# Lec1_Summary

# Part 1 - Introduction

## What is Mobile Computing?

⇒the process of **distributed computation** (in more than one device and also connected with servers) on diversified mobile devices connected through **cellular and wireless network** using **standard internet communication protocols.**

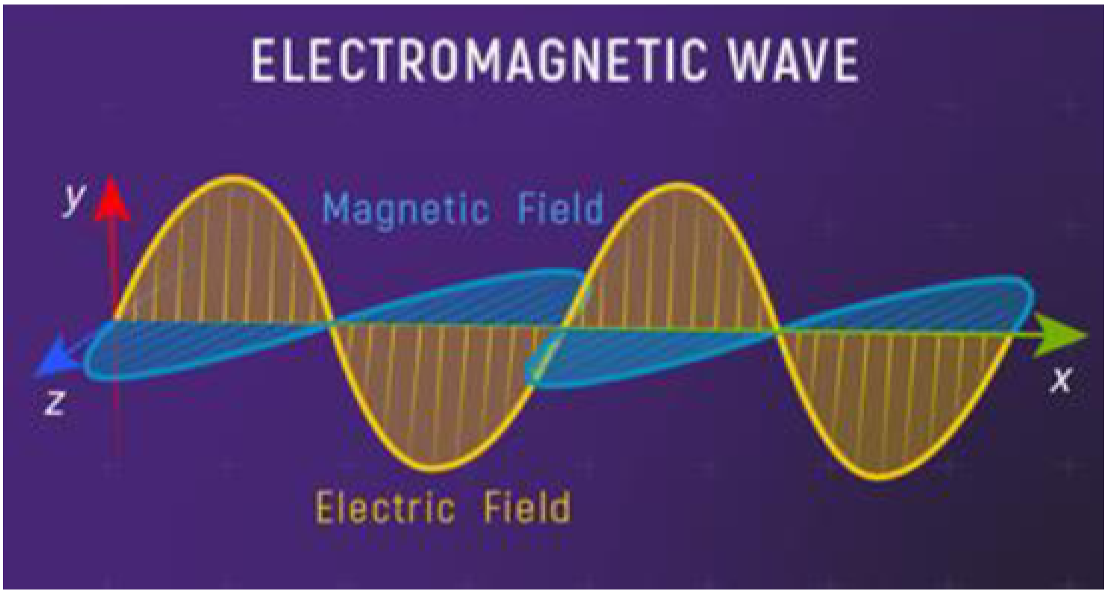
## How Communication Happens?

### Mechanical Waves

⇒ Require a medium (solid, liquid, or gas), ex: sound waves, water waves

### Electromagnetic Waves

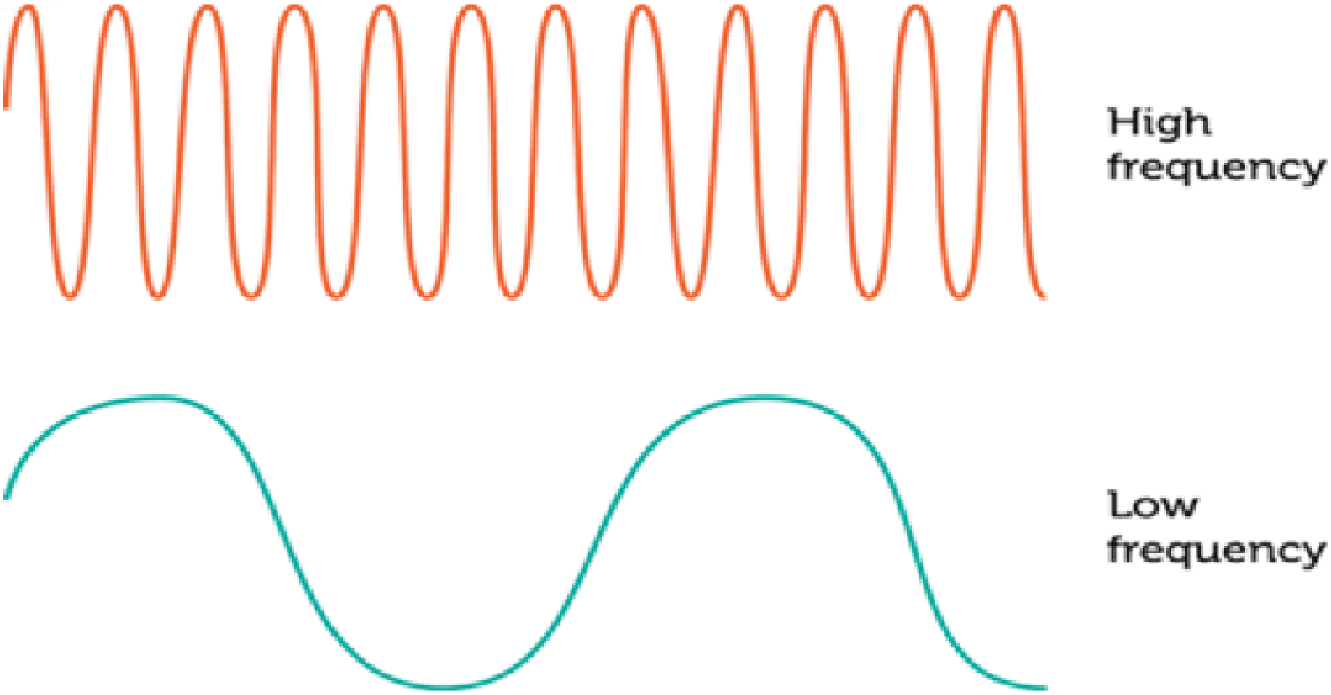
⇒ used in mobile communication (wireless communication)
⇒ not require a medium
⇒ ex: Radio waves, microwaves, infrared, visible light, ultraviolet, X-rays, and gamma rays.



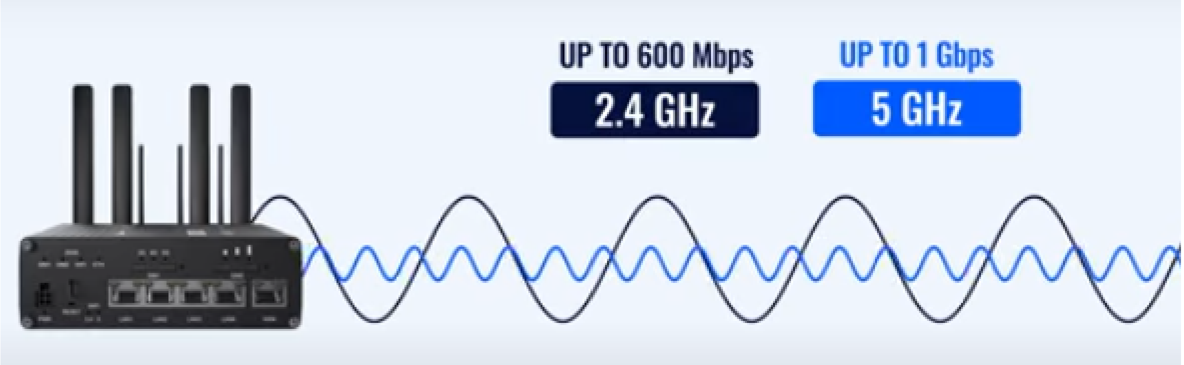
## Terminologies

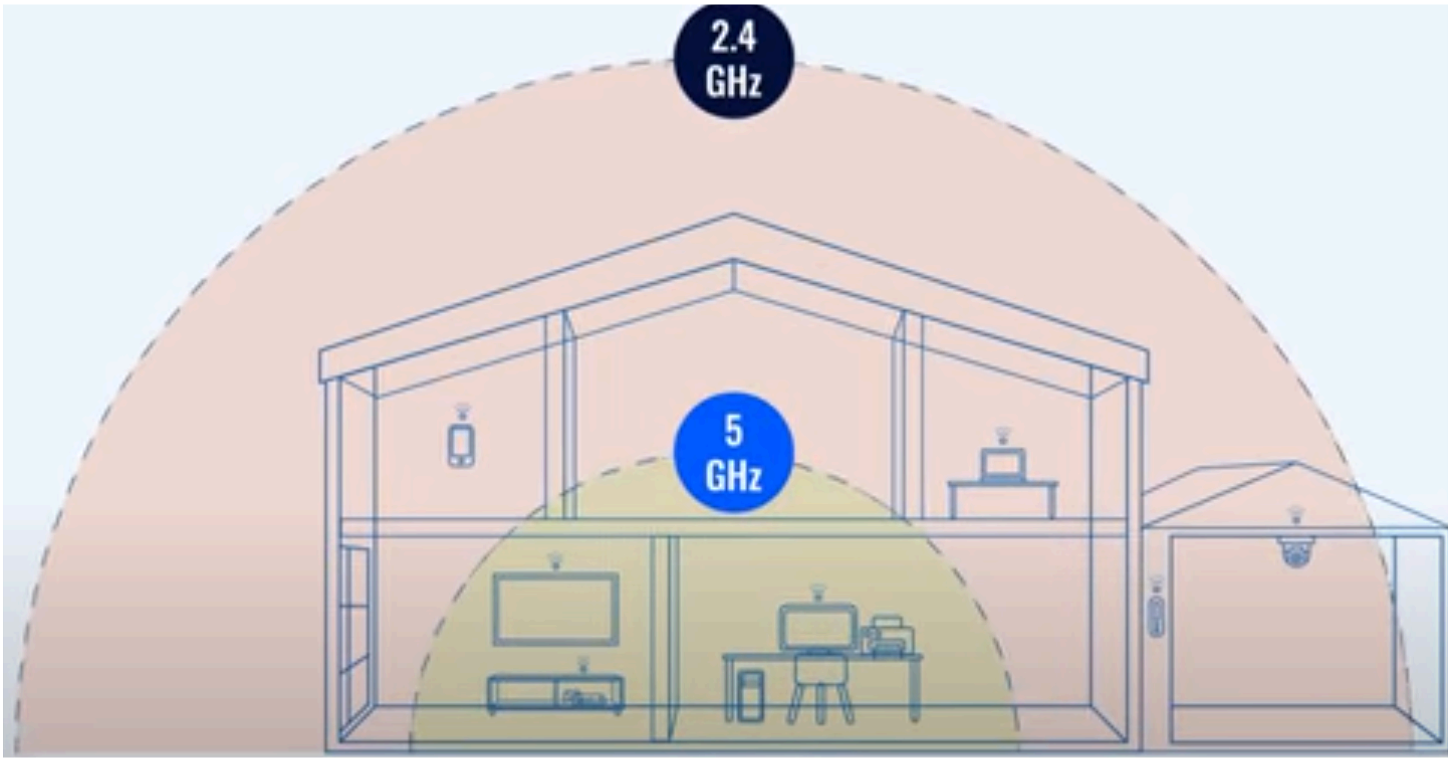
### Frequency

⇒ the number of occurrences of a repeating event per unit of time
⇒ measured in hertz(Hz)



**Low Freq Vs High Freq**



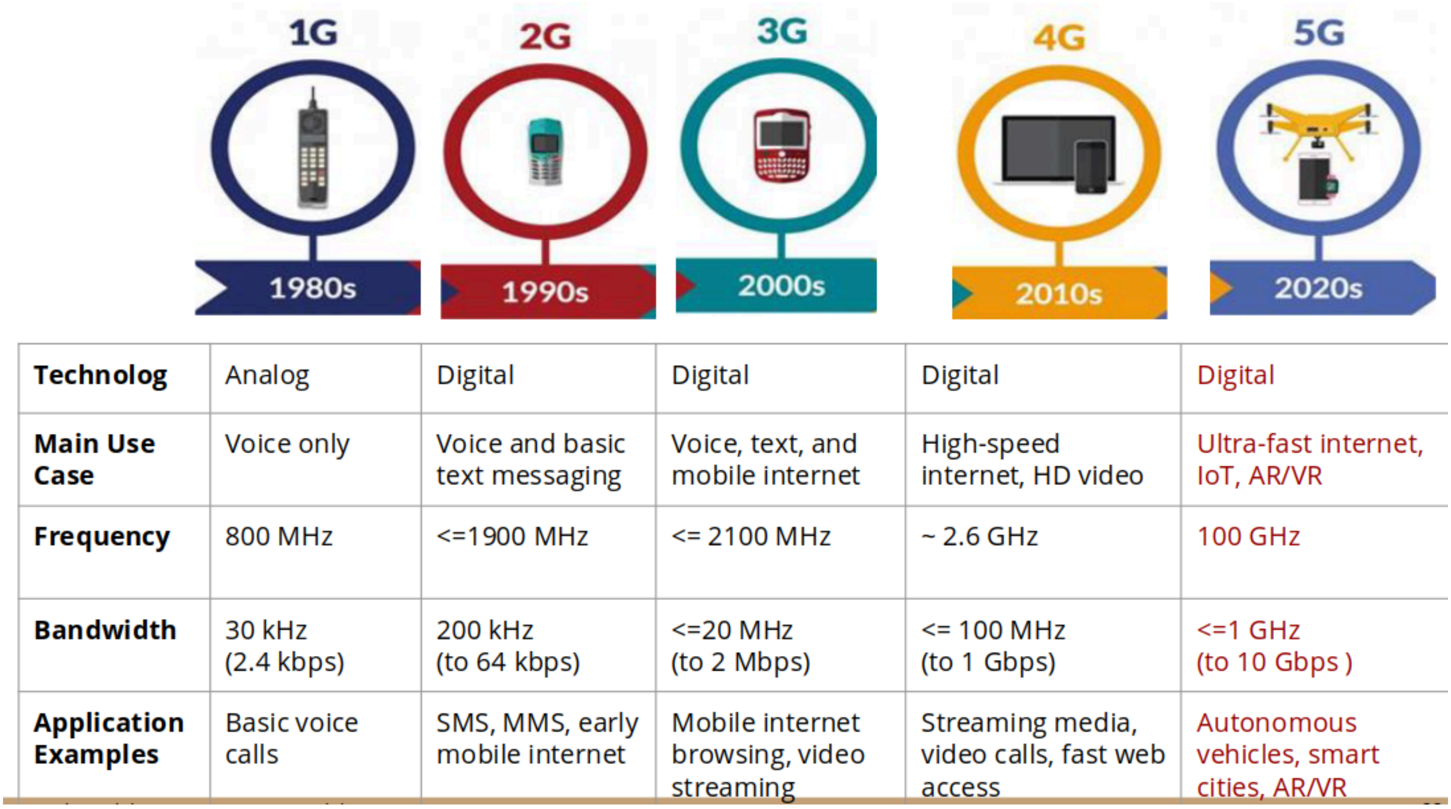


⇒ Low Frequency travel father and penetrate walls better

⇒ Higher frequency carry more about data but cover shorter distances and not good when there is closed area

- watch this video for more info(~2m) and is the source of photos above: [2.4 vs 5 GHz Wi-Fi: What's the Difference? - YouTube]

**The evolution of mobile network technology through its five generations**


1G
1980s
2G
1990s
3G
2000s
4G
2010s
5G
2020s

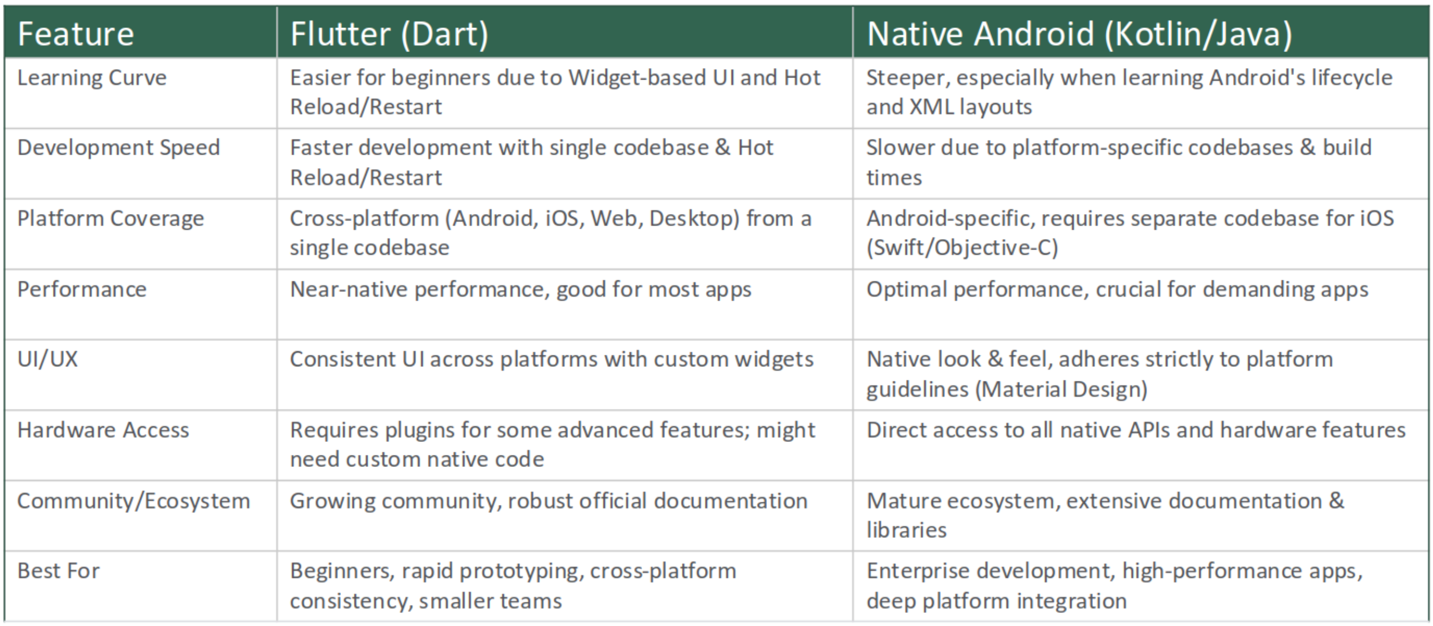
| | | | | | |
|---|---|---|---|---|---|
| **Technolog** | Analog | Digital | Digital | Digital | Digital |
| **Main Use Case** | Voice only | Voice and basic text messaging | Voice, text, and mobile internet | High-speed internet, HD video | Ultra-fast internet, IoT, AR/VR |
| **Frequency** | 800 MHz | <=1900 MHz | <= 2100 MHz | ~ 2.6 GHz | 100 GHz |
| **Bandwidth** | 30 kHz<br>(2.4 kbps) | 200 kHz<br>(to 64 kbps) | <=20 MHz<br>(to 2 Mbps) | <= 100 MHz<br>(to 1 Gbps) | <=1 GHz<br>(to 10 Gbps ) |
| **Application Examples** | Basic voice calls | SMS, MMS, early mobile internet | Mobile internet browsing, video streaming | Streaming media, video calls, fast web access | Autonomous vehicles, smart cities, AR/VR |

# Part 2 - Android

## Why Android?

1. Market Demand
2. Diverse Opportunities: doors to various roles
3. Learning Resources
4. Integration with other technologies

## Android VS Flutter



| Feature | Flutter (Dart) | Native Android (Kotlin/Java) |
|---|---|---|
| Learning Curve | Easier for beginners due to Widget-based UI and Hot Reload/Restart | Steeper, especially when learning Android's lifecycle and XML layouts |
| Development Speed | Faster development with single codebase & Hot Reload/Restart | Slower due to platform-specific codebases & build times |
| Platform Coverage | Cross-platform (Android, iOS, Web, Desktop) from a single codebase | Android-specific, requires separate codebase for iOS (Swift/Objective-C) |
| Performance | Near-native performance, good for most apps | Optimal performance, crucial for demanding apps |
| UI/UX | Consistent UI across platforms with custom widgets | Native look & feel, adheres strictly to platform guidelines (Material Design) |
| Hardware Access | Requires plugins for some advanced features; might need custom native code | Direct access to all native APIs and hardware features |
| Community/Ecosystem | Growing community, robust official documentation | Mature ecosystem, extensive documentation & libraries |
| Best For | Beginners, rapid prototyping, cross-platform consistency, smaller teams | Enterprise development, high-performance apps, deep platform integration |

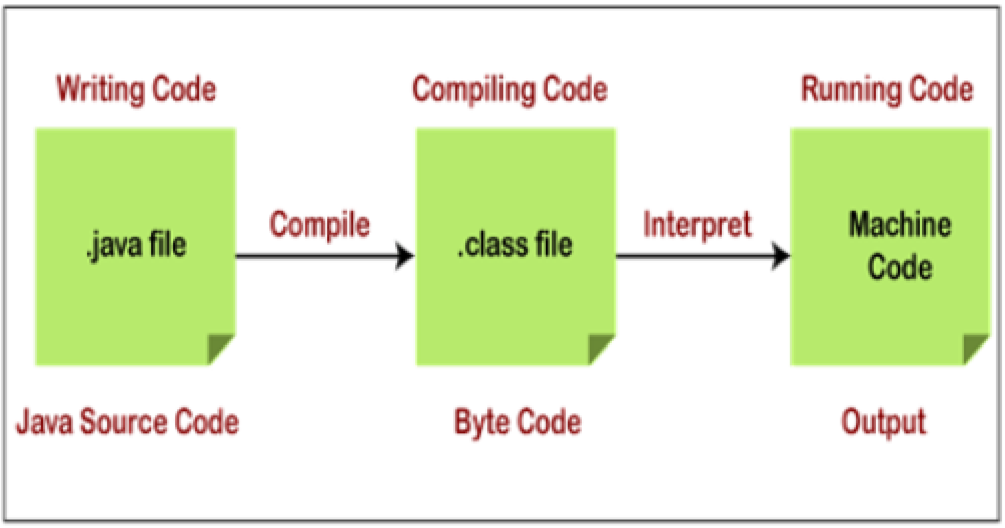
## Why Java

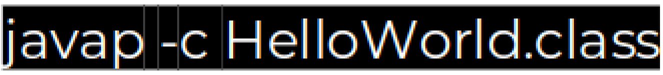
1. Versatility
2. Platform Independence: run in any device that has Java Virtual Machine (JVM)
3. Strong Communicty Support
4. Object-Oriented Programming (OOP)
5. Robustness and Security: strong memory management and security feature
6. Enterprise-Level Applications
7. In nutshell

## Compiler VS Interpreter

| Compler | Interpreter |
|---|---|
| - Translate entire source code to machine code<br>- output is an executable file that can be run independently of the original source code.<br>- Faster<br>- ex: C, C++, Rust | - translates source code line-by-line or statement-by-statement at runtime, executing it immediately.<br>- no executable file generated<br>- slower at runtime<br>- Python, ruby, JS |

> **Java Behavior:**





```
javap -c HelloWorld.class
```

1. Compile source code → bytecode
2. bytecode is analogous to machine code but run with virtual machine for the host hardware (Linux, windows,..etc)

## Android Platform Architecture

⇒ Android is an open source, Linux-based software stack.
⇒ to access kernel of the phone : use Android Debug Bridge (**adb**) or Debug Bridge Enhanced (**adb-enhanced**).

### 1. Linux Kernel

- relies on underlying functionalities of the kernel
  - Threading
  - low level memory management (shared memory)

> **Advantage to use linux:** security features and lets device manufacturers develop hardware drivers for a well-known kernel.

> **Commands:**

- `brew install adb` → to access kernel
- `adb shell` → access the kernel shell
- `uname -r` → kernel version
- `uname -a` → logs
- `cat /proc/cpuinfo` → cpu info
- `cat /proc/meminfo` → memory info
- `top` → memory and cpu usage (like task manager in windows)
- `iostat` → I/O statistics

### 2. HAL (Hardware Abstraction Layer)

- function: expose device hardware capabilities to the higher level Java API framework.
  - Bluetooth module
  - the camera
  - Audio
- when a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component.

> To interact with hardware components via the HAL use the services via adb shell. commands:

- `service list`: to list all services
- `service call audio <command>`: to call specific service

## 3. Android Runtime (ART)

- Android version 5 and higher → runs its own process with its own instance of the Android runtime (ART).
- ART Run Dalvik executable format (DEX) files.
- DEX:
  - **d8** tool compile Java sources into DEX bytecode
  - DEX is a bytecode format specifically for Andriod

## 4. Native C/C++ Libraries

- ART and HAL, built from native code required native C and C++ libraries.
- to access these native libraries directly in android → use the Android NDK (Native Development Kit)

## 5. Java API Framework

⇒ The entire feature-set of the Android Os available through APIs written in the java language.

> some of modular system components and services that are provided by Java API Framework

- **View System:** building the app's **user interface** (UI)—all the buttons, text boxes, and screens.
- **Resource Manager:** manage **non-code parts**, like images, layout files, and text strings for different languages.
- **Notification Manager:** lets your app **create notifications** in the phone's status bar.
- **Activity Manager:** manages the app's **lifecycle** (when it opens, closes, or goes to the background) and the back stack (what happens when you press the "back" button).

- **Content Providers: sharing data** between apps, like how your app asks for permission to read the phone's main contact list.

## System Apps

- **What They Are:** These are the core, preinstalled apps that come with Android, such as the Email, Calendar, and Camera.
- **No Special Status:** This means that third- party app can become the user's default web browser, SMS messenger.
- These apps have **two functions**:
  - Apps for the user.
  - Provide core services that other developers can use. For example, using SMS feature, a developer can have their app "invoke" for installed SMS app to send a message.