



**SCHOOL OF COMPUTING**  
**DEPARTMENT OF COMPUTER SCIENCE**  
**AND ENGINEERING**

**Mobile Application Development SBS1603**

## UNIT 1

**Introduction to Mobile Computing - Introduction to Android Development Environment - Factors in Developing Mobile Applications - Mobile Software Engineering - Frameworks and Tools - Generic UI Development - Android User**

### **Introduction to Mobile Computing**

**Mobile application development** is the process by which **application software** is **developed** for small low-power **handheld devices** such as **personal digital assistants**, **enterprise digital assistants** or **mobile phones**. These applications are either **pre-installed** on phones during manufacture, or downloaded by customers from various **mobile software distribution platforms**.

Mobile software is developed by using different platforms and programming languages based on the target mobile device. There are many different hardware components found in mobile devices so their applications are developed using different software architectures.

Most of the methodologies in use are based on the model-driven approach which has three different views of the application development process: (1) the application itself and its structure, (2) the business logic and (3) the graphical user interface of the application.

### **Weapons of Mobile Development**

- Hardware
- Computers
- Mobile Phones
- Framework
- Programming Language
- Integrated Development Environment
- Compiler

Simulator  
Monitors

## Platforms Available

1. **J2ME**  
Programming Language: Java  
Integrated Development Environment: Netbeans/Eclipse  
FrameWork: KVM  
Hardware Deployment: Multiple  
Installer Packaging Options: Jad/Jar
2. **iOS (Apple)**  
Programming Language: Objective C  
Integrated Development Environment: Xcode  
FrameWork: Cocoa Touch  
Hardware Deployment: iPhone, iPad, iPod  
Installer Packaging Options: .app/.ipa  
Development Tool Cost
3. **Android**  
Programming Language: Java  
Integrated Development Environment: Eclipse/Netbeans  
FrameWork: Dalvik VM  
Hardware Deployment: Android Only  
Installer Packaging Options: .apk  
Development Tool Cost
4. **BlackBerry**  
Programming Language: Java  
Integrated Development Environment: JDE - BlackBerry Java Development Environment  
FrameWork: Dalvik VM  
Hardware Deployment: Android Only  
Installer Packaging Options: .alx, .cod
5. **Windows Mobile**  
Programming Language: C#, VB.NET, Basic4ppc  
Integrated Development Environment: Visual Studio 2008, 2005, 2003, Basic4ppc IDE  
FrameWork: .Net Compact Framework  
Hardware Deployment: Windows Mobiles  
Installer Packaging Options: .ota/.cab

## Typical Applications

---

- Location Based Applications
- Mobile workforce & field workers
- Mobile Coupons & Discounts
- Remote Employees Collaboration

- GPS Tracking & Dispatching
- Mobile Money Transfers
- Content Delivery & Publishing

## Platforms supported

---

- **iOS** - platform developed by the company Apple supported by mobile devices: iPhone, iPod Touch, iPad. Among standard applications iOS are Google maps, YouTube, iTunes, iBooks etc. iOS is based on the C programming language.
- **BlackBerry** – compact operating system for mobile development with the basic applications like text messaging, internet faxing, web browsing etc. Blackberry works on some devices – Smart phones and communicators – produced by the company RIM. Many BlackBerry Smart phones have full-size QWERTY keyboard, which eases the work with texts.
- **Android** - an open operating system for cell phones and smart books based on Linux. Android is supported by over 34 major software, hardware and telecoms companies including Google, HTC, Motorola, Qualcomm, and T-Mobile. Android apps are mostly written in the Java programming language.
- **Windows Mobile / Windows Phone 7** – operating system supported by the company Microsoft based on Microsoft Win32 API. Windows mobile can work on some devices including Pocket PC, smart phones, and communicators. The current version for Windows phones is somewhat analogical to the desktop Windows versions (functions, lay-out).
- **Symbian platform** - It's specifics are memory economy and high programs speed. The main language of applications development is C++. Symbian foundation includes 40 companies: Nokia, Sony Ericsson, Siemens, Panasonic, Fujitsu, Samsung, Sony, Sony Ericsson, Motorola etc. The Symbian Foundation maintains the code for the open source software platform based on Symbian OS and software assets contributed by Nokia, NTT DOCOMO, and Sony Ericsson.
- **Java ME** – This platform mostly supports portable applications. It is successfully used on the most cell phones and portative organizers. According to the information of the company Sun Microsystems, 2004 there were produced 579 million mobile devices supported by Java. It makes the technology java ME to a dominant java technology in the world.
- **BREW** Binary Runtime Environment for Wireless (BREW) —a platform for mobile application development. The big advantage of the platform is the ease of applications transportation among different devices supporting BREW. BREW allows realization of many applications like games, message and data exchange etc. Brew applications are developed on the base of the programming languages C/C++.

## Characteristics of Mobile Applications Development

**1. Mobile devices are deeply personal.** The phone is something that belongs to just one individual. It's private, personal, and likely to be carried with you everywhere you go. We like to customize our phones and they have become a symbol of social status. The phone I carry, and the content I have, become part of my personal brand.

**2. Mobile devices are hyper-social.** The phone holds all of your contacts and allows you to do much more with them. Basic communications include voice calling, text messaging (sending a message of 140 characters or less. SMS or Short Message Service was the first Twitter), multimedia messaging (sending video or photos along with text messages), and e-mail. Beyond the basics, you may want to consider online presence, physical proximity, or relationship strength, to allow for more “contextual communications.”

**3. Mobile devices are location-aware.** Even if you don’t know exactly where you are, your phone does! It can also help you discover what is located nearby, get directions, or even interpret what is right in front of you. If your friends opt-in, you can display their present location. This is one of the most powerful features of mobile devices and one that, until recently, was reserved for native applications. However, the most recent versions of Fireworks, Chrome, and the Android browser, now have location APIs as well.

**4. Mobile devices promote quick focused usage.** Assuming that one is on the move, I will stop to navigate my phone when I need a vital bit of information like my shopping list, current traffic, flight arrival times, stock prices, or recent messages. The exception to this is entertainment. More and more, mobiles fill those empty spaces when we’re standing in line, waiting, or sitting for long idle periods — we reach for our phones to watch our favorite podcasts, scan tweets, listen to music, or immerse ourselves in a good film.

**5. Mobile devices are sometimes connected.** When planning your mobile offering, it’s important to develop for both the online and offline scenarios. Sometimes your users will be riding the subway, on a plane, traveling abroad or otherwise unable to connect to the Internet, their home network, or both. Although unlimited data plans do exist, they’re not as prevalent as you may think. So if you want to avoid costly phone bills you need to consider how users will engage with your content when they’re offline.

**6. Mobile devices support a spontaneous lifestyle.** Carrying all of your PIM (personal information management) data means you can respond to events as they unfold and share new data in real time. For example, if traffic is blocked, I can meet my party at a different restaurant. If I’m standing in line at the cinema and tickets sell out, I can quickly locate an alternative and have my friends meet me there.

**7. Connectivity.** Apps are always online as the device is constantly logged in to the mobile network. This allows user specific information or notifications being pushed to the App as they are available. In combination with the ubiquity of mobile applications, this is the most important characteristic. A service, that must be called actively to get a reaction is useless in a mobile environment. And with the growing number of Apps on each smartphone, this push-functionality becomes critical to keep an App in the users mind.

## **8. Convenience**

An emotional design and a simple (one-handed) handling guarantee a high acceptance. A good App can do its job in different contexts and fast varying situations (changing environmental light and noise, unsteady movement of the device, etc.). So the information architecture and the

overall usability must be planned with care to create a fitting and joyful interaction flow. Of course good content also counts to convenience. Analyzing the users' needs and creating a useful idea out of it is still essential.

## **9. Localization**

Localization and the possibility to offer location-based information is a key feature that makes mobility vivid and practical. It separates the wheat from the chaff as it embeds the App to the users' context. Sure, this feature might not make sense for every App but localization must not always be thought of as the big thing. Just think of automatically associate the location with a note or photo or by just limiting possible options or by sorting places. It can be useful in little, just creating a good experience for the user.

## **10. Reachability**

Reachability covers a more social attribute given by the nature of mobile applications itself. A good App can really be used – and more important makes sense – anywhere at any time. The core of mobile devices is to be used anywhere at any time. The same is true for Apps where reachability has become availability. Not in sense of usage, but in sense of updated information and perpetual usefulness.

## **11. Security**

Security has several facets. The data transferred over the network must be encrypted through the carrier network. As some Apps sync data with online, web-based applications, the storage of this data on the server must also be secured. Another aspect concerns the data on the device itself. I don't want anybody playing around with my mobile phone getting access to my bank account data. Mobility is delicate, and so is the data aggregated and generated in this context.

## **12. Personalization**

Creating personalized content based on individual usage or context is another characteristic. It builds on all previous characteristics as it is a kind of melting down of all of them. I want my App fitting my needs and I want my App behaving like I want it to do. This need covers not only personalized content but also control over data stored, shared or used for further actions. The option of turning localization on or off is true personalization. An individual background or personal categories are convenience.

## **Benefits**

- Most of these devices have **easy functionality and use**
- Mobile Application development is a **key part** in the evolution of mobile phones
  - How does this application help me?
  - Does it make my work easier?
  - Does it enable me to easily use it?
- Very popular mobile applications
  - Game applications

- Weather information
  - Sports scores and updates applications
  - The Bible, Koran
  - The Dictionary
  - Fitness and health applications etc...
- There are also **free trial applications** that are in development stages
- Select an application that helps you have **functionality and style** as well, like
  - The blackberry has a set of sophisticated and fabulous features
  - The stock viewer enables you to keep track of your stocks and business listings
  - iPhone's breathtaking application is the text n drive
- The mobile application development industry truly is bringing **innovative solutions to the technological realm.**

## Mobile Application Architecture

### Overview

A mobile application will normally be structured as a multi-layered application consisting of user experience, business, and data layers. When developing a mobile application, you may choose to develop a thin Web-based client or a rich client. If you are building a rich client, the business and data services layers are likely to be located on the device itself. If you are building a thin client, the business and data layers will be located on the server. Figure 1 illustrates common rich client mobile application architecture.

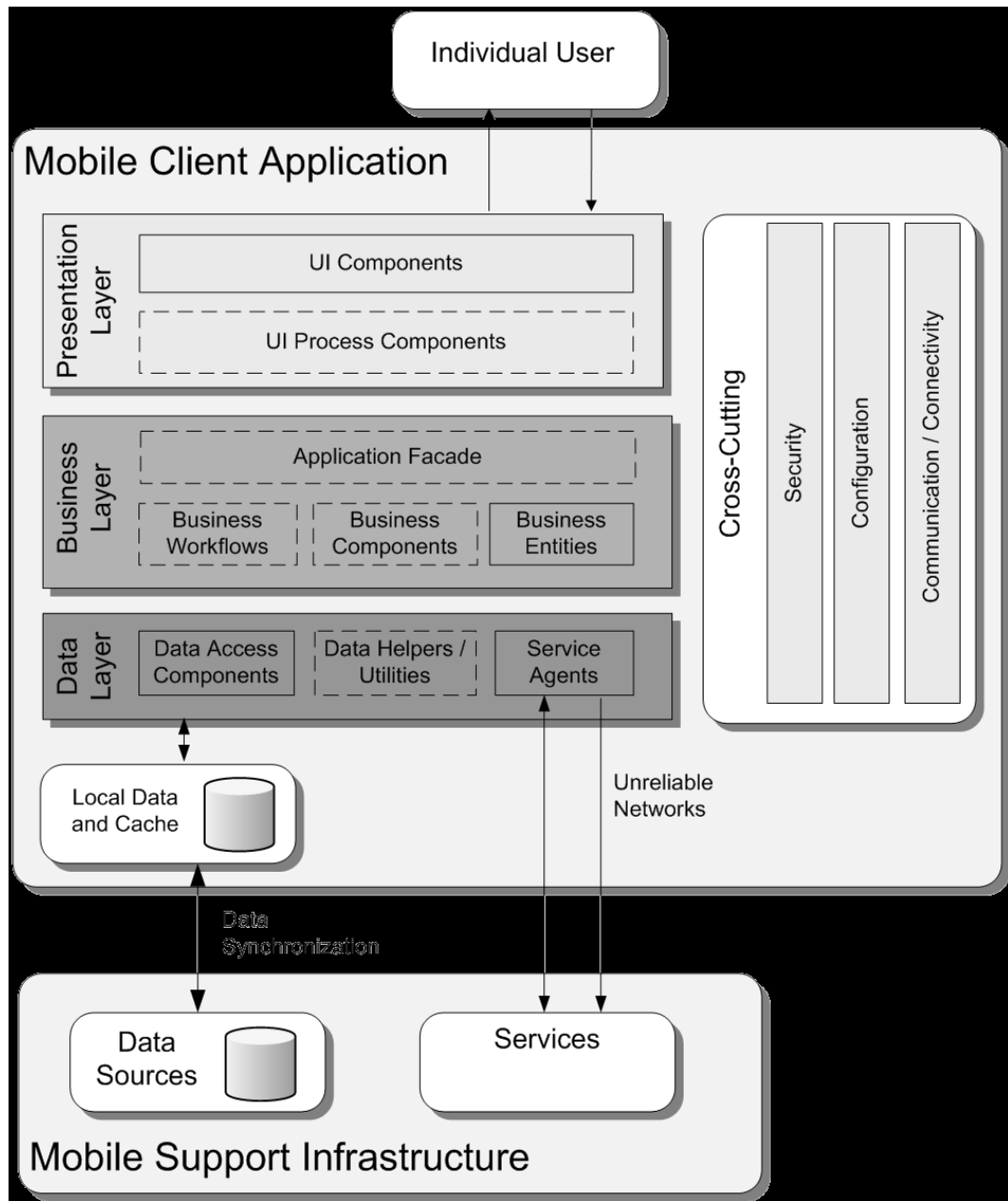


Figure 1 Common rich client mobile application architecture



Irrespective of applications, the design is decomposed into logical groupings of software components. This logical grouping is called Layers. Layers help to differentiate between the different kinds of tasks performed by the components, making it easier to create a design that supports reusability of components. Each logical layer contains a number of discrete component types grouped into sub layers, with each sub layer performing a specific type of task.

An application can consist of basically three layers design. They are,

- **Presentation layer.** This layer contains the user oriented functionality responsible for managing user interaction with the system, and generally consists of components that provide a common bridge into the core business logic encapsulated in the business layer.
- **Business layer.** This layer implements the core functionality of the system, and encapsulates the relevant business logic. It generally consists of components, some of which may expose service interfaces that other callers can use.
- **Data layer.** This layer provides access to data hosted within the boundaries of the system, and data exposed by other networked systems; perhaps accessed through services. The data layer exposes generic interfaces that the components in the business layer can consume.

### **Presentation Layer:**

The presentation layer contains the components that implement and display the user interface and manage user interaction. This layer includes controls for user input and display, in addition to components that organize user interaction.

The presentation layer will usually include the following:

- **User Interface components.** These are the application's visual elements used to display information to the user and accept user input.
- **Presentation Logic components.** Presentation logic is the application code that defines the logical behavior and structure of the application in a way that is independent of any specific user interface implementation. When implementing the Separated Presentation pattern, the presentation logic components may include Presenter, Presentation Model, and ViewModel components. The presentation layer may also include Presentation Layer Model components that encapsulate the data from your business layer, or Presentation Entity components that encapsulate business logic and data in a form that is easily consumable by the presentation layer.

### **Business Layer**

The business layer will usually include the following:

- **Application façade.** This optional component typically provides a simplified interface to the business logic components, often by combining multiple business operations into a single operation that makes it easier to use the business logic. It reduces dependencies

because external callers do not need to know details of the business components and the relationships between them.

- **Business Logic components.** Business logic is defined as any application logic that is concerned with the retrieval, processing, transformation, and management of application data; application of business rules and policies; and ensuring data consistency and validity. To maximize reuse opportunities, business logic components should not contain any behavior or application logic that is specific to a use case or user story. Business logic components can be further subdivided into the following two categories:
  - **Business Workflow components.** After the UI components collect the required data from the user and pass it to the business layer, the application can use this data to **perform a business process**. Many business processes involve multiple steps that must be performed in the correct order, and may interact with each other through an orchestration. Business workflow components define and coordinate long running, multistep business processes, and can be implemented using business process management tools. They work with business process components that instantiate and perform operations on workflow components.
  - **Business Entity components.** Business entities, or more generally business objects, **encapsulate the business logic and data** necessary to represent real world elements, such as Customers or Orders, within your application. They store data values and expose them through properties; contain and manage business data used by the application; and provide stateful programmatic access to the business data and related functionality. Business entities also validate the data contained within the entity and encapsulate business logic to ensure consistency and to implement business rules and behavior.

## Data Layer

The data layer may include the following:

- **Data Access components.** These components abstract the logic required to access the underlying data stores. They centralize common data access functionality in order to make the application easier to configure and maintain. Some data access frameworks may require the developer to identify and implement common data access logic in separate reusable helper or utility data access components. Other data access frameworks, including many Object/Relational Mapping (O/RM) frameworks, implement such components automatically, reducing the amount of data access code that the developer must write.
- **Service agents.** When a business component must access data provided by an external service, you might need to implement code to manage the semantics of communicating with that particular service. Service agents implement data access components that isolate the varying requirements for calling services from your application, and may provide additional services such as caching, offline support, and basic mapping between the format of the data exposed by the service and the format your application requires.

## Crosscutting concerns

The majority of applications design will contain common functionality that spans layers and tiers. This functionality typically supports operations such authentication, authorization, caching, communication, exception management, logging and instrumentation, and validation. Such functionality is generally described as *crosscutting concerns* because it affects the entire application, and should be centralized in one location in the code where possible.

Some common functionality is

### ***Authentication***

Failure to design and implement a good authentication strategy can leave your application vulnerable to spoofing attacks, dictionary attacks, session hijacking, and other types of attacks. Consider the following guidelines when designing an authentication strategy:

- Identify your trust boundaries and authenticate users and calls across the trust boundaries. Consider that calls might need to be authenticated from the client as well as from the server (mutual authentication).
- Enforce the use of strong passwords or password phrases.
- If you have multiple systems within the application or users must be able to access multiple applications with the same credentials, consider a single sign-on strategy.
- Do not transmit passwords over the wire in plain text, and do not store passwords in a database or data store as plain text. Instead, store a hash of the password.

### ***Authorization***

Failure to design and implement a good authorization strategy can make your application vulnerable to information disclosure, data tampering, and elevation of privileges. Consider the following guidelines when designing an authorization strategy:

- Identify your trust boundaries and authorize users and callers across the trust boundary.
- Protect resources by applying authorization to callers based on their identity, groups, or roles. Minimize granularity by limiting the number of roles you use where possible.
- Consider using role-based authorization for business decisions. Role-based authorization is used to subdivide users into groups (roles) and then set permissions on each role rather than on individual users. This eases management by allowing you to administer a smaller set of roles rather than a larger set of users.
- Consider using resource-based authorization for system auditing. Resource-based authorization sets permissions on the resource itself; for example, an access control list (ACL) on a Windows resource uses the identity of the original caller to determine access rights to the resource. If you use resource-based authorization in WCF, you must to impersonate the original caller through the client or presentation layer, through the WCF service layer, and to the business logic code that accesses the resource.
- Consider using claims-based authorization when you must support federated authorization based on a mixture of information such as identity, role, permissions, rights, and other factors. Claims-based authorization provides additional layers of abstraction that make it easier to separate authorization rules from the authorization and authentication mechanism. For example, you can authenticate a user with a certificate or

with username/password credentials and then pass that claim-set to the service to determine access to resources.

### ***Caching***

Caching can improve the performance and responsiveness of your application. However, a poorly designed caching strategy can degrade performance and responsiveness. You should use caching to optimize reference data lookups, avoid network round trips, and avoid unnecessary and duplicate processing. When you implement caching, you must decide when to load the cache data and how and when to remove expired cached data. Try to preload frequently used data into the cache asynchronously or by using a batch process to avoid client delays. Consider the following guidelines when designing a caching strategy:

- Choose an appropriate location for the cache. If application is deployed in Web farm, avoid using local caches that must be synchronized;
- Consider caching data in a ready-to-use format when working with an in-memory cache. For example, use a specific object instead of caching raw database data. Consider using Microsoft Velocity to implement in-memory caching.
- Do not cache volatile data, and do not cache sensitive data unless you encrypt it.
- Do not depend on data still being in your cache; it may have been removed. Implement a mechanism to handle cache failures, perhaps by reloading the item from the source.
- Be especially careful when accessing the cache from multiple threads. If you are using multiple threads, ensure that all access to the cache is thread-safe to maintain consistency.

### ***Communication***

Communication is concerned with the interaction between components across layers and tiers. The mechanism you choose depends on the deployment scenarios your application must support. Consider the following guidelines when designing communication mechanisms:

- Consider using message-based communication when crossing physical or process boundaries; and object-based communication when in process (when crossing only logical boundaries). To reduce round trips and improve communication performance across physical and process boundaries, design coarse-grained (chunky) interfaces that communicate less often but with more information in each communication. However, where appropriate, consider exposing a fine-grained (chatty) interface for use by in process calls and wrapping these calls in a coarse-grained façade for use by processes that access it across physical or process boundaries.
- If your messages do not need to be received in a specific order and do not have dependencies on each other, consider using asynchronous communication to avoid blocking processing or UI threads.
- Consider using Microsoft Message Queuing to queue messages for later delivery in case of system or network interruption or failure. Message Queuing can perform transacted message delivery and supports reliable once-only delivery.
- Choose an appropriate transport protocol, such as HTTP for Internet communication and TCP for intranet communication. Consider how you will determine the appropriate message exchange patterns, connection based or connectionless communication, reliability guarantees (such as service level agreements), and authentication mechanism.

- Ensure that you protect messages and sensitive data during communication by using encryption, digital certificates, and channel security features.

### ***Configuration Management***

Designing a good configuration management mechanism is important for the security and flexibility of your application. Failure to do so can make your application vulnerable to a variety of attacks, and also leads to an administrative overhead for your application. Consider the following guidelines when designing for configuration management:

- Carefully consider which settings must be externally configurable. Verify that there is an actual business need for each configurable setting, and provide the minimal configuration options necessary to meet these requirements. Excessive configurability can result in systems that are more complicated, and may leave the system vulnerable to security breaches and malfunctions due to incorrect configuration.
- Decide if you will store configuration information centrally and have it downloaded or applied to users at startup (for example, through Active Directory Group Policy). Consider how you will restrict access to your configuration information. Consider using least privileged process and service accounts, and encrypt sensitive information in your configuration store.
- Categorize the configuration items into logical sections based on whether they apply to users, application settings, or environmental settings. This makes it easier to divide configuration when you must support different settings for different sets of users, or multiple environments.
- Categorize the configuration items into logical sections if your application has multiple tiers. If your server application runs in a Web farm, decide which parts of the configuration are shared and which parts are specific to the machine on which the application is running. Then choose an appropriate configuration store for each section.
- Provide a separate administrative UI for editing configuration information.

### ***Exception Management***

Designing a good exception management strategy is important for the security and reliability of your application. Failure to do so can make it very difficult to diagnose and solve problems with your application. It can also leave your application vulnerable to Denial of Service (DoS) attacks, and it may reveal sensitive and critical information. Raising and handling exceptions is an expensive process, so it is important that the design also takes into account performance issues. A good approach is to design a centralized exception management mechanism for your application, and to consider providing access points within your exception management system (such as WMI events) to support enterprise level monitoring systems such as Microsoft System Center. Consider the following guidelines when designing an exception management strategy:

- Design an appropriate exception propagation strategy that wraps or replaces exceptions, or adds extra information as required. For example, allow exceptions to bubble up to boundary layers where they can be logged and transformed as necessary before passing them to the next layer. Consider including a context identifier so that related exceptions can be associated across layers to assist in performing root cause analysis of errors and faults. Also ensure that the design deals with unhandled exceptions. Do not catch internal

exceptions unless you can handle them or you must add more information, and do not use exceptions to control application flow.

- Ensure that a failure does not leave the application in an unstable state, and that exceptions do not allow the application to reveal sensitive information or process details. If you cannot guarantee correct recovery, allow the application to halt with an unhandled exception in preference to leaving it running in an unknown and possibly corrupted state.
- Design an appropriate logging and notification strategy for critical errors and exceptions that stores sufficient details about the exception to allow support staff to recreate the scenario, but does not reveal sensitive information in exception messages and log files.

### ***Logging and Instrumentation***

Designing a good logging and instrumentation strategy is important for the security and reliability of your application. Failure to do so can make your application vulnerable to repudiation threats, where users deny their actions, and log files may be required for legal proceedings to prove wrongdoing. You should audit and log activity across the layers of your application, which can help to detect suspicious activity and provide early indication of a serious attack. Auditing is usually considered most authoritative if the audits are generated at the precise time of resource access, and by the same routines that access the resource. Instrumentation can be implemented by using performance counters and events to give administrators information about the state, performance, and health of an application. Consider the following guidelines when designing a logging and instrumentation strategy:

- Design a centralized logging and instrumentation mechanism that captures system- and business-critical events. Avoid logging and instrumentation that is too fine grained, but consider additional logging and instrumentation that is configurable at run time for obtaining extra information and to aid debugging.
- Create secure log file management policies. Do not store sensitive information in the log files, and protect log files from unauthorized access. Consider how you will access and pass auditing and logging data securely across application layers, and ensure that you suppress but correctly handle logging failures.
- Consider allowing your log sinks, or trace listeners, to be configurable so that they can be modified at run time to meet deployment environment requirements.

### ***State Management***

State management concerns the persistence of data that represents the state of a component, operation, or step in a process. State data can be persisted using different formats and stores. The design of a state management mechanism can affect the performance of your application; maintaining even small volumes of state information can adversely affect performance and the ability to scale out your application. You should only persist data that is required, and you must understand the options that are available for managing state. Consider the following guidelines when designing a state management mechanism:

- Keep your state management as lean as possible; persist the minimum amount of data required to maintain state.
- Make sure that your state data is serializable if it must be persisted or shared across process and network boundaries.

- Choose an appropriate state store. Storing state in process and in memory is the technique that can offer the best performance, but only if your state does not have to survive process or system restarts. Persist your state to a local disk or local SQL Server if you want it available after a process or system restart. Consider storing state in a central location such as a central SQL Server if state is critical for your application, or if you want to share state between several machines.

### ***Validation***

Designing an effective validation mechanism is important for the usability and reliability of your application. Failure to do so can leave your application open to data inconsistencies, business rule violations, and a poor user experience. In addition, failing to adequately validate input may leave your application vulnerable to security issues such as cross-site scripting attacks, SQL injection attacks, buffer overflows, and other types of input attacks. Unfortunately there is no standard definition that can differentiate valid input from malicious input. In addition, how your application actually uses the input influences the risks associated with exploitation of the vulnerability. Consider the following guidelines when designing a validation mechanism:

- Whenever possible, design your validation system to use allow lists that define specifically what is acceptable input, rather than trying to define what comprises invalid input. It is much easier to widen the scope of an allow list later than it is to narrow a block list.
- Do not rely on only client-side validation for security checks. Instead, use client-side validation to give the user feedback and improve the user experience. Always use server-side validation to check for incorrect or malicious input because client-side validation can be easily bypassed.
- Centralize your validation approach in separate components if it can be reused, or consider using a third-party library such as the patterns & practices Enterprise Library Validation Block. Doing so will allow you to apply a consistent validation mechanism across the layers and tiers of your application.
- Be sure to constrain, reject, and sanitize user input. In other words, assume that all user input is malicious. Identify your trust boundaries and validate all input data for length, format, type, and range as it crosses trust boundaries.

### **Design Considerations**

The following design guidelines provide information about different aspects you should consider when designing a mobile application. Follow these guidelines to ensure that your application meets your requirements and performs efficiently in scenarios common to mobile applications.

- **Decide if you will build a rich client, a thin Web client, or rich Internet application (RIA).** If your application requires local processing and must work in an occasionally connected scenario, consider designing a rich client. A rich client application will be more complex to install and maintain. If your application can depend on server processing and will always be fully connected, consider designing a thin client. If your application requires a rich user interface (UI), only limited access to local resources, and must be portable to other platforms, design an RIA client.

- **Determine the device types you will support.** When choosing which device types to support, consider screen size, resolution (DPI), CPU performance characteristics, memory and storage space, and development tool environment availability. In addition, factor in user requirements and organizational constraints. You may require specific hardware such as GPS or a camera and this may impact not only your application type, but also your device choice.
- **Design considering occasionally connected limited-bandwidth scenarios when required.** If your mobile device is a stand-alone device, you will not need to account for connection issues. When network connectivity is required, Mobile applications should handle cases when a network connection is intermittent or not available. It is vital in this case to design your caching, state management, and data-access mechanisms with intermittent network connectivity in mind. Batch communications for times of connectivity. Choose hardware and software protocols based on speed, power consumption, and “chattiness,” and not just on ease of programming.
- **Design a UI appropriate for mobile devices, taking into account platform constraints.** QMobile devices require a simpler architecture, simpler UI, and other specific design decisions in order to work within the constraints imposed by the device hardware. Keep these constraints in mind and design specifically for the device instead of trying to reuse the architecture or UI from a desktop or Web application. The main constraints are memory, battery life, ability to adapt to difference screen sizes and orientations, security, and network bandwidth.
- **Design a layered architecture appropriate for mobile devices that improves reuse and maintainability.** Depending on the application type, multiple layers may be located on the device itself. Use the concept of layers to maximize separation of concerns, and to improve reuse and maintainability for your mobile application. However, aim to achieve the smallest footprint on the device by simplifying your design compared to a desktop or Web application.
- **Design considering device resource constraints such as battery life, memory size, and processor speed.** Every design decision should take into account the limited CPU, memory, storage capacity, and battery life of mobile devices. Battery life is usually the most limiting factor in mobile devices. Backlighting, reading and writing to memory, wireless connections, specialized hardware, and processor speed all have an impact on the overall power usage. When the amount of memory available is low, the Microsoft® Windows Mobile® operating system may ask your application to shut down or sacrifice cached data, slowing program execution. Optimize your application to minimize its power and memory footprint while considering performance during this process.

## Overview of Mobile communication Infrastructure

- **1G Wireless networks**
  - Data rate 2.4 kbps
  - Basic voice service
  - Analog based protocols
- **2G Wireless networks**
  - Data rate 64kbps
  - Roaming services
  - TDMA, CDMA technology used
  - Designed for voice service
  - Improved coverage and capacity



- First digital standard (GSM, CDMA)
- **2.5G (GPRS)**
  - Implemented a packet-switched domain in addition to the circuit-switched domain
  - The first major step in the evolution of GSM networks to 3G occurred with the introduction of General Packet Radio Service (GPRS)
- **2.75G (EDGE)**
  - Enhanced Data rates for GSM Evolution (EDGE), Enhanced GPRS (EGPRS)
  - EDGE was deployed on GSM networks beginning in 2003.
- **3G Wireless networks**
  - Data rate 200 kbps
  - 3.5G and 3.75G with high data rate
  - Designed for voice with some data (multimedia, text, internet)
  - First mobile broadband 2000 kbps
- **4G Wireless networks**
  - Data rate 1,00,000 kbps
  - Global roaming
  - Designed primary for data
  - IP-based protocols
  - True mobile broadband
  - WiFi and WiMAX technology
  - **4G LTE** – Long-Term Evolution - High speed data for mobile phones and data terminals

## 2G network - GSM

Digital cellular phone beginning in commercial use based on 2nd generation technology what we called 2G. The well known 2G networking technology are GSM, CDMA, PCS etc.

- GSM stands for Global System for Mobile Communication and is an open, digital cellular technology used for transmitting mobile voice and data services.
- The GSM emerged from the idea of cell-based mobile radio systems at Bell Laboratories in the early 1970s.
- The GSM (formally, Groupe Speciale Mobile) is the name of a standardization group established in 1982 to create a common European mobile telephone standard.
- The 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.
- The 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:

- 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

### **GSM ( Global System for Mobile Communication) Architecture:**

The GSM network can be broadly divided into:

- Radio Subsystem (RSS)
- Network Switching Subsystem (NSS)
- Operation Support Subsystem (OSS)

### **Radio SubSystem(RSS):**

The radio subsystem is comprised of all the radio specific elements, i.e.

- **Mobile Station(MS):**

It consist of a

- **ME (Mobile Equipment):** It is a handheld and portable device and uniquely identified by an IMEI(International Mobile Equipment Identity).
- **SIM (Subscriber Identity Module):** It is responsible for storing the subscriber's identification number, contact list, the networks the subscriber is authorized to use, encryption keys etc.

- **Base Station Subsystem(BSS):**

BSS consists of a Base Station Controller (**BSC**) and one or more Base Transceiver Station (**BTS**).

Each BTS defines a cell size and includes radio antenna, radio transceiver and a link to a BSC. A GSM cell can have a radius of between 100m and 35km, depending on the environment.

BSC is responsible for reserving radio frequencies, manages hand off mobile unit from one cell to another within BSS, and control paging.

### **Network SubSystem (NSS):**

It provides link between cellular network and public switched telecommunications networks. The main functionalities are:

- Controls hand offs cells in different BSSs.
- Authenticates users and validates accounts.

Central element of NSS is **Mobile Switching Center (MSC)** which is supported by four databases that it controls.

**Home Location Register (HLR):**

It is used to store information about each subscriber that belongs to it.

**Visitor Location Register (VLR):**

It is used to determine the current position of the subscriber has entered.

**Authentication Center (AuC):**

It is used for authentication activities of the system.

**Equipment Identity Register (EIR):**

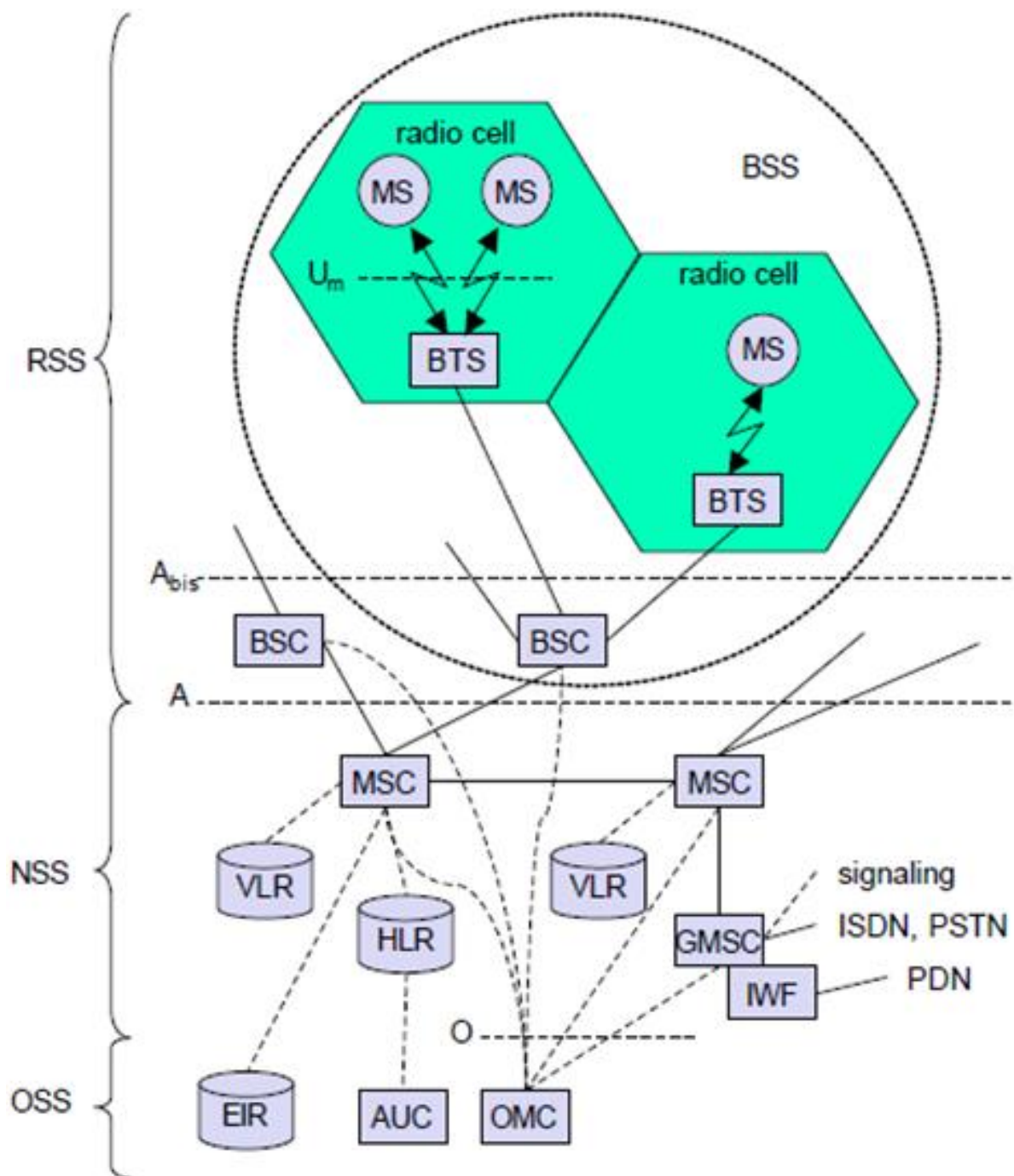
It keeps track of the type of equipment that exists at the mobile station i.e it can be IMEI number used for tracking of device.

**Operation Support Subsystem (OSS):**

The operations and maintenance center (OMC) is connected to all equipment in the switching system and to the BSC. The implementation of OMC is called the operation and support system (OSS).

Here are some of the OMC functions:

- Administration and commercial operation (subscription, end terminals, charging and statistics).
- Security Management.
- Network configuration, Operation and Performance Management.
- Maintenance Tasks.



## Advantages of GSM over Analog system

- Reduced RF transmission power and longer battery life.
- International roaming capability.
- Better security against fraud.
- Encryption capability for information security and privacy.

## Limitations of GSM

- No end-to-end encryption of user data
- Reduced concentration while driving
- Electromagnetic radiation
- Abuse of private data possible
- High complexity of the system
- Several incompatibilities within the GSM standards

## 2.5G - GPRS

### GPRS:

GPRS stands for **General Packet Radio System**. GPRS provides packet radio access for mobile Global System for Mobile Communications (GSM) and time-division multiple access (TDMA) users.

GPRS is important as a migration step toward third-generation (3G) networks and allows network operators to implement an IP-based core architecture for data applications, which will continue to be used and expanded for 3G services for integrated voice and data applications.

GPRS is a new bearer service for GSM that greatly improves and simplifies wireless access to packet data networks, e.g., to the Internet. It applies a packet radio principle to transfer user data packets in an efficient way between GSM mobile stations and external packet data networks. Packets can be directly routed from the GPRS mobile stations to packet switched networks.

Networks based on the Internet Protocol (IP) (e.g., the global Internet or private/corporate intranets) and X.25 networks are also supported in the current versions of GPRS.

The following three key features describe wireless packet data:

- **The always online feature:** Removes the dial-up process, making applications only one click away.
- **An upgrade to existing systems:** Operators do not have to replace their equipment; rather, GPRS is added on top of the existing infrastructure.
- **An integral part of future 3G systems:** GPRS is the packet data core network for 3G systems EDGE and WCDMA.

### Goals of GPRS:

GPRS is the first step toward an end-to-end wireless infrastructure and has the following goals:

- Open architecture

- Consistent IP services
- Same infrastructure for different air interfaces
- Integrated telephony and Internet infrastructure
- Leverage industry investment in IP
- Service innovation independent of infrastructure

## **Benefits of GPRS:**

### ***Higher Data Rate:***

Users of GPRS benefit from shorter access times and higher data rates. In conventional GSM, the connection setup takes several seconds and rates for data transmission are restricted to 9.6 kbit/s. GPRS in practice offers session establishment times below one second and ISDN-like data rates up to several ten kbit/s.

### **Easy Billing:**

GPRS packet transmission offers a more user friendly billing than that offered by circuit switched services. In circuit switched services, billing is based on the duration of the connection. This is unsuitable for applications with burst traffic. The user must pay for the entire airtime, even for idle periods when no packets are sent (e.g., when the user reads a Web page).

In contrast to this, with packet switched services, billing can be based on the amount of transmitted data. The advantage for the user is that he or she can be "online" over a long period of time but will be billed based on the transmitted data volume.

GPRS enables a variety of new and unique services to the mobile wireless subscriber. These mobile services have unique characteristics that provide enhanced value to customers. These characteristics include the following:

- **Mobility:** The ability to maintain constant voice and data communications while on the move
- **Immediacy** Allows subscribers to obtain connectivity when needed, regardless of location and without a lengthy login session.
- **Localization** Allows subscribers to obtain information relevant to their current location.

The combination of these characteristics provides a wide spectrum of possible applications that can be offered to mobile subscribers. In general, applications can be separated into two high-level categories: corporate and consumer. These include:

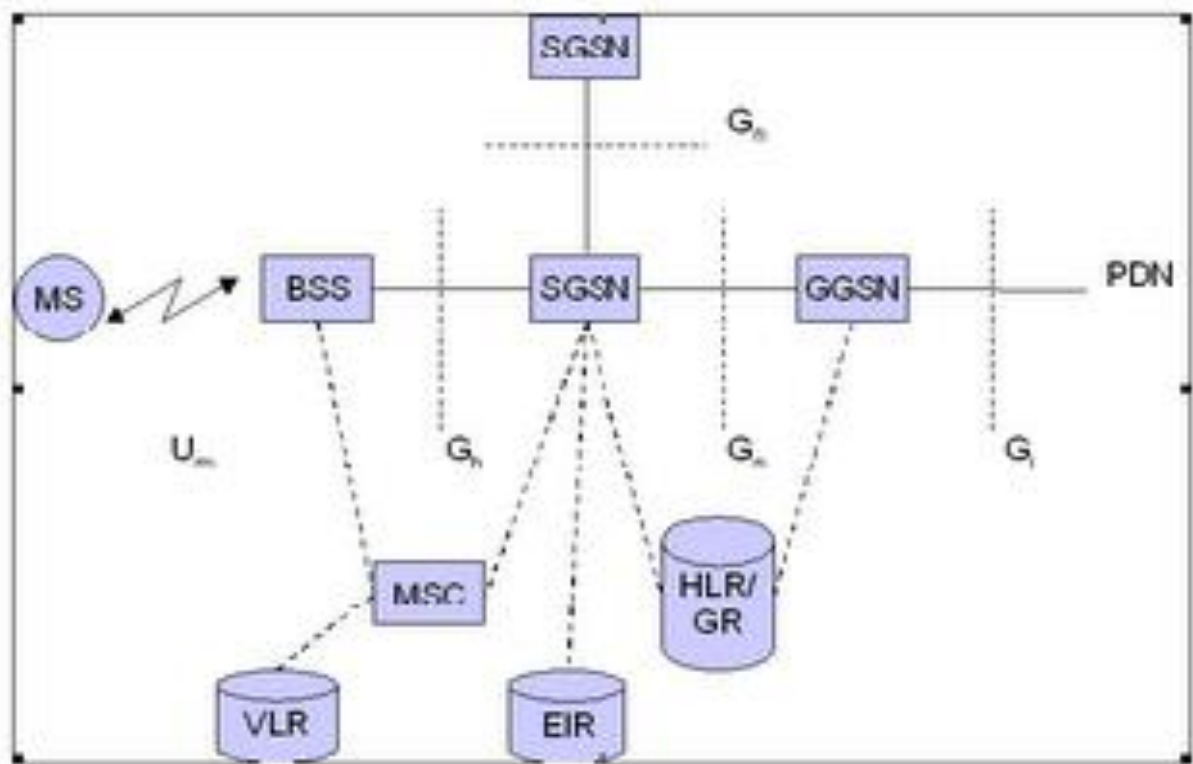
- **Communications:** E-mail, fax, unified messaging and intranet/Internet access etc.
- **Value-added services:** Information services and games etc.
- **E-commerce:** Retail, ticket purchasing, banking and financial trading etc.
- **Location-based applications:** Navigation, traffic conditions, airline/rail schedules and location finder etc.
- **Vertical applications:** Freight delivery, fleet management and sales-force automation.

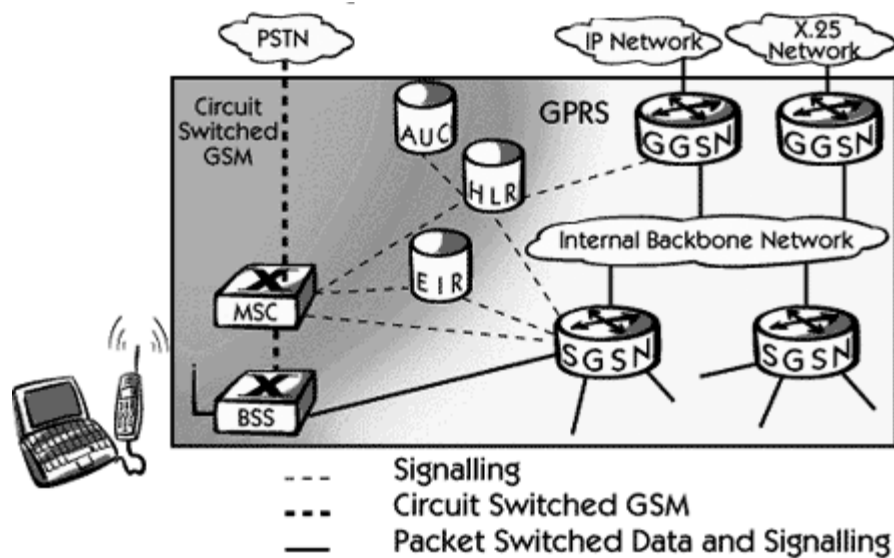
- **Advertising:** Advertising may be location sensitive. For example, a user entering a mall can receive advertisements specific to the stores in that mall.

It is also possible to send SMS messages over GPRS. In addition, it is planned to implement supplementary services, such as call forwarding unconditional (CFU), call forwarding on mobile subscriber not reachable (CFNRc), and closed user group (CUG).

### GPRS Architecture:

GPRS is a data network that overlays a second-generation GSM network. This data overlay network provides packet data transport at rates from 9.6 to 171 kbps. Additionally, multiple users can share the same air-interface resources simultaneously.





GPRS attempts to reuse the existing GSM network elements as much as possible, but to effectively build a packet-based mobile cellular network, some new network elements, interfaces, and protocols for handling packet traffic are required.

Therefore, GPRS requires modifications to numerous GSM network elements as summarized below:

GSM Network Element	Modification or Upgrade Required for GPRS.
Mobile Station (MS)	New Mobile Station is required to access GPRS services. These new terminals will be backward compatible with GSM for voice calls.
BTS	A software upgrade is required in the existing base transceiver site.
BSC	The base station controller (BSC) requires a software upgrade and the installation of new hardware called the packet control unit (PCU). The PCU directs the data traffic to the GPRS network and can be a separate hardware element associated with the BSC.
GPRS Support Nodes (GSNs)	The deployment of GPRS requires the installation of new core network elements called the serving GPRS support node (SGSN) and gateway GPRS support node (GGSN).
Databases (HLR, VLR, etc.)	All the databases involved in the network will require software upgrades to handle the new call models and functions introduced by GPRS.

### GPRS Mobile Stations:

New Mobile Station are required to use GPRS services because existing GSM phones do not handle the enhanced air interface or packet data. A variety of MS can exist, including a high-speed version of current phones to support high-speed data access, a new PDA device with an



embedded GSM phone, and PC cards for laptop computers. These mobile stations are backward compatible for making voice calls using GSM.

### **GPRS Base Station Subsystem:**

Each BSC requires the installation of one or more Packet Control Units (PCUs) and a software upgrade. The PCU provides a physical and logical data interface to the base station subsystem (BSS) for packet data traffic. The BTS can also require a software upgrade but typically does not require hardware enhancements.

When either voice or data traffic is originated at the subscriber mobile, it is transported over the air interface to the BTS, and from the BTS to the BSC in the same way as a standard GSM call. However, at the output of the BSC, the traffic is separated; voice is sent to the mobile switching center (MSC) per standard GSM, and data is sent to a new device called the SGSN via the PCU over a Frame Relay interface.

### **GPRS Support Nodes:**

Following two new components, called GPRS support nodes (GSNs), are added:

#### ***Gateway GPRS support node (GGSN):***

The Gateway GPRS Support Node acts as an interface and a router to external networks. The GGSN contains routing information for GPRS mobiles, which is used to tunnel packets through the IP based internal backbone to the correct Serving GPRS Support Node. The GGSN also collects charging information connected to the use of the external data networks and can act as a packet filter for incoming traffic.

#### ***Serving GPRS support node (SGSN):***

The Serving GPRS Support Node is responsible for authentication of GPRS mobiles, registration of mobiles in the network, mobility management, and collecting information for charging for the use of the air interface.

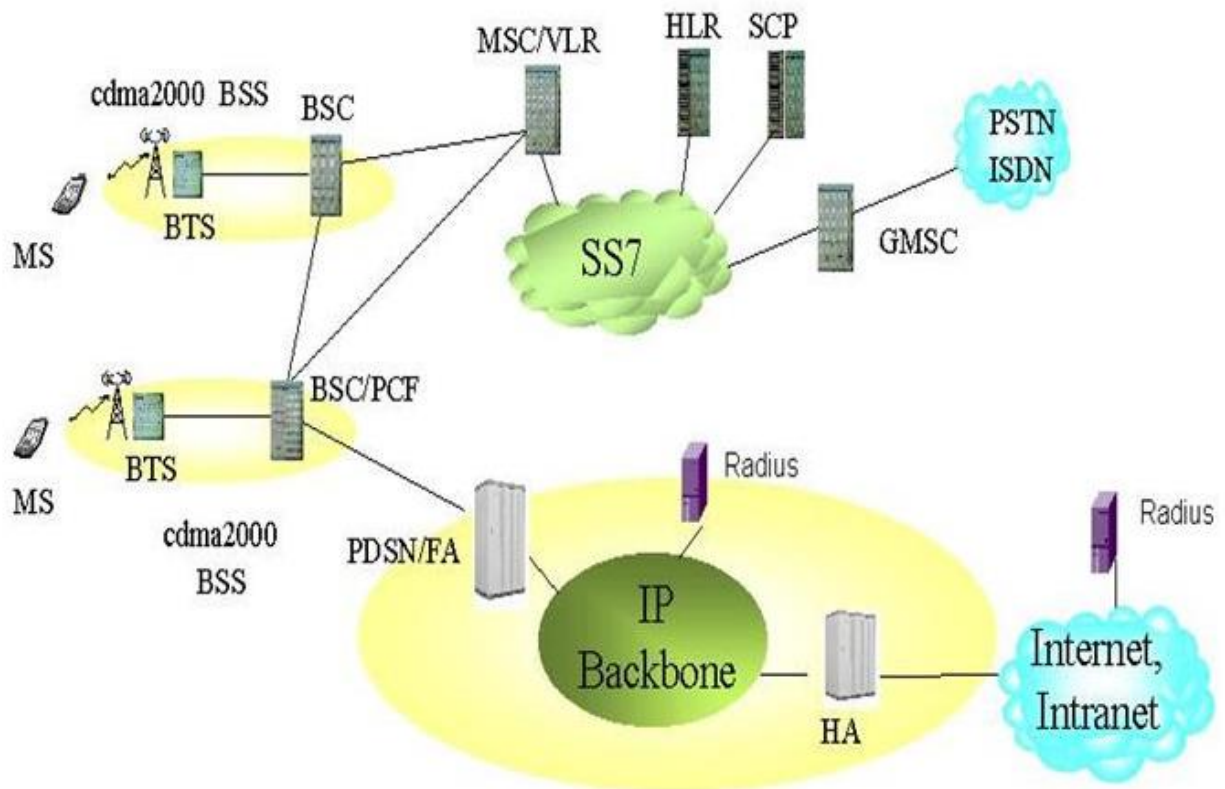
### **CDMA:**

CDMA stands for the **Code Division Multiple Access** it's basically the telecommunication technology that is used by the various radio communicating companies for accessing the channels in the minimum time. It's quite different from the other multiplexing techniques such as TDMA which divided time according to the access frequency. Unlike the TDMA, CDMA permits the users to share the bandwidth frequencies for sending and receiving the signals at the same time. It forms a specific spectrum spread for the signaling. In this way all the modulated signals after coding increase the high Data Bandwidth instead of direct signal communication.

- All users share the same frequency all the time!

- To pick out the signal of specific user, this signal is modulated with a unique code sequence

## CDMA Architecture



## Components of CDMA:

- **Mobile Station (MS):**
  - The MS is the mobile subscriber equipment, which can originate and receive calls and communicate with the BTS.
- **Base Transceiver Station (BTS):**
  - The BTS transmits and receives radio signals, realizing communication between the radio system and the mobile station.
- **Base Station Controller (BSC):**
  - Base Transceiver Station (BTS) control and management
  - call connection and disconnection
  - mobility management
  - stable and reliable radio link provision for the upper-layer services by soft/hard handoff
  - power control
  - radio resource management.

- **Packet Control Function (PCF):**
  - The PCF implements the R-P connection management. Because of the shortage of radio resources, some radio channels should be released when subscribers do not send or receive data, but the PPP connection is maintained continuously.
  - The PCF can shield radio mobility for the upper-layer services via handoff.
- **Mobile Switching Center (MSC):**
  - The MSC implements the service switching between the calling and called subscribers.
  - One MSC is connected with multiple BSCs. The MSC can also be connected to the PSTN, ISDN or other MSCs.
  - It provides the interface between the radio network and PSTN.

### **Advantages of CDMA**

Most efficient use of available bandwidth the foremost advantage is the large band width and its equivalent effect utilization for sending and receiving the data.

#### **No separate time Access divisions**

In CDMA technology there is no time limitation slot division for using the band width and sending and receiving the data like in TDMA.

#### **Power and bandwidth frequency**

Power frequency remains constant in both sending and receiving and uniform bandwidth keeps the transmission faster and in time.

#### **Resource allocation**

Another advantage of CDMA is the flexible resource allocation which means that a certain kind of pin code is provide to the active users to communicate with each other. Fixed time and bandwidth keeps the simultaneous exchange of message at both sender and receivers end and specific number of packets with limited length keeps the extra traffic out of the channel.

### **Disadvantage of CDMA**

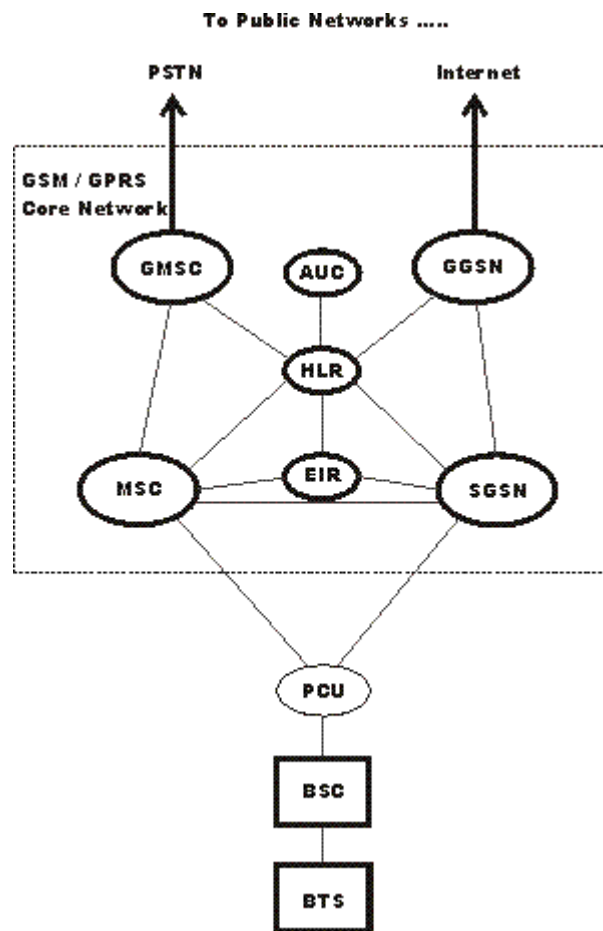
The main disadvantage of CDMA is that it can't be used for the multi channel or multiple transmission paths. All the communication is performed over a single path. This make it fixed for singular purposes which is main drawback of CDMA technology.

### **EDGE: Enhanced Data Rates for GSM Evolution**

**EDGE technology** is an extended version of **GSM**. It allows the clear and fast transmission of data and information. EDGE is also termed as IMT-SC or single carrier. **EDGE technology** is preferred over GSM due to its flexibility to carry packet switch data and circuit switch data. EDGE technology can help you connect to the internet. This technology supports the

packet switching system. EDGE develops a broadband internet connection for its users. EDGE technology helps its users to exploit the multimedia services. EDGE technology does not involve the expense of additional hardware and software technologies. It only requires the base station to install EDGE technology transceiver. EDGE technology has its edge because it can make use of both switch circuit technology and packet circuit technology. EDGE technology is also believed to support EGPRS or in other words enhanced general packet radio service.

### EDGE Architecture:



### GSM EDGE network architecture

The main new network architecture entities that are needed for the EDGE upgrade are:

- **SGSN:** GPRS Support Node - this forms a gateway to the services within the network.
- **GGSN:** Gateway GPRS Support Node which forms the gateway to the outside world.
- **PCU:** Packet Control Unit which differentiates whether data is to be routed to the packet switched or circuit switched networks.

A simplified view of the GSM EDGE network architecture can be seen in the diagram below. From this it can be seen that it is very similar to the more basic GSM network architecture, but with additional elements.

### *SGSN*

The SGSN or Serving GPRS Support Node element of the GPRS network provides a number of tasks focussed on the IP elements of the overall system. It provides a variety of services to the mobiles:

- Packet routing and transfer
- Mobility management
- Authentication
- Attach/detach
- Logical link management
- Charging data

There is a location register within the SGSN and this stores location information (e.g., current cell, current VLR). It also stores the user profiles (e.g., IMSI, packet addresses used) for all the GPRS users registered with the particular SGSN.

### **GGSN**

The GGSN, Gateway GPRS Support Node is one of the most important entities within the GSM EDGE network architecture.

The GGSN organises the inter-working between the GPRS / EDGE network and external packet switched networks to which the mobiles may be connected. These may include both Internet and X.25 networks.

The GGSN can be considered to be a combination of a gateway, router and firewall as it hides the internal network to the outside. In operation, when the GGSN receives data addressed to a specific user, it checks if the user is active, then forwarding the data. In the opposite direction, packet data from the mobile is routed to the right destination network by the GGSN.

### **PCU**

The PCU or Packet Control Unit is a hardware router that is added to the BSC. It differentiates data destined for the standard GSM network (circuit switched data) and data destined for the EDGE network (Packet Switched Data). The PCU itself may be a separate physical entity, or more often these days it is incorporated into the base station controller, BSC, thereby saving additional hardware costs.

### **3G Concepts:**

**3G mobile technology** is an edge over telephone and internet. Different 2G technologies are now gather and utilized at one place to give the mobile technology a new dimension and these called **3G mobile technologies**. Combination of all the quality features of different 2G versions brings excellent one best device with high data transfer rate, maximum width of the frequency band, WiFi facility, and first-rate roaming. Let see the technologies used in 3g mobile and their properties one by one.

#### ***CDMA 2000 (3G Mobile Technology)***

Third Generation Partnership Project 2 (TGPP 2) introduced this CDMA 2000. Five countries Telecommunication standards organization did joint venture to design this technology. This technology advancement was already entertained by many other systems and serving as **3G Mobile Technology**.

Properties:

- Able to use verbal and textual service concurrently.
- High data transfer rate approximately 200 kbps.
- CDMA 2000 provides wireless facility.

#### ***W-CDMA (UMTS) - (3G Mobile Technology)***

W-CDMA stands for Wideband Code Division Multiple Access is a third generation's important feature based on radio transmission system. It is designed by ETSI Alpha organization. It is quite challenging to apply it because of its complex features and versatile properties. Properties:

- Make hybrid with IS-95 (digital cellular standard) component of 2G technology responsible for the high frequency.
- It is able to download 14.7 Mb/s.
- Provide wideband known as Spread Spectrum in addition to code division multiple accesses.
- Improved audio-visual effects.

### **WiFi:**

**Wireless Fidelity** – is refers to IEEE 802.11 standard for Wireless Local Area Networks (WLANs). Wi-Fi network connect computers to each other , to the internet and to the wired network. It works on physical and datalink layer.

Wi-Fi networks use radio technologies to transmit and receive data at high speed:

- IEEE 802.11b
- IEEE 802.11a
- IEEE 802.11g

**IEEE 802.11b:**

- Operates at 2.4GHz radio spectrum
- 11 Mbps (theoretical speed) – within 30m range
- 4-6 Mbps (actual speed)
- 100 -150 feet range
- Most popular, Least Expensive
- Interference from mobile phones and Bluetooth devices which can reduce the transmission speed.

**IEEE 802.11a:**

- Operates at 5 GHz
- 54 Mbps (theoretical speed)
- 15 – 20 Mbps (actual speed)
- 50 – 75 feet range
- More expensive and not compatible with 802.11b.

**IEEE 802.11g:**

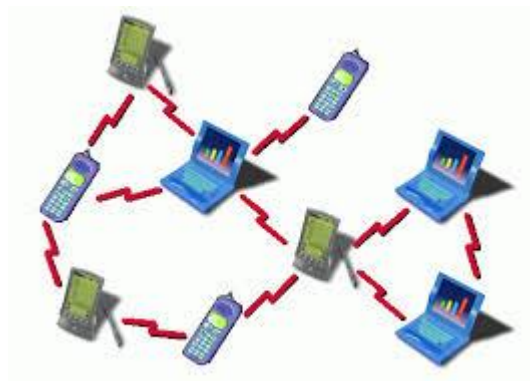
- Combine the features of both standards (a,b).
- 100 – 150 feet range
- 2.4 GHz radio frequencies and compatible with 'b'.

**Elements of Wi-Fi:**

- **Access Point (AP)** – The AP is a wireless LAN Transceiver or base station that can connect one or many wireless devices simultaneously to the Internet.
- **Wi-Fi cards** – They accept the wireless signal and relay information. They can be an internal and external.
- **Safeguards** - Firewalls and anti-virus software protect networks from uninvited users and keep information secure.

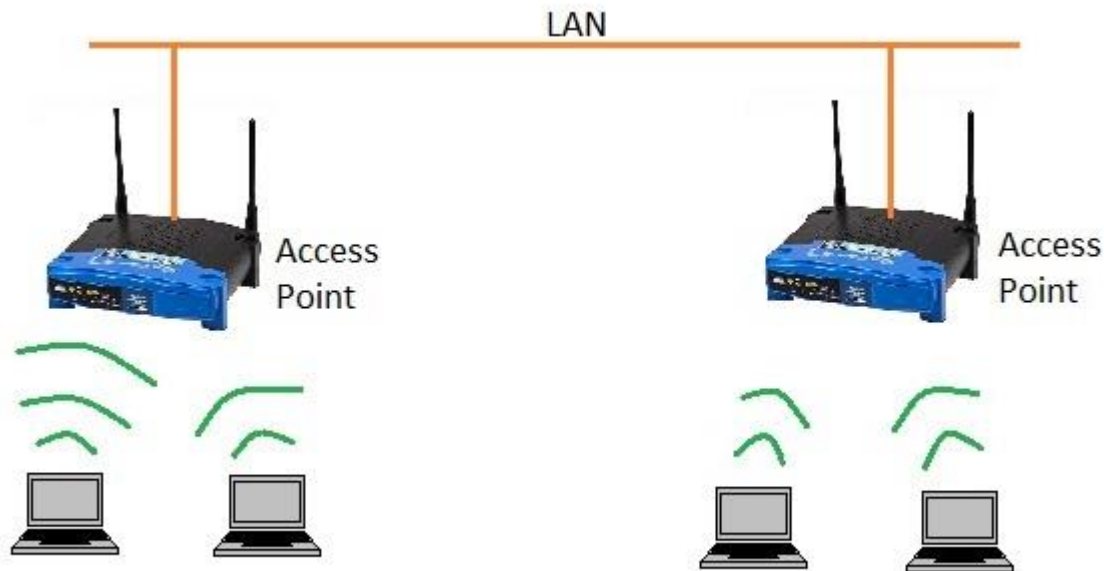
**Wi-Fi Network Topologies:**

- **Peer -to- topology ( Ad-hoc Mode):**



- AP is not required.
- clients can connect to each other directly.
- It is useful for setting up a wireless network quickly and easily.

- **AP – based topology (Infrastructure Mode):**



- The client communicates through Access Point.
- Any communication has to go through AP.
- If a mobile station wants to communicate with another mobile station, it needs to send the information to AP first, then AP sends it to the destination Mobile station.

### **Hotspots:**

A Hotspot is a geographical area that has a readily accessible wireless network. Hotspots are equipped with Broadband Internet connection and one or more Access Points that allow users to access the internet wirelessly. It can be setup in any public location that can support an Internet connection.

### **Working of Wi-Fi Network:**

- A Wi-Fi hotspot is created by installing an access point to an internet connection.
- An Access Point act as a base station.
- When Wi-Fi enabled device encounters a hotspot the device can then connect to that network wirelessly.
- A single Access point can support up to 30 users and can function within a range of 100 – 150 feet indoors and up to 300 feet outdoors.
- Many access points can be connected to each other via Ethernet cables to create a single large network.

### **Advantages:**

- Mobility
- Ease of Installation



- Flexibility
- Cost
- Reliability
- Security
- Roaming
- Speed

#### **Limitations:**

- Interference
- Degradation in performance
- High power consumption
- Limited range

#### **WiMAX: (Worldwide Interoperability for Microwave Access)**

It is a telecommunications protocol that provides fixed and mobile Internet access. The current WiMAX revision provides up to 40 Mbit/s.

WiMAX is a wireless Internet service designed to cover wide geographical areas serving large number of users at low cost.

WiMAX is IEEE 802.16 standard defining wide area (Wireless MAN) wireless data networking.

A WiMAX system consists of two parts:

- **A Transmitter**

A single WiMAX tower can provide coverage to a very large area – as big as 3,000 sq. miles.

- **A Receiver**

The receiver and antenna could be a small box or PCMCIA card or they could be built into a laptop as the way Wi-Fi access is today.

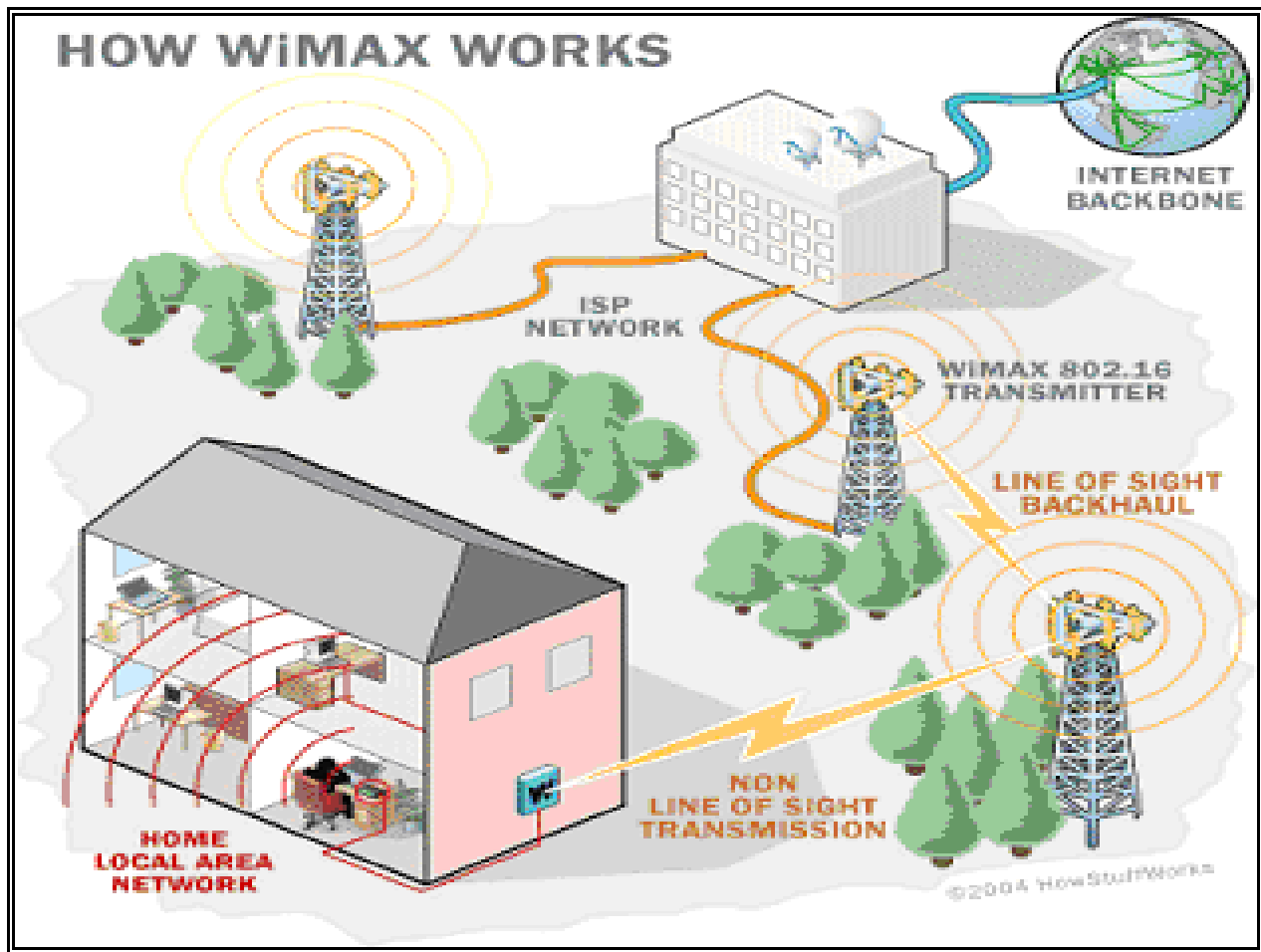
#### **Forms of Wireless Service:**

- **NLOS (Non Line of Sight)**

Wi-Fi sort of service, where a small antenna on your computer connects to the tower. In this mode, WiMAX uses a **lower frequency range** -- 2 GHz to 11 GHz (similar to WiFi). Lower-wavelength transmissions are not as easily disrupted by physical obstructions -- they are better able to diffract, or bend, around obstacles.

- **LOS ( Line of Sight)**

A fixed dish antenna points straight at the WiMAX tower from a rooftop or pole. The line-of-sight connection is stronger and more stable, so it's able to send a lot of data with fewer errors. Line-of-sight transmissions use **higher frequencies**, with ranges reaching a possible 66 GHz. At higher frequencies, there is less interference and lots more bandwidth.



## Advantages:

- A Single WiMAX main Station can serve hundreds of users.
- Endpoints install within days instead of the weeks required for wired connections.
- Data rates as high as 280 Mbps and distances of 30 miles are possible.
- Users can operate mobile within 3-5 miles of a base station at data rates up to 75 Mbps.
- No FCC radio licensing is required.
- Less expensive than DSL or coaxial cable.

## Disadvantages:

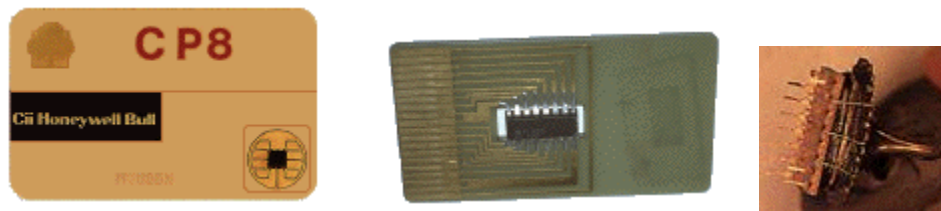
- Line-of-Sight is required for long distance (5-30 mile) connections.
- Heavy rains can disrupt the service.
- Other wireless electronics in the vicinity can interfere with the WiMAX connection and cause a reduction in data throughput or even a total disconnect.

## Comparison with Wi-Fi

- WiMAX is a long range system, covering many kilometres, that uses licensed or unlicensed spectrum to deliver connection to a network, in most cases the Internet. Wi-Fi uses unlicensed spectrum to provide access to a local network. Wi-Fi is more popular in end user devices.
- Wi-Fi runs on the Media Access Control's CSMA/CA protocol, which is connectionless and contention based, whereas WiMAX runs a connection-oriented MAC.
- WiMAX and Wi-Fi have quite different quality of service (QoS) mechanisms:  
WiMAX uses a QoS mechanism based on connections between the base station and the user device. Each connection is based on specific scheduling algorithms

## Hardware Architecture:

### Smart Cards



Smart Card is a plastic card with an embedded microprocessor and a memory large enough to store programs made by the card-issuing company. The exact structure of a smart card is specified by international standards: the plastic card must have dimensions of 85.60mm x 53.98mm x 0.80mm and must be able to bend a specified amount without damage for instance. A printed circuit and an integrated circuit chip (microcontroller) are embedded on the card. As silicon cannot handle bending very well, the chip must be very small. The printed circuit is a thin gold plate that provides electrical contacts to the outside world and also protects the chip from mechanical stress and electrical static.

Smart Cards do not have an internal power source but need power to operate. Therefore they only operate when in the presence of a Card Accepting Device (CAD) which supplies their power requirements. Most Smart Cards come into physical contact with CADs while others do not.

### Life-cycle of a Smart Card

The manufacture of a Smart Card consists of several stages.

- Stage 1. The processor is made and tested by a chip-manufacturing company. A fabrication key (FK) is added to protect the chip from fraudulent modification until the next stage. The FK of each chip is unique and is derived from a master manufacturer key.

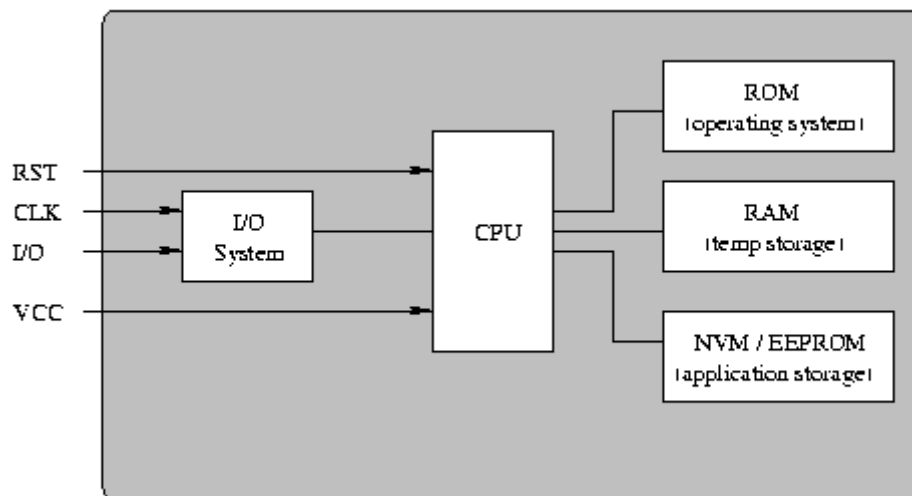
- Stage 2. The chip is mounted on a plastic card and the FK replaced by a personalization key. This is protected from later change by a personalization lock and physical memory access instructions are disabled.
- Stage 3. The card issuer (e.g. a bank) writes data files and application data to the card. Information regarding the identity of the card holder (e.g. a client of the bank) is stored as well.
- Stage 4. The card is used by the card holder. Access to its information is limited by the application's security policies.
- Stage 5. Card is invalidated either by blocking all PINS or by writing an invalidation lock. In the second case, only read instructions may remain active for analysis purposes. In the first case, neither writes nor reads are allowed.

## Elements of Smart Card Architecture

Smart Cards are thin cards with an embedded chip, and this automatically poses its own unique challenges of architectural design. However, it turns out that the solutions tend to be a scaling down of conventional chips rather than inventing an all-new chip.

### Central Processing Unit

Traditionally this is an 8-bit microcontroller but increasingly more powerful 16 and 32-bit chips are being used. However, none have multi-threading and other powerful features that are common in standard computers. Smart Card CPUs execute machine instructions at a speed of approximately 1 MIPS. A coprocessor is often included to improve the speed of encryption computations.



### Memory System

There are three main types of memory on cards:

- RAM. 1K. This is needed for fast computation and response. Only a tiny amount is available.

- EEPROM (Electrically Erasable PROM). Between 1 to 24K. Unlike RAM, its contents are not lost when power is. Applications can run off and write to it, but it is very slow and one can only read/write to it so many (100 000) times.
- ROM. Between 8 to 24K. The Operating System and other basic software like encryption algorithms are stored here.

## **Input/Output**

This is via a single I/O port that is controlled by the processor to ensure that communications are standardized, in the form of APDUs (A Protocol Data Unit).

## **Interface Devices (IFDs)**

Smart Cards need power and a clock signal to run programs, but carry neither. Instead, these are supplied by the Interface Device - usually a Smart Card Reader - in contact with the card. This obviously means that a Smart Card is nothing more than a storage device while being warmed in your pocket.

In addition to providing the power and clock signals, the reader is responsible for opening a communication channel between application software on the computer and the operating system on the card. Nearly all Smart Card readers are actually reader/writers, that is, they allow an application to write to the card as well as read from it.

The communication channel to a Smart Card is half-duplex. This means that data can either flow from the IFD to the card or from the card to the IFD but data cannot flow in both directions at the same time. The receiver is required to sample the signal on the serial line at the same rate as the transmitter sends it in order for the correct data to be received. This rate is known as the bit rate or baud rate. Data received by and transmitted from a Smart Card is stored in a buffer in the Smart Card's RAM. As there isn't very much RAM, relatively small packets (10 - 100 bytes) of data are moved in each message.

## **Operating Systems**

The operating system found on the majority of Smart Cards implements a standard set of commands (usually 20 - 30) to which the Smart Card responds. Smart Card standards such as ISO 7816 and CEN 726 describe a range of commands that Smart Cards can implement. Most Smart Card manufacturers offer cards with operating systems that implement some or all of these standard commands (and possibly extensions and additions). The relationship between the Smart Card reader and the Smart Card is a master/slave relationship. The reader sends a command to the Smart Card, the card executes the command and returns the result (if any) to the reader and waits for another command.

Microsoft released a miniaturized version of Windows for Smart Cards in late 1998, and early versions of a Gnu O/S have been released.

## **File Systems**

Most operating systems also support a simple file system based on the ISO 7816 standard. A Smart Card file is actually just a contiguous block. Files are organized in a hierarchical tree format. Once a file is allocated, it cannot be extended and so files must be

created to be the maximum size that they are expected to be. Each file has a list of which parties are authorized to perform which operations on it. There are different types of files: linear, cyclic, transparent, SIM, etc. The usual create, delete, read, write and update file operations can be performed on all of them. Certain other operations are supported only on particular types of files.

Type	Special Operations	Example
Linear	seek	credit card account table
Cyclic	read next, read previous	transaction log
Transparent	read and write binary	picture
SIM file	encrypt, decrypt	cellular telephone

## Software

Smart Cards are either Soft-Mask or Hard-Mask, depending on whether most of the application is in EEPROM or ROM. Hard-Mask cards are more expensive. Some application-specific data/instructions always needs to be stored on EEPROM. Cards do not as a rule run anything off RAM.

When programming a Smart Card, it is standard practice to get the program running on a simulator first for debugging, since EEPROM can only be written to a finite number of times in its lifetime.

Test-running also happens on a different level: banks commonly use a soft mask card for pilot testing new applications and then to move on to more customer-resistant hard mask cards for larger deployments. However, some applications have limited deployments that are never taken to hard mask, as hard masking is expensive in both time and money. Hard masks also may not be justified for some applications, such as an employee identification card for small companies.

## Programming Languages

Most SmartCards are currently programmed in low-level languages based on proprietary SmartCard operating systems. Some of the programming has been done in the chip's native instruction set (generally Motorola 6805, Intel 8051, or Hitachi H8).

In 1998- 2000, a new type of card has shown up, sometimes called a re-configurable card. These have a more robust operating system that permits the addition or deletion of application code after the card is issued. Such cards are generally programmed in Java and are therefore called Java Cards. Other relatively popular languages relate to Windows for SmartCards or MEL (the Multos programming language) or even Basic.

## All in all smart Cards have a number of advantages over magnetic stripe cards:

- Greater reliability.
- Reduction in tampering and counterfeiting – due to high security mechanisms. Storage capacity is increased by up to 100 times.

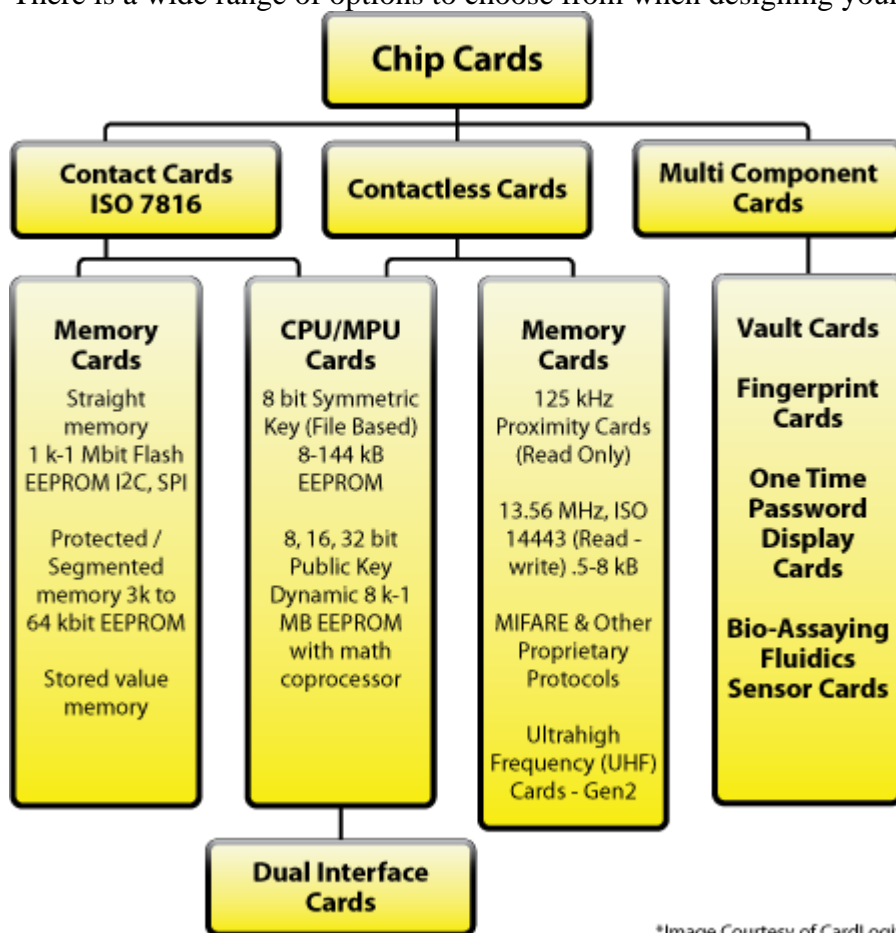
- Smartcards are multi-functional.
- The anticipated working life of a smartcard is ten years compared to that of a magnetic stripe card at three.

## Types of Smart Card

Smart cards are defined according to

1). How the card data is read and written 2). The type of chip implanted within the card and its capabilities.

There is a wide range of options to choose from when designing your



system.

\*Image Courtesy of CardLogix

## Applications of Smart Cards

The most immediate areas in which Smart Cards have been establishing their positions include:

- *Financial services* - Financial institutions are looking to use Smart Cards to deliver higher value-added services to businesses and consumers at a lower cost per transaction.

These services include money on a card, corporate card programs, and targeted marketing programs based on analysis of consumers' buying patterns.

- *Affinity programs* - Airlines, retailers, and other companies that offer a range of ancillary services and loyalty programs along with their basic product want to use Smart Cards to deliver these programs with a higher level of service, improved ease of use, and at a lower cost. For example, airlines want to use Smart Cards not only as a vehicle for issuing and carrying tickets - even though the single benefit of being able to securely order/provide a ticket directly to chip cards via the Internet is substantial. Airlines also want to use the cards to provide tie-ins to their frequent-flyer programs and to cross-marketing deals with auto rentals and hotels, as well as to provide simplified access to private airline lounges.
- *Cellular phones* - Cellular phone services in the United States are losing \$1.5 million per day because of fraud. Although Smart Cards offer a mechanism to secure cellular phones against fraudulent use, only Java Cards offer the ability to download new functions into a phone in real time.
- *Set-top boxes* - Subscription satellite and cable services suffer from fraud problems similar to those in the cellular phone business. Once again, Java Cards offer security and the ability to add/update customer functions available to consumers in real time.
- *Secure network access* - Smart Cards can carry an individual's digital signature. With this ability, they provide a special mechanism to secure access to computer networks within a corporation, they help ensure that only individuals with the proper authority can get access to specific network resources, and they reduce the likelihood that hackers can break into a system.

## **SIM:**

A **subscriber identity module** or **subscriber identification module (SIM)** is an integrated circuit that securely stores the service-subscriber key (IMSI) used to identify a subscriber on mobile telephony devices (such as mobile phones and computers).

A SIM is held on a removable **SIM card**, which can be transferred between different mobile devices. SIM cards were first made the same size as a credit card (85.60 mm × 53.98 mm × 0.76 mm). The development of physically smaller mobile devices prompted the development of a smaller SIM card, the mini-SIM card. Mini-SIM cards have the same thickness as full-size cards, but their length and width are reduced to 25 mm × 15 mm.

A SIM card contains its unique serial number (ICCID), internationally unique number of the mobile user (IMSI), security authentication and ciphering information, temporary information related to the local network, a list of the services the user has access to and two passwords (PIN for usual use and PUK for unlocking).



## Design

### SIM chip structure and packaging

There are three operating voltages for SIM cards: 5 V, 3 V and 1.8 V. The operating voltage of the majority of SIM cards launched before 1998 was 5 V. SIM cards produced subsequently are compatible with 3 V and 5 V or with 1.8 V and 3 V.

Dual SIM phones are now made by some mobile phone manufacturers, which save the user from carrying around a separate phone for every number. There are two types: the first allows one to switch between the SIMs, whilst the second allows both SIMs to be active simultaneously.

SIM operating systems come in two main types: native and Java Card. Native SIMs are based on proprietary, vendor-specific software, whereas the Java Card SIMs are based on standards, particularly Java Card, which is a subset of the Java programming language specifically targeted at embedded devices. Java Card allows the SIM to contain programs that are hardware independent and interoperable.

## Data

SIM cards store network-specific information used to authenticate and identify subscribers on the network. The most important of these are the ICCID, IMSI, Authentication Key (Ki), Local Area Identity (LAI) and Operator-Specific Emergency Number. The SIM also stores other carrier-specific data such as the SMSC (Short Message Service Center) number, Service Provider Name (SPN), Service Dialing Numbers (SDN), Advice-Of-Charge parameters and Value Added Service (VAS) applications.

### ICCID

Each SIM is internationally identified by its Integrated circuit card identifier (ICCID). ICCIDs are stored in the SIM cards and are also engraved or printed on the SIM card body during a process called personalization. The ICCID is defined by the ITU-T recommendation E.118 as the *Primary Account Number*. The GSM Phase 1 defined the ICCID length as 10 octets with operator-specific structure.

The number is composed of the following subparts:

Issuer identification number (IIN)

Maximum of seven digits:

- Major industry identifier (MII), 2 fixed digits, 89 for telecommunication purposes.
- Country code, 1-3 digits.
- Issuer identifier, 1-4 digits.

#### Individual account identification

- Individual account identification number. Its length is variable but every number under one IIN will have the same length.

#### Check digit

- Single digit calculated from the other digits using the Luhn algorithm.

### **International mobile subscriber identity (IMSI)**

SIM cards are identified on their individual operator networks by a unique IMSI. Mobile operators connect mobile phone calls and communicate with their market SIM cards using their IMSIs. The format is:

- The first 3 digits represent the Mobile Country Code (MCC).
- The next 2 or 3 digits represent the Mobile Network Code (MNC). 3-digit MNC codes are allowed by E.212 but are mainly used in the United States and Canada.
- The next digits represent the mobile station identification number. Normally there will be 10 digits but would be fewer in the case of a 3-digit MNC or if national regulations indicate that the total length of the IMSI should be less than 15 digits.

### **Authentication key ( $K_i$ )**

The  $K_i$  is a 128-bit value used in authenticating the SIMs on the mobile network. Each SIM holds a unique  $K_i$  assigned to it by the operator during the personalization process. The  $K_i$  is also stored in a database (known as Authentication Center or AuC) on the carrier's network.

#### Authentication process:

1. When the Mobile Equipment starts up, it obtains the International Mobile Subscriber Identity (IMSI) from the SIM card, and passes this to the mobile operator requesting access and authentication. The Mobile Equipment may have to pass a PIN to the SIM card before the SIM card will reveal this information.
2. The operator network searches its database for the incoming IMSI and its associated  $K_i$ .
3. The operator network then generates a Random Number (RAND, which is a nonce) and signs it with the  $K_i$  associated with the IMSI (and stored on the SIM card), computing another number known as Signed Response 1 (SRES\_1).
4. The operator network then sends the RAND to the Mobile Equipment, which passes it to the SIM card. The SIM card signs it with its  $K_i$ , producing SRES\_2, which it gives to the Mobile Equipment along with encryption key  $K_c$ . The Mobile Equipment passes SRES\_2 on to the operator network.
5. The operator network then compares its computed SRES\_1 with the computed SRES\_2 that the Mobile Equipment returned. If the two numbers match, the SIM is authenticated and the Mobile Equipment is granted access to the operator's network.  $K_c$  is used to encrypt all further communications between the Mobile Equipment and the network.

## **Location area identity**

The SIM stores network state information, which is received from the Location Area Identity (LAI). Operator networks are divided into Location Areas, each having a unique LAI number. When the device changes locations, it stores the new LAI to the SIM and sends it back to the operator network with its new location. If the device is power cycled, it will take data off the SIM, and search for the previous LAI. This saves time by avoiding having to search the whole list of frequencies that the telephone normally would.

## **SMS messages and contacts**

Most SIM cards will orthogonally store a number of SMS messages and phone book contacts. The contacts are stored in simple 'Name and number' pairs: entries containing multiple phone numbers and additional phone numbers will usually not be stored on the SIM card. When a user tries to copy such entries to a SIM the handset's software will break them up into multiple entries, discarding any information that isn't a phone number. The number of contacts and messages stored depends on the SIM; early models would store as few as 5 messages and 20 contacts while modern SIM cards can usually store over 250 contacts.

## ***Formats***

Micro-SIM and mini-SIM, as normally supplied in full-sized carrier cards

SIM cards are available in three standard sizes. The first is the size of a credit card (85.60 mm  $\times$  53.98 mm  $\times$  0.76 mm). The newer, most popular version has the same thickness, but has a length of 25 mm and a width of 15 mm, and has one of its corners truncated (chamfered) to prevent misinsertion. The newest incarnation, known as the micro-SIM or 3FF, has dimensions of 15 mm  $\times$  12 mm. Most cards of the two smaller sizes are supplied as a credit card size with the smaller standard card held in place by a few plastic links; it can easily be broken off to be used in a device that uses the smaller SIM.

The mini-SIM card has the same contact arrangement as the full-size SIM card and is normally supplied within a full-size card carrier, attached by a number of linking pieces. This arrangement allows for such a card to be used in a device requiring a full-size card, or to be used in a device requiring a mini-SIM card after cleanly breaking the scorings manufactured in the outline of a mini-SIM card.

Even smaller device sizes have prompted the development of a yet smaller card size, the *3FF card* or *micro-SIM*. Micro-SIM cards have the same thickness and contact arrangement again, but the length and width are further reduced to 15 mm  $\times$  12 mm. The specifications for the 3FF card or micro-SIM also include additional functionality beyond changing the physical card size.



**SCHOOL OF COMPUTING**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**Mobile Application Development SBS1603**

## UNIT II

Android Programming Introduction: History of Android, Introduction to Android Operating Systems, Android Development Tools, Android Architecture. Overview of object oriented programming using Java: OOPs Concepts: Inheritance, Polymorphism, Interfaces, Abstract class, Threads, Overloading and Overriding, Java Virtual Machine.

### **Android Programming Introduction:**

- ✓ Android is an operating system and programming platform developed by Google for mobile phones and other mobile devices, such as tablets.
- ✓ It can run on many different devices from many different manufacturers. Android includes a software development kit (SDK) that helps you write original code and assemble software modules to create apps for Android users.
- ✓ Android also provides a marketplace to distribute apps. All together, Android represents an ecosystem for mobile apps.
- ✓ Android is an open source and Linux-based Operating System for mobile devices such as smartphones and tablet computers. Android was developed by the Open Handset Alliance, led by Google, and other companies.
- ✓ Android offers a unified approach to application development for mobile devices which means developers need only develop for Android, and their applications should be able to run on different devices powered by Android.
- ✓ The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.
- ✓ On June 27, 2012, at the Google I/O conference, Google announced the next Android version, 4.1 Jelly Bean. Jelly Bean is an incremental update, with the primary aim of improving the user interface, both in terms of functionality and performance.
- ✓ The source code for Android is available under free and open source software licenses. Google publishes most of the code under the Apache License version 2.0 and the rest, Linux kernel changes, under the GNU General Public License version 2.

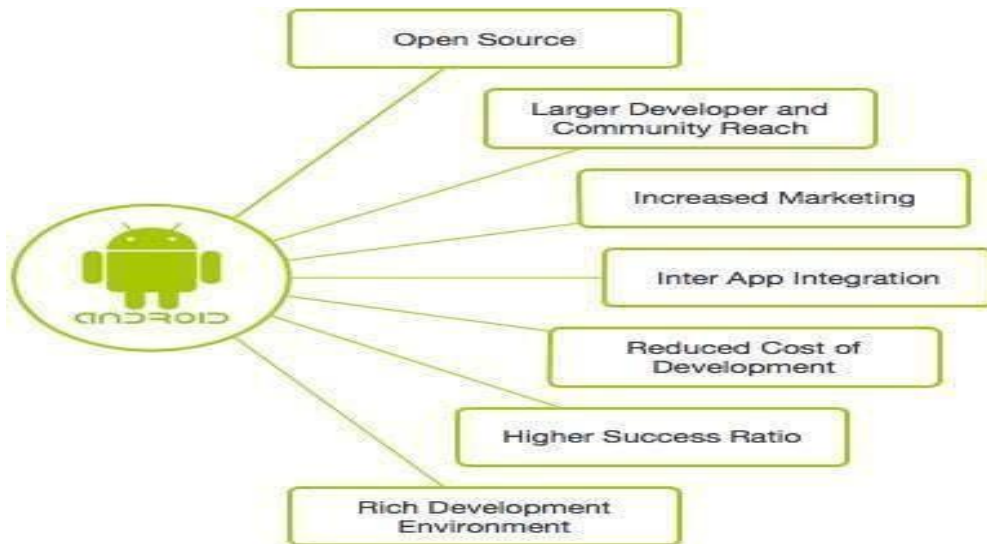


Figure 2.1 Features of Android

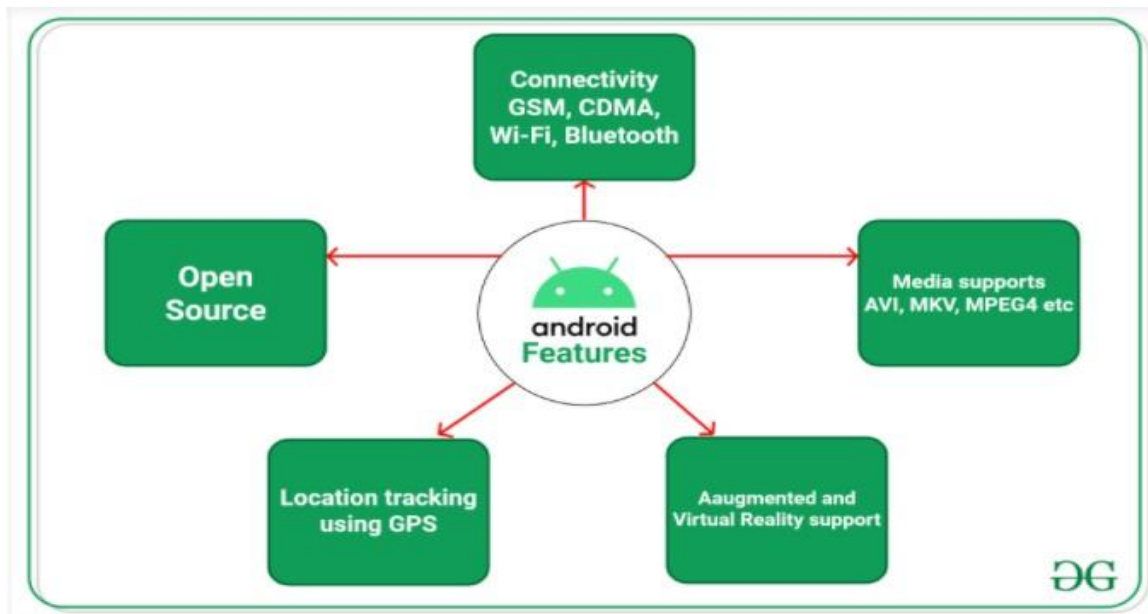


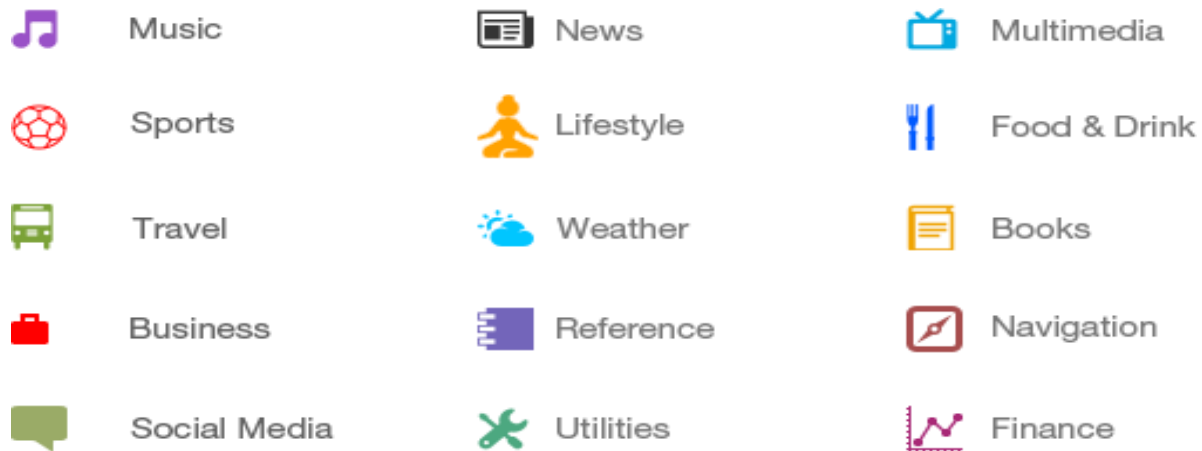
Figure 2.2 Features of Android

Sr.No.	Feature & Description
1	<p>Beautiful UI</p> <p>Android OS basic screen provides a beautiful and intuitive user interface.</p>
2	<p>Connectivity</p> <p>GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.</p>
3	<p>Storage</p> <p>SQLite, a lightweight relational database, is used for data storage purposes.</p>
4	<p>Media support</p> <p>H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP.</p>
5	<p>Messaging</p> <p>SMS and MMS</p>
6	<p>Web browser</p> <p>Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3.</p>
7	<p>Multi-touch</p> <p>Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero.</p>
8	<p>Multi-tasking</p> <p>User can jump from one task to another and same time various application can run simultaneously.</p>
9	<p>Resizable widgets</p> <p>Widgets are resizable, so users can expand them to show more content or shrink them to save space.</p>
10	<p>Multi-Language</p> <p>Supports single direction and bi-directional text.</p>

**Table 2.1 Features of Android**

## Categories of Android applications:

There are many android applications in the market. The top categories are –



## History of Android

- 1) Initially, Andy Rubin founded Android Incorporation in Palo Alto, California, United States in October, 2003.
- 2) In 17th August 2005, Google acquired android Incorporation. Since then, it is in the subsidiary of Google Incorporation.
- 3) The key employees of Android Incorporation are Andy Rubin, Rich Miner, Chris White and Nick Sears.
- 4) Originally intended for camera but shifted to smart phones later because of low market for camera only.
- 5) Android is the nick name of Andy Rubin given by coworkers because of his love to robots.
- 6) In 2007, Google announces the development of android OS.
- 7) In 2008, HTC launched the first android mobile.

## Programming Languages used in Developing Android Applications

- ❖ Java
- ❖ Kotlin



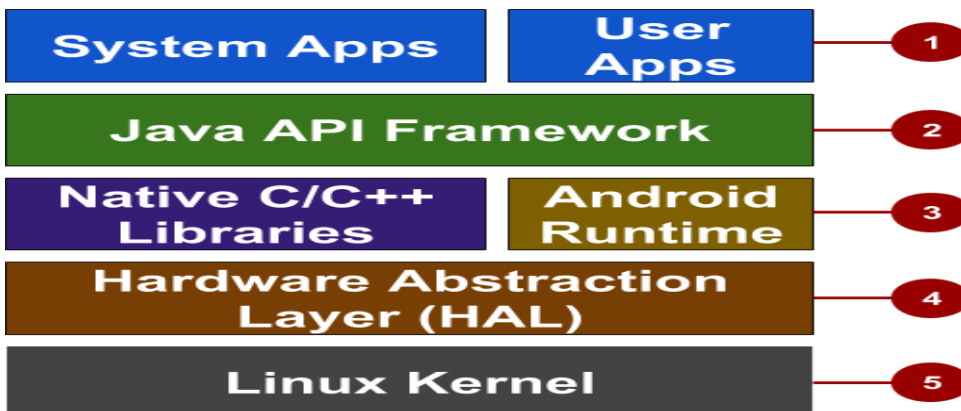
Developing the Android Application using Kotlin is preferred by Google, as Kotlin is made an official language for Android Development, which is developed and maintained by JetBrains. Previously before the Java is considered the official language for Android Development. Kotlin is made official for Android Development in Google I/O 2017.

### The challenges of Android app development:

While the Android platform provides rich functionality for app development, there are still a number of challenges you need to address, such as:

- ✓ Building for a multiscreen world
- ✓ Getting performance right
- ✓ Keeping your code and your users more secure
- ✓ Making sure your app is compatible with older platform versions
- ✓ Understanding the market and the user

### Major components of the Android stack



**Figure 2.3 Major Components of Android**

- ✓ **Apps:** Your apps live at this level, along with core system apps for email, SMS messaging, calendars, internet browsing, and contacts.
- ✓ **Java API framework:** All features for Android development, such as UI components, resource management, and lifecycle management, are available through application programming interfaces (APIs). You don't need to know the details of how the APIs work. You only need to learn how to use them.
- ✓ **Libraries and Android runtime:** Each app runs in its own process, with its own instance of the Android runtime. Android includes a set of core runtime libraries that

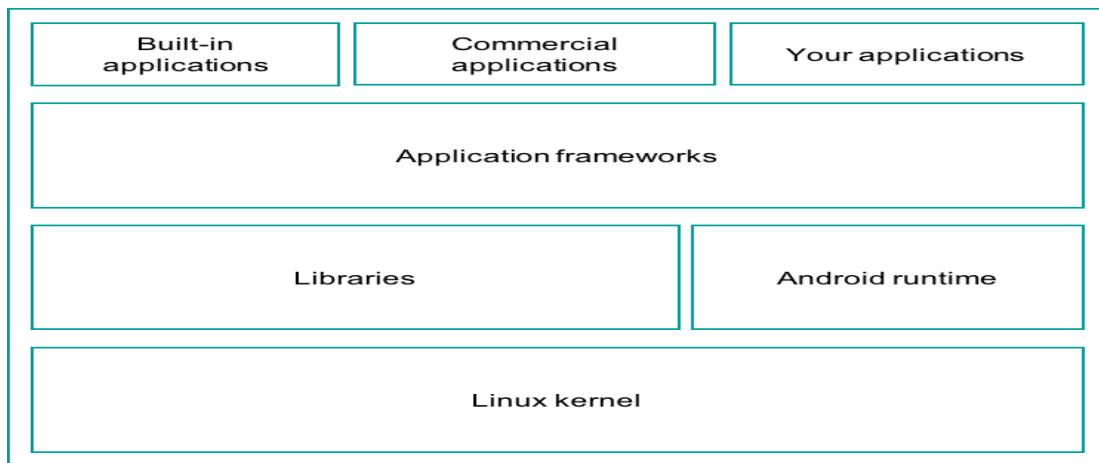
provide most of the functionality of the Java programming language. Many core Android system components and services are built from native code that require native libraries written in C and C++. These native libraries are available to apps through the Java API framework.

✓ **Hardware abstraction layer (HAL):**

This layer provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework. The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera or Bluetooth module.

✓ **Linux kernel:**

The foundation of the Android platform is the Linux kernel. The layers above the Linux kernel rely on the Linux kernel for threading, low-level memory management, and other underlying functionality. Using a Linux kernel enables Android to take advantage of Linux-based security features and allows device manufacturers to develop hardware drivers for a well-known kernel



**Figure 2.4. Android software layers**

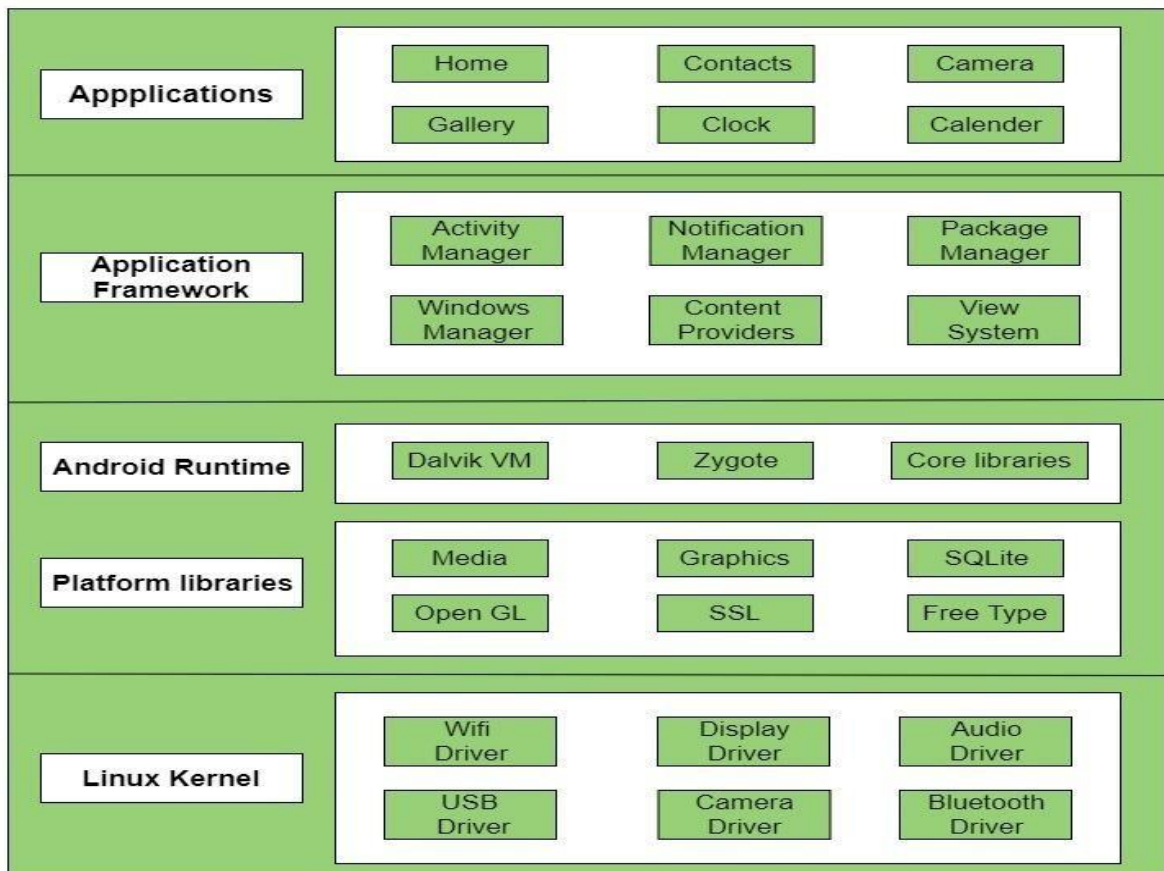
- ✓ **Applications:** Built-in applications, such as phone, contacts, browser, and more. The specific applications vary by Android version and manufacturer. Commercial applications from market places, such as Google Play, Amazon, and more. Side-loaded applications, including the ones you will build. You install these via USB cable.
- ✓ **Application frameworks,** such as telephony manager, location manager, notification manager, content providers, windowing, resource manager, and more.
- ✓ **Libraries,** such as graphics libraries, media libraries, database libraries, sensors, and so on.

- ✓ **The Android runtime** is responsible for executing and managing applications as they run.
- ✓ **Linux Kernel**, including power, file system, drivers, process management, and more.

## Android Architecture

**Android architecture** or **Android software stack** is categorized into five parts:

1. Linux kernel
2. native libraries (middleware),
3. Android Runtime
4. Application Framework
5. Applications



**Figure 2.5. Android Architecture**

✓ **Applications –**

Applications is the top layer of android architecture. The pre-installed applications like home, contacts, camera, gallery etc and third party applications downloaded from the play store like chat applications, games etc. will be installed on this layer only. It runs within the Android run time with the help of the classes and services provided by the application framework.

✓ **Application framework –**

Application Framework provides several important classes which are used to create an Android application. It provides a generic abstraction for hardware access and also helps in managing the user interface with application resources. Generally, it provides the services with the help of which we can create a particular class and make that class helpful for the Applications creation.

It includes different types of services activity manager, notification manager, view system, package manager etc. which are helpful for the development of our application according to the prerequisite.

✓ **Application runtime –**

Android Runtime environment is one of the most important part of Android. It contains components like core libraries and the Dalvik virtual machine(DVM). Mainly, it provides the base for the application framework and powers our application with the help of the core libraries.

Like Java Virtual Machine (JVM), **Dalvik Virtual Machine (DVM)** is a register-based virtual machine and specially designed and optimized for android to ensure that a device can run multiple instances efficiently. It depends on the layer Linux kernel for threading and low-level memory management. The core libraries enable us to implement android applications using the standard JAVA or Kotlin programming languages.

✓ **Platform libraries –**

The Platform Libraries includes various C/C++ core libraries and Java based libraries such as Media, Graphics, Surface Manager, OpenGL etc. to provide a support for android development.

- ✓ **Media** library provides support to play and record an audio and video formats.
- ✓ **Surface manager** responsible for managing access to the display subsystem.
- ✓ **SGL** and **OpenGL** both cross-language, cross-platform application program interface (API) are used for 2D and 3D computer graphics.
- ✓ **SQLite** provides database support and **FreeType** provides font support.

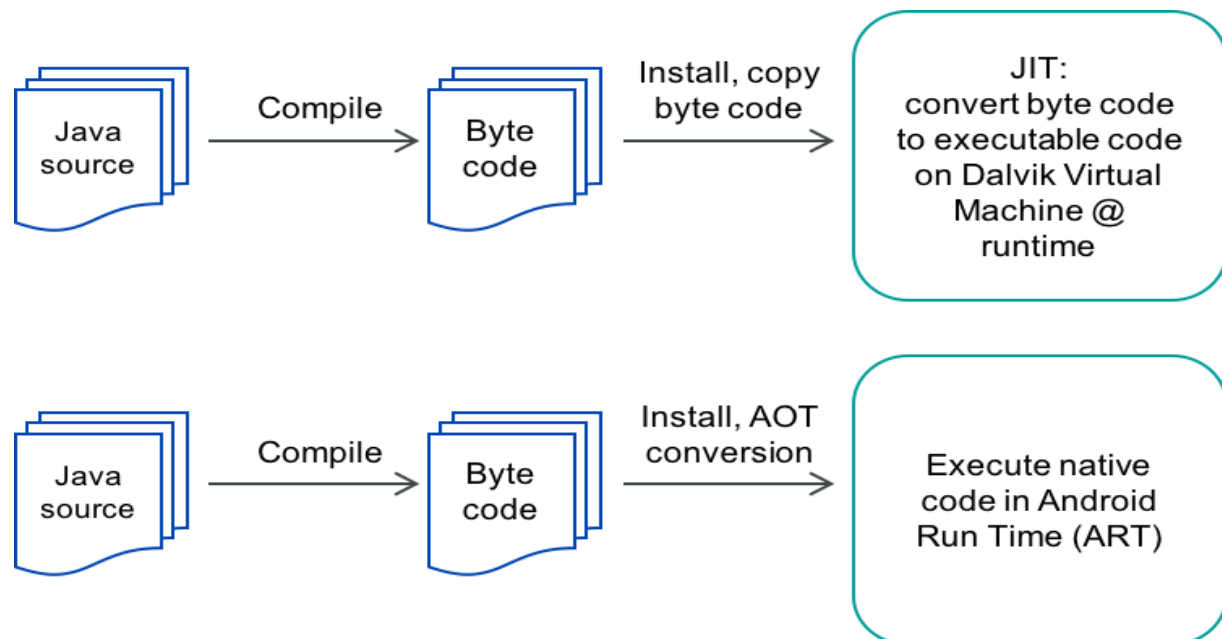
- ✓ **Web-Kit** This open source web browser engine provides all the functionality to display web content and to simplify page loading.
- ✓ **SSL (Secure Sockets Layer)** is security technology to establish an encrypted link between a web server and a web browser.
- ✓ **Linux Kernel** –

Linux Kernel is heart of the android architecture. It manages all the available drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc. which are required during the runtime.

The Linux Kernel will provide an abstraction layer between the device hardware and the other components of android architecture. It is responsible for management of memory, power, devices etc.

The features of Linux kernel are:

- ✓ **Security:** The Linux kernel handles the security between the application and the system.
- ✓ **Memory Management:** It efficiently handles the memory management thereby providing the freedom to develop our apps.
- ✓ **Process Management:** It manages the process well, allocates resources to processes whenever they need them.
- ✓ **Network Stack:** It effectively handles the network communication.
- ✓ **Driver Model:** It ensures that the application works properly on the device and hardware manufacturers responsible for building their drivers into the Linux build.



**Figure 2. 6 Dalvik VM versus Android runtime**

The android developer tools let you create interactive and powerful application for android platform. The tools can be generally categorized into two types.

1. SDK tools
2. Platform tools

#### 1. SDK tools

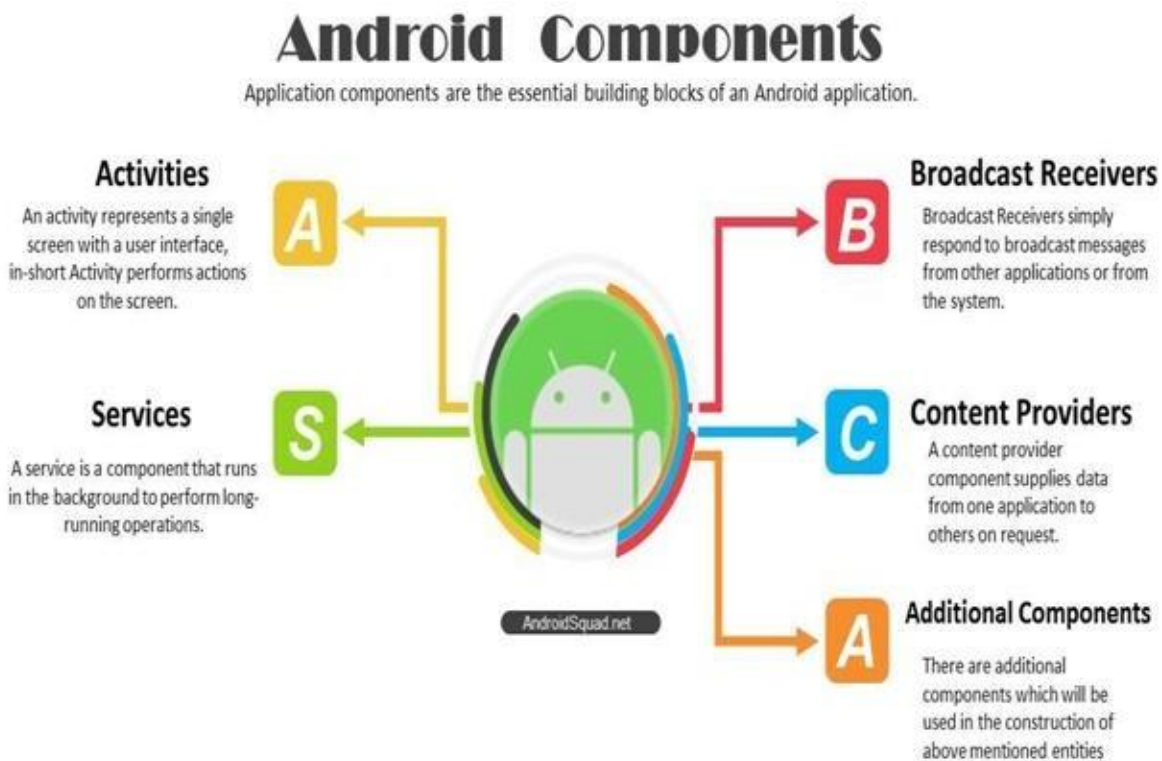
SDK tools are generally platform independent and are required no matter which android platform you are working on. When you install the Android SDK into your system, these tools get automatically installed. The list of SDK tools has been given below –

Sr.No	Tool & description
1	<b>android</b> This tool lets you manage AVDs, projects, and the installed components of the SDK
2	<b>ddms</b> This tool lets you debug Android applications
3	<b>Draw 9-Patch</b> This tool allows you to easily create a NinePatch graphic using a WYSIWYG editor
4	<b>emulator</b> This tools let you test your applications without using a physical device
5	<b>mksdcard</b> Helps you create a disk image (external sdcard storage) that you can use with the emulator
6	<b>proguard</b> Shrinks, optimizes, and obfuscates your code by removing unused code
7	<b>sqlite3</b>

	Lets you access the SQLite data files created and used by Android applications
8	<b>traceview</b> Provides a graphical viewer for execution logs saved by your application
9	<b>Adb</b> Android Debug Bridge (adb) is a versatile command line tool that lets you communicate with an emulator instance or connected Android-powered device.

**Table 2.3. Android Development Tools**

## Main Components of Android



## Figure 2. 7 Main Components of Android

An Android application consists of one or more of the following four classifications:

1. **Activities:** An application that has a visible user interface is implemented via an activity. When you select an application from the Home screen or application launcher, an activity is started.
2. **Services:** You can use a service for any application that needs to persist for a long time, such as a network monitor or update-checking application.
3. **Content providers:** The easiest way to think about content providers is to view them as a database server. A content provider's job is to manage access to persisted data, such as the contacts on a phone. If your application is very simple, you might not necessarily create a content provider, however if you are building a larger application or one which makes data available to multiple activities and/or applications, a content provider is the proscribed means of accessing your data.
4. **Broadcast receivers:** You can launch an Android application to process a specific element of data or respond to an event, such as receiving a text message.

## Overview of object oriented programming using Java

Object-Oriented Programming or OOPs refers to languages that use objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

### OOPs Concepts:

- ✓ Class
- ✓ Objects
- ✓ Data Abstraction
- ✓ Encapsulation
- ✓ Inheritance
- ✓ Polymorphism
- ✓ Dynamic Binding



- ✓ Message Passing

Apart from these concepts, there are some other terms which are used in Object-Oriented design:

- ✓ Coupling
- ✓ Cohesion
- ✓ Association
- ✓ Aggregation
- ✓ Composition

### **1. Class:**

A class is a user-defined data type. It consists of data members and member functions, which can be accessed and used by creating an instance of that class. It represents the set of properties or methods that are common to all objects of one type. A class is like a blueprint for an object.

***For Example:*** Consider the Class of Cars. There may be many cars with different names and brands but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range, etc. So here, Car is the class, and wheels, speed limits, mileage are their properties.

### **2. Object:**

It is a basic unit of Object-Oriented Programming and represents the real-life entities. An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated. An object has an identity, state, and behavior. Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other's data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.

## What is an Object



**Object:** is a bundle of data and its behaviour(often known as methods).

Objects have two characteristics: They have **states** and **behaviors**.

### Examples of states and behaviors

#### Example 1:

**Object:** House

**State:** Address, Color, Area

**Behavior:** Open door, close door

For example “Dog” is a real-life Object, which has some characteristics like color, Breed, Bark, Sleep, and Eats.

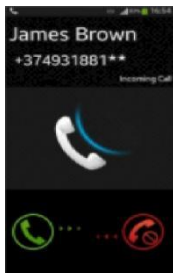


**Figure 2. 8. Example of an object**

### 3. Data Abstraction:

Data abstraction is one of the most essential and important features of object-oriented programming. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation. Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car, but he does not know about how on pressing the accelerator the speed is increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.

## Object Oriented Programming : Abstraction



Abstraction refers to the quality of dealing with ideas rather than events. It basically deals with hiding the details and showing the essential things to the user. If you look at the image here, whenever we get a call, we get an option to either pick it up or just reject it. But in reality, there is a lot of code that runs in the background. So you don't know the internal processing of how a call is generated, that's the beauty of abstraction. Therefore, abstraction helps to reduce complexity. You can achieve abstraction in two ways:

a) Abstract Class

b) Interface

**Abstract class:** Abstract class in java contains the 'abstract' keyword. Now what does the abstract keyword mean? If a class is declared abstract, it cannot be instantiated, which means you cannot create an object of an abstract class. Also, an abstract class can contain abstract as well as concrete methods.

*Note.* You can achieve 0-100% abstraction using abstract class.

To use an abstract class, you have to inherit it from another class where you have to provide implementations for the abstract methods there itself, else it will also become an abstract class.

Let's look at the syntax of an abstract class:

```
1  Abstract class Bob { // abstract class mobile
2  Abstract void run(); // abstract method
```

**Interface:** Interface in java is a blueprint of a class or you can say it is a collection of abstract methods and static constants. In an interface, each method is public and abstract but it does not contain any constructor. Along with abstraction, interface also helps to achieve multiple inheritance.

*Note.* You can achieve 100% abstraction using interfaces.

So an interface basically is a group of related methods with empty bodies. Let us understand interfaces better by taking an example of a 'ParentCar' interface with its related methods.

```
1  public interface ParentCar {
2  public void changeGear( int newGear );
3  public void speedUp ( int increment );
4  public void applyBrakes ( int decrement );
```



**Figure 2. 9 Advantages of Interface**

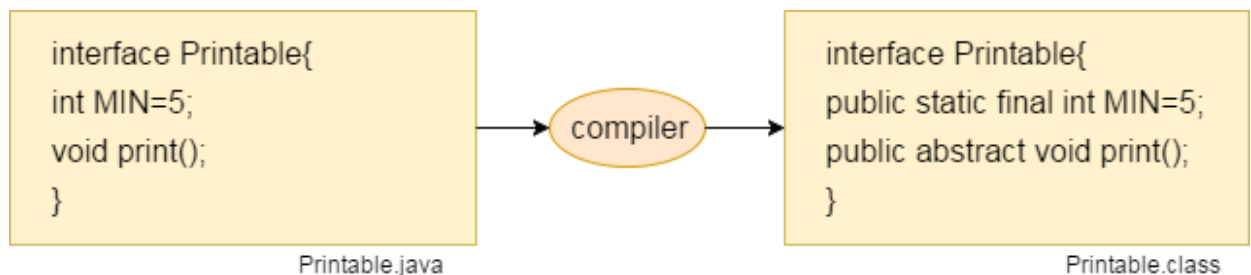
How to declare an interface?

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

### Syntax:

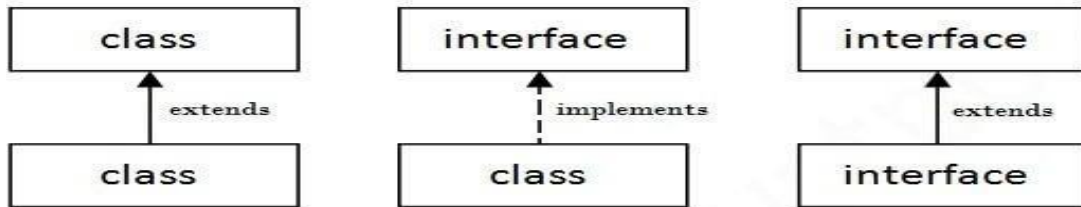
```
interface <interface_name>{  
  
    // declare constant fields  
    // declare methods that abstract  
    // by default.  
}
```

In other words, Interface fields are public, static and final by default, and the methods are public and abstract.



**Figure 2. 10 The relationship between classes and interfaces**

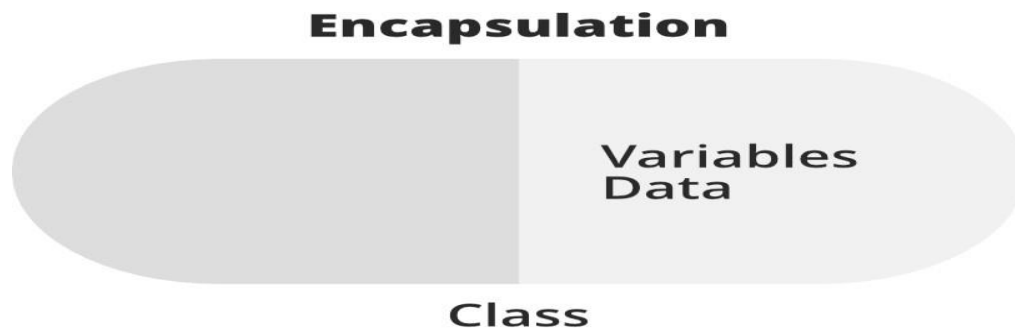
As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



**Figure 2. 10 Example of class extends another class**

#### **4. Encapsulation:**

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. In Encapsulation, the variables or data of a class are hidden from any other class and can be accessed only through any member function of their class in which they are declared. As in encapsulation, the data in a class is hidden from other classes, so it is also known as data-hiding.



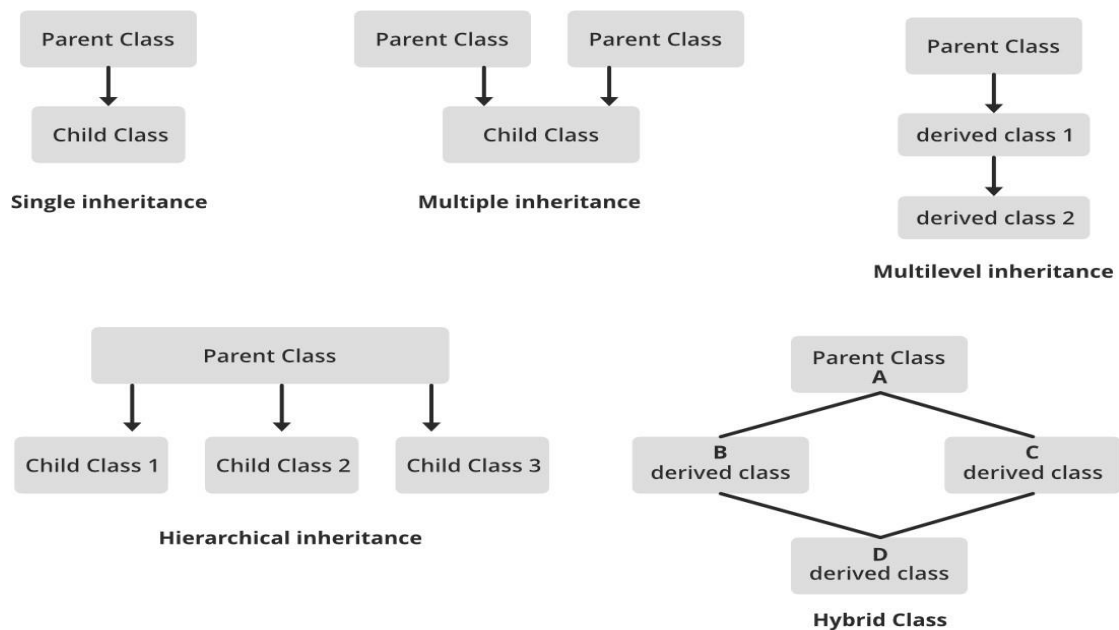
**Figure 2. 11 Example of Encapsulation**

Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section, etc. The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly, the sales section handles all the sales-related activities and keeps records of all the sales. Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of the sales

section. He will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is. Here the data of the sales section and the employees that can manipulate them are wrapped under a single name “sales section”.

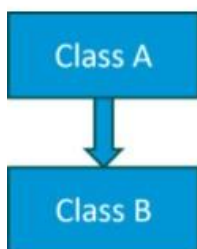
## 5. Inheritance:

Inheritance is an important pillar of OOP(Object-Oriented Programming). The capability of a class to derive properties and characteristics from another class is called Inheritance. When we write a class, we inherit properties from other classes. So when we create a class, we do not need to write all the properties and functions again and again, as these can be inherited from another class that possesses it. Inheritance allows the user to reuse the code whenever possible and reduce its redundancy.



**Figure 2. 12. Types of Inheritance**

### 1. Single Inheritance:



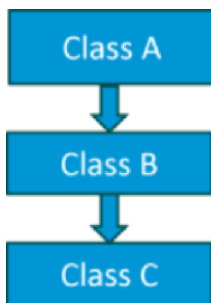
In single inheritance, one class inherits the properties of another. It enables a derived class to inherit the properties and behavior from a single parent class. This will in turn enable code reusability as well as add new features to the existing code.

Here, Class A is your parent class and Class B is your child class which inherits the properties and behavior of the parent class.

```
class A
```

```
class B extends A {
```

## 2. Multilevel Inheritance:



When a class is derived from a class which is also derived from another class, i.e. a class having more than one parent class but at different levels, such type of inheritance is called Multilevel Inheritance

If we talk about the flowchart, class B inherits the properties and behavior of class A and class C inherits the properties of class B. Here A is the parent class for B and class B is the parent class for C. So in this case class C implicitly inherits the properties and methods of class A along with Class B. That's what is

multilevel inheritance.

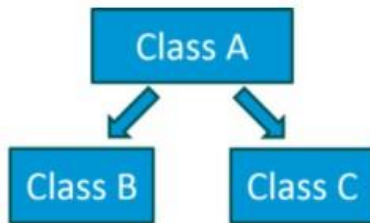
Let's see the syntax for multilevel inheritance in java:

```
class A
```

```
class B extends A {
```

```
class C extends B {
```

### 3. Hierarchical Inheritance:



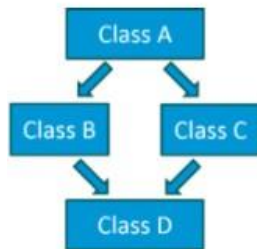
When a class has more than one child classes (sub classes) or in other words, more than one child classes have the same parent class, then such kind of inheritance is known as **hierarchical**.

If we talk about the flowchart, Class B and C are the child classes which are inheriting from the parent class i.e Class A.

Let's see the syntax for hierarchical inheritance in Java:

```
1 | Class A{  
2 | ---  
3 | }  
4 | Class B extends A{  
5 | ---  
6 | }  
7 | Class C extends A{  
8 | ---  
9 | }
```

### 4. Hybrid Inheritance:



Hybrid inheritance is a combination of *multiple* inheritance and *multilevel* inheritance. Since multiple inheritance is not supported in Java as it leads to ambiguity, so this type of inheritance can only be achieved through the use of the interfaces.

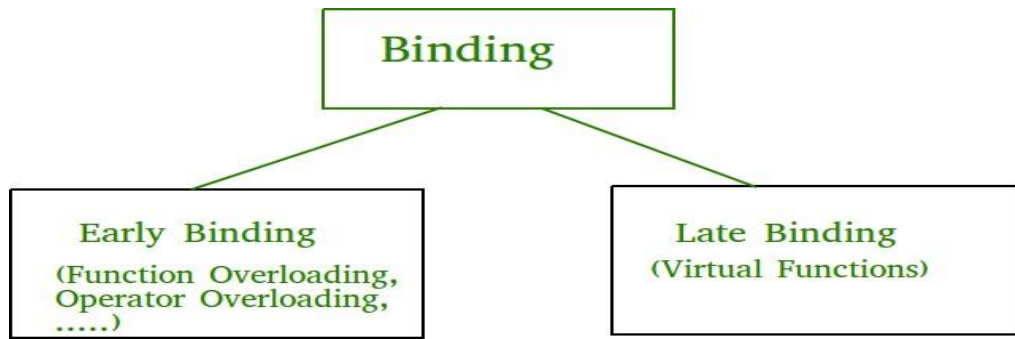
If we talk about the flowchart, class A is a parent class for class B and C, whereas Class B and C are the parent class of D which is the only child class

of B and C.

### 7. Dynamic Binding:

Binding means matching the function call with the correct function definition by the compiler. It takes place either at compile time or at runtime.





**Figure 2. 13. Types of Binding**

- ✓ In early binding, the compiler matches the function call with the correct function definition at compile time. It is also known as Static Binding or Compile-time Binding.
- ✓ In the case of late binding, the compiler matches the function call with the correct function definition at runtime. It is also known as Dynamic Binding or Runtime Binding.

### **8. Message Passing:**

It is a form of communication used in object-oriented programming as well as parallel programming. Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function, and the information to be sent.

**Message Passing** is nothing but sending and receiving of information by the objects same as people exchange information. So this helps in building systems that simulate real life. Following are the basic steps in message passing.

- Creating classes that define objects and its behavior.
- Creating objects from class definitions
- Establishing communication among objects

In OOPs, Message Passing involves specifying the name of objects, the name of the function, and the information to be sent.

### **Coupling:**

- ✓ Coupling refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other.

- ✓ If a class has the details information of another class, there is strong coupling. In Java, we use private, protected, and public modifiers to display the visibility level of a class, method, and field.
- ✓ You can use interfaces for the weaker coupling because there is no concrete implementation.

### **Cohesion:**

- ✓ Cohesion refers to the level of a component which performs a single well-defined task.
- ✓ A single well-defined task is done by a highly cohesive method. The weakly cohesive method will split the task into separate parts.
- ✓ The java.io package is a highly cohesive package because it has I/O related classes and interface.
- ✓ However, the java.util package is a weakly cohesive package because it has unrelated classes and interfaces.

### **Association:**

- ✓ Association represents the relationship between the objects. Here, one object can be associated with one object or many objects.
- ✓ There can be four types of association between the objects:
  - One to One
  - One to Many
  - Many to One, and
  - Many to Many

For example, One country can have one prime minister (one to one), and a prime minister can have many ministers (one to many). Also, many MP's can have one prime minister (many to one), and many ministers can have many departments (many to many). Association can be unidirectional or bidirectional.

### **Aggregation:**

- ✓ Aggregation is a way to achieve Association.
- ✓ Aggregation represents the relationship where one object contains other objects as a part of its state.
- ✓ It represents the weak relationship between objects. It is also termed as a *has-a* relationship in Java.

- ✓ Like, inheritance represents the *is-a* relationship. It is another way to reuse objects.

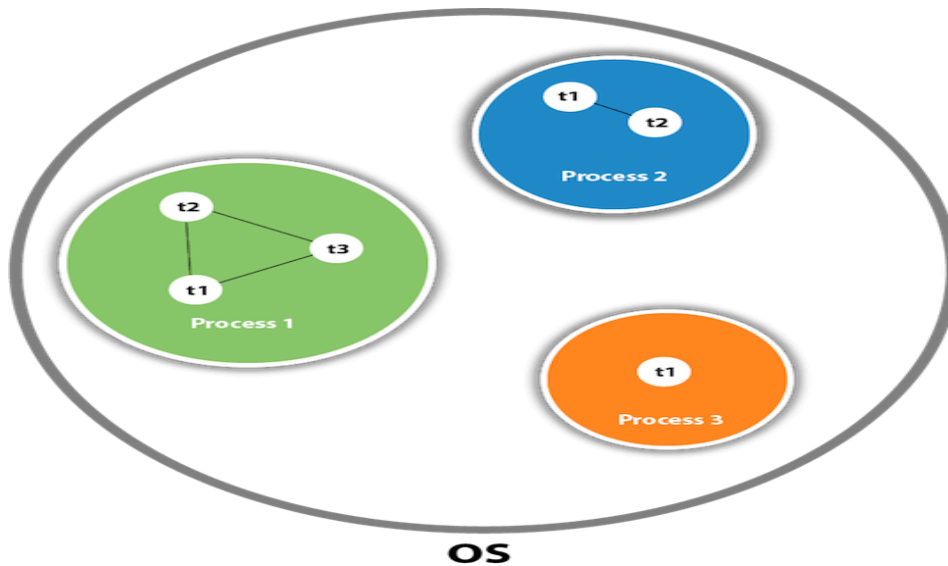
### Composition:

- ✓ Composition allows the reuse of code.
- ✓ Java doesn't support multiple inheritances but by using composition we can achieve it.
- ✓ Composition offers better test-ability of a class.
- ✓ By using composition, we are flexible enough to replace the implementation of a composed class with a better and improved version.
- ✓ By using composition, we can also change the member objects at run time, to dynamically change the behavior of your program.

### Threads

- ✓ Threads allow a program to operate more efficiently by doing multiple things at the same time.
- ✓ Threads can be used to perform complicated tasks in the background without interrupting the main program.
- ✓ A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.
- ✓ **Multithreading in Java** is a process of executing multiple threads simultaneously.
- ✓ However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.
- ✓ Java Multithreading is mostly used in games, animation, etc.
- ✓ Advantages of Java Multithreading
  - 1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.
  - 2) You **can perform many operations together, so it saves time.**
  - 3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.
- ✓ A thread is a lightweight sub process, the smallest unit of processing. It is a separate path of execution.

- ✓ Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.



**Figure 2. 14 Example of Thread**

## **Multitasking**

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

- Process-based Multitasking (Multiprocessing)
- Thread-based Multitasking (Multithreading)

### **1) Process-based Multitasking (Multiprocessing)**

- Each process has an address in memory. In other words, each process allocates a separate memory area.
- A process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

### **2) Thread-based Multitasking (Multithreading)**

- Threads share the same address space.
- A thread is lightweight.

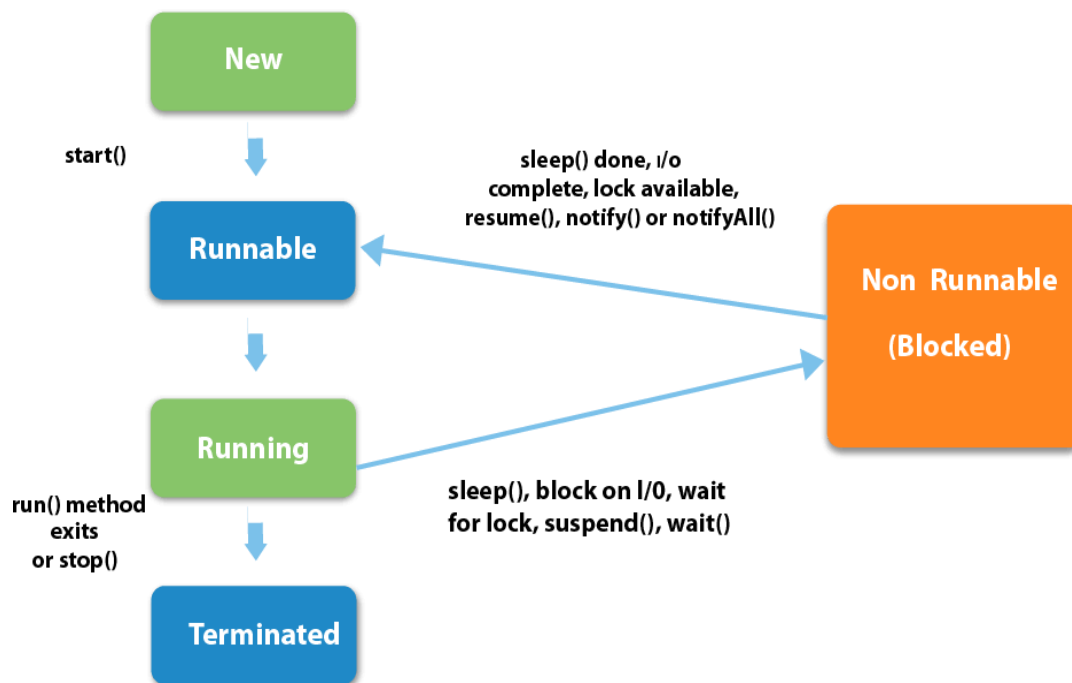
- Cost of communication between the thread is low.

### Life cycle of a Thread (Thread States)

A thread can be in one of the five states. According to sun, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state.

But for better understanding the threads, we are explaining it in the 5 states. The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

- 1.New
- 2.Runnable
3. Running
- 4.Non-Runnable (Blocked)
5. Terminated



**Figure 2. 14 Thread Lifecycle**

- 1) New

The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

## 2) Runnable

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

## 3) Running

The thread is in running state if the thread scheduler has selected it.

## 4) Non-Runnable (Blocked)

This is the state when the thread is still alive, but is currently not eligible to run.

## 5) Terminated

A thread is in terminated or dead state when its run() method exits.

How to create thread:

There are two ways to create a thread:

- By extending Thread class
- By implementing Runnable interface.

## Method Overloading in java

- ✓ If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.
- ✓ If we have to perform only one operation, having same name of the methods increases the readability of the program.
- ✓ Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

## Advantage of method overloading

Method overloading increases the readability of the program.

There are two ways to overload the method in java

1. By changing number of arguments

## 2. By changing the data type

### 1) Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

```
class Adder{
    static int add(int a,int b){return a+b;}
    static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
    public static void main(String[] args){
        System.out.println(Adder.add(11,11));
        System.out.println(Adder.add(11,11,11));
    }
}
```

### 2) Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

```
class Adder{
    static int add(int a, int b){return a+b;}
    static double add(double a, double b){return a+b;}
}
class TestOverloading2{
    public static void main(String[] args){
        System.out.println(Adder.add(11,11));
        System.out.println(Adder.add(12.3,12.6));
    }
}
```

## **Method overriding in Java.**

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

### Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

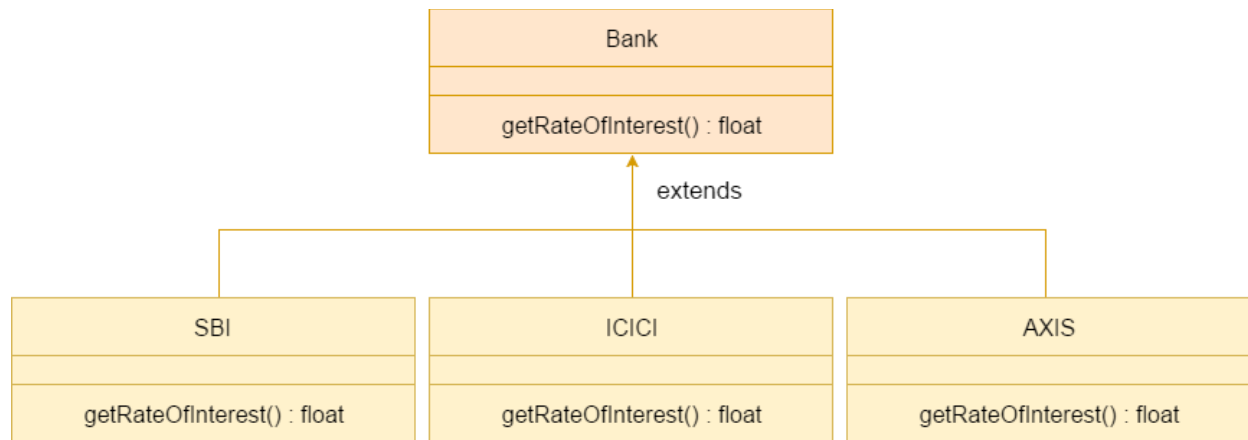
### Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

### A real example of Java Method Overriding

Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7%, and 9% rate of interest.





**Figure 2. 15 A real example of Java Method Overriding**

/Java Program to demonstrate the real scenario of Java Method Overriding

//where three classes are overriding the method of a parent class.

//Creating a parent class.

```
class Bank{
int getRateOfInterest(){return 0;}
}
```

//Creating child classes.

```
class SBI extends Bank{
int getRateOfInterest(){return 8;}
}
```

```
class ICICI extends Bank{
int getRateOfInterest(){return 7;}
}
```

```
class AXIS extends Bank{
int getRateOfInterest(){return 9;}
}
```

//Test class to create objects and call the methods

```
class Test2{
public static void main(String args[]){
SBI s=new SBI();
ICICI i=new ICICI();
```

```
AXIS a=new AXIS();
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
}
}
```

### **Java Virtual Machine:**

- ✓ The JVM manages system memory and provides a portable execution environment for Java-based applications.
- ✓ JVM (Java Virtual Machine) is an abstract machine.
- ✓ It is a specification that provides runtime environment in which java bytecode can be executed.
- ✓ JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent). It is:
  - **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.
  - **An implementation** Its implementation is known as JRE (Java Runtime Environment).
  - **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

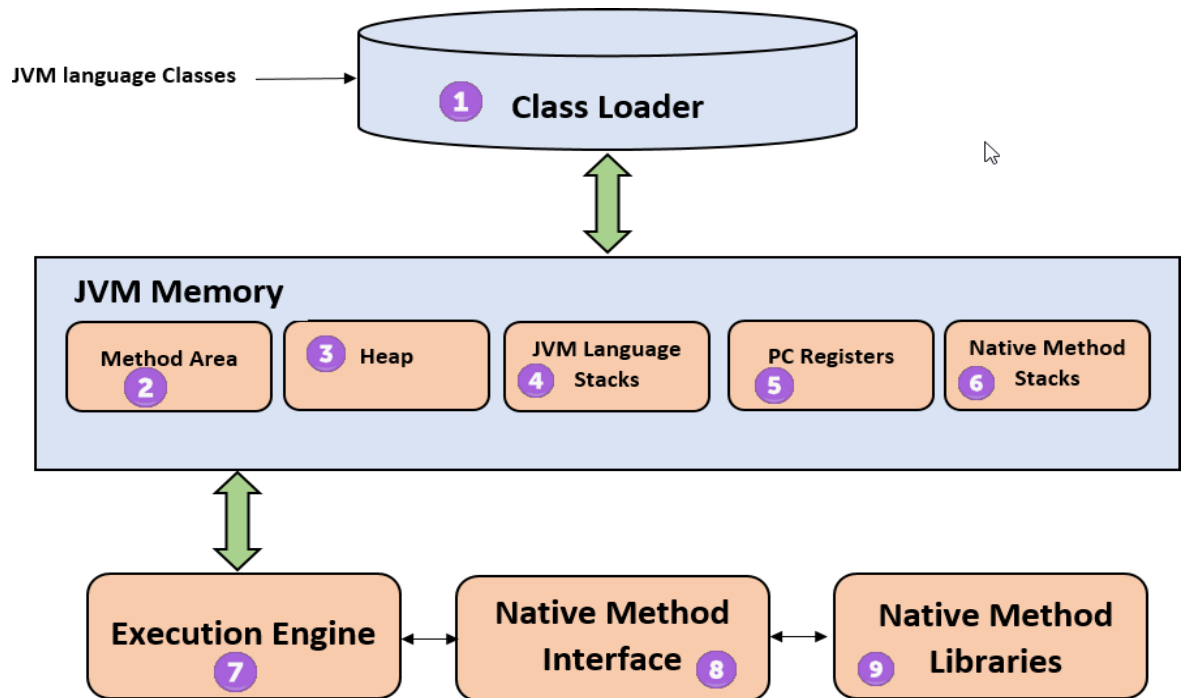
### **The JVM performs following operation:**

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

### **JVM provides definitions for the:**

- Memory area

- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.



**Figure 2.16 Archicture of Java Virtual Machine**

**1) Class Loader** The class loader is a subsystem used for loading class files. It performs three major functions viz. Loading, Linking, and Initialization.

## **2) Method Area**

JVM Method Area stores class structures like metadata, the constant runtime pool, and the code for methods.

## **3) Heap**

All the Objects, their related instance variables, and arrays are stored in the heap. This memory is common and shared across multiple threads.

## **4) JVM language Stacks**

Java language Stacks store local variables, and it's partial results. Each thread has its own JVM stack, created simultaneously as the thread is created. A new frame is created whenever a method is invoked, and it is deleted when method invocation process is complete.

### 5) PC Registers

PC register store the address of the Java virtual machine instruction which is currently executing. In Java, each thread has its separate PC register.

### 6) Native Method Stacks

Native method stacks hold the instruction of native code depends on the native library. It is written in another language instead of Java.

### 7) Execution Engine

It is a type of software used to test hardware, software, or complete systems. The test execution engine never carries any information about the tested product.

### 8) Native Method interface

The Native Method Interface is a programming framework. It allows Java code which is running in a JVM to call by libraries and native applications.

### 9) Native Method Libraries

Native Libraries is a collection of the Native Libraries(C, C++) which are needed by the Execution Engine.

## Software Code Compilation & Execution process

In order to write and execute a software program, you need the following

- ✓ **Editor** – To type your program into, a notepad could be used for this
- ✓ **Compiler** – To convert your high language program into native machine code
- ✓ **Linker** – To combine different program files reference in your main program together.
- ✓ **Loader** – To load the files from your secondary storage device like Hard Disk, Flash Drive, CD into RAM for execution. The loading is automatically done when you execute your code.
- ✓ **Execution** – Actual execution of the code which is handled by your OS & processor.



**SCHOOL OF COMPUTING**  
**DEPARTMENT OF COMPUTER SCIENCE**  
**AND ENGINEERING**

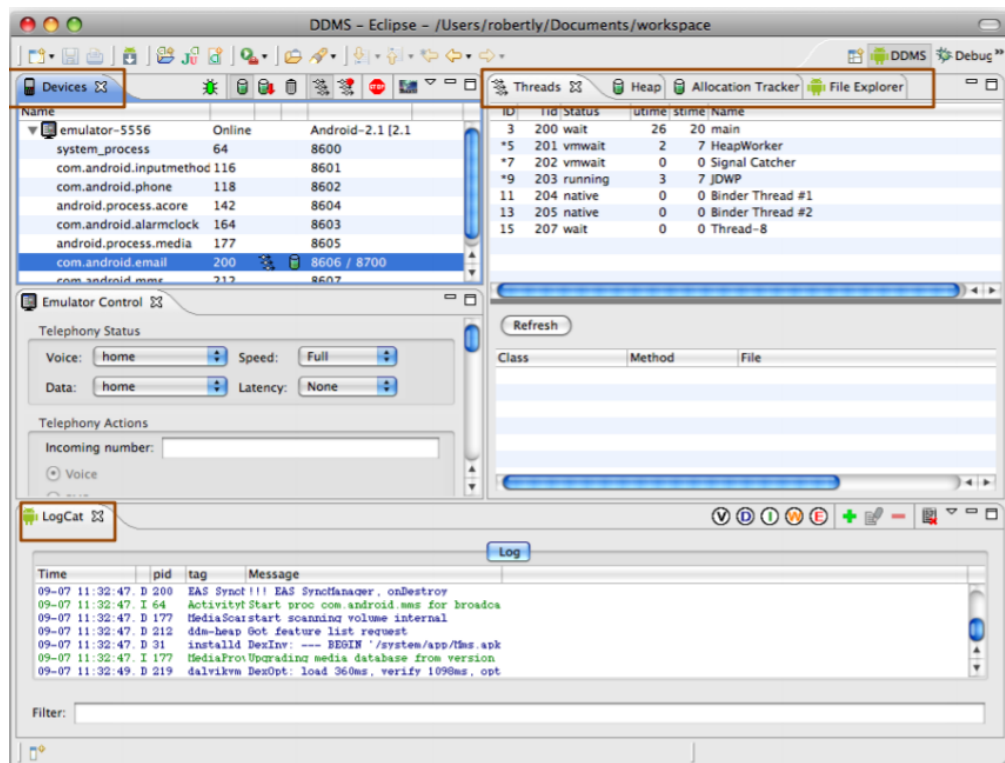
**Mobile Application Development SBS1603**

## UNIT III

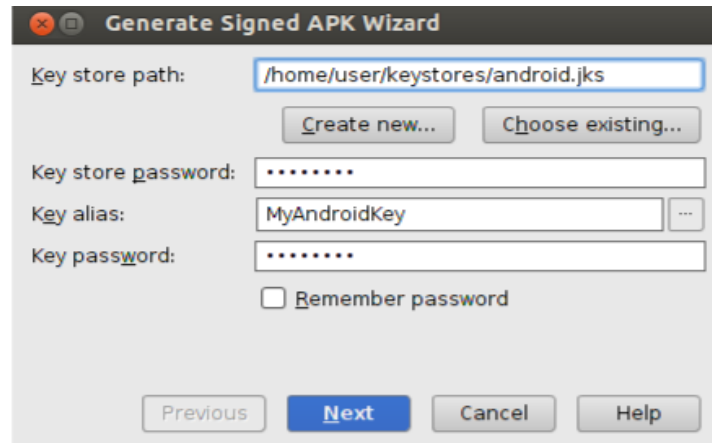
Development Tools: Installing and using Eclipse with ADT plug-in, Installing Virtual machine for Android sandwich/Jelly bean (Emulator), configuring the installed tools, creating an android project – Hello Word, run on emulator, Deploy it on USB-connected Android device. The manifest file - Layout resource - User Interface Architecture: Application context, intents, Activity life cycle, multiple screen sizes.

### ANDROID DEVELOPMENT TOOLS

- The Android Software Development Kit (Android SDK) contains the necessary tools to create, compile and package Android applications.
- The Android SDK contains the Android debug bridge (adb), which is a tool that allows you to connect to a virtual or real Android device, for the purpose of managing the device or debugging your application.
- The Android SDK contains a tool called dx which converts Java class files into a .dex (Dalvik Executable) file.
- The Android tooling uses Gradle as build system.
- Android Studio includes a debugging tool called the Dalvik Debug Monitor Server (DDMS), which provides port-forwarding services, screen capture on the device, thread and heap information on the device, logcat, process, and radio state information, incoming call and SMS spoofing, location data spoofing, and more.



- Android requires that all apps be digitally signed with a certificate before they can be installed. Android uses this certificate to identify the author of an app, and the certificate does not need to be signed by a certificate authority.
- Android apps often use self-signed certificates. The app developer holds the certificate's private key.



- **ANDROID IDE (INTEGRATED DEVELOPMENT ENVIRONMENT)**
  - Eclipse + ADT Plugin
- **INSTALLING AND USING ECLIPSE WITH ADT PLUG-IN**

#### Installing the Eclipse Plugin

Android offers a custom plugin for the Eclipse IDE, called Android Development Tools (ADT). This plugin provides a powerful, integrated environment in which to develop Android apps. It extends the capabilities of Eclipse to let you quickly set up new Android projects, build an app UI, debug your app, and export signed (or unsigned) app packages (APKs) for distribution.

If you need to install Eclipse, you can download it from [eclipse.org/mobile](http://eclipse.org/mobile).

**Note:** If you prefer to work in a different IDE, you do not need to install Eclipse or ADT. Instead, you can directly use the SDK tools to build and debug your application.

#### Download the ADT Plugin

1. Start Eclipse, then select Help > Install New Software.
2. Click Add, in the top-right corner.
3. In the Add Repository dialog that appears, enter "ADT Plugin" for the Name and the following URL for the Location:
4. <https://dl-ssl.google.com/android/eclipse/>
5. Click OK.
6. If you have trouble acquiring the plugin, try using "http" in the Location URL, instead of "https" (https is preferred for security reasons).
7. In the Available Software dialog, select the checkbox next to Developer Tools and click Next.

8. In the next window, you'll see a list of the tools to be downloaded. Click **Next**.
9. Read and accept the license agreements, then click **Finish**.
10. If you get a security warning saying that the authenticity or validity of the software can't be established, click **OK**.
11. When the installation completes, restart Eclipse.

## Configure the ADT Plugin

1. Once Eclipse restarts, you must specify the location of your Android SDK directory:
2. In the "Welcome to Android Development" window that appears, select **Use existing SDKs**.
3. Browse and select the location of the Android SDK directory you recently downloaded and unpacked.
4. Click **Next**.
5. Your Eclipse IDE is now set up to develop Android apps, but you need to add the latest SDK platform tools and an Android platform to your environment. To get these packages for your SDK, continue to Adding Platforms and Packages.

## Troubleshooting Installation

1. If you are having trouble downloading the ADT plugin after following the steps above, here are some suggestions:
2. If Eclipse can not find the remote update site containing the ADT plugin, try changing the remote site URL to use http, rather than https. That is, set the Location for the remote site to:
3. <http://dl-ssl.google.com/android/eclipse/>
4. If you are behind a firewall (such as a corporate firewall), make sure that you have properly configured your proxy settings in Eclipse. In Eclipse, you can configure proxy information from the main Eclipse menu in **Window** (on Mac OS X, **Eclipse**) > **Preferences** > **General** > **Network Connections**.

If you are still unable to use Eclipse to download the ADT plugin as a remote update site, you can download the ADT zip file to your local machine and manually install it:

1. 

Package	Size	MD5 Checksum
<a href="#">ADT-21.1.0.zip</a>	13564671 bytes	f1ae183891229784bb9c33bcc9c5ef1e

ad the ADT Plugin zip file (do not unpack it):

2. Start Eclipse, then select **Help** > **Install New Software**.
3. Click **Add**, in the top-right corner.
4. In the Add Repository dialog, click **Archive**.
5. Select the downloaded ADT-21.1.0.zip file and click **OK**.
6. Enter "ADT Plugin" for the name and click **OK**.
7. In the Available Software dialog, select the checkbox next to Developer Tools and click **Next**.
8. In the next window, you'll see a list of the tools to be downloaded. Click **Next**.



9. Read and accept the license agreements, then click **Finish**.
10. If you get a security warning saying that the authenticity or validity of the software can't be established, click **OK**.
11. When the installation completes, restart Eclipse.

To update your plugin once you've installed using the zip file, you will have to follow these steps again instead of the default update instructions.

## INSTALLING VIRTUAL MACHINE FOR ANDROID JELLY BEAN (EMULATOR)

Initially, install VirtualBox on your Windows PC.

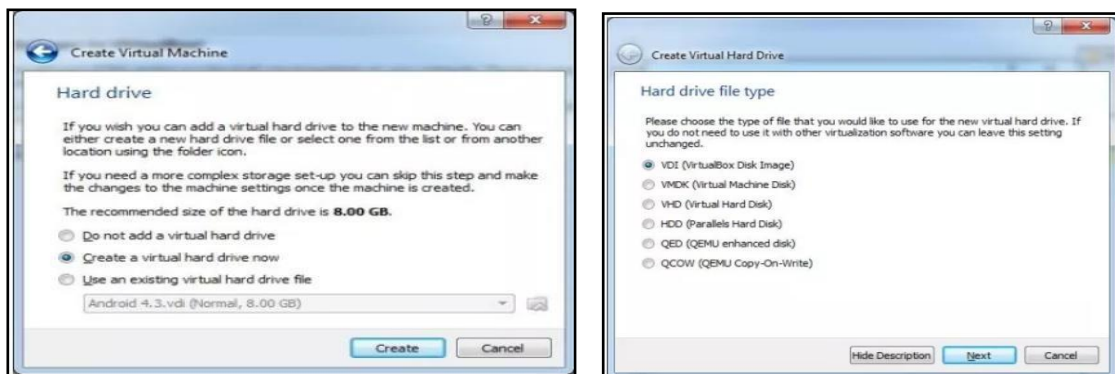
### Instructions to Install Android 4.3 on a windows computer

**Step 1:** Install and open **Virtualbox**.

**Step 2:** Click on new and enter a name and the operating system details for the virtual machine. Select type as Linux and version as other and click next.

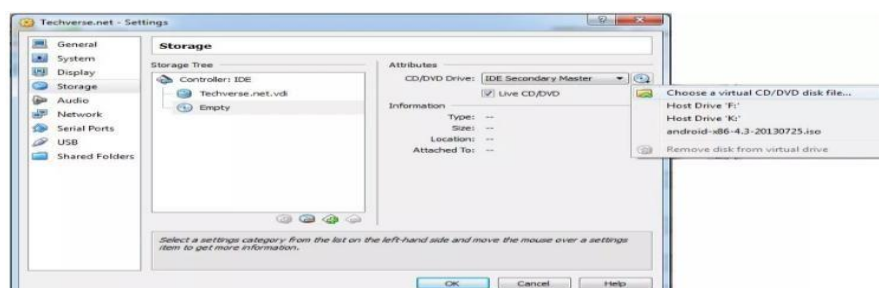
**Step 3:** Enter the amount of ram you would like to allot for the virtual machine and click next. Android 4.3 requires at least 1Gb of ram but its not necessary.

**Step 4:** In the next window select — Create a virtual hard disk — and then select VDI .



**Step 5:** According to your space requirement you can either select dynamically allocated or fixed size for your storage space . i selected fixed size because i want to allocate only 8Gb of storage space to android.

**Step 6:** Your virtual machine is now set. all you need to do is add the location of the Android 4.3 image file. Click on the settings button in virtualbox. Under the settings navigate to storage, below the storage tree select empty and click on the disk image and select — **choose a virtual CD/DVD disk file** — and select the android 4.3 image. Check the Live CD/DVD box and click ok.



**Step 7:** Double click on your virtual machine to start it and click OK for all the dialog boxes . Select Install Android-X86 to hard disk and click OK for all the dialog boxes .

**Step 8:** in the next window you have to create a partition for installing Android. The new partition will not mess up anything with your windows computer. From now onwards you have to use your up, down, left and right keys on your keyboard to Select **-create/modify partitions -** and click OK.

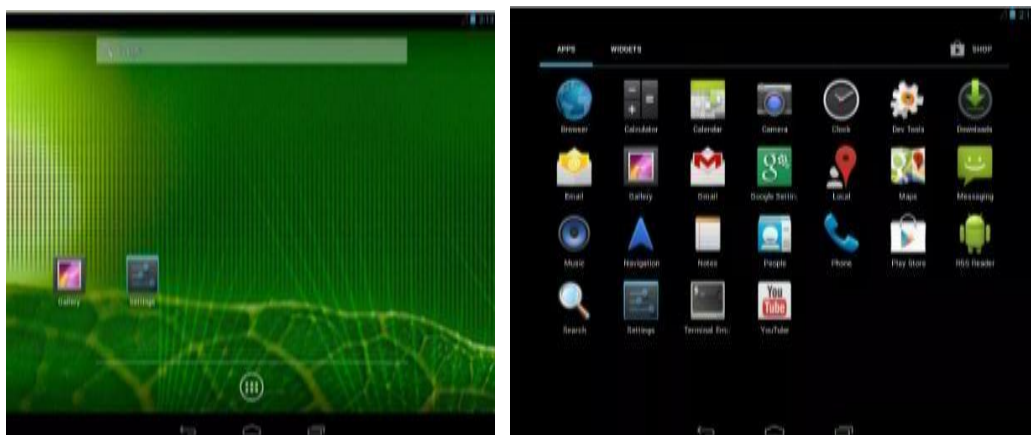
**Step 9:** In the next windows select new > primary and then specify the size of the new partition

**Step 10:** Your new partition has been created. Select write and press enter and type **-yes -** and press enter again when prompted. In the next window select quit and press enter.

**Step 11:** In the next window select the Sda1 and press enter. select the ext3 file system and press enter. When prompted to install grub loader select yes. Select now when prompted to make system directory as read-write.

**Step 12:** Now android 4.3 is successfully installed on your virtual machine. select Run Android 4.3 and press enter. click OK for any other dialog boxes that appear. You will now see the android loading screen.

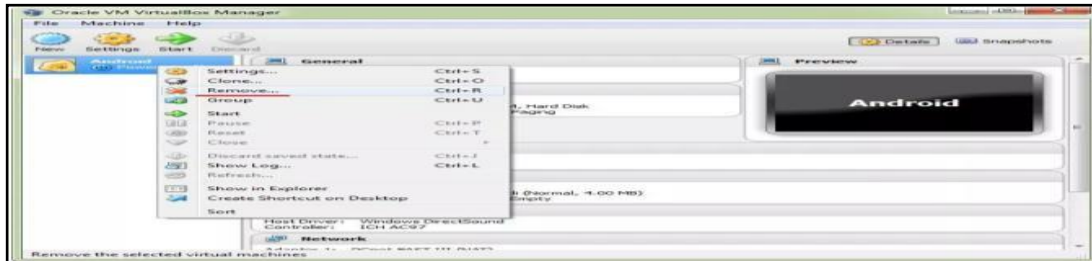
**Step 13:** Select your language and enter, now fill in the Gmail details and all the details



that are asked.

**Step 14:** Now We have successfully installed Android 4.3 on windows computer.

- In order to uninstall the android 4.3 virtual box, right click on the virtual machine and select remove. Next select  
–**delete all files** to remove Android 4.3 completely from PC.



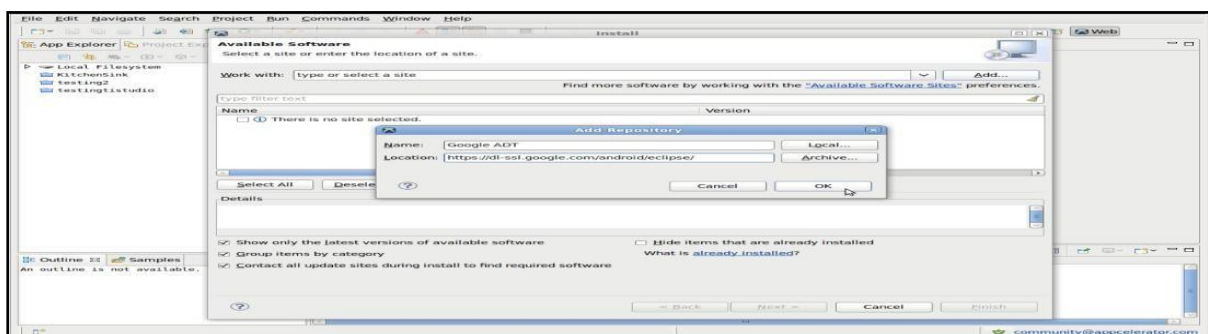
## ➤ CONFIGURING THE INSTALLED TOOLS

Install the Eclipse Java

Development Tools Plugin

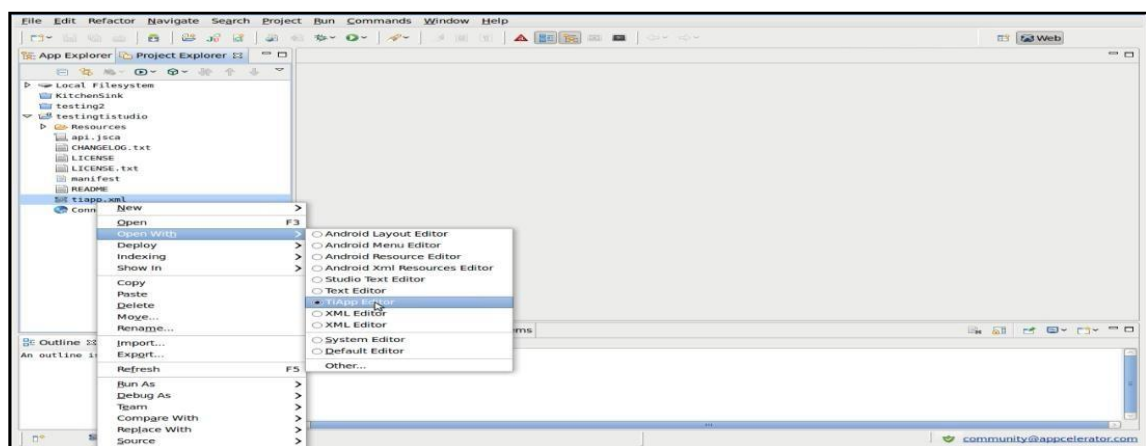
Install the ADT plugin

- On Google's official Android Developers website, under the section Downloading the ADT Plugin, copy the update URL to the clipboard
- From the Studio menu bar, select **Help > Install New Software...**
- Click the **Add** button, to add the Google ADT Plugin update site
- In the **Name** field, enter something descriptive, such as **Google ADT**
- Paste the Google ADT update site URL, copied to the clipboard in the previous step, into the **Location** field
- Click the **OK** button



- Using the **Work with** drop-down menu, select the **Google ADT** entry that you added in the previous step

- Wait for the package list to be populated
  - Select all the resulting ADT packages
  - Click the **Next** button
  - Click the **Next** button on the **Install Details** screen that follows
  - Select each package in turn from the left-hand pane and accept the respective license agreement
  - Click the **Finish** button
  - Once the installation is complete, click the **Restart Now** button
- **CONFIGURE STUDIO TO USE THE SDKS**
- Using the perspective icon(s) in the top-right hand corner, select the one titled **Web**
  - Using the **Project Explorer** tab in the left-hand pane, right-click an existing project and select **Open Project**
  - Browse the resulting project file list, right-click tiapp.xml in root of project, select **Open With > TiApp Editor**



- Choose your preferred Titanium SDK version from the **SDK Version** drop-down list
- Check the Android SDK / Target Android Platform section of the Titanium Compatibility Matrix, to determine which Android versions are compatible with your chosen Titanium SDK. For example, Titanium SDK 1.6.X is compatible with Android versions 1.6 to 2.3. This information will be needed for the configuration in the following steps
- Close tiapp.xml
- From the Studio menu, select **Window > Preferences** or **Studio >**

**Preference** for Mac OS X to open the **Preferences** dialog

- Navigate to **Android**
- Click the **Browse...** button to configure the **Android SDK Location**
- Select a target Android SDK from the list, ensuring that its version is within the range compatible with the Titanium SDK you have chosen, as discovered in the previous step
- Navigate to **Studio > Platforms > Android**
- Click the **Browse...** button to configure the **Android SDK Home**
- Select a target Android SDK from the **Default Android SDK** drop-down list, ensuring that its version is within the range compatible with the Titanium SDK you have chosen, as discovered in the previous step
- Click **OK** to save preference changes

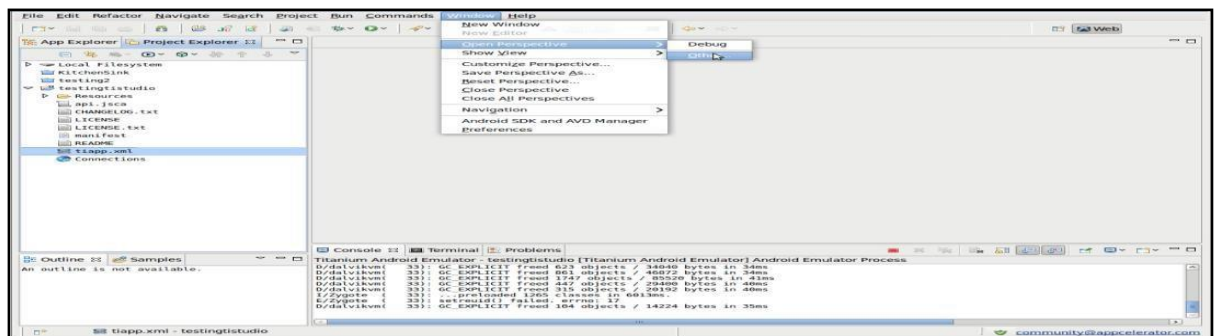
As explained in the [Android SDK / Target Android Platform](#), if you require advanced Android features, such Maps, remember to choose a target that includes the enhanced Google APIs, listed as **Google APIs** in the **Default**

**Android SDK**

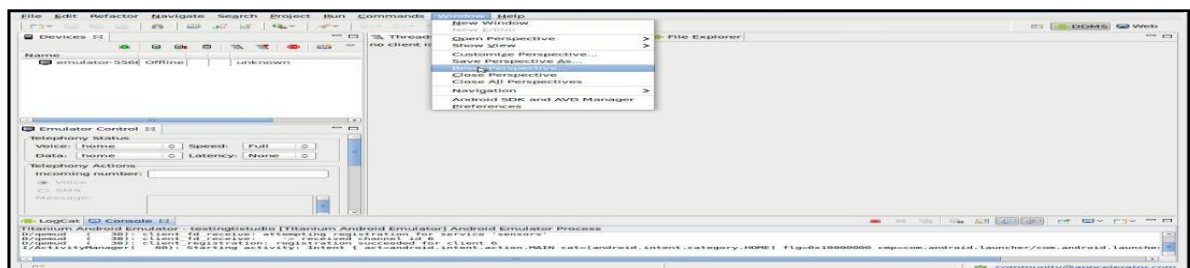
list Launch the emulator and

app

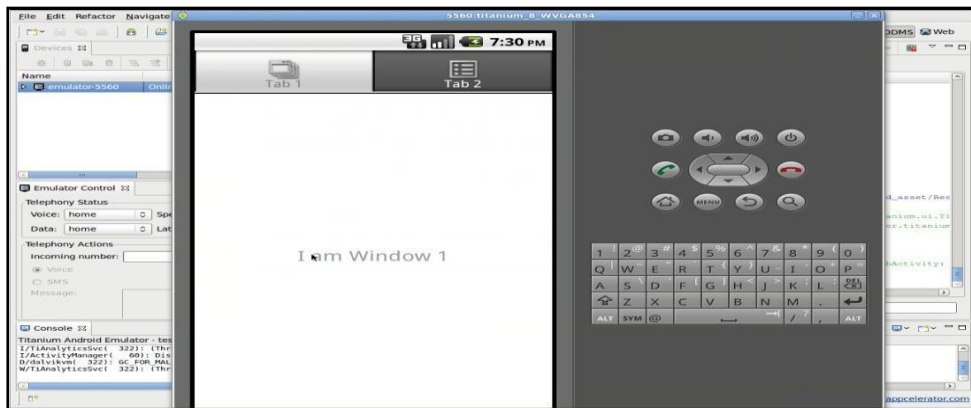
- Using the **Project Explorer**, select the project that was opened earlier
- Using the **Launch** toolbar button, located between the **Project Explorer** tab and its file list, select **Android Emulator** to launch the project app
- Add the DDMS perspective button
- While the emulator boots, open the ADT perspective. From the Studio menu bar, select the **Window > Open Perspective > Other...**
- Select **DDMS** (which stands for, Dalvik Debug Monitor Server) from the list of available perspectives
- To ensure that the perspective is in its default state, select the **Window > Reset Perspective...** menu



Click the **OK** button, when the resulting *Do you want to reset the current DDMS perspective to its defaults?* dialog displays



- Click the **LogCat** viewer tab above the bottom pane, to watch the console output while the emulator boots
- If you wish, you may relocate this tab to the main pane, next to the **File Explorer** tab, using a simple drag and drop gesture
- To show only Ti.API log messages, create a logcat filter using the green *plus* icon in the logcat toolbar and set the by Log Tag field to "TiAPI". See [Reading and Writing Logs](#) for more information about logcat filters.
- Once booted, unlock the emulator and wait for the app to launch

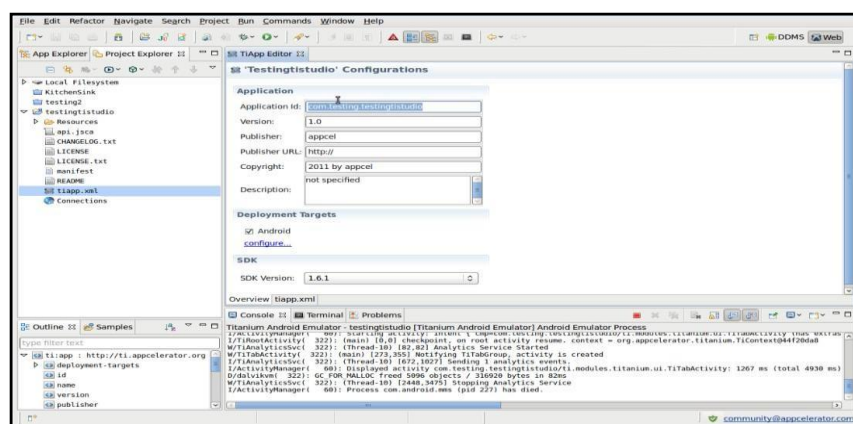


- Now that the emulator is running, select it from the list of devices in the left-hand pane, and inspect it using the tools
- For example, select the File Explorer tab, and navigate to the directory `/data/data/vourAppId`

The `/data/data/yourAppId` directory is equivalent to

`Titanium.Filesystem.applicationDataDirectory AppId` was defined when

the project was created, as shown in the **TiApp Editor** (see below)

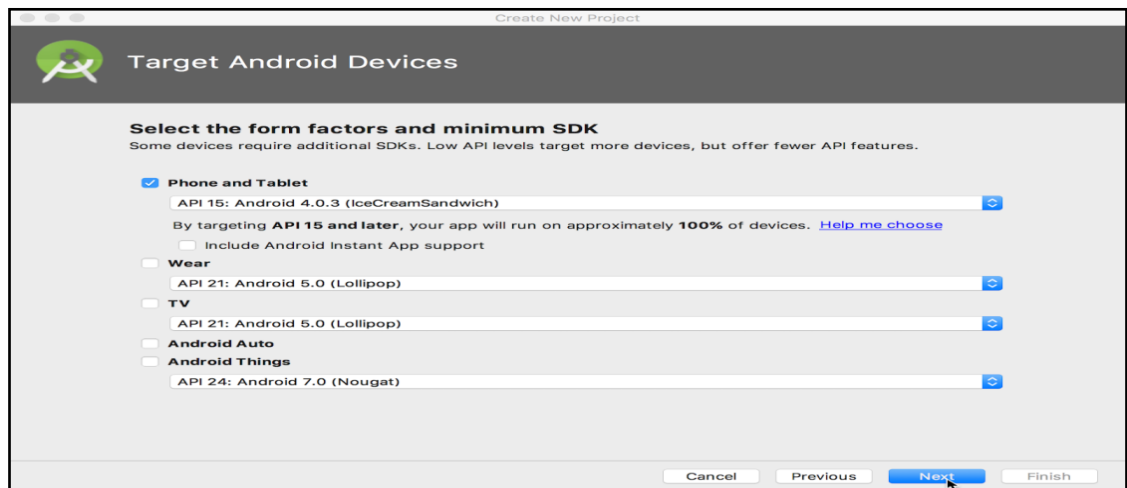


## ➤ CREATING AN ANDROID PROJECT

## ➤ CREATE THE APP PROJECT



1. Open Android Studio if it is not already opened.
2. In the main **Welcome to Android Studio** window, click **Start a new Android Studio project**.
3. In the **Create Android Project** window, enter **Hello World** for the **Application name**.
4. Verify that the default **Project location** is where you want to store your Hello World app and other Android Studio projects, or change it to your preferred directory.
5. Accept the default **android.example.com** for **Company Domain**, or create a unique company domain. If you are not planning to publish your app, you can accept the default. Be aware that changing the package name of your app later is extra work.
6. Leave unchecked the options to **Include C++ support** and **Include Kotlin support**, and click **Next**.
7. On the **Target Android Devices** screen, **Phone and Tablet** should be selected. Ensure that **API 15: Android 4.0.3 IceCreamSandwich** is set to Minimum SDK; if not, use the popup menu to set it.



These are the settings. As of this writing, these settings make Hello World app compatible with 97% of Android devices active on the Google Play Store.

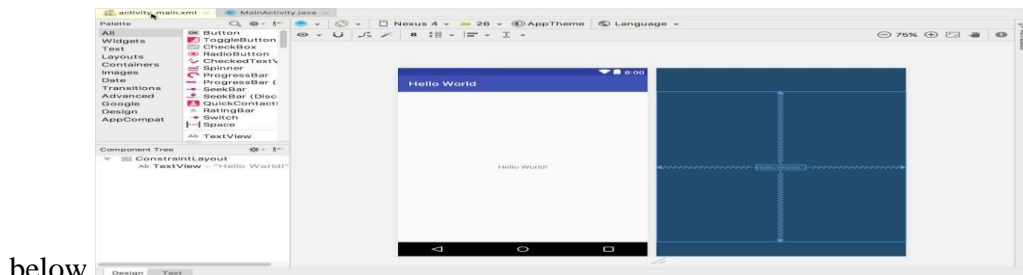
8. Leave unchecked the **Include Instant App support** and all other options. Then click **Next**. If your project requires additional components for your chosen target SDK, Android Studio will install them automatically.
9. The **Add an Activity** window appears. An Activity is a single, focused thing

that the user can do. It is a crucial component of any Android app. An Activity typically has a layout associated with it that defines how UI elements appear on a screen. Android Studio provides Activity templates to help you get started. For the Hello World project, choose **Empty Activity** as shown below, and click **Next**.

10. The **Configure Activity** screen appears (which differs depending on which template you chose in the previous step). By default, the empty Activity provided by the template is named MainActivity. You can change this if you want, but this lesson uses MainActivity.
11. Make sure that the **Generate Layout file** option is checked. The layout name by default is activity\_main. You can change this if you want, but this lesson uses activity\_main.
12. Make sure that the **Backwards Compatibility (App Compat)** option is checked. This ensures that your app will be backwards-compatible with previous versions of Android.
13. Click **Finish**.

Android Studio creates a folder for your projects, and builds the project with [Gradle](#). The Android Studio editor appears. Follow these steps:

1. Click the **activity\_main.xml** tab to see the layout editor.
2. Click the layout editor **Design** tab, if not already selected, to show a graphical rendition of the layout as shown



below.

3. Click the **MainActivity.java** tab to see the code editor as shown below.

➤ **Explore the Project > Android pane**



1. If not already selected, click the **Project** tab in the vertical tab column on the left side of the Android Studio window. The Project pane appears.
2. To view the project in the standard Android project hierarchy, choose **Android** from the popup menu at the top of the Project pane, as shown below.


### ➤ **EXPLORE THE MANIFESTS FOLDER**

The manifests folder contains files that provide essential information about your app to the Android system, which the system must have before it can run any of the app's code.

1. Expand the **manifests** folder.
2. Open the **AndroidManifest.xml** file.

The AndroidManifest.xml file describes all of the components of your Android app. All components for an app, such as each Activity, must be declared in this XML file. In other course lessons you will modify this file to add features and feature permissions. For an introduction, see App Manifest Overview.


### ➤ **RUN ON EMULATOR**

1. Let's create an Android virtual device (AVD). In order to run an emulator on your computer, you have to create a configuration that describes the virtual device. In Android Studio, select **Tools > Android > AVD Manager**, or click the AVD Manager icon  in the toolbar. The **Your Virtual Devices** screen appears. If you've already created virtual devices, the screen shows them; otherwise you see a blank list.
2. Click the **+Create Virtual Device**. The **Select Hardware** window appears showing a list of pre-configured hardware devices. For each device, the table provides a column for its diagonal display size (**Size**), screen resolution in pixels (**Resolution**), and pixel density (**Density**).
3. Choose a device such as **Nexus 5x** or **Pixel XL**, and click **Next**. The **System Image** screen appears.
4. Click the **Recommended** tab if it is not already selected, and choose which version of the Android system to run on the virtual device (such as **Oreo**). Click the link to start the download, and click **Finish** when it's done.
5. After choosing a system image, click **Next**. The **Android Virtual Device (AVD)**

window appears. You can also change the name of the AVD. Check your configuration and click **Finish**.

### ➤ **Run the app on the virtual device**

Let's run your Hello World app.

1. In Android Studio, choose **Run > Run app** or click the **Run** icon  in the toolbar.
2. The **Select Deployment Target** window, under **Available Virtual Devices**, select the virtual device, which you just created, and click **OK**



The emulator starts and boots just like a physical device. Your app builds, and once the emulator is ready, Android Studio will upload the app to the emulator and run it.

### ➤ **DEPLOY IT ON USB-CONNECTED ANDROID DEVICE**

#### **Configure the Android device**

In order to install an application directly to your device, you need to configure it to use a USB connection. The configuration settings vary by device.

For Android 4.2 and later devices, you need to enable **Developer options** by opening **Settings**, click **About** then click the **Build number** item seven times. If you do not do this, you will not see the **Developer options** item in **Settings**.

1. Open **Settings**.
2. Click **Security**.
3. Enable **Unknown sources**, that is, check this option. This permits the device to install apps that do not originate from Google Play.
4. Back out to **Settings**.
5. Click **Developer options**.

6. If available: Set the switch in the title bar to on.
7. Enable **USB debugging**, that is, check this option. This permits the device to install apps over a USB connection.
8. Optional: Enable **Stay awake**, that is, check this option. This option keeps the screen on and disables the lock screen while the device is connected to USB.
9. Optional: Enable **Allow mock locations**, that is, check this option. This option creates fake GPS locations to test location services.
10. Back out of or close **Settings**.

### Install the USB driver (Windows only)

Developers on Windows may need to install a USB driver specific to the manufacturer and model of the device on which they'll be testing. The driver enables your Windows computer to communicate with your Android device. Google provides download links to the drivers at [Android Developer: OEM USB Drivers](#).

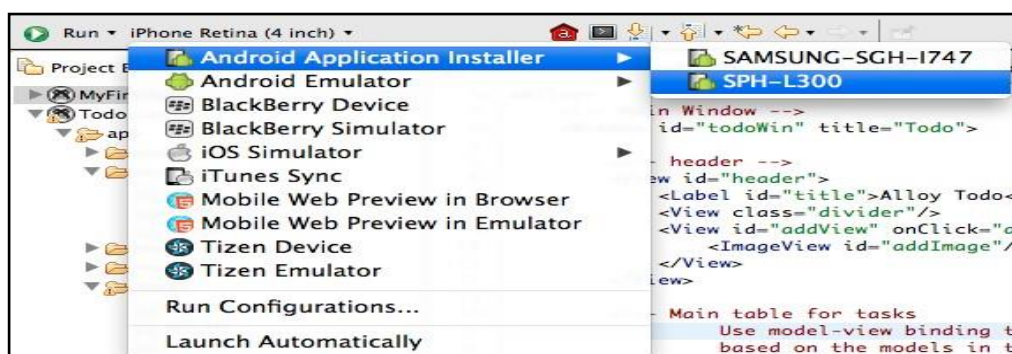
#### ➤ Connect the device

Connect the Android device to your computer using an USB cord. Note that some USB cables are only power cables and do not allow communications with the device. Make sure you use a USB cable that allows a data connection.

For 4.2 devices, an "Allow USB debugging?" dialog will appear once connected via USB. Click the **OK** button.

Deploy the application using Axway Appcelerator Studio

Once you have configured your device and connected it to your computer's USB port, you are ready to deploy your app to it.



In Studio, first select the project in the **Project Explorer** view, then in the global tool bar, select **Run** from the **Launch Mode** drop-down list and an Android device from the **Target** drop-down list under the **Android Application Installer**

category. If the **Launch Automatically** option is enabled under the **Target** dropdown list, the application will be automatically launched after the device is selected. If not, you need to click the **Run** button to start the build process. Your app will be built, installed to your device and automatically launched.

## ➤ **USER INTERFACE ARCHITECTURE**

### ➤ **APPLICATION CONTEXT**

Application Context is It is an instance which is the singleton and can be accessed in an activity via **getApplicationContext()**. This context is tied to the lifecycle of an application. The application context can be used where you need a context whose lifecycle is separate from the current context or when you are passing a context beyond the scope of an activity.

### ➤ **Activity Context**

This context is available in an activity. This context is tied to the lifecycle of an activity. The activity context should be used when you are passing the context in the scope of an activity or you need the context whose lifecycle is attached to the current context.

### ➤ **getContext() in ContentProvider**

This context is the application context and can be used similar to the application context. This can be accessed via **getContext()** method.

### ➤ **INTENTS**

- Intent is a simple message object that is used to communicate between android components such as activities, content providers, broadcast receivers and services. Intents are also used to transfer data between activities.
- The implicit intent is the intent where instead of defining the exact components, you define the action that you want to perform for different activities.
- An explicit intent is Intent where you explicitly define the component that needs to be called by the Android System.
- An explicit intent is one that you can use to launch a specific app component, such as a particular activity or service in your app.

### ➤ **MULTIPLE SCREEN SIZES**

- Android devices come in all shapes and sizes, so your app's layout needs to be flexible. That is, instead of defining your layout with rigid dimensions that assume a certain

screen size and aspect ratio, your layout should gracefully respond to different screen sizes and orientations.

- The best way to create a responsive layout for different screen sizes is to use Constraint Layout as the base layout in your UI. Constraint Layout allows you to specify the position and size for each view according to spatial relationships with other views in the layout. This way, all the views can move and stretch together as the screen size changes.
- The easiest way to build a layout with Constraint Layout is to use the Layout Editor in Android Studio. It allows you to drag new views to the layout, attach their constraints to the parent view and other sibling views, and edit the view's properties, all without editing any XML by hand.

## ➤ **USER INTERFACE DESIGN**

### • **FORM WIDGETS**

- Widgets enable users to interact with an Android Studio application page. There are various kinds of widgets, such as Buttons and TextViews.
- To see all the widgets at your disposal, create a new application project called —Widgets and select "empty activity". Call your activity —MainActivity.
- There are two components of each Android activity: the XML (Extensible Markup Language) design (the beauty) and the Java text (the brains).
- On the activity\_main.xml page, you can see the full widgets palette underneath the various layout options.
- As you can see, there are 20 widgets available for you to use. In this guide, we'll discuss TextViews and Buttons, which are probably the most common widgets in Android development.

### ➤ **TEXT FIELDS**

- A text field allows the user to type text into your app. It can be either single line or multi-line. Touching a text field places the cursor and automatically displays the keyboard. In addition to typing, text fields allow for a variety of other activities, such as text selection (cut, copy, paste) and data look-up via auto-completion.
- You can add a text field to your layout with the EditText object. You should usually do so in your XML layout with a <EditText> element.
- Text fields can have different input types, such as number, date, password, or email address.

### ➤ **LAYOUTS**

- A layout defines the visual structure for a user interface, such as the UI for an activity or app widget. Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and there are other attributes which are specific to that layout. Following are common attributes and will be applied to all the layouts.

#### ➤ **BUTTON CONTROL**

- **Button** is a user interface control which is used to perform an action whenever the user click or tap on it. Buttons in android will contains a text or an icon or both and perform an action when user touches it. Different types of buttons available are **ImageButton**, **ToggleButton**, **RadioButton**.

#### ➤ **TOGGLE BUTTONS**

- A toggle button allows the user to change a setting between two states. You can add a basic toggle button to your layout with the Toggle Button object. Android 4.0 (API level 14) introduces another kind of toggle button called a switch that provides a slider control, which you can add with a Switch object. Switch Compat is a version of the Switch widget which runs on devices back to API 7.

#### ➤ **SPINNERS / COMBO BOXES**

- Spinners provide a quick way to select one value from a set. In the default state, a spinner shows its currently selected value. Touching the spinner displays a dropdown menu with all other available values, from which the user can select a new one.

#### ➤ **IMAGES**

```
public abstract class Image
    extends Object implements AutoCloseable
    java.lang.Object
    ↳ android.media.Image
```

- single complete image buffer to use with a media source such as a Media Codec or a CameraDevice. This class allows for efficient direct application access to the pixel data of the Image through one or more Byte Buffers. Each buffer is encapsulated in a Plane that describes the layout of the pixel data in that plane. Due to this direct access, and unlike the Bitmap class, Images are not directly usable as UI resources.

#### ➤ **MENU**

- In android, **Options Menu** is a primary collection of menu items for an activity and it is

useful to implement actions that have a global impact on the app, such as Settings, Search, etc. In case, if we define items for the options menu in both activity or fragment, then those items will be combine and display in UI.

#### ➤ **DIALOG**

- A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed.
- The Dialog class is the base class for dialogs, but you should avoid instantiating Dialog directly. Instead, use one of the following subclasses:
- Alert Dialog : A dialog that can show a title, up to three buttons, a list of selectable items, or a custom layout.
- DatePickerDialog or TimePickerDialog : A dialog with a pre-defined UI that allows the user to select a date or time.

## **Android Activity Lifecycle**

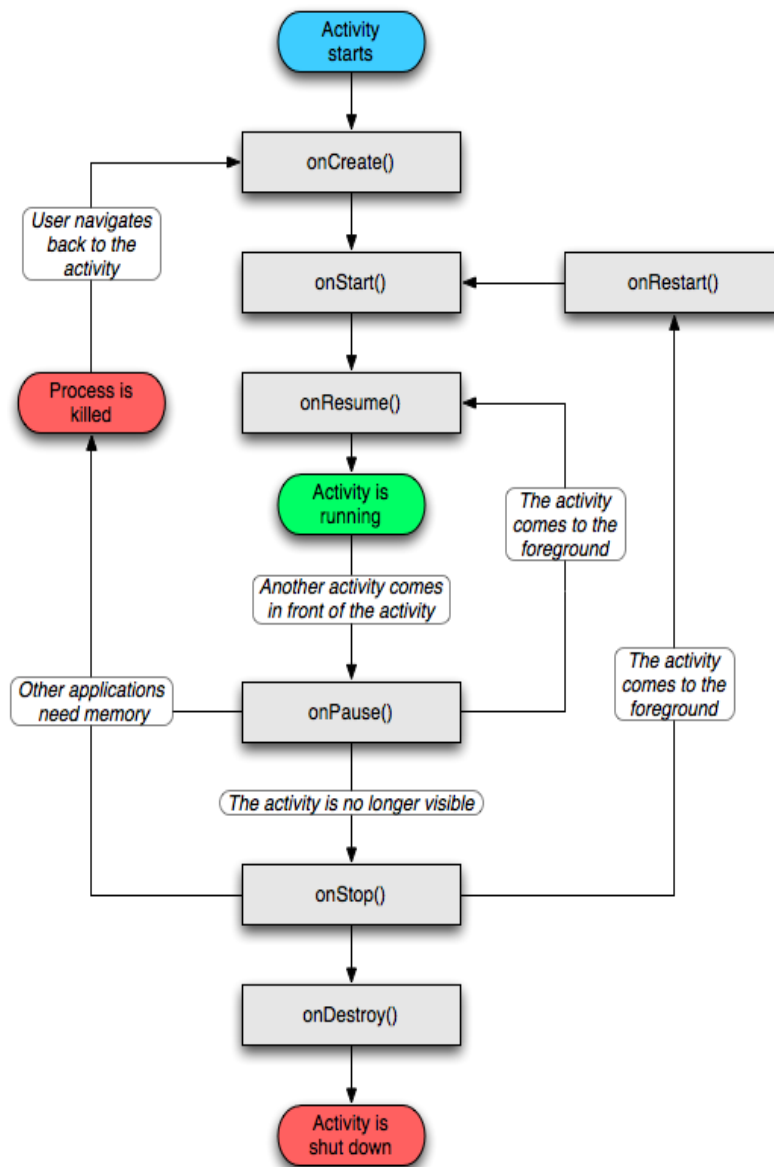
- Android Activity Lifecycle is controlled by 7 methods of android.app.Activity class. The android Activity is the subclass of ContextThemeWrapper class.
- An activity is the single screen in android. It is like window or frame of Java.
- By the help of activity, you can place all your UI components or widgets in a single screen.

## **Android Activity Lifecycle methods**

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

Method	Description
<b>onCreate()</b>	called when activity is first created.
<b>onStart()</b>	called when activity is becoming visible to the user.
<b>onResume()</b>	called when activity will start interacting with the user.
<b>onPause()</b>	called when activity is not visible to the user.
<b>onStop()</b>	called when activity is no longer visible to the user.
<b>onRestart()</b>	called after your activity is stopped, prior to start.

<b>onDestroy()</b>	called before the activity is destroyed.
--------------------	--



**onStart()** callback method will invoke when an activity entered into **Started** State by completing `onCreate()` method. The `onStart()` method will make an activity visible to the user and this method execution will finish very quickly.

an activity entered into **Resumed** state, the system invokes `onResume()` call back method.



**Resumed** state until an another activity happens to take focus away from the app like getting a phone call or screen turned off

After an activity returned from **Paused** state to **Resumed** state, the system again will call `onResume()` method

`onPause()`

Whenever the user leaves an activity or the current activity is being Paused then the system invokes `onPause()` method. The `onPause()` method is used to pause operations like stop playing the music when the activity is in a paused state or pass an activity while switching from one app to another app because every time only one app can be focused.

**onStop()**

The system will invoke `onStop()` callback method when an activity no longer visible to the user, the activity will enter into Stopped state. This happens due to current activity entered into **Resumed** state or newly launched activity covers complete screen or it's been destroyed.

**onRestart()**

The system will invoke `onRestart()` method when an activity restarting itself after stopping it. The `onRestart()` method will restore the state of activity from the time that is being stopped.

Example Program:

```
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d("lifecycle","onCreate invoked");
    }

    @Override
    protected void onStart() {
        super.onStart();
        Log.d("lifecycle","onStart invoked");
    }
}
```

```

@Override
    protected void onResume() {
        super.onResume();
        Log.d("lifecycle","onResume invoked");
    }
@Override
    protected void onPause() {
        super.onPause();
        Log.d("lifecycle","onPause invoked");
    }
@Override
    protected void onStop() {
        super.onStop();
        Log.d("lifecycle","onStop invoked");
    }
@Override
    protected void onRestart() {
        super.onRestart();
        Log.d("lifecycle","onRestart invoked");
    }
@Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d("lifecycle","onDestroy invoked");
    }
}

```

- **Main Activity**

Each application always has a main activity which serves as the starting point for the application “setContentView(R.layout.activity\_main);” which will help to load all UI components defined in **activity\_main.xml** file

- **Log.d()** method is used to generate log messages.
- **super** keyword refers to superclass (parent) objects. It is used to call superclass methods, and to access the superclass constructor.
- **savedInstanceState** is a reference to a Bundle object that is passed into the onCreate method of every **Android** Activity.

- **R** is an Java-class that is auto-generated from your resources by the build process. The **R.layout** member is a auto-generated class that contains all IDs for **layouts**. R.java is a class (with inner classes, like layout or string) generated during the build process with references to your app's resources.
- **Android Toast** can be used to display information for the short period of time. A toast contains message to be displayed quickly and disappears after sometime.

```
Toast.makeText(getApplicationContext(), "I am onCreate() Method",
    Toast.LENGTH_SHORT).show();
```

- **getApplicationContext()** method returns the instance of Context .Context as a state of your Application. It is used to manipulate resources and services used by the application like database, local files, class loaders, shared preferences.

## • Introduction to Intents

- Intents in android are used as message passing mechanism that works both within your application and between applications.
- Three of the core components of an application — activities, services, and broadcast receivers are activated through messages, called intents

### • Intent for an Activity:

Every screen in Android application represents an activity. To start a new activity you need to pass an Intent object to startActivity() method. This Intent object helps to start a new activity and passing data to the second activity.

### • Intent for Services:

Services work in background of an Android application and it does not require any user Interface. Intents could be used to start a Service that performs one-time task (for example: Downloading some file) or for starting a Service you need to pass Intent to startService() method.

### • Intent for Broadcast Receivers:

There are various message that an app receives, these messages are called as Broadcast Receivers. (For example, a broadcast message could be initiated to intimate that the file downloading is completed and ready to use). Android system initiates some broadcast message on several events, such as System Reboot, Low Battery warning message etc.

Intents are really easy to handle and it facilitates communication of components and activities of your application. Moreover, you can communicate to another application and send some data to another application using Intents.

E.g., Intents can be used to start an activity to send email.

### Types of Intents

#### 1. Implicit Intents

#### 2. Explicit Intents

**Explicit Intents** are used to connect the application internally. **Explicit Intent** specifies the component. And intent provides the external class to be invoked. Explicit Intents are used to connect the application internally. In Explicit we use the name of component which will be affected by Intent. For Example: If we know class name then we can navigate the app from One Activity to another activity using Intent. In the similar way we can start a service to download a file in background process. Explicit Intent work internally within an application to perform navigation and data transfer.

Syntax or method of intent calling

```
Intent i= new Intent (FirstActivity.this, SecondActivity.class);  
startActivity(i);
```

If you want pass some information or data to the new Activity you are calling, you can do this by calling putExtra() method before the startActivity() method. This method accepts key-value pair as its parameter.

```
i.putExtra("key1", "I am value1");  
i.putExtra("key2", "I am value2");  
startActivity(i);
```

To receive the data in the new Activity and use it accordingly, you need to call the getIntent() method and then getStringExtra() method in the java class of the Activity you want to open through explicit intent. getStringExtra() method takes the key as the parameter.

### APPLICATION CONTEXT

- Application Context is It is an instance which is the singleton and can be accessed in an activity via **getApplicationContext()**. This context is tied to the lifecycle of an application. The application context can be used where you need a context whose lifecycle is separate from the current context or when you are passing a context beyond the scope of an activity.

➤ **Activity Context**

- This context is available in an activity. This context is tied to the lifecycle of an activity. The activity context should be used when you are passing the context in the scope of an activity or you need the context whose lifecycle is attached to the current context.

➤ **getContext() in ContentProvider**

- This context is the application context and can be used similar to the application context. This can be accessed via **getContext()** method.
-



**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**Mobile Application Development SBS1603**

## UNIT I

User Interface Design: Form widgets, Text Fields, Layouts, Button control, toggle buttons, Spinners (Combo boxes), Images, Menu (Option Menu, Context Menu, Sub menu), Intents, Toast and Dialog-Tabs, Styles (Attributes) and Themes, Adapters – Array Adapters, Base Adapters – Notifications.

### User Interface Design

A typical user interface of an android application consists of action bar and the application content area.

- Main Action Bar
- View Control
- Content Area
- Split Action Bar.

The basic unit of android application is the activity. A UI is defined in an xml file. During compilation, each element in the XML is compiled into equivalent Android GUI class with attributes represented by methods.

### View and ViewGroups

An activity is consist of views. A view is just a widget that appears on the screen. It could be button e.t.c. One or more views can be grouped together into one GroupView. Example of ViewGroup includes layouts.

### Types of layout

There are many types of layout. Some of which are listed below –

- Linear Layout
- Absolute Layout
- Table Layout
- Frame Layout
- Relative Layout

### Linear Layout

Linear layout is further divided into horizontal and vertical layout. It means it can arrange views in a single column or in a single row. Here is the code of linear layout(vertical) that includes a text view.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
```

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello" />
</LinearLayout>
```

## AbsoluteLayout

The AbsoluteLayout enables you to specify the exact location of its children. It can be declared like this.

```
<AbsoluteLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android" >

    <Button
        android:layout_width="188dp"
        android:layout_height="wrap_content"
        android:text="Button"
        android:layout_x="126px"
        android:layout_y="361px" />
</AbsoluteLayout>
```

## TableLayout

The TableLayout groups views into rows and columns. It can be declared like this.

```
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent" >

    <TableRow>
        <TextView
            android:text="User Name:"
            android:width="120dp"
            />

        <EditText
            android:id="@+id/txtUserName"
            android:width="200dp" />
    </TableRow>

</TableLayout>
```



## RelativeLayout

The RelativeLayout enables you to specify how child views are positioned relative to each other. It can be declared like this.

```
<RelativeLayout
    android:id="@+id/RLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android" >
</RelativeLayout>
```

## FrameLayout

The FrameLayout is a placeholder on screen that you can use to display a single view. It can be declared like this.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/lblComments"
    android:layout_below="@+id/lblComments"
    android:layout_centerHorizontal="true" >

    <ImageView
        android:src="@drawable/droid"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</FrameLayout>
```

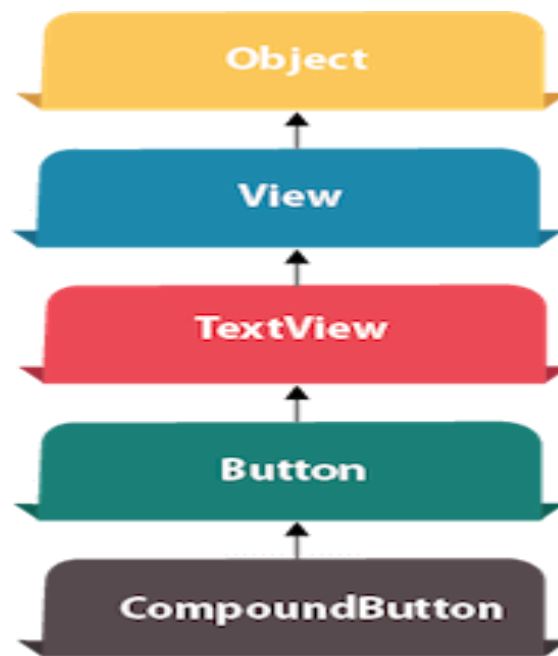
Apart from these attributes, there are other attributes that are common in all views and ViewGroups. They are listed below –

Sr.No	View & description
1	<b>layout_width</b> Specifies the width of the View or ViewGroup
2	<b>layout_height</b> Specifies the height of the View or ViewGroup
3	<b>layout_marginTop</b> Specifies extra space on the top side of the View or ViewGroup
4	<b>layout_marginBottom</b> Specifies extra space on the bottom side of the View or ViewGroup
5	<b>layout_marginLeft</b> Specifies extra space on the left side of the View or ViewGroup
6	<b>layout_marginRight</b> Specifies extra space on the right side of the View or ViewGroup
7	<b>layout_gravity</b> Specifies how child Views are positioned
8	<b>layout_weight</b> Specifies how much of the extra space in the layout should be allocated to the View

**Table 4.1 Attributes of views and View groups**

## Android Widgets

- ✓ There are given a lot of **android widgets** with simplified examples such as Button, EditText, AutoCompleteTextView, ToggleButton, DatePicker, TimePicker, ProgressBar etc.
- ✓ Android widgets are easy to learn. The widely used android widgets with examples are given below:



**Figure 4.1 Android Widgets**

Android Button represents a push-button. The `android.widget.Button` is subclass of `TextView` class and `CompoundButton` is the subclass of `Button` class. There are different types of buttons in android such as `RadioButton`, `ToggleButton`, `CompoundButton` etc.

### Android Button Example with Listener

Here, we are going to create two textfields and one button for sum of two numbers. If user clicks button, sum of two input values is displayed on the Toast. We can perform action on button using different types such as calling listener on button or adding `onClick` property of button in activity's xml file

```
button.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {
```

```

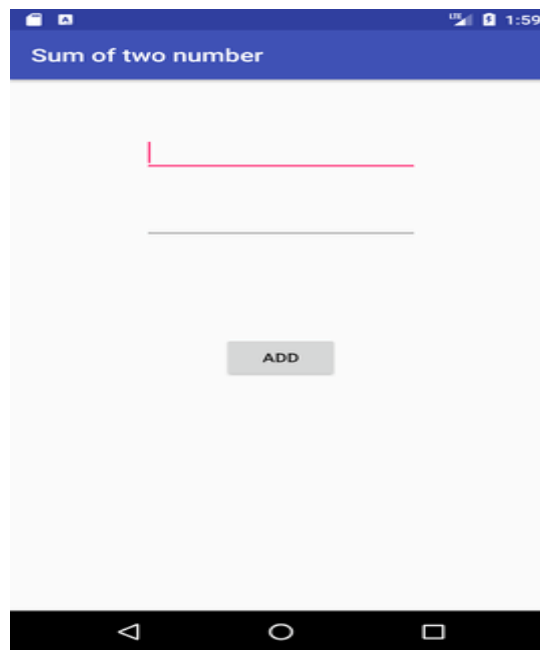
        //code
    }
});

<Button
    android:onClick="methodName"
/>

```

Drag the component or write the code for UI in activity\_main.xml

First of all, drag 2 textfields from the Text Fields palette and one button from the Form Widgets palette as shown in the following figure.



**Figure 4.2 Button Example**

The generated code for the UI components will be like this:

**File: activity\_main.xml**

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"

```

```
android:layout_height="match_parent"  
tools:context="example.javatpoint.com.sumoftwonumber.MainActivity">
```

#### **<EditText**

```
    android:id="@+id/editText1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignParentTop="true"  
    android:layout_centerHorizontal="true"  
    android:layout_marginTop="61dp"  
    android:ems="10"  
    android:inputType="number"  
    tools:layout_editor_absoluteX="84dp"  
    tools:layout_editor_absoluteY="53dp" />
```

#### **<EditText**

```
    android:id="@+id/editText2"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_below="@+id/editText1"  
    android:layout_centerHorizontal="true"  
    android:layout_marginTop="32dp"  
    android:ems="10"  
    android:inputType="number"  
    tools:layout_editor_absoluteX="84dp"  
    tools:layout_editor_absoluteY="127dp" />
```

#### **<Button**

```
    android:id="@+id/button"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_below="@+id/editText2"  
    android:layout_centerHorizontal="true"
```

```
        android:layout_marginTop="109dp"
        android:text="ADD"
        tools:layout_editor_absoluteX="148dp"
        tools:layout_editor_absoluteY="266dp" />
</RelativeLayout>
```

Activity class

Now write the code to display the sum of two numbers.

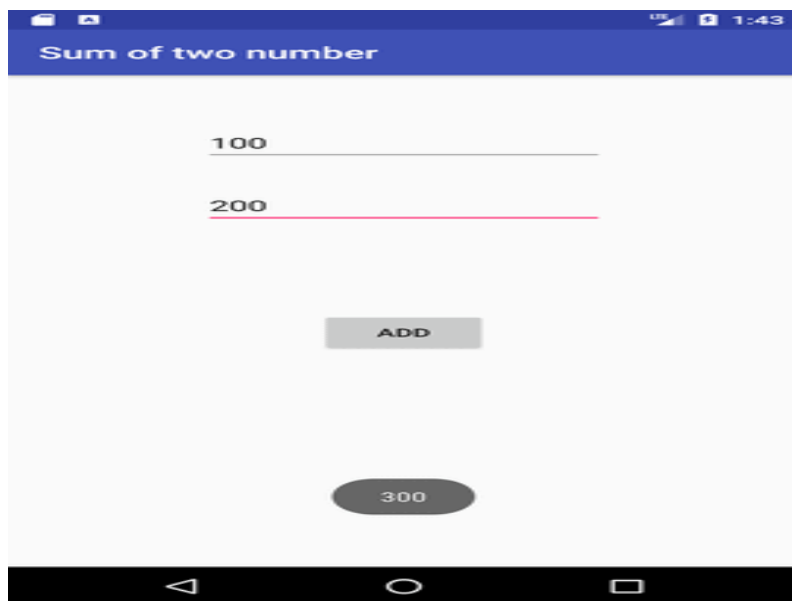
### **File: MainActivity.java**

```
package example.javatpoint.com.sumoftwonumber;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
    private EditText edittext1, edittext2;
    private Button buttonSum;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        addListenerOnButton();
    }
    public void addListenerOnButton() {
        edittext1 = (EditText) findViewById(R.id.editText1);
        edittext2 = (EditText) findViewById(R.id.editText2);
        buttonSum = (Button) findViewById(R.id.button);
        buttonSum.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String value1=edittext1.getText().toString();
```

```
String value2=edittext2.getText().toString();  
int a=Integer.parseInt(value1);  
int b=Integer.parseInt(value2);  
int sum=a+b;  
Toast.makeText(getApplicationContext(),String.valueOf(sum), Toast.LENGTH_LONG  
G).show();  
    }  
    });  
    }  
}
```

---

### Output:



**Example 4.3 Button Example**

## Android Toast

- ✓ Android Toast can be used to display information for the short period of time. A toast contains message to be displayed quickly and disappears after sometime
- ✓ The android.widget.Toast class is the subclass of java.lang.Object class.
- ✓ Toast class is used to show notification for a particular interval of time. After sometime it disappears. It doesn't block the user interaction.

Constant	Description
public static final int LENGTH_LONG	displays view for the long duration of time.
public static final int LENGTH_SHORT	displays view for the short duration of time.

**Table 4.2. Toast Constant**

Method	Description
public static Toast makeText(Context context, CharSequence text, int duration)	Makes the toast containing text and duration.
public void show()	Displays toast.
public void setMargin (float horizontalMargin, float verticalMargin)	Changes the horizontal and vertical margin difference.

**Table 4.3. Toast Methods**



## Android Toast Example

```
Toast.makeText(getApplicationContext(),"Hello Javatpoint",Toast.LENGTH_SHORT).show();
```

Another code:

```
Toast toast=Toast.makeText(getApplicationContext(),"Hello Javatpoint",Toast.LENGTH_SHORT);  
toast.setMargin(50,50);  
toast.show();
```

Here, `getApplicationContext()` method returns the instance of Context.

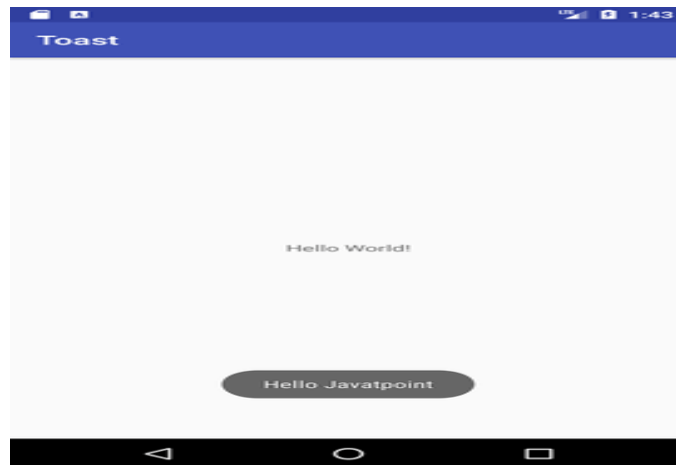
## Full code of activity class displaying Toast

File: MainActivity.java

```
package example.javatpoint.com.toast;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.widget.Toast;  
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        //Displaying Toast with Hello Javatpoint message  
        Toast.makeText(getApplicationContext(),"Hello Javatpoint",Toast.LENGTH_SHORT).show();  
    }  
}
```

---

*Output:*



**Figure 4.4 Toast output**

### **Android ToggleButton**

- ✓ **Android Toggle Button** can be used to display checked/unchecked (On/Off) state on the button.
- ✓ It is beneficial if user have to change the setting between two states. It can be used to On/Off Sound, Wifi, Bluetooth etc.
- ✓ Since Android 4.0, there is another type of toggle button called *switch* that provides slider control. Android ToggleButton and Switch both are the subclasses of CompoundButton class.
- ✓ ToggleButton class provides the facility of creating the toggle button.
- ✓ XML Attributes of ToggleButton class

The 3 XML attributes of ToggleButton class.

XML Attribute	Description
android:disabledAlpha	The alpha to apply to the indicator when disabled.
android:textOff	The text for the button when it is not checked.
android:textOn	The text for the button when it is checked.

**Table 4.4 Toggle Button Attribute**

## Methods of ToggleButton class

The widely used methods of ToggleButton class are given below.

Method	Description
CharSequence getTextOff()	Returns the text when button is not in the checked state.
CharSequence getTextOn()	Returns the text for when button is in the checked state.
void setChecked(boolean checked)	Changes the checked state of this button.

**Table 4.5 Methods of ToggleButton class**

## Android ToggleButton Example

activity\_main.xml

Drag two toggle button and one button for the layout. Now the activity\_main.xml file will look like this:

File: activity\_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/a
pk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.javatpoint.com.togglebutton.MainActivity">
```

### **<ToggleButton**

```
    android:id="@+id/toggleButton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_marginLeft="8dp"  
    android:layout_marginTop="80dp"  
    android:text="ToggleButton"  
    android:textOff="Off"  
    android:textOn="On"  
    app:layout_constraintEnd_toStartOf="@+id/toggleButton2"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```

### **<ToggleButton**

```
    android:id="@+id/toggleButton2"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_marginRight="60dp"  
    android:layout_marginTop="80dp"  
    android:text="ToggleButton"  
    android:textOff="Off"  
    android:textOn="On"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```



**Figure 4.4 Toggle Button Output**

### **Android Spinner**

- ✓ **Android Spinner** is like the combobox of AWT or Swing. It can be used to display the multiple options to the user in which only one item can be selected by the user.
- ✓ Android spinner is like the drop down menu with multiple values from which the end user can select only one value.
- ✓ Android spinner is associated with AdapterView. So you need to use one of the adapter classes with spinner.
- ✓ Android Spinner class is the subclass of AsbSpinner class.

Android Spinner Example:

In this example, we are going to display the country list. You need to use **ArrayAdapter** class to store the country list.

activity\_main.xml

Drag the Spinner from the palette, now the activity\_main.xml file will like this:

File: activity\_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/a  
pk/res/android"
```

```
    xmlns:app="http://schemas.android.com/apk/res-auto"
```

```
    xmlns:tools="http://schemas.android.com/tools"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
```

```
    tools:context="example.javatpoint.com.spinner.MainActivity">
```

```
    <Spinner
```

```
        android:id="@+id/spinner"
```

```
        android:layout_width="149dp"
```

```
        android:layout_height="40dp"
```

```
        android:layout_marginBottom="8dp"
```

```
        android:layout_marginEnd="8dp"
```

```
        android:layout_marginStart="8dp"
```

```
        android:layout_marginTop="8dp"
```

```
        app:layout_constraintBottom_toBottomOf="parent"
```

```
        app:layout_constraintEnd_toEndOf="parent"
```

```
        app:layout_constraintHorizontal_bias="0.502"
```

```
        app:layout_constraintStart_toStartOf="parent"
```

```
        app:layout_constraintTop_toTopOf="parent"
```

```
        app:layout_constraintVertical_bias="0.498" />
```

```
</android.support.constraint.ConstraintLayout>
```

Activity class

**File: MainActivity.java**

```
package example.javatpoint.com.spinner;
```

```
import android.support.v7.app.AppCompatActivity;
```

```
import android.os.Bundle;
```

```
import android.view.View;
```

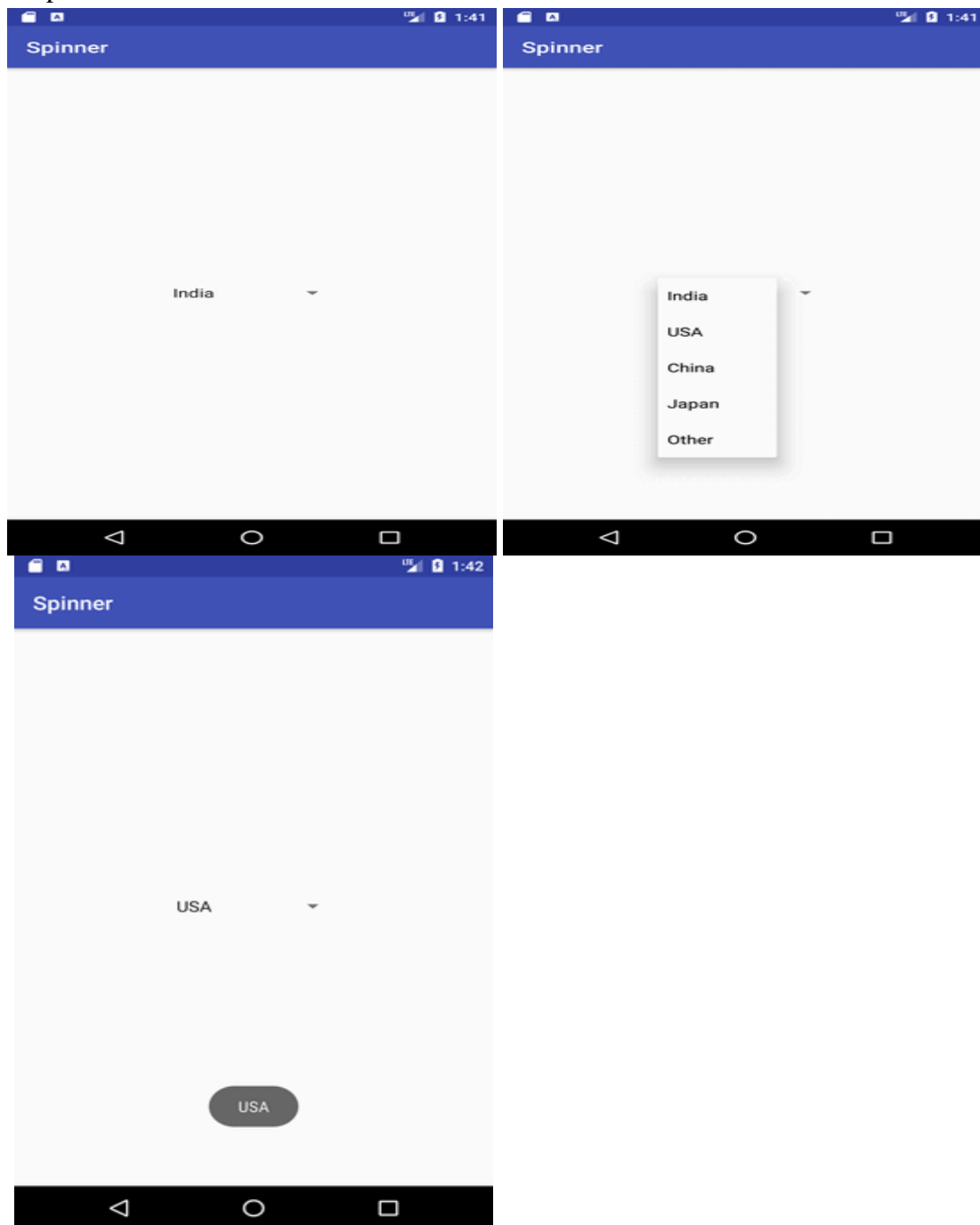
```
import android.widget.AdapterView;
```

```
import android.widget.ArrayAdapter;
```

```
import android.widget.Spinner;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity implements
    AdapterView.OnItemSelectedListener {
    String[] country = { "India", "USA", "China", "Japan", "Other" };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //Getting the instance of Spinner and applying OnItemSelectedListener on it
        Spinner spin = (Spinner) findViewById(R.id.spinner);
        spin.setOnItemSelectedListener(this);
        //Creating the ArrayAdapter instance having the country list
        ArrayAdapter aa = new ArrayAdapter(this,android.R.layout.simple_spinner_item,country);
        aa.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        //Setting the ArrayAdapter data on the Spinner
        spin.setAdapter(aa);
    }
    //Performing action onItemSelected and onNothing selected
    @Override
    public void onItemSelected(AdapterView<?> arg0, View arg1, int position, long id) {
        Toast.makeText(getApplicationContext(),country[position] , Toast.LENGTH_LONG).show
    };
    }
    @Override
    public void onNothingSelected(AdapterView<?> arg0) {
        // TODO Auto-generated method stub
    }
}
```

---

Output:



**Figure 4.5 Android Spinner**



## Android Option Menu

- ✓ **Android Option Menus** are the primary menus of android. They can be used for settings, search, delete item etc.
- ✓ To inflating the menu by calling the **inflate()** method of **MenuInflater** class. To perform event handling on menu items, you need to override **onOptionsItemSelected()** method of Activity class.

### Android Option Menu Example

File: activity\_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout xmlns:android="http://schemas.android.com
/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.javatpoint.com.optionmenu.MainActivity">
    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay">
        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/AppTheme.PopupOverlay" />
        </android.support.design.widget.AppBarLayout>
        <include layout="@layout/content_main" />
    </android.support.design.widget.CoordinatorLayout>
```

File: context\_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/a
pk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context="example.javatpoint.com.optionmenu.MainActivity"
    tools:showIn="@layout/activity_main">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

File: menu\_main.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="example.javatpoint.com.optionmenu.MainActivity">

    <item android:id="@+id/item1"
        android:title="Item 1"/>
    <item android:id="@+id/item2"
        android:title="Item 2"/>
    <item android:id="@+id/item3"
```

```
        android:title="Item 3"
        app:showAsAction="withText"/>
</menu>
```

Activity class

This class displays the content of menu.xml file and performs event handling on clicking the menu items.

File: MainActivity.java

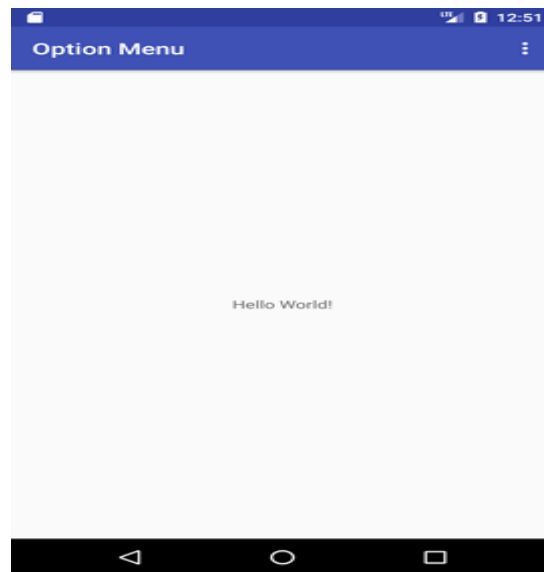
```
package example.javatpoint.com.optionmenu;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();
```

```
switch (id){  
    case R.id.item1:  
        Toast.makeText(getApplicationContext(),"Item 1 Selected",Toast.LENGTH_LONG).show();  
        return true;  
    case R.id.item2:  
        Toast.makeText(getApplicationContext(),"Item 2 Selected",Toast.LENGTH_LONG).show();  
  
        return true;  
    case R.id.item3:  
        Toast.makeText(getApplicationContext(),"Item 3 Selected",Toast.LENGTH_long());  
        return true;  
    default:  
        return super.onOptionsItemSelected(item);  
}  
}  
}
```

---



**Figure 4.5** Output after clicking on the menu button.

---

## Android Context Menu

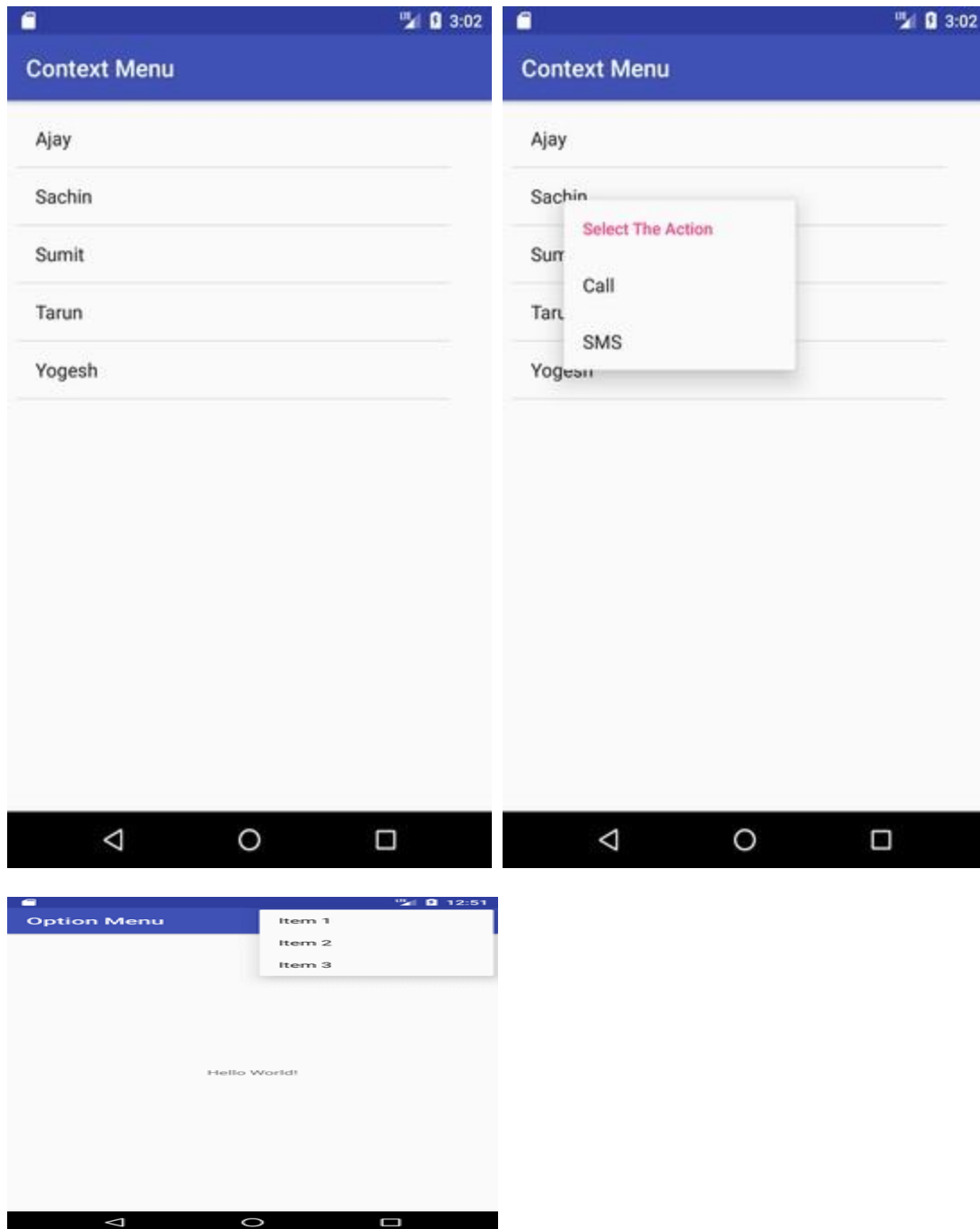
- ✓ Android context menu appears when user press long click on the element. It is also known as floating menu.
- ✓ It affects the selected content while doing action on it.
- ✓ It doesn't support item shortcuts and icons.

### activity\_main.xml

*File: activity\_main.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/a
pk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.javatpoint.com.contextmenu.MainActivity">

    <ListView
        android:layout_width="368dp"
        android:layout_height="495dp"
        android:id="@+id/listView"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```



**Figure 4.6 Android Context Menu**

## Android Intent

- ✓ **Android Intent** is the message that is passed between components such as activities, content providers, broadcast receivers, services etc.
- ✓ It is generally used with startActivity() method to invoke activity, broadcast receivers etc.
- ✓ The **dictionary meaning** of intent is intention or purpose. So, it can be described as the intention to do action.
- ✓ The LabeledIntent is the subclass of android.content.Intent class.

Android intents are mainly used to:

- Start the service
- Launch an activity
- Display a web page
- Display a list of contacts
- Broadcast a message
- Dial a phone call etc.

Types of Android Intents

There are two types of intents in android:

1. Implicit Intent
2. Explicit Intent

1) Implicit Intent

**Implicit Intent** doesn't specify the component. In such case, intent provides information of available components provided by the system that is to be invoked.

For example,

```
Intent intent=new Intent(Intent.ACTION_VIEW);
```

```
intent.setData(Uri.parse("http://www.javatpoint.com"));
startActivity(intent);
```

2) Explicit Intent

**Explicit Intent** specifies the component. In such case, intent provides the external class to be invoked.

```
Intent i = new Intent(getApplicationContext(), ActivityTwo.class);
startActivity(i);
```

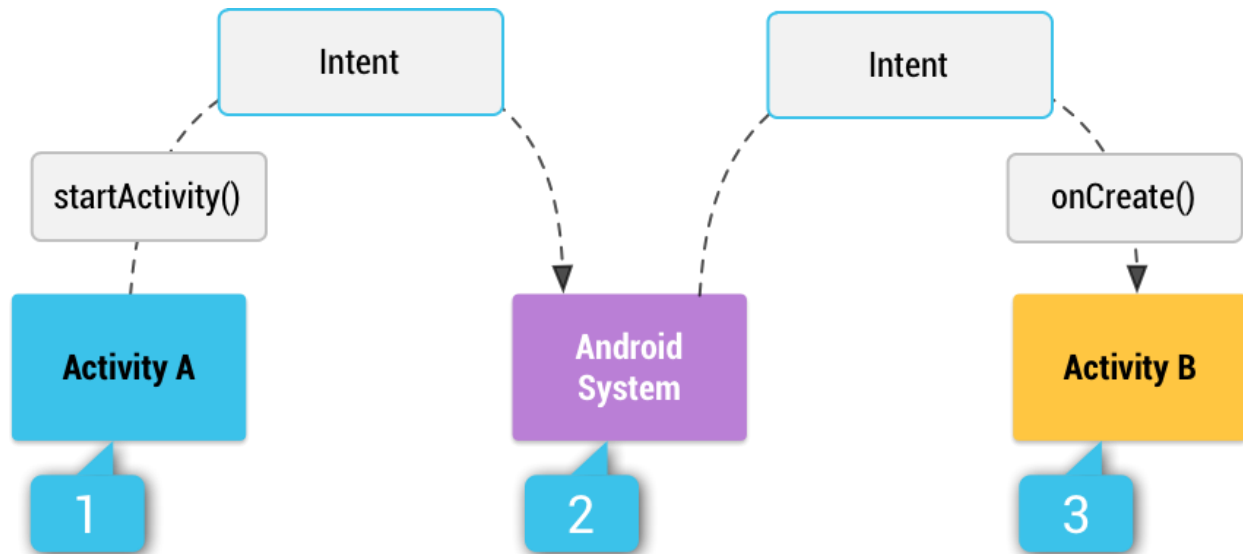


Figure 4.7 Flow of actions by using the Intents

### Use cases of Intents

Intents are used to have communication between various components of the Android Application. These communications can be done in various ways but in general, there are three use cases of the Intents:

#### 1. Starting an Activity

You can use the Intents to start a particular activity by using Intents. For example, if you are having two activities namely **LoginActivity** and **MainActivity** then, you can start the **MainActivity** by clicking the login button present on the **LoginActivity**. By using the **startActivity()**, you can start the desired Activity using the Intents.

#### 2. Starting a Service

**service** as a component that will perform a particular task in the background. So, you can use Intents to start a service also. For API level 21 or higher, you can use Job Scheduler to start a service. For API level lower than 21, you can use the Service class to achieve the same.



### 3. Delivering a broadcast

A broadcast is a message that is received by the application from the system. A very common example of a broadcast can be **Device Charging** message. So, you can use a broadcast to send some kind of message to the applications present in the device. This is done by passing an Intent to `sendBroadcast()` or to `sendOrderedBroadcast()`.

Some of the basic information that an Intent contains are:

1. **Action:** An action is a string that specifies the action to be performed by a particular Activity. For example, you can use the **ACTION\_VIEW** with `startActivity()` when your application contains some information like images that can be shown to the user. Another action that can be performed is **ACTION\_SEND**, which is used to share some data with another application like in Email applications.
2. **Data:** While creating an Intent, you can pass the data and the type of data on which the action is to be performed by the Android System with the help of Intents. The URI object is used to reference the data that will be used to perform some action on it. For example, if you want to edit some data then you have to pass the URI of the data in your **ACTION\_EDIT** action.
3. **Category:** Category is used in case of Explicit Intents where you need to specify the type of application that will be used to perform a particular action. For example, if you want to send some data then only data sending applications should be made available for choice to the users. You can specify a category with the help of `addCategory()`. Any number of categories can be added to the Intent.
4. **Component Name:** The component name is the name of the component that is to be started. You can set the component name by using `setComponent()` or `setClass()` or with the Intent Constructor.
5. **Extras:** You can add extra data to an Intent in the form of key-value pairs and this extra information can be passed from one Activity to the other. `putExtra()` is used to add some extra data to the Intents and this method accepts two parameters i.e. the key and its corresponding value.

## Adapter

- ✓ An adapter acts like a bridge between a data source and the user interface. It reads data from various data sources, converts it into View objects and provide it to the linked Adapter view to create UI components.
- ✓ The data source or dataset can be an Array object, a List object etc.
- ✓ You can create your own Adapter class by extending the `BaseAdapter` class, which is the parent class for all other adapter class. Android SDK also provides some ready-to-use adapter classes, such as `ArrayAdapter`, `SimpleAdapter` etc
- ✓ Adapter and Adapter View are so popular, that every time you see any app with a List of items or Grid of items, you can say for sure that it is using Adapter and Adapter View.
- ✓ Generally, when we create any List or Grid of data, we think we can use a loop to iterate over the data and then set the data to create the list or grid.
- ✓ But what if the data is a set of 1 million products. Then using a loop will not only consume a lot of time, making the app slow, also it might end up eating all the runtime memory.
- ✓ All these problems are solved by using Adapter and Adapter View.

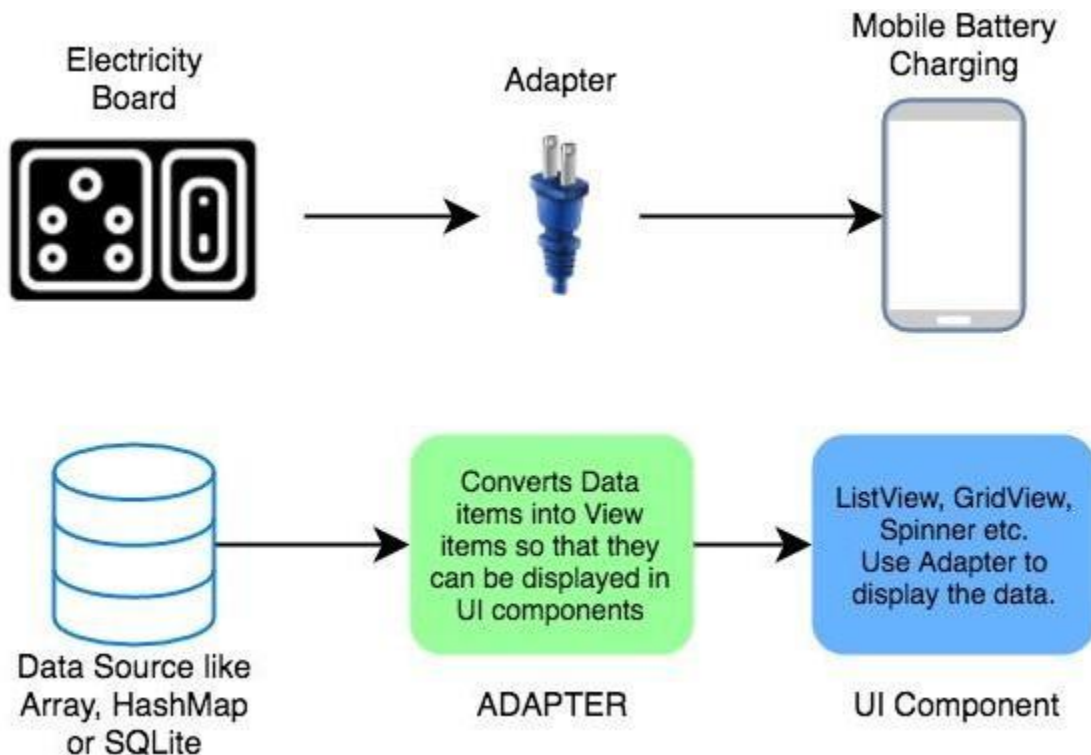


Figure 4.8 Adapter View

- ✓ Adapter View, is a View object, and can be used just like we use any other interface widget. The only catch here is, that it needs an Adapter to provide content to it as it is incapable of displaying data on its own.

## **Adapter View**

- ✓ An Adapter View can be used to display large sets of data efficiently in form of List or Grid etc, provided to it by an Adapter.
- ✓ An Adapter View is capable of displaying millions of items on the User Interface, while keeping the memory and CPU usage very low and without any noticeable lag. Different Adapters follow different strategies for this, but the default Adapter provided in Android SDK follow the following tricks:
  1. It only renders those View objects which are currently on-screen or are about to some on-screen. Hence no matter how big your data set is, the Adapter View will always load only 5 or 6 or maybe 7 items at once, depending upon the display size. Hence saving memory.
  2. It also reuses the already created layout to populate data items as the user scrolls, hence saving the CPU usage.

There are the some commonly used Adapter in Android used to fill the data in the UI components.

1. BaseAdapter – It is parent adapter for all other adapters
2. ArrayAdapter – It is used whenever we have a list of single items which is backed by an array
3. Custom ArrayAdapter – It is used whenever we need to display a custom list
4. SimpleAdapter – It is an easy adapter to map static data to views defined in your XML file
5. Custom SimpleAdapter – It is used whenever we need to display a customized list and needed to access the child items of the list or grid

### **1. BaseAdapter In Android:**

- ✓ BaseAdapter is a common base class of a general implementation of an Adapter that can be used in ListView, GridView, Spinner etc.
- ✓ Whenever we need a customized list in a ListView or customized grids in a GridView we create our own adapter and extend base adapter in that.
- ✓ Base Adapter can be extended to create a custom Adapter for displaying a custom list item.

ArrayAdapter is also an implementation of BaseAdapter.

```
public class CustomAdapter extends BaseAdapter {

    @Override
    public int getCount() {
        return 0;
    }

    @Override
    public Object getItem(int i) {
        return null;
    }

    @Override
    public long getItemId(int i) {
        return 0;
    }

    @Override
    public View getView(int i, View view, ViewGroup viewGroup) {

        return null;
    }
}
```

## ArrayAdapter:

ArrayAdapter is more simple and commonly used Adapter in android.

- Whenever you have a list of single type of items which is backed by an array, you can use ArrayAdapter. For instance, list of phone contacts, countries or names.
- By default, ArrayAdapter expects a Layout with a single TextView, If you want to use more complex views means more customization in grid items or list items, please avoid ArrayAdapter and use custom adapters.

ArrayAdapter is an implementation of BaseAdapter so if we need to create a custom list view or a grid view then we have to create our own custom adapter and extend ArrayAdapter in that custom class. By doing this we can override all the function's of BaseAdapter in our custom adapter.

### Here is code of ArrayAdapter in Android:

```
ArrayAdapter(Context context, int resource, int textViewResourceId, T[] objects)
```

In the above code snippet is the implementation of a ArrayAdapter. Below is the description of all the parameters used for the implementation of a ArrayAdapter to show a list of elements in a list view or in a spinner.

### Parameters used in ArrayAdapter:

Lets discuss parameter in ArrayAdapter class:

- **context:**

The first parameter is used to pass the context means the reference of current class. Here this is a keyword used to show the current class reference. We can also use `getApplicationContext()`, `getActivity()` in the place of this keyword. `getApplicationContext()` is used in a Activity and `getActivity()` is used in a Fragment.

Below is the example code in which we set the current class reference in a adapter.

```
ArrayAdapter arrayAdapter = new ArrayAdapter(this, int resource, int textViewResourceId, T[] objects);
```

- **resource:**

The second parameter is resource id used to set the layout(xml file) for list items in which you have a text view.

```
ArrayAdapter arrayAdapter = new ArrayAdapter(this, R.layout.list_view_items, int textViewResourceId, T[] objects);
```

- **textViewResourceId:**

The third parameter is textViewResourceId which is used to set the id of TextView where you want to display the actual text.

Below is the example code in which we set the id(identity) of a text view.

```
ArrayAdapter arrayAdapter = new ArrayAdapter(this, R.layout.list_view_items, R.id.textView, T[] objects);
```

- **objects:**

- The fourth parameter is an array of objects, used to set the array of elements in the textView. We can set the object of array or array list here.
- Below is the example code in which we set the Animal array in adapter to display the Animal name's list.

```
String animalList[] = {"Lion","Tiger","Monkey","Elephant","Dog","Cat","Camel"};  
ArrayAdapter arrayAdapter = new ArrayAdapter(this, R.layout.list_view_items, R.id.textView, animalList);
```

## Notifications

Android Notification provides short, timely information about the action happened in the application, even it is not running. The notification displays the icon, title and some amount of the content text.

### Set Android Notification Properties

The properties of Android notification are set using **NotificationCompat.Builder** object. Some of the notification properties are mention below:

- **setSmallIcon():** It sets the icon of notification.
- **setContentTitle():** It is used to set the title of notification.
- **setContentText():** It is used to set the text message.
- **setAutoCancel():** It sets the cancelable property of notification.
- **setPriority():** It sets the priority of notification.

Android **Toast** class provides a handy way to show users alerts but problem is that these alerts are not persistent which means alert flashes on the screen for a few seconds and then disappears.

Android notification message fills up the void in such situations. Android notification is a message that we can display to the user outside of our application's normal UI. Notifications in android are built using **NotificationCompat** library.

### Creating Android Notification

```
NotificationManager notificationManager = (NotificationManager)
```

```
getSystemService(NOTIFICATION_SERVICE);
```

The **Notification.Builder** provides an builder interface to create an Notification object as shown below:

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this);
```

### Android Notification Methods.

- ✓ **Notification build()** : Combines all of the options that have been set and returns a new Notification object
- ✓ **NotificationCompat.Builder setAutoCancel (boolean autoCancel)** : Setting this flag will make it such that the notification is automatically canceled when the user clicks it in the panel
- ✓ **NotificationCompat.Builder setContent (RemoteViews views)** : Supplies a custom RemoteViews to use instead of the standard one
- ✓ **NotificationCompat.Builder setContentInfo (CharSequence info)** : Sets the large text at the right-hand side of the notification
- ✓ **NotificationCompat.Builder setContentIntent (PendingIntent intent)** : Supplies a PendingIntent to send when the notification is clicked
- ✓ **NotificationCompat.Builder setContentText (CharSequence text)** : Sets the text (second row) of the notification, in a standard notification
- ✓ **NotificationCompat.Builder setContentTitle (CharSequence title)** : Sets the text (first row) of the notification, in a standard notification
- ✓ **NotificationCompat.Builder setDefaults (int defaults)** : Sets the default notification options that will be used. An example is;
- ✓ **mBuilder.setDefaults(Notification.DEFAULT\_LIGHTS | Notification.DEFAULT\_SOUND)**
- ✓ **NotificationCompat.Builder setLargeIcon (Bitmap icon)** : Sets the large icon that is shown in the ticker and notification
- ✓ **NotificationCompat.Builder setNumber (int number)** : Sets the large number at the right-hand side of the notification

- ✓ **NotificationCompat.Builder setOngoing (boolean ongoing)** : Sets whether this is an ongoing notification
- ✓ **NotificationCompat.Builder setSmallIcon (int icon)** : Sets the small icon to use in the notification layouts
- ✓ **NotificationCompat.Builder setStyle (NotificationCompat.Style style)** : Adds a rich notification style to be applied at build time
- ✓ **NotificationCompat.Builder setTicker (CharSequence tickerText)** : Sets the text that is displayed in the status bar when the notification first arrives
- ✓ **NotificationCompat.Builder setVibrate (long[] pattern)** : Sets the vibration pattern to use
- ✓ **NotificationCompat.Builder setWhen (long when)** : Sets the time that the event occurred. Notifications in the panel are sorted by this time

## Create and Send Notifications

### ✓ Step 1 - Create Notification Builder

As a first step is to create a notification builder using `NotificationCompat.Builder.build()`. You will use Notification Builder to set various Notification properties like its small and large icons, title, priority etc.

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)
```

### ✓ Step 2 - Setting Notification Properties

Once you have **Builder** object, you can set its Notification properties using Builder object as per your requirement. But this is mandatory to set at least following –

- A small icon, set by **setSmallIcon()**
- A title, set by **setContentTitle()**
- Detail text, set by **setContentText()**

```
mBuilder.setSmallIcon(R.drawable.notification_icon);
mBuilder.setContentTitle("Notification Alert, Click Me!");
mBuilder.setContentText("Hi, This is Android Notification Detail!");
```

You have plenty of optional properties which you can set for your notification. To learn more about them, see the reference documentation for `NotificationCompat.Builder`.

### ✓ Step 3 - Attach Actions

This is an optional part and required if you want to attach an action with the notification. An action allows users to go directly from the notification to an **Activity** in your application, where they can look at one or more events or do further work.

The action is defined by a **PendingIntent** containing an **Intent** that starts an Activity in your application. To associate the `PendingIntent` with a gesture, call the appropriate method



of *NotificationCompat.Builder*. For example, if you want to start Activity when the user clicks the notification text in the notification drawer, you add the `PendingIntent` by calling **setContentIntent()**.

A `PendingIntent` object helps you to perform an action on your applications behalf, often at a later time, without caring of whether or not your application is running.

```
Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
stackBuilder.addParentStack(ResultActivity.class);
```

```
// Adds the Intent that starts the Activity to the top of the stack
stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);
mBuilder.setContentIntent(resultPendingIntent);
```

#### ✓ Step 4 - Issue the notification

Finally, you pass the `Notification` object to the system by calling `NotificationManager.notify()` to send your notification. Make sure you call **NotificationCompat.Builder.build()** method on builder object before notifying it. This method combines all of the options that have been set and return a new **Notification** object.

```
NotificationManager mNotificationManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);

// notificationID allows you to update the notification later on.
mNotificationManager.notify(notificationID, mBuilder.build());
```

## Android - Styles and Themes

### Defining Styles

A style is defined in an XML resource that is separate from the XML that specifies the layout. This XML file resides under **res/values/** directory of your project and will have **<resources>** as the root node which is mandatory for the style file. The name of the XML file is arbitrary, but it must use the .xml extension.

To define multiple styles per file using **<style>** tag but each style will have its name that uniquely identifies the style. Android style attributes are set using **<item>** tag as shown below –

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="CustomFontStyle">
    <item name="android:layout_width">fill_parent</item>
```

```
<item name="android:layout_height">wrap_content</item>
<item name="android:capitalize">characters</item>
<item name="android:typeface">monospace</item>
<item name="android:textSize">12pt</item>
<item name="android:textColor">#00FF00</item>/>
</style>
</resources>
```

## Using Styles

Once your style is defined, you can use it in your XML Layout file using **style** attribute as follows –

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/text_id"
        style="@style/CustomFontStyle"
        android:text="@string/hello_world" />

</LinearLayout>
```

## Style Inheritance

Android supports style Inheritance in very much similar way as cascading style sheet in web design. You can use this to inherit properties from an existing style and then define only the properties that you want to change or add.

To implement a custom theme create or edit MyAndroidApp/res/values/themes.xml and add the following –

```
<resources>
...
<style name="MyCustomTheme" parent="android:style/Theme">
<item name="android:textColorPrimary">#ffff0000</item>
</style>
...
</resources>
```

In your AndroidManifest.xml apply the theme to the activities you want to style –

```
<activity
    android:name="com.myapp.MyActivity"
    ...
    android:theme="@style/MyCustomTheme"
/>
```

## Applying Colors to Theme Attributes

Your color resource can then be applied to some theme attributes, such as the window background and the primary text color, by adding `<item>` elements to your custom theme. These attributes are defined in your `styles.xml` file. For example, to apply the custom color to the window background, add the following two `<item>` elements to your custom theme, defined in `MyAndroidApp/res/values/styles.xml` file –

```
<resources>
...
<style name="MyCustomTheme" ...>
    <item name="android:windowBackground">@color/my_custom_color</item>
    <item name="android:colorBackgroundCacheHint">@color/my_custom_color</item>
</style>
...
</resources>
```

## Using a Custom Nine-Patch With Buttons

A nine-patch drawable is a special kind of image which can be scaled in width and height while maintaining its visual integrity. Nine-patches are the most common way to specify the appearance of Android buttons, though any drawable type can be used.



A Sample of Nine-Patch button

### Steps to create Nine-Patch Buttons

- Save this bitmap as `/res/drawable/my_nine_patch.9.png`
- Define a new style
- Apply the new button style to the `buttonStyle` attribute of your custom theme

## Define a new Style

```
<resources>
...
<style name="MyCustomButton" parent="android:Widget.Button">
  <item name="android:background">@drawable/my_nine_patch</item>
</style>
...
</resources>
```

## Apply the theme

```
<resources>
...
<style name="MyCustomTheme" parent="...">
  ...
  <item name="android:buttonStyle">@style/MyCustomButton</item>
</style>
...
</resources>
```



Figure 4.9 Using a Custom Nine-Patch with Buttons

## Android Themes

- ✓ A theme is nothing but an Android style applied to an entire Activity or application, rather than an individual View.
- ✓ Thus, when a style is applied as a theme, every **View** in the Activity or application will apply each style property that it supports. For example, you can apply the same **CustomFontStyle** style as a theme for an Activity and then all text inside that **Activity** will have green monospace font.
- ✓ To set a theme for all the activities of your application, open the **AndroidManifest.xml** file and edit the **<application>** tag to include the **android:theme** attribute with the style name. For example –

```
<application android:theme="@style/CustomFontStyle">
```

- ✓ But if you want a theme applied to just one Activity in your application, then add the **android:theme** attribute to the **<activity>** tag only. For example –

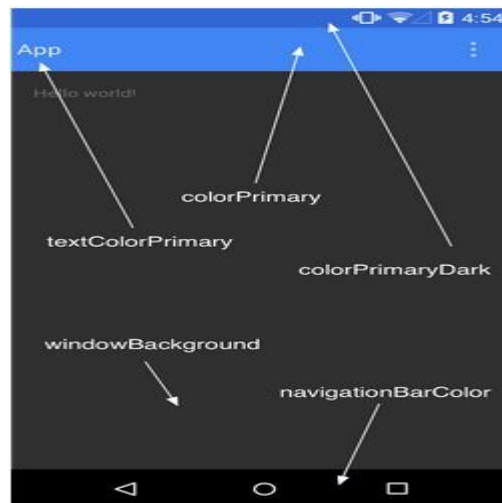
```
<activity android:theme="@style/CustomFontStyle">
```

- ✓ There are number of default themes defined by Android which you can use directly or inherit them using **parent** attribute as follows –

```
<style name="CustomTheme" parent="android:Theme.Light">  
...  
</style>
```

### Styling the colour palette

The layout design can implementable based on them based colours, for example as following design is designed based on them colour(blue)



**Figure4.10** Above layout has designed based on style.xml file, Which has placed at res/values/

```
<resource>
  <style name="AppTheme" parent="android:Theme.Material">
    <item name="android:color/primary">@color/primary</item>
    <item name="android:color/primaryDark">@color/primary_dark</item>
    <item name="android:colorAccent/primary">@color/accent</item>
  </style>
</resource>
```

## Default Styles & Themes

The Android platform provides a large collection of styles and themes that you can use in your applications. You can find a reference of all available styles in the **R.style** class. To use the styles listed here, replace all underscores in the style name with a period



**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

## **Mobile Application Development – SBS1603**

**UNIT V**

Database: Understanding of SQLite database, connecting with the database. Threads – Maps – GPS – MMS – Sensors.

## Understanding of SQLite database:

- ✓ Android SQLite is a very lightweight database which comes with Android OS. Android SQLite combines a clean SQL interface with a very small memory footprint and decent speed. For Android, SQLite is “baked into” the Android runtime, so every Android application can create its own SQLite databases.
- ✓ Android SQLite native API is not JDBC, as JDBC might be too much overhead for a memory-limited smartphone.
- ✓ Once a database is created successfully its located in **data/data//databases/** accessible from Android Device Monitor.
- ✓ SQLite is a typical **relational database**, containing tables (which consists of rows and columns), indexes etc. We can create our own tables to hold the data accordingly. This structure is referred to as a **schema**.

## Database - Package

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

## The Database Structure

- ✓ The first thing needed is the database structure. We create database structure according to the system. But here we are not building an application, and it is only an example demonstrating the use of SQLite Database.

employees		
id	int	PK
name	varchar(200)	
department	varchar(200)	
joiningdate	datetime	
salary	double	

Table 5.1 Database Structure

- ✓ Now we have only a single table, but in real-world scenarios, you will have multiple tables with some complex relationships. Also, remember one thing whenever you create a table create a column named id with int as PRIMARY KEY and AUTOINCREMENT



## Database - Creation

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

```
SQLiteDatabase mydatabase = openOrCreateDatabase("your database name",MODE_PRIVATE,null);
```

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

Sr.No	Method & Description
1	<b><code>openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorHandler)</code></b> This method only opens the existing database with the appropriate flag mode. The common flags mode could be <code>OPEN_READWRITE</code> <code>OPEN_READONLY</code>
2	<b><code>openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)</code></b> It is similar to the above method as it also opens the existing database but it does not define any handler to handle the errors of databases
3	<b><code>openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)</code></b> It not only opens but create the database if it not exists. This method is equivalent to <code>openDatabase</code> method.
4	<b><code>openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)</code></b> This method is similar to above method but it takes the File object as a path rather then a string. It is equivalent to <code>file.getPath()</code>

**Table 5.2. Database Packages**

## Database - Insertion

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

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

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

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

Sr.No	Method & Description
1	<b>execSQL(String sql, Object[] bindArgs)</b> This method not only insert data , but also used to update or modify already existing data in database using bind arguments

### Database - Fetching

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

```
Cursor resultSet = mydatabase.rawQuery("Select * from TutorialPoint",null);
resultSet.moveToFirst();
String username = resultSet.getString(0);
String password = resultSet.getString(1);
```

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

Sr.No	Method & Description
1	<b>getColumnCount()</b> This method return the total number of columns of the table.
2	<b>getColumnIndex(String columnName)</b> This method returns the index number of a column by specifying the name of the column
3	<b>getColumnName(int columnIndex)</b> This method returns the name of the column by specifying the index of the column
4	<b>getColumnNames()</b> This method returns the array of all the column names of the table.

5	<b>getCount()</b> This method returns the total number of rows in the cursor
6	<b>getPosition()</b> This method returns the current position of the cursor in the table
7	<b>isClosed()</b> This method returns true if the cursor is closed and return false otherwise

### Database - Helper class

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

```
public class DBHelper extends SQLiteOpenHelper {
    public DBHelper(){
        super(context,DATABASE_NAME,null,1);
    }
    public void onCreate(SQLiteDatabase db) {}
    public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {}
}
```

### Example

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

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

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

2	Modify src/MainActivity.java file to get references of all the XML components and populate the contacts on listView.
3	Create new src/DBHelper.java that will manage the database work
4	Create a new Activity as DisplayContact.java that will display the contact on the screen
5	Modify the res/layout/activity_main to add respective XML components
6	Modify the res/layout/activity_display_contact.xml to add respective XML components
7	Modify the res/values/string.xml to add necessary string components
8	Modify the res/menu/display_contact.xml to add necessary menu components
9	Create a new menu as res/menu/mainmenu.xml to add the insert contact option
10	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified **MainActivity.java**.

```
package com.example.sairamkrishna.myapplication;

import android.content.Context;
import android.content.Intent;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;

import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;

import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import java.util.ArrayList;
import java.util.List;

public class MainActivity extends ActionBarActivity {
    public final static String EXTRA_MESSAGE = "MESSAGE";
    private ListView obj;
```

```

DBHelper mydb;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mydb = new DBHelper(this);
    ArrayList array_list = mydb.getAllCotacts();
    ArrayAdapter arrayAdapter=new ArrayAdapter(this,android.R.layout.simple_list_item_1,
array_list);

    obj = (ListView)findViewById(R.id.listView1);
    obj.setAdapter(arrayAdapter);
    obj.setOnItemClickListener(new OnItemClickListener(){
        @Override
        public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,long arg3) {
            // TODO Auto-generated method stub
            int id_To_Search = arg2 + 1;

            Bundle dataBundle = new Bundle();
            dataBundle.putInt("id", id_To_Search);

            Intent intent = new Intent(getApplicationContext(),DisplayContact.class);

            intent.putExtras(dataBundle);
            startActivity(intent);
        }
    });
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item){
    super.onOptionsItemSelected(item);

    switch(item.getItemId()) {
        case R.id.item1:Bundle dataBundle = new Bundle();
        dataBundle.putInt("id", 0);

```

```

        Intent intent = new Intent(getApplicationContext(),DisplayContact.class);
        intent.putExtras(dataBundle);

        startActivity(intent);
        return true;
        default:
        return super.onOptionsItemSelected(item);
    }
}

public boolean onKeyDown(int keycode, KeyEvent event) {
    if (keycode == KeyEvent.KEYCODE_BACK) {
        moveTaskToBack(true);
    }
    return super.onKeyDown(keycode, event);
}
}

```

Following is the modified content of display contact activity **DisplayContact.java**

```

package com.example.sairamkrishna.myapplication;

import android.os.Bundle;
import android.app.Activity;
import android.app.AlertDialog;

import android.content.DialogInterface;
import android.content.Intent;
import android.database.Cursor;

import android.view.Menu;
import android.view.MenuItem;
import android.view.View;

import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class DisplayContact extends Activity {
    int from_Where_I_Am_Coming = 0;
    private DBHelper mydb ;

    TextView name ;
    TextView phone;
    TextView email;
    TextView street;
    TextView place;

```

```

int id_To_Update = 0;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_display_contact);
    name = (TextView) findViewById(R.id.editTextName);
    phone = (TextView) findViewById(R.id.editTextPhone);
    email = (TextView) findViewById(R.id.editTextStreet);
    street = (TextView) findViewById(R.id.editTextEmail);
    place = (TextView) findViewById(R.id.editTextCity);

    mydb = new DBHelper(this);

    Bundle extras = getIntent().getExtras();
    if(extras !=null) {
        int Value = extras.getInt("id");

        if(Value>0){
            //means this is the view part not the add contact part.
            Cursor rs = mydb.getData(Value);
            id_To_Update = Value;
            rs.moveToFirst();

            String nam =
rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_NAME));
            String phon =
rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_PHONE));
            String emai =
rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_EMAIL));
            String stree =
rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_STREET));
            String plac =
rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_CITY));

            if (!rs.isClosed()) {
                rs.close();
            }
            Button b = (Button)findViewById(R.id.button1);
            b.setVisibility(View.INVISIBLE);

            name.setText((CharSequence)nam);
            name.setFocusable(false);
            name.setClickable(false);

            phone.setText((CharSequence)phon);

```

```

        phone.setFocusable(false);
        phone.setClickable(false);

        email.setText((CharSequence)emai);
        email.setFocusable(false);
        email.setClickable(false);

        street.setText((CharSequence)stree);
        street.setFocusable(false);
        street.setClickable(false);

        place.setText((CharSequence)plac);
        place.setFocusable(false);
        place.setClickable(false);
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    Bundle extras = getIntent().getExtras();

    if(extras !=null) {
        int Value = extras.getInt("id");
        if(Value>0){
            getMenuInflater().inflate(R.menu.display_contact, menu);
        } else{
            getMenuInflater().inflate(R.menu.menu_main menu);
        }
    }
    return true;
}

public boolean onOptionsItemSelected(MenuItem item) {
    super.onOptionsItemSelected(item);
    switch(item.getItemId()) {
        case R.id.Edit_Contact:
            Button b = (Button)findViewById(R.id.button1);
            b.setVisibility(View.VISIBLE);
            name.setEnabled(true);
            name.setFocusableInTouchMode(true);
            name.setClickable(true);

            phone.setEnabled(true);
            phone.setFocusableInTouchMode(true);

```



```

phone.setClickable(true);

email.setEnabled(true);
email.setFocusableInTouchMode(true);
email.setClickable(true);

street.setEnabled(true);
street.setFocusableInTouchMode(true);
street.setClickable(true);

place.setEnabled(true);
place.setFocusableInTouchMode(true);
place.setClickable(true);

return true;
case R.id.Delete_Contact:

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage(R.string.deleteContact)
    .setPositiveButton(R.string.yes, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            mydb.deleteContact(id_To_Update);
            Toast.makeText(getApplicationContext(), "Deleted Successfully",
                Toast.LENGTH_SHORT).show();
            Intent intent = new Intent(getApplicationContext(),MainActivity.class);
            startActivity(intent);
        }
    })
    .setNegativeButton(R.string.no, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            // User cancelled the dialog
        }
    });

AlertDialog d = builder.create();
d.setTitle("Are you sure");
d.show();

return true;
default:
return super.onOptionsItemSelected(item);

}
}

public void run(View view) {

```

```

Bundle extras = getIntent().getExtras();
if(extras !=null) {
    int Value = extras.getInt("id");
    if(Value>0){
        if(mydb.updateContact(id_To_Update,name.getText().toString(),
            phone.getText().toString(), email.getText().toString(),
                street.getText().toString(), place.getText().toString())){
            Toast.makeText(getApplicationContext(), "Updated",
Toast.LENGTH_SHORT).show();
            Intent intent = new Intent(getApplicationContext(),MainActivity.class);
            startActivity(intent);
        } else{
            Toast.makeText(getApplicationContext(), "not Updated",
Toast.LENGTH_SHORT).show();
        }
    } else{
        if(mydb.insertContact(name.getText().toString(), phone.getText().toString(),
            email.getText().toString(), street.getText().toString(),
                place.getText().toString())){
            Toast.makeText(getApplicationContext(), "done",
                Toast.LENGTH_SHORT).show();
        } else{
            Toast.makeText(getApplicationContext(), "not done",
                Toast.LENGTH_SHORT).show();
        }
        Intent intent = new Intent(getApplicationContext(),MainActivity.class);
        startActivity(intent);
    }
}
}
}
}

```

Following is the content of Database class **DBHelper.java**

```

package com.example.sairamkrishna.myapplication;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Hashtable;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.DatabaseUtils;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase;

public class DBHelper extends SQLiteOpenHelper {

```

```

public static final String DATABASE_NAME = "MyDBName.db";
public static final String CONTACTS_TABLE_NAME = "contacts";
public static final String CONTACTS_COLUMN_ID = "id";
public static final String CONTACTS_COLUMN_NAME = "name";
public static final String CONTACTS_COLUMN_EMAIL = "email";
public static final String CONTACTS_COLUMN_STREET = "street";
public static final String CONTACTS_COLUMN_CITY = "place";
public static final String CONTACTS_COLUMN_PHONE = "phone";
private HashMap hp;

public DBHelper(Context context) {
    super(context, DATABASE_NAME , null, 1);
}

@Override
public void onCreate(SQLiteDatabase db) {
    // TODO Auto-generated method stub
    db.execSQL(
        "create table contacts " +
        "(id integer primary key, name text,phone text,email text, street text,place text)"
    );
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // TODO Auto-generated method stub
    db.execSQL("DROP TABLE IF EXISTS contacts");
    onCreate(db);
}

public boolean insertContact (String name, String phone, String email, String street,String
place) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put("name", name);
    contentValues.put("phone", phone);
    contentValues.put("email", email);
    contentValues.put("street", street);
    contentValues.put("place", place);
    db.insert("contacts", null, contentValues);
    return true;
}

public Cursor getData(int id) {
    SQLiteDatabase db = this.getReadableDatabase();

```

```

        Cursor res = db.rawQuery( "select * from contacts where id="+id+"", null );
        return res;
    }

    public int numberOfRows(){
        SQLiteDatabase db = this.getReadableDatabase();
        int numRows = (int) DatabaseUtils.queryNumEntries(db, CONTACTS_TABLE_NAME);
        return numRows;
    }

    public boolean updateContact (Integer id, String name, String phone, String email, String
street,String place) {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        contentValues.put("name", name);
        contentValues.put("phone", phone);
        contentValues.put("email", email);
        contentValues.put("street", street);
        contentValues.put("place", place);
        db.update("contacts", contentValues, "id = ? ", new String[] { Integer.toString(id) } );
        return true;
    }

    public Integer deleteContact (Integer id) {
        SQLiteDatabase db = this.getWritableDatabase();
        return db.delete("contacts",
            "id = ? ",
            new String[] { Integer.toString(id) });
    }

    public ArrayList<String> getAllCotacts() {
        ArrayList<String> array_list = new ArrayList<String>();

        //hp = new HashMap();
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor res = db.rawQuery( "select * from contacts", null );
        res.moveToFirst();

        while(res.isAfterLast() == false){
            array_list.add(res.getString(res.getColumnIndex(CONTACTS_COLUMN_NAME)));
            res.moveToNext();
        }
        return array_list;
    }
}

```

Following is the content of the **res/layout/activity\_main.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textView"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:textSize="30dp"
        android:text="Data Base" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Tutorials Point"
        android:id="@+id/textView2"
        android:layout_below="@+id/textView"
        android:layout_centerHorizontal="true"
        android:textSize="35dp"
        android:textColor="#ff16ff01" />

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView"
        android:layout_below="@+id/textView2"
        android:layout_centerHorizontal="true"
        android:src="@drawable/logo"/>

    <ScrollView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/scrollView"
        android:layout_below="@+id/imageView"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true">
```

```
<ListView
    android:id="@+id/listView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true" >
</ListView>
```

```
</ScrollView>
```

```
</RelativeLayout>
```

Following is the content of the **res/layout/activity\_display\_contact.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/scrollView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    tools:context=".DisplayContact" >

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="370dp"
        android:paddingBottom="@dimen/activity_vertical_margin"
        android:paddingLeft="@dimen/activity_horizontal_margin"
        android:paddingRight="@dimen/activity_horizontal_margin"
        android:paddingTop="@dimen/activity_vertical_margin">

        <EditText
            android:id="@+id/editTextName"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_marginTop="5dp"
            android:layout_marginLeft="82dp"
            android:ems="10"
            android:inputType="text" >
        </EditText>

        <EditText
            android:id="@+id/editTextEmail"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignLeft="@+id/editTextStreet"
```

```
android:layout_below="@+id/editTextStreet"
android:layout_marginTop="22dp"
android:ems="10"
android:inputType="textEmailAddress" />
```

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/editTextName"
    android:layout_alignParentLeft="true"
    android:text="@string/name"
    android:textAppearance="?android:attr/textAppearanceMedium" />
```

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editTextCity"
    android:layout_alignParentBottom="true"
    android:layout_marginBottom="28dp"
    android:onClick="run"
    android:text="@string/save" />
```

```
<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/editTextEmail"
    android:layout_alignLeft="@+id/textView1"
    android:text="@string/email"
    android:textAppearance="?android:attr/textAppearanceMedium" />
```

```
<TextView
    android:id="@+id/textView5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/editTextPhone"
    android:layout_alignLeft="@+id/textView1"
    android:text="@string/phone"
    android:textAppearance="?android:attr/textAppearanceMedium" />
```

```
<TextView
    android:id="@+id/textView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
android:layout_above="@+id/editTextEmail"
android:layout_alignLeft="@+id/textView5"
android:text="@string/street"
android:textAppearance="?android:attr/textAppearanceMedium" />
```

```
<EditText
    android:id="@+id/editTextCity"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignRight="@+id/editTextName"
    android:layout_below="@+id/editTextEmail"
    android:layout_marginTop="30dp"
    android:ems="10"
    android:inputType="text" />
```

```
<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/editTextCity"
    android:layout_alignBottom="@+id/editTextCity"
    android:layout_alignParentLeft="true"
    android:layout_toLeftOf="@+id/editTextEmail"
    android:text="@string/country"
    android:textAppearance="?android:attr/textAppearanceMedium" />
```

```
<EditText
    android:id="@+id/editTextStreet"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editTextName"
    android:layout_below="@+id/editTextPhone"
    android:ems="10"
    android:inputType="text" >
```

```
<requestFocus />
```

```
</EditText>
```

```
<EditText
    android:id="@+id/editTextPhone"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editTextStreet"
    android:layout_below="@+id/editTextName"
    android:ems="10"
    android:inputType="phone|text" />
```



```
</RelativeLayout>
</ScrollView>
```

Following is the content of the **res/value/string.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Address Book</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="Add_New">Add New</string>
    <string name="edit">Edit Contact</string>
    <string name="delete">Delete Contact</string>
    <string name="title_activity_display_contact">DisplayContact</string>
    <string name="name">Name</string>
    <string name="phone">Phone</string>
    <string name="email">Email</string>
    <string name="street">Street</string>
    <string name="country">City/State/Zip</string>
    <string name="save">Save Contact</string>
    <string name="deleteContact">Are you sure, you want to delete it.</string>
    <string name="yes">Yes</string>
    <string name="no">No</string>
</resources>
```

Following is the content of the **res/menu/main\_menu.xml**

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

    <item android:id="@+id/item1"
        android:icon="@drawable/add"
        android:title="@string/Add_New" >
    </item>

</menu>
```

Following is the content of the **res/menu/display\_contact.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/Edit_Contact"
        android:orderInCategory="100"
        android:title="@string/edit"/>

    <item
```

```
android:id="@+id/Delete_Contact"  
android:orderInCategory="100"  
android:title="@string/delete"/>
```

```
</menu>
```

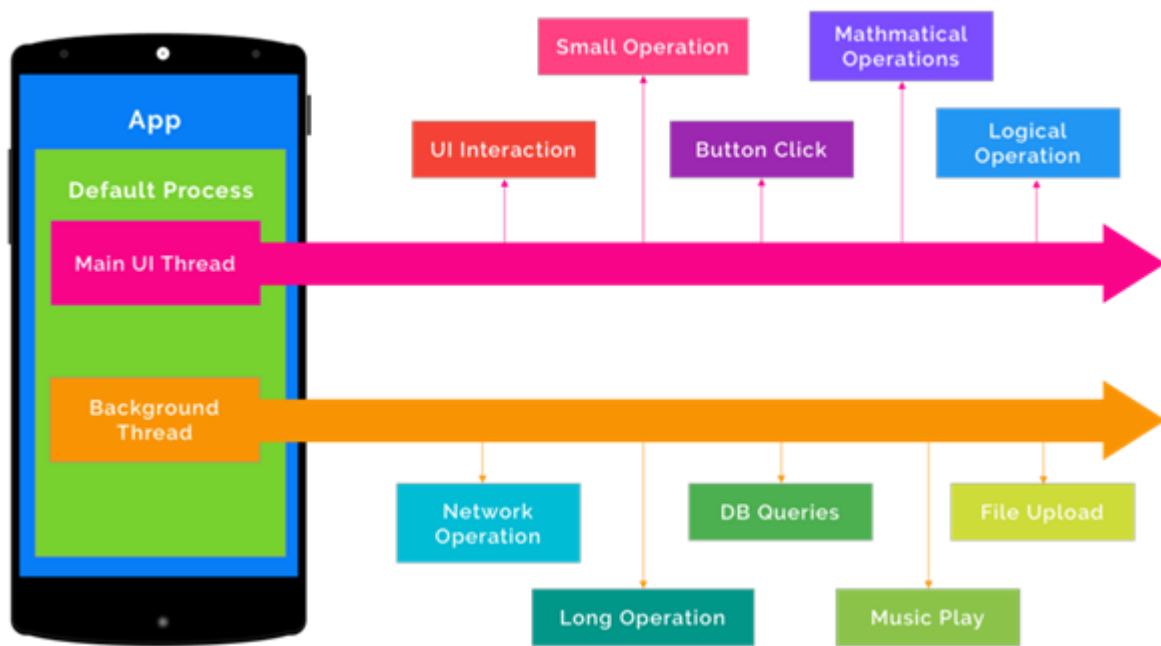
This is the default **AndroidManifest.xml** of this project

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    package="com.example.sairamkrishna.myapplication" >  
  
    <application  
        android:allowBackup="true"  
        android:icon="@mipmap/ic_launcher"  
        android:label="@string/app_name"  
        android:theme="@style/AppTheme" >  
  
        <activity  
            android:name=".MainActivity"  
            android:label="@string/app_name" >  
  
            <intent-filter>  
                <action android:name="android.intent.action.MAIN" />  
                <category android:name="android.intent.category.LAUNCHER" />  
            </intent-filter>  
  
        </activity>  
  
        <activity android:name=".DisplayContact"/>  
  
    </application>  
</manifest>
```

## Threads

- ✓ Threads are the cornerstone of any multitasking operating system and can be thought of as mini-processes running within a main process, the purpose of which is to enable at least the appearance of parallel execution paths within applications.
- ✓ When an Android application is first started, the runtime system creates a single thread in which all application components will run by default. This thread is generally referred to as the main thread. The primary role of the main thread is to handle the user interface in terms of event handling and interaction with views in the user interface. Any additional components that are started within the application will, by default, also run on the main thread.

- ✓ Any component within an application that performs a time consuming task using the main thread will cause the entire application to appear to lock up until the task is completed. This will typically result in the operating system displaying an “Application is unresponsive” warning to the user. Clearly, this is far from the desired behavior for any application. In such a situation, this can be avoided simply by launching the task to be performed in a separate thread, allowing the main thread to continue unhindered with other tasks



**Figure 5.1 Android Thread**

**In Android, you can categorize all threading components into two basic categories:**

1. **Threads that are attached to an activity/fragment:** These threads are tied to the lifecycle of the activity/fragment and are terminated as soon as the activity/fragment is destroyed.
  - A. AsyncTask
  - B. Loaders

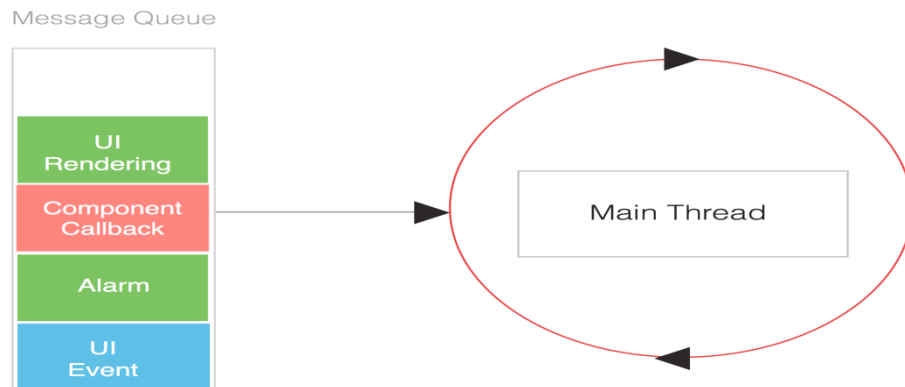
2. **Threads that are not attached to any activity/fragment:** These threads can continue to run beyond the lifetime of the activity/fragment (if any) from which they were spawned.

- A. Service
- B. Intent Service

For the above two threading components, There are five types of thread are using in Android Mobile Development which areas:-

- C. Main thread
- D. UI thread
- E. Worker thread
- F. Any thread
- G. Binder thread

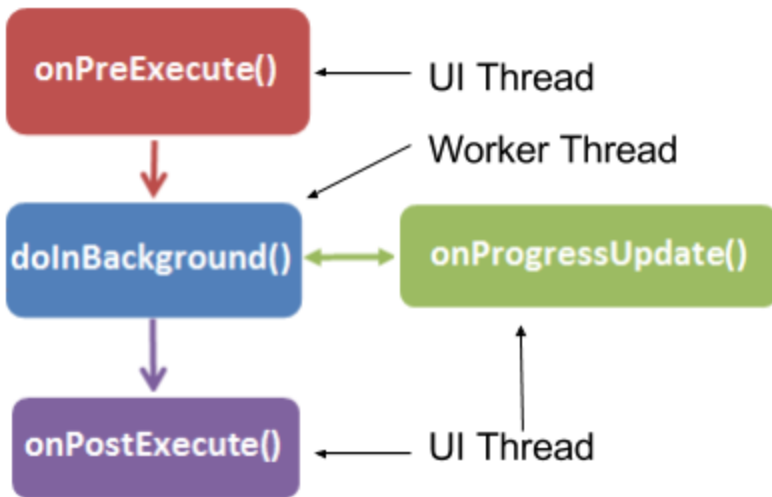
## 1. Main Thread



### **Figure 5.2 Example of Main Thread**

- ✓ When an application is launched in Android, it creates the first thread of execution, known as the “main” thread. The main thread is responsible for dispatching events to the appropriate user interface widgets as well as communicating with components from the Android UI toolkit.
- ✓ To keep your application responsive, it is essential to avoid using the main thread to perform any operation that may end up keeping it blocked.
- ✓ Network operations and database calls, as well as loading of certain components, are common examples of operations that one should avoid in the main thread.
- ✓ When they are called in the main thread, they are called synchronously, which means that the UI will remain completely unresponsive until the operation completes.
- ✓ For this reason, they are usually performed in separate threads, which thereby avoids blocking the UI while they are being performed (i.e., they have performed asynchronously from the UI).

## 2. UI Thread



**Figure 5.3 Example of UI Thread**

- ✓ The `UiThread` is the main thread of execution for your application. This is where most of your application code is run. All of your application components (Activities, Services, ContentProviders, BroadcastReceivers) are created in this thread, and any system calls to those components are performed in this thread.
- ✓ For instance, application is a single Activity class. Then all of the lifecycle methods and most of your event handling code is run in this `UiThread`. These are methods like `onCreate`, `onPause`, `onDestroy`, `onClick`, etc.
- ✓ Additionally, this is where all of the updates to the UI are made. Anything that causes the UI to be updated or changed HAS to happen on the UI thread. When you explicitly spawn a new thread to do work in the background, this code is not run on the `UiThread`.

- ✓ So what happens if this background thread needs to do something that changes the UI?  
This is what the `runOnUiThread` is for.
- ✓ Actually you're supposed to use a Handler. It provides these background threads the ability to execute code that can modify the UI.
- ✓ They do this by putting the UI-modifying code in a Runnable object and passing it to the `runOnUiThread` method.

### 3. Worker Thread:

Worker threads are background threads. They are the threads that are created separately, other than the UI thread. Since blocking the UI thread is restricted according to the rule, the user should run the child processes and tasks in worker threads.

An example of the creation and working of worker thread is given below:-

```
Public void onClick(View v) { new Thread(new Runnable() {  
  
public void run() {  
  
Bitmap b = loadImageFromNetwork("http://example.com/image.png");  
  
mImageView.setImageBitmap(b);}  
  
}).start(); }
```

- ✓ In the above example code, the download operation is handled by a second thread other than the UI thread. But the program violates the second rule.

- ✓ The imageView from UI thread is manipulating from this worker thread. According to the second rule, UI could not be accessed from outside the UI thread. The solution for such a restriction is `runOnUiThread(Runnable)` method.
- ✓ The main or UI thread can be accessed from other threads using `runOnUiThread(Runnable)` method. As a result, the specified runnable action passed through this method will run on the UI thread.
- ✓ The action will execute immediately if the current thread is in the UI itself. Else the action will be posted to the event queue.

### 3. Any Thread:

```
@Target([AnnotationTarget.FUNCTION, AnnotationTarget.PROPERTY_GETTER,  
AnnotationTarget.PROPERTY_SETTER, AnnotationTarget.CONSTRUCTOR,  
AnnotationTarget.CLASS, AnnotationTarget.FILE, AnnotationTarget.VALUE_PARAMETER])  
class AnyThread
```

- ✓ Denotes that the annotated method can be called from any thread (e.g. it is “thread-safe”.)  
If the annotated element is a class, then all methods in the class can be called from any thread.
- ✓ The main purpose of this method is to indicate that you believe a method can be called from any thread; static tools can then check that nothing you call from within this method or class has more strict threading requirements.

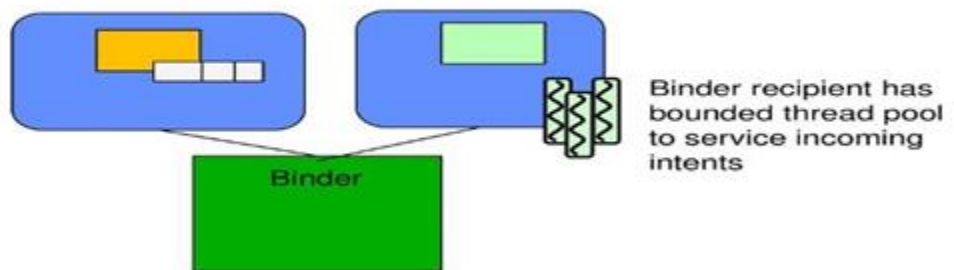


**Example:**

```
<code>
@AnyThread
public void deliverResult(D data) { ... }
</code>
```

**5. Binder thread:**

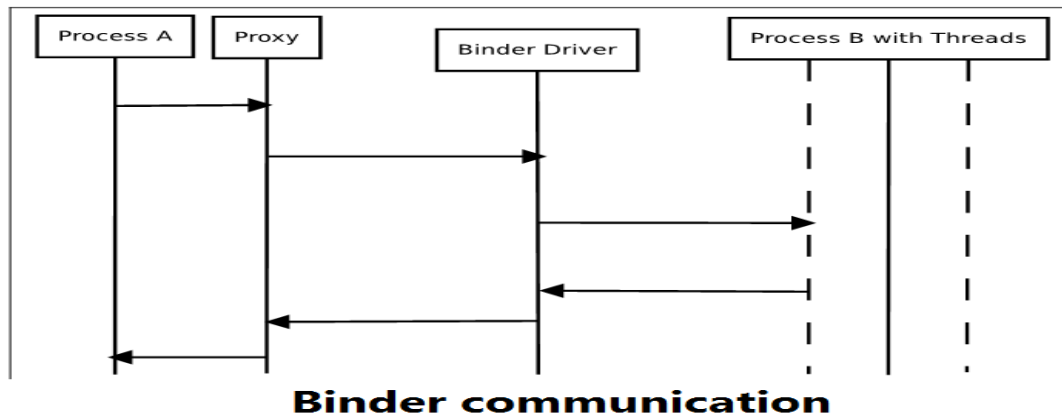
## Binder Thread Scheduling



**Figure 5.4: Binder Thread**

- ✓ The initiator thread effectively transfers to the recipient activity.
- ✓ Max of 15 binder threads per process
- ✓ Max 1 MB of Transaction Buffer
  
- ✓ Binder thread represents a separate thread of your service. Binder is a mechanism that provides Inter-Process Communication.
  
- ✓ Let's consider an example. Imagine that you have service Process B (see picture). And you have several applications that communicate with this service B (one of this application is, for instance, Process A).
  
- ✓ Thus, one service B should provide different results simultaneously to different applications. Thus, you need to run several replicas of Service B for different

applications. Android runs these replicas in different threads of Process B and these threads are called “Binder Thread #N”.



**Figure 5.5 Binder Communication**

## Android Google Map

Android provides facility to integrate Google map in our application. Google map displays your current location, navigate location direction, search location etc. We can also customize Google map according to our requirement.

### Types of Google Maps

There are four different types of Google maps, as well as an optional to no map at all. Each of them gives different view on map. These maps are as follow:

1. **Normal:** This type of map displays typical road map, natural features like river and some features build by humans.
2. **Hybrid:** This type of map displays satellite photograph data with typical road maps. It also displays road and feature labels.
3. **Satellite:** Satellite type displays satellite photograph data, but doesn't display road and feature labels.

4. **Terrain:** This type displays photographic data. This includes colors, contour lines and labels and perspective shading.
5. **None:** This type displays an empty grid with no tiles loaded.

## Syntax of different types of map

1. `googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);`
2. `googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);`
3. `googleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);`
4. `googleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);`

## Methods of Google map

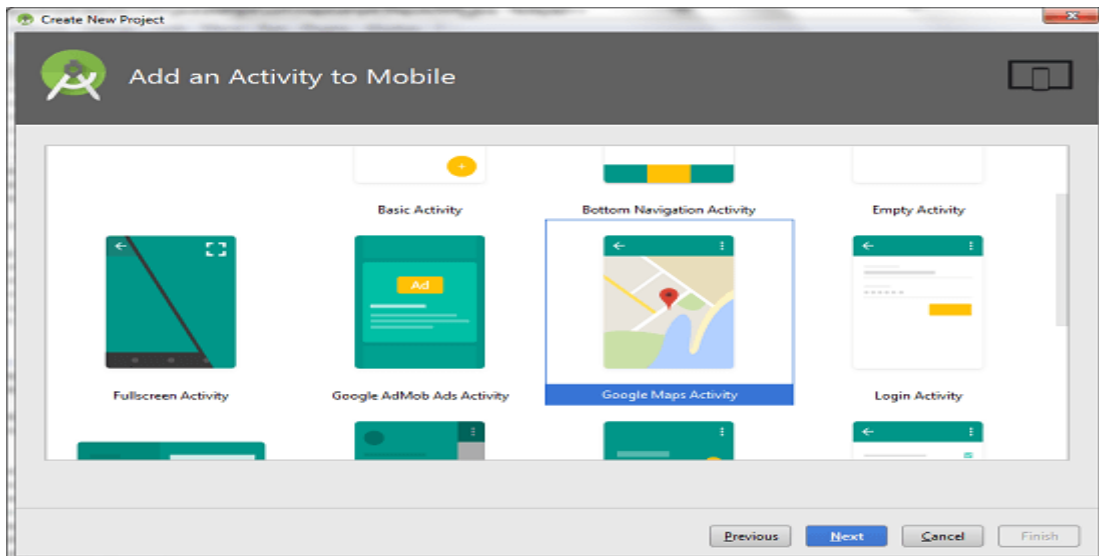
Google map API provides several methods that help to customize Google map. These methods are as following:

Methods	Description
<code>addCircle(CircleOptions options)</code>	This method add circle to map.
<code>addPolygon(PolygonOptions options)</code>	This method add polygon to map.
<code>addTileOverlay(TileOverlayOptions options)</code>	This method add tile overlay to the map.
<code>animateCamera(CameraUpdate update)</code>	This method moves the map according to the update with an animation.
<code>clear()</code>	This method removes everything from the map.
<code>getMyLocation()</code>	This method returns the currently displayed user location.

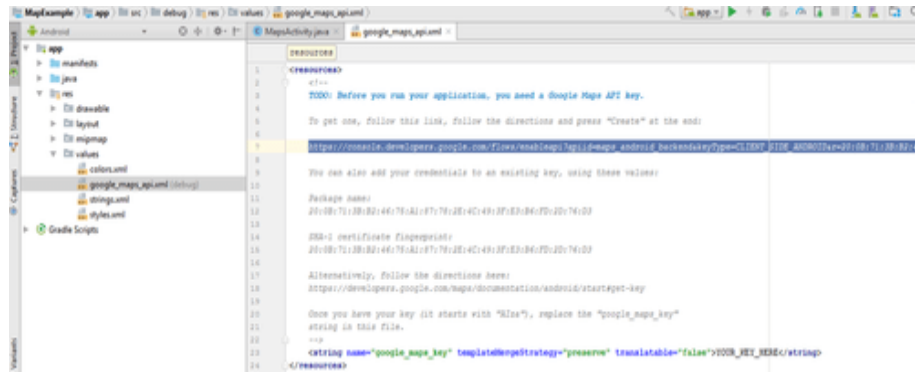
<code>moveCamera(CameraUpdate update)</code>	This method reposition the camera according to the instructions defined in the update.
<code>setTrafficEnabled(boolean enabled)</code>	This method set the traffic layer on or off.
<code>snapshot(GoogleMap.SnapshotReadyCallback callback)</code>	This method takes a snapshot of the map.
<code>stopAnimation()</code>	This method stops the camera animation if there is any progress.

## Example of Google Map

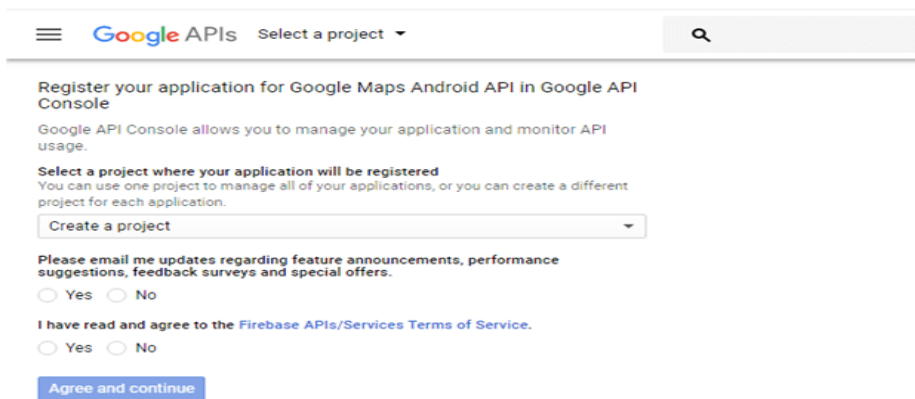
Let's create an example of Google map integrating within our app. For doing this we select Google Maps Activity.



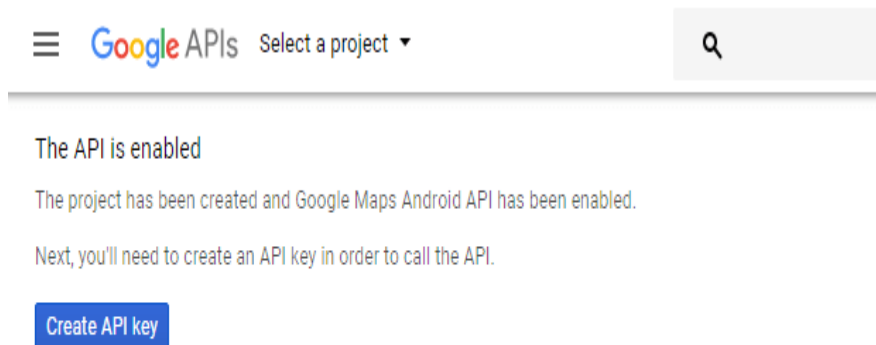
Copy the URL from `google_map_api.xml` file to generate Google map key.



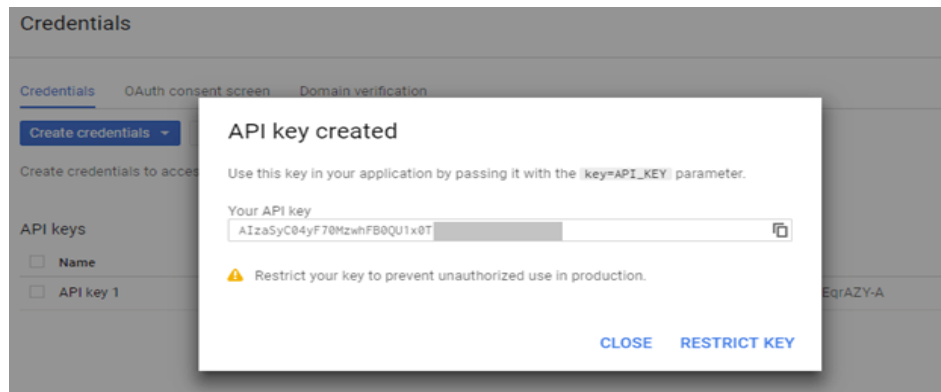
Paste the copied URL at the browser. It will open the following page.



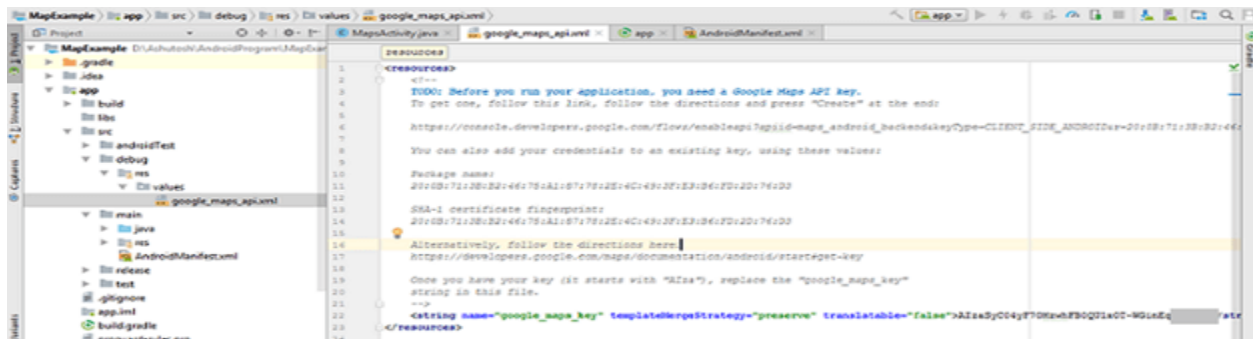
Click on Create API key to generate API key.



After clicking on Create API key, it will generate our API key displaying the following screen.



Copy this generated API key in our *google\_map\_api.xml* file



## activity\_maps.xml

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:map="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.com.mapexample.MapsActivity" />
```

## MapsActivity.java

To get the GoogleMap object in our MapsActivity.java class we need to implement the OnMapReadyCallback interface and override the onMapReady() callback method.

```
package example.com.mapexample;
import android.support.v4.app.FragmentActivity;
import android.os.Bundle;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;
public class MapsActivity extends FragmentActivity implements OnMapReadyCall
back{

    private GoogleMap mMap;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);

        SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragm
entManager()
            .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
    }
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
        LatLng sydney = new LatLng(-34, 151);
        mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydn
ey"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
    }
}
```

## Required Permission

Add the following user-permission in AndroidManifest.xml file.

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

## AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
package="example.com.mapexample">  
  <!--  
    The ACCESS_COARSE/FINE_LOCATION permissions are not required to use  
  
    Google Maps Android API v2, but you must specify either coarse or fine  
    location permissions for the 'MyLocation' functionality.  
  -->  
  <uses-  
permission android:name="android.permission.ACCESS_FINE_LOCATION" />  
  <uses-  
permission android:name="android.permission.ACCESS_COARSE_LOCATION" />  
  <uses-permission android:name="android.permission.INTERNET" />  
  <application  
    android:allowBackup="true"  
    android:icon="@mipmap/ic_launcher"  
    android:label="@string/app_name"  
    android:roundIcon="@mipmap/ic_launcher_round"  
    android:supportsRtl="true"  
    android:theme="@style/AppTheme">  
    <meta-data
```



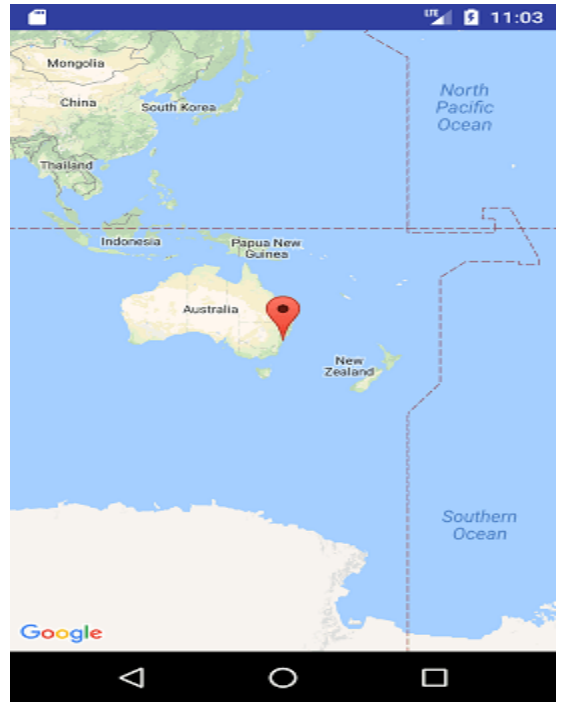
```

        android:name="com.google.android.geo.API
_KEY"
        android:value="@string/google_maps_key"
    />

    <activity
        android:name=".MapsActivity"
        android:label="@string/title_activity_maps">

        <intent-filter>
            <action android:name="android.intent.ac
tion.MAIN" />
            <category android:name="android.inte
nt.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
</manifest>

```



## Callback methods in Google Map

1. **OnMapReadyCallback:** This callback interface invokes when its instance is set on MapFragment object. The onMapReady(GoogleMap) method of OnMapReadyCallback interface is called when the map is ready to be used. In the onMapReady(GoogleMap) method, we can add markers, listeners, and other attributes.
2. **LocationListener:** This interface is used to receive notification when the device location has changed. The abstract method of LocationListener onLocationChanged(Location) is called when the location has changed.
3. **GoogleApiClient.ConnectionCallbacks:** This interface provides callback methods onConnected(Bundle) and onConnectionSuspended(int) which are called when the device is connected and disconnected.

4. **GoogleApiClient.OnConnectionFailedListener:** This interface provides callbacks method `onConnectionFailed(ConnectionResult)` which is called when there was an error in connecting the device to the service.

The **setMyLocationEnabled()** method of `GoogleMap` is used to enable location layer, which allows device to interact with current location.

## Introduction to SMS and MMS

Short Message Service is one of the killer applications which are one of the most prominent communication tools. Even a kid knows how to use SMS application. Android comes with a built-in SMS application that enables us to send and receive SMS messages. However, it is possible that we would want to integrate SMS application. SMS technology can be used to send/receive text messages or data messages. Android supports full SMS functionality within our applications through the `SmsManager` class. We can use this SMS Manager to replace the native SMS application to send text messages, respond to incoming texts, or use SMS as a data transport layer.

Multimedia messaging service (MMS) messages allow users to send and receive messages which includes multimedia attachments such as videos, photos and audio.

`SmsManager` manages SMS operations such as sending data, text, and SMS messages. This object is get by calling the static method **getDefault()**. The default SMS app is the one which is selected by the user in system settings. The default SMS app is able to write to the SMS Provider. SMS Providers are the tables defined within the `Telephony` class. Only the default SMS app receives the **SMS\_DELIVER\_ACTION** broadcast when the user receives an SMS or the **WAP\_PUSH\_DELIVER\_ACTION** broadcast when the user receives an MMS.

Application which wants to behave as the user's default SMS app must be able to handle the following intents:

- In a broadcast receiver we have to include an intent filter for **SMS\_DELIVER\_ACTION**. The broadcast receiver would require the **BROADCAST\_SMS** permission as well. This will allow our app to directly receive incoming SMS messages.

- In a broadcast receiver it has to include an intent filter for WAP\_PUSH\_DELIVER\_ACTION and it has to include the "MIME type". The broadcast receiver would also require the BROADCAST\_WAP\_PUSH permission. This will allow our app to directly receive incoming MMS messages.
- In our activity that would deliver new message should include an intent filter for ACTION\_SENDTO with schemas like sms:, smsto:, mms:, and mmsto:. This will allow our app to receive intents from other apps that would want to deliver a message.
- In a service it must include an intent filter for ACTION\_RESPOND\_VIA\_MESSAGE with schemas like sms:, smsto:, mms:, and mmsto:. This service would also require the SEND\_RESPOND\_VIA\_MESSAGE permission. This will allow users to respond to incoming phone calls with an immediate text message using your app.

Example :

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

    android:layout_width="fill_parent"

    android:layout_height="fill_parent"

    android:orientation="vertical" >

    <TextView

        android:layout_width="fill_parent"

        android:layout_height="wrap_content"

        android:text="Enter the phone number of recipient" />

    <EditText

        android:id="@+id/txtPhoneNo"

        android:layout_width="fill_parent"

        android:layout_height="wrap_content" />

    <TextView

        android:layout_width="fill_parent"

        android:layout_height="wrap_content"

        android:text="Message" />
```

```

<EditText

    android:id="@+id/txtMessage"

    android:layout_width="fill_parent"

    android:layout_height="150px"

    android:gravity="top" />

<Button

    android:id="@+id/btnSendSMS"

    android:layout_width="fill_parent"

    android:layout_height="wrap_content"

    android:text="Send SMS" />

</LinearLayout>

```

We can send sms in android via intent. You need to write only 4 lines of code the send sms in android.

*//Getting intent and PendingIntent instance*

```

Intent intent=new Intent(getApplicationContext(),MainActivity.class);
PendingIntent pi=PendingIntent.getActivity(getApplicationContext(), 0, intent,0)
;

```

*//Get the SmsManager instance and call the sendTextMessage method to send message*

```

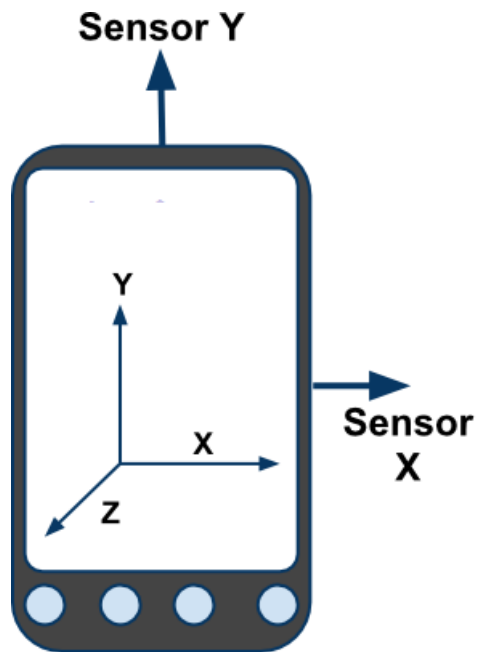
SmsManager sms=SmsManager.getDefault();
sms.sendTextMessage("mobile num", null, "Mobile application", pi,null);

```

## Android Sensor

**Sensors** can be used to monitor the three-dimensional device movement or change in the environment of the device.

Android provides sensor api to work with different types of sensors.



---

## Types of Sensors

Android supports three types of sensors:

### 1) Motion Sensors

These are used to measure acceleration forces and rotational forces along with three axes.

### 2) Position Sensors

These are used to measure the physical position of device.

### 3) Environmental Sensors

These are used to measure the environmental changes such as temperature, humidity etc.

#### Android Sensor API

Android sensor api provides many classes and interface. The important classes and interfaces of sensor api are as follows:

##### 1) SensorManager class

The **android.hardware.SensorManager** class provides methods :

- to get sensor instance,
- to access and list sensors,
- to register and unregister sensor listeners etc.

You can get the instance of SensorManager by calling the method `getSystemService()` and passing the `SENSOR_SERVICE` constant in it.

```
1. SensorManager sm = (SensorManager)getSystemService(SENSOR_SERVICE);
```

##### 2) Sensor class

The `android.hardware.Sensor` class provides methods to get information of the sensor such as sensor name, sensor type, sensor resolution, sensor type etc.

##### 3) SensorEvent class

**Its instance is created by the system. It provides information about the sensor.**

##### 4) SensorEventListener interface

**It provides two call back methods to get information when sensor values (x,y and z) change or sensor accuracy changes.**

Public and abstract methods	Description
-----------------------------	-------------

<b>void onAccuracyChanged(Sensor sensor, int accuracy)</b>	it is called when sensor accuracy is changed.
<b>void onSensorChanged(SensorEvent event)</b>	it is called when sensor values are changed.

## Android simple sensor app example

1. A sensor example that prints x, y and z axis values. Here, we are going to see that.
2. A sensor example that changes the background color when device is shuffled.

### activity\_main.xml

There is only one textview in this file.

*File: activity\_main.xml*

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="92dp"
        android:layout_marginTop="114dp"
```

```
        android:text="TextView" />
    </RelativeLayout>
```

## Activity class

Let's write the code that prints values of x axis, y axis and z axis.

*File: MainActivity.java*

```
package com.example.sensorsimple;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;
import android.hardware.SensorManager;
import android.hardware.SensorEventListener;
import android.hardware.SensorEvent;
import android.hardware.Sensor;
import java.util.List;
public class MainActivity extends Activity {
    SensorManager sm = null;
    TextView textView1 = null;
    List list;

    SensorEventListener sel = new SensorEventListener(){
        public void onAccuracyChanged(Sensor sensor, int accuracy) {}
        public void onSensorChanged(SensorEvent event) {
            float[] values = event.values;
            textView1.setText("x: "+values[0]+"\\ny: "+values[1]+"\\nz: "+values[2]);
        }
    };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```



```

setContentView(R.layout.activity_main);

/* Get a SensorManager instance */
sm = (SensorManager) getSystemService(SENSOR_SERVICE);

textView1 = (TextView)findViewById(R.id.textView1);

list = sm.getSensorList(Sensor.TYPE_ACCELEROMETER);
    if(list.size()>0){
        sm.registerListener(sel, (Sensor) list.get(0), SensorManager.SENSOR_DELA
Y_NORMAL);
    }else{
        Toast.makeText(getBaseContext(), "Error: No Accelerometer.", Toast.LENG
TH_LONG).show();
    }
}

@Override
protected void onStop() {
    if(list.size()>0){
        sm.unregisterListener(sel);
    }
    super.onStop();
}
}

```

