of computing ensures that users are not tied or pinned to a single physical location, but are able to operate from anywhere. It incorporates all aspects of wireless communications.

Symbian: Symbian is a very popular platform but for only one reason, cheapest smart phone OS out there. it is good, all other platforms are better, and even more importantly will get even better over time, Symbian is finally starting to lose ground.

Windows Phone: it is better for business solutions, windows phone is more a end consumer product, oriented to the social networking and online boom.

Blackberry: Great at keeping your data safe, while not as good as iOS or Android as a platform it offers many things that no other hardware does.

Android: The API is easy to use with basically infinite tools, its as flexible as iOS without all the crazy restrictions

- A software platform and operating system for mobile devices
- Based on the Linux kernel and Virus Free
- Android Open Source Project
- Developed by Google and later the Open Handset Alliance
- Allows writing managed code in the Java language

Brief about Mobile Computing OS

A mobile operating system, also known as a mobile OS, a mobile platform, or a handheld operating system, is the operating system that controls a mobile device or information appliance—similar in principle to an operating system.

Such as Windows, Mac OS, or Linux that controls a desktop computer or laptop.

However, they are currently somewhat simpler, and deal more with the wireless versions of broadband and local connectivity, mobile multimedia formats, and different input methods.

Typical examples of devices running a mobile operating system are smart phones, personal digital assistants (PDAs), and information appliances, or what are sometimes referred to as smart devices, which may also include embedded systems, or other mobile devices and wireless devices.

Symbian OS

Symbian OS has become a standard operating system for smartphones, and is licensed by more than 85 percent of the world's handset manufacturers. The Symbian OS is designed for the specific requirements of 2.5G and 3G mobile phones.

Windows Mobile

The Windows Mobile platform is available on a variety of devices from a variety of wireless operators. You will find Windows Mobile software on Dell, HP, Motorola, Palm and i-mate products. Windows Mobile powered devices are available on GSM or CDMA networks.

Palm OS or Personal Digital Assistant (PDA) - Sometimes called pocket computers

Since the introduction of the first Palm Pilot in 1996, the Palm OS platform has provided mobile devices with essential business tools, as well as capability to access the Internet or a central corporate database via a wireless connection.

Mobile Linux:

The first company to launch phones with Linux as its OS was Motorola in 2003. Linux is seen as a suitable option for higher-end phones with powerful processors and larger amounts of memory.

MXI (Motion eXperience Interface)

MXI is a universal mobile operating system that allows existing full-fledged desktop and mobile applications written for Windows, Linux, Java, and Palm are enabled immediately on mobile devices without any redevelopment.

MXI allows for interoperability between various platforms, networks, software and hardware components.

No Touchscreen interface instead uses Stylus.

Android OS

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.

iOS

iOS (originally iPhone OS) is a mobile operating system created and developed by Apple Inc. and distributed exclusively for Apple hardware.

It is the operating system that presently powers many of the company's mobile devices, including the iPhone, iPad, and iPod touch.

There are two Main Aspects of mobile computing:

User mobility: users communicate "anytime, anywhere, with anyone" (example: read/write email on web browser)

- Between different geographical locations
- Between different networks
- Between different communication devices
- Between different applications

Device portability: devices can be connected anytime, anywhere to the network

- Between different geographical locations
- Between different networks

User mobility refers to a user who has access to the same or similar telecommunication services at different places, i.e., the user can be mobile, and the services will follow him or her.

With **device portability**, the communication device moves

- Many mechanisms in the network and inside the device have to make sure that communication is still possible while the device is moving.

Aspects of mobility

We can define a computing environment as mobile if it supports one or more of the following characteristics:

User Mobility

User should be able to move from one physical location to another location and use the same service.

For Example, User moves from London to New York and uses the Internet in either place to access the corporate application.

Network Mobility

User should be able to move from one network to another network and use the same service.

For Example, User moves from Bangalore to Chennai and uses the same GSM phone to access the corporate application.

Bearer Mobility

User should be able to move from one bearer to another while using the same service.
For Example, User is unable to access the WAP bearer due to some problem in the GSM network then he should be able to use voice or SMS bearer to access that same corporate application.

Device Mobility

User should be able to move from one device to another and use the same service.

For Example, User is using a PC to do his work. During the day, while he is on the street he would like to use his Palmtop to access the corporate application.

Session Mobility

A user session should be able to move from one user- agent environment to another.

For Example, An unfinished session moving from a mobile device to a desktop computer is a good example.

Service Mobility

User should be able to move from one service to another.

For Example, User is writing a mail. Suddenly, he needs to refer to something else. In a PC, user simply opens another service and moves between them. User should be able to do the same in small footprint wireless devices.

Host Mobility

User should be able to move while the device is a host computer.

For Example, The laptop computer of a user is a host for grid computing network. It is connected to a LAN port. Suddenly, the user realizes that he needs to leave for an offsite meeting.

He disconnects from the LAN and should get connected to wireless LAN while his laptop being the host for grid computing network.

Types of Computing:

Mobile Computing: This computing environment moves along with the user. This is similar to the telephone number of a GSM (Global System for Mobile communication) phone, which moves with the phone. The offline (local) and real-time (remote) computing environment will move with the user. In real-time mode the user will be able to use all his remote data and services online.

• **Anywhere, Anytime Information:** This is the generic definition of ubiquity, where the information is available anywhere, all the time.
• **Virtual home Environment:** Virtual Home Environment (VHE) is defined as an environment in a foreign network such that the mobile users can experience the same computing experience as they have in their home or corporate computing environment. For example, one would like to keep the room heater on when one has stepped outside for about 15 minutes.
Nomadic (நாடோடி) Computing: The computing environment is nomadic and moves along with the mobile user. This is true for both local and remote services.

Pervasive Computing: A computing environment, which is pervasive in nature and can be made available in any environment.
Ubiquitous (எங்கும்) Computing: A (nobody will notice its presence) everyplace computing environment. The user will be able to use both local and remote services.
Global Service Portability: Making a service portable and available in every environment. Any service of any environment will be available globally.
Wearable Computers: Wearable computers can be worn by humans like a hat, shoe or clothes (these are wearable accessories). Wearable computers need to have some additional attributes compared to standard mobile devices. Wearable computers are always on; operational while on the move; hands-free, context-aware (with different types of sensors). Wearable computers need to be equipped with proactive attention and notifications. The ultimate wearable computers will have sensors implanted in the body and supposedly integrate with the human nervous system. These are part of a new discipline of research categorized by "Cyborg" (Cyber Organism).

Specific absorption rate (**SAR**) is a measure of the rate at which energy is absorbed by the human body when exposed to a radio frequency (RF) electromagnetic field. It can also refer to absorption of other forms of energy by tissue, including ultrasound.

Virtual home environment (VHE) refers to a network-supported mobile computing environment that allows a user to access the same computing environment on the road as they would have at home or their place of business.

The following are the four different characteristics exhibit by the communication devices.

1) **Fixed and wired:** This category describes the fixed desktop computers in the office, which are connected through the wired network. The weight and power consumption of these devices does not permit for mobility, these devices use fixed network for information accessing

2) **Mobile and wired**: The most of the portable computing devices fall within this category. We can easily carry these devices (ex: laptops) from one place to another place and reconnecting to the organization's network via the telephone lines or a modem.

3) **Fixed and wireless**: This kind of technique is especially used for installing networks in historical places such as buildings to avoid damages by fixing cables and other equipment. This method can also be used for fastest network setup.

4) **Mobile and wireless**: This is the most recent technique, in which no cable restricts, the user, who can roam between different wireless networks. Ex: GSM –Global system for mobile communications.

Mobile Computing - Major Advantages

Location Flexibility

This has enabled users to work from anywhere as long as there is a connection established. A user can work without being in a fixed position. Their mobility ensures that they are able to carry out numerous tasks at the same time and perform their stated jobs.

Saves Time

The time consumed or wasted while travelling from different locations or to the office and back, has been slashed. One can now access all the important documents and files over a secure channel or portal and work as if they were on their computer. It has enhanced telecommuting in many companies. It has also reduced unnecessary incurred expenses.

Enhanced Productivity

Users can work efficiently and effectively from whichever location they find comfortable. This in turn enhances their productivity level.

Ease of Research

Research has been made easier, since users earlier were required to go to the field and search for facts and feed them back into the system. It has also made it easier for field officers and researchers to collect and feed data from wherever they are without making unnecessary trips to and from the office to the field.

Entertainment

Video and audio recordings can now be streamed on-the-go using mobile computing. It's easy to access a wide variety of movies, educational and informative material. With the improvement and availability of high speed data connections at considerable cost, one is able to get all the entertainment they want as they browse the internet for streamed data. One is able to watch news, movies, and documentaries among other entertainment offers over the internet. This was not possible before mobile computing dawned on the computing world.

Streamlining of Business Processes

Business processes are now easily available through secured connections. Looking into security issues, adequate measures have been put in place to ensure authentication and authorization of the user accessing the services.

Some business functions can be run over secure links and sharing of information between business partners can also take place.

Meetings, seminars and other informative services can be conducted using video and voice conferencing. Travel time and expenditure is also considerably reduced.

Applications of Mobile Computing

1. **Vehicles:**

- Transmission of news, road condition, weather, music via DAB
- Personal communication using GSM
- Position via GPS
- Local ad-hoc network with vehicles close-by to prevent accidents, guidance system, redundancy
- Vehicle data (e.g., from busses, high-speed trains) can be transmitted in advance for maintenance

2. **Emergencies:** • early transmission of patient data to the hospital, current status, first diagnosis • replacement of a fixed infrastructure in case of earthquakes, hurricanes, fire etc. • Wireless networks are the only means of communication in the case of natural disasters such as earthquakes. •

3. Business: • A travelling salesman today needs instant access to the company's database: to ensure that files on his or her laptop reflect the current situation, to enable the company to keep track of all activities of their travelling employees, to keep databases consistent etc.

4. Replacement of wired networks • wireless networks can also be used to replace wired networks, • it is often impossible to wire remote sensors for weather forecasts, earthquake detection, or to provide environmental information. • Wireless connections, e.g., via satellite, can help in this situation. • Wired network is infrequent and inflexible • Many computer fairs use WLANs as a replacement for cabling. • wireless networks are computers, sensors, or information displays • in historical buildings, where excess cabling may destroy valuable walls or floors.

• 5. Infotainment and more • wireless networks can provide up-to-date information at any appropriate location. • Another growing field of wireless network applications lies in entertainment and games to enable, e.g., ad-hoc gaming networks as soon as people meet to play together.

6. Location dependent services • Many research efforts in mobile computing and wireless networks try to hide • the fact that the network access has been changed • it is important for an application to 'know' something about the location or the user might need location information for further activities. Several services that might depend on the actual location can be distinguished: • Follow-on services – automatic call-forwarding, transmission of the actual workspace to the current location • Information services – „push": e.g., current special offers in the supermarket – „pull": e.g., where is the Black Forrest Cherry Cake? • Support services – caches, intermediate results, state information etc. „follow" the mobile device through the fixed network • Privacy – who should gain knowledge about the location

Limitations of Mobile Computing:

1. **Resource constraints** - Battery needs and recharge requirements are the biggest constraints of mobile computing. When a power outlet or portable generator is not available, mobile computers must rely entirely on battery power. Combined with the compact size of many mobile devices, this often means unusually expensive batteries must be used to obtain the necessary battery life.

2. **Interference** - There may be interference in wireless signals affecting the quality of service. Weather, terrain, and the range from the nearest signal point can all interfere with signal reception. Reception in tunnels, some buildings, and rural areas is often poor.

3. **Bandwidth** — There may be bandwidth constraints due to limited spectrum availability at given instant causing connection latency. Mobile Internet access is generally slower than direct cable connections, using technologies such as GPRS and EDGE, and more recently 3G networks. These networks are usually available within range of commercial cell phone towers. Higher speed wireless LANs are inexpensive but have very limited range.

4. **Dynamic changes in communication environment** - We know that there may be variations in signal power within a region it causes link delays and connection loss.

5. **Network issues** — Due to the ad hoc networks some issues relating discovery of connection, service to destination, and connection stability.

6. **Interoperability** (இயங்குதன்மை — The varying protocol standards available between different regions may lead to interoperability issues.

7. **Security constraints** - Protocols conserving privacy of communication may be violated. Sometimes physical damage or loss of mobile device is probable than static computing system.

8. **Potential health hazards**: People who use mobile devices while driving are often distracted from driving are thus assumed more likely to be involved in traffic accidents. Cell phones may interfere with sensitive medical devices. There are allegations that cell phone signals may cause health problems.

9. **Human interface with device**: Screens and keyboards tend to be small, which may make them hard to use. Alternate input methods such as speech or handwriting recognition require training.

Advantages of Mobile Computing:

1. No location constraint: Mobile computing frees the user from being tied to a location and increased bandwidth and speed of transmission makes it possible to work on the move.
2. It saves time and enhances productivity with a better return on investment (RoI)
3. It provides entertainment, news and information on the move with streaming data, video and audio Streamlining of business processes: Mobility has enabled streamlining of business processes, cumbersome emails, paper processing, delays in communication and transmission.
4. Newer job opportunities for IT professionals have emerged and IT businesses now have an added service in their portfolio which only will keep growing as per indicative mobile computing trends.

Android Emulator is used to run, debug and test the android application. If you don't have the real device, it can be the best way to run, debug and test the application.

Mobile Application Testing

One thing is self-explanatory in case of mobile testing. To perform mobile testing, you need a mobile device. This is to access that how our product will work and look like on a given mobile set.

Suppose we are developing an application for flight ticket booking system. Once the product is entirely developed, as a part of mobile testing, we need to check if the application is working as expected with all the majorly used devices like Android phones, iOS, Blackberry phones, and other different types of tablets and iPads.

To do this kind of check, we need to acquire each such device and then we can check if the application behaves as per expectation. Yes you thought right, as a product owner one will defiantly find this very expensive to procure such a large number of mobile devices and carry out testing. So is there any smart alternate available?

The solution to this problem is to use Mobile Simulators and Mobile Emulators. These are primarily software programs designed to provide simulation for important features of a smartphone. They are very similar in nature, so sometimes, they are used interchangeably.

	Real Device	Emulator / Simulator
Price	Getting real devices will cost you a lot.	It is almost free, we just need to download and install them
Processing Speed	It has faster processing; however network latency may be normal.	It is slower as compared to actual devices. It has observed less latency than real devices connected to the local network or in the cloud.
Debugging	Debugging is not that easy.	It provides step-by-step debugging of an application. Also, it provides an efficient way for capturing screenshots.
Web-app Testing	Web applications can be tested in a normal way.	Testing a web application is much easier.
Reliability	Testing on a real device has a major advantage that it always gives accurate results.	It cannot simulate all types of user interactions; hence it may lead to false results sometimes. So it scores low when it comes to reliability.

An emulator can replace the original for 'real' use. A simulator is a model for analysis.

A simulator is an environment which models but an emulator is one that replicates the usage as on the original device or system.

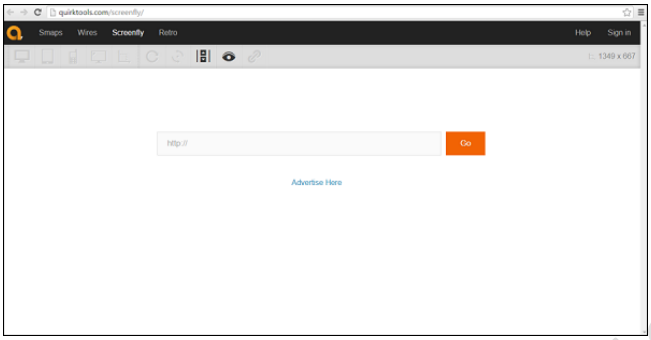
Available Tools for Mobile UI Testing

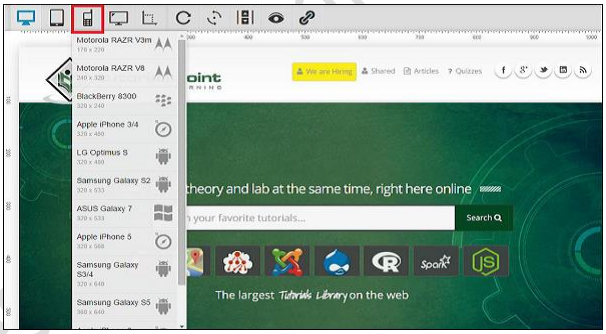
There are quite a few tools available in the market to make mobile UI testing smoother and simpler. For example −

- Google chrome extension
- **Screenfly**
- Browser Stack

Screenfly

Screenfly is a free and easy-to-use tool. To use this, you just need to type in Quirktools in your web browser. You will see the following screen.





Enter the website under test and click **Go**. Select the mobile device in which you want to view the website.

There are many Android testing frameworks available in the market. Let's take a look at the top 5 on the stack.

- **Robotium** − Robotium is an open-source test framework for developing functional, system and acceptance test scenarios. It is very similar to Selenium.
- **UIAutomator** − UIAutomator is a test framework by Google that provides advance UI testing of native Android apps and games. It has a Java library containing API to create functional UI tests and also an execution engine to run the tests.
- **Appium** − Appium is an open-source test automation framework to test native and hybrid apps and mobile web apps. Appium library functions inside the framework make calls to the Appium server running in the background which operates the connected device.
- **Calabash** − Calabash is a functional testing framework that can be used for both iOS and Android functional testing. On paper, it must be one of the easiest frameworks to use and even non-developers should be able to create functional tests using it.
- **Selendroid** − Selendroid is a relatively new kid on the block and can be used to functionally test your Android applications. Apparently, if you are used to Selenium, Selendroid should be an easy way to use your knowledge to create your functional tests for Android.

Like Android testing frameworks, there are many iOS testing frameworks available in the market. Here we will talk about a few popular ones.

- **Appium** − Appium is an open-source test automation framework to test native and hybrid apps and mobile web apps. Appium library functions inside the framework make calls to the Appium server running in background which operates the connected device.
- **Calabash** − Calabash is a functional testing framework that can be used for both iOS and Android functional testing. On paper, it must be one of the easiest frameworks to use and even non-developers should be able to create functional tests using it.
- **Zucchini** − Zucchini is an open-source visual functional testing framework for iOS applications based on Apple UIAutomation.
- **UI Automation** − For your more typical functional tests (or black-box tests), in which you're going to write code that simulates an end-user navigating your app, there is UI Automation. UI Automation is provided by Apple and is the Apple-sanctioned way of performing iOS functional testing.
- **FRANK – BDD for iOS** − If you want to do end-to-end testing in iOS and wish you could use BDD and Cucumber, no worries — there's a tool called **Frank** that will allow you to create acceptance tests and requirements using Cucumber.

Mobile applications

- Mobile applications (also known as mobile apps) are software programs developed for mobile devices such as smartphones and tablets.
- Compact software programs that performs specific tasks for the mobile user.
- A Software application that runs in a handheld device such as a Smartphone or other portable device.

Compared to desktop or notebook computers, mobile devices have relatively:

Low processing power

Limited RAM

Limited permanent storage capacity

Small screens with low resolution

Higher costs associated with data transfer

Slower data transfer rates with higher latency (delay)

Less reliable data connections

Limited battery life

It's important to keep these restrictions in mind when creating new applications.

10 steps: How to Create a Successful Mobile Application?

Step 1: A great imagination leads to a great app

To create a successful mobile application, the first thing you need to keep in mind is:

- Identify a problem which can be resolved by your app
- Decide the features of your app

The app should provide customer with tangible benefits including reducing costs via productivity enhancements, new revenue or improving the customer experience.

Step 2: Identify

To create a successful mobile app, you need to identify or be clear about:

- Application target users

An app should always be developed keeping in mind the target users of an application. Having a clear vision regarding the target group, enhance the success ratio of an app.

- Mobile platforms and devices to be supported

Mobile platforms and devices should be selected keeping in mind hardware performance, battery life, ruggedness and required peripherals. Certain factors that needs to be considered while selecting mobile platforms and devices includes coverage, device support, performance and other features.

- Revenue model

The app market is booming like never before. To ensure this resource and generate revenue, app developer need to select appropriate approach in accordance with the app. There are different models of generating revenue from mobile applications which include paid applications, separate app and in-app fermiums (Business model), advertisements, subscription and pay per download.

These techniques can be employed to generate revenue. However, the developer's approach has to be in accordance with the application. It is highly essential for the developer to attract the user and spend money on the various aspects of the application.

At this point you should also think about your finances, how much money you wish to set aside for the development, marketing and eventual release of your app.

Step 3: Design your app

Designing your app is yet another significant factor responsible for success of an app in the market. An app developer should concentrate on the UI design, multi-touch gestures for touch-enabled devices and consider platform design standards as well.

Designing an app is becoming increasingly popular as it create an instant impact on the mind of the user while ensuring usability of an app.

Step 4: Identify approach to develop the app - native, web or hybrid

Selecting the right approach for developing an app is highly important. Ideally, app development approach must be in accordance with the time and budget constraints of a client.

- **Native:**

Native apps enables in delivering the best user experience but require significant time and skill to be developed. These apps are basically platform specific and require expertise along with knowledge. Native apps are costly as well as time taking to be developed and deliver the highest user experience amongst all the approaches.

- **Web:**

Web apps are quick and cheap ones to develop and can run on multiple platforms. These are developed using HTML5, CSS and JavaScript code. These web apps are less powerful than native apps.

- **Hybrid:**

Hybrid approach is the latest approach to develop any app. This approach combines prebuilt native containers with on-the-fly web coding in order to achieve the best of both worlds. In this

approach, the developer augments the web code with native language to create unique features and access native APIs which are not yet available through JavaScript.

Step 5: Develop a prototype

Next stage, after identifying the approach is developing a prototype. It is actually the process of taking your idea and turning it into an application with some basic functionality. A prototype makes it quite easier to sell your idea to potential buyers who can now actually view the tangible benefits instead of just visualizing or reading product description. It is quite helpful in attracting investors and working with manufacturers and finding licensees.

Even while working on a prototype, do ensure you take measures to secure your app against unauthorized usage and access to data.

Step 6: Integrate an appropriate analytics tool

There is also a need to incorporate appropriate analytics which gives you a detailed picture of how many visitors use your webs, how they arrived on your site and how can they keep coming back.

Some of the mobile analytics tool which helps in this process:

Google Analytics

Localytics

Mixpanel

Preemptive

With data sciences, including predictive analytics coming up in mobile apps, it can make your apps highly marketable.

Step 7: Identify beta-testers. Listen to their feedback and integrate relevant ones

Beta testing is the first opportunity to get feedback from your target customers. It is especially important as it enhances your visibility in the app store. It not only reduce product risk but get you that initial push in the app store. To identify beta testers is another important task to ensure success of an app.

Preparing for beta launch:

Define target customer

It is highly important to identify and clearly define your target audience. This will enable you identify the right testers during your beta tester recruiting. Early market research helps in understanding market analysis which eases the process of beta testing.

Eliminate bugs

Before beta testing your app on different platforms you need to take into account majority of the devices which eliminate device specific bugs. Alpha testing with a small number of users enables

to clear out maximum bugs. At the same time, device coverage plan is significant for quality assurance of mobile app.

Identify goals

Beta testing is the best opportunity to get real feedback from target customers. It provides a great opportunity to further understand target market and their requirements. Identifying goals for beta testing helps in focusing the efforts. These goals reduce your product launch risk.

Step 8: Release / deploy the app

Deploying an app requires plan, schedule and control of the movement of releases to test and live environments. The major objective of Deployment Management is to ensure the integrity of the live environment is protected and that the correct components are released.

Step 9: Capture the metrics

There has been significant rise in the mobile app users in the present decade. As a result, the need to collect accurate metrics is highly important. As the number of consumers using mobile applications steadily rises, the need to collect accurate metrics from them is increasingly important. Unfortunately, many of the methods used to measure apps are taken from web analytics.

Step 10: Upgrade your app with improvements and new features

After capturing the metrics it becomes important to upgrade your app with improvements and innovative features. A mobile app without innovative features loses its usability in long run. Upgrading your app with innovative features enhances its visibility along with downloads of an app. Also ensure you keep updating your app to meet new guidelines offered by the various platforms, don't let your apps stagnate.

Mobile App Functions

The purposes of these apps run the gamut, from utility, productivity, and navigation to entertainment, sports, fitness, and just about any others imaginable. Social media is one of the most popular fields of mobile app development and adoption. In fact, Facebook was the most widely used app in 2017 across all platforms.
Many online entities have both mobile websites and mobile apps. In general, the difference lies in purpose: An app is usually smaller in scope than a mobile website, offers more interactivity, and presents more specific information in a format that's easy and intuitive to use on a mobile device.

Operating System Compatibility

A mobile app developer creates an app specifically for the operating system in which it will run. For example, mobile apps for the iPad are supported by Apple's iOS, but not Google's Android. An Apple app can't run on an Android phone, and vice versa.
Often, developers create a version for each; for example, a mobile app in the Apple Store might have a counterpart in Google Play.

Why Mobile Apps Are Different From "Regular" Apps

Many mobile apps have corresponding programs meant to run on desktop computers. Mobile apps have to work with different constraints than their desktop equivalents, however.

Mobile devices have a wide range of screen sizes, memory capacities, processor capabilities, graphical interfaces, buttons, and touch functions, and developers must accommodate them all.
For example, mobile app users (like website visitors) don't want to scroll sideways to see text, images, or interactive touchpoints, nor do they want to struggle reading tiny text. An additional consideration for mobile app developers is the touch interface common to mobile devices.

User Interface

The mechanism through which user access a software application is referred to as the *user interface*.

Nearly all "touchy-feely" considerations of user interface design fall within the purview of human factors.

For example, a voice user interface is better suited for an application designed for finding directions while driving than a graphical user interface because drivers cannot safely read or view the directions while driving though they can hear the directions safely. Therefore, mobile applications must be designed with *multiple channels* in mind: We will not limit ourselves to just voice or just graphical user interfaces.

Let us take a close look at these three key aspects. We will have the most focus on the first, the "user-friendliness" of the application in this text. This is one of the keys to what makes or breaks any application, particularly a mobile application. An application that is difficult to use is one that does not attract users.

For this, not only do we need to make the application easy and efficient to navigate, but we need to consider things like color, noise, timing, esthetic quality, and a variety of other qualitative factors that make the user "like" the look-and-feel of the application. A user interface is well designed when the program behaves exactly how the user thought it would. So, another factor is to think of what a typical user considers desirable. Once again, this might include a variety of esthetics, timing, color, etc.

This shows us that there is a cognitive element involved in interacting with user interfaces that not only is important in the greater picture of human-to-computer interface (HCI) but also is something that affects human factors. A background color that is very bright may seem nice the first time the device is used, but over time it may seem less and less desirable. Finally, there is the health element.

In fact, this is one of the prime areas of focus in the study of human factors; ergonomics and human factors are associated with things such as keyboard shape (so-called ergonomic keyboards).

Usability, Human Factors, and Other Considerations for Developing User Interfaces:

1. ***Intuitiveness*** (உள்ளுணர்வு) : User interfaces should be intuitive. The first time a user uses an application, he or she should be able to navigate his or her way through without too much trouble, assuming a reasonable amount of familiarity with the application domain.

2. ***Consistency*****:** A software application should present user interface components that are consistent with each other and consistent with their operating environments. For example, if one screen refers to the gender of a user by allowing the user to select between *man* and *woman*, other screens should not refer to gender in different terms such as *male* or *female*. Also, the user interface should be consistent with the user's operating environment.

3. ***Learnability*****:** The user should be able to learn how to use the user interface within the first few times of using it and remember how to use it without having to refer to manuals. This goes hand in hand with the user interface being intuitive.

4. ***Non-intrusively helpful*****:** The user interface and the underlying application should provide help and hints. There can never be too much in the way of help and hints on a user interface. A key in implementing hints and help is to make them so that they do not hinder efficient use of the application. The little helper that pops up on the screen every few minutes without an explicit invocation of the user can be annoying and cut down on the efficient use of the user interface.

5. ***Accommodating Expert Users***: A good user interface provides shortcuts for the expert users. Applications should be efficient and fast to use for expert users. As a user learns how to use the system better, he or she should be able to access the information and perform the tasks faster and faster.

6. ***Trustable***: The user interface should be predictable, trustable, and easily understood. There should be a simple set of rules that are used in building the user interface that allow the user to be able to guess what the reaction of the user interface may be.

7. ***Robustness*** (வலுவான): A good user interface should gracefully recover from user errors (e.g., display the proper dialogue boxes to guide the user when an error happens), should convey the relation to the application logic easily to the user (e.g., make sure that the user knows which data are changing, when transactions are committed, etc.), and should be fast enough and let the user know when there are long waits for responses.

Let us enumerate again the requirements on the human factor aspects of mobile application design that are related to the condition of the user:

1. *Short Transaction Cycles*: Mobile users typically do not perform tasks that involve great amounts of data entry or long transaction cycles. Mobile users typically use the devices at their disposal to perform a few quick tasks.

2. *Expectations of Consumer Devices*: Mobile users have much higher expectations for consumer devices than for PCs. For example, users cannot handle waiting for their MP3 player, PDA, or cell phone to spend several minutes to "boot-up." Users expect to turn a device on, wait for a maximum of several seconds, and then begin to use the device.

3. *Lack of Focus*: Mobile users are not focused on the task of computing. Because the mobile user is frequently using the mobile application while moving (driving, walking, going from place to place, etc.), he or she has to do multiple things at the same time. This becomes a big consideration in the human factor aspects of the user interface design.

4. *Intermittent Network Connectivity*: Mobile devices have unreliable connections to the network, so the device may be disconnected from the network at any time.

5. *Multichannel User Interfaces*: As we will see later on in this chapter, mobile applications use a large variety of user interfaces to communicate with the user. This gives a mobile application more flexibility

Text-to-speech (TTS)

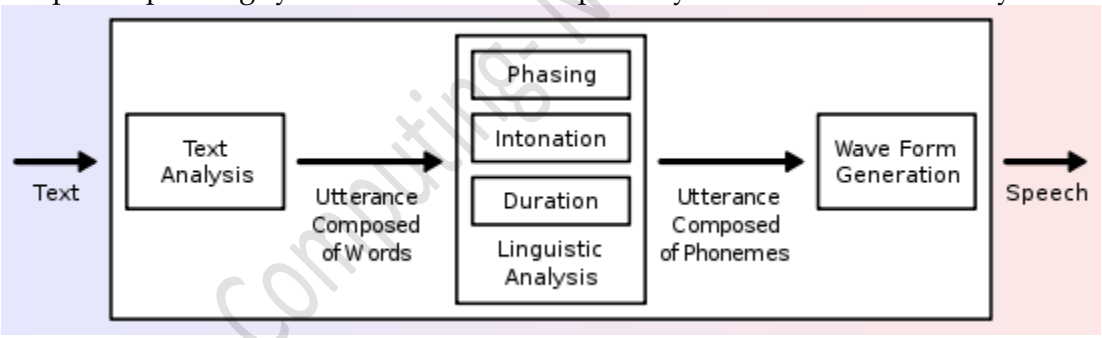
Text-to-speech (TTS) is a type of speech synthesis application that is used to create a spoken sound version of the text in a computer document, such as a help file or a Web page. TTS can enable the reading of computer display information for the visually challenged person, or may simply be used to augment the reading of a text message. Current TTS applications include voice-enabled e-mail and spoken prompts in voice response systems.

Speech synthesis is the artificial production of human speech. A computer system used for this purpose is called a **speech computer** or **speech synthesizer**, and can be implemented in software or hardware products.

A **text-to-speech** (**TTS**) system converts normal language text into speech; other systems render symbolic linguistic representations like phonetic transcriptions into speech.

Synthesized speech can be created by concatenating pieces of recorded speech that are stored in a database. Systems differ in the size of the stored speech units; a system that stores phones or diaphones provides the largest output range, but may lack clarity. For specific usage domains, the storage of entire words or sentences allows for high-quality output. Alternatively, a synthesizer can incorporate a model of the vocal tract and other human voice characteristics to create a completely "synthetic" voice output.

The quality of a speech synthesizer is judged by its similarity to the human voice and by its ability to be understood clearly. An intelligible text-to-speech program allows people with visual impairments or reading disabilities to listen to written works on a home computer. Many computer operating systems have included speech synthesizers since the early 1990s.



A text-to-speech system (or "engine") is composed of two parts: A front-end and a back-end. The front-end has two major tasks. First, it converts raw text containing symbols like numbers and abbreviations into the equivalent of written-out words. This process is often called *text normalization*, *pre-processing*, or *tokenization*. The front-end then assigns phonetic transcriptions to each word, and divides and marks the text into prosodic units, like phrases, clauses, and sentences. The process of assigning phonetic transcriptions to words is called *text-to-phoneme* or *grapheme-to-phoneme* conversion. Phonetic transcriptions and prosody information together make up the symbolic linguistic representation that is output by the front-end. The back-end—often referred to as the *synthesizer*—then converts the symbolic linguistic representation into sound. In certain systems, this part includes the computation of the *target prosody* (pitch contour, phoneme durations), which is then imposed on the output speech.

Text-to-speech applications are those applications that change written language into spoken language.

Overview

Text-to-speech fundamentally functions as a pipeline that converts text into PCM digital audio. The elements of the pipeline are:

- Text normalization
- Homograph disambiguation
- Word Pronunciation
- Prosody
- Concatenate wave segments

Text Normalization

JText Normalization it is that part of text-to-speech program that converts any input text into a series of spoken words. At basic level, text normalization converts a string like "My name is Ritesh" to a series of words, “My”, “name”, “is”, “Ritesh", along with a marker indicating that a period occurred (a comma). However, this gets more complicated when strings like "John rode home at 23.5 mph", where "23.5 mph" is converted to "twenty three point five miles per hour". Here’s how text normalization works:

First, text normalization isolates words in the text. For the most part this is as minor as looking for a sequence of alphabetic characters, allowing for an occasional apostrophe, space and hyphen.

Text normalization then searches for numbers, times, dates, and other symbolic representations. These are analyzed and converted to words. (Example: "$54.32" is converted to "fifty four dollars and thirty two cents.") Someone needs to code up the rules for the conversion of these symbols into words, since they differ depending upon the language and context.

Next, abbreviations are converted, such as "in." for "inches", and "St." for "street" or "saint". The normalizer will use a database of abbreviations and what they are expanded to. Some of the expansions depend upon the context of surrounding words, like "St. John" and "John St.”

The text normalizer might perform other text transformations such as internet addresses. "http://www.Microsoft.com" is usually spoken as "w w w dot Microsoft dot com".

Whatever remains is punctuation. The normalizer will have rules dictating if the punctuation causes a word to be spoken or if it is silent. (Example: Periods at the end of sentences are not spoken, but a period in an Internet address is spoken as "dot.")

The rules will vary in complexity depending upon the engine.

Homograph Disambiguation

So by reading till here you were wondering that all of the main task has happened but dude there are other things also whose care must be take otherwise your text to speech won't work efficiently. This stage mainly deals with pronunciation of words.

Actually it's not a stage by itself, but is combined into the text normalization or pronunciation components. I've separated homograph disambiguation out since it doesn't fit cleanly into either.

In English and many other languages, there are hundreds of words that have the same text, but different pronunciations. A common example in English is "read," which can be pronounced "reed" or "red" depending upon its meaning. A "homograph" is a word with the same text as another word, but with a different pronunciation. The concept extends beyond just words, and into abbreviations and numbers. "Ft." has different pronunciations in "Ft. Wayne" and "100 ft.". Likewise, the digits "1997" might be spoken as "nineteen ninety seven" if the author is talking about the year, or "one thousand nine hundred and ninety seven" if the author is talking about the number of people at a concert.

So the above procedure is quite tough how a computer could know that when to pronounce "read" as "reed" or as "red". One way is by judging what out what the text is actually talking about and decides which meaning is most appropriate given the context. Once the right meaning is know, it's usually easy to guess the right pronunciation.

Text-to-speech engines figure out the meaning of the text, and more specifically of the sentence, by parsing the sentence and figuring out the part-of-speech for the individual word (see how complicated it is, do you remember how many part of speech are there?). This is done by guessing the part-of-speech based on the word endings, or by looking the word up in a lexicon. Sometimes a part of speech will be ambiguous until more context is known, such as for "read." Of course, disambiguation of the part-of-speech may require hand-written rules.

Once the homographs have been disambiguated, the words are sent to the next stage to be pronounced.

Word Pronunciation

The pronunciation module accepts the text, and outputs a sequence of phonemes, just like you see in a dictionary.

To get the pronunciation of a word, the text-to-speech engine first looks the word up in its own pronunciation lexicon. If the word is not in the lexicon then the engine reverts to "letter to sound" rules.

Now what is Letter-to-sound rules, it guess the pronunciation of a word from the text. They're kind of the inverse of the spelling rules you were taught in school. There are a number of techniques for

guessing the pronunciation, but the algorithm described here is one of the more easily implemented ones.

The letter-to-sound rules are "trained" on a lexicon of hand-entered pronunciations. The lexicon stores the word and it's pronunciation, such as:

hello h eh l oe

An algorithm is used to segment the word and figure out which letter "produces" which sound. You can clearly see that "h" in "hello" produces the "h" phoneme, the "e" produces the "eh" phoneme, the first "l" produces the "l" phoneme, the second "l" nothing, and "o" produces the "oe" phoneme. Of course, in other words the individual letters produce different phonemes. The "e" in "he" will produce the "ee" phoneme.

Once the words are segmented by phoneme, another algorithm determines which letter or sequence of letters is likely to produce which phonemes. The first pass figures out the most likely phoneme generated by each letter. "H" almost always generates the "h" sound, while "o" almost always generates the "ow" sound. A secondary list is generated, showing exceptions to the previous rule given the context of the surrounding letters. Hence, an exception rule might specify that an "o" occurring at the end of the word and preceded by an "l" produces an "oe" sound. The list of exceptions can be extended to include even more surrounding characters.

When the letter-to-sound rules are asked to produce the pronunciation of a word they do the inverse of the training model. To pronounce "hello", the letter-to-sound rules first try to figure out the sound of the "h" phoneme. It looks through the exception table for an "h" beginning the word followed by "e"; Since it can't find one it uses the default sound for "h", which is "h". Next, it looks in the exceptions for how an "e" surrounded by "h" and "l" is pronounced, finding "eh". The rest of the characters are handled in the same way.

This technique can pronounce any word, even if it wasn't in the training set, and does a very reasonable guess of the pronunciation, sometimes better than humans. It doesn't work too well for names because most names are not of English origin, and use different pronunciation rules. (Example: "Mejia" is pronounced as "meh-jee-uh" by anyone that doesn't know it is Spanish.) Some letter-to-sound rules first guess what language the word came from, and then use different sets of rules to pronounce each different language.

Word pronunciation is further complicated by people's laziness. People will change the pronunciation of a word based upon what words precede or follow it, just to make the word easier to speak. An obvious example is the way "the" can be pronounced as "thee" or "thuh". Other effects including the dropping or changing of phonemes. A commonly used phrase such as "What you doing?" sounds like "Wacha doin?"

Once the pronunciations have been generated, these are passed onto the prosody stage.

Prosody

Now you will be thinking its all over but “kahani abhi baki hai dost” Now since you are able to pronounce the word but by merely pronouncing the word is not sufficient you need to speak in a tone, in a speed and other factors are also involved in a speech. This section will deals with all these factors.

Prosody is the pitch, speed, and volume that syllables, words, phrases, and sentences are spoken with. Without prosody text-to-speech sounds very robotic, and with bad prosody text-to-speech sounds like it’s drunk.

The technique that engines use to synthesize prosody varies, but there are some general techniques.

First, the engine identifies the beginning and ending of sentences. In English, the pitch will tend to fall near the end of a statement, and rise for a question. Likewise, volume and speaking speed ramp up when the text-to-speech first starts talking, and fall off on the last word when it stops. Pauses are placed between sentences.

Engines also identify phrase boundaries, such as noun phrases and verb phrases. These will have similar characteristics to sentences, but will be less pronounced. The engine can determine the phrase boundaries by using the part-of-speech information generated during the homograph disambiguation. Pauses are placed between phrases or where commas occur.

Algorithms then try to determine which words in the sentence are important to the meaning, and these are emphasized. Emphasized words are louder, longer, and will have more pitch variation. Words that are unimportant, such as those used to make the sentence grammatically correct, are de-emphasized. In a sentence such as "John and Bill walked to the store", the emphasis pattern might be "JOHN and BILL walked to the STORE." The more the text-to-speech engine "understands" what’s being spoken, the better it’s emphasis will be.

Next, the prosody within a word is determined. Usually the pitch and volume rise on stressed syllables.

All of the pitch, timing, and volume information from the sentence level, phrase level, and word level are combined together to produce the final output. The output from the prosody module is just a list of phonemes with the pitch, duration, and volume for each phoneme.

Play Audio

The speech synthesis is almost done by this point. All the text-to-speech engine has to do is convert the list of phonemes and their duration, pitch, and volume, into digital audio.

Methods for generating the digital audio will vary, but many text-to-speech engines generate the audio by concatenating short recordings of phonemes. The recordings come from a real person. In a simplistic form, the engine receives the phoneme to speak, loads the digital audio from a database, does some pitch, time, and volume changes, and sends it out to the sound card.

It isn't quite that simple for a number of reasons.

Most noticeable is that one recording of a phoneme won't have the same volume, pitch, and sound quality at the end, as the beginning of the next phoneme. This causes a noticeable glitch in the audio. An engine can reduce the glitch by blending the edges of the two segments together so at their intersections they both have the same pitch and volume. Blending the sound quality, which is determined by the harmonics generated by the voice, is more difficult, and can be solved by the next step.

The sound that a person makes when he/she speaks a phoneme, changes depending upon the surrounding phonemes. If you record "cat" in sound recorder, and then reverse it, the reversed audio doesn't sound like "tak", which has the reversed phonemes of cat. Rather than using one recording per phoneme (about 50), the text-to-speech engine maintains thousands of recordings (usually 1000-5000). Ideally it would have all possible phoneme context combinations recorded, 50 * 50 * 50 = 125,000, but this would be too many. Since many of these combinations sound similar, one recording is used to represent the phoneme within several different contexts.

Even a database of 1000 phoneme recordings is too large, so the digital audio is compressed into a much smaller size, usually between 8:1 and 32:1 compression. The more compressed the digital audio, the more muted the voice sounds.

Once the digital audio segments have been concatenated they're sent off to the sound card, making the computer talk.

Generating a Voice

You might be wondering, "How do you get thousands of recordings of phonemes?"

The first step is to select a voice talent. The voice talent then spends several hours in a recording studio reading a wide variety of text. The text is designed so that as many phonemes sequence combinations are recorded as possible. You at least want them to read enough text so there are several occurrences of each of the 1000 to 5000 recording slots.

After the recording session is finished, the recordings are sent to a speech recognizer which then determines where the phonemes begin and end. Since the tools also know the surrounding phonemes, it's easy to pull out the right recordings from the speech. The only trick is to figure out which recording sounds best. Usually an algorithm makes a guess, but someone must listening to the phoneme

recordings just to make sure they’re good.

The selected phoneme recordings are compressed and stored away in the database. The result is a new voice.

Speech-Synthesis Languages and Tools

SSML (Speech Synthesis Markup Language)

SSML is designed with the following basic principles in mind:

1. *SSML is an XML-based language*. Not only is this valuable in terms of providing a standard textual parsing mechanism.

2. *Text normalization is provided*. Text normalization is the ability to tell the system to pronounce 40# as "forty pounds" and not "forty number sign".

3. *SSML supports pronunciation specification using phonemes*. Phonemes are those strange-looking characters used in the dictionary to show how something is pronounced.

4. *It has the ability to specify some of the qualities of speech*. The SSML specification refers to the ability of changing the pitch, timing, speaking rate, and a variety of other features that make machine-generated pronunciations more humanlike, such as *prosody*.

5. *It has the ability to integrate audio into the generated output*. Many platforms have special functionality in producing more humanlike speech. SSML provides a hook for such functionality so that if some of the audio is produced by functionality outside of SSML it can be integrated.

6. *It can apply styling in a modular manner*. With Web-based GUIs, we have the ability to apply CSS to modularize the formatting and look-and-feel. One of the considerations in design of SSML was the ability to apply ACSS (which we will look at in the next section) to modularize the "sound-and-feel" of the speech being generated by the speech-synthesis system. An example may be generating speech with a British accent for users of a given system in England as opposed to generating speech with an American accent for users of a given system in the United States.

Unit – II & IV

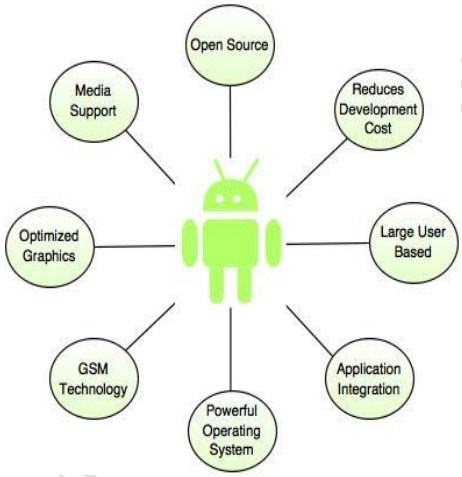
Android:

Android is an operating system which is based on the Linux kernel. Android system is also called as **Android Open Source Project (AOSP)**, led by Google. Android is used for mobile devices, such as smart phones and tablet computers. The Android application makes life more comfortable and advanced for the users.

Features of Android:

1) **Application Framework**: It enabling reuse and replacement of components.
2) **Dalvik virtual Machine:** It is optimized for mobile devices
3) **Integrated browser:** It is based on open source Web Kit engine
4) **Optimized Graphics:** Powered by a custom 2D graphics Library, 3D graphics based on the OpenGL ES (Open Graphics Library for Embedded System).
5) **SQ Lite:** Used for structured data storage
6) **Media Support:** Used for common audio, video, and still image formats (MPEG4, H. 264, MP3, AAC,AMR, JPG, PNG, GIF)
7) **GSM Telephony, Bluetooth, EDGE, 3G, and WIFI, Camera, GPS, compass, and accelerometer:** All are hardware dependent. So its feature changes according to the hardware.
8) **Rich Development Environment:** Including a device emulator, tools for debugging, memory and performance profiling, and a plug-in for the Eclipse IDE

The diagram shows some features of Android:



- Android is a powerful, open source, Linux – based operating system.
- It provides a rich development environment for building the applications.
- Android has a higher success ratio, because it reduces the development cost.
- Android has a large developer community which integrates the internal application.
- It has reached the top of the smart phone market segment and day by day its user base is growing strong.
- Android supports audio, video formats like JPEG, PNG, GIF, BMP, MP3, MP4, MIDI, AMR, AMR-WB, MPEG-4 etc.

Advantages of Android

- Android is largely supported by Google allowing you to use various services of Google.
- Android is an open source and runs on mobile devices, tablets etc.
- It is multitasking that means you can run many applications at the same time. For example, you can browse Facebook while listening the song.

- The Android operating system is available on mobile phones from various manufacturers like Samsung, Motorola, HTC, Sony Ericsson etc.
- Using Android phone, you can easily check e-mail from Gmail if your Gmail account is integrated with Google Services.
- User can easily access a variety of settings quickly and easily.

Disadvantages of Android

- Android requires continuous Internet connection if you are using Google services.
- Android shows error & forces to close the large apps/games, which is very annoying.
- It takes large amount of mobile data if a large number of background processes are running.
- It increases the usage of RAM and decreases battery performance when many processes are running in the background.

Android Application Components

Following are the Android Application Components:

1. Activity
2. Services
3. Content Providers
4. Broadcast Receivers

1. Activity

- Activity is also known as Widgets.
- Activity represents a single screen with a user interface.
- It is an individual user interface screen in an Android Application where visual elements called Views.
- It interacts with the user to do only one thing, such as unlock screen, dial a phone, etc.
- If new activity starts, then previous activity is stopped, but the data is preserved.
- An application consists of multiple activities.

For example, an email application has one activity to display a list of new emails, another activity is to compose email, reading email and so on.

2. Services

- Services performs the action without user interaction in the background, but does not get initiated without user invocation.
- It does not require a user interface.
- It is an android application component which runs in a background and has no visual UI.
- It is used to perform the processing part of your application in the background.

For example, music player application. When the music station is playing the song, the user can open another application and the song plays in the background.

3. Content Providers

- Content providers are the android application component that provide a flexible way to make data available across applications.
- It manages common data based on permissions.
- It manages the data which is being shared by more than one application.

4. Broadcast Receivers

- Broadcast receivers are used to receive messages which are broadcast by the Android applications.
- They respond to broadcast messages from other applications.

For example, the warning where the battery is getting low, change of time zone, etc.

What is a widget?

In Android, the word widget is a generic term for a bit of self-contained code that displays a program, or a piece of a program, that is also (usually) a shortcut to a larger application. We see them every day on web pages, on our computer desktop and on our smartphones, but we never give too much thought into how great they are. Widgets first appeared in Android in version 1.5.

Android Architecture

Android architecture is a stack of software components. It is in the form of a software application, operating system, run-time environment, middleware, native libraries and services. Each part of the stack and the elements within each layer are integrated and provide optimal application development and execution environment for mobile devices.

It is categorized into five parts as below:

1. Linux Kernel
2. Native Libraries
3. Android Runtime
4. Application Framework
5. Applications

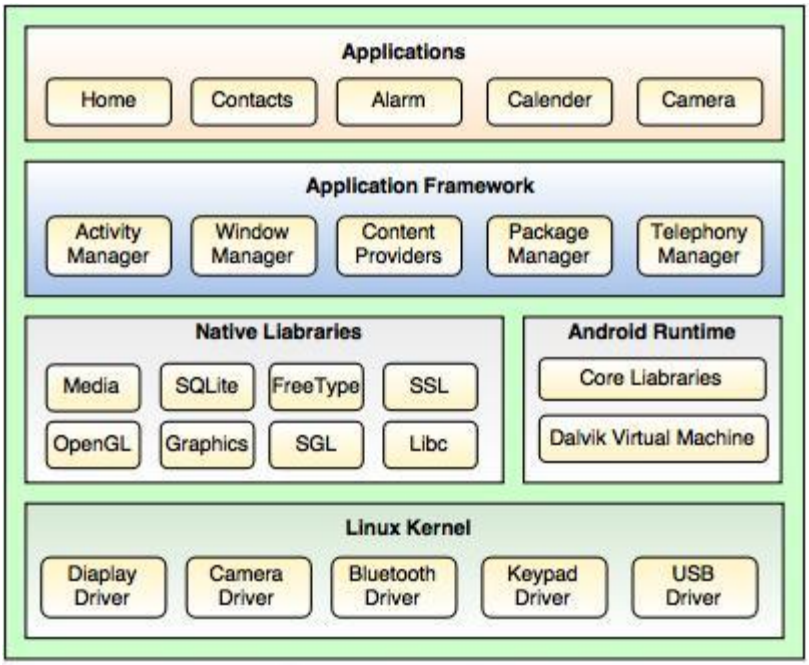


Fig. Android Architecture

1. Linux Kernel

- Linux is the heart of Android architecture.
- It provides a level of abstraction between the hardware devices and the upper layers of the Android software stack.
- The Android operating system is based on the Linux kernel.
- The Linux kernel is responsible for various device drivers such as Camera driver, Display driver, Bluetooth driver, Keypad driver, Memory management, Process management, Power management, etc.

2. Native Libraries

- The native libraries such as Media, WebKit, SQLite, OpenGL, FreeType, C Runtime library (libc) etc. are situated on the top of a Linux kernel.
- Media library is responsible for playing and recording audio and video formats, FreeType is for font support, WebKit is for browser support, SQLite is for database, SSL is for Internet security etc.

3. Android Runtime

- Android Runtime includes core libraries and Dalvik Virtual Machine (DVM) which is responsible to run android application.

- Dalvik Virtual Machine (**DVM**) is like Java Virtual Machine (**JVM**) in Java, but DVM is optimized for mobile Devices.
- DVM makes use of the Linux core features like memory management and multi-threading, which are essential in the Java language.
- DVM provides fast performance and consumes less memory.

Dalvik Virtual Machine

Dalvik VM (DVM) was developed by Google, led by Dan Bornstein, to optimize the performance of Java applications on mobile devices with limited capabilities (Dalvik is actually the name of a town in Iceland). DVM takes the traditional Java classes (".class") and combines them into one or more Dalvik Executable (".dex") files. By removing duplicate information among the Java classes, it reduces the resultant file size compared with the traditional JAR file. However, as a result, the DVM is not binary-compatible with Java Virtual Machine (JVM). You need to convert the ".class" into ".dex".

4. Application Framework

- Android framework provides a lot of classes and interfaces for Android application development and higher level services to the applications in the form of Java classes.
- It includes Android API's such as Activity manager, Window manager, Content Provider, Telephony Manager, etc.
- Activity manger is responsible for controlling all the aspects of the application lifecycle and activity stack, Content provider is responsible for allowing the applications to publish and share the data with the other applications, View system is responsible for creating application user interfaces ,Notifications Manager − Allows applications to display alerts and notifications to the user, etc.

5. Applications

- Applications are situated on the top of the Application framework.
- The applications such as Home, Contact, Alarm, Calendar, Camera, Browsers, etc. use the Android framework which uses Android runtime and libraries. Android runtime and Native libraries use Linux kernel.
- The user can write his/her application to be installed on this layer only.

Dalvik Virtual Machine | DVM

Dalvik is a purpose built virtual machine designed specifically for android which was developed by *Dan Bornstein and his team*. Strictly it was developed for mobile devices. While developing Dalvik Virtual Machine Dan Bornstein and his team realize the constraints specific to the mobile environment which is not going to change in future at least, like battery life, processing power and many more. So they optimized the Dalvik

virtual machine. Dalvik virtual machine uses register based architecture. With this architecture Dalvik virtual machine has few advantages over JAVA virtual machine such as:

1. Dalvik uses its own 16 bit instruction set than java 8 bit stack instructions, which reduce the Dalvik instruction count and raised its interpreter speed.
2. Dalvik use less space, which means an uncompressed **.dex** file is smaller in size(few bytes) than compressed java archive file(.jar file).

Why Java Virtual Machine is not used?

There is a key aspect of replacing the Java virtual machine with the Dalvik virtual machine is its licensing. The java language, java tools and java libraries are free but the java virtual machine (which is a Stack Machines) is not. Nowadays there are other open source alternatives to Oracle Java virtual machine such as Open JDK and Apache Harmony projects. Android provides a full featured platform by developing a truly open source and license friendly virtual machine and it also encouraged the developers and mobile companies to adopt for a variety of devices without having to worry about the license.

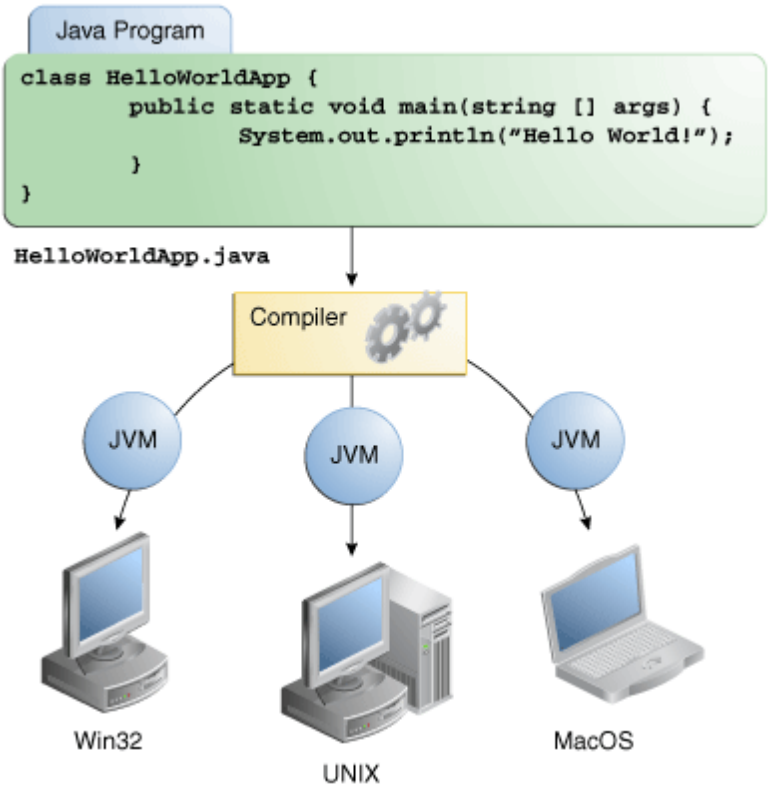
Role of Dalvik Virtual Machine

The role of Dalvik virtual machine is that, In java we write and compile java program using java compiler and run that bytecode on the java virtual machine. On the other side, In android we still write and compile java source file(bytecode) on java compiler, but at that point we recompile it once again using Dalvik compiler to Dalvik bytecode(dx tool converts java .class file into .dex format) and this Dalvik bytecode is then executed on the Dalvik virtual machine.

*Note: Dalvik team have added **Just In Time (JIT) compiler** to the Dalvik Virtual Machine. The JIT is a software component which takes application code, analyzes it, and actively translates it into a form that runs faster, doing so while the application continues to run.*

The Java VM is stack-based virtual machine. Once Java compiler translates Java source code into .class (byte code), the JVM runtime environment executes byte code or .jar files emulating JVM instruction set by interpreting or by Just-in Time (JIT) compiling.

As shown in the following picture, the byte code generated by the compiler can be used by multiple Java virtual machines running on different operating systems. For example, byte code generated on Windows OS can be fed to JVM running on Linux OS without the need to recompile.



Dalvik Virtual Machine (DVM) is register-based and the use of registers instead of stack is mostly due to use of DVM for mobile devices. Android is an open source mobile platform from Google and is based on Linux kernel with DVM as the process or application VM. Unlike regular desktop computing, mobile devices have special design constrains such as:

- Limited memory
- Limited processor speed
- Variety of screen sizes and resolutions
- No swap space
- Battery power

The DVM architecture design takes into account most the above mentioned constraints making it different from JVM.

JVM vs. DVM

Java Virtual Machines and Dalvik Virtual Machines have significant differences in the architecture design and functionality. We present comparison of two VMs using memory usage, architecture design, compilation techniques, and library support as parameters.

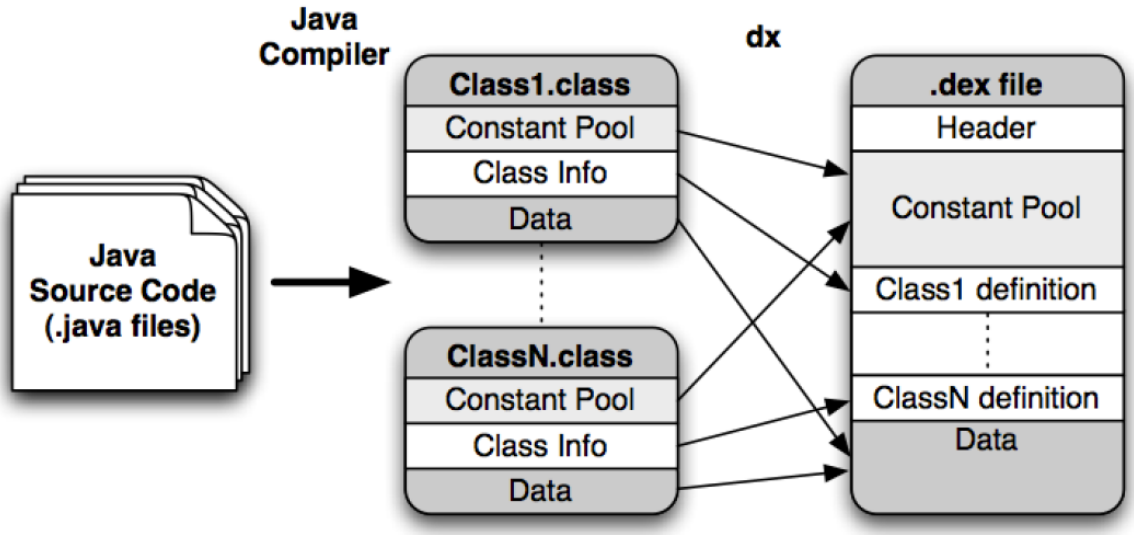
Memory Usage Comparison

Java Virtual machine (JVM) uses heap memory for its application. It has a built-in garbage collector that manages the internal memory. When a java program needs memory, it requests memory from JVM. In case there is no memory left, JVM automatically reclaims memory for reuse using garbage collector without memory allocation and deallocation. This feature eliminates memory leaks and other memory-related problems. However, JVM uses most of its resources on garbage collection, which leads to serious performance problem. For example, JVM has trouble releasing more of its memory when an "out of memory" exception is thrown. JVM uses big proportion of its memory for runtime libraries created in shared memory. On the contrary, in Dalvik VM, programs are commonly written in Java, compiled to byte code, and then converted from JVM-compatible .class files to Dalvik Executable files, which can be executed directly. The compact Dalvik Executable format has low memory requirement, which is suitable for systems with limited memory and processor speed, i.e. mobile phones, tablet computers, embedded devices such as smart TVs.

Architecture Comparison

The JVM architecture is designed to support most of the popular operating systems whereas DVM architecture is specifically targeted for the Android platform. Since mobile devices run in a constraint environment, they must make efficient use of storage, memory, battery power, and processor power. Dalvik's register-based architecture allows Android to be more efficient and faster compared to the stack-based design of the JVM. Also, Android platform is designed to run apps from thousands of users and vendors; it must provide a high-level of security for individual apps as well as for the platform itself. Android provides security by giving each app its own virtual machine which is different than the JVM approach where all applications share same virtual machine.

The following picture shows a dex file which combines multiple .class files into a single file. Different items such as constants, variables, methods, and classes are grouped into separate sections in the dex file and then are accessed by respective classes through pool indexing.



Because each app in Android runs in its own process with its own instance of the Dalvik virtual machine, a device can run multiple VMs efficiently with minimum memory. This feature in part possible due to use of Dalvik Executable (.DEX) file format on DVM.

Multiple Instance and JIT Comparison

JVM runtime executes .class or .jar files using a just-in-time compiler (JIT). JIT causes delay in initial execution of an application due to the time it takes to load and compile the byte code. In the worst case, it may crash the system if resources become unavailable for applications. These become an obvious disadvantage when it is used in limited system resources such as tablets and cell phones. Dalvik uses ahead-of-time optimization that involves the instruction modification. Therefore, it allows multiple instances of VM to run simultaneous with low memory requirement.

Reliability Comparison

In current standard Java runtime systems, the failure of a single component can have significant impacts on other components. In the worst case, a malicious or erroneous component may crash the whole system. On the other hand, Dalvik runs every instance of VM in its own separate process. Separate processes prevent all applications from crashing in case if the VM for a specific app crashes.

Supported Libraries Comparison

The Dalvik VM like JVM has built-in support for core java programming packages. In addition to core packages, Dalvik has its own set of packages such as com.google.* and android.*. The following table lists subset of packages for Dalvik and standard Java.

Libraries	Dalvik	Standard Java
java.io	Yes	Yes
java.net	Yes	Yes
android.*	Yes	no
com.google.*	Yes	No
javax.swing.*	No	Yes

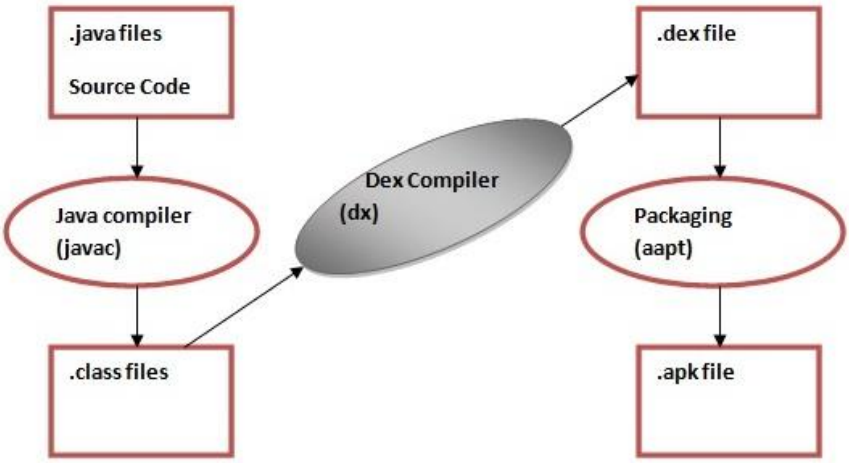
As we know the modern JVM is high performance and provides excellent memory management. But it needs to be optimized for low-powered handheld devices as well.

The **Dalvik Virtual Machine (DVM)** is an android virtual machine optimized for mobile devices. It optimizes the virtual machine for *memory, battery life* and *performance*.

Dalvik is a name of a town in Iceland. The Dalvik VM was written by Dan Bornstein.

The Dex compiler converts the class files into the .dex file that run on the Dalvik VM. Multiple class files are converted into one dex file.

Let's see the compiling and packaging process from the source file:



The javac tool compiles the java source file into the class file.

The dx tool takes all the class files of your application and generates a single .dex file. It is a platform-specific tool.

The Android Assets Packaging Tool (aapt) handles the packaging process.

I Phone OS

iOS, which was previously called iPhone OS, is a mobile operating system developed by Apple Inc. Its first release was in 2007, which included iPhone and iPod Touch. iPad (1st Generation) was released in April 2010 and iPad Mini was released in November 2012.

Apple iOS features

Apple iOS includes the following features:

- Wi-Fi, Bluetooth and cellular connectivity, along with VPN support;
- Integrated search support, which enables simultaneous search through files, media, applications and email;
- Gesture recognition supports - for example, shaking the device to undo the most recent action;
- Push email;
- Safari mobile browser;
- Integrated front- and rear-facing cameras with video capabilities;
- Direct access to the Apple App Store and the iTunes catalog of music, podcasts, television shows and movies available to rent or purchase;
- Compatibility with Apple's cloud service, iCloud;
- Siri personal assistant;
- Cross-platform communications between Apple devices through AirDrop

	Android	IOs
Source model	Open source	Closed, with open source components.
OS family	Linux	OS X, UNIX
Initial release	September 23, 2008	July 29, 2007
Customizability	A lot. Can change almost anything.	Limited unless jailbroken
Developer	Google, Open Handset Alliance	Apple Inc.
Widgets	Yes	No, except in Notification Center
Available language(s)	100+ Languages	34 Languages
File transfer	Easier than iOS. Using USB port and Android File Transfer desktop app. Photos can be transferred via USB without apps.	More difficult. Media files can be transferred using iTunes desktop app. Photos can be transferred out via USB without apps.
Available on	Many phones and tablets. Major manufacturers are Samsung, Motorola, LG, HTC and Sony. Nexus and Pixel line of devices is pure Android, others bundle manufacturer software.	iPod Touch, iPhone, iPad, Apple TV (2nd and 3rd generation)
Calls and messaging	Google Hangouts. 3rd party apps like Facebook Messenger, WhatsApp, Google Duo and Skype all work on Android and iOS both.	iMessage, FaceTime (with other Apple devices only). 3rd party apps like Google Hangouts, Facebook Messenger, WhatsApp, Google Duo and Skype all work on Android and iOS both.
Internet browsing	Google Chrome (or Android Browser on older versions; other browsers are available)	Mobile Safari (Other browsers are available)

App store , Affordability and interface	Google Play – 1,000,000+ apps. Other app stores like Amazon and Getjar also distribute Android apps. (unconfirmed ".APKs")	Apple app store – 1,000,000+ apps
Video chat	Google Duo and other 3rd party apps	FaceTime (Apple devices only) and other 3rd party apps
Voice commands	Google Now, Google Assistant	Siri
Working state	Current	Current
Maps	Google Maps	Apple Maps (Google Maps also available via a separate app download)
Latest stable release and Updates	Android 8.0.0, Oreo (Aug 21, 2017)	11 (Sep 19, 2017)
Alternative app stores and side loading	Several alternative app stores other than the official Google Play Store. (e.g. Aptoide, Galaxy Apps)	Apple blocks 3rd party app stores. The phone needs to be jailbroken if you want to download apps from other stores.
Battery life and management	Many Android phone manufacturers equip their devices with large batteries with a longer life.	Apple batteries are generally not as big as the largest Android batteries. However, Apple is able to squeeze decent battery life via hardware/software optimizations.
Open source	Kernel, UI, and some standard apps	The iOS kernel is not open source but is based on the open-source Darwin OS.
File manager	Yes. (Stock Android File Manager included on devices running Android 7.1.1)	Not available

Photos & Videos backup	Apps available for automatic backup of photos and videos. Google Photos allows unlimited backup of photos. OneDrive, Amazon Photos and Dropbox are other alternatives.	Up to 5 GB of photos and videos can be automatically back up with iCloud. All other vendors like Google, Amazon, Dropbox, Flickr and Microsoft have auto-backup apps for both iOS and Android.
Security	Android software patches are available soonest to Nexus device users. Manufacturers tend to lag behind in pushing out these updates. So at any given time a vast majority of Android devices are not running updated fully patched software.	Most people will never encounter a problem with malware because they don't go outside the Play Store for apps. Apple's software updates support older iOS devices also.
Rooting, bootloaders, and jailbreaking	Access and complete control over your device is available and you can unlock the bootloader.	Complete control over your device is not available.
Cloud services	Native integration with Google cloud storage. 15GB free, $2/mo for 100GB, 1TB for $10. Apps available for Amazon Photos, OneDrive and Dropbox.	Native integration with iCloud. 5GB free, 50GB for $1/mo, 200GB for $3/mo, 1TB for $10/mo. Apps available for Google Drive and Google Photos, Amazon Photos, OneDrive and Dropbox.
Interface	Touch Screen	Touch Screen
Supported versions	Android 5.0 & later (Android 4.4 is also supported but with patches)	iOS 8 & later
First version	Android 1.0, Alpha	iOS 1.0

iOS Architecture

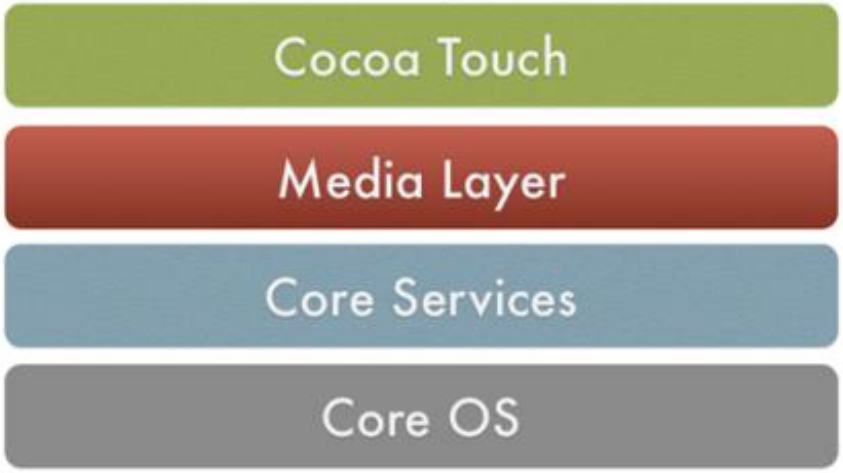
Architecture of iOs is a layered architecture.

At the uppermost level iOS works as an intermediary between the underlying hardware and the apps you make. Apps do not communicate to the underlying hardware directly.

Apps talk with the hardware through a collection of well-defined system interfaces.

Lower layers gives the basic services which all application relies on and higher level layer gives sophisticated graphics and interface related services.

Apple provides most of its system interfaces in special packages called frameworks.



Cocoa Touch Layer

The Cocoa Touch layer provides the following frameworks for iPhone app development:

Contents:

1. UIKit Framework (UIKit.framework)
2. Map Kit Framework (MapKit.framework)
3. Push Notification Service
4. Message UI Framework (MessageUI.framework)
5. Address Book UI Framework (AddressUI.framework)
6. Game Kit Framework (GameKit.framework)

UI Kit Framework (UIKit.framework)

UI Kit framework is a vast and feature rich Objective-C based programming interface. It is, without question, the framework with which you will spend most of your time working. Entire books could, and probably will, be written about the UIKit framework alone. Some of the key features of UIKit are as follows:

- User interface creation and management (text fields, buttons, labels, colors, fonts etc)
- Application lifecycle management
- Application event handling (e.g. touch screen user interaction)
- Cut, copy, and paste functionality
- Web and text content presentation and management
- Data handling
- Inter-application integration
- Push notification in conjunction with Push Notification Service
- Accessibility
- Accelerometer, battery, proximity sensor, camera and photo library interaction.

Map Kit Framework (Map Kit. framework)

If you have spent any appreciable time with an iPhone then the chances are you have needed to use the Maps application more than once, either to get a map of a specific area or to generate driving directions to get you to your intended destination. The Map Kit framework provides you with a programming interface that enables you to build map based capabilities into your own applications. This allows you to, amongst other things, display scrollable maps for any location, display the map corresponding to the current geographical location of the device and annotate the map in a variety of ways.

Push Notification Service

The Push Notification Service allows applications to notify users of an event even when the application is not currently running on the device. Since the introduction of this service it has most commonly been used by news based applications. Typically when there is breaking news the service will generate a message on the device with the news headline and provide the user the option to load the corresponding news app to read more details. This alert is typically accompanied by an audio alert and vibration of the device. This feature should be used sparingly to avoid annoying the user with frequent interruptions.

Message UI Framework (Message UI.framework)

The Message UI framework provides everything you need to allow users to compose and send email messages from within your application. In fact, the framework even provides the user interface elements through which the user enters the email addressing information and message content. Alternatively, this information can be pre-defined within your application and then displayed for the user to edit and approve prior to sending.

Address Book UI Framework (AddressUI.framework)

Given that a key function of the iPhone is as a communications device and digital assistant it should not come as too much of a surprise that an entire framework is dedicated to the integration of the address book data into your own applications. The primary purpose of the framework is to enable you to access, display, edit and enter contact information from the iPhone address book from within your own application.

Game Kit Framework (GameKit.framework)

The Game Kit framework provides peer-to-peer connectivity and voice communication between multiple devices and users allowing those running the same app to interact. When this feature was first introduced it was anticipated by Apple that it would primarily be used in multi-player games (hence the choice of name) but the possible applications for this feature clearly extend far beyond games development.

OS Media Layer:

The role of the Media layer is to provide the iPhone OS with audio, video, animation and graphics capabilities. As with the other layers comprising the iPhone OS stack, the Media layer comprises a number of frameworks that can be utilized when developing iPhone apps. In this section we will look at each one in turn.

Contents

7 Open Audio Library (OpenAL)

8 Media Player framework (MediaPlayer.framework)

Core Graphics Framework (CoreGraphics.framework)

The iPhone Core Graphics Framework (otherwise known as the Quartz 2D API) provides a lightweight two dimensional rendering engine. Features of this framework include PDF document creation and presentation, vector based drawing, transparent layers, path based drawing, anti-aliased rendering, color manipulation and management, image rendering and gradients. Those familiar with the Quartz 2D API running on MacOS X will be pleased to learn that the implementation of this API is the same on the iPhone.

Quartz Core Framework (QuartzCore.framework)

The purpose of the Quartz Core framework is to provide animation capabilities on the iPhone. It provides the foundation for the majority of the visual effects and animation used by the UIKit framework and provides an Objective-C based programming interface for creation of specialized animation within iPhone apps.

OpenGL ES framework (OpenGLES.framework)

For many years the industry standard for high performance 2D and 3D graphics drawing has been OpenGL. Originally developed by the now defunct Silicon Graphics, Inc (SGI) during the 1990s in the form of GL, the open version of this technology (OpenGL) is now under the care of a non-profit consortium comprising a number of major companies including Apple, Inc., Intel, Motorola and ARM Holdings.

OpenGL for Embedded Systems (ES) is a light weight version of the full OpenGL specification designed specifically for smaller devices such as the iPhone.

Version 3.0 of the iPhone OS supports both OpenGL ES 1.1 and 2.0 on certain iPhone models (such as the iPhone 3GS). Earlier versions of the iPhone OS and older models support only OpenGL ES version 1.1.

iPhone Audio Support

The iPhone OS is capable of supporting audio in AAC, Apple Lossless (ALAC), A-law, IMA/ADPCM, Linear PCM, μ-law, DVI/Intel IMA ADPCM, Microsoft GSM 6.10 and AES3-2003 formats through the support provided by the following frameworks.

AV Foundation framework (AVFoundation.framework)

An Objective-C based framework designed to allow the playback, recording and management of audio content.

Core Audio Frameworks (CoreAudio.framework, AudioToolbox.framework and AudioUnit.framework)

The frameworks that comprise Core Audio for the iPhone OS define supported audio types, playback and recording of audio files and streams and also provide access to the device's built-in audio processing units.

Open Audio Library (OpenAL)

OpenAL is a cross platform technology used to provide high-quality, 3D audio effects (also referred to as positional audio). Positional audio can be used in a variety of applications though is typically using to provide sound effects in games.

Media Player framework (MediaPlayer.framework)

The iPhone OS Media Player framework is able to play video in .mov, .mp4, .m4v, and .3gp formats at a variety of compression standards, resolutions and frame rates.

OS Core Services Layer:

The iPhone Core Services layer provides much of the foundation on which the above layers are built and consists of the following frameworks.

Contents

Address Book framework (AddressBook.framework)

The Address Book framework provides programmatic access to the iPhone Address Book contact database allowing applications to retrieve and modify contact entries.

Core Data Framework (CoreData.framework)

This framework is provided to ease the creation of data modeling and storage in Model-View-Controller (MVC) based applications. Use of the Core Data framework significantly reduces the amount of code that needs to be written to perform common tasks when working with structured data in an application.

Core Foundation Framework (CoreFoundation.framework)

The Core Foundation is a C-based Framework that provides basic functionality such as data types, string manipulation, raw block data management, URL manipulation, threads and run loops, date and times, basic XML manipulation and port and socket communication. Additional XML capabilities beyond those included with this framework are provided via the libXML2 library. Though this is a C-based interface, most of the capabilities of the Core Foundation framework are also available with Objective-C wrappers via the Foundation Framework.

Foundation Framework (Foundation.framework)

The Foundation framework is the standard Objective-C framework that will be familiar to those that have programmed in Objective-C on other platforms (most likely Mac OS X). Essentially, this consists of Objective-C wrappers around much of the C-based Core Foundation Framework.

Core Location Framework (CoreLocation.framework)

The Core Location framework allows you to obtain the current geographical location of the device (latitude and longitude) and compass readings from with your own applications. The method used by the device to provide coordinates will depend on the data available at the time the information is requested and the hardware support provided by the particular iPhone model on which the app is running (GPS and compass are only featured on recent models). This will either be based on GPS readings, WiFi network data or cell tower triangulation (or some combination of the three).

Store Kit Framework (StoreKit.framework)

The purpose of the Store Kit framework is to facilitate commerce transactions between your application and the Apple App Store. Prior to version 3.0 of the iPhone OS, it was only possible to charge a customer for an app at the point that they purchased it from the App Store. iPhone OS 3.0 introduced the concept of the "in app purchase" whereby the user can be given the option make additional payments from within the application. This might, for example, involve implementing a subscription model for an application, purchasing additional functionality or even buying a faster car for you to drive in a racing game.

SQLite library

Allows for a lightweight, SQL based database to be created and manipulated from within your iPhone application.

OS Core OS Layer:

The Core OS Layer is the bottom layer of the iPhone OS stack and sits directly on top of the device hardware. The layer provides a variety of services including low level networking, access to external accessories and the usual fundamental operating system services such as memory management, file system handling and threads.

Contents

CFNetwork Framework (CFNetwork.framework)

The CFNetwork framework provides a C-based interface to the TCP/IP networking protocol stack and low level access to BSD sockets. This enables application code to be written that works with HTTP, FTP and Domain Name servers and to establish secure and encrypted connections using Secure Sockets Layer (SSL) or Transport Layer Security (TLS).

External Accessory framework (ExternalAccessory.framework)

Provides the ability to interrogate and communicate with external accessories connected physically to the iPhone via the 30-pin dock connector or wirelessly via Bluetooth.

Security Framework (Security.framework)

The iPhone OS Security framework provides all the security interfaces you would expect to find on a device that can connect to external networks including certificates, public and private keys, trust policies, keychains, encryption, digests and Hash-based Message Authentication Code (HMAC).

System (LibSystem)

As we have previously mentioned, the iPhone OS is built upon a UNIX-like foundation. The System component of the Core OS Layer provides much the same functionality as

any other UNIX like operating system. This layer includes the operating system kernel (based on the Mach kernel developed by Carnegie Mellon University) and device drivers. The kernel is the foundation on which the entire iPhone OS is built and provides the low level interface to the underlying hardware. Amongst other things the kernel is responsible for memory allocation, process lifecycle management, input/output, inter-process communication, thread management, low level networking, file system access and thread management.

As an app developer your access to the System interfaces is restricted for security and stability reasons. Those interfaces that are available to you are contained in a C-based library called LibSystem. As with all other layers of the iPhone OS stack, these interfaces should be used only when you are absolutely certain there is no way to achieve the same objective using a framework located in a higher OS layer.

Event Handling

What is an Event?

Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page are the activities that causes an event to happen.

User interacts with the interface so as to execute specific tasks. Every action the user performs while interacting with the user interface triggers an event

Types of Event

The events can be broadly classified into two categories:

- **Foreground Events** - Those events which require the direct interaction of user. They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page etc.
- **Background Events** - Those events that require the interaction of end user are known as background events. Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events.

What is Event Handling?

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have the code which is known as event handler that is executed when an event occurs.

The concept of handling events is termed listening in Android.

Following are the three concepts related to Android Event Management,

1. **Event Listener** contains a single callback method. It is an interface in the View class.

2. **Event Handler** is the method that handles the event. The Event Listener calls the Event Handlers.

3. **Event Listener Registration** is a process, where an Event Handler gets registered with an Event Listener. Event Handler is called when the Event Listener fires the event.

Callback

A callback function is one which is passed as an argument to another function and is invoked after the completion of the parent function.

In other words callback is a piece of executable code that is passed as an argument to other code, which is expected to call back (execute) the argument at some convenient time.

(OR)

Callback is a function passed as a parameter in another function. The callback is then called from inside the function.

Listener - It is also known as event handler. Listener is responsible for generating response to an event. Listener waits until it receives an event. Once the event is received, the listener process the event and then returns.

Event Handler	**Event Listener & Description**
onClick()	OnClickListener() This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. You will use onClick() event handler to handle such event.
onLongClick()	OnLongClickListener()

	This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event.
onFocusChange()	OnFocusChangeListener() This is called when the widget looses its focus ie. user goes away from the view item. You will use onFocusChange() event handler to handle such event.
onKey()	OnFocusChangeListener() This is called when the user is focused on the item and presses or releases a hardware key on the device. You will use onKey() event handler to handle such event.
onTouch()	OnTouchListener() This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use onTouch() event handler to handle such event.
onMenuItemClick()	OnMenuItemClickListener() This is called when the user selects a menu item. You will use onMenuItemClick() event handler to handle such event.
onCreateContextMenu()	onCreateContextMenuItemListener() This is called when the context menu is being built(as the result of a sustained "long click)

Event based programming is programming in which the code is based on events, which are similar to broadcasts. For example, a "when mouse moved" event can trigger all scripts when the mouse is moved. Events have their own attributes, called **event attributes**. For example, *when mouse moved* can have the attributes *current mouse x position*, *previous mouse x position*, *distance moved*, etc.

Lightweight User Interface Toolkit (LWUIT) *is a Widget toolkit developed by Sun Microsystems to enable easier Java ME user interface development for existing devices, including not only traditional Java ME environments like mobile phones, but also TVs and set top boxes.*

Java Platform, Micro Edition or Java ME *is a computing platform for development and deployment of portable code for embedded and mobile devices (micro-controllers, sensors, gateways, mobile phones, personal digital assistants, TV set-top boxes, printers).[1] Java ME was formerly known as Java 2 Platform, Micro Edition or J2ME.*

What is UI Toolkit?

UI Toolkit is a set of routines and utility programs that gives you the tools you need to create user interfaces for your applications. Toolkit helps you provide a user interface that is easy to use and includes the following:

- Color
- Drop-down menus
- Cascading menus
- Pop-up help
- ActiveX controls
- Tabbed dialogs

The UI Toolkit environment establishes and maintains input, prompts, messages, menus, help, and other user interface details—so you don't have to write such code yourself. Toolkit utilities handle the user-interface details for you and ensure consistency throughout your application.

UI Toolkit is structured to support large applications. Its design assumes that the major functions of your application will be supported by individual routines, while the main routine will consist primarily of a small decision apparatus that makes the appropriate calls to these routines. These routines and programs interact with window library files that UI Toolkit creates when it processes the script files you have written.

Understanding Application Priority and Process States

Figure 3-3 shows the priority tree used to determine the order of application termination.

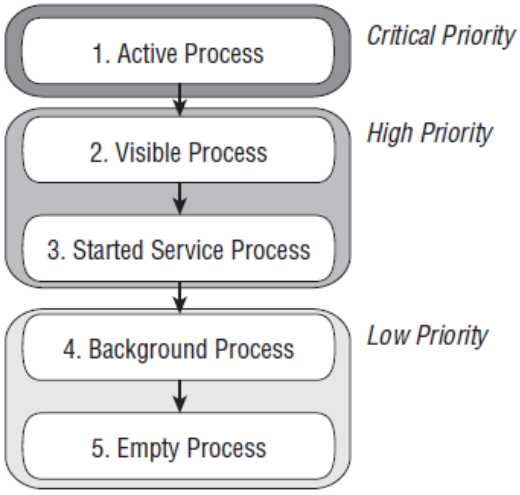


Figure 3-3

Active Processes: Active (foreground) processes are those hosting applications with components currently interacting with the user. These are the processes Android is trying to keep responsive by reclaiming resources. There are generally very few of these processes, and they will be killed only as a last resort.

Visible Processes: Visible, but inactive processes are those hosting "visible" Activities. As the name suggests, visible Activities are visible, but they aren't in the foreground or responding to user events. This happens when an Activity is only partially obscured (by a non-full-screen or transparent Activity). There are generally very few visible processes, and they'll only be killed in extreme circumstances to allow active processes to continue.

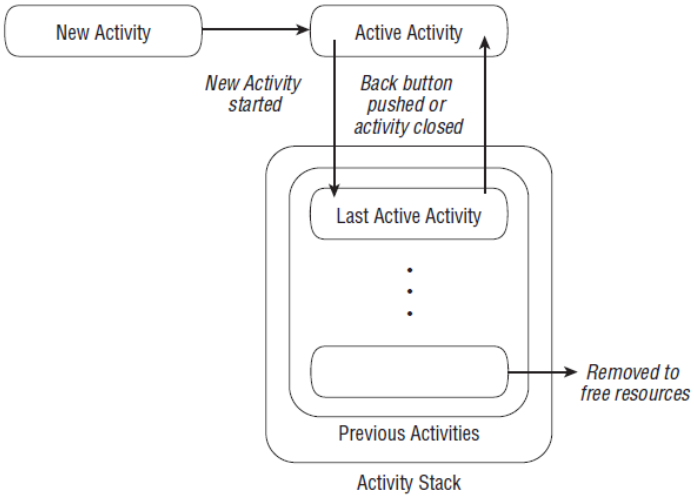
Started Service Processes: Processes hosting Services that have been started. Services support ongoing processing that should continue without a visible interface. Because Services don't interact directly with the user, they receive a slightly lower priority than visible Activities. **They are still considered to be foreground processes and won't be killed unless resources are needed for active or visible processes.**

Background Processes: Hosting Activities that aren't visible and that don't have any services that have been started are considered background processes. There will generally be a large number of background processes that **Android will kill using a last-seen-first-killed pattern to obtain resources for foreground processes.**

Empty Processes: To improve overall system performance, Android often retains applications in memory after they have reached the end of their lifetimes. Android maintains this cache to improve the start-up time of applications when they're re-launched. These processes are routinely killed as required.

Activity Stacks

The state of each Activity is determined by its position on the Activity stack, a last-in–first-out collection of all the currently running Activities. When a new Activity starts, the current foreground screen is moved to the top of the stack. If the user navigates back using the Back button, or the foreground Activity is closed, the next Activity on the stack moves up and becomes active. This process is illustrated here



Activity States

Active: When an Activity is at the top of the stack, it is the visible, focused, foreground activity that is receiving user input. Android will attempt to keep it alive at all costs, killing Activities further down the stack as needed, to ensure that it has the resources it needs. When another Activity becomes active, this one will be paused.

Paused: In some cases, your Activity will be visible but will not have focus; at this point, it's paused. This state is reached if a transparent or non-full-screen Activity is active in front of it. When paused, an Activity is treated as if it were active; however, it doesn't receive user input events. In extreme cases, Android will kill a paused Activity to recover resources for the active Activity. When an Activity becomes totally obscured, it becomes stopped.

Stopped: When an Activity isn't visible, it "stops." The Activity will remain in memory retaining all state and member information; however, it is now a prime candidate for execution when the system requires memory elsewhere. When an Activity is stopped, it's important to save data and the current UI state. Once an Activity has exited or closed, it becomes inactive.

Inactive: After an Activity has been killed, and before it's been launched, it's inactive. Inactive Activities have been removed from the Activity stack and need to be restarted before they can be displayed and used.

Android Activity Lifecycle

An activity is the single screen in android.

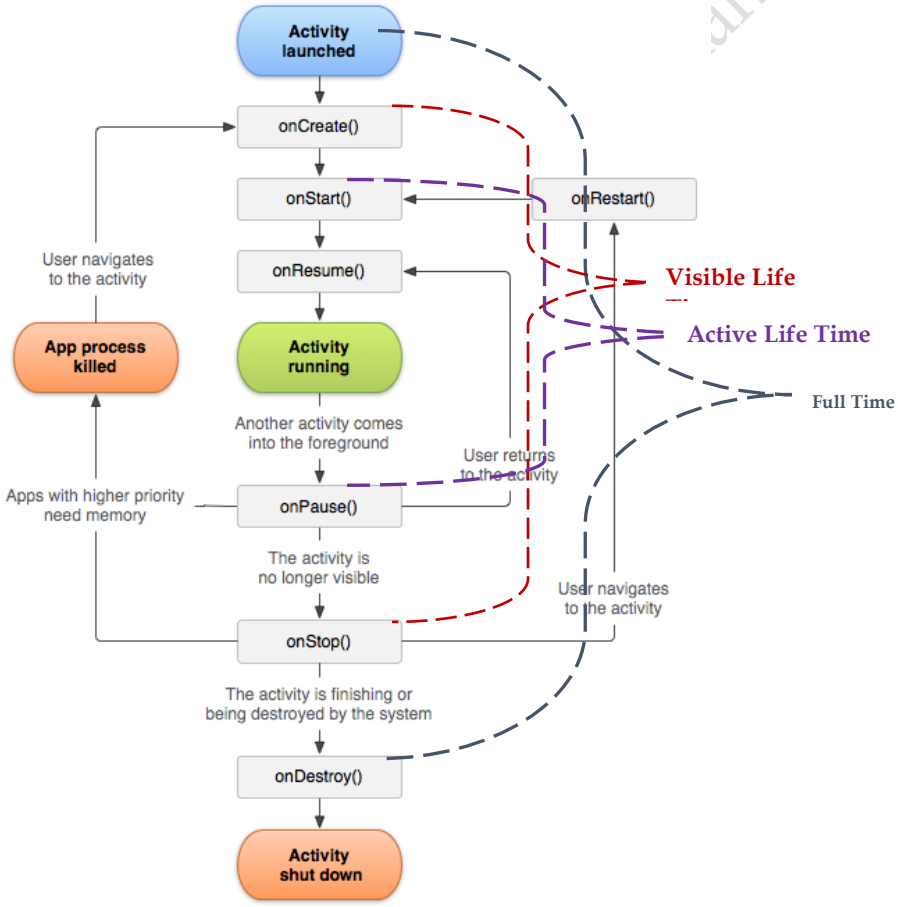
The 7 lifecycle method of Activity describes how activity will behave at 6 different states.

Activity have six states:

1) Created
2) Started
3) Resumed
4) Paused
5) Stopped
6) Destroyed

Activity lifecycle have seven methods:

1) onCreate()
2) onStart()
3) onResume()
4) onPause()
5) onStop()
6) onRestart()
7) onDestroy()



Android Activity Lifecycle methods

Let's see the 7 lifecycle methods of android activity.

Method	Description
onCreate	Called when activity is first created.
onStart	Called when activity is becoming visible to the user.
onResume	Called when activity will start interacting with the user.
onPause	Called when activity is not visible to the user.
onStop	Called when activity is no longer visible to the user.
onRestart	Called after your activity is stopped, prior to start.
onDestroy	Called before the activity is destroyed.

Intent

An Intent is an "intention" to perform an action;

An **Intent** is basically a message that is passed between **components** (such as **Activities**, **Services, Broadcast Receivers,** and **Content Providers**).

(or)

Intents are used as a message-passing mechanism that lets you declare your intention that an action be performed, usually with (or on) a particular piece of data.

- Intent is an intention to do something.
- Intent contains an action carrying some information.
- Intent is used to communicate between android components.
- Intent is used to communicate, share data between components.

Android intents are mainly used to:

- Start an Activity
- Start a Service
- Pass data in same application or different application
- Deliver a Broadcast Message

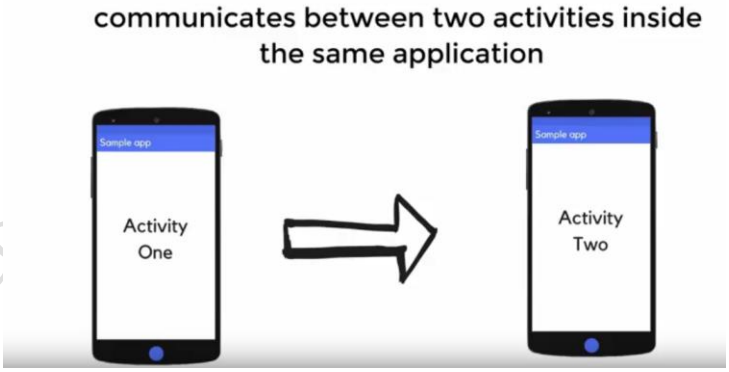
Any android application comprises one or more activities. In order to launch another activity from a particular activity we've to use a particular app component that android has called **Intent.** An Intent is basically an *intention* to do an action. It's a way to communicate between Android components (not just activities) to request an action from and by different components. It's like a message that Android listens for and react accordingly by identifying and invoking the appropriate app's appropriate component (like an Activity, Service, Content Provider, etc.) within that same application or some other app. If multiple apps are capable of responding to the message then Android provides the user with a list of those apps from which a choice can be made.

Using intents you can accomplish a lot of things like navigating from one activity to another, start an external app's activity so that the user can attain tasks like taking pictures via camera app, picking a contact using the contacts app, pinning location on maps using google maps, sending an email via Gmail, sharing status using WhatsApp or Facebook, etc.

There are two types of Intents: Explicit and Implicit.

Explicit Intent: It is used to call a specific component. When you know which component you want to launch and you do not want to give the user free control over which component to use. For example, you have an application that has 2 activities. Activity A and activity B. You want to launch activity B from activity A. In this case you define an explicit intent targeting activity B and then use it to directly call it.

"Communicates between two activities inside the same application."

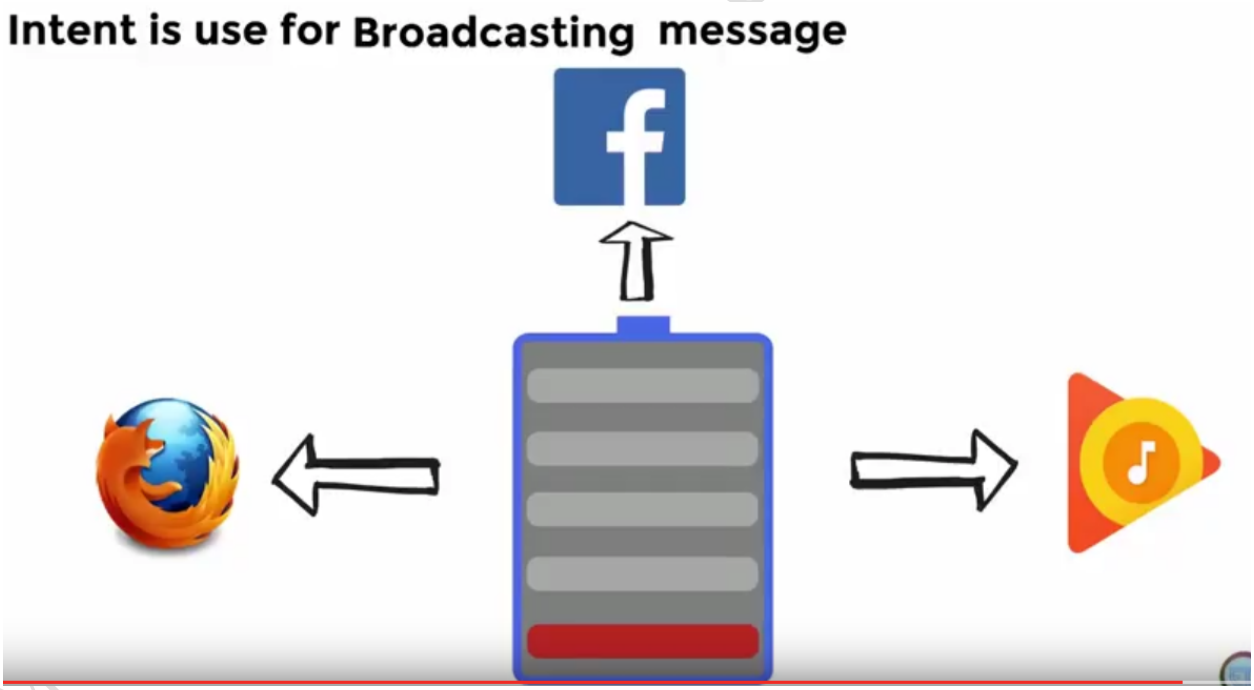


Implicit Intent: It is used when you have an idea of what you want to do, but you do not know which component should be launched. Or if you want to give the user an option to choose between a list of components to use. If these Intents are send to the Android system it searches for all components which are registered for the specific action and the data type. If only one component is found, Android starts the component directly.

"Communicates between two activities of different application."

Typically we use explicit intents to fire components from our own app whereas implicit intents to launch components from other third party apps.

Intents are used for Broadcasting Messages:



Intents can also be used to broadcast messages across the system. Any application can register a Broadcast Receiver to listen for, and react to, these broadcast Intents. This lets you create event-driven applications based on internal, system, or third-party application events.

Android uses broadcast Intents to announce system events, like changes in Internet connection status or battery charge levels. The native Android applications, such

as the phone dialer and SMS manager, simply register components that listen for specific broadcast Intents — such as "incoming phone call" or "SMS message received" — and react accordingly.

For Ex. Suppose your phone has a low battery then Your Android operating system will pass a message to all the application saying that Your battery is low and other applications will receive this message via the broadcast receivers, this communication between the OS and the application is carried out by **intents.**

Services

Android offers the Service class to create application components specifically to handle operations and functionality that should run invisibly, without a user interface.

A **service** is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed.

(Or)

A **service** is a component which runs in the background without direct interaction with the user. As the service has no user interface, it is not bound to the lifecycle of an activity.

By using Services, you can ensure that your applications continue to run and respond to events, even when they're not in active use.

Services run without a dedicated GUI, but, like Activities and Broadcast Receivers, they still execute in the main thread of the application's process. To help keep your applications responsive,

Unlike Activities, which present a rich graphical interface to users, Services run in the background updating your Content Providers, firing Intents, and triggering notifications.

They are the perfect way to perform regular processing or handle events even after your application's Activities are invisible, inactive, or have been closed.

With no visual interface, Services are started, stopped, and controlled from other application components including other Services, Activities, and Broadcast Receivers. If your application regularly, or continuously, performs actions that don't depend directly on user input, Services may be the answer.

Started Services receive higher priority than inactive or invisible Activities, making them less likely to be terminated by the run time's resource management. The only time Android will stop a Service prematurely is when it's the only way for a foreground Activity to gain required resources; if that happens, your Service will be restarted automatically when resources become available.

Applications that update regularly but only rarely or intermittently need user interaction are good candidates for implementation as Services. MP3 players and sports-score monitors are examples of applications that should continue to run and update without an interactive visual component (Activity) visible.

- There is nor **onPause() or onResume()** in Services as like in Activity.
- Services doesn't have UI.

Services can be of two types – Started and Bound

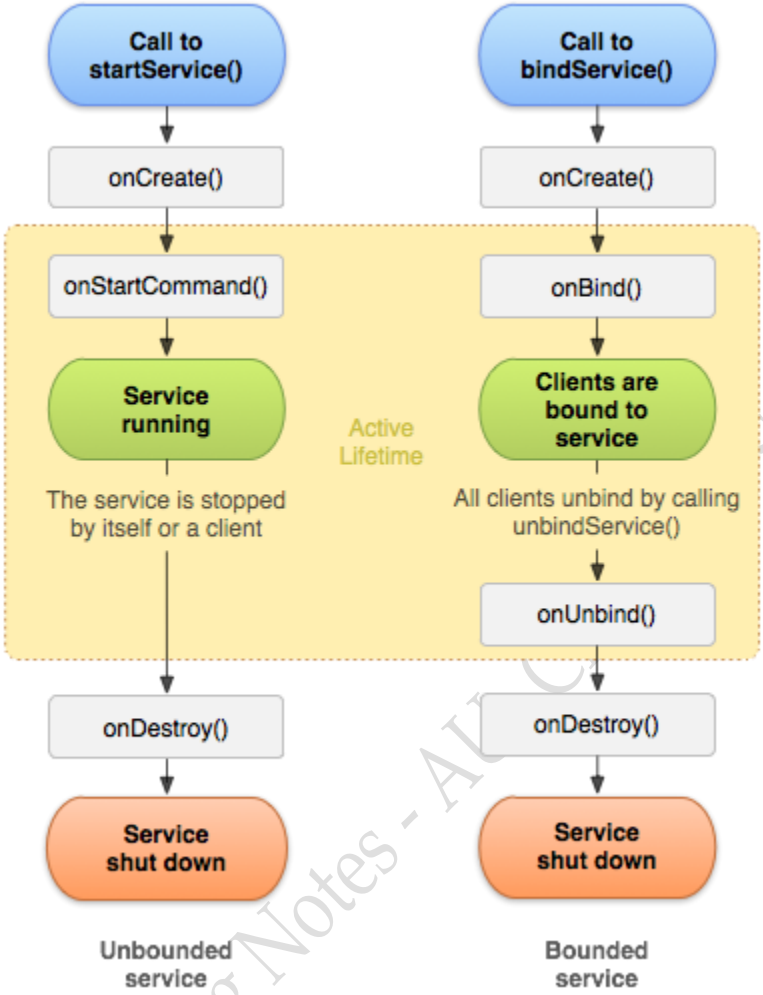
Started Service

Started services are those that are launched by other application components like an Activity or a Broadcast Receiver. They can run indefinitely in the background until stopped or destroyed by the system to free up resources.

Bound Services

A Bound Service is the server in a client- server interface. A Bound Service allows components (such as Activities) to bind to the service, send requests, receive responses and even perform inter-process communication (IPC).

While working with Android services, there comes a situation where we would want the service to communicate with an activity. To accomplish this task one has to bind a service to an activity, this type of service is called an android bound service. After a service is bound to an activity one can return the results back to the calling activity.



onBind()

The system calls this method when another component wants to bind with the service by calling *bindService()*. If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an *IBinder* object. You must always implement this method, but if you don't want to allow binding, then you should return *null*.

onUnbind()

The system calls this method when all clients have disconnected from a particular interface published by the service.

Started services cannot return results/values or interact with its starting component. Bound services on the other hand can send data to the launching component (client). So for example a bound service might be playing an audio file and sending data regarding audio start/pause/stop and the time elapsed to the launching Activity component so that the UI can be updated accordingly.

Understanding Started and Bound Service by background music example

Suppose, I want to play music in the background, so call start Service () method. But I want to get information of the current song being played, I will bind the service that provides information about the current song.

Running a Service in the Foreground

A Foreground Service is one where the user is actively aware of it putting it on high priority hence won't be killed when the system is low on memory. It must provide a notification for the status bar which cannot be dismissed unless the Service is stopped or removed from the foreground. A good example of such a notification is a music player that shows the current song and other action buttons like play/pause, next, previous, etc. in a notification in the status bar. VOIP calls or file download apps could also start a foreground Service and show similar notifications.

Stopping a service

You stop a service via the **stopService()** method. No matter how frequently you called the **startService(intent)** method, one call to the **stopService()** method stops the service.

A service can terminate itself by calling the **stopSelf()** method. This is typically done if the service finishes its work.

Toasts

A toast provides simple feedback about an operation in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive. For example, navigating away from an email before you send it triggers a "Draft saved" toast to let you know that you can continue editing later. Toasts automatically disappear after a timeout.

Android offers several techniques for applications to communicate with users without an Activity. You'll learn how to use Notifications and Toasts to alert and update users without interrupting the active application.

Toasts are a transient, non-modal dialog-box mechanism used to display information to users without stealing focus from the active application. You'll learn to display Toasts from any application component to send unobtrusive on-screen messages to your users.

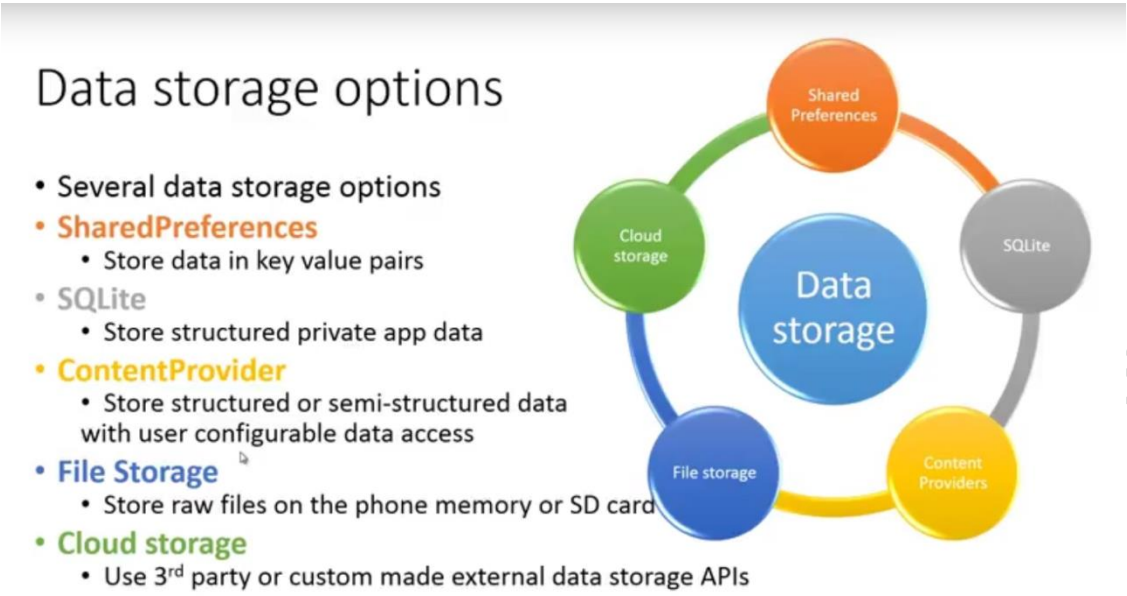
Where Toasts are silent and transient, *Notifications* represent a more robust mechanism for alerting users. In many cases, when the user isn't actively using the mobile phone it sits silent and unwatched in a pocket or on a desk until it rings, vibrates, or flashes. Should a user miss these alerts, status bar icons are used to indicate that an event has occurred. All these attention-grabbing antics are available to your Android application through Notifications.

Storing and Retrieving Data

Saving and loading data is an essential requirement for most applications. At a minimum, Activities should save their User Interface (UI) state each time they move out of the foreground.

Android provides the SQLite database library. The SQLite database offers a powerful native SQL database over which you have total control.

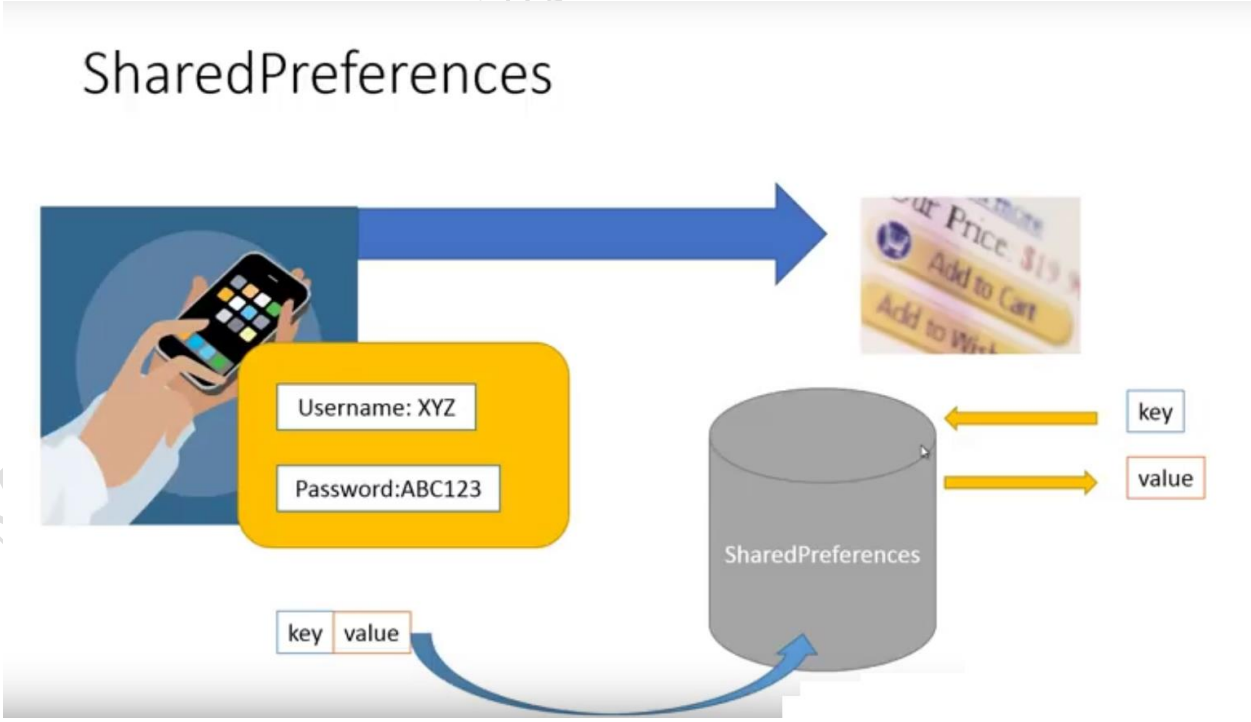
Android provides several options for you to save persistent application data. The solution you choose depends on your specific needs, such as whether the data should be private to your application or accessible to other applications (and the user) and how much space your data requires.



Store private primitive data in key-value pairs.

Shared Preferences

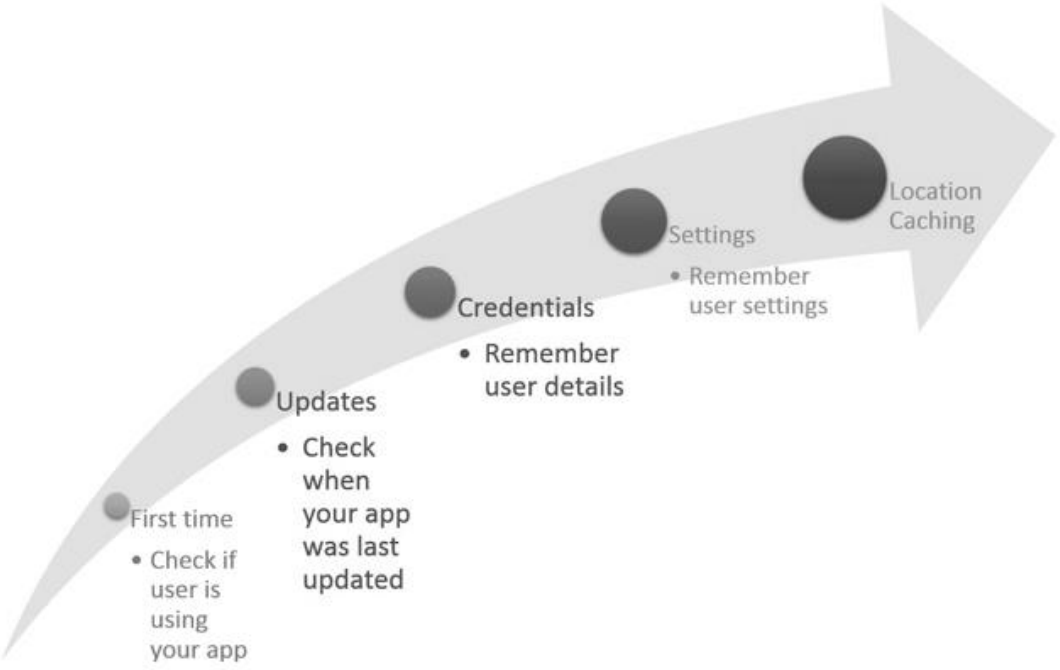
When storing the UI state, user preferences, or application settings, you want a lightweight mechanism to store a known set of values. Shared Preferences let you save groups of key/value pairs of primitive data as named preferences.



- SharedPreferences is used by apps to save data in name-values pairs, like a Bundle
- Data is stored in XML file in the directory data/data/<package-name>/shared-prefs folder
- Store data such as username, password, theme settings, other application settings
- SharedPreferences only allows you to save primitive data types (that is, booleans, floats, longs, ints, and strings)

key	value
firstName	Bugs
lastName	Bunny
location	Earth

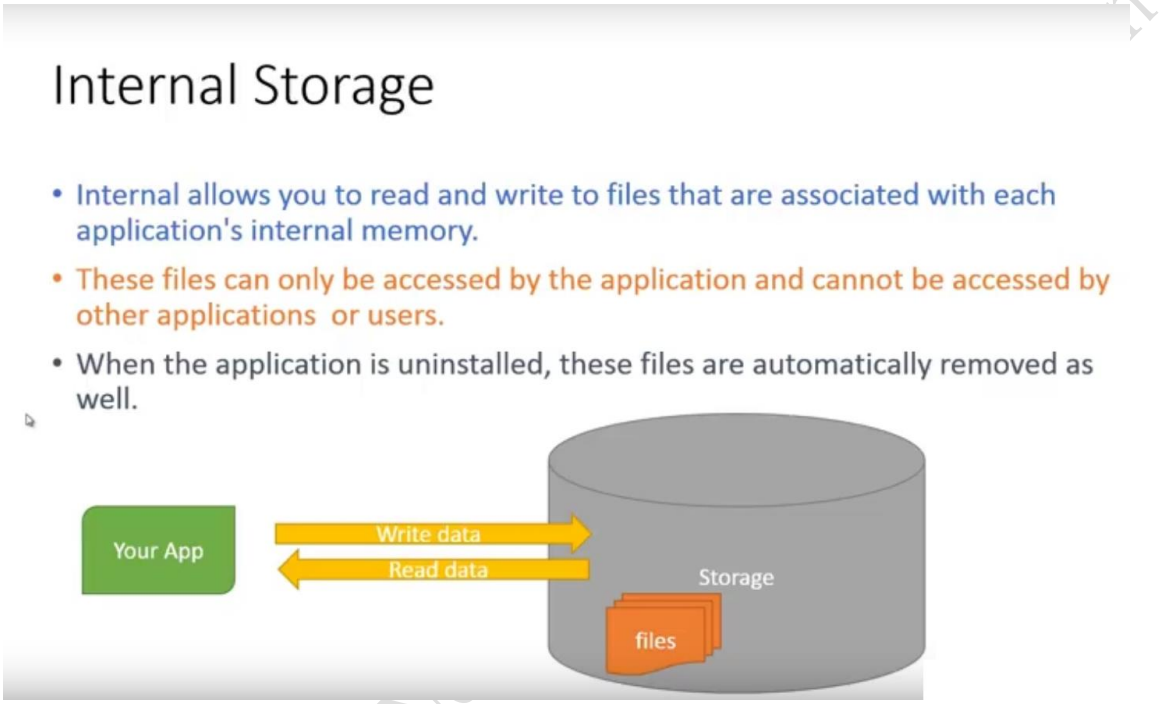
Some uses of SharedPreferences



Internal Storage

Store private data on the device memory.

You can save files directly on the device's internal storage. By default, files saved to the internal storage are private to your application and other applications cannot access them (nor can the user). When the user uninstalls your application, these files are removed.



SharedPreferences vs Internal/External storage

- **SharedPrefences**
- Storing data internally in XML files.
- More efficient due to less read/write overhead.
- Only primitive types and at most, arrays of strings can be stored

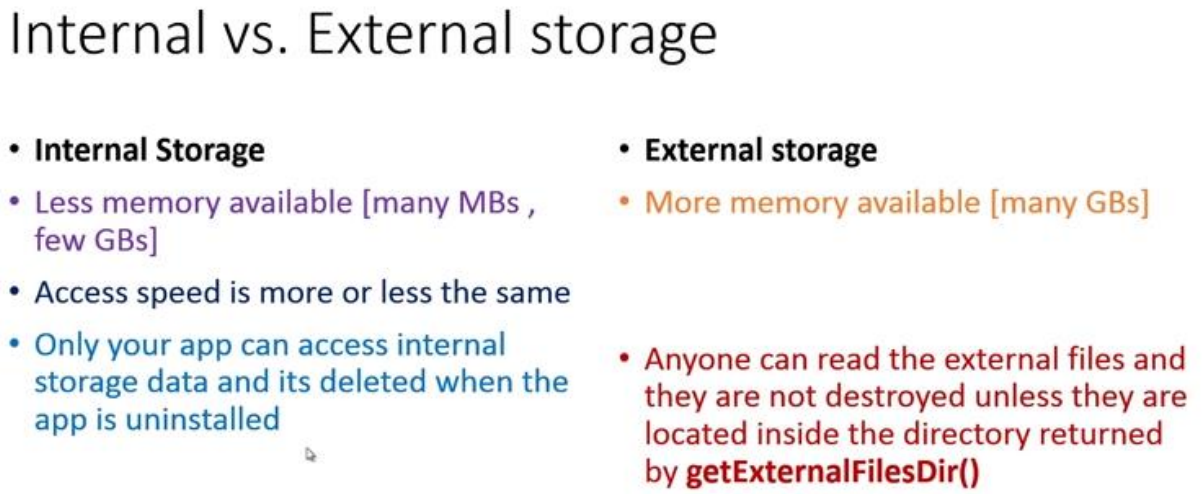
- **Internal/External storage**
- Storing data and files to the phone's external **Secure Digital (SD)** card.
- Less Efficient
- Store complex data, media, images

External Storage

Store public data on the shared external storage.

Every Android-compatible device supports a shared "external storage" that you can use to save files. This can be a removable storage media (such as an SD card) or an internal (non-removable) storage. Files saved to the external storage are world-readable and can be modified by the user when they enable USB mass storage to transfer files on a computer.

Caution: External storage can become unavailable if the user mounts the external storage on a computer or removes the media, and there's no security enforced upon files you save to the external storage. All applications can read and write files placed on the external storage and the user can remove them.



SQ Lite Databases:

Store structured data in a private database.

Android provides full support for SQLite databases. Any databases you create will be accessible by name to any class in the application, but not outside the application.

When managed, structured data is the best approach; Android offers the SQ Lite relational database library. Every application can create its own databases over which it has total control.

What is SQLite?

SQLite is an Open Source database. SQLite supports standard relational database features like SQL syntax, transactions and prepared statements. The database requires

limited memory at runtime (approx. 250 KByte) which makes it a good candidate from being embedded into other runtimes.

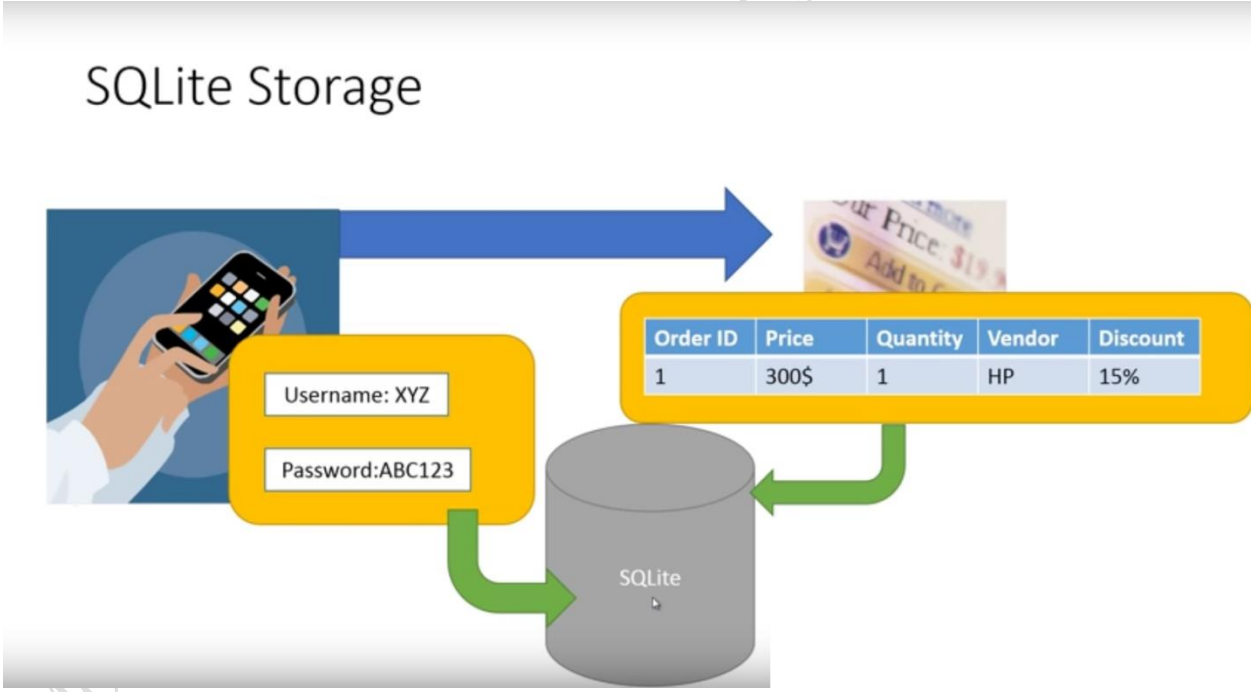
SQLite is embedded into every Android device. Using a SQLite database in Android does not require a setup procedure or administration of the database.

You only have to define the SQL statements for creating and updating the database. Afterwards the database is automatically managed for you by the Android platform.

(Or)

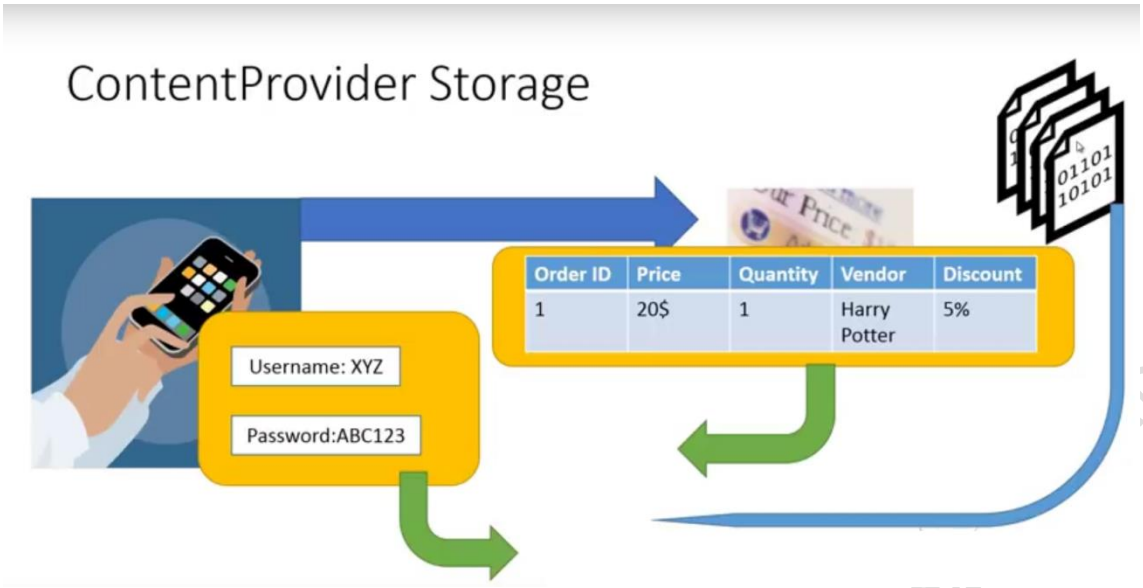
SQLite is an **open-source relational database** i.e. used to perform database operations on android devices such as storing, manipulating or retrieving persistent data from the database.

It is embedded in android by default. So, there is no need to perform any database setup or administration task.



Content Provider Storage:

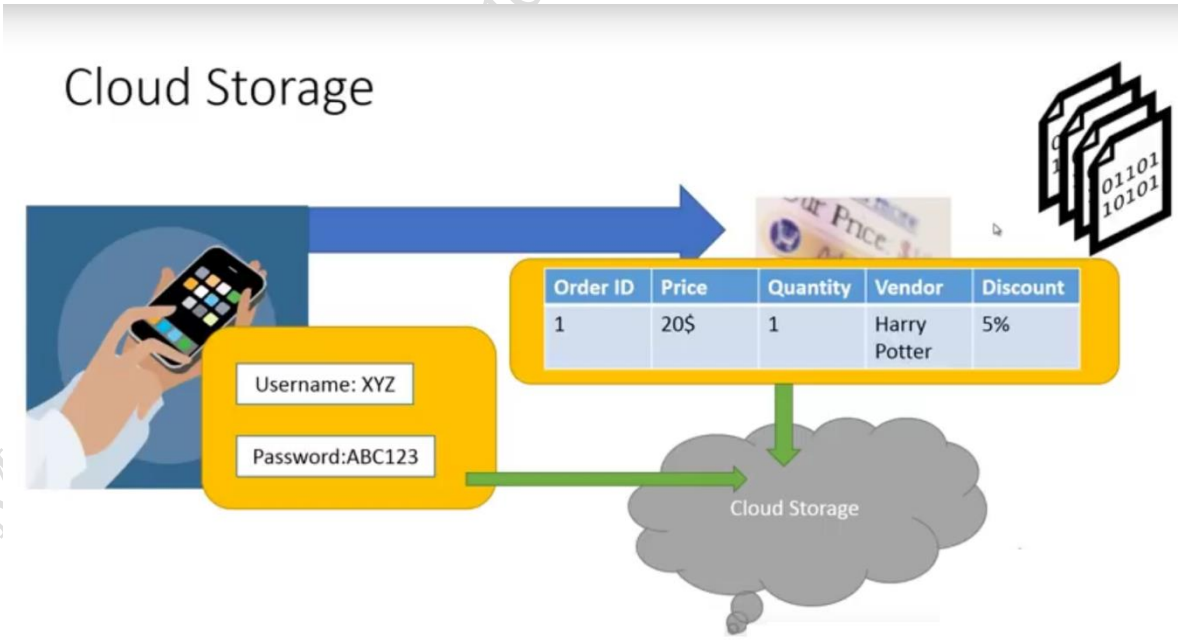
Content Providers Rather than a storage mechanism in their own right, Content Providers let you expose a well-defined interface for using and sharing private data. You can control access to Content Providers using the standard permission system.



Network Connection:

Store data on the web with your own network server.

Android provides a way for you to expose even your private data to other applications with a content provider. A content provider is an optional component that exposes read/write access to your application data, subject to whatever restrictions you want to impose.



Location based Services (LBS)

The services that let you find the device's current location. They include technologies like GPS and Google's cell-based location technology. You can specify which location-sensing technology to use explicitly by name, or implicitly by defining a set of criteria in terms of accuracy, cost, and other requirements.

Maps and location-based services use latitude and longitude to pinpoint geographic locations, but your users are more likely to think in terms of an address. Android provides a Geocoder that supports forward and reverse geocoding. Using the Geocoder, you can convert back and forth between latitude/longitude values and real-world addresses.

Used together, the mapping, geocoding, and location-based services provide a powerful toolkit for incorporating your phone's native mobility into your mobile applications.

Using Location-Based Services

- Find and track the device location.
- Create proximity alerts.
- Turn geographical locations into street addresses and vice versa.
- Create and customize map-based Activities using Map View and Map Activity.

Location-based services (LBS) is an umbrella term used to describe the different technologies used to find the device's current location. The two main LBS elements are:

- **Location Manager:** Provides hooks to the location-based services.
- **Location Providers:** Each of which represents a different location-finding technology used to determine the device's current location.

Using the Location Manager, you can:

- Obtain your current location.
- Track movement.
- Set proximity alerts for detecting movement into and out of a specified area.

Geocoder

Geocoding lets you translate between street addresses and longitude/latitude map coordinates. This can give you a recognizable context for the locations and coordinates used in location-based services and map-based Activities.

The Geocoder class provides access to two geocoding functions:

- Forward Geocoding finds the latitude and longitude of an address.
- Reverse Geocoding Finds the street address for a given latitude and longitude.

Packaging an Android Application: The .apk File

Android provides an application called apk builder for generating installable Android application files, which have the extension **.apk**. An .**apk** file is in ZIP file format, just like many other Java-oriented application formats, and contains the application manifest, compiled application classes, and application resources. Android provides the utility **aapt (*stands for* *Android Asset Packaging Tool*. *This tool is part of the SDK (and build system) and allows you to view, create, and update Zip-compatible archives (zip, jar, apk). It can also compile resources into binary assets.)*** for packaging the files that make up an **.apk** file, but developers typically prefer to allow their development environment to use this utility to build their applications for them. Most users simply rely on their IDE to build their **.apk**.

Once a developer has created an .apk file, he can choose to make it available for installation onto a device in one of several ways:

- Using the .**adb** (Android Debug Bridge is a versatile command-line tool that lets you communicate with a device (an emulator or a connected Android device). interface directory, or more commonly by using an IDE (integrated development environment)
- Using an SD card
- Making the file available on a web server
- Uploading the file to the Android Market, and then selecting Install

Placing an Application for Distribution in the Android Market

Putting an application on the Android Market is remarkably easy. The only prerequisite is that you have a Google account such as a Gmail account. A $25 credit card transaction and some information about yourself are all you need to start uploading applications to the Android Market. Charging for applications and getting paid takes only slightly more information and effort—you don't even need a website or a corporate entity.

API – Application program interface

A*pplication* ***p****rogram* ***i****nterface* (**API**) is a set of routines, protocols, and tools for building software applications. An API specifies how software components should interact. Additionally, APIs are used when programming graphical user interface (GUI) components. A good API makes it easier to develop a program by providing all the building blocks.

Introducing Notifications

Notifications are a way for your applications to alert users, without using an Activity. Notifications are handled by the Notification Manger, and currently include the ability to:

- Create a new status bar icon.
- Display additional information (and launch an Intent) in the extended status bar window.
- Flash the lights/LEDs.
- Vibrate the phone.
- Sound audible alerts (ringtones, media store sounds).

Notifications are the preferred way for invisible application components (Broadcast Receivers, Services, and inactive Activities) to alert users that events have occurred that require attention.

As a User Interface metaphor, Notifications are particularly well suited to mobile devices. It's likely that your users will have their phones with them at all times but quite unlikely that they will be paying attention to them, or your application, at any given time. Generally, users will have several applications open in the background, and they won't be paying attention to any of them. In this environment, it's important that your applications be able to alert users when specific events occur that require their attention.

Notifications can be persisted through insistent repetition, or (more commonly) by using an icon on the status bar. Status bar icons can be updated regularly or expanded to show additional information using the expanded status bar window shown in Figure.

Introducing the Notification Manager

The ***Notification Manager*** is a system Service used to handle Notifications. Get a reference to it using the ***getSystemService*** method, as shown in the snippet below:

```
String svcName = Context.NOTIFICATION_SERVICE;

NotificationManager notificationManager;

notificationManager = (NotificationManager)getSystemService(svcName);
```

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

Triggering Notifications

To fire a Notification, pass it in to the notify method on the Notification Manager along with an integer reference ID, as shown in the following snippet:

```
int notificationRef = 1;

notificationManager.notify(notificationRef, notification);
```

To update a Notification that's already been fired, re-trigger, passing the same reference ID. You can pass in either the same Notification object or an entirely new one. As long as the ID values are the same, the new Notification will be used to replace the status icon and extended status window details.

You also use the reference ID to cancel Notifications by calling the cancel method on the Notification

Manager, as shown below:

```
notificationManager.cancel(notificationRef);
```

Canceling a Notification removes its status bar icon and clears it from the extended status window.

Advanced Notification Techniques

In the following sections, you'll learn to enhance Notifications to provide additional alerting through Hardware, in particular, by making the device ring, flash, and vibrate.

Making Sounds

Using an audio alert to notify the user of a device event (like incoming calls) is a technique that predates the mobile, and has stood the test of time. Most native phone events from incoming calls to new messages and low battery are announced by an audible ringtone.

Android lets you play any audio file on the phone as a Notification by assigning a location URI to the sound property, as shown in the snippet below:

notification.sound = ringURI;

Vibrating the Phone

You can use the phone's vibration function to execute a vibration pattern specific to your Notification. Android lets you control the pattern of a vibration; you can use vibration to convey information as well as get the user's attention.

To set a vibration pattern, assign an array of longs to the Notification's vibrate property. Construct the array so that every alternate number is the length of time (in milliseconds) to vibrate or pause, respectively.

Before you can use vibration in your application, you need to be granted permission. Add a uses-permission to your application to request access to the device vibration using the following code snippet:

<uses-permission android:name="android.permission.VIBRATE"/>

The following example shows how to modify a Notification to vibrate in a repeating pattern of 1 second on, 1 second off, for 5 seconds total.

long[] vibrate = new long[] { 1000, 1000, 1000, 1000, 1000 };

notification.vibrate = vibrate;

Flashing the Lights

Notifications also include properties to configure the color and flash frequency of the device's LED.

The **ledARGB** property can be used to set the LED's color, while the **ledOffMS and ledOnMS** properties let you set the frequency and pattern of the flashing LED. You can turn the LED on by setting the ledOnMS property to 1 and the ledOffMS property to 0, or turn it off by setting both properties to 0.

Once you have configured the LED settings, you must also add the FLAG_SHOW_LIGHTS flag to the Notification's flags property.

The following code snippet shows how to turn on the red device LED:

```
notification.ledARGB = Color.RED;
notification.ledOffMS = 0;
notification.ledOnMS = 1;
notification.flags = notification.flags | Notification.FLAG_SHOW_LIGHTS;
```

Controlling the color and flash frequency is another opportunity to pass additional information to users.

Alarms

Alarms are an application independent way of firing Intents at predetermined times.

Alarms are set outside the scope of your applications, so they can be used to trigger application events or actions even after your application has been closed. They can be particularly powerful in combination with Broadcast Receivers, allowing you to set Alarms that launch applications or perform actions without applications needing to be open and active until they're required.

Alarms in Android remain active while the device is in sleep mode and can optionally be set to wake the device; however, all Alarms are canceled whenever the device is rebooted.

Alarm operations are handled through the ***AlarmManager,*** a system Service accessed via ***getSystemService*** as shown below:

```
AlarmManager alarms =
(AlarmManager)getSystemService(Context.ALARM_SERVICE);
```

To create a new Alarm, use the **set method** and specify an alarm type, trigger time, and a Pending Intent to fire when the Alarm triggers. If the Alarm you set occurs in the past, it will be triggered immediately.

Choose an alarm type

One of the first considerations in using a repeating alarm is what its type should be.

There are two general clock types for alarms: **"elapsed real time"** and **"real time clock" (RTC).** Elapsed real time uses the "time since system boot" as a reference, and real time clock uses UTC (wall clock) time. This means that elapsed real time is suited to setting an alarm based on the passage of time (for example, an alarm that fires every 30

seconds) since it isn't affected by time zone/locale. The real time clock type is better suited for alarms that are dependent on current locale.

Both types have a "wakeup" version, which says to wake up the device's CPU if the screen is off. This ensures that the alarm will fire at the scheduled time. This is useful if your app has a time dependency for example, if it has a limited window to perform a particular operation. If you don't use the wakeup version of your alarm type, then all the repeating alarms will fire when your device is next awake.

If you simply need your alarm to fire at a particular interval (for example, every half hour), use one of the elapsed real time types. In general, this is the better choice.

If you need your alarm to fire at a particular time of day, then choose one of the clock-based real time clock types. Note, however, that this approach can have some drawbacks the app may not translate well to other locales, and if the user changes the device's time setting, it could cause unexpected behavior in your app. Using a real time clock alarm type also does not scale well, as discussed above. We recommend that you use a "elapsed real time" alarm if you can.

There are four alarm types available. Your selection will determine if the time value passed in the ***set method*** represents a specific time or an elapsed wait:

- **RTC_WAKEUP** Wakes up the device to fire the Intent at the clock time specified when setting the Alarm.
- **RTC** Will fire the Intent at an explicit time, but will not wake the device.
- **ELAPSED_REALTIME** The Intent will be fired based on the amount of time elapsed since the device was booted, but will not wake the device. The elapsed time includes any period of time the device was asleep. Note that the time elapsed is since it was last booted.
- **ELAPSED_REALTIME_WAKEUP** Will wake up the device if necessary and fire the Intent after a specified length of time has passed since the device was booted.

Android Telephony

The telephony APIs let your applications access the underlying telephone hardware, making it possible to create your own dialer or integrate call handling and phone state monitoring into your applications.

Making Phone Calls

The best practice is to use Intents to launch a dialer application to initiate new phone calls. There aretwo Intent actions you can use to dial a number; in both cases, you should specify the number to dial using the ***tel:*** schema as the data component of the **Intent**:

- ***Intent.ACTION_CALL*** Automatically initiates the call, displaying the in-call application. You

should only use this action if you are replacing the native dialer, otherwise you should use the ACTION_DIAL action as described below. Your application must have the CALL_PHONE permission granted to broadcast this action.

- **Intent.ACTION_DIAL** Rather than dial the number immediately, this action starts a dialer

application, passing in the specified number but allowing the dialer application to manage the call initialization (the default dialer asks the user to explicitly initiate the call). This action doesn't require any permissions and is the standard way applications should initiate calls.

The following skeleton code shows the basic technique for dialing a number:

Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:1234567"));

startActivity(intent);

Monitoring Phone Calls

One of the most popular reasons for monitoring phone state changes is to detect, and react to, incoming and outgoing phone calls.

Calls can be detected through changes in the phone's call state. Override the **onCallState** Changed method in a Phone State Listener implementation, and register it as shown below to receive notifications when the call state changes:

```
PhoneStateListener callStateListener = new PhoneStateListener() {
public void onCallStateChanged(int state, String incomingNumber) {
// TODO React to incoming call.
}
};
telephonyManager.listen(callStateListener,
PhoneStateListener.LISTEN_CALL_STATE);
```

The **onCallStateChanged** handler receives the phone number associated with incoming calls, and the state parameter represents the current call state as one of the following three values:

- TelephonyManager.CALL_STATE_IDLE When the phone is neither ringing nor in a call
- TelephonyManager.CALL_STATE_RINGING When the phone is ringing
- TelephonyManager.CALL_STATE_OFFHOOK If the phone is currently on a call

Tracking Cell Location Changes

You can get notifications whenever the current cell location changes by overriding

onCellLocationChanged on a Phone State Listener implementation. Before you can register to listen for cell location changes, you need to add the ACCESS_COARSE_LOCATION permission to your

application manifest.

<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>

The onCellLocationChanged handler receives a CellLocation object that includes methods for extracting the **cell ID (getCid)** and the **current LAC (getLac)**.

The following code snippet shows how to implement a Phone State Listener to monitor cell location changes, displaying a Toast that includes the new location's cell ID:

```
PhoneStateListener cellLocationListener = new PhoneStateListener()
{
public void onCellLocationChanged(CellLocation location)
{
GsmCellLocation gsmLocation = (GsmCellLocation)location;
Toast.makeText(getApplicationContext()),
String.valueOf(gsmLocation.getCid()),
Toast.LENGTH_LONG).show();
}
};
telephonyManager.listen(cellLocationListener,
PhoneStateListener.LISTEN_CELL_LOCATION);
```

Tracking Service Changes

The onServiceStateChanged handler tracks the service details for the device's cell service. Use the ServiceState parameter to find details of the current service state.

The getState method on the Service State object returns the current service state as one of:

- ServiceState.STATE_IN_SERVICE Normal phone service is available.
- ServiceState.STATE_EMERGENCY_ONLY Phone service is available but only for emergency calls.
- ServiceState.STATE_OUT_OF_SERVICE No cell phone service is currently available.
- ServiceState.STATE_POWER_OFF The phone radio is turned off (usually when airplane mode is enabled).

A series of getOperator* methods is available to retrieve details on the operator supplying the cell phone service, while getRoaming tells you if the device is currently using a roaming profile.

The following example shows how to register for service state changes and displays a Toast showing the operator name of the current phone service:

```
PhoneStateListener serviceStateListener = new PhoneStateListener()
{
public void onServiceStateChanged(ServiceState serviceState)
 {
if (serviceState.getState() == ServiceState.STATE_IN_SERVICE)
 {
String toastText = serviceState.getOperatorAlphaLong();
Toast.makeText(getApplicationContext(), toastText, Toast.LENGTH_SHORT);
}
}
};
telephonyManager.listen(serviceStateListener,
PhoneStateListener.LISTEN_SERVICE_STATE);
```

Multimedia

The only modern technology that can compete with mobile phones for ubiquity is the portable digital

Media player. As a result, the multimedia capabilities of portable devices are a significant consideration for many consumers.

Android's open platform- and provider-agnostic philosophy ensures that it offers a multimedia library capable of playing and recording a wide range of media formats, both locally and streamed.

Android exposes this library to your applications, providing comprehensive multimedia functionality including recording and playback of audio, video, and still-image media stored locally, within an application, or streamed over a data connection.

At the time of print, Android supported the following multimedia formats:

- JPEG
- PNG
- OGG
- Mpeg 4
- 3GP
- MP3
- Bitmap

Playing Media Resources

Multimedia playback in Android is handled by the ***MediaPlayer*** class. You can play back media stored as application resources, local files, or from a network URI.

To play a media resource, create a new Media Player instance, and assign it a media source to play using the ***setDataSource*** method. Before you can start playback, you need to call prepare, as shown in the following code snippet:

```
String MEDIA_FILE_PATH = Settings.System.DEFAULT_RINGTONE_URI.toString();

MediaPlayer mpFile = new MediaPlayer();

try {
```

mpFile.setDataSource(MEDIA_FILE_PATH);

mpFile.prepare();

mpFile.start();

}

catch (IllegalArgumentException e) {}

catch (IllegalStateException e) {}

catch (IOException e) {}

Alternatively, the static create methods work as shortcuts, accepting media resources as a parameter and preparing them for playback, as shown in the following example, which plays back an application resource:

MediaPlayer mpRes = MediaPlayer.create(context, R.raw.my_sound);

Note that if you use a create method to generate your MediaPlayer object, prepare is called for you. Once a Media Player is prepared, call start as shown below to begin playback of the associated media resource.

mpRes.start();

mpFile.start();

The Android Emulator simulates audio playback using the audio output of your development platform.

The Media Player includes stop, pause, and seek methods to control playback, as well as methods to find the duration, position, and image size of the associated media.

To loop or repeat playback, use the **setLooping** method. *When playing video resources,* **getFrame** *will take a screen grab of video media at the specified frame and return a bitmap resource.*

Once you've finished with the Media Player, be sure to call release to free the associated resources, as shown below:

mpRes.release();

mpFile.release();

Since Android only supports a limited number of simultaneous Media Player objects, not releasing them can cause runtime exceptions.

Using the Camera

The popularity of digital cameras (particularly within phone handsets) has caused their prices to drop just as their size has shrunk dramatically. It's now becoming difficult to even find a mobile phone without a camera, and Android devices are unlikely to be exceptions.

To access the camera hardware, you need to add the CAMERA permission to your application manifest, as shown here:

<uses-permission android:name="android.permission.CAMERA"/>

This grants access to the Camera Service. The Camera class lets you adjust camera settings, take pictures, and manipulate streaming camera previews.

To access the Camera Service, use the static open method on the Camera class. When your application has finished with the camera, remember to relinquish your hold on the Service by calling release following the simple use pattern shown in the code snippet below:

```
Camera camera = Camera.open();
[ ... Do things with the camera ... ]
camera.release();
```

Memory Management

Memory is an important resource for any computing system. Furthermore, when considering programming mobile devices memory is a critical resource, because in an attempt to keep the cost of the device low, manufacturers include only a restricted amount of it in devices although all the running programs are competing for it. Moreover, in addition to forming a considerable cost factor, memory chips also consume some power, the amount of which depends on the amount of memory included in the device.

While the stack and the heap are in principle just memory areas. Additional considerations should be paid to the fact that when using the heap, sharing of data is easy and natural, whereas with stack-based variables one should use references with care, if at all, because as the execution proceeds the stack increases and decreases, and may overwrite referred data.

Stack

As a rule of thumb, transient objects, i.e., those that live only a limited period of time, are to be stored in the stack. The idea is that if an object lives a short period of time in any case, it is easier to allocate it in stack, because the object will be deallocated automatically as the execution advances. Furthermore, using stack memory is usually already reserved, and searching for a suitable memory area need not be performed upon allocation. This issue further advocates the use of stack for transient objects.
On the downside, automatic allocation and deallocation of variables creates a potential problem for using references to variables in the stack, as the variables can be erased and replaced by some other variables. Assuming that references are always made from later activations, this can never cause a problem. In practice, however, ensuring this in design and in particular in the maintenance phase can turn out to be hard.

Heap

All data structures whose size or structure can be altered dynamically must be allocated to the heap. The rationale is that since their size cannot be known in advance, it is impossible to reserve enough space from the stack. Another reason that sometimes leads to using the heap rather than the stack is that the object must live despite the phase of the program. In other words, if the role of an object is global, then memory for it should be allocated from heap.

As already discussed, inside methods such variables can be declared static, which essentially make them global. Moreover, if large objects are to be allocated, they can be allocated to the heap to avoid exhausting the stack, whose size can be limited for a program. If there is no automation regarding garbage collection in the run-time system, it is a responsibility of the caller of this function to ensure that memory is released when the data structure is no longer needed. At the same time, it is a good practice to reset the

reference to the area, so that it will not be accidentally used. It is also important to remember that using the heap can be slower than using the stack, as the running program may have to search for a suitable memory area.

Stack	Heap
Stack is used for static memory allocation	Heap for dynamic memory allocation
stored in the computer's RAM	stored in the computer's RAM
Variables allocated on the *stack* are stored directly to the memory and access to this memory is very fast, and it's allocation is dealt with when the program is compiled Variable allocation is fast on stack	Variables allocated on the *heap* have their memory allocated at run time and accessing this memory is a bit slower Variable allocation is heap it's slow.
The stack is always reserved in a *LIFO order*	Element of the heap have no dependencies with each other and can always be *accessed randomly* at any time.
You can use the stack if you know exactly how much data you need to allocate before compile time and it is not too big	You can use heap if you don't know exactly how much data you will need at runtime or if you need to allocate a lot of data.
Stack is a linear data structure	Heap is a nonlinear data structure.

Design Patterns for Limited Memory

When composing designs for devices with a limited amount of memory, the most important principle is not to waste memory.

Here, we focus on their application in the design of programs running in mobile devices.

Linear Data Structures

The principal rule is to favor linear data structures. Linear data structures are generally better for memory management than non-linear ones for several reasons, as listed in the following:

• **Less fragmentation**. Linear data structures occupy memory place from one location, whereas non-linear ones can be located in different places. Obviously, the former results in less possibility for fragmentation.
• **Less searching overhead.** Reserving a linear block of memory for several items only takes one search for a suitable memory element in the run-time environment, whereas non-linear structures require one request for memory per allocated element. Combined

with a design where one object allocates a number of child objects, this may also lead to a serious performance problem.
• **Design-time management**. Linear blocks are easier to manage at design time, as fewer reservations are made. This usually leads to cleaner designs.
• **Monitoring.** Addressing can be performed in a monitored fashion, because it is possible to check that the used index refers to a legal object.
• **Cache improvement.** When using linear data structures, it is more likely that the next data element is already in cache, as cache works internally with blocks of memory. A related issue is that most caches expect that data structures are used in increasing order of used memory locations. Therefore, it is beneficial to reflect this in designs where applicable.
• **Index uses less memory.** An absolute reference to an object usually consumes 32 bits, whereas by allocating objects to a vector of 256 objects, assuming that this is the upper limit of objects, an index of only 8 bits can be used. Furthermore, it is possible to check that there will be no invalid indexing.

Basic Design Decisions

In the following, we introduce some basic principles helping in using linear data structures. The purpose is not to introduce a complete checklist, but rather offer some examples on how linear data structures can be benefited from when composing designs.

Allocate all memory at the beginning of a program. This ensures that the application always has all the memory it needs, and memory allocation can only fail at the beginning of the program. Reserving all the resources is particularly attractive when the most important or mandatory features like emergency calls, for instance, are considered, for which resources must always be available. In general, this type of an approach is best suited for devices that have been optimized for one purpose, and it cannot be generally applied in smartphones except only in some restricted special cases.

Allocate memory for several items, even if you only need one. Then, one can build a policy where a number of objects is reserved with one allocation request. These objects can then be used later when needed. This reduces the number of allocation requests, which leads to a less complex structure in the memory. The approach also improves performance, as there will be fewer memory allocations, and cache use is improved.

Use standard allocation sizes. With a standard allocation size, it is easy to reuse a deallocated area in the memory when the next reservation is made. As a result, fragmentation of memory can be prevented, at least to some extent.

Reuse objects. Reusing old objects might require using a pool of free objects. This requires some data structure for managing free and used data structures. This implies

that the programmer actively participates in the process of selecting object construction and destruction policy in the design.

Release early, allocate late. By always deallocating as soon as possible the programmer can give more options for memory management, because new objects can be allocated to the area that has just been released as well. In contrast, by allocating memory as late as possible, the developer can ensure that all possible deallocations have been performed before the allocation. In particular, one should ensure that objects occupying a large amount of memory are deallocated before allocating new objects. The reason is that in many implementations, heap gives the first suitable memory area, or, in a stack-like implementation, on one end. Then, when large objects are deallocated before allocating others, fragmentation can potentially be prevented, or at very least its effect can be lessened.

Use permanent storage or ROM when applicable. In many situations, it is not even desirable to keep all the data structures in the program memory due to physical restrictions. For instance, in a case when the battery is removed from the device, all unsaved data will be lost. For such situations, it is advisable to introduce the custom to save all data to permanent storage as soon as possible. This can be eased with a user interface that forces the user to commit to completing an entry to calendar or contacts, for instance. A similar fashion can be derived for static data, such as dynamic library and application identifiers or strings used in applications. Furthermore, even if there is no risk of losing data, it may be beneficial from the memory consumption point of view to write large, seldom used objects to permanent storage, so that the device's memory is preserved for more important data.

DYNAMIC LINK LIBRARIES

The terms EXE and DLL are very common in programming. When coding, **you can either export your final project to either a DLL or an EXE.** The term EXE is a shortened version of the word executable as it identifies the file as a program. On the other hand, DLL stands for **Dynamic Link Library, which commonly contains functions and procedures that can be used by other programs**.

In the basest application package, **you would find at least a single EXE file that may or may not be accompanied with one or more DLL files**. An EXE file contains the entry point or the part in the code where the operating system is supposed to begin the execution of the application. **DLL files do not have this entry point and cannot be executed on their own.**

The most major advantage of DLL files is in its reusability. A DLL file can be used in other applications as long as the coder knows the names and parameters of the functions and procedures in the DLL file. Because of this capability, **DLL files are ideal for distributing**

device drivers. The DLL would facilitate the communication between the hardware and the application that wishes to use it. The application would not need to know the intricacies of accessing the hardware just as long as it is capable of calling the functions on the DLL.

Launching an EXE would mean creating a process for it to run on and a memory space. This is necessary in order for the program to run properly. Since a **DLL is not launched by itself and is called by another application, it does not have its own memory space and process**. **It simply shares the process and memory space of the application that is calling it**. Because of this, a DLL might have limited access to resources as it might be taken up by the application itself or by other DLLs.

No.	.Exe	.DLL
1	.EXE is Executable File	DLL is Dynamic Link Library
2	.exe is run individually	.dll can't run individually
3	.exe Has Main Function	.dll doesn't contain Main Function
4	Mainly is for standalone application	.dll give support to exe
5	Only one .exe file exists per application.	Many .dll files may exist in one application.
6	Exe cannot be shared with other applications	.dll can be shared with other applications
7	Exe is for single use whereas you can use Dll for multiple use	A DLL file can be reused by other applications while an EXE cannot
8	EXE is an Out-Process Component.	Dll is an In-Process Component

A DLL file, short for Dynamic Link Library, is a type of file that contains instructions that other programs can call upon to do certain things. This way, multiple programs can share the abilities programmed into a single file, and even do so simultaneously.

Static and Dynamic DLLs

- **Static DLL**
  - (Commonly) Instantiated at application startup
  - Resides in the memory until the application terminates
- **Dynamic DLL (plugin)**
  - Loaded and unloaded whenever needed
  - E.g. different plugin for different messaging types (email/SMS/MMS)

Plug-in

In computing, a plug-in (or plugin, add-in, addin, add-on, addon, or extension) is a software component that adds a specific feature to an existing computer program.

Rules of Thumb for Using Dynamically Loaded Libraries

- ***Reusable or shareable components*** should be implemented using dynamically loaded libraries, as otherwise all the applications that use the components must contain them separately. This in turn consumes memory. In addition, sharing can take place in a form where the same model, implemented as a dynamically loaded library, is reused in several different devices that require a specialized user interface for each device.

- ***Variation or management point*** can be preferable to implement in terms of dynamic libraries. This makes variation or management more controlled, as it can be directly associated with a software component. Moreover, the library can be easily changed, if updates or modifications are needed.

- ***Software development processes*** such as automated testing may require that all the input is in a form that can be directly processed. Then, performing the tests may require that binary components are delivered.

- ***Organizational unit*** can be authorized to compose a single library that is responsible for a certain set of functions. Requesting a library then results in a separate deliverable that can be directly integrated into the final system.

What Constitutes an Application?

The most basic definition of an application is that it is a piece of software that can be started and terminated individually, and that it performs a certain task. Furthermore, it is often necessary to associate a user interface with an application, as otherwise observing the behavior of the application might be difficult.

In the technical sense, an application can be taken as a piece of executable code that can be triggered to execution by the user or the system under some special conditions.

Workflow for Application Development

Perhaps the most important design concern in the design of an application running in a mobile device is the consistency of user experience. Actions must be simple and single yet focused, and they must be accomplished with ease and using only a minimal number of keystrokes. This has an obvious effect on the way in which applications must be designed.

A common workflow for the development of applications for the mobile setting, with special focus on usability and user activities, has been defined by **Salmre (2005**), consisting of:

1. Scoping
2. Performance considerations
3. User interface design
4. Data Model and Memory Concerns, and
5. Communications and I/O.

In the following, we summarize this workflow.

Scoping

Before starting the design of an application for the mobile setting, **one must have the fundamental purpose of the application**, **including both what the application can do and what it cannot**. In particular, when implementing a mobile version of a desktop application, a subset of functions must be selected that will be included in the implementation.
Furthermore, the physical characteristics of the device must be taken into account, if they imply restrictions.
Scoping can be helped by conceptualizing (idea) the application with pictures, mockups (model), and creating prototypes. This will also help when communicating the scope and the purpose of the application to other developers. **One should also consider the relative importance of the functions to users**. For instance, if clock times are rarely entered, it may be enough to use a somewhat inconvenient user interface; while the operation may be annoying, it is needed so seldom that the user can still execute it. However, for entries that are frequent, a well-considered user interface should be implemented.

Performance Considerations

When **scoping has been completed**, the **next step is to consider performance**. To begin with, general responsiveness metrics are needed for applications. **This includes, for instance, defining how fast it should be to open a menu in the application**. **The overall responsiveness is an important part of the user experience**. In addition to generic responsiveness, specific metrics should be created for the most important scenarios. This forces the application designer to consider the chains of events that allow the user to carry out certain procedures.

One way to design for performance is to use an older (or simply less capable) hardware for early experiments. While this gives a pessimistic view on the possibilities of implementing the application, **the design can be initiated before the actual target device**

is available, and with lesser assumptions, it is more likely that the users will be satisfied with the performance.

All assumptions should be tested with a real implementation. **A commonly used approach is to start with some key features and their performance, and to continue to less important features only when the key features have an acceptable level of performance**. Taking into account that in the future more will be expected of the application is usually a good rule of thumb. In particular, an idea where the code is first completed in full in order to determine the worst bottlenecks is usually flawed, because the overall performance is often the most important aspect. Then, data structures, their layout in memory, used algorithms, and the way the user interface is constructed are issues that should be considered first, not individual lines of code. In other words, root causes of performance problems should be focused on instead of their symptoms.

One should also consider that overly focusing on performance can be harmful for portability. Therefore, while it is important to consider that the selected implementation principles are able to satisfy performance requirements, one should not be bound to optimize the development solely for performance. Rather, a reality check on what can be realistically accomplished is to be performed.

User Interface Design

As already discussed, before advancing to the technical design of a mobile application, it is important to study key use cases and features that characterize the application**. If the performance provided by the prototype implementation is good enough in studies, it is time to focus on the right user interface.**
Besides scoping and innovations, one can consider end-user productivity and responsiveness as the most important principles of user interface design. The former means that the actions that are typical and natural for the end-user can be easily and rapidly carried out. The latter means that the user has the feeling of being in control while performing the activities, which commonly implies minimizing the time the user has to wait for activities to complete, and even more importantly, the user is never left wondering what the device is actually doing. The design is further hardened by the tendency of users to perform repeated actions if no response is observed immediately.
This encourages designs where feedback on user-initiated operations is given, even if the actual operation is still in progress behind the scenes. This may require a strategy where the user is tricked into believing that an already completed task takes place only on her command in a proactive fashion (for instance, some application can be always active even if the user has never started it), or that the device lets the user believe the task is completed while it in fact is not (for example, the phone claims to be ready after a reboot even if it has not yet loaded contacts from SIM).

Moreover, in some cases one has to design an enforced flow of control, but at the same time avoid the user becoming frustrated. A further challenge is to keep the user aware of what has really been saved to disk, if the user wishes to turn off the device. Of particular importance in designing the user interface are the available facilities. It is not realistic to copy the user interface greeted in one type of device to another type of device, and expect that usability and user experience will be preserved. Instead, one should consider what seems natural to the user when a certain type of device is available and use that as the starting point of user interface design.

The situation is worsened by the fact that different actions are natural with different devices. For instance, it seems completely realistic to edit Excel macros when using a Communicator type of device, but being able to read the figures might be enough in a normal mobile phone where more restricted resources are available. In general, the design is of course influenced by the size of the screen as well as the restricted input mechanisms. To some extent, this can be solved by using PCs for some of the tasks, and only transferring the outcome to a mobile device. In addition, one can consider whether to aim at special-purpose devices and applications or to a single tool that does everything. One view to this problem is provided by Norman (1998), where an application- and purpose-specific approach is considered to lead to simpler use than a multipurpose approach. In practice, however, it seems that also the latter approach is constantly gaining interest, at least when considering available devices. One contributing factor to this is the cost of manufacturing. New hardware features can be cheaper when they are integrated in a cell phone rather than implementing them in a separate device. Moreover, software features can be virtually free.

Data Model and Memory Concerns

As already discussed, mobile devices offer rather restricted facilities for application development. This is related to unit price of devices, where more sophisticated hardware leads to an increasing price per device, but also power consumption and the size of the device imply certain restrictions. The outcome can be a device where several handicaps exist, but the assumed use cases can be implemented with ease.

The way in which data is represented has an impact on how it can be located in the memory, on how the system behaves in peak conditions, and on how the application disposes data. For an application developer, this implies that data structures and memory use in general must be carefully considered. Also dynamically loaded libraries can be considered as an issue that is closely related to data model and memory concerns, as their technical implementation can rely on DLLs.

Communications and I/O

The way communications and I/O are defined determines how the application communicates with the resources that are located beyond its control. This includes

devices' internal resources, such as files and subsystems, as well as resources that are external to the device, and require a communications mechanism before an access.

For instance, the latter includes socket-based communications, files on servers, Web Services, and remote databases, to name some options. The way in which the application handles local and remote resources has a major effect on usability. Accessing local resources is usually fast, whereas communicating with remote resources is slow, at least with the current implementation techniques.
A decision to load some data from a remote location in anticipation of the user's actions can in some cases result in major improvements in user experience. However, in general this is impossible, and should only be carried out in special cases, where users' intentions can be accurately modeled in advance.

Another important aspect to consider with communications and I/O is the level of abstraction of transmitted and stored data. For example, one can consider the following levels of abstraction in using files:

1. binary streams, where the data is stored in a fashion that is unreadable without auxiliary software,
2. text streams, where data becomes more readable, but may still remain somewhat unstructured and unreadable for a human reader
3. XML forward-only readers and writers, where more meta-information is included
4. XML Document Object Model, where complex automatic processing of included data is usually enabled.

The different levels of abstraction offer different facilities for manipulating data.

The more abstract the level, the easier it is to process the data and the more self contained the files are. This implies that developers' productivity improves, as programming, debugging, and maintenance will be easier, and it is more likely that potentially available standard components can be used, or reuse options exist within the company. However, at the same time the amount of overhead in transmitting, processing, and storing increases, which means that the approach may not be suited for cases where a large amount of data must be processed in a short period of time. This can lead to contradicting requirements in application development that complicate the design. The design is made more difficult by the fact that it is seldom a practical way to include several implementations of the same feature in the device, even if their characteristics would be different. In addition, costs associated with the connection may become an important factor if a cellular data connection is assumed.
For instance, one may wish to download as much data as possible when wireless LAN connection is available, but accept only minimal connectivity when using GPRS.

Unit – V

IEEE 802.11 WLAN

The IEEE (Institute of Electrical and Electronics Engineers) describes itself as "the world's largest technical professional society -- promoting the development and application of electro technology and allied sciences for the benefit of humanity, the advancement of the profession, and the well-being of our members."

IEEE 802.11 is a set of media access control (MAC) and physical layer (PHY) specifications for implementing wireless local area network (WLAN) computer communication in the 900 MHz and 2.4, 3.6, 5, and 60 GHz frequency bands. They are created and maintained by the Institute of Electrical and Electronics Engineers (IEEE) LAN/MAN Standards Committee (IEEE 802).

WiFi stands for Wireless Fidelity. WiFi It is based on the IEEE 802.11 family of standards and is primarily a local area networking (LAN) technology designed to provide in-building broadband coverage.

Radio Signals

Radio Signals are the keys, which make WiFi networking possible. These radio signals transmitted from WiFi antennas are picked up by WiFi receivers, such as computers and cell phones that are equipped with WiFi cards. Whenever, a computer receives any of the signals within the range of a WiFi network, which is usually 300 — 500 feet for antennas, the WiFi card reads the signals and thus creates an internet connection between the user and the network without the use of a cord.

Access points, consisting of antennas and routers, are the main source that transmit and receive radio waves. Antennas work stronger and have a longer radio transmission with a radius of 300-500 feet, which are used in public areas while the weaker yet effective router is more suitable for homes with a radio transmission of 100-150 feet.

WiFi Hotspots

A WiFi hotspot is created by installing an access point to an internet connection. The access point transmits a wireless signal over a short distance. It typically covers around 300 feet. When a WiFi enabled device such as a Pocket PC encounters a hotspot, the device can then connect to that network wirelessly.

Most hotspots are located in places that are readily accessible to the public such as airports, coffee shops, hotels, book stores, and campus environments. 802.11b is the most common specification for hotspots worldwide. The 802.11g standard is backwards compatible with .11b but .11a uses a different frequency range and requires separate hardware such as an a, a/g, or a/b/g adapter.

Some Hotspots require WEP key to connect, which is considered as private and secure. As for open connections, anyone with a WiFi card can have access to that hotspot. So in order to have internet access under WEP, the user must input the WEP key code.

There are two types of wireless networks:

The stations of the wireless network can communicate directly with each other, we called Ad Hoc network type, or via relay terminals called APs (Access Points, PA) then it is an infrastructure network. The second type is by far the most common in business.

- Type networks **Ad Hoc**, where stations communicate directly.
- **Infrastructure type** networks where stations communicate through access points.

There are several variations of WiFi. In short, 802.11b and 802.11g are compatible them and both operate with the radio waves of a frequency of 2.4 GHz. The 802.11b reached a speed of 11 Mb / s and 802.11g rises to 54 Mb / s. The 802.11a is not compatible with 802.11b and 802.11g, because it works with the waves a radio frequency of 5 GHz. It can reach 54 Mb / s. The 802.11n allows to achieve a real flow rate greater than 100 Mb / s. It is capable of operating at 2.4 GHz or 5 GHz and is compatible with the 802.11b / g and 802.11a. Unfortunately. Most 802.11n equipment available today use only tape 2.4 GHz (and are therefore not compatible with the 802.11a). Today the WiFi version of the most used is far 802.11g. It should be rapidly overtaken by 802.11n.

802.11 Architecture

The 802.11architecture defines two types of services and three different types of stations

802.11 Services

The two types of services are

1. Basic services set (BSS)
2. Extended Service Set (ESS)

1. Basic Services Set (BSS)

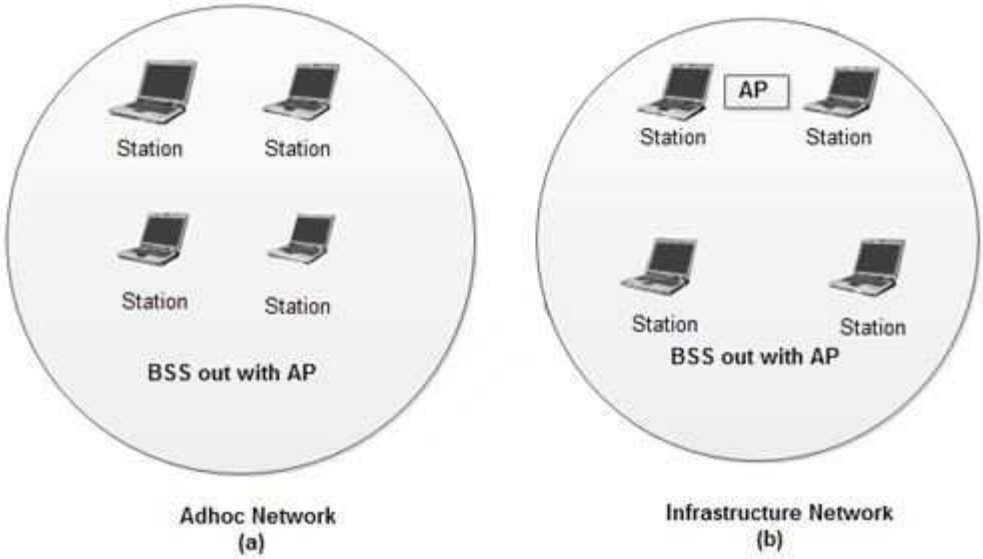
- The basic services set contain stationary or mobile wireless stations and a central base station called access point (AP).

• The use of access point is optional.

• If the access point is not present, it is known as stand-alone network. Such a

BSS cannot send data to other BSSs. This type of architecture is known as adhoc architecture.

• The BSS in which an access point is present is known as an infrastructure network.



Basic Service Sets

2. Extend Service Set (ESS)

• An extended service set is created by joining two or more basic service sets (BSS) having access points (APs).

• These extended networks are created by joining the access points of basic services sets through a wired LAN known as distribution system.

• The distribution system can be any IEET LAN.

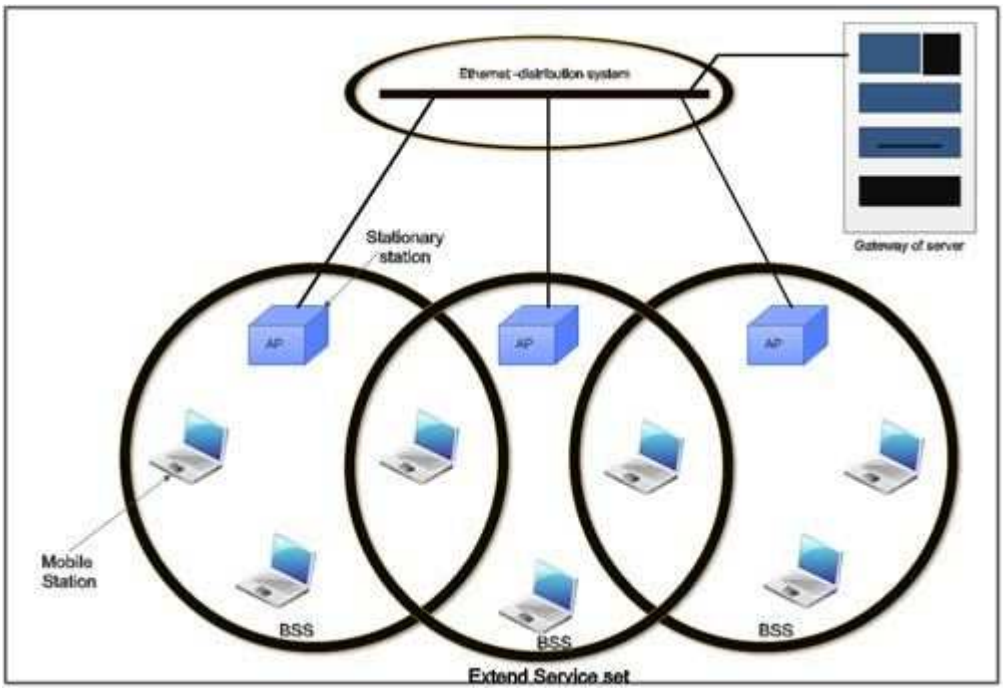
There are two types of stations in ESS:

(i) **Mobile stations**: These are normal stations inside a BSS.

(ii) **Stationary stations**: These are AP stations that are part of a wired LAN.

• Communication between two stations in two different BSS usually occurs via two APs.

• A mobile station can belong to more than one BSS at the same time.



802.11 Station Types

IEEE 802.11 defines three types of stations on the basis of their mobility in wireless LAN. These are:

1. No-transition Mobility

2. BSS-transition Mobility

3. ESS-transition Mobility

1. **No-transition .Mobility**: These types of stations are either stationary *i.e.* immovable or move only inside a BSS.

2. **BSS-transition mobility**: These types of stations can move from one BSS to another but the movement is limited inside an ESS.

3. **ESS-transition mobility**: These types of stations can move from one ESS to another. The communication mayor may not be continuous when a station moves from one ESS to another ESS.

Table 7.5. 802.11 Wireless Standards

IEEE Standard	Frequency/Medium	Speed	Topology	Transmission Range	Access Method
802.11	2.4GHz RF	1 to 2Mbps	Ad hoc/infrastructure	20 feet indoors.	CSMA/CA
802.11a	5GHz	Up to 54Mbps	Ad hoc/infrastructure	25 to 75 feet indoors; range can be affected by building materials.	CSMA/CA
802.11b	2.4GHz	Up to 11Mbps	Ad hoc/infrastructure	Up to 150 feet indoors; range can be affected by building materials.	CSMA/CA
802.11g	2.4GHz	Up to 54Mbps	Ad hoc/infrastructure	Up to 150 feet indoors; range can be affected by building materials.	CSMA/CA
802.11n	2.4GHz/5GHz	Up to 600Mbps	Ad hoc/infrastructure	175+ feet indoors; range can be affected by building materials.	CSMA/CA

CSMA CA vs CSMA CD

Carrier Sense Multiple Access or CSMA is a Media Access Control (MAC) protocol that is used to control the flow of data in a transmission media so that packets do not get lost and data integrity is maintained. There are two modifications to CSMA, the CSMA CD **(Collision Detection)** and *CSMA CA* **(Collision Avoidance)**, each having its own strengths.

CSMA operates by sensing the state of the medium in order to prevent or recover from a collision. A collision happens when two transmitters transmit at the same time. The data gets scrambled, and the receivers would not be able to discern one from the other thereby causing the information to get lost. The lost information needs to be resent so that the receiver will get it.

CSMA CD operates by detecting the occurrence of a collision. Once a collision is detected, CSMA CD immediately terminates the transmission so that the transmitter does not have to waste a lot of time in continuing. The last information can be retransmitted. In comparison, CSMA CA does not deal with the recovery after a collision. What it does is to check whether the medium is in use. If it is busy, then the transmitter waits until it is

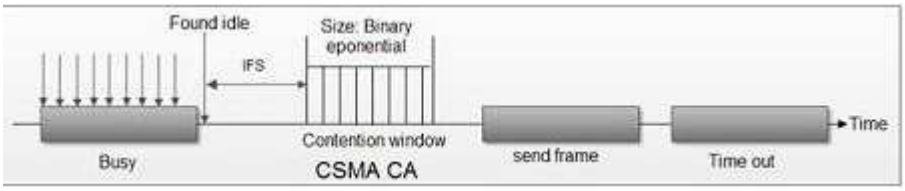
idle before it starts transmitting. This effectively minimizes the possibility of collisions and makes more efficient use of the medium.

Another difference between CSMA CD and CSMA CA is where they are typically used. CSMA CD is used mostly in wired installations because it is possible to detect whether a collision has occurred. With wireless installations, it is not possible for the transmitter to detect whether a collision has occurred or not. That is why wireless installations often use CSMA CA instead of CSMA CD.

CSMA/CA protocol is used in wireless networks because they cannot detect the collision so the only solution is collision avoidance.

CSMA/CA avoids the collisions using three basic techniques.

(i) Interframe space (ii) Contention window (iii) Acknowledgements



1. Interframe Space (IFS)

• Whenever the channel is found idle, the station does not transmit immediately. It waits for a period of time called interframe space (IFS).

• When channel is sensed to be idle, it may be possible that same distant station may have already started transmitting and the signal of that distant station has not yet reached other stations.

• Therefore the purpose of IFS time is to allow this transmitted signal to reach other stations.

• If after this IFS time, the channel is still idle, the station can send, but it still needs to wait a time equal to contention time.

• IFS variable can also be used to define the priority of a station or a frame.

2. Contention Window

• Contention window is an amount of time divided into slots.

• A station that is ready to send chooses a random number of slots as its wait time.

• The number of slots in the window changes according to the binary exponential back-off strategy. It means that it is set of one slot the first time and then doubles each time the station cannot detect an idle channel after the IFS time.

• This is very similar to the p-persistent method except that a random outcome defines the number of slots taken by the waiting station.

• In contention window the station needs to sense the channel after each time slot.

• If the station finds the channel busy, it does not restart the process. It just stops the timer & restarts it when the channel is sensed as idle.

3. Acknowledgement

• Despite all the precautions, collisions may occur and destroy the data.

• The positive acknowledgment and the time-out timer can help guarantee that receiver has received the frame.

GSM

Cellular System Architecture

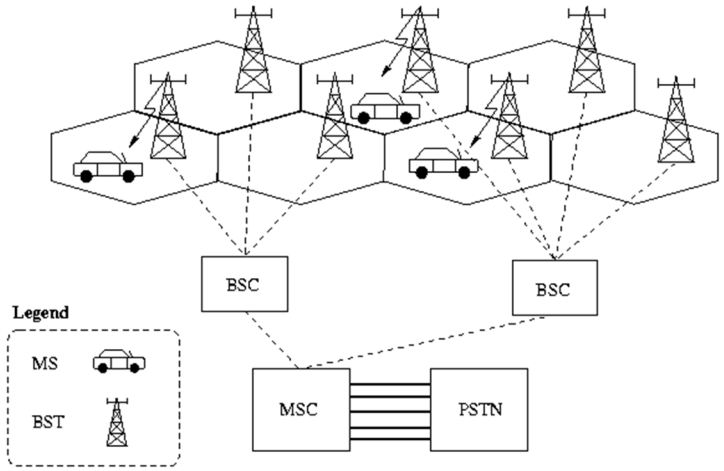
Cells: - A cell is the basic geographic unit of a cellular system. Shape of the areas into which a coverage region is divided.

Cells are base stations transmitting over small geographic areas that are represented as hexagons.

Each cell size varies depending on the landscape.

Because of constraints imposed by natural terrain and man-made structures

Clusters:- A cluster is a group of cells No channels are reused within a cluster.



Cellular Approach

With limited frequency resource, cellular principle can serve thousands of subscribers at an affordable cost. In a cellular network, total area is subdivided into smaller areas called "cells". Each cell can cover a limited number of mobile subscribers within its boundaries. Each cell can have a base station with a number of RF channels.

Frequencies used in a given cell area will be simultaneously reused at a different cell which is geographically separated. For example, a typical seven-cell pattern can be considered.

Total available frequency resources are divided into seven parts, each part consisting of a number of radio channels and allocated to a cell site. In a group of 7 cells, available frequency spectrum is consumed totally. The same seven sets of frequency can be used after certain distance.

The group of cells where the available frequency spectrum is totally consumed is called a cluster of cells.

Two cells having the same number in the adjacent cluster, use the same set of RF channels and hence are termed as "Co-channel cells". The distance between the cells using the same frequency should be sufficient to keep the co-channel (co-chl) interference to an acceptable level. Hence, the cellular systems are limited by Co-channel interference.

Hence a cellular principle enables the following.

- More efficient usage of available limited RF source.
- Manufacturing of every piece of subscriber's terminal within a region with the same set of channels so that any mobile can be used anywhere within the region.

Shape of Cells

For analytical purposes a "Hexagon" cell is preferred to other shapes on paper due to the following reasons.

- A hexagon layout requires fewer cells to cover a given area. Hence, it envisages fewer base stations and minimum capital investment.

- Other geometrical shapes cannot effectively do this. For example, if circular shaped cells are there, then there will be overlapping of cells.
- Also for a given area, among square, triangle and hexagon, radius of a hexagon will be the maximum which is needed for weaker mobiles.

In reality cells are not hexagonal but irregular in shape, determined by factors like propagation of radio waves over the terrain, obstacles, and other geographical constraints. Complex computer programs are required to divide an area into cells. One such program is "Tornado" from Siemens.

Cellular Hierarchy

Pico cell - (<10 meters)

A **Pico cell** is a small cellular base station typically covering a small area, such as in-building (offices, shopping malls, train stations, stock exchanges, etc.), or more recently in-aircraft. In cellular networks, Pico cells are typically used to extend coverage to indoor areas where outdoor signals do not reach well, or to add network capacity in areas with very dense phone usage, such as train stations or stadiums. Pico cells provide coverage and capacity in areas difficult or expensive to reach using the more traditional macro cell approach

Microcell (100-1000 meters)

A **microcell** is a cell in a mobile phone network served by a low power cellular base station (tower), covering a limited area such as a mall, a hotel, or a transportation hub. A microcell is usually larger than a Pico cell, though the distinction is not always clear. A microcell uses power control to limit the radius of its coverage area.

Typically the range of a microcell is less than two kilometers wide, whereas standard base stations may have ranges of up to 35 kilometers.

Macrocell (>1000 Meters)

Magacell (cells with global coverage)

Femtocell

In telecommunications, a **femtocell** is a small, low-power cellular base station, typically designed for use in a home or small business. A broader term which is more widespread in the industry is small cell, with femtocell as a subset. It is also called femto Access Point(AP). It connects to the service provider's network via broadband (such as DSL or cable); current designs typically support four to eight simultaneously active mobile phones in a residential setting depending on version number and femtocell hardware, and eight to 16 mobile phones in enterprise settings. A femtocell allows service providers

to extend service coverage indoors or at the cell edge, especially where access would otherwise be limited or unavailable. Although much attention is focused on WCDMA, the concept is applicable to all standards, including GSM, CDMA2000, TD-SCDMA, WiMAX and LTE solutions.

What is GSM?

If you are in Europe or Asia and using a mobile phone, then most probably you are using GSM technology in your mobile phone.

- GSM stands for **G**lobal **S**ystem for **M**obile Communication. It is a digital cellular technology used for transmitting mobile voice and data services.
- The concept of GSM emerged from a cell-based mobile radio system at Bell Laboratories in the early 1970s.
- GSM is the name of a standardization group established in 1982 to create a common European mobile telephone standard.
- GSM is the most widely accepted standard in telecommunications and it is implemented globally.
- GSM is a circuit-switched system that divides each 200 kHz channel into eight 25 kHz time-slots. GSM operates on the mobile communication bands 900 MHz and 1800 MHz in most parts of the world. In the US, GSM operates in the bands 850 MHz and 1900 MHz.
- GSM owns a market share of more than 70 percent of the world's digital cellular subscribers.
- GSM makes use of narrowband Time Division Multiple Access (TDMA) technique for transmitting signals.
- GSM was developed using digital technology. It has an ability to carry 64 kbps to 120 Mbps of data rates.
- Presently GSM supports more than one billion mobile subscribers in more than 210 countries throughout the world.
- GSM provides basic to advanced voice and data services including roaming service. Roaming is the ability to use your GSM phone number in another GSM network.

Why GSM?

Listed below are the features of GSM that account for its popularity and wide acceptance.

- Improved spectrum efficiency
- International roaming
- Low-cost mobile sets and base stations (BSs)
- High-quality speech
- Compatibility with Integrated Services Digital Network (ISDN) and other telephone company services
- Support for new services

General Features of GSM

GSM (Global System for Mobile Communications) is a second-generation (2G) digital mobile telephones standard using a combination Time Division Multiple Access (TDMA) and Frequency Division Multiple Access (FDMA) to share the bandwidth among as many subscribers as possible.

1. GSM provides only 9.6 kbps data connection. Increase in data rates can be achieved when GSM changes into a radio service based on wide band code division multiple access, and not TDMA.

2. GSM digitizes and compresses voice data, then sends it down a channel with two other streams of user data, each in its own time slot. It operates at either the 900, 1800 or 1,900 MHz frequency bands.

3. The uplink and down link frequencies for GSM are different and therefore a channel has a pair of frequencies 80 MHz apart. The separation between uplink and downlink frequencies is called duplex distance.

4. In a channel the separation between adjacent carrier frequencies is known as channel separation which is 200 kHz in case of GSM.

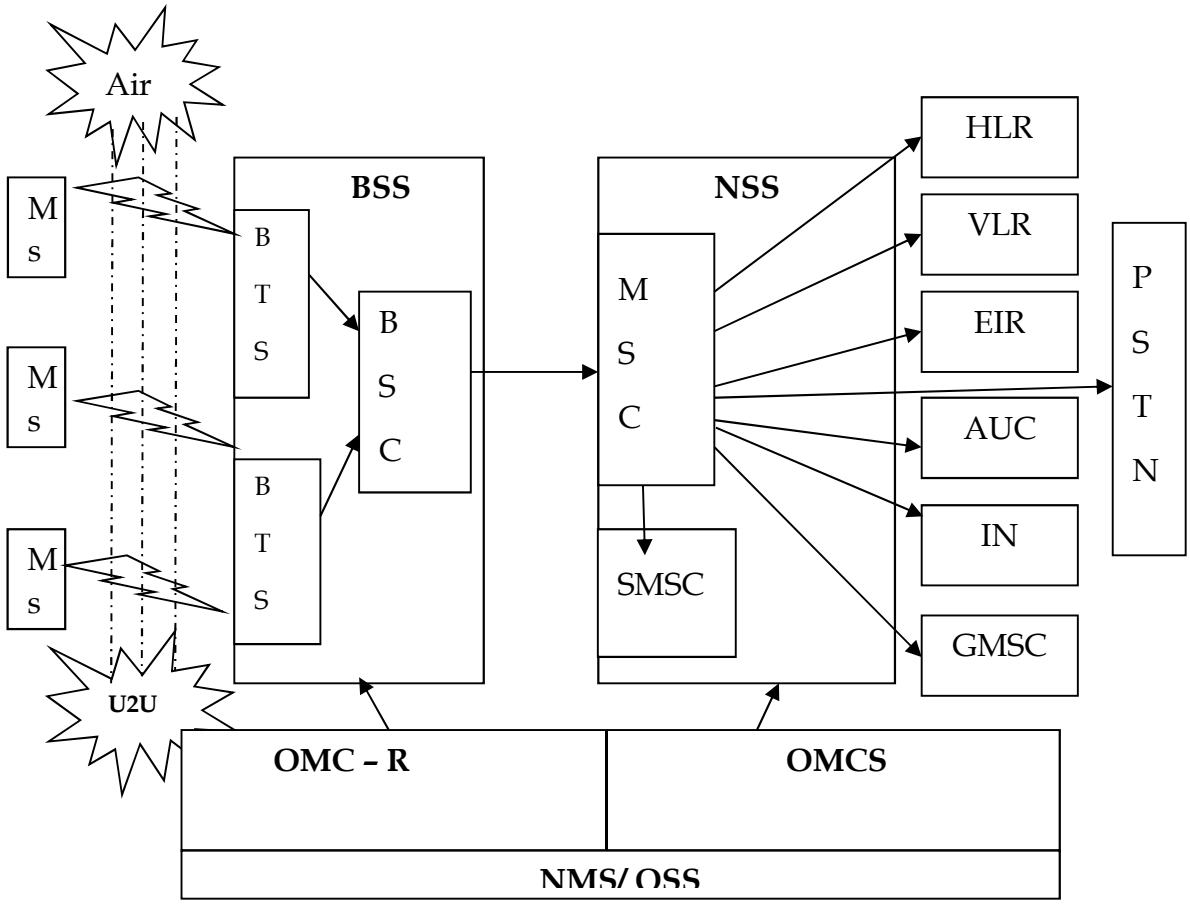
5. The services supported by GSM are telephony, fax and SMS, call forwarding, caller 10, call waiting and the like.

6. GSM supports data at rates up to 9.6 kbps on POTS (Plain Old Telephone Service), ISDN, Packet Switched Public Data Networks, and Circuit Switched Public Data Networks.

8. Being a digital system, GSM does not require a modem between subscriber and GSM network.

GSM- Architecture

Global System for Mobile Communication



Main Entity (BSS-NSS-NMS/OSS)

BSS – Base station subsystem ⟶ **BTS –** Base transceiver station & **BSC** –Base Station Controller

NSS – Network station subsystem ⟶ **MSC**-Mobile Switching Center,

HLR-Home Location Register, **VLR**-Visitor Location Register**, EIR**- Equipment Identity Register (White, Grey & Black List users), **AUC –** Authentication Center, **IN**- Intelligent Network, **GMSC**-Gateway MSC, **SMSC**-Short message service center

NMS – Network Management system ⟶ **OMC-R** - Operation Maintenance Center for Radio **OMC-S -** Operation Maintenance Center for Switch

OSS- Operation Station subsystem

PSTN – Public Switched Telephone Network

Base Transceiver Station

- BTS is a wireless transmission and receiving terminal.
- All mobile device connected with a nearest BTS.
- A group of BTS connected with a BSC
- Each BTS has a identification number in the network. Based on site and capacity it can be separated to 3-6 cell. There is a antenna for each cell.

Base Station Controller

- BSC is a GSM node that controls one or more BTS in the network. BTS can be connected using microwave or optical fiber.
- BSC connected with MSC for voice and signal communication. For data communication its connected with SGSN
- Mobile device handover intelligence between BTS which is called BTS handover and call setup controlled by BSC.
- Radio network management including radio frequency controlled by BSC

Mobile Switching Center

- MSC is the sub center of large network or center of small network (GSM Core Network)
- MSC is related with switching, call setup, release.
- MSC control a group of BSC
- In large GSM network MSC connected with STP for signal routing and MGW for Voice switching.

Home Location Register

- HLR is a central database contains mobile subscriber's details information which is used for core network. Every subscriber should be identified with IMSI/MSISDN pair and uniquely associated with one HLR

Visitor Location Register

- VLR is a subscriber database having subscriber's details information. VLR response will be faster than HLR, VLR store some additional information for which HLR need to communicate with voice network or radio network.

Equipment Identity Register

- EIR is a central database, which contains subscriber's handset IMEI. If network configured for check in EIR, then for any subscriber if the IMEI is enlisted in EIR, the mobile device will not be able to use in the network.
- EIR can be used to prevent stolen or unauthorized mobile device use

Short Message Service Center

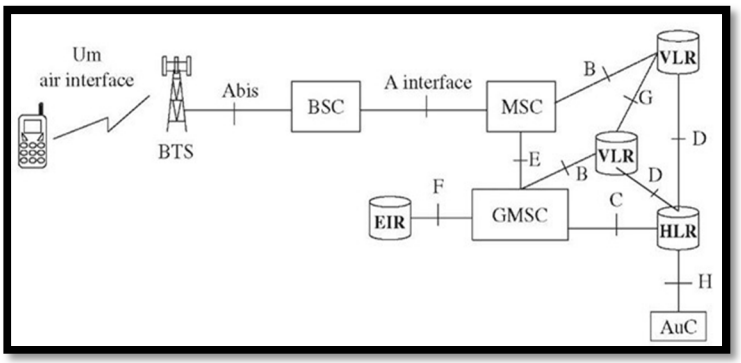
- SMSC is a node that deliver SMS to subscriber. Any SMS either from subscriber to subscriber or application to subscriber must be sent via a SMSC.
- Every subscriber SIM store a service center number where the subscriber initiated SMS send.
- SMSC communicate with HLR for subscriber location and VLR for delivering the SMS

OSS

- The OSS (Operational Support Systems) supports operation and maintenance of the system and allows engineers to monitor, diagnose, and troubleshoot every aspect of the GSM network.

GSM Interfaces

The air interface between the MS and the BTS is called Um. The GSM air interface is based on time division multiple access (TDMA) with frequency division duplex (FDD). TDMA allows multiple users to share a common RF channel on a time-sharing basis, while FDD enables different frequencies to be used in uplink (MS to BTS) and downlink (BTS to MS) directions.



GSM interface	Description with position
Um	It is the air interface used between MS and BTS. Also referred as Air interface.
A	It is used between BSC and MSC.
Abis	It is used between BTS and BSC.
B	It is used between MSC and VLR.
C	It is used between HLR and GMSC. Also between MSC and HLR.
D	It is used between HLR and VLR.
E	It is used between MSC and another MSC or G-MSC.
F	It is used between EIR and MSC and between EIR and G-MSC.
G	It is used between VLR and another VLR.

1.***Um interface*** The "air" or radio interface standard that is used for exchanges between a mobile (ME) and a base station (BTS / BSC). For signalling, a modified version of the ISDN LAPD, known as LAPDm is used.

2.***Abis interface*** This is a BSS internal interface linking the BSC and a BTS, and it has not been totally standardised. The Abis interface allows control of the radio equipment and radio frequency allocation in the BTS.

3.***A interface*** The A interface is used to provide communication between the BSS and the MSC. The interface carries information to enable the channels, timeslots and the like to be allocated to the mobile equipments being serviced by the BSSs. The messaging required within the network to enable handover etc to be undertaken is carried over the interface.

4. ***B interface*** The B interface exists between the MSC and the VLR . It uses a protocol known as the MAP/B protocol. As most VLRs are collocated with an MSC, this makes the interface purely an "internal" interface. The interface is used whenever the MSC needs access to data regarding a MS located in its area.

5. ***C interface*** The C interface is located between the HLR and a GMSC or a SMS-G. When a call originates from outside the network, i.e. from the PSTN or another mobile network it ahs to pass through the gateway so that routing information required to complete the call may be gained. The protocol used for communication is MAP/C, the letter "C" indicating that the protocol is used for the "C" interface. In addition to this, the MSC may optionally forward billing information to the HLR after the call is completed and cleared down.

6. ***D interface*** The D interface is situated between the VLR and HLR. It uses the MAP/D protocol to exchange the data related to the location of the ME and to the management of the subscriber.
7. ***E interface*** The E interface provides communication between two MSCs. The E interface exchanges data related to handover between the anchor and relay MSCs using the MAP/E protocol.
8. ***F interface*** The F interface is used between an MSC and EIR. It uses the MAP/F protocol. The communications along this interface are used to confirm the status of the IMEI of the ME gaining access to the network.
9. ***G interface*** The G interface interconnects two VLRs of different MSCs and uses the MAP/G protocol to transfer subscriber information, during e.g. a location update procedure.
10. ***H interface*** The H interface exists between the MSC the SMS-G. It transfers short messages and uses the MAP/H protocol.
11. ***I interface*** The I interface can be found between the MSC and the ME. Messages exchanged over the I interface are relayed transparently through the BSS.

GSM Services

GSM offers much more than just voice telephony. Contact your local GSM network operator to the specific services that you can avail.

GSM offers three basic types of services:

- Telephony services or teleservices
- Data services or bearer services
- Supplementary services

Teleservices

The abilities of a Bearer Service are used by a Teleservice to transport data. These services are further transited in the following ways:

Voice Calls

The most basic Teleservice supported by GSM is telephony. This includes full-rate speech at 13 kbps and emergency calls, where the nearest emergency-service provider is notified by dialing three digits.

Short Text Messages

Short Messaging Service (SMS) service is a text messaging service that allows sending and receiving text messages on your GSM mobile phone. In addition to simple text messages, other text data including news, sports, financial, language, and location-based data can also be transmitted.

Bearer Services

Data services or Bearer Services are used through a GSM phone. to receive and send data is the essential building block leading to widespread mobile Internet access and mobile data transfer. GSM currently has a data transfer rate of 9.6k. New developments that will push up data transfer rates for GSM users are HSCSD (high speed circuit switched data) and GPRS (general packet radio service) are now available.

Supplementary Services

Supplementary services are additional services that are provided in addition to teleservices and bearer services. These services include caller identification, call forwarding, call waiting, multi-party conversations, and barring of outgoing (international) calls, among others. A brief description of supplementary services is given here:

- **Conferencing** : It allows a mobile subscriber to establish a multiparty conversation, i.e., a simultaneous conversation between three or more subscribers to setup a conference call. This service is only applicable to normal telephony.
- **Call Waiting** : This service notifies a mobile subscriber of an incoming call during a conversation. The subscriber can answer, reject, or ignore the incoming call.
- **Call Hold** : This service allows a subscriber to put an incoming call on hold and resume after a while. The call hold service is applicable to normal telephony.
- **Call Forwarding** : Call Forwarding is used to divert calls from the original recipient to another number. It is normally set up by the subscriber himself. It can be used by the subscriber to divert calls from the Mobile Station when the subscriber is not available, and so to ensure that calls are not lost.
- **Call Barring** : Call Barring is useful to restrict certain types of outgoing calls such as ISD or stop incoming calls from undesired numbers. Call barring is a flexible service that enables the subscriber to conditionally bar calls.

- **Number Identification** : There are following supplementary services related to number identification:
    - **Calling Line Identification Presentation** : This service displays the telephone number of the calling party on your screen.
    - **Calling Line Identification Restriction** : A person not wishing their number to be presented to others subscribes to this service.
    - **Connected Line Identification Presentation** : This service is provided to give the calling party the telephone number of the person to whom they are connected. This service is useful in situations such as forwarding's where the number connected is not the number dialled.
    - **Connected Line Identification Restriction** : There are times when the person called does not wish to have their number presented and so they would subscribe to this person. Normally, this overrides the presentation service.
    - **Malicious Call Identification** : The malicious call identification service was provided to combat the spread of obscene or annoying calls. The victim should subscribe to this service, and then they could cause known malicious calls to be identified in the GSM network, using a simple command.
- **Advice of Charge (AoC)** : This service was designed to give the subscriber an indication of the cost of the services as they are used. Furthermore, those service providers who wish to offer rental services to subscribers without their own SIM can also utilize this service in a slightly different form. AoC for data calls is provided on the basis of time measurements.
- **Closed User Groups (CUGs)** : This service is meant for groups of subscribers who wish to call only each other and no one else.
- **Unstructured supplementary services data (USSD)** : This allows operator-defined individual services.

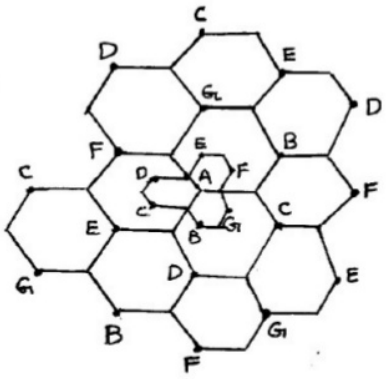
Why cell splitting and sectoring?

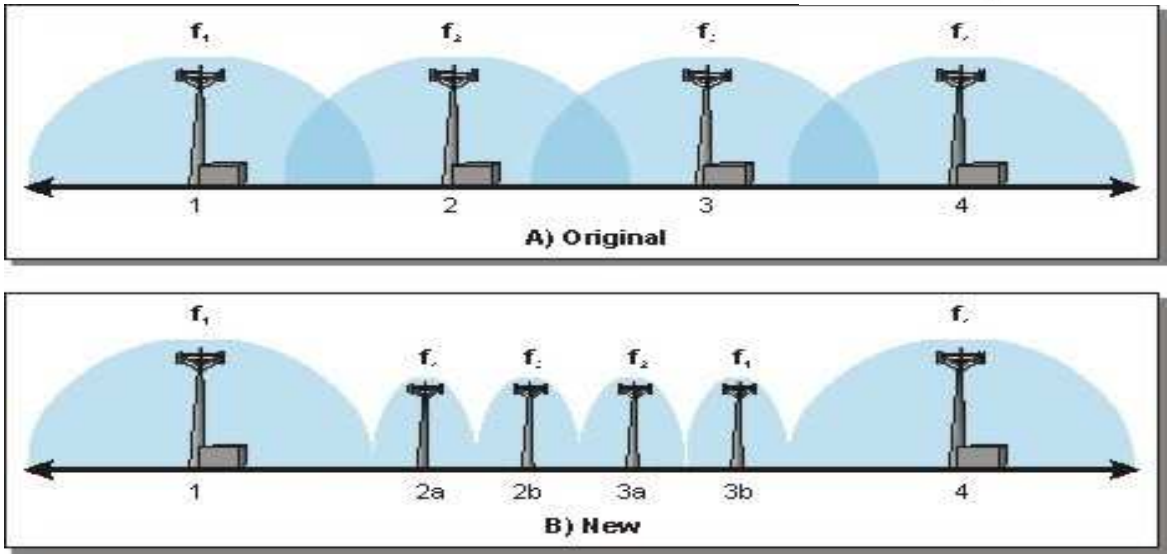
As users increases channel capacity decreases. Techniques are needed to provide extra channels.

Cell splitting and sectoring increases capacity.

Cell Splitting

Cell splitting is the process of subdividing a congested cell into smaller cells such that each smaller cell has its own base station with reduced antenna height and reduced transmitter power. It increases the capacity of a cellular system since number of times channels are reused increases.

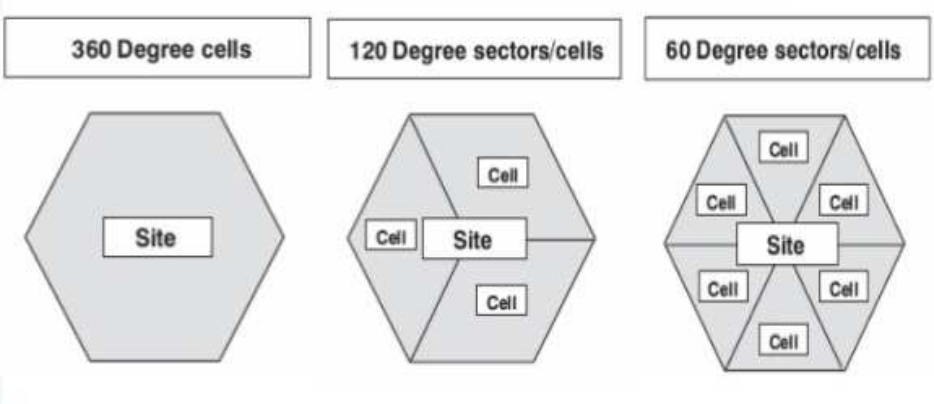




Advantages: Reduce the reuse distance between the cells. Use small antenna and they transmit less power as compare to the larger.

Limitations: Handoffs are more frequent. Channel assignments become difficult. All cells are not split simultaneously so special care have to be taken for proper allocation of problem.

Cell sectoring: To overcome some limitations like co- channel interference, cell sectoring is done. It involves replacing an Omni-directional antenna at the base station by several directional antennas.



Advantages: It improves S/I ratio (signal-to-interference). It reduces interference which increases capacity. It enables to reduce the cluster size and provides an additional freedom in assigning channels. Here we don't touch the reuse distance and we will not touch the Cell size. We replace antenna only which is used in cellular Base station.

Limitations: Increased number of antennas at each base station. Since sectoring reduces the coverage area of a particular group of channels, the number of handoffs increases as well.

Need of Frequency Reuse

The Geographical area is divided as small hexagonal shaped structure known as cell.

Each cell has its own range of frequencies (Spectrum).

By doing so, we can allocate the spectrum for very few cells.

We'll eventually ran out of all the frequencies by allocating to each cell.

Which makes impossible to cover all the areas, hence the concept of frequency reuse is introduced.

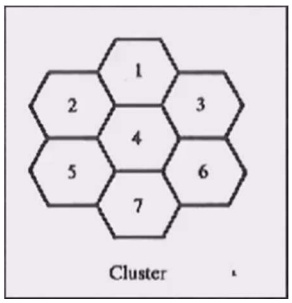
Problem

Consider this Cluster, with 7 cells, here the spectrum is allocated uniquely for each cell.

There are only 7 unique spectrum is available for communication. Each number represents each range of frequency.

This cluster only covers a small geographical area, but we need to cover all the area.

Hence we go for frequency reuse.



Solution

People thought that what if we use the same range of frequency, which is already used.

Many people opposed this theory, and said that it will cause dramatic interference during the communication.

But with specific distance, the above problem can be overcome.

This theory developed and opened the gateway for new research.

This concept is called as Frequency Reuse.

Frequency Reuse

Frequency reuse is a technique of reusing frequencies and channels within a communication system to improve capacity and spectral efficiency.

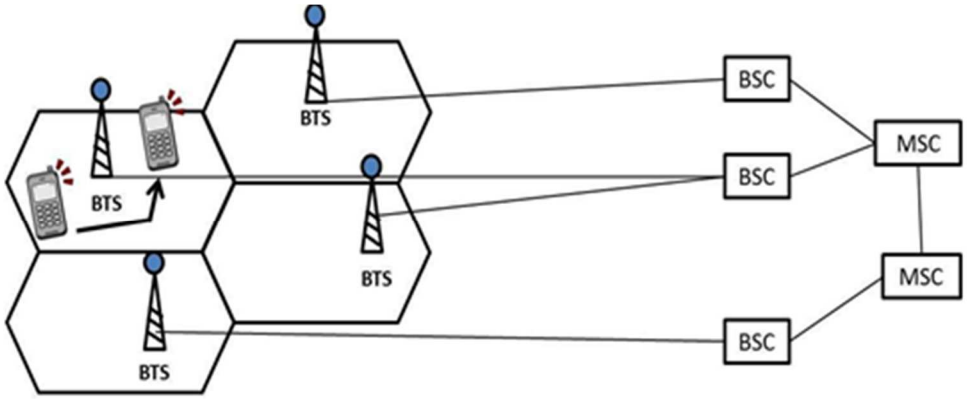
Frequency reuse is one of the fundamental concepts on which commercial wireless systems are based that involve the partitioning of an RF radiating area into cells.

It is also called as frequency planning.

There are four basic types of handoffs in GSM network:

1. Intra-cell handover:

Such a kind of handover is performed to optimize the traffic load in the cell or to improve quality of a connection by changing carrier frequency.



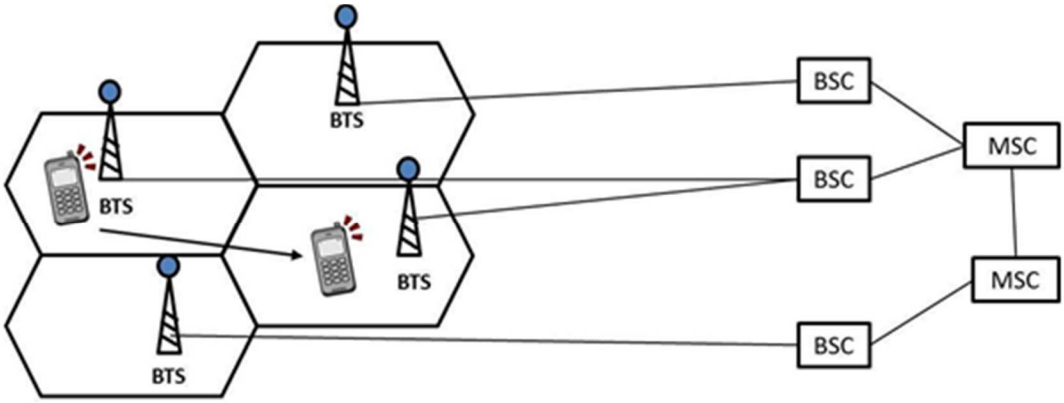
Intra-cell handover

2 Inter-cell handover:

It is also known as Intra-BSC handover.

Here the mobile moves from one cell to another but remains within the same BSC (Base station controller).

Here the BSC handles the handover process



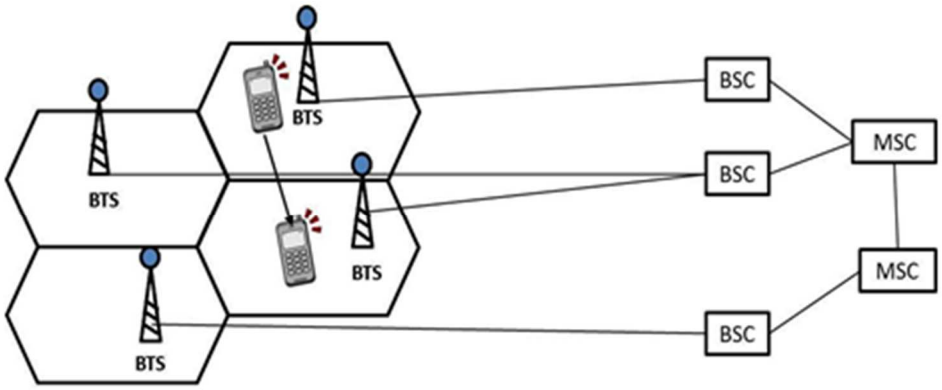
Intra-BSC handover

3 Inter-BSC handover:

It is also called as Intra-MSC handover.

As BSC can control only a limited number of cells, we might usually need to transfer a mobile from one BSC to another BSC.

Here the MSC handles the handover process.

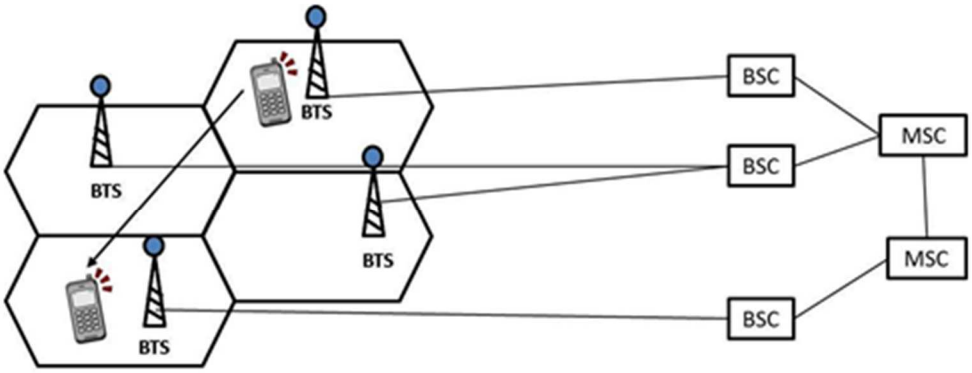


Inter-BSC handover

4 Inter-MSC handover:

It occurs when a mobile move from one MSC region to another MSC.

MSC cover a large area. It can be imagined as a handover from Maharashtra MSC to Gujarat MSC while travelling.



Inter-MSC handover

GSM - Addresses and Identifiers

GSM treats the users and the equipment in different ways. Phone numbers, subscribers, and equipment identifiers are some of the known ones. There are many other identifiers that have been well-defined, which are required for the subscriber's mobility management and for addressing the remaining network elements. Vital addresses and identifiers that are used in GSM are addressed below.

International Mobile Station Equipment Identity (IMEI)

The International Mobile Station Equipment Identity (IMEI) looks more like a serial number which distinctively identifies a mobile station internationally. This is allocated by the equipment manufacturer and registered by the network operator, who stores it in the Equipment Identity Register (EIR). By means of IMEI, one recognizes obsolete, stolen, or non-functional equipment.

Following are the parts of IMEI:

- **Type Approval Code (TAC)** : 6 decimal places, centrally assigned.
- **Final Assembly Code (FAC)** : 6 decimal places, assigned by the manufacturer.
- **Serial Number (SNR)** : 6 decimal places, assigned by the manufacturer.
- **Spare (SP)** : 1 decimal place.

Thus, IMEI = TAC + FAC + SNR + SP. It uniquely characterizes a mobile station and gives clues about the manufacturer and the date of manufacturing.

International Mobile Subscriber Identity (IMSI)

Every registered user has an original International Mobile Subscriber Identity (IMSI) with a valid IMEI stored in their Subscriber Identity Module (SIM).

IMSI comprises of the following parts:

- **Mobile Country Code (MCC)** : 3 decimal places, internationally standardized.
- **Mobile Network Code (MNC)** : 2 decimal places, for unique identification of mobile network within the country.
- **Mobile Subscriber Identification Number (MSIN)** : Maximum 10 decimal places, identification number of the subscriber in the home mobile network.

Mobile Subscriber ISDN Number (MSISDN)

The authentic telephone number of a mobile station is the Mobile Subscriber ISDN Number (MSISDN). Based on the SIM, a mobile station can have many MSISDNs, as each subscriber is assigned with a separate MSISDN to their SIM respectively.

Listed below is the structure followed by MSISDN categories, as they are defined based on international ISDN number plan:

- **Country Code (CC)**: Up to 3 decimal places.
- **National Destination Code (NDC)**: Typically 2-3 decimal places.
- **Subscriber Number (SN)**: Maximum 10 decimal places.

Mobile Station Roaming Number (MSRN)

Mobile Station Roaming Number (MSRN) is an interim location dependent ISDN number, assigned to a mobile station by a regionally responsible Visitor Location Register (VLA). Using MSRN, the incoming calls are channelled to the MS.

The MSRN has the same structure as the MSISDN.

- **Country Code (CC)**: of the visited network.
- **National Destination Code (NDC)**: of the visited network.
- **Subscriber Number (SN)**: in the current mobile network.

Location Area Identity (LAI)

Within a PLMN, a Location Area identifies its own authentic Location Area Identity (LAI). The LAI hierarchy is based on international standard and structured in a unique format as mentioned below:

- **Country Code (CC)**: 3 decimal places.
- **Mobile Network Code (MNC)**: 2 decimal places.
- **Location Area Code (LAC)**: maximum 5 decimal places or maximum twice 8 bits coded in hexadecimal (LAC < FFFF).

Temporary Mobile Subscriber Identity (TMSI)

Temporary Mobile Subscriber Identity (TMSI) can be assigned by the VLR, which is responsible for the current location of a subscriber. The TMSI needs to have only local significance in the area handled by the VLR. This is stored on the network side only in the VLR and is not passed to the Home Location Register (HLR).

Together with the current location area, the TMSI identifies a subscriber uniquely. It can contain up to 4 × 8 bits.

Local Mobile Subscriber Identity (LMSI)

Each mobile station can be assigned with a Local Mobile Subscriber Identity (LMSI), which is an original key, by the VLR. This key can be used as the auxiliary searching key for each mobile station within its region. It can also help accelerate the database access. An LMSI is assigned if the mobile station is registered with the VLR and sent to the HLR. LMSI comprises of four octets (4x8 bits).

Cell Identifier (CI)

Using a Cell Identifier (CI) (maximum 2 × 8) bits, the individual cells that are within an LA can be recognized. When the Global Cell Identity (LAI + CI) calls are combined, then it is uniquely defined.

GSM - Protocol Stack

GSM architecture is a layered model that is designed to allow communications between two different systems. The lower layers assure the services of the upper-layer protocols. Each layer passes suitable notifications to ensure the transmitted data has been formatted, transmitted, and received accurately.

MS Protocols

Based on the interface, the GSM signaling protocol is assembled into three general layers:

- **Layer 1** : The physical layer. It uses the channel structures over the air interface.
- **Layer 2** : The data-link layer. Across the Um interface, the data-link layer is a modified version of the Link access.
- **Layer 3** : GSM signaling protocol's third layer is divided into three sub layers:
    - Radio Resource Management (RR),
    - Mobility Management (MM), and
    - Connection Management (CM).

MS to BTS Protocols

The RR layer is the lower layer that manages a link, both radio and fixed, between the MS and the MSC. For this formation, the main components involved are the MS, BSS, and MSC. The responsibility of the RR layer is to manage the RR-session, the time when

a mobile is in a dedicated mode, and the radio channels including the allocation of dedicated channels.

The MM layer is stacked above the RR layer. It handles the functions that arise from the mobility of the subscriber, as well as the authentication and security aspects. Location management is concerned with the procedures that enable the system to know the current location of a powered-on MS so that incoming call routing can be completed.

The CM layer is the topmost layer of the GSM protocol stack. This layer is responsible for Call Control, Supplementary Service Management, and Short Message Service Management. Each of these services are treated as individual layer within the CM layer. Other functions of the CC sub layer include call establishment, selection of the type of service (including alternating between services during a call), and call release.

BSC Protocols

The BSC uses a different set of protocols after receiving the data from the BTS. The Abis interface is used between the BTS and BSC. At this level, the radio resources at the lower portion of Layer 3 are changed from the RR to the Base Transceiver Station Management (BTSM). The BTS management layer is a relay function at the BTS to the BSC.

The RR protocols are responsible for the allocation and reallocation of traffic channels between the MS and the BTS. These services include controlling the initial access to the system, paging for MT calls, the handover of calls between cell sites, power control, and call termination. The BSC still has some radio resource management in place for the frequency coordination, frequency allocation, and the management of the overall network layer for the Layer 2 interfaces.

To transit from the BSC to the MSC, the BSS mobile application part or the direct application part is used, and SS7 protocols is applied by the relay, so that the MTP 1-3 can be used as the prime architecture.

MSC Protocols

At the MSC, starting from the BSC, the information is mapped across the A interface to the MTP Layers 1 through 3. Here, Base Station System Management Application Part (BSS MAP) is said to be the equivalent set of radio resources. The relay process is finished by the layers that are stacked on top of Layer 3 protocols, they are BSS MAP/DTAP, MM, and CM. This completes the relay process. To find and connect to the users across the network, MSCs interact using the control-signaling network.

Location registers are included in the MSC databases to assist in the role of determining how and whether connections are to be made to roaming users.

Each GSM MS user is given a HLR that in turn comprises of the user's location and subscribed services. VLR is a separate register that is used to track the location of a user. When the users move out of the HLR covered area, the VLR is notified by the MS to find the location of the user. The VLR in turn, with the help of the control network, signals the HLR of the MS's new location. With the help of location information contained in the user's HLR, the MT calls can be routed to the user.

Model-View-Controller (MVC)

A variety of design patterns are used to develop the WebSphere Commerce framework. Any solution extending from WebSphere Commerce should adhere to these high-level design patterns.

Model-View-Controller design pattern

The model-view-controller (MVC) design pattern specifies that an application consist of a data model, presentation information, and control information. The pattern requires that each of these be separated into different objects.

Command design pattern

WebSphere Commerce Server accepts requests from browser-based thin-client applications; from applications such as the Sales Center; and remote applications. For example, a request may come from a remote procurement system, or from another commerce server.

Display design pattern

Display pages return a response to a client. Typically, display pages are implemented as JSP pages.

Model-View-Controller (MVC)

The model-view-controller (MVC) design pattern specifies that an application consist of a data model, presentation information, and control information. The pattern requires that each of these be separated into different objects.

MVC is more of an architectural pattern, but not for complete application. MVC mostly relates to the UI / interaction layer of an application.

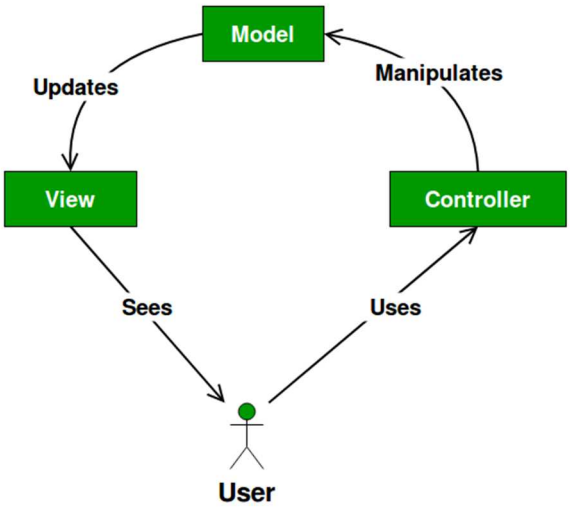
MVC Design Pattern

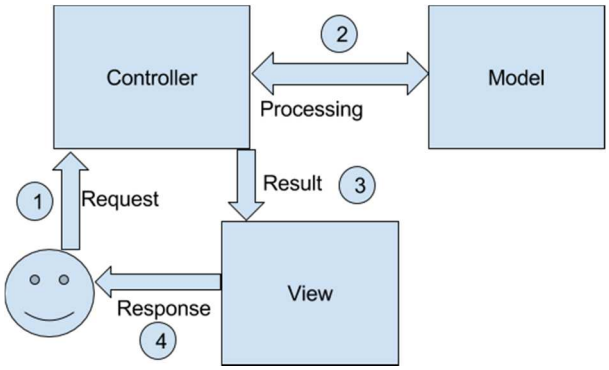
With MVC design pattern, we have following components on which our design depends:

The model which is transferred from one layer to the other.

The View which is responsible to show the data present in the application.

The controller is responsible to accept a request from a user, modify a model and send it to the view which is shown to the user.





The *model* (for example, the data information) contains only the pure application data; it contains no logic describing how to present the data to a user.

The *view* (for example, the presentation information) presents the model's data to the user. The view knows how to access the model's data, but it does not know what this data means or what the user can do to manipulate it.

Finally, the *controller* (for example, the control information) exists between the view and the model. It listens to events triggered by the view (or another external source) and executes the appropriate reaction to these events. In most cases, the reaction is to

call a method on the model. Since the view and the model are connected through a notification mechanism, the result of this action is then automatically reflected in the view.