SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Mobile Application Development

Introduction to App Development

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Introduction to the module

- Student Centered Learning module
- Module Code: IT2010
- Credit Value: 4
- Method of Delivery
  - Lectures – 1 hour/week
  - Tutorials (practical) – 2 hours/week
  - Labs - 2 hours/week
- Courseweb Enrollment Key: IT2010

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Lecture Plan

- Introduction to App Development
- Mobile Platforms and Application Development fundamentals
- Mobile Interface Design Concepts and UI/UX Design
- Introduction to Android Operating System
- Main Components of Android Application
- Sensors and Media Handling in Android Applications
- Data Handling in Android Applications
- Android Application Testing and security aspects

Assessment Criteria

Assessment	Weight (%)
Midterm (MCQ)	20
Mini Project - Phase 01	10
Mini Project - Phase 02	20
Final Exam	50

Lecturer Panel

- Lecturer in-charge of the module
  - Mr. Thusithanjana Thilakarathna (thusithanjana.t@sliit.lk)
- Malabe
  - Mr. Nelum Chathuranga Amarasena (nelum.a@sliit.lk)
  - Ms. Devanshi Ganegoda (devanshi.g@sliit.lk)
  - Ms. Karthiga Rajendran (karthiga.r@slit.lk)
  - Ms. Piyumika Samarasekara (piyumika.s@sliit.lk)
  - Ms. Rivoni De Zoysa (rivoni.d@sliit.lk)
- Metro
  - Ms. Rivoni De Zoysa (rivoni.d@sliit.lk)
- Kandy
  - Ms.Gihani Gunarathna (gihani.g@sliit.lk)
  - Mr. Buwaneka Senevirathne (buwaneka.s@sliit.lk)
- Matara
  - Ms.Suriyaa Kumari (suriyaa.k@sliit.lk)

Learning Outcomes of the Lecture

- At the end of this Lecture students will be able to:
  - Understand the fundamentals of mobile Application Development
  - Explain different mobile platforms and related technologies.
  - Code a simple program in Kotlin

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Why a Mobile App?

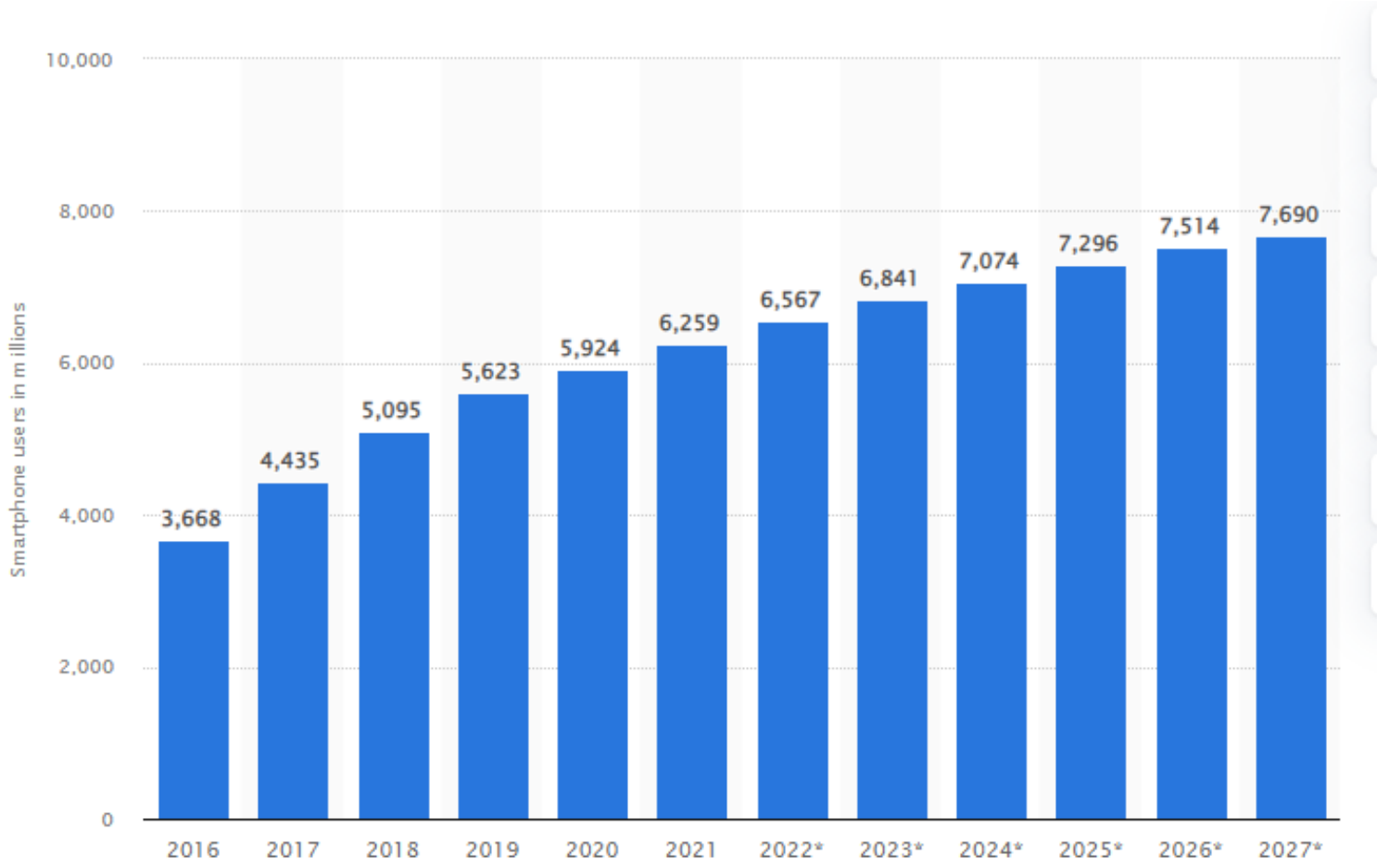
Mobile phones are no longer the ordinary communication device. It has various incredible features and opportunities offered to the users.

The number of smartphone users is growing up day by day. (In 2020, it's 3.5 billions)

Business organizations are more like to have mobile applications for their business instead of investing in a mobile friendly version of their websites.

Good mobile application will add value to the business.

Smartphone users' growth around the world



Reference: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

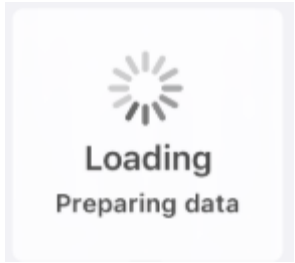
Mobile Application Development

- Mobile application development refers to the creation of apps for use on devices such as tablets, smartphones, automobiles and watches.
- Mobile development often incorporates features of mobile devices that may not be available on desktop devices.

  An example of this is the ability to operate a device or play a game simply by moving the smartphone around in space.

Key Features of Mobile Applications

- Great UI (User Interface)
- Fast loading time and high performance
- Extremely helpful user support
- Adapts to a user's needs
- Compatible with a mobile platform



Reasons for Mobile App failures

- The app doesn't have a market
- The app does not have adequate security
- The app does not perform quickly enough
- The app does not fully consider UX/UI
- The app's listing in the marketplace is not persuasive
- Hard to adjust web version to the smartphone screen
- Due to limited functions

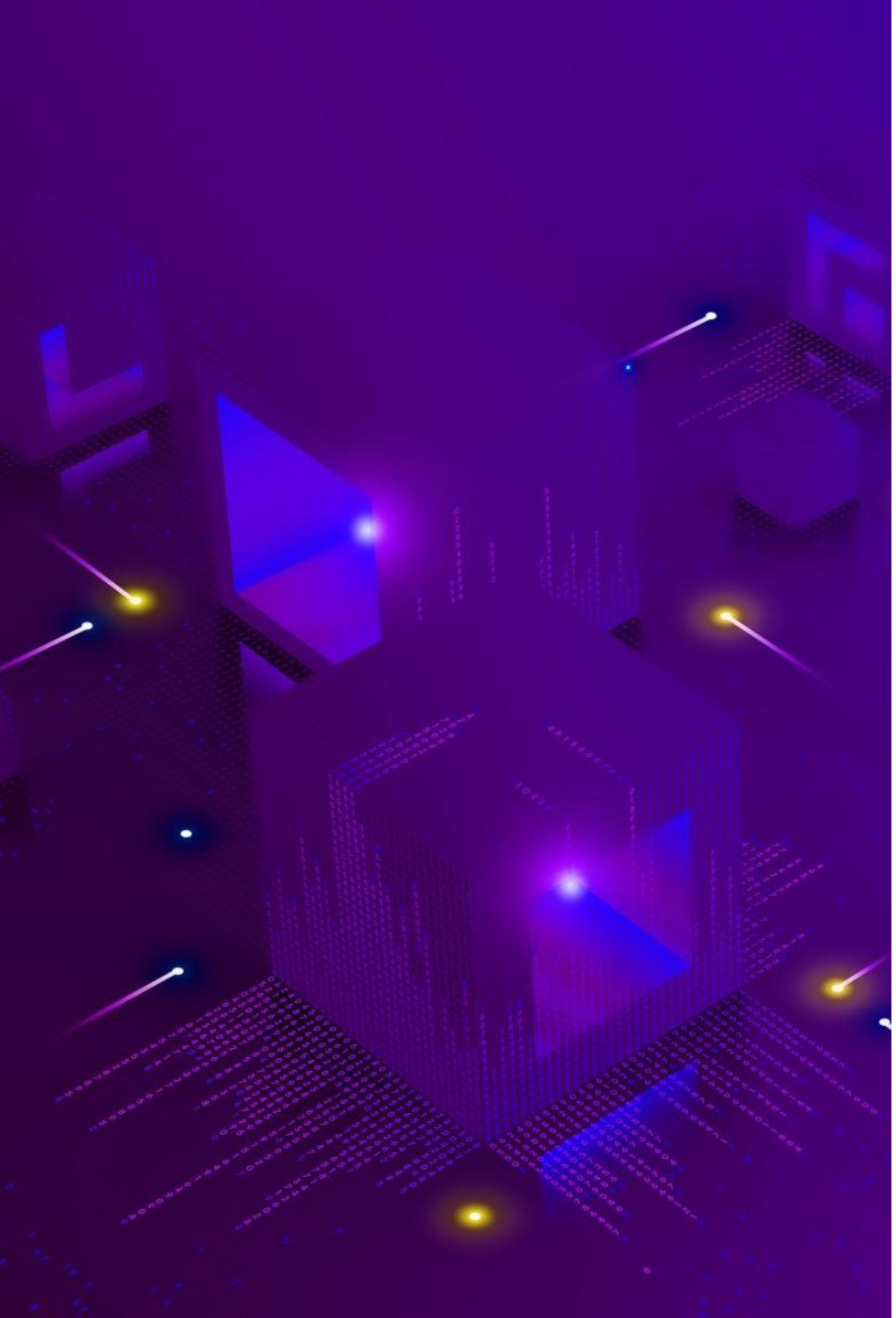
Mobile app development platforms

Android – Android Studio

iOS – Xcode

HarmonyOS – DevEco Studio

Cross Platform – Flutter, React Native, Xamarin

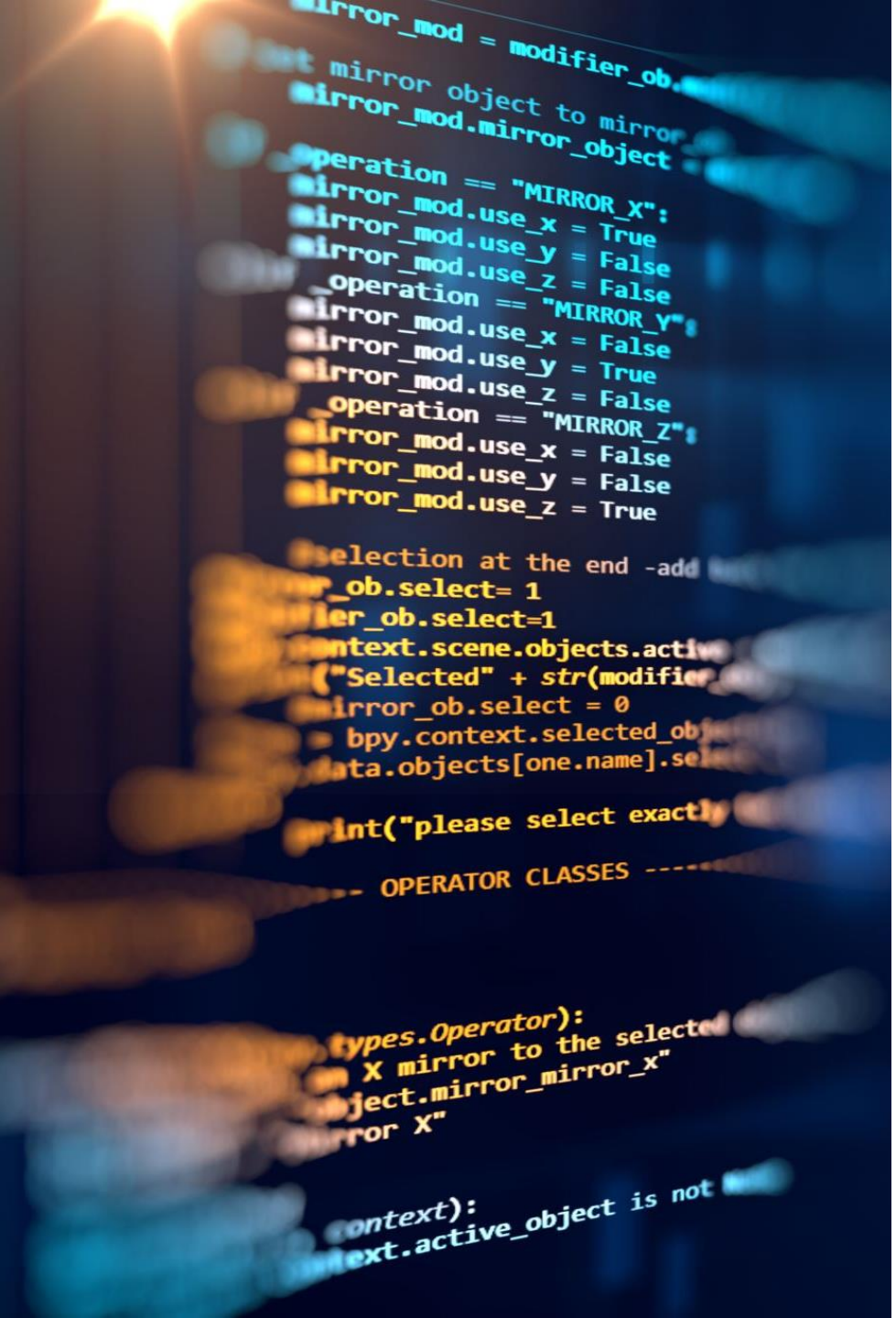


Android Studio

- Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development.
- It is a replacement for the Eclipse Android Development Tools (E-ADT) as the primary IDE for native Android application development.
- Android Studio was announced on May 16, 2013, at the Google I/O conference.
- At the end of 2015, Google dropped support for Eclipse ADT, making Android Studio the only officially supported IDE for Android development.
- On May 7, 2019, Kotlin replaced Java as Google's preferred language for Android app development. Java is still supported, as is C++.

Kotlin

- Modern, concise and safe programming language
- Easy to pick up, so you can create powerful applications immediately.
- https://kotlinlang.org/



Why is Android development Kotlin-first?

- **Expressive and concise:** You can do more with less. Express your ideas and reduce the amount of boilerplate code. 67% of professional developers who use Kotlin say Kotlin has increased their productivity.
- **Safer code:** Kotlin has many language features to help you avoid common programming mistakes such as null pointer exceptions. Android apps that contain Kotlin code are 20% less likely to crash.
- **Interoperable:** Call Java-based code from Kotlin or call Kotlin from Java-based code. Kotlin is 100% interoperable with the Java programming language, so you can have as little or as much of Kotlin in your project as you want.
- **Structured Concurrency:** Kotlin coroutines make asynchronous code as easy to work with as blocking code. Coroutines dramatically simplify background task management for everything from network calls to accessing local data.

Things you should know about Kotlin

- Variable Declaration
- Type inference
- Null Safety
- Conditionals
- Functions
- Classes
- Asynchronous code

Kotlin: Variable Declaration

- Use ‘val’ for a variable whose value never changes. You can't reassign a value to a variable that was declared using val.
- Use var for a variable whose value can change.

```
var count: Int = 10
```

```
var count: Int = 10
count = 15
```

Type Inference

- the Kotlin compiler can infer the type based on the type of the assigned value.

```
val name = "James Bond"
val age = 50
val isFlag = true
val pi = 3.14f
```

Null Safety

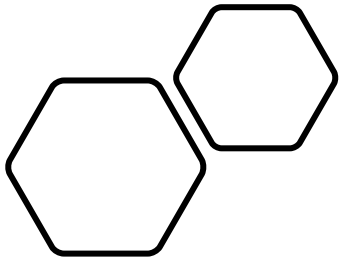
- In some languages, a reference type variable can be declared without providing an initial explicit value.
- In these cases, the variables usually contain a null value.
- Kotlin variables can't hold null values by default. This means that the following snippet is invalid:

```
// Fails to compile
val languageName: String = null
```

- For a variable to hold a null value, it must be of a nullable type.
- You can specify a variable as being nullable by suffixing its type with ?, as shown in the following example:

```
val languageName: String? = null
```

Thank you!



The tutorial sessions will include more Kotlin programming exercises.

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Mobile Application Development

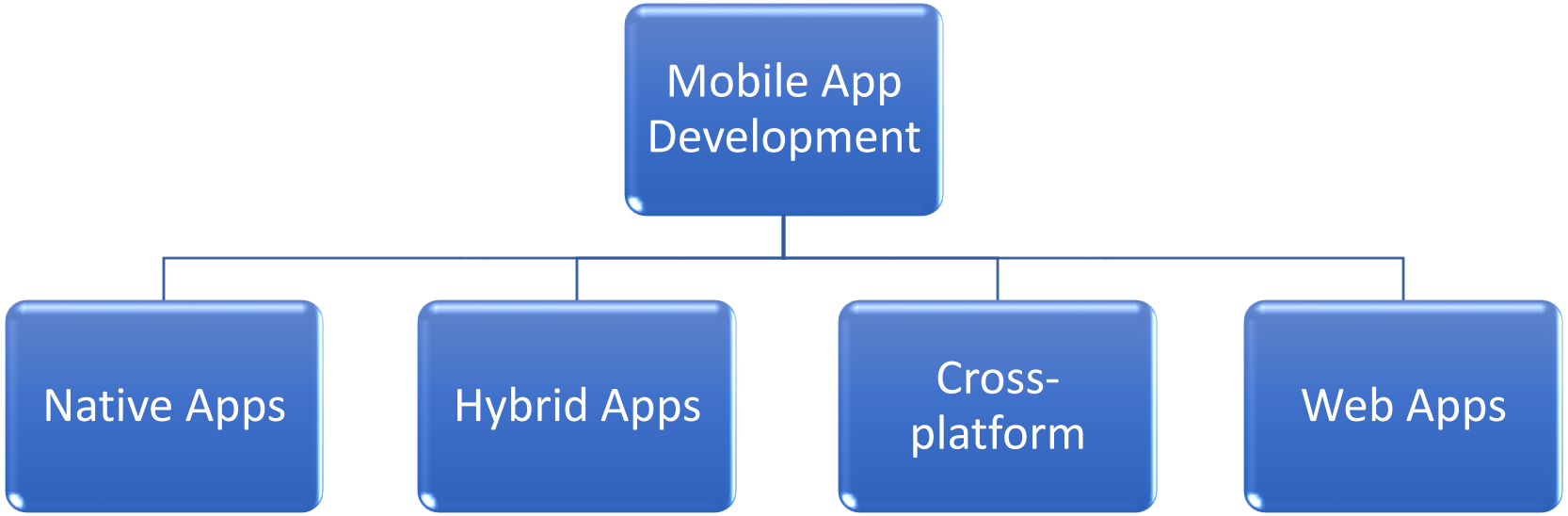
Mobile Platforms and Application Development fundamentals

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Lecture Plan

- Introduction to App Development
- Mobile Platforms and Application Development fundamentals
- Mobile Interface Design Concepts and UI/UX Design
- Introduction to Android Operating System
- Main Components of Android Application
- Sensors and Media Handling in Android Applications
- Data Handling in Android Applications
- Android Application Testing and security aspects

Mobile Application Development



Native Mobile Application

- A native mobile app is an application developed using platform-specific development tools.
- These apps are developed individually for each of the three popular mobile operating systems.

Android

- Android is a mobile operating system developed by Google, based on a modified version of the Linux kernel and other open source software. It is primarily designed for touchscreen mobile devices such as smartphones and tablets.
- Android is the most popular mobile operating system at present.
- Founders of android were Rich Miner, Nick Sears, Chris White, and Andy Rubin.

Android versions

Android Devices

Devices using android operating system

Smartphones

- Samsung
- Sony
- HTC
- Google
- LG
- Lenovo
- Oppo
- Huawei

Android Devices

Tablets

- Samsung Galaxy Tab
- Asus ZenPad
- Huawei MediaPad
- Lenovo Yoga Tab
- Amazon Fire HD
- Sony Xperia Z4 Tablet
- Nvidia Shield Tablet K1

Android Devices

TV

- Sony Bravia Smart TV
- Sharp Smart TV
- Philips Smart TV

Smartwatch

- Ticwatch
- LG Watch Style
- Misfit Vapor
- Asus ZenWatch
- Fossil Q Venture

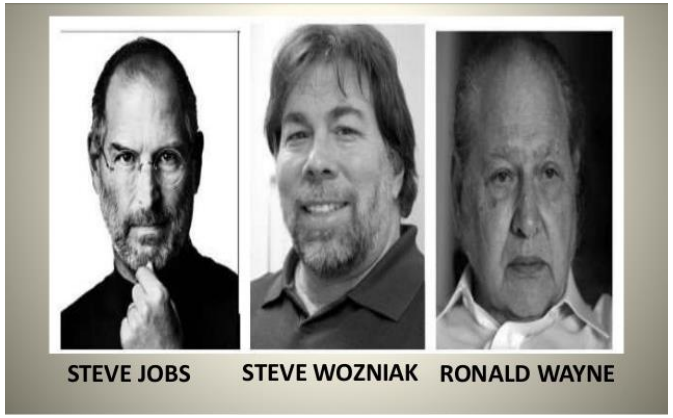
Android Devices

Development Environments

- **Android Studio**
- Eclipse
- Apache Cordova
- App Inventor for Android
- C++ Builder
- Blue J
- FlashDevelop
- Titanium

iOS

- iOS is a mobile operating system created and developed by Apple Inc.
- It is exclusively designed for Apple hardware.
- It is the second most popular mobile operating system globally after Android.
- Founders of iOS/Apple were Steve Jobs, Steve Wozniak, and Ronald Wayne



iOS Devices

Devices using iOS operating system

- iPhone
- iPod Touch
- iPad
- iPad Mini
- iPad Pro
- Apple TV
- Apple Watch

iOS

Development Environments

- Xcode
- AppCode
- Apache Cordova

Windows Mobile

- Windows Mobile is a discontinued family of mobile operating systems developed by Microsoft.
- Its origin dated back to Windows CE in 1996, though Windows Mobile itself first appeared in 2000 as PocketPC 2000.
- It was renamed "Windows Mobile" in 2003, at which point it came in several versions and was aimed at business and enterprise consumers

Windows Mobile

Devices using windows mobile operating system

- Dopod 515
- Krome Intellekt iQ200
- Mitac Mio 8390 and 8860
- Motorola MPx200
- O2 Xphone
- Orange SPV E200 and e100
- QTEK 7070 and 8080
- Sagem myS-7

Windows Mobile

Development Environments

- Visual Studio
- Apache Cordova

Hybrid App Development

- Less time for development.
- Allows for code sharing.
- Blend web elements with mobile ones.
- Create codebase using standard web technologies (HTML, CSS, JavaScript)

Tools:

Example

Cross-platform mobile application development

Cross-platform mobile application development refers to the development of mobile apps that can be used on multiple mobile platforms.

Cross-platform mobile application development

Development Environments

- Apache Cordova
- PhoneGap
- Xamarine
- Ionic
- Framework 7
- React Native
- Jasonette

Cross-platform mobile application development

Advantages

- Codes can be reused
- Controls Cost
- Quicker development time
- Easier Implementation
- Sameness and Uniformity

Cross-platform mobile application development

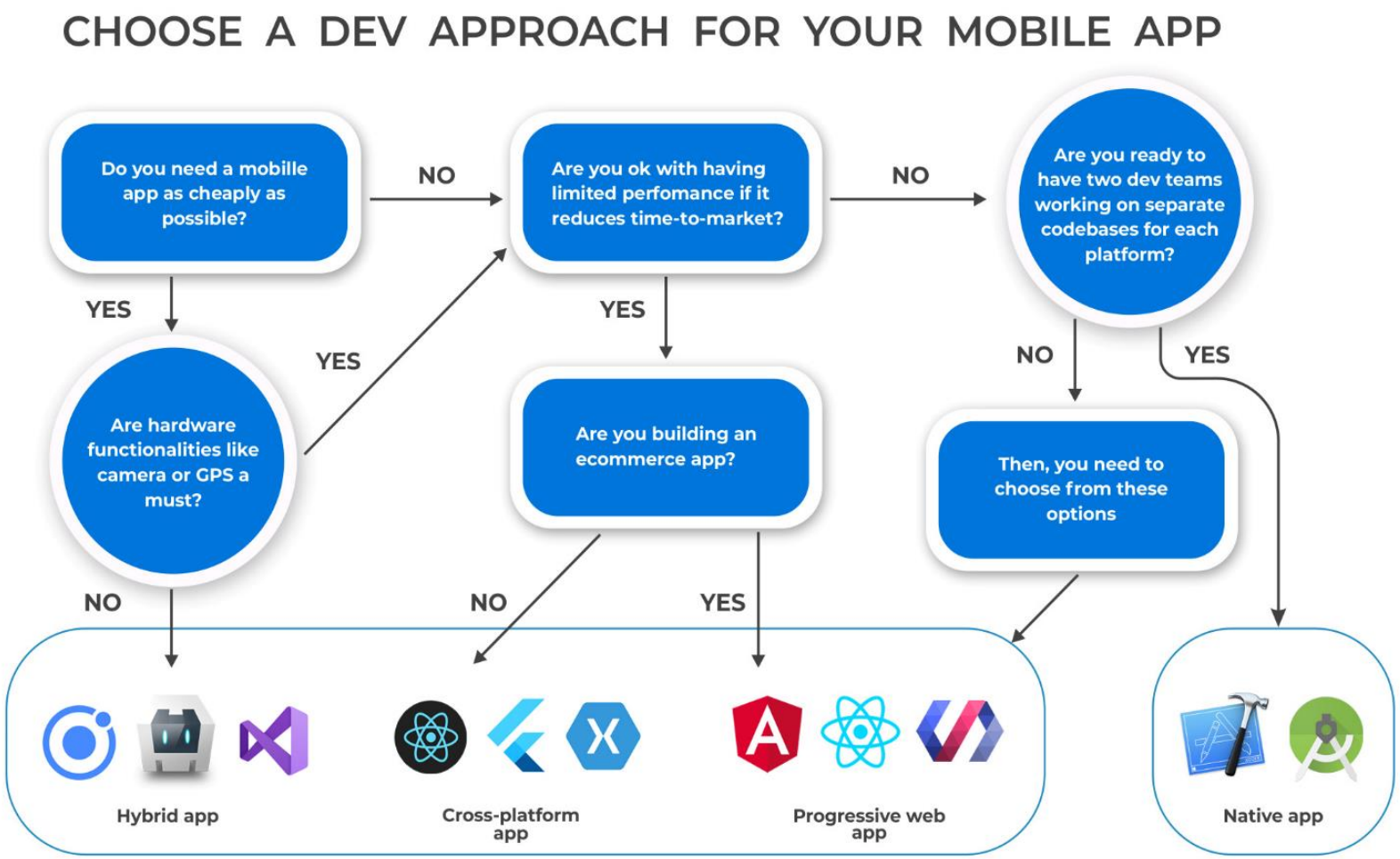
Disadvantages

- Loss of Flexibility
- Problems in platform Integration
- Diversity in user Interaction
- Poor user experience
- Difficulty in satisfying all users

App Type	Native	Hybrid	Cross-platform
Tools	• XCode • AppCode • Android Studio	• Ionic • Apache Cordova • Visual Studio	• React Native • Xamarin • Flutter
Rendering Engine	Native	Browser	Native
Libraries	Not much dependency on open-source libraries and platforms	Highly dependent on different libraries and frameworks	Highly dependent on different libraries and frameworks
Debugging	Native debugging tools	Native + web development debugging tools	Depends on the framework
Codebase	Separate codebase – one per platform	Single codebase with potential platform-specific abilities	Single codebase with potential platform-specific abilities

App Type	Native	Hybrid	Cross-platform
Pros	• Full access to device's/ OS's features • Powerful performance • Native UI (updating along with the OS) • Efficient App Running • High-quality functionality and UX • Access to all native APIs and the platform-specific functionality	• Lower development cost • Different OS support • Code reuse • Cost effective development • Big customization capabilities	• Different OS support • UI performance is almost as fast as native • Code reuse • Cost-effective development
Cons	• No multi-platform support • High dev cost if different OS support is needed • No code reuse	• Slower performance • Limited access to OS features • No interaction with other native apps	• *Slower performance • Limited access to OS features • Poor interaction with other native apps

Choose a Development approach for your Mobile App



Fundamentals of Mobile Application Development

Choice of Technology

- In advance to choosing any technology platform, one must ensure it is feasible in every way possible.
- Most appearing platforms are Android, iOS and Windows, and they are evolving rapidly with frequent handy updates. These platforms make it practically possible for developers to build unique features and impressive interface to deliver outstanding user experience.
- Choosing the right platform means your apps will be supported by numerous devices used by customers.

Fundamentals of Mobile Application Development

Clear recognition of requirements

- Define and set your final goals where you want to reach so that you can make a clear strategy and avoid confusion down the path of development.
- Knowing your goals enrich your vision and helps you develop apps that hit the precise pain point.
- Detailed analysis of the product and target audience helps to build an effective app

Fundamentals of Mobile Application Development

Dynamic Functionalities

- Mobile application users like to explore a heterogeneous set of interactive functionalities such as GPS, transactions, messages, responsiveness, sensors, and even audio/video.
- Most application use these interactive functionalities to attract users.

Fundamentals of Mobile Application Development

Security and Speed Efficiency

- Security problems are potential threats to customers who will become the end users of the app. Choose a reliable, secure, authentic resources and industry-standard processes to build the app to ensure its highly secure.
- A mobile app should respond instantly to process customer requests in time. Ensure that the application is effective normal internet environment.

Fundamentals of Mobile Application Development

Testing Quality and Consistency

- Testing the app is a crucial stage for any developer as it confirms whether or not the app is ready to deploy.
- An ideal app testing method must include testing on different devices of varied screen sizes in order to measure its performance and view its compatibility.
- Developer must also necessarily maintain the consistency while coding the app to make sure the entire mobile app development process, along with its documentation and program updates and interface, is genuine, consistent and clear.

Fundamentals of Mobile Application Development

Introduce a Pilot Version

- Once the development team is confident that they have built a well-tested, mature and fully functional app, they can go for launching the pilot product.
- The course of ideal mobile app development must end with the launch of pilot version.
- It helps developers receive the feedback and responses from the users and judge the success of the app.

Mobile Application Design Tools (Prototyping tools)

"If a picture is worth a thousand words, a prototype is worth a 1000 meetings"

Tom & David Kelley

Mobile Application Design Tools (Prototyping tools)

- Invision
- UXPin
- Sketch
- Slicy
- Skala Preview
- PlaceIt
- AdobeColor
- FontFace Ninja
- Illustrator & Photoshop
- Omnigraffle
- Proto.io
- After Effects
- Fluid UI
- Adobe XD
- Figma

Thank you!

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Mobile Application Development

Mobile Platforms and Application Development fundamentals

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Lecture Plan

- Introduction to App Development
- Mobile Platforms and Application Development fundamentals
- Mobile Interface Design Concepts and UI/UX Design
- Introduction to Android Operating System
- Main Components of Android Application
- Sensors and Media Handling in Android Applications
- Data Handling in Android Applications
- Android Application Testing and security aspects

Simplicity of the UI

- Different Perspectives for the design
  - User
  - Manager
  - Engineer
- User's perspective is the main thing
- Make things feel simple to use
- Making things simple does not mean it is created using simple technologies

Simpler than a bike.
Until you try to ride it.

Complexity makes it Unsustainable

- Having too many features
- User will not use everything
- Users may feel, they are paying for unnecessary features
- There will be massive legacy code that makes the product more expensive to maintain

All that unnecessary power comes at a price

Simplicity should not be faked

- Appear to be simple
- No real usage
- If you must show a magic character to explain how things work, it is not simplicity
- Simplicity isn't something you can stick on top of a user interface

Understand the user's environment

Offices

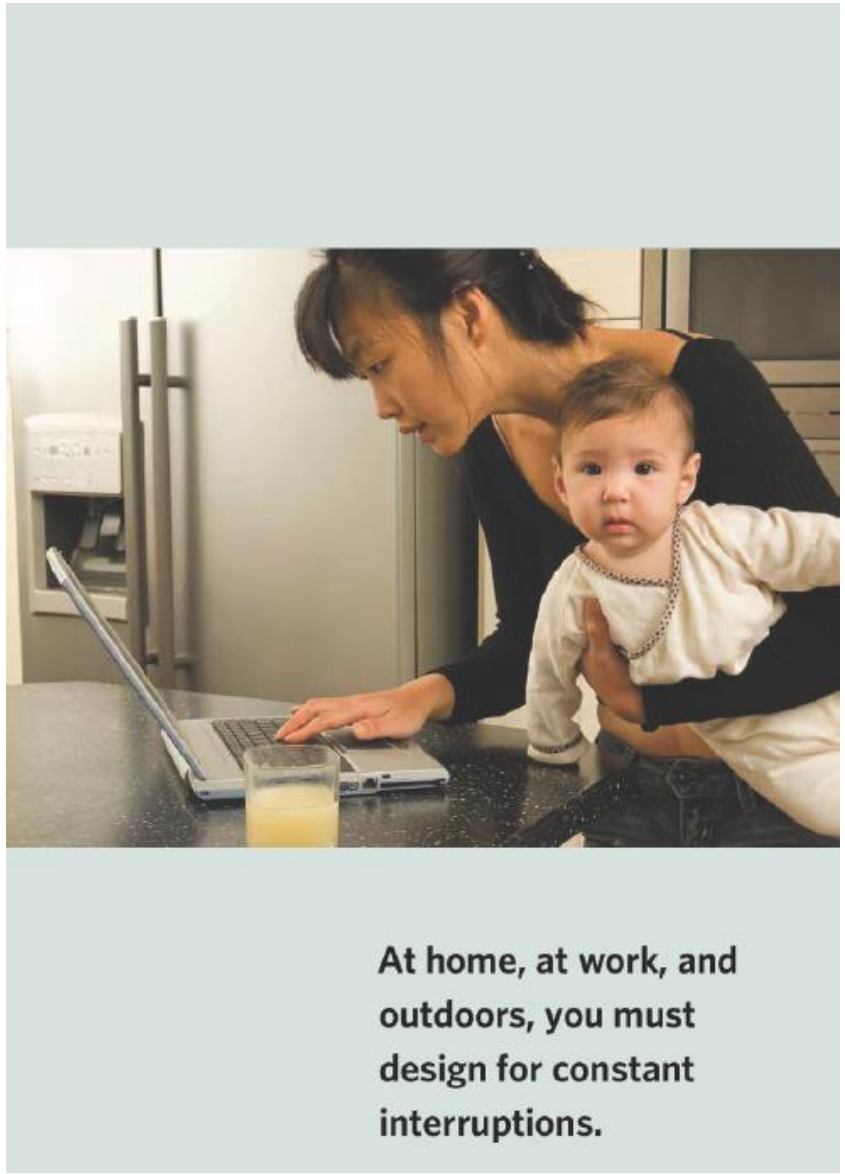
- People are getting interrupted
- Telephone, messages emails
- Last minute work

Homes

- Do multiple things at ones (Laptop While Watching TV)
- Home Broadband may not be reliable
- Mothers, Fathers, Children

Outdoors

- Busy streets
- Carrying bags
- In Ques
- Bright sun light
- Large devices may not be carried



At home, at work, and outdoors, you must design for constant interruptions.

Types of users

Experts

- Will explore the product
- Want never seen technology
- They will spend time with your product

Willing Adopters

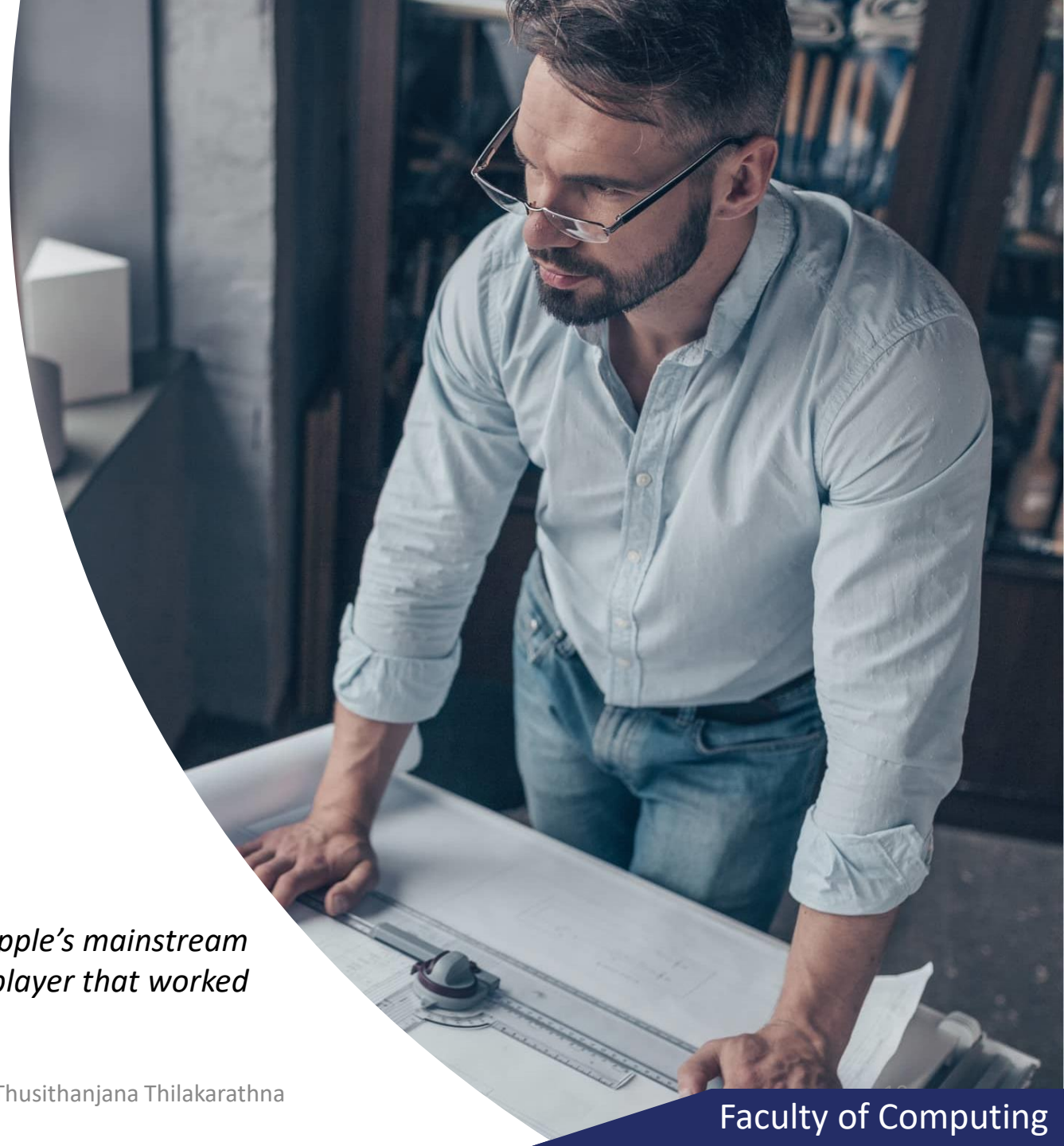
- Already using similar products
- Not comfortable with new products
- They need easy ways to adopt to new features
- Very fewer of these type of users

Mainstreamers

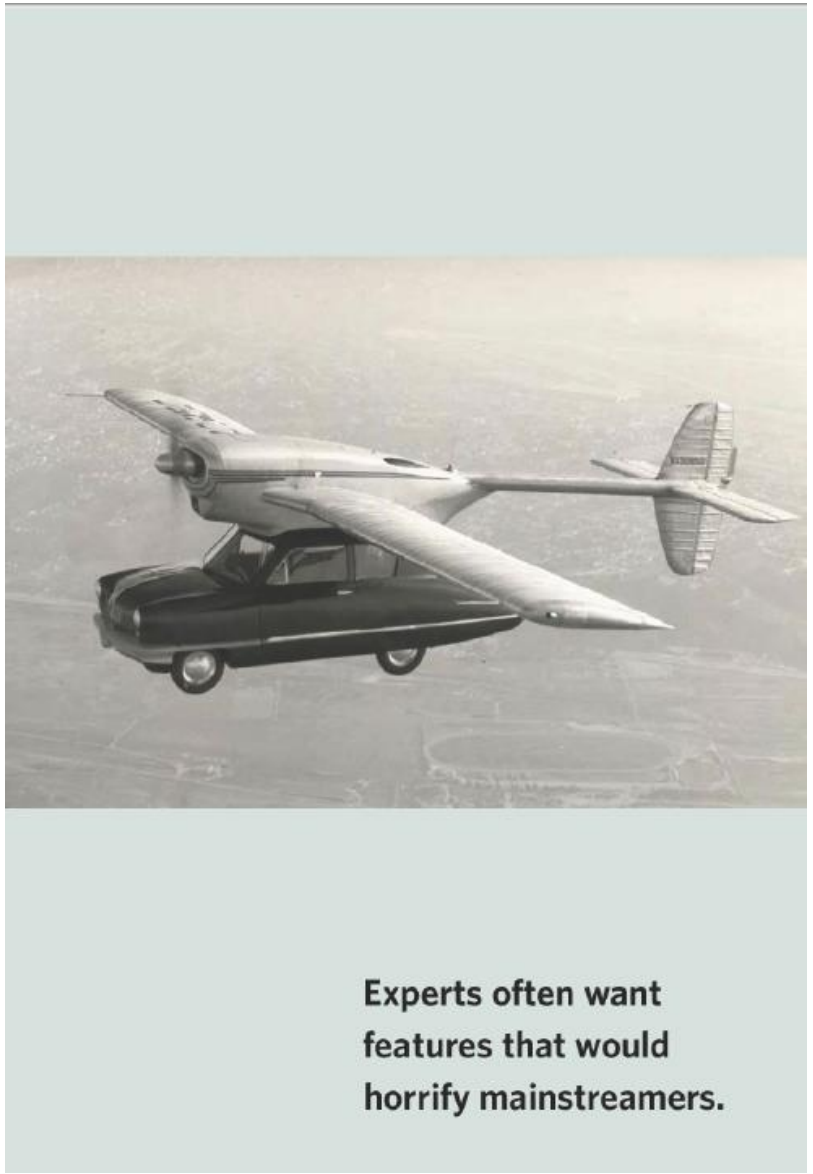
- Majority is this category
- They don’t use the technology for its own sake; use it to get the job done
- Only use few key features
- “I just want my phone to work” attitude

Experts can be ignored

- Highly technical
- won’t face issues in your app like mainstreamers
- They care about highly technical details
- Always expect wonders
- Their demands may be complicated for the other users



Apple’s expert customers wanted a flying car. Apple’s mainstream customers just wanted an MP3 player that worked


Experts often want
features that would
horrify mainstreamers.

Design for the Mainstream

- Usable design tends to focus on this group
- You can learn a lot by watching these people
- They are the majority
- They just want your app to work
- They hate the complexity

> We will build a motor car for the great multitude. It will be…small enough for the individual to run and care for. It will be constructed…after the simplest designs modern engineering can devise. But it will be so low in price that no man making a good salary will be unable to own one.
>
> — *Henry Ford, on the Model T*

Mass appeal comes from focusing on the mainstream

What mainstreamers want

- Mainstreamers are interested in getting the job done now; experts are interested in customizing their settings first
- Mainstreamers value ease of control; experts value precision of control
- Mainstreamers want reliable results; experts want perfect results.
- Mainstreamers are afraid of breaking something; experts want to take things apart to see how they work.
- Mainstreamers want a good match; experts want an exact match
- Mainstreamers want examples and stories; experts want principles

Mainstreamers don't want to build from scratch.

Emotional Need

When we thought about why people would use our software, we realized that they had a lot on their plate. They wanted to achieve a lot and still feel in control. They needed to be able to capture a thousand items and yet not feel overwhelmed when they looked at the list. So, we put a lot of effort into making sure that they'd only ever look at a handful of the most important things, but they'd be able to find all their other notes and reminders just when they needed them.

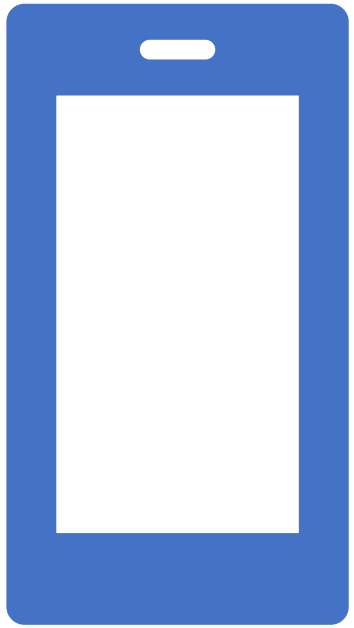
Jürgen Schweizer
Developer
Award winning iPhone to-do list app

A getting things done app needs to be more than just a notepad. It needs to help users feel organized and relaxed. The Things app does this because it has a simple, flexible way of categorizing users' to-do items.


O2-UK
12:19
Lists
Inbox
3
Today
1
13
Next
Scheduled
Someday
Projects
Monday meetings
Even a to-do list has to
satisfy emotional needs.

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

UI and UX in Mobile Applications

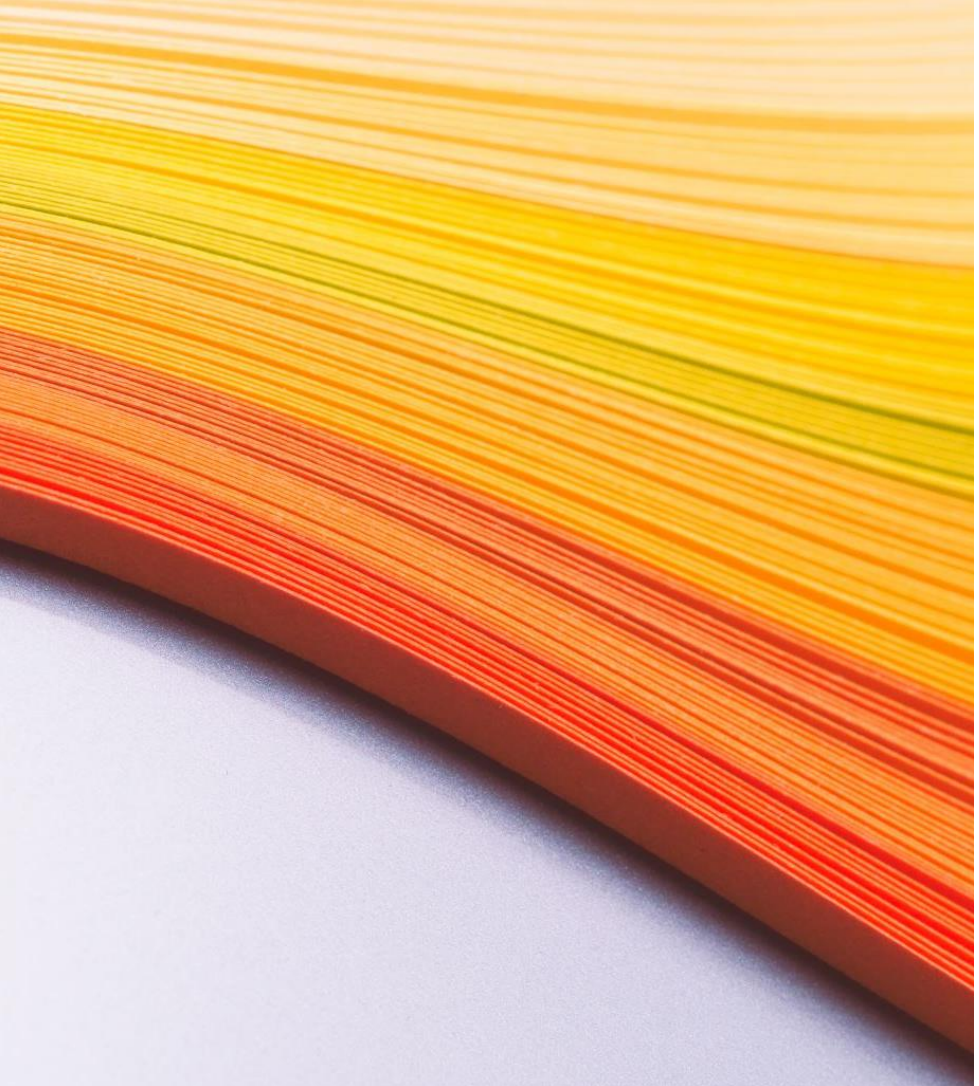


User Interface

- What user can see and interact with
- Its not the appearance, but how it works
- It is the first thing that user see, it directly effects the user's view
- Visual elements greatly impacts an emotional connection with the user

Usage of colors in your app

- The key to picking the right colors for your app is to first understand the basics.
- You can then go on to apply to any UI that you're designing for.
- It's very easy to overdo or pick the wrong colors, which is why you may find yourself spending more time than you anticipated simply figuring out what the color of buttons in your product should be, for example.
- Understanding the basics is the first step to knowing what to do and also knowing what not to do, which is equally important when picking colors.



Consider the following when selecting colors

- UI Hierarchy
- Content Legibility
- Brand Color
- Primary Color
- Secondary Color
- Surface and Background

60-30-10

- Primary Color 60%
  - Background
- Secondary Color 30%
- Accent Color 10%
  - Buttons
  - Pop Ups
  - Highlights

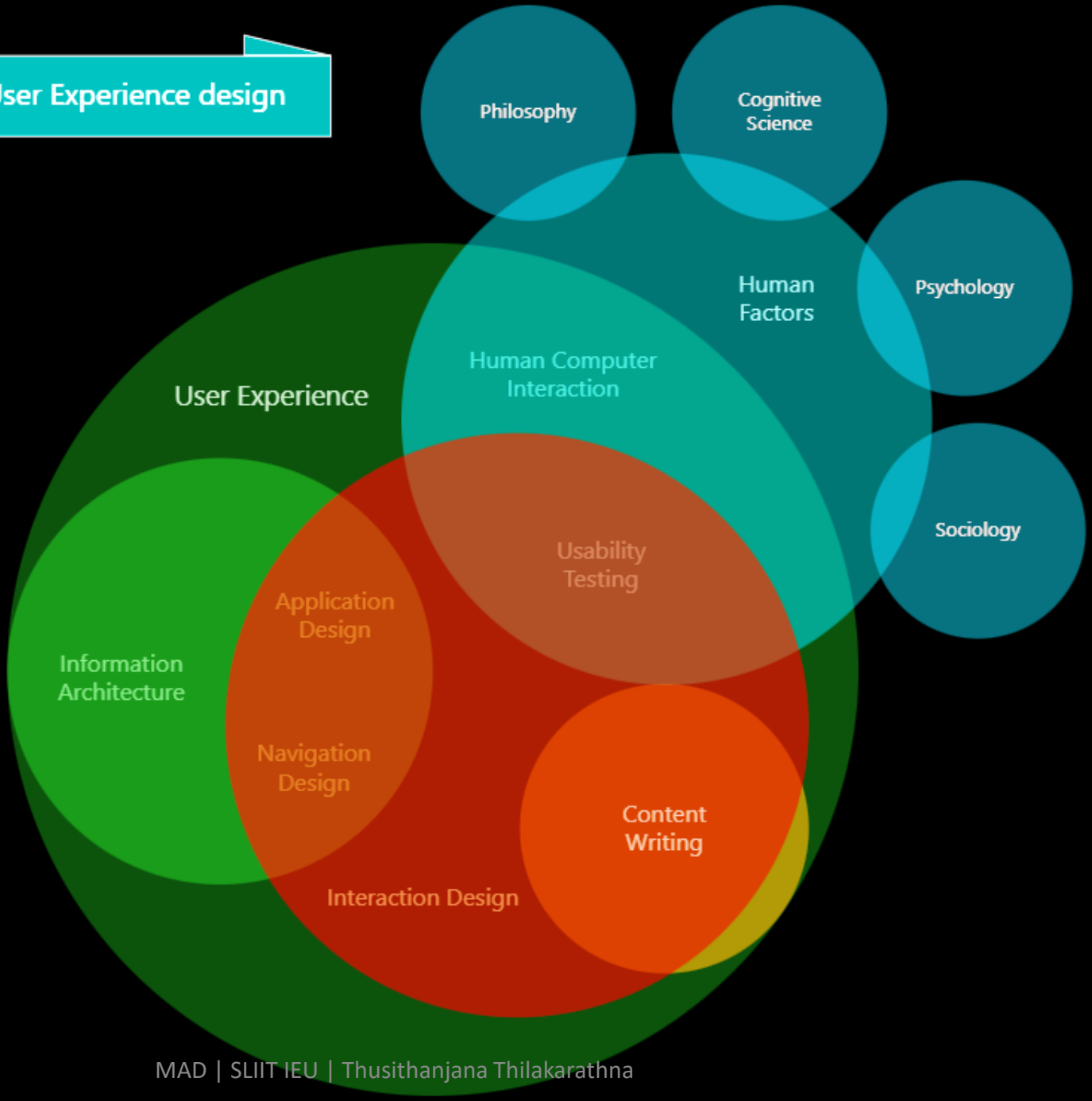
User Experience

- Enhancing user satisfaction of an app, while involving the user's opinions and feelings **before, during, and after** their interaction with an app.
- UX of a mobile application influences how users observe it
- Its about the value addition
- Ease of use
- Help to fulfil user's needs
- Includes all aspects of the end-user's interaction with the company, and its products/services.

Design Thinking

- Read the article on Design Thinking
  - https://www.designorate.com/design-thinking-guide-what-why-how/
- Read more about Double Diamond design thinking process
  - https://www.designorate.com/the-double-diamond-design-thinking-process-and-how-to-use-it/

The disciplines of User Experience design



xml

- XML stands for extensible Markup Language.
- XML was designed to store and transport data.
- XML was designed to be both human- and machine-readable.
- XML does not do anything
- Someone must write a piece of software to send, receive, store, or display it
- Refer to https://www.w3schools.com/xml/default.asp
- Xmlns - The xmlns attribute specifies the xml namespace for a document.

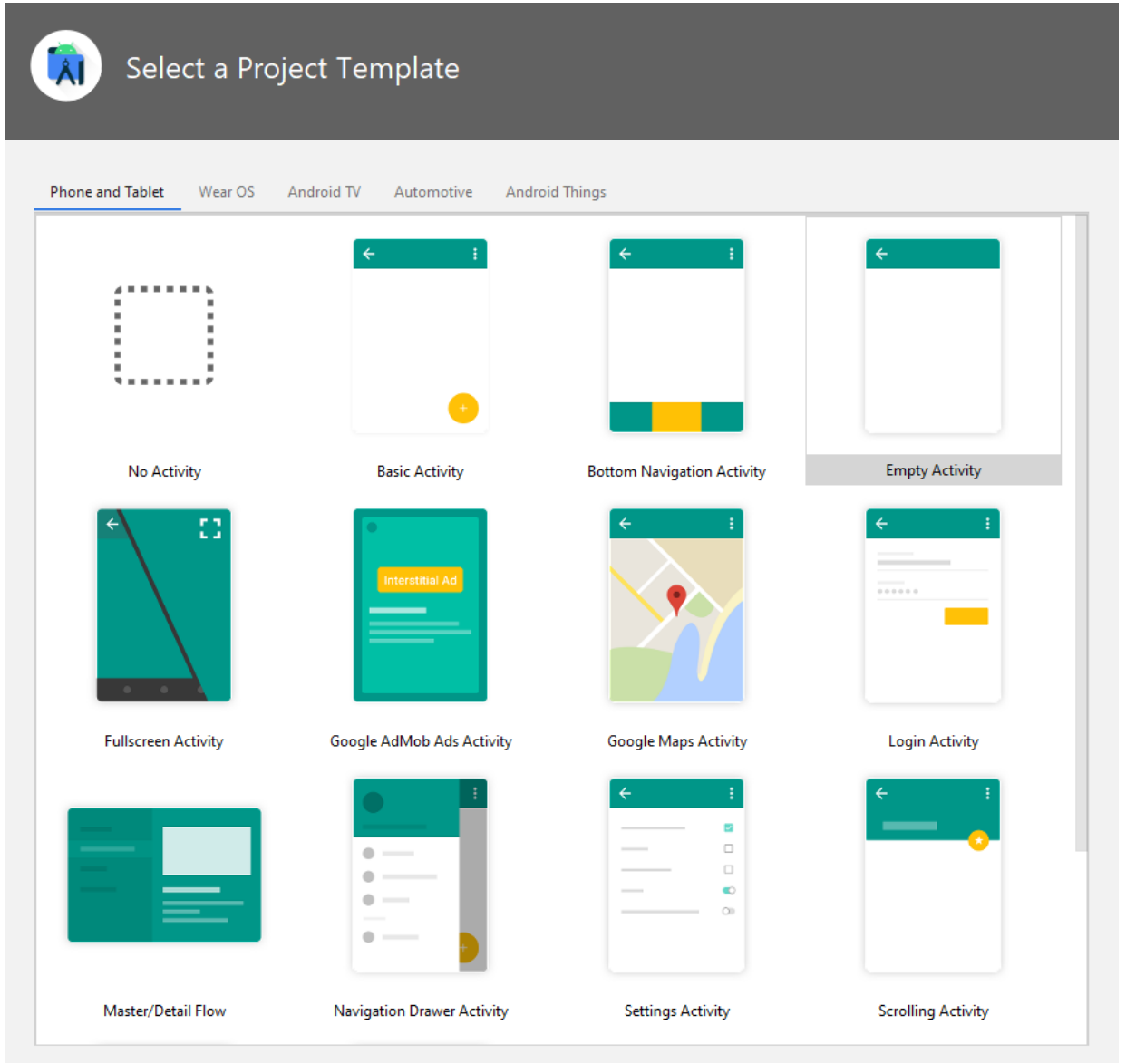
Android Studio

- Android Studio provides the fastest tools for building apps on every type of Android device.
- Intelligent Code Editor
- Flexible Build System
- Realtime Profilers
- Insightful APK Analyzer
- Fast Emulator

Android UI elements

Text	Buttons	Widgets	Layouts	Containers	Helpers	Addons
• Labels • Inputs	• Buttons • Image Buttons • Radio Buttons • Toggle Buttons • Switch • Floating Action Buttons	• View • Web View • Calendar • Progress bar • Rating bar • Dividers • Search View	• Constraint Layout • Linear Layout • Frame Layout • Table Layout	• Spinner • Recycler View • Card View • Toolbar	• Groups • Barriers • Flow	• Google widgets • List views

Android Templates



SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Mobile Application Development

Mobile Platforms and Application Development fundamentals

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Lecture Plan

- Introduction to App Development
- Mobile Platforms and Application Development fundamentals
- Mobile Interface Design Concepts and UI/UX Design
- Introduction to Android Operating System
- Main Components of Android Application
- Sensors and Media Handling in Android Applications
- Data Handling in Android Applications
- Android Application Testing and security aspects

"When the opportunities comes, this is like aligning the stars"

-Andy Rubin-
Co-funder of Android

Learning Outcomes of the Lecture

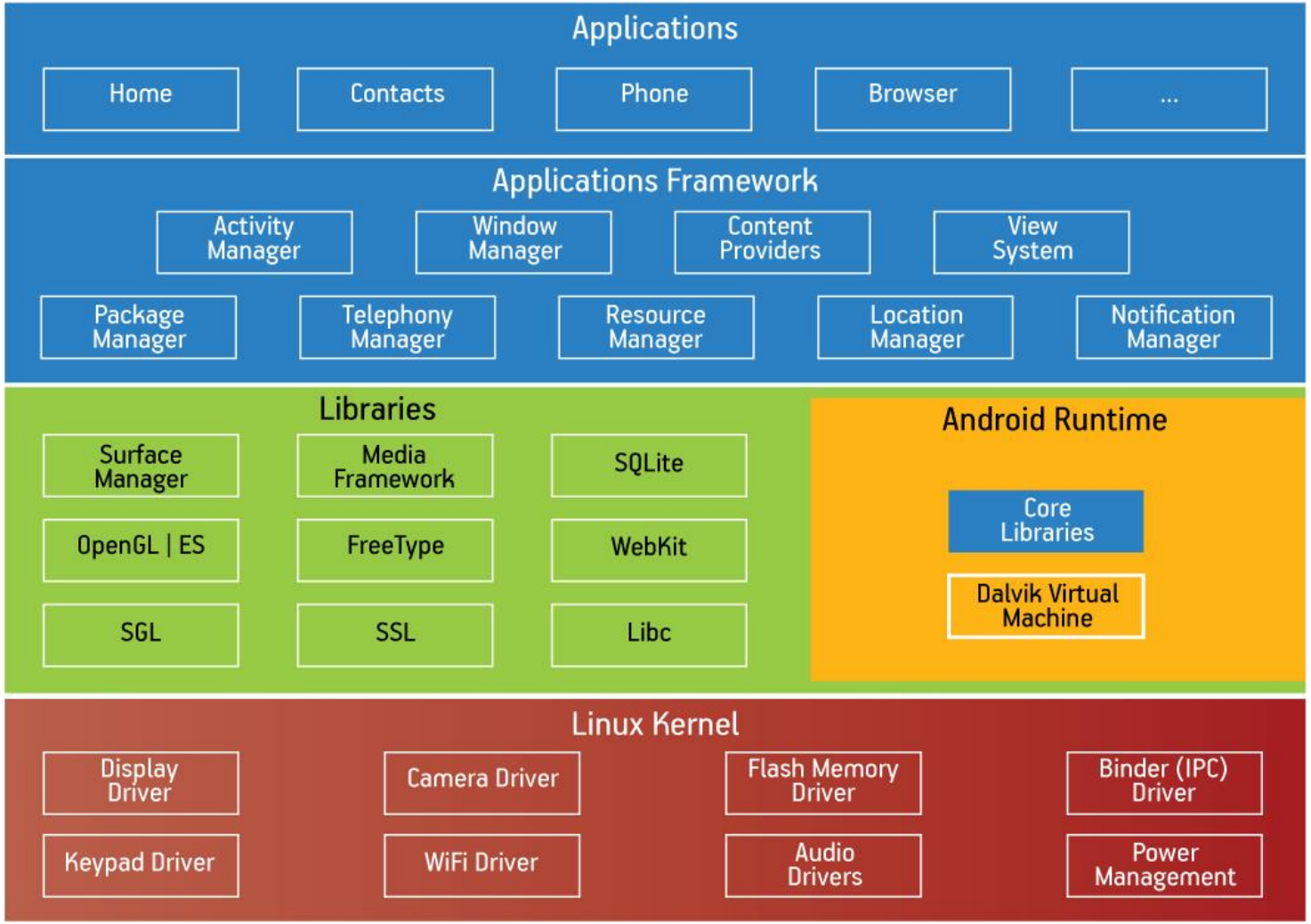
At the end of this Lecture students will be able to:

- Illustrate the architecture of android
- Describe the life cycle of android application
- Recognize the folder hierarchy and components of android application project

Features of Android

- Attractive UI (User Interface)
- Connectivity
- Storage
- Media support
- Messaging
- Web browser
- Multi-tasking
- Multi-touch
- Resizable widgets
- GCM (Google Cloud Messaging)
- Wi-Fi Direct
- Android Beam
- Multi language

Android Platform Architecture



1. Linux Kernel

- This layer provides a level of abstraction between the hardware of the device and contains all the essential hardware drivers, such as the camera, keyboard, screen, etc.
- Kernel handles networks and a wide range of device drivers, which eliminate interference with hardware peripherals.

- Why it's Linux?
  - Portability
  - Security
  - Features

2. Libraries

- This layer operates on top of Linux kernel

- This layer includes,
    - Open source web browser engine Webkit
    - SQLite database
    - Libraries to play and record (video & audio)
    - SSL libraries and etc.

3. Application Framework

- Set of activities that forms the environment in which apps are run and managed.

- This layer provides higher-level services to applications in the form of Java classes. So that they can be reused by other application development process.

  Key services;
  - Activity Manager
  - Content Providers
  - Resource Manager
  - Notifications Manager
  - View System

4. Applications

- This layer contains, native apps provided with the OS and the third-party apps installed by the users will get installed here.

Market store for android apps

- Google Play
- SlideME
- Opera Mobile Store
- Mobango
- F-droid A
- mazon Appstore

Android Runtime

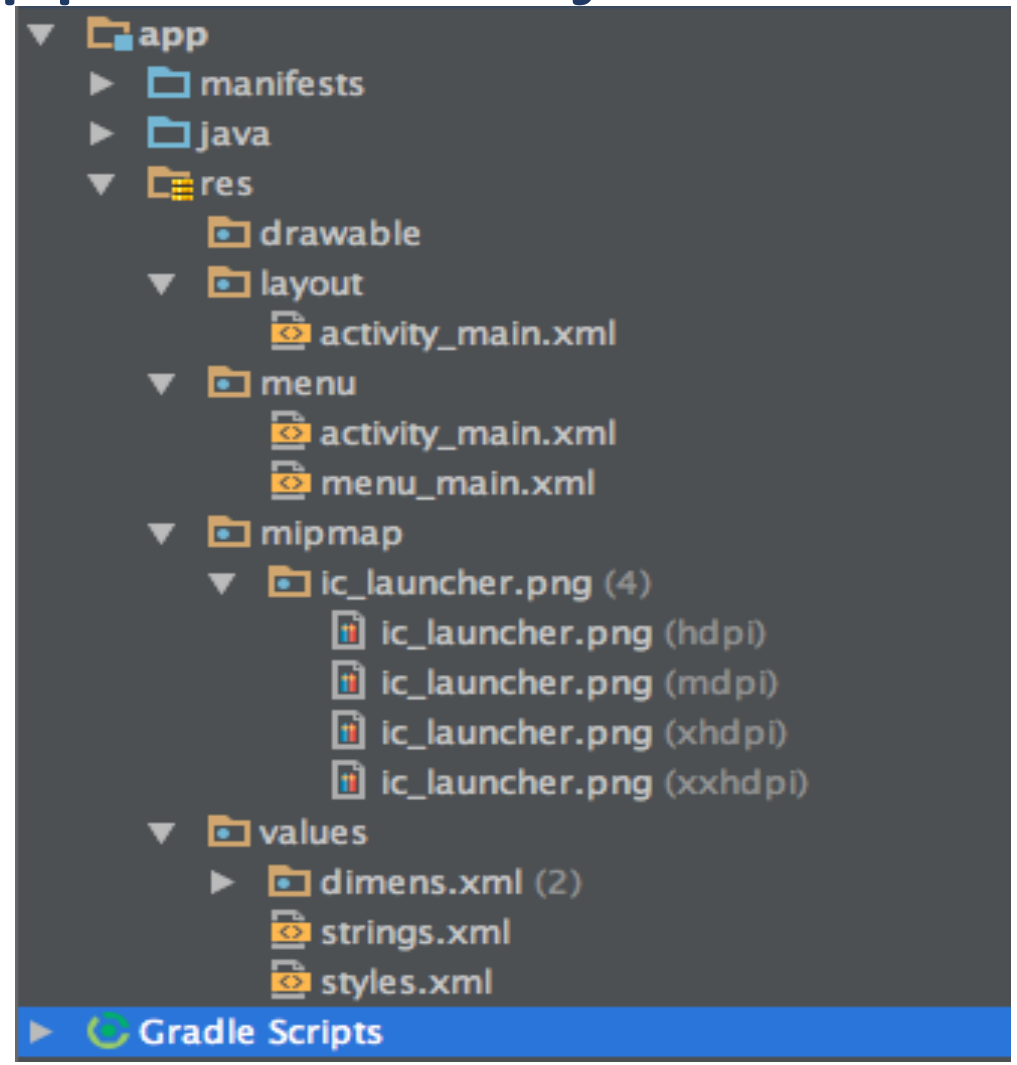
This is a section of second layer. Consists of,

1. **Core Libraries** –
   - These libraries enable developers to develop android applications using Java programming language

Cont'd…

2. **Dalvik Virtual Machine** –
   - Kind of Java Virtual Machine specially designed and optimized for Android
   - Makes use of Linux core features like memory management and multi-threading, which is fundamental in the Java language
   - Enables the application to run in its own process, with its own instance

Android Application Project Structure

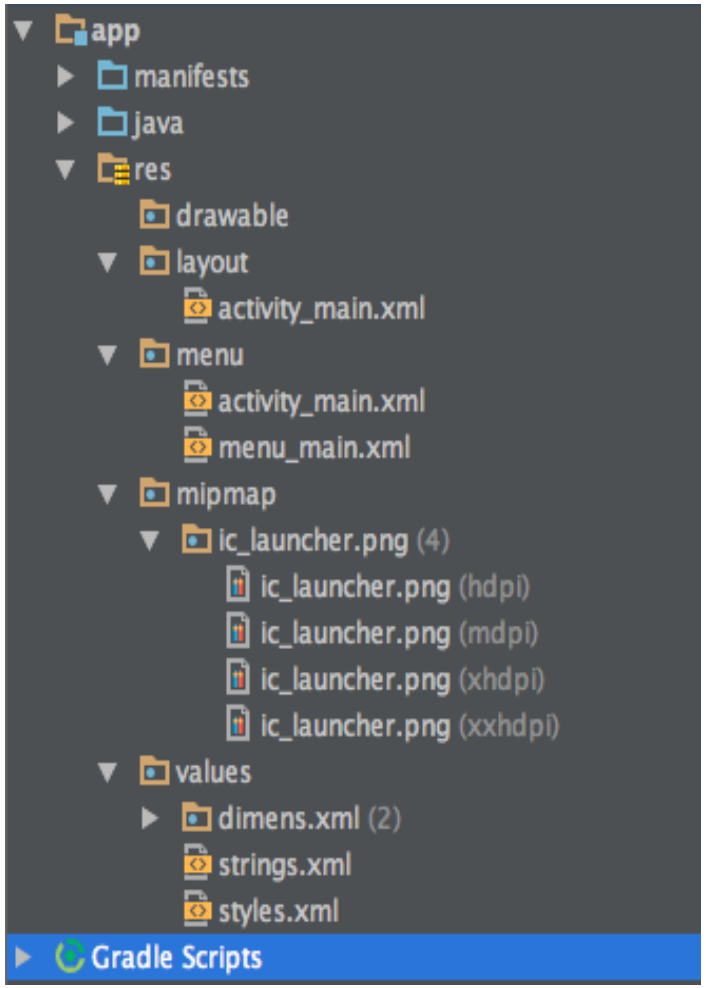


Project File Structure

(based on Android Studio)

- Once a project is created in Android Studio, the project view will content all the project files (as shown in the the image)

- Here, the files are organized into directories



Cont'd…

Some important directories are,

- **src** - Contains all source files (code) and resource files in subdirectories such as,
  - androidTest
  - **Main**
  - build.gradle (module)

- **gradle (project) -** This defines your build configuration that apply to all modules.

Cont'd… (src/main)

main directory contain subdirectories within it,

- **java** - contains Java code sources
- **AndroidManifest.xml** - Describes the nature of the application and each of its components.
- **res** - Contains all non-code resources
  - The XML files here can be divided into corresponding sub-directories
    - **drawable** –
      consists of Bitmap files or XML files
      Ex:
        - bitmap files,
        - shapes,
        - animation drawables
        - other drawable

Cont'd…

layout –
XML files that define a user interface layout

menu – XML files that define app menus (context menu, options menu)

mipmap – Drawable files for different launcher icon densities

values – XML files that contain simple values such as, string, style, color

R.Java file

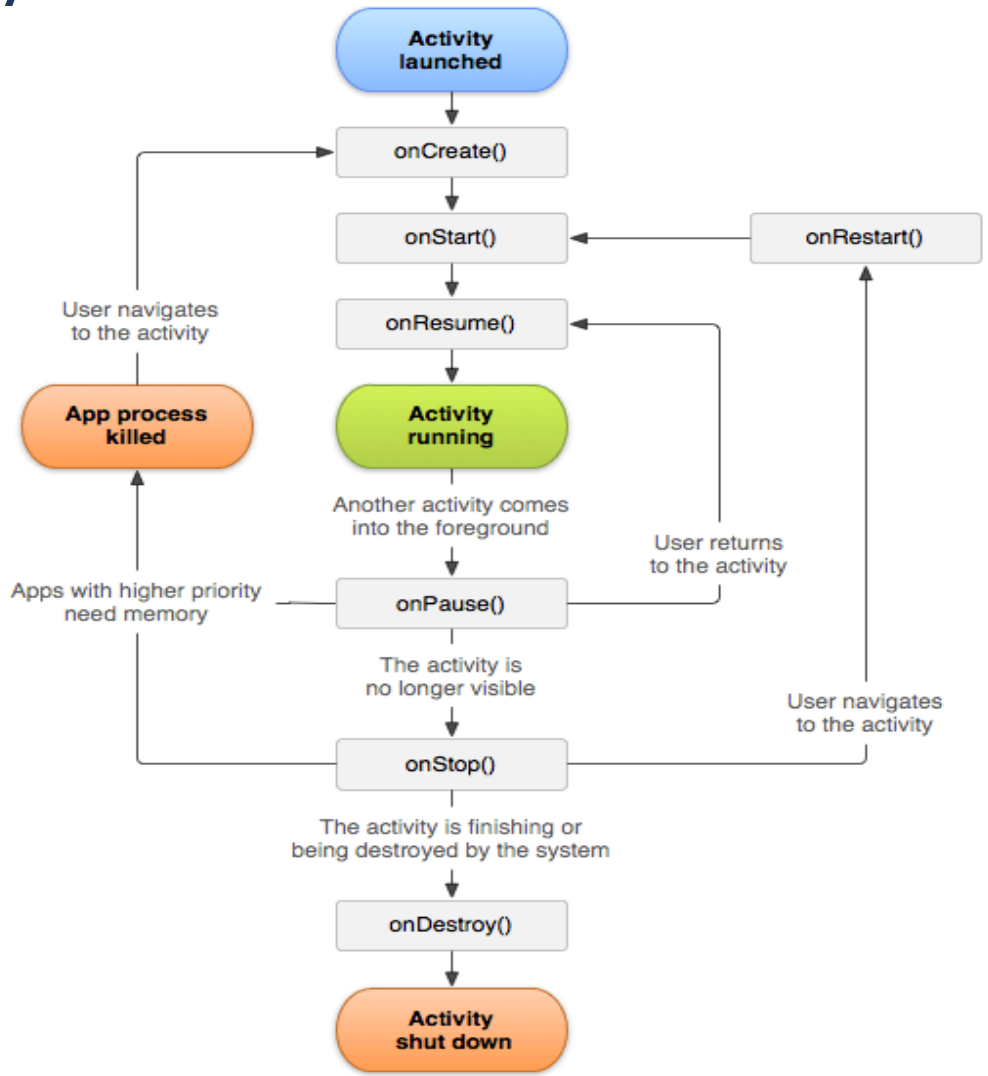
- Resource file that contains resource IDs for all the resources of res/ directory.
- This is an abstraction between different resources (XML file, any UI component [icon], audio & etc. and the java file)
- Auto generated file by AAPT (Android Asset Packaging Tool).

Activity & Activity Life Cycle

Activity

- An activity is like a frame or window in java that represents GUI
- It represents one screen of android
- They perform actions on the screen

Activity Life Cycle



Reference: https://www.javatpoint.com/images/androidimages/Android-Activity-Lifecycle.png

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Cont'd…

- **onCreate():** called when activity is first created
- **onStart():** called when activity is becoming visible to the user
- **onResume():**called when activity will start interacting with the user
- **onPause():** called when activity is not visible to the user
- **onStop():** called when activity is no longer visible to the user
- **onRestart():** called after your activity is stopped, prior to start
- **onDestroy():** called before the activity is destroyed

References

1. https://developer.android.com/
2. https://www.tutorialspoint.com/
3. https://www.javatpoint.com

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Summary

1. Overview to Android system
2. Android architecture & Android application architecture
3. Mobile App development life cycle
4. Android app development life cycle
5. Android app project structure
6. Activity & Activity life cycle

Thank You!!!

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Mobile Application Development

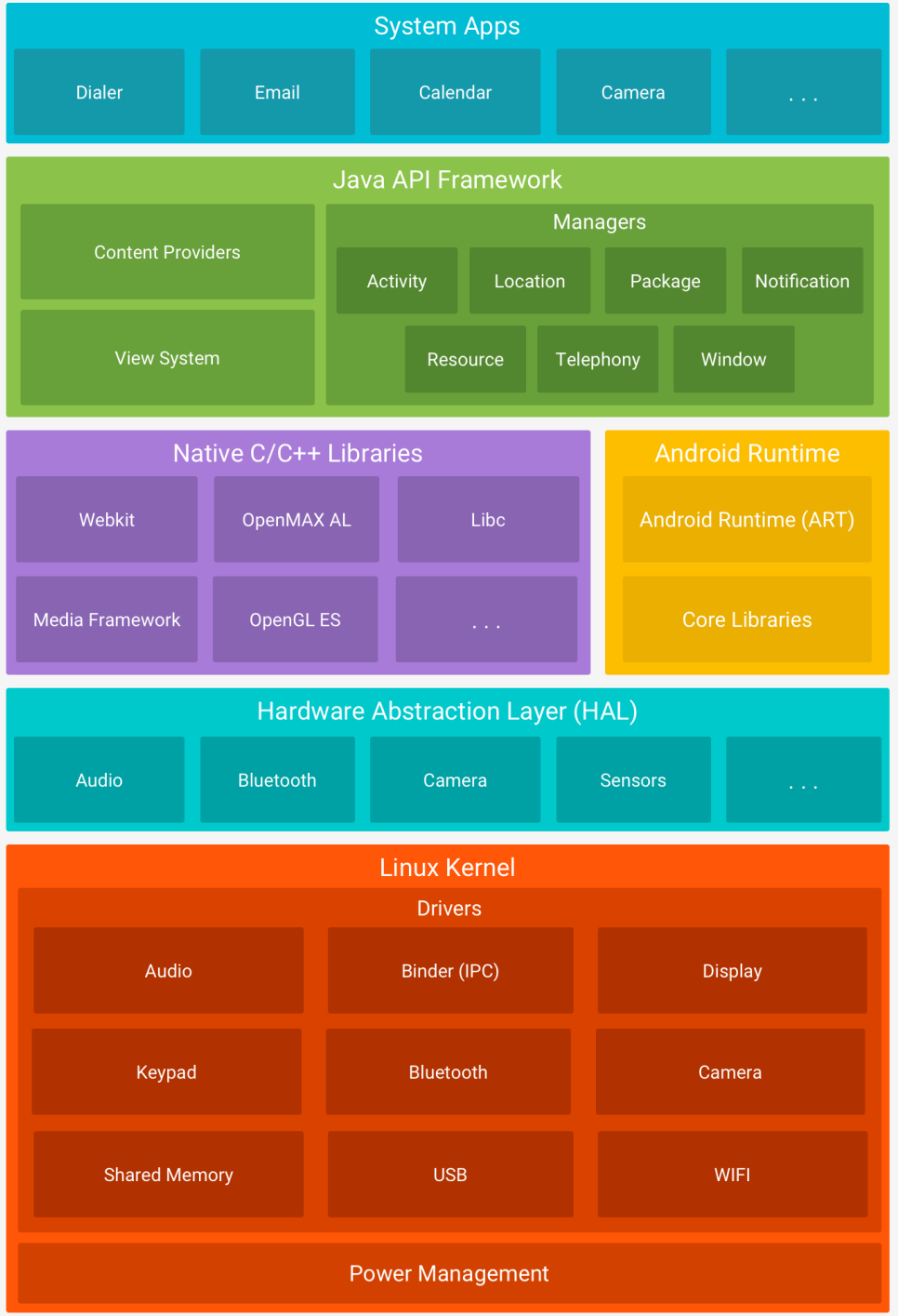
Mobile Platforms and Application Development fundamentals

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Lecture Plan

- Introduction to App Development
- Mobile Platforms and Application Development fundamentals
- Mobile Interface Design Concepts and UI/UX Design
- Introduction to Android Operating System
- Main Components of Android Application
- Sensors and Media Handling in Android Applications
- Data Handling in Android Applications
- Android Application Testing and security aspects

Android Platform Architecture



Difference between DVM and ART

- Dalvik Virtual Machine (DVM) was the original virtual machine used by Android before Android 5.0 Lollipop. It was designed specifically for the mobile environment and optimized for low-memory devices. DVM used a just-in-time (JIT) compiler to translate bytecode to machine code at runtime, which allowed for faster performance compared to other virtual machines at the time.
- Android Runtime (ART) was introduced with Android 5.0 Lollipop as a replacement for Dalvik. ART uses Ahead-of-Time (AOT) compilation, which means that code is compiled to native machine code during installation rather than at runtime. This approach provides faster startup times and improved overall performance, but it requires more storage space to store the compiled code.

Key differences between Dalvik and ART

- Compilation method: Dalvik uses JIT compilation, while ART uses AOT compilation.
- Performance: ART is generally faster than Dalvik because it compiles code ahead of time, leading to faster startup times and improved overall performance.
- Storage requirements: ART requires more storage space than Dalvik because it compiles code to native machine code during installation, while Dalvik compiles code at runtime.
- Compatibility: While both virtual machines can run the same Android apps, some apps may not work properly on ART if they are not compatible with the AOT compilation process.
- Read more: https://developer.android.com/guide/platform

Android Core Building Blocks

- Activities: Activities represent a single screen with a user interface. They are responsible for interacting with the user and managing the app's UI.
- Services: Services are components that run in the background and perform long-running operations. They are used to perform tasks that do not require user interaction, such as downloading files or playing music.
- Broadcast Receivers: Broadcast Receivers are components that respond to system-wide broadcast messages. They allow an app to receive and respond to events, such as a low battery warning or a network connection change.
- Content Providers: Content Providers are components that manage a shared set of app data that can be accessed by other apps. They allow an app to share data with other apps, such as contacts, images, or videos.

Activity Manager

- This class gives information about, and interacts with, activities, services, and the containing process.
- Several of the methods in this class are for informational or debugging reasons only, and your app's runtime behavior should not be altered by using them.
- Most application developers should not have the need to use this class, most of whose methods are for specialized use cases.
- However, a few methods are more broadly applicable.
  - For instance, isLowRamDevice() enables your app to detect whether it is running on a low-memory device and behave accordingly.
  - clearApplicationUserData() is for apps with reset-data functionality.

Activity

Activity is a core component of the Android operating system that provides a single, focused thing that a user can do.

Each activity represents a screen in the app's user interface and is responsible for managing user interactions, displaying UI elements, and responding to events such as button clicks or screen rotations.

Activities are created and managed by the Android framework and can be launched by other activities, system components, or apps.

To create an activity, you need to define a subclass of the Activity class and implement its lifecycle methods, such as onCreate(), onPause(), and onDestroy().

The lifecycle of an activity is an important concept to understand, as it determines how the activity behaves in response to various system events, such as when the user switches to another app or rotates the screen.

Activities can also be configured to work with other Android components, such as fragments, services, and broadcast receivers, to create more complex and flexible apps.

Activity Initial code

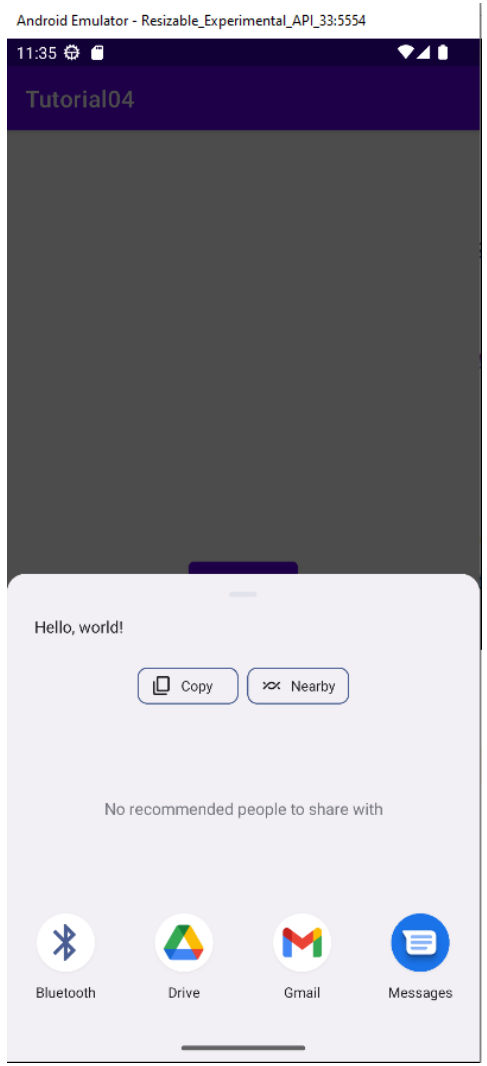
```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

}
```

UI will be loaded to the screen from this method

Intents

- An Intent is an object that can be used to request an action from another app component.
- It is a messaging system used to communicate between Android components such as activities, services, broadcast receivers, and content providers.
- The Intent provides a way to start an activity, send a broadcast, or deliver a message to another app component.
- It can also carry data between components using extras, which are key-value pairs.
- Intents can be explicit or implicit. An explicit intent specifies the component to be invoked by name, while an implicit intent describes the type of action to be performed and leaves the system to find an appropriate component to handle it.
- Android Intent is a powerful and flexible mechanism that enables communication between different app components and facilitates app integration.

Implicit intent



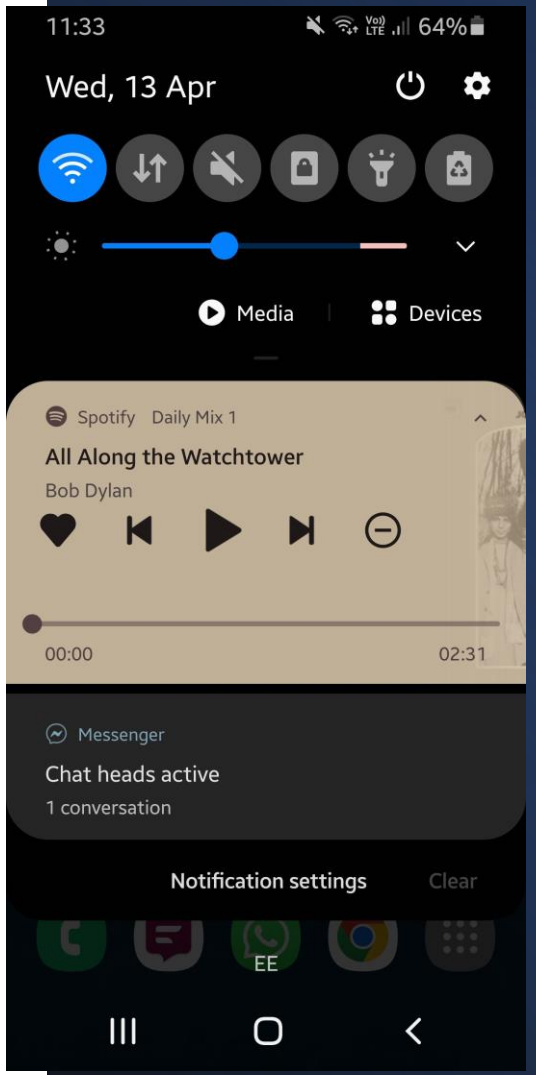

```
button.setOnClickListener { it: View!
    val intent = Intent(Intent.ACTION_SEND)
    intent.type = "text/plain"
    intent.putExtra(Intent.EXTRA_TEXT, value: "Hello, world!")
    startActivity(Intent.createChooser(intent, title: "Share via"))

}
```

Explicit intent

```
btnBack.setOnClickListener {
    val intent = Intent(this, MainActivity::class.java)
    startActivity(intent)
    finish()
}
```

Android Services

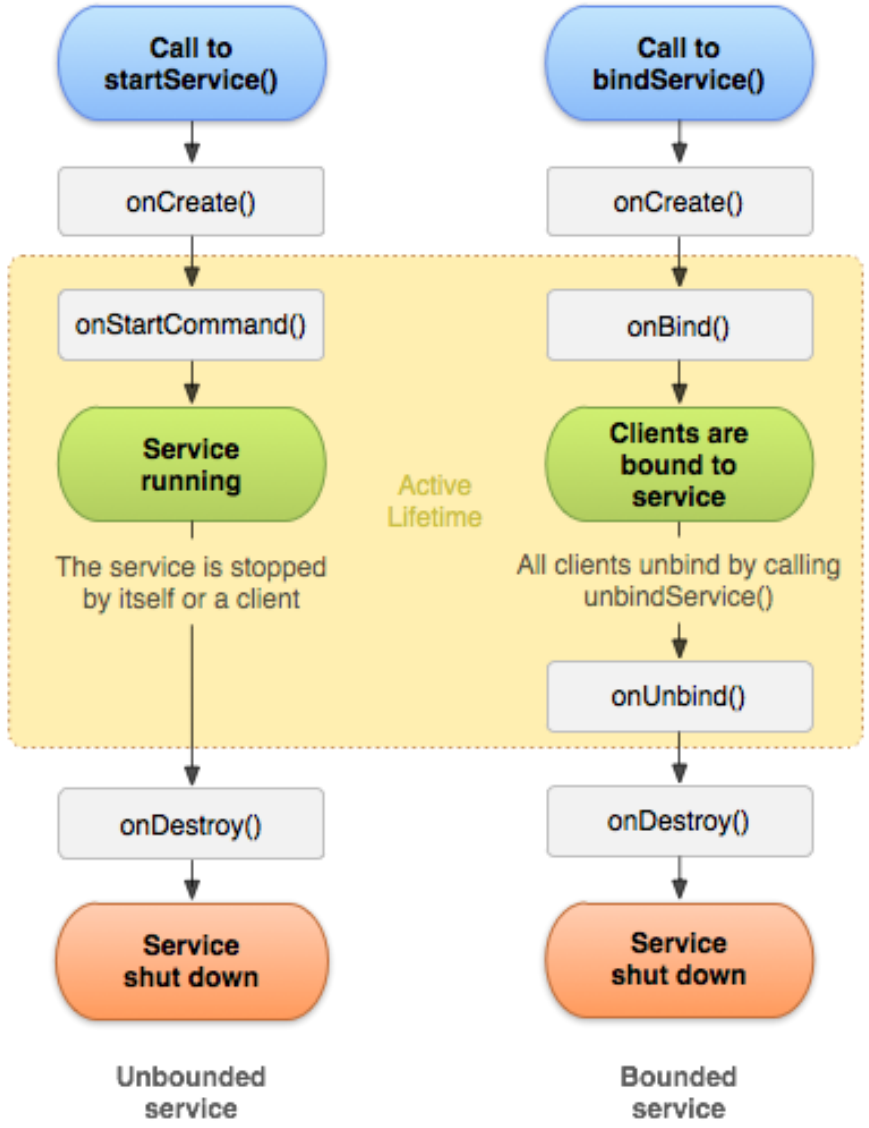
- Android Services are a fundamental component of the Android operating system.
- They enable developers to create long-running background tasks that can perform operations even when an app is not in the foreground.
- This enables developers to build applications that can perform complex tasks in the background without interrupting the user experience.



Types of Android Services

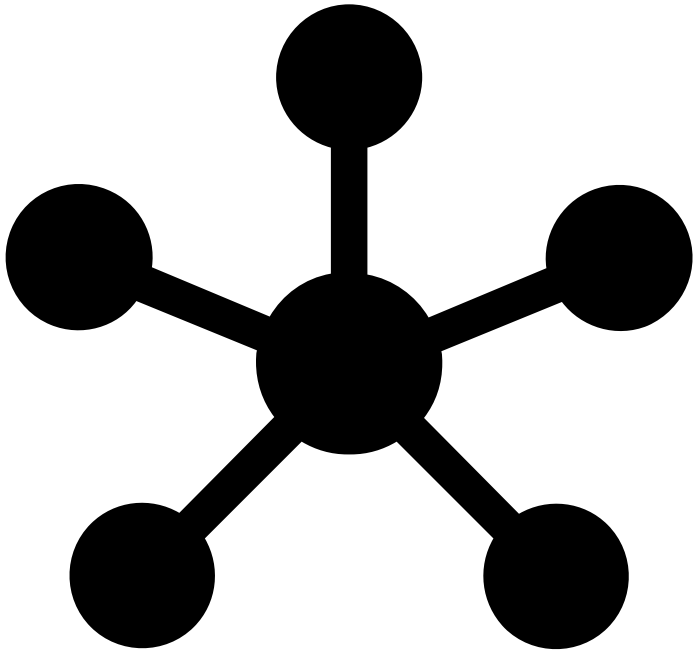
- There are two types of Android Services: Started Services and Bound Services.
- Started Services: These services are started by calling startService() method and can run indefinitely in the background, even after the app is closed.
- Bound Services: These services are bound to a client component (such as an Activity or a Fragment) by calling bindService() method, and they provide a client-server interface for communication between components.

Service Lifecycle



Service Communication

- Services can communicate with other components of an app using IPC mechanisms such as intents.
- A service can send an intent to a broadcast receiver to notify other components of an app of a change in state.
- A service can also communicate with a client component (such as an Activity or a Fragment) using a Messenger or a Binder.

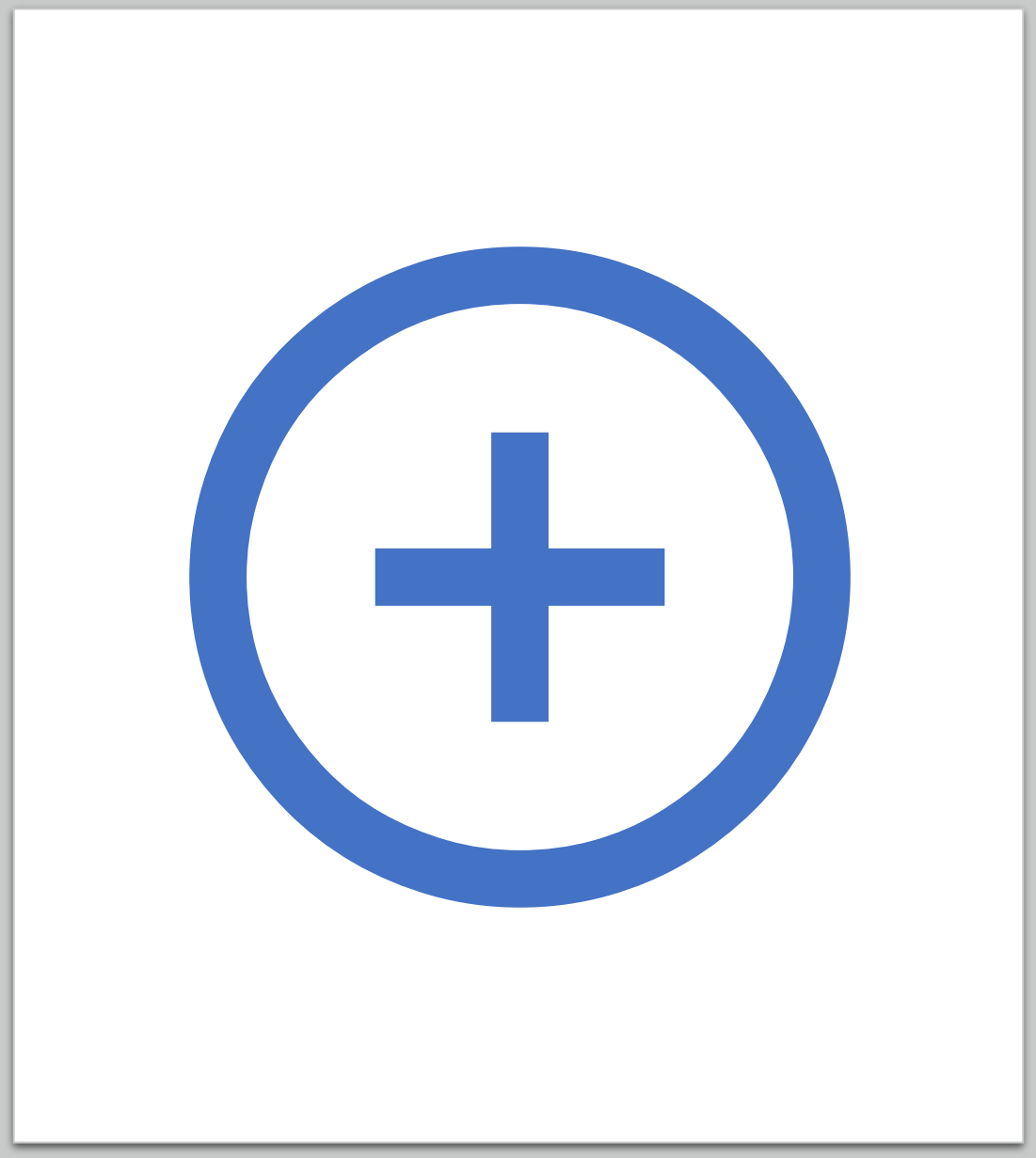


Best practices on using services

- It's critical to use services wisely to guarantee top performance and efficiency.
- Services should be designed to use as few resources as possible and to release those resources when they are no longer needed.
- It is also important to ensure that services do not drain the device's battery by running indefinitely in the background.

Broadcast Receivers

- Android Broadcast Receivers are a fundamental component of the Android operating system that enable apps to receive and respond to system-wide or app-specific broadcast messages.
- Broadcast messages are sent by the system or other apps, and they can be used to trigger specific actions or events within an app.



Types of Broadcast Receivers

- There are two types of Broadcast Receivers: Static and Dynamic.
- Static Broadcast Receivers are declared in the app's manifest file and are used to listen for system-wide broadcast messages.
- Dynamic Broadcast Receivers are registered programmatically in an app's code and are used to listen for app-specific broadcast messages.

Broadcast Receiver Lifecycle

- Broadcast Receivers have a lifecycle that consists of several callback methods, such as onReceive().
- The onReceive() method is called when a broadcast message is received by the receiver.
- The receiver can then process the message and perform any necessary actions or trigger any necessary events.

Registering Broadcast Receivers

Broadcast Receivers can be registered statically in the app's manifest file, or dynamically in the app's code using the registerReceiver() method.

When registering a Broadcast Receiver, developers must specify the broadcast message they wish to listen for using an IntentFilter.

Best Practices on using Broadcast Receivers

- To ensure optimal performance and efficiency, it is important to use Broadcast Receivers judiciously.
- Broadcast Receivers should be designed to use as few resources as possible and to release those resources when they are no longer needed.
- It is also important to ensure that Broadcast Receivers do not drain the device's battery by running indefinitely in the background.

Examples for Broadcast Receivers

Battery Level Receiver

- This receiver can be used to detect when the device's battery level changes.
- When the battery level changes, the receiver can perform specific actions, such as displaying a notification or changing the behavior of the app.

Network Connectivity Receiver

- This receiver can be used to detect changes in network connectivity.
- When the device connects to a network or loses network connectivity, the receiver can perform specific actions, such as fetching data from a server or displaying an error message to the user.

SMS Receiver

- This receiver can be used to detect when the device receives an SMS message.
- When an SMS message is received, the receiver can perform specific actions, such as displaying a notification or triggering a specific event within the app.

Screen State Receiver

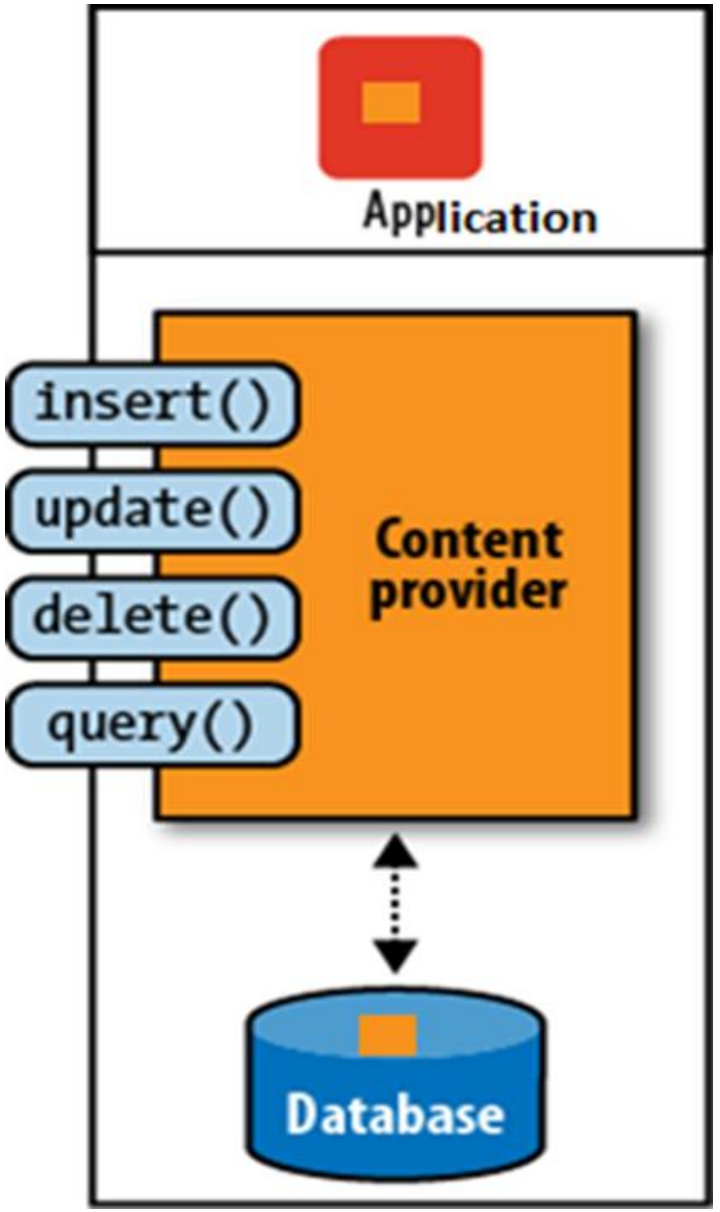
- This receiver can be used to detect changes in the device's screen state, such as when the screen is turned on or off.
- When the screen state changes, the receiver can perform specific actions, such as starting or stopping a service or changing the behavior of the app.

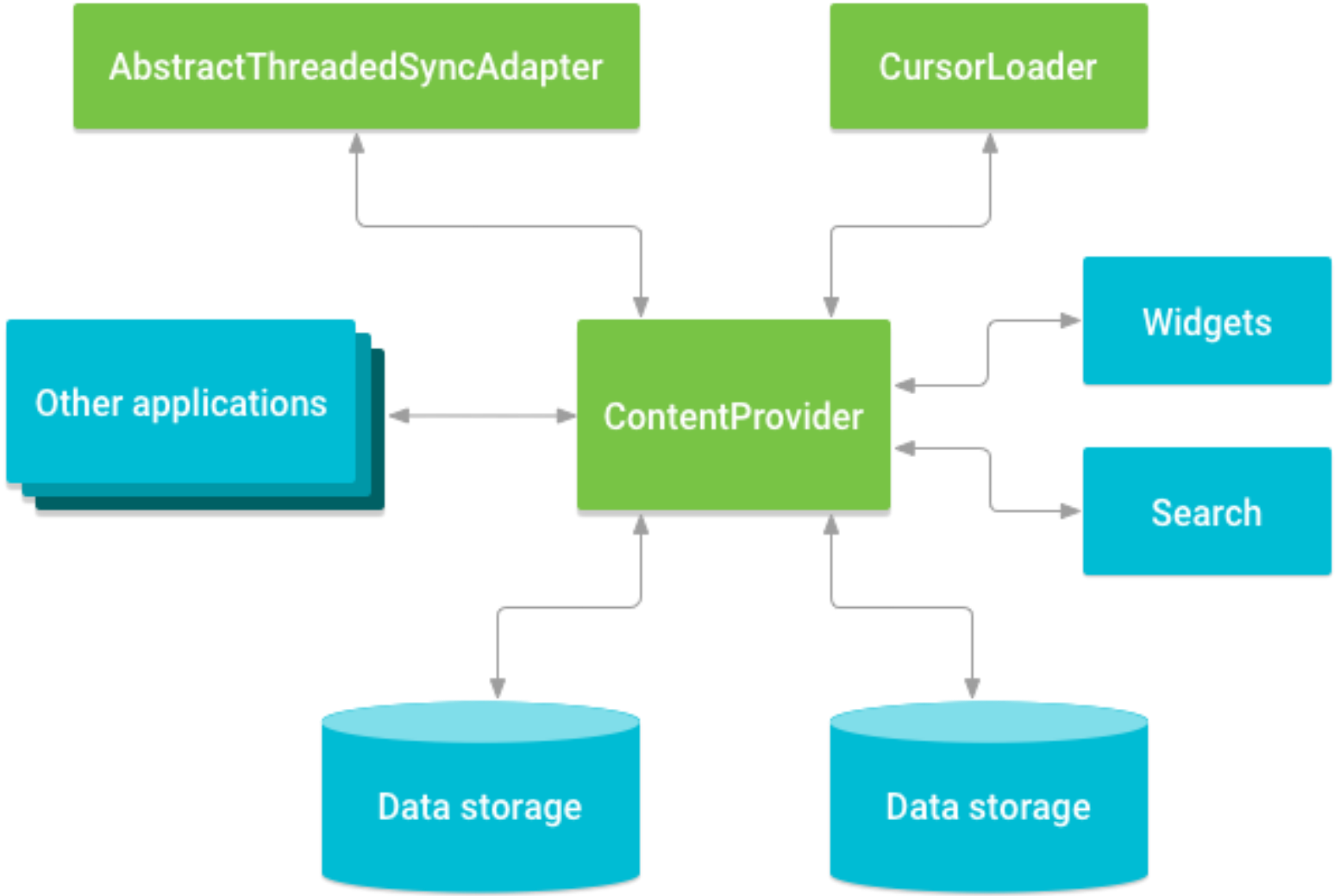
Power State Receiver

- This receiver can be used to detect changes in the device's power state, such as when the device is plugged in or unplugged.
- When the power state changes, the receiver can perform specific actions, such as starting or stopping a service or changing the behavior of the app.

Content Providers

- Android Content Providers are a fundamental component of the Android operating system that allow apps to securely share data with other apps.
- Content Providers enable apps to expose a structured set of data to other apps and provide a mechanism for data storage and retrieval.
- There are two types of Content Providers: built-in and custom.
- Built-in Content Providers are part of the Android operating system and provide access to system-wide data, such as contacts, media, and settings.
- Custom Content Providers are created by app developers and provide access to app-specific data, such as user preferences and app settings.




AbstractThreadedSyncAdapter
CursorLoader
Other applications
ContentProvider
Widgets
Search
Data storage
Data storage

Content Provider Components

Content Providers consist of several key components, including a content provider class, a data contract, and a set of URIs.

The content provider class implements the methods for accessing and modifying data, while the data contract defines the structure of the data and its relationships.

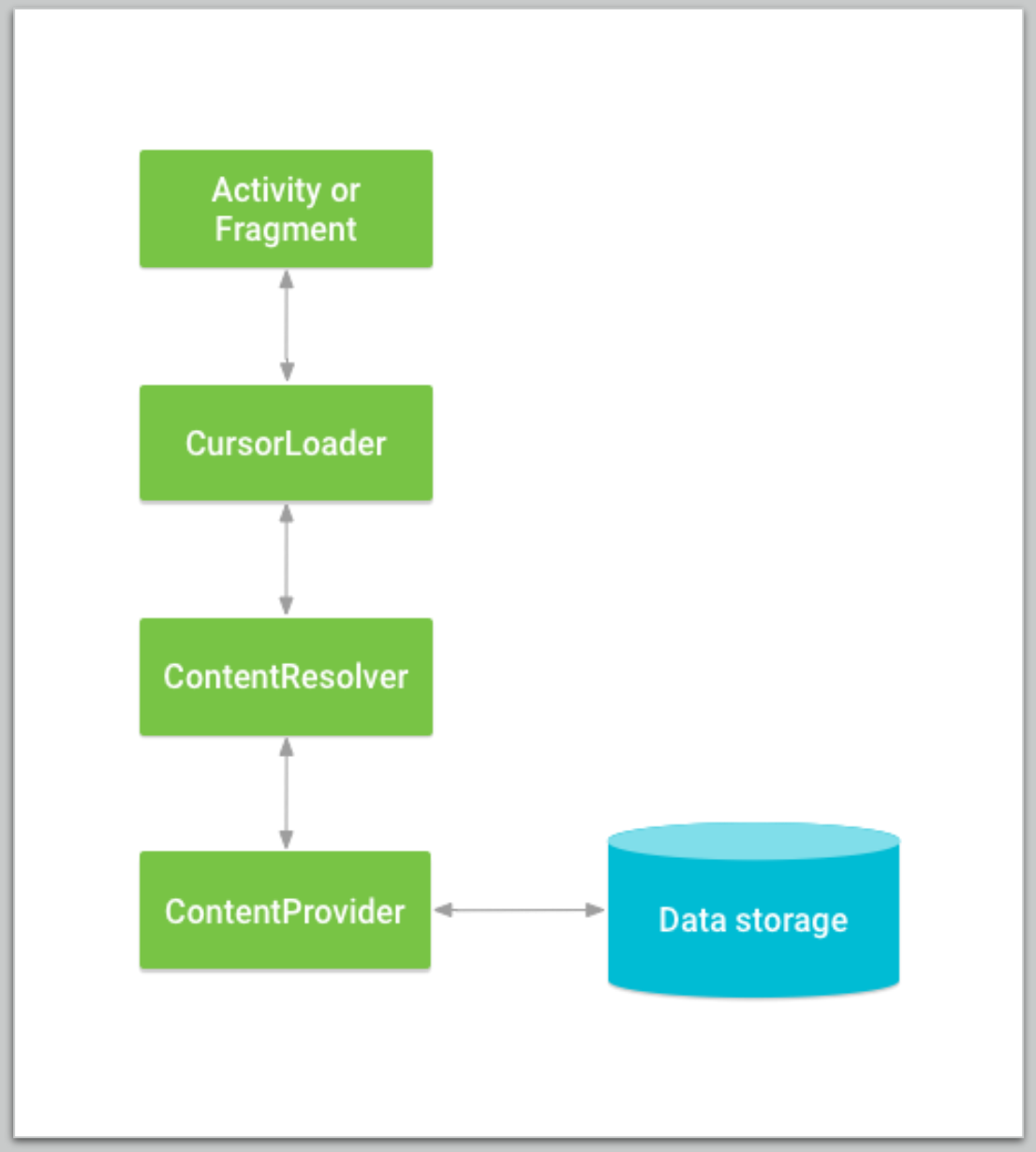
The set of URIs specifies the location of the data and how it can be accessed.

Content Provider Permissions

- Content Providers require permissions to access and modify data, and these permissions must be declared in the app's manifest file.
- Permissions can be set at the level of individual data items or at the level of the entire content provider.
- App developers must ensure that the permissions they grant to Content Providers are appropriate and do not compromise user privacy or security.

Using Content Providers

- Other apps can access Content Providers by sending requests using a ContentResolver object.
- The ContentResolver object sends requests to the Content Provider's methods, which perform the requested operations on the data.
- Developers can use Content Providers to enable sharing of data between different components of their own app or between different apps.



Examples of Content Providers

- Contacts Provider
- Media Store Provider
- Settings Provider
- Calendar Provider
- Call Log Provider
- User Dictionary Provider

Thank you!

- References
  - https://developer.android.com/guide/components/activities/intro-activities
  - https://developer.android.com/guide/components/services
  - https://developer.android.com/reference/android/content/BroadcastReceiver
  - https://developer.android.com/guide/topics/providers/content-provider-basics

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Mobile Application Development

Mobile Platforms and Application Development fundamentals

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Lecture Plan

- Introduction to App Development
- Mobile Platforms and Application Development fundamentals
- Mobile Interface Design Concepts and UI/UX Design
- Introduction to Android Operating System
- Main Components of Android Application
- Data Handling in Android Applications
- Sensors and Media Handling in Android Applications
- Android Application Testing and security aspects

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Persistance techniques in Android Applications

- Data persistence refers to the ability of an app to store data locally on a device even after the app is closed or the device is turned off.
- The choice of data persistence technique depends on the type and amount of data that needs to be stored, as well as the level of security and accessibility required for that data.
- There are several techniques available for data persistence in Android applications

Different techniques for data persistence in Android applications

- Shared Preferences
- Internal Storage
- External Storage
- SQLite Database
- Content Providers
- Cloud Storage

Shared Preference

- Shared Preferences are used to store primitive data types, such as Boolean, float, int, long, and string.
- This technique is suitable for small amounts of data and is used to store user preferences or settings.
- The key-value pairs are written to XML files that persist across user sessions, even if your app is killed. You can manually specify a name for the file or use per-activity files to save your data.

```
fun save(isLogged:Boolean, userName:String){
    val sharedPref = this
        .getSharedPreferences( name: "MyPrefs",Context.MODE_PRIVATE)
    val editor = sharedPref?.edit()
    editor?.putBoolean("isUserLoggedIn", isLogged)
    editor?.putString("username", userName)
    editor?.apply()
}
fun getUser(){
    val sharedPref = this
        .getSharedPreferences( name: "MyPrefs", Context.MODE_PRIVATE)
    val isUserLoggedIn = sharedPref?.getBoolean("isUserLoggedIn", false) ?: false
    val username = sharedPref?.getString("username", "")
}
```

Internal Storage

- Internal storage is used to store private data in the device's memory.
- This technique is suitable for storing sensitive data that should not be accessible to other apps.
- The system provides a private directory on the file system for each app.
- When the user uninstalls your app, the files saved on the internal storage are removed.

```
fun writeToFile(){
    val filename = "myfile.txt"
    val fileContents = "Hello world!"

    this.openFileOutput(filename, Context.MODE_PRIVATE).use { it: FileOutputStream!
        it?.write(fileContents.toByteArray())
    }
}

fun readFromFile(){
    val filename = "myfile.txt"
    this.openFileInput(filename).use { it: FileInputStream!
        val fileContents = it?.bufferedReader().use { it: BufferedReader?
            it?.readText()
        }
        Log.d( tag: "MainActivity",  msg: "File contents: $fileContents") ^use
    }
}
```

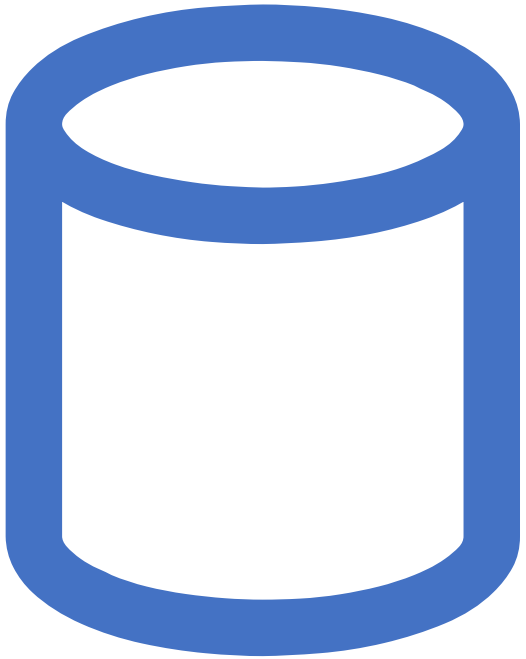
External Storage

- External storage is used to store data in public directories that can be accessed by other apps or the user.
- This technique is suitable for storing large files or data that can be shared between different apps.

```
vate fun isExternalStorageWritable(): Boolear
 val state = Environment.getExternalStorageSt
 return Environment.MEDIA_MOUNTED == state
```

```
vate fun isExternalStorageReadable(): Boolear
 val state = Environment.getExternalStorageSt
 return Environment.MEDIA_MOUNTED == state ||
```

```
    if (isExternalStorageWritable()) {
        val file = File(getExternalFilesDir( type: null), filename)
        FileOutputStream(file).use { it: FileOutputStream
            it.write(fileContents.toByteArray())
        }
    }

}

fun readFromExternalFile(){
    val filename = "myfile.txt"

    if (isExternalStorageReadable()) {
        val file = File(getExternalFilesDir( type: null), filename)
        FileInputStream(file).use { it: FileInputStream
            val fileContents = it.bufferedReader().use { it: BufferedReader
                it.readText()
            }
```

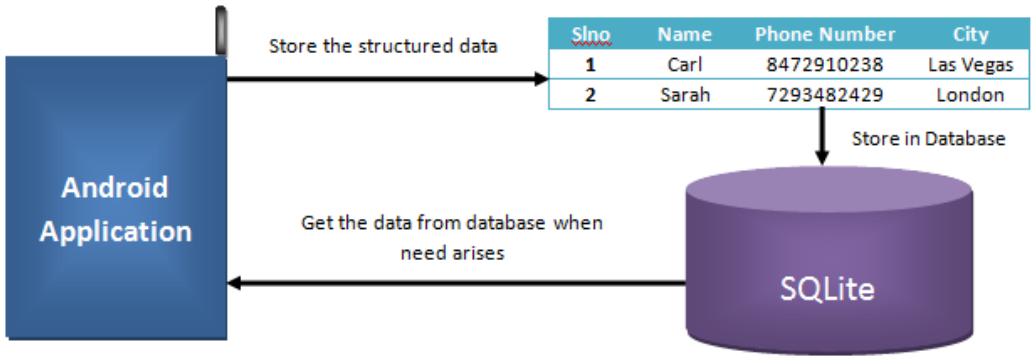


SQLite Databases

- SQLite is a lightweight database that can be used to store structured data.
- This technique is suitable for storing large amounts of structured data that can be queried and sorted.
- The Android SDK includes a sqlite3 shell tool that allows you to browse table contents, run SQL commands, and perform other useful functions on SQLite databases.

SQLite

- A SQL database engine which is developed using C that accesses its storage files directly.
- A software library that implements a self-contained, server less, zero-configuration, transactional SQL database engine.

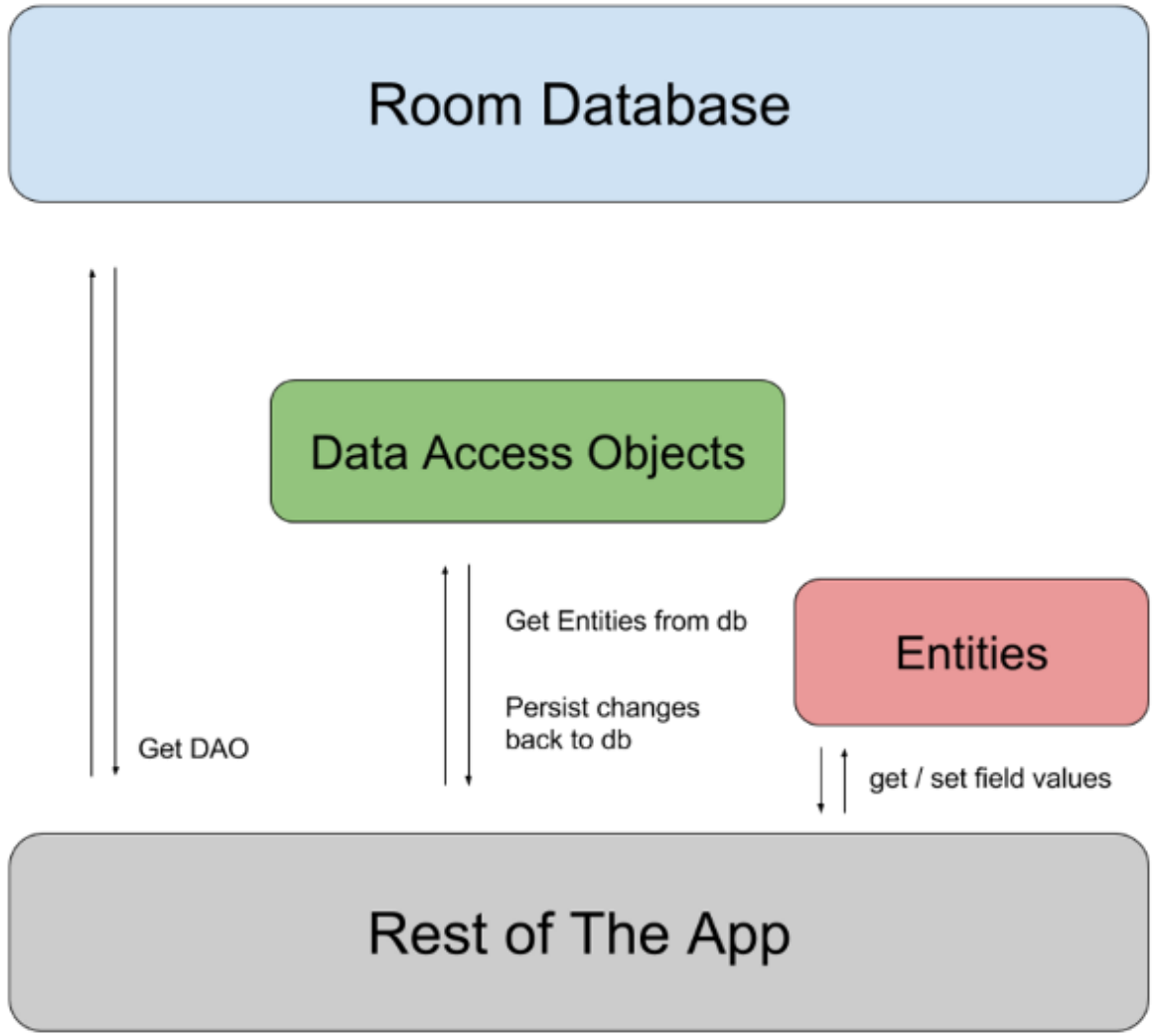


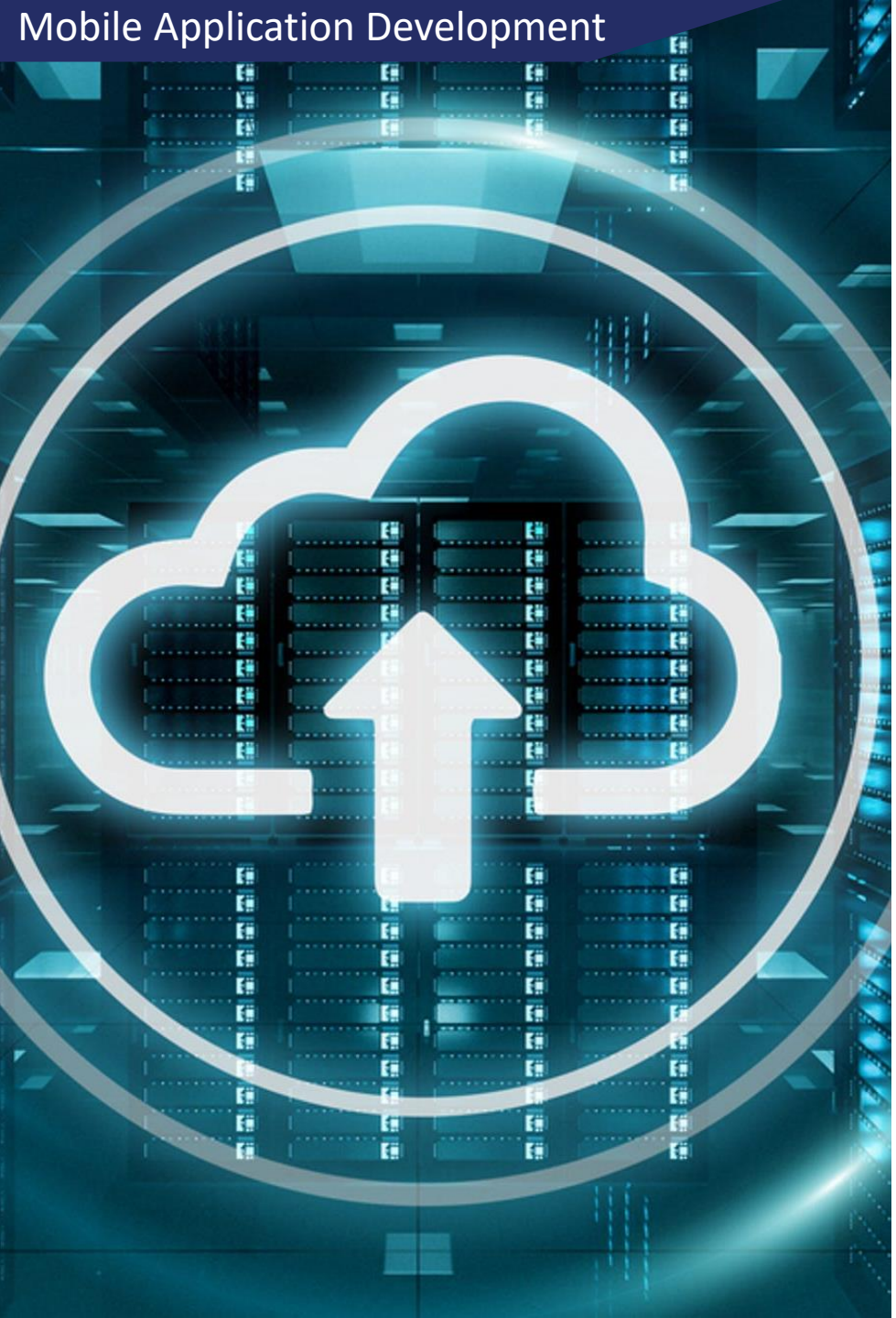
Save data in a local database using Room

- Apps that handle non-trivial amounts of structured data can benefit greatly from persisting that data locally.
- The most common use case is to cache relevant pieces of data so that when the device cannot access the network, the user can still browse that content while they are offline.
- The Room persistence library provides an abstraction layer over SQLite to allow fluent database access while harnessing the full power of SQLite.
- Benefits of Room
  - Compile-time verification of SQL queries.
  - Convenience annotations that minimize repetitive and error-prone boilerplate code.
  - Streamlined database migration paths.

Primary components in Room

- The database class that holds the database and serves as the main access point for the underlying connection to your app's persisted data.
- Data entities that represent tables in your app's database.
- Data access objects (DAOs) that provide methods that your app can use to query, update, insert, and delete data in the database.

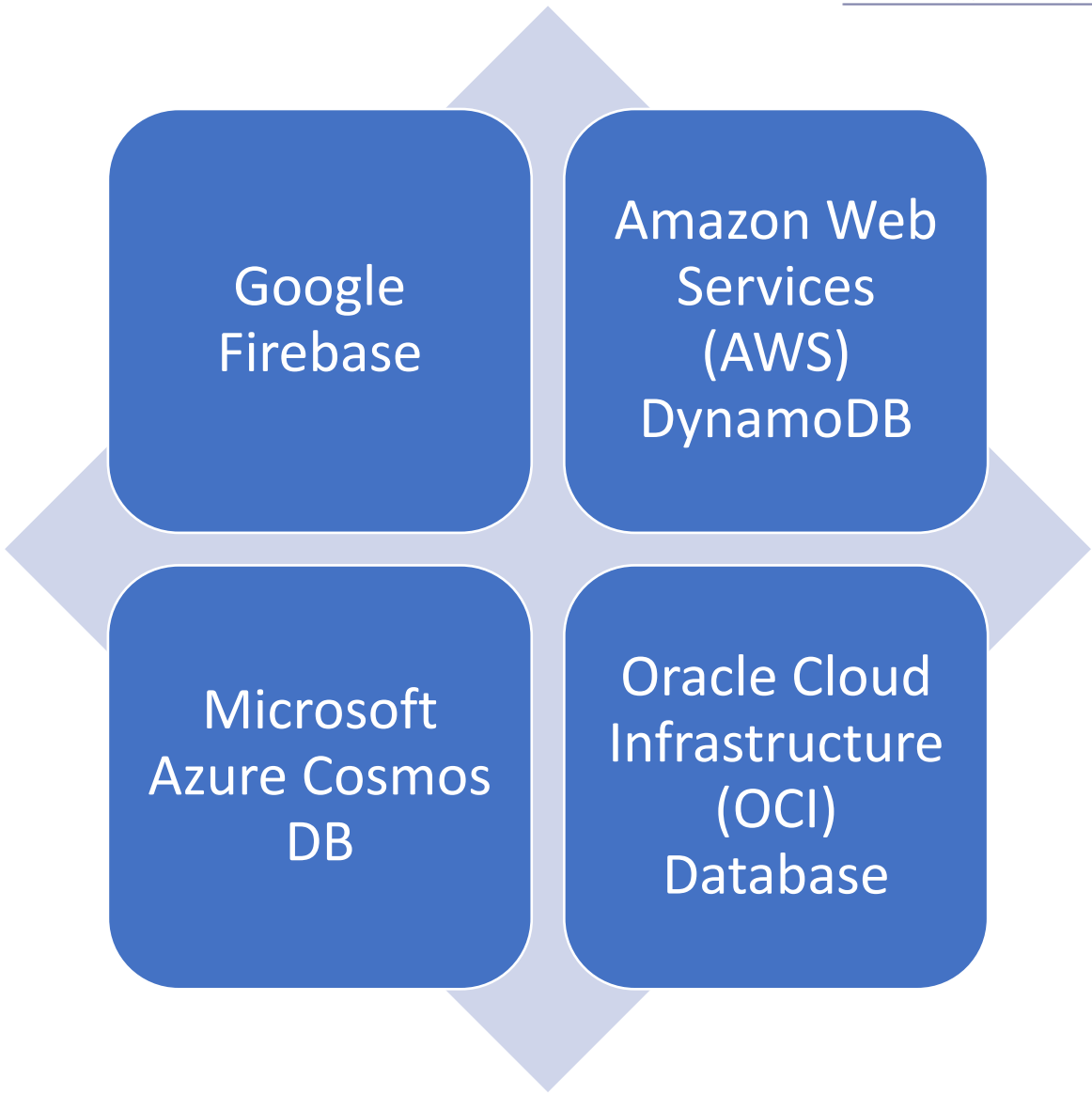




Cloud Storage

- Cloud storage refers to the storage of data on remote servers that can be accessed over the internet.
- There are several cloud storage providers available that offer APIs for Android developers to integrate cloud storage into their apps.
- Some popular cloud storage providers include Google Drive, Dropbox, Amazon S3, and Microsoft OneDrive.
- Cloud database refers to a database service that is hosted in the cloud and can be accessed over the internet.

Cloud Database Options

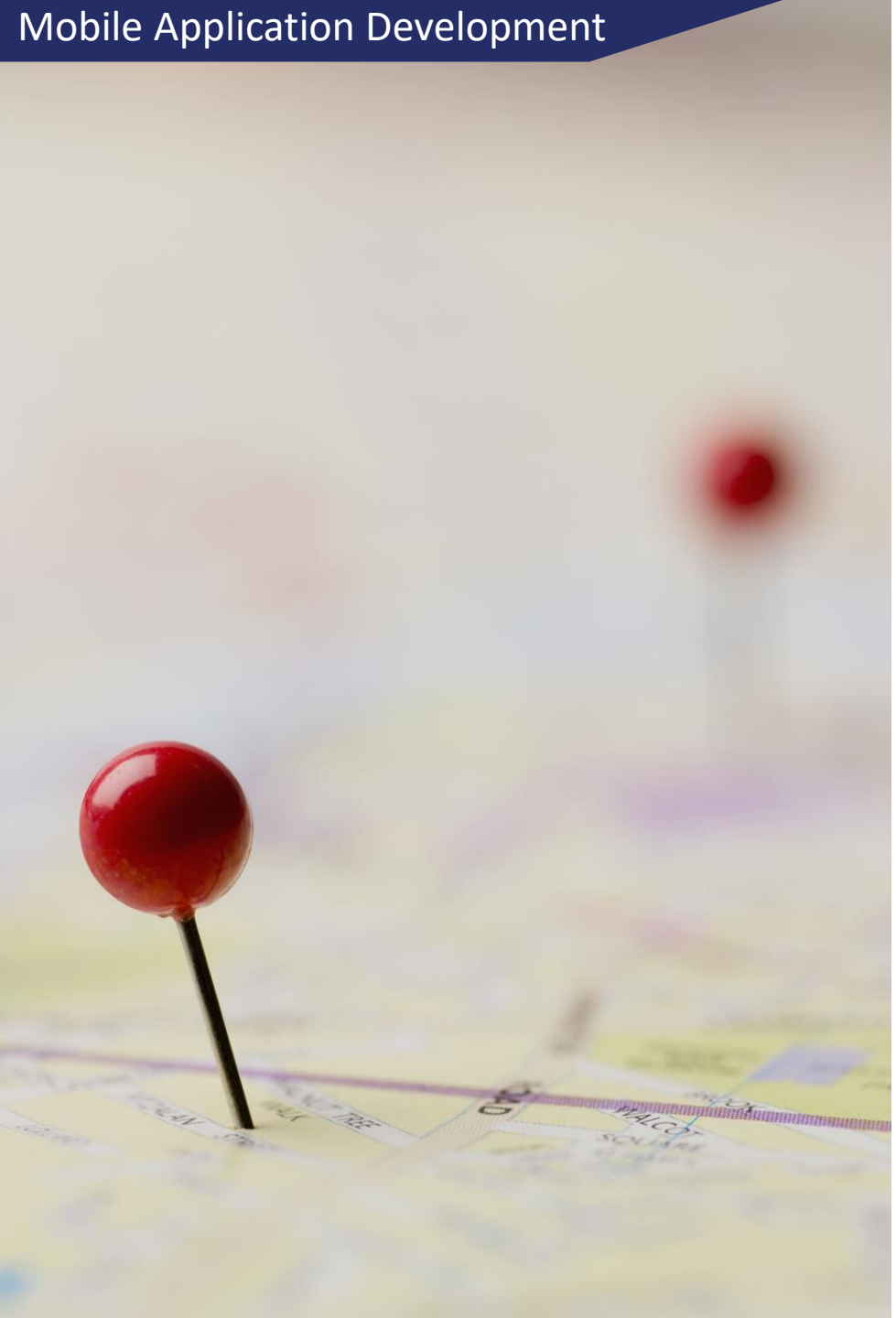


Firebase

- Firebase is a comprehensive mobile development platform that provides a suite of tools and services for Android developers to build high-quality apps.
- Firebase offers a real-time cloud database, user authentication, cloud messaging, analytics, crash reporting, and more.
- Firebase Realtime Database is a NoSQL cloud database that stores data in JSON format and synchronizes data between clients in real-time. It provides offline data persistence and scales to handle millions of concurrent users.
- Firebase Authentication provides a simple way for users to authenticate with your app using email/password, social media accounts, or phone numbers. It also integrates with Google Sign-In and offers robust security features.
- Firebase Cloud Messaging (FCM) allows you to send push notifications to your app users, even when the app is not running in the foreground. It supports both Android and iOS devices and provides advanced targeting and customization options.

Firebase Continue.

- Firebase Analytics helps you understand how users engage with your app and provides insights into user behavior, demographics, and app performance. It also integrates with Google Analytics and AdMob.
- Firebase Crash Reporting provides real-time reports on app crashes and exceptions, including detailed stack traces and device information. It also integrates with Google Play Console for easy app distribution and management.
- Firebase Performance Monitoring helps you optimize your app's performance by identifying slow network requests, rendering issues, and other performance bottlenecks. It provides detailed traces and metrics for each network request and UI interaction.
- Firebase provides a powerful and easy-to-use platform for Android developers to build and scale their apps. With Firebase, you can focus on building great user experiences and leave the infrastructure and backend management to Google.



SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Thank you!

References

- https://developer.android.com/training/data-storage/room
- https://developer.android.com/training/data-storage/shared-preferences
- https://firebase.google.com/docs

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

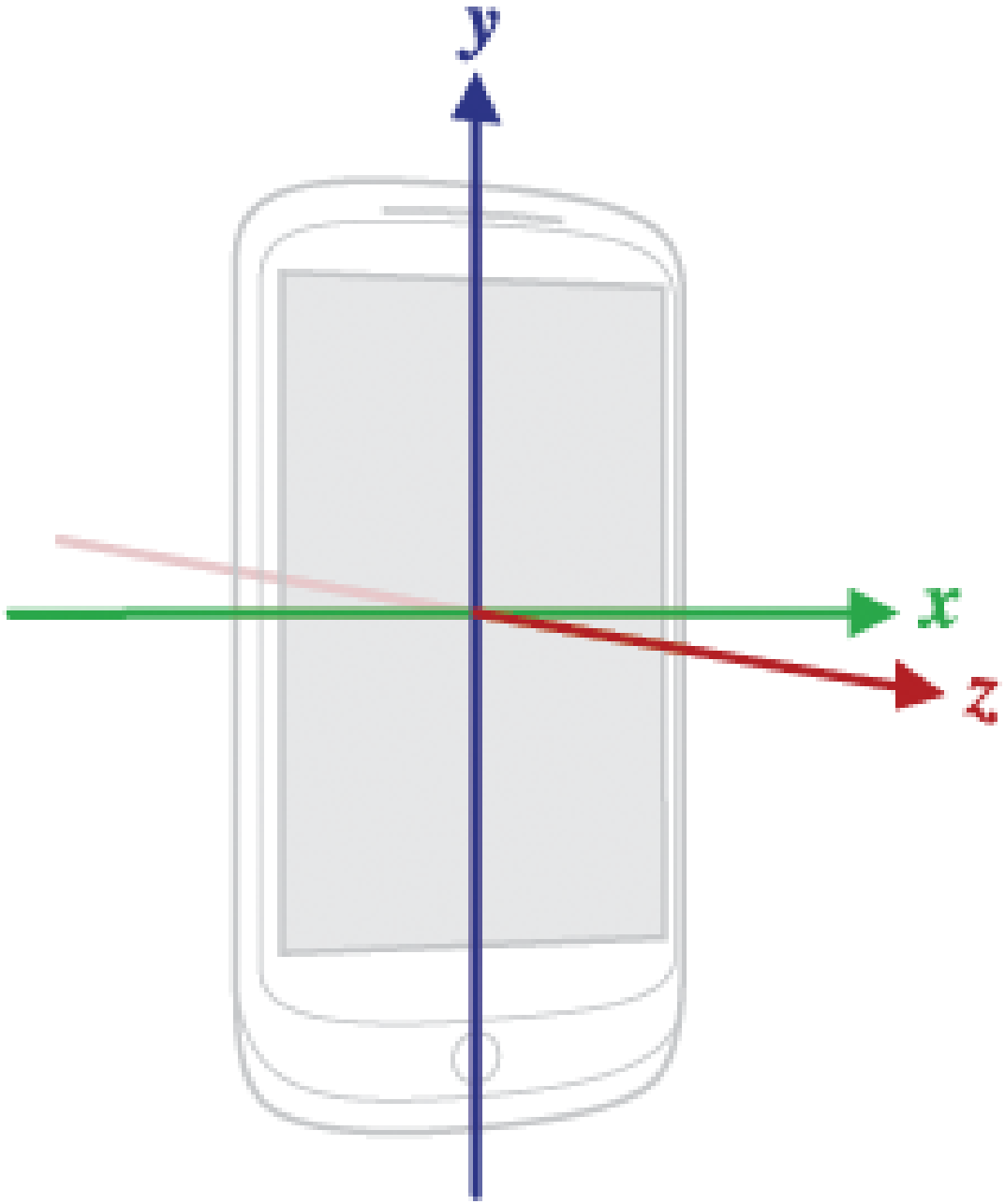
Mobile Application Development

Mobile Platforms and Application Development fundamentals

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Lecture Plan

- Introduction to App Development
- Mobile Platforms and Application Development fundamentals
- Mobile Interface Design Concepts and UI/UX Design
- Introduction to Android Operating System
- Main Components of Android Application
- Data Handling in Android Applications
- Sensors and Media Handling in Android Applications
- Android Application Testing and security aspects



Sensors

- Most Android-powered devices have built-in sensors.
- These sensors measure motion, orientation, and various environmental conditions.
- The sensors can provide raw data with high precision and accuracy.
- The sensors are useful for monitoring three-dimensional device movement or positioning, or changes in the ambient environment near a device.

Sensor Categories

Motion sensors

- These sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.

Environmental sensors

- These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.

Position sensors

- These sensors measure the physical position of a device. This category includes orientation sensors and magnetometers.

Introduction to sensors

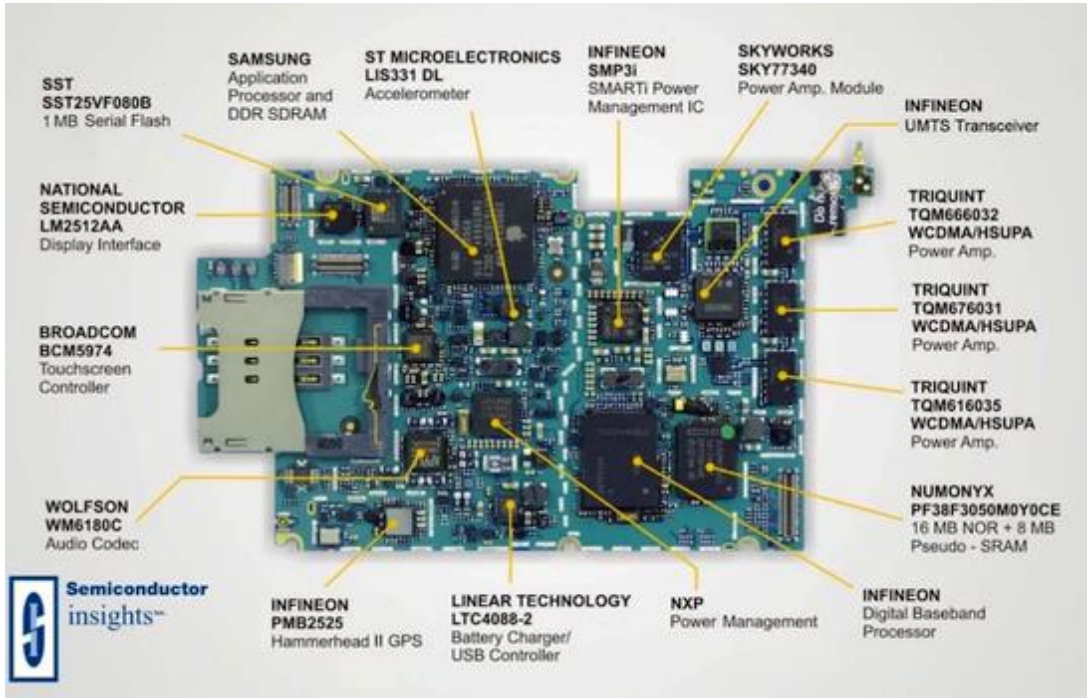
- Android sensor framework provides access to various sensors
- Two types of sensors: hardware-based and software-based
- Hardware-based sensors are physical components in a device
- They measure environmental properties directly
- Software-based sensors mimic hardware-based sensors
- They get their data from hardware-based sensors
- Software-based sensors are also known as virtual or synthetic sensors
- Examples of software-based sensors are linear acceleration and gravity sensors.

Sensors supported by Android Versions

Android Version	Supported Sensor Types
Android 1.5 (Cupcake)	Accelerometer, Magnetic field
Android 1.6 (Donut)	Accelerometer, Magnetic field, Orientation, Temperature
Android 2.1 (Eclair)	Accelerometer, Magnetic field, Orientation, Temperature, Proximity, Light
Android 2.2 (Froyo)	Accelerometer, Magnetic field, Orientation, Temperature, Proximity, Light, Pressure
Android 2.3 (Gingerbread)	Accelerometer, Magnetic field, Orientation, Temperature, Proximity, Light, Pressure
Android 4.0 (Ice Cream Sandwich)	Accelerometer, Magnetic field, Orientation, Gyroscope, Light, Proximity, Pressure
Android 4.1 - 4.3 (Jelly Bean)	Accelerometer, Magnetic field, Orientation, Gyroscope, Light, Proximity, Pressure, Humidity, Temperature
Android 4.4 (KitKat)	Accelerometer, Magnetic field, Orientation, Gyroscope, Light, Proximity, Pressure, Humidity, Temperature, Step counter, Step detector
Android 5.0 (Lollipop)	Accelerometer, Magnetic field, Orientation, Gyroscope, Light, Proximity, Pressure, Humidity, Temperature, Step counter, Step detector
Android 6.0 (Marshmallow)	Accelerometer, Magnetic field, Orientation, Gyroscope, Light, Proximity, Pressure, Humidity, Temperature, Step counter, Step detector
Android 7.0 – 12.0	Accelerometer, Magnetic field, Orientation, Gyroscope, Light, Proximity, Pressure, Humidity, Temperature, Step counter, Step detector, Heart rate

Sensor framework

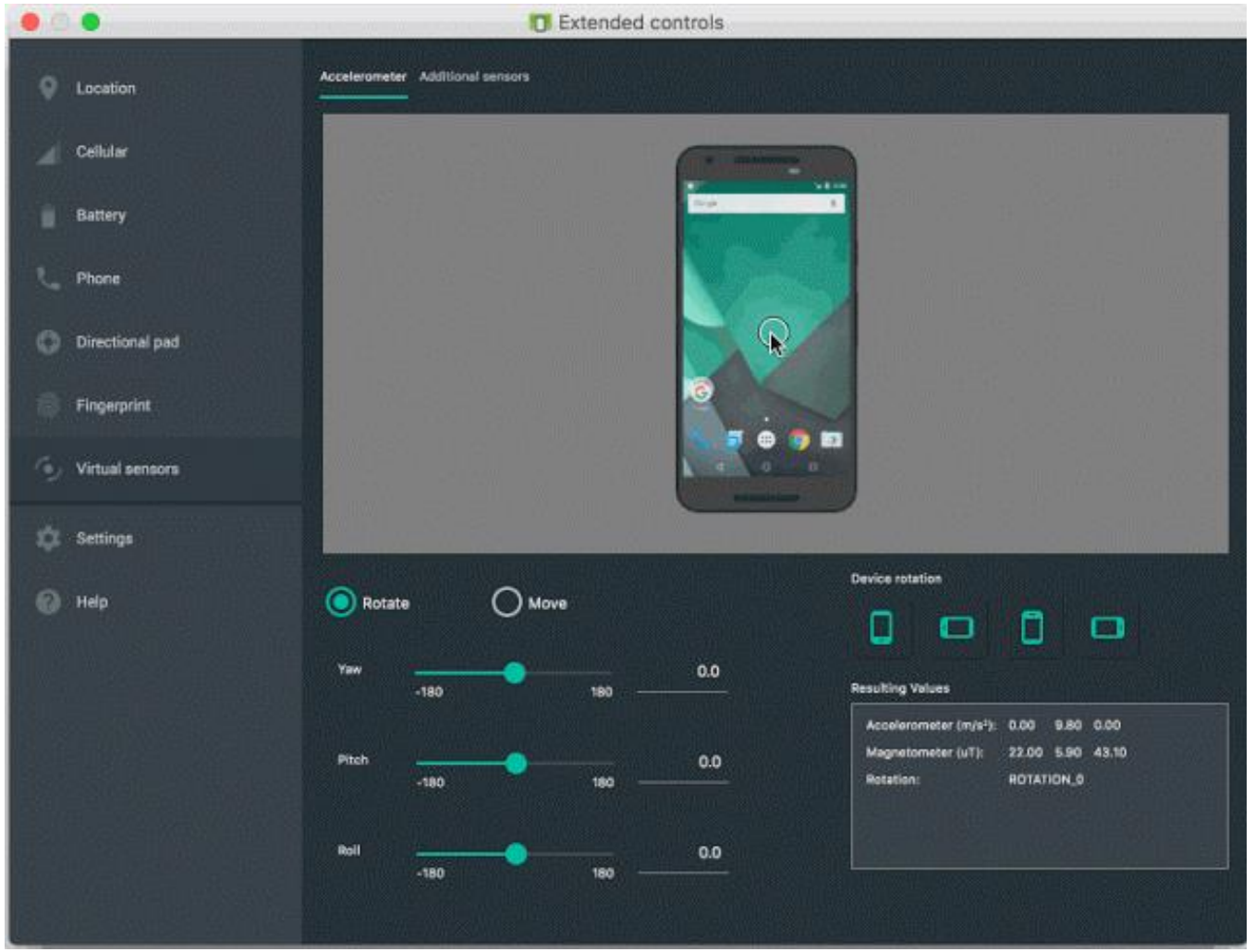
- You can access these sensors and acquire raw sensor data by using the Android sensor framework.
- The framework is part of the android.hardware package and includes classes like SensorManager, Sensor, SensorEvent, and SensorEventListener.
- SensorManager is used to create an instance of the sensor service and provides methods to access, list, and calibrate sensors.
- Sensor is used to create an instance of a specific sensor and provides methods to determine its capabilities.
- SensorEvent is used to create a sensor event object that includes information like raw sensor data, sensor type, accuracy, and timestamp.
- SensorEventListener is used to create callback methods that receive notifications when sensor values or accuracy change.
- The framework can be used to identify sensors and their capabilities at runtime and monitor sensor events to acquire raw sensor data.
- Identifying sensors and their capabilities is useful for features that rely on specific sensors or types, while monitoring sensor events is how raw sensor data is acquired.



Best Practices when using Sensors

- Use the appropriate sensor for the job
- Minimize sensor usage
- Handle sensor events efficiently
- Consider sensor accuracy
- Test on multiple devices
- Handle sensor errors gracefully
- Respect user privacy

Test with the Android Emulator



Media Handling

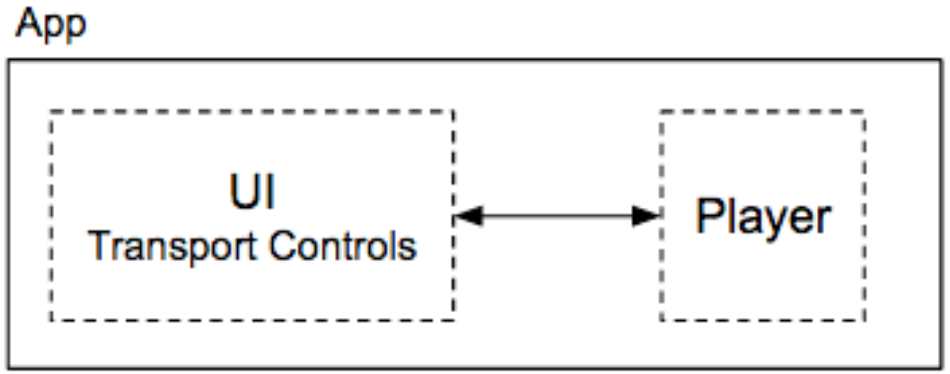
- Media, such as images, videos, and audio, is an integral part of many Android apps, from social media and entertainment apps to news and productivity apps.
- Effective media handling can significantly enhance the user experience of an app, making it more engaging, informative, and enjoyable to use.
- Poorly optimized media handling can lead to slow loading times, stuttering or freezing during playback, and excessive battery drain, all of which can negatively impact user experience.
- Media handling in Android can be complex, with multiple file formats and codecs to support, various APIs to choose from, and performance optimization techniques to implement, making it an area where developers need to have expertise to create high-quality apps.
- Ensuring proper media handling is also important for app security, as media can potentially contain malicious code or vulnerabilities that can be exploited by attackers.

Media Types

- Android can handle wide range of media types of images, videos, and audio.
- For Images:
  - JPEG: widely used for photographs and other images with many colors
  - PNG: used for images with transparency or where lossless compression is required
  - GIF: used for simple animations or small file size images with limited colors
  - WebP: developed by Google for web use, provides both lossy and lossless compression
- For Videos:
  - MP4: commonly used for video on the web and mobile devices, supports a variety of codecs
  - AVI: widely used for storing video on PCs, supports multiple codecs
  - MOV: developed by Apple for QuickTime, widely used for video on Macs and iPhones
  - MKV: open-source format, supports multiple audio and subtitle tracks, often used for high-definition video
- For Audio:
  - MP3: widely used for music and other audio files, provides good quality with reasonable file size
  - AAC: developed as a successor to MP3, provides better quality at lower bitrates
  - WAV: uncompressed format used for high-quality audio, often used for recording and editing
  - Ogg Vorbis: open-source format, provides good quality at lower bitrates, often used for streaming and gaming
- Refer the following link for more details
  - Supported media formats | Android Developers

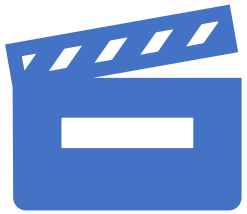
Media app Architecture

- A multimedia application that plays audio or video usually has two parts
- A player that takes digital media in and renders it as video and/or audio
- A UI with transport controls to run the player and optionally display the player's state
- In Android you can build your own player from the ground up, or you can choose from these options:
  - The MediaPlayer class provides the basic functionality for a bare-bones player that supports the most common audio/video formats and data sources.
  - **ExoPlayer** is an open source library that's built on top of lower-level media framework components like MediaCodec and AudioTrack. ExoPlayer supports high-performance features like DASH which are not available in MediaPlayer. You can customize the ExoPlayer code, making it easy to add new components. ExoPlayer can only be used with Android version 4.1 and higher.



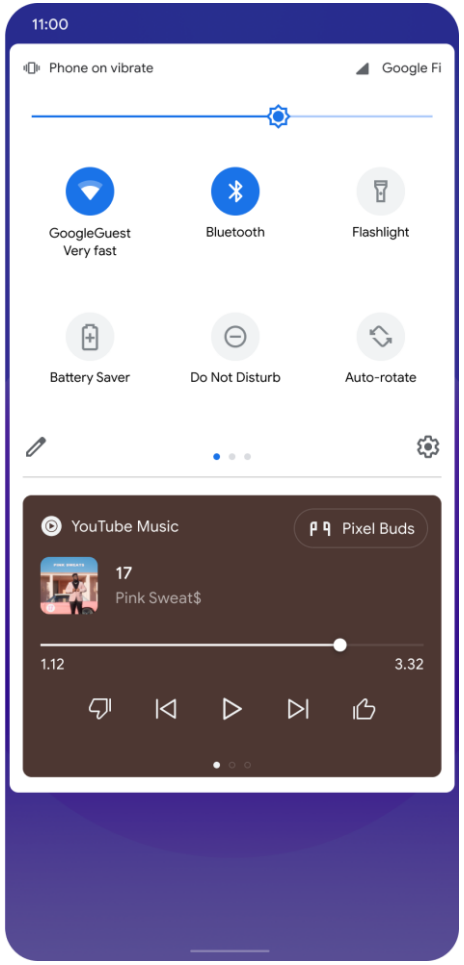
Media APIs

Media Player



Exo Player

Media Store

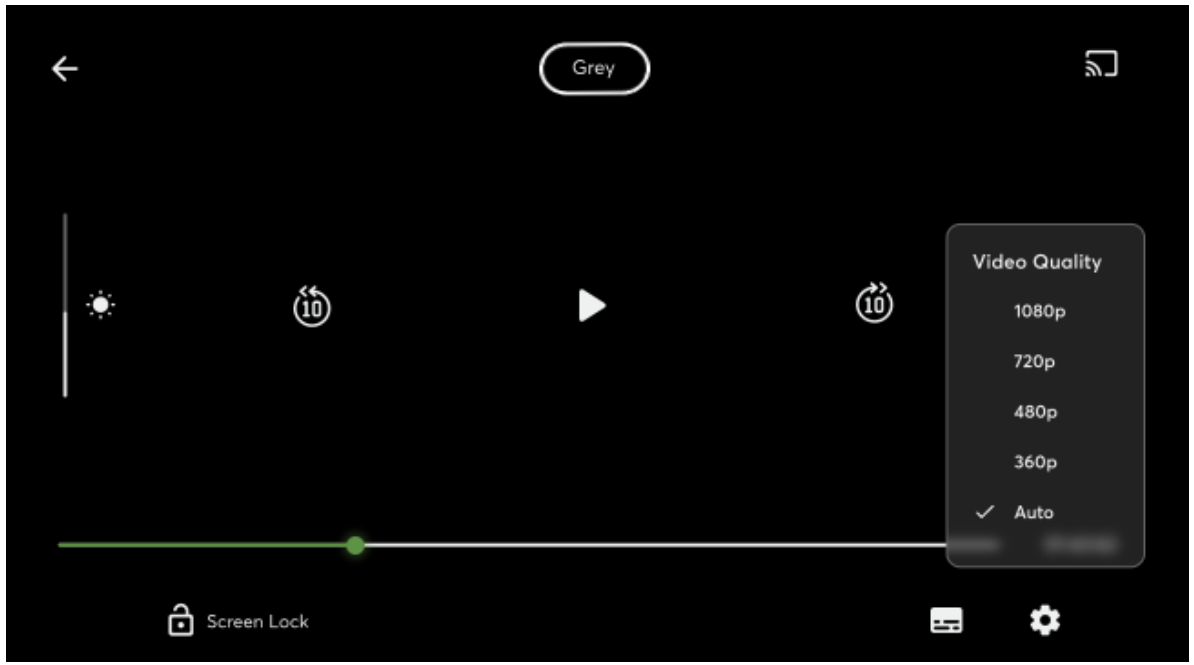


Media Player API

- Provides a basic framework for playing audio and video in an Android app.
- Supports common media formats, including MP4, 3GP, and MP3.
- Provides methods for controlling playback, such as play, pause, stop, and seek.
- Supports playback from different sources, such as local files, remote URLs, and streams.
- Can be used to play media in the foreground or the background.

ExoPlayer

- A more advanced media player that provides additional features and flexibility compared to MediaPlayer.
- Supports a wider range of media formats and codecs, including adaptive streaming formats such as DASH and HLS.
- Provides more control over buffering, network connectivity, and quality levels.
- Supports features such as seamless switching between audio and video tracks, trick play (fast-forward and rewind), and DRM.
- Offers better performance and battery efficiency compared to MediaPlayer



MediaStore

- A content provider that provides access to the media files on the device, including images, videos, and audio.
- Provides methods for querying and retrieving media metadata, such as title, artist, and duration.
- Allows apps to create, update, and delete media files in the device's media library.
- Can be used to display and play media files from the device's media library in an app.

Working With Images

- Using the ImageView widget
  - ImageView is a UI widget in Android that is used to display images in an app's user interface.
  - To display an image in an ImageView, you need to set the image resource or image URL using the setImageResource() or setImageURI() method, respectively.
  - You can also set the image programmatically using a Bitmap object with the setImageBitmap() method.
  - To resize or scale the image to fit the ImageView, you can use the setScaleType() method with one of the available scale types, such as FIT_CENTER or CENTER_CROP.
- Loading images from the network
- Loading images from the device's storage:

Working With Images

- Using the ImageView widget
- Loading images from the network
  - To load images from the network, you can use a networking library such as Retrofit or Volley to fetch the image from a remote URL.
  - Once you have the image data, you can create a Bitmap object from it using the BitmapFactory class, and then set the Bitmap in an ImageView using the setImageBitmap() method.
- Loading images from the device's storage:

Working With Images

- Using the ImageView widget
- Loading images from the network
- Loading images from the device's storage:
  - To load images from the device's storage, you can use the MediaStore API to query the device's media library for the image file.
  - Once you have the image file URI, you can use the ContentResolver and BitmapFactory classes to create a Bitmap object from the file, and then set the Bitmap in an ImageView using the setImageBitmap() method.

Optimizing image loading and handling large images in an Android app

- Caching:
  - Caching involves storing frequently used images in memory or on disk to avoid the overhead of reloading the image every time it's needed.
  - There are several caching libraries available for Android, such as Picasso, Glide, and Fresco, that can handle caching and other optimizations automatically.
- Down sampling:
  - Down sampling involves reducing the resolution of the image to a smaller size to reduce memory usage and improve performance.
  - You can use the BitmapFactory options to downsample images when loading them, for example by setting the inSampleSize option to a value that reduces the image size by a factor of 2, 4, or 8.
- Compressing:
  - Compressing involves reducing the size of the image file itself to reduce the amount of data that needs to be loaded and displayed.
  - You can use image compression tools or libraries, such as WebP or PNGQuant, to reduce the file size of the image without compromising quality.
- Handling large images:
  - Large images can cause performance issues, especially on older or less powerful devices.
  - One technique for handling large images is to use tiling or slicing to break the image into smaller pieces and load only the visible portion of the image at any given time.
  - Another technique is to load a lower resolution version of the image initially and then load the full resolution version when the user zooms in or requests it.

Working with Videos

Handling different video formats and resolutions:

- Android supports a wide range of video formats and resolutions, including MP4, 3GP, and WebM.
- To play different video formats, you can use the MediaPlayer or ExoPlayer APIs, which support a range of codecs and formats.
- To handle different video resolutions, you can use the SurfaceView or TextureView classes to display the video, which can handle different aspect ratios and resolutions.

Working with Videos

Implementing seeking and playback controls:

- To implement seeking and playback controls, you can use the MediaController class, which provides a set of standard playback controls, such as play, pause, seek, and volume control.
- You can attach a MediaController to a MediaPlayer or ExoPlayer instance using the setMediaController() method.
- Alternatively, you can create custom playback controls using UI widgets, such as buttons or seek bars, and handle the playback actions in code.

Working with Videos

Video playback in the background:

- To play video in the background, you can use the MediaPlayer or ExoPlayer APIs in conjunction with a Service or a Foreground Service.
- When using a Service, you can create a new thread or handler to manage the playback and send updates to the UI as needed.
- When using a Foreground Service, you need to create a notification to inform the user that video playback is ongoing, and handle user interaction with the notification to control the playback.

Working with audio

- Android supports a wide range of audio formats, including MP3, AAC, Ogg Vorbis, and WAV.
- To play different audio formats, you can use the MediaPlayer or ExoPlayer APIs, which support a range of codecs and formats.
- To play audio in the background, you can use the MediaPlayer or ExoPlayer APIs in conjunction with a Service or a Foreground Service.
- When using a Service, you can create a new thread or handler to manage the playback and send updates to the UI as needed.
- When using a Foreground Service, you need to create a notification to inform the user that audio playback is ongoing, and handle user interaction with the notification to control the playback.

Thank You!

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Mobile Application Development

Mobile Platforms and Application Development fundamentals

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Lecture Plan

- Introduction to App Development
- Mobile Platforms and Application Development fundamentals
- Mobile Interface Design Concepts and UI/UX Design
- Introduction to Android Operating System
- Main Components of Android Application
- Sensors and Media Handling in Android Applications
- Data Handling in Android Applications
- Android Application Testing and security aspects

Purpose of Software Testing

- What is a software test?
  - A piece of software which executes another piece of software.
  - Validates if the code results as expected.
  - Software unit tests help the developer to verify that the logic of piece of the program is correct.

Android App Testing

- Android app testing is a complex task due to the existence of multiple device manufacturers, device models, Android OS versions, screen sizes, and network conditions.

✓ Testing on Real Android Devices

Testing against a wide selection of devices from various manufacturers with different screen resolutions and Android OS versions.

✓ Immediate new Android version support

Supported for newly release devices and Android versions

Android App Testing

✓Test complex scenarios and custom UI elements

- Testing coverage integrations with device components, peripherals, and system apps such as camera, audio, GPS, Google now, Google Assistant or Google Maps.

- Automate customized actions and UI elements such as sliders, pickers, tables, gestures, etc.

✓ Test performance to ensure a great user experience

- Able to catch performance issues before deployment.

To-Do before Mobile Testing for first release

1. Research on OS and Devices
2. Test Bed
3. Test plan
4. Automation Tools
5. Testing techniques or methods

Best Practices in Android App Testing

- ✓Device Selection
- ✓Beta Testing of the Application
- ✓Connectivity
- ✓Manual or Automated Testing

Testing Types for Mobile Apps

1. Functional Testing
   - ✓ Checks whether the application is working based on the requirements.
   - ✓ The flow of use cases and various business rules are tested.

Eg:

To validate whether – all required mandatory fields are working as required.

- the device is able to perform required multitasking requirements.

- navigation between relevant modules in the app as per the requirement.

- the user receives appropriate error messages like “network error, try again after some time”, etc..

Testing Types for Mobile Apps

2. Android UI Testing

- ✓ User-centric testing of the application is done under this.
- ✓ Normally performed by manual users.

Eg: testing – visibility of text in various screens of the app

- interactive messages
- alignment of data
- look and feel of the app for different screens
- whether the buttons are in required size and suitable to big fingers.
- whether the icons are natural and consistent with the app.
- size of fields, etc..

Testing Types for Mobile Apps

3. Compatibility Testing

- ✓Performed to ensure that the app is fit across all the devices because they have different size, resolution, screen, version and hardware.
- ✓So, mostly done in form of two matrices
  1. OS vs. App
  2. Device model vs. App

Eg: To ensure – the UI of the app is as per the screen size o the device and no text/control is partially invisible or inaccessible.

- text is readable for all users.
- call/alarm functionality is enabled whenever the app is running.

Testing Types for Mobile Apps

4. Interface Testing / Integration Testing

- ✓This is done after all the modules of the app are completely developed.

- ✓Includes a complete end-to-end testing of the app, interaction with other apps like Maps and social apps, usage of microphone to enter text, usage of camera to scan a barcode or to take a picture, etc.

Testing Types for Mobile Apps

5. Network Testing

- ✓Mainly done to verify the response time in which the activity is performed like refreshing data after sync or loading data after login.
- ✓This is an in-house testing.
- ✓Done for both strong WiFi connection and the mobile data network.
- ✓Request/response to/from the service is tested for various conditions.
- ✓App should talk to the immediate service to carry out the process.

Testing Types for Mobile Apps

6. Performance Testing

- ✓Performance of the app under some conditions are checked.
- ✓Tested from both application end and the app server end.
- ✓Conditions- Low memory in the device
  - The battery in extremely at a low level.
  - Poor/Bad network reception.

Testing Types for Mobile Apps

7. Installation Testing

- ✓This is done to ensure that the installation of the app is going smoothly without ending up in errors or partial installation etc.
- ✓ Upgrade and uninstallation testing are carried out as part of this testing.

Testing Types for Mobile Apps

8. Security Testing

- ✓Testing for the data flow for encryption and decryption mechanism is tested under this.

Eg: -To validate whether the app is not permitting an attacker to access sensitive content or functionality without proper authentication.

- To validate the app has a strong password protection system.

- To prevent from insecure data storage in the keyboard cache of the applications.

Testing Types for Mobile Apps

9. Field Testing

- ✓Done specifically for the mobile data network.
- ✓Doing only after the whole app is developed.
- ✓Verify the behavior of the app when the phone has 2G or 3G connection.
- ✓This testing verifies if the app is crashing under slow network connection or if it is taking too long to load the information.

Testing Types for Mobile Apps

10. Interrupt Testing (Offline Scenario Verification)

✓Offline conditions – Condition where the communication breaks in the middle

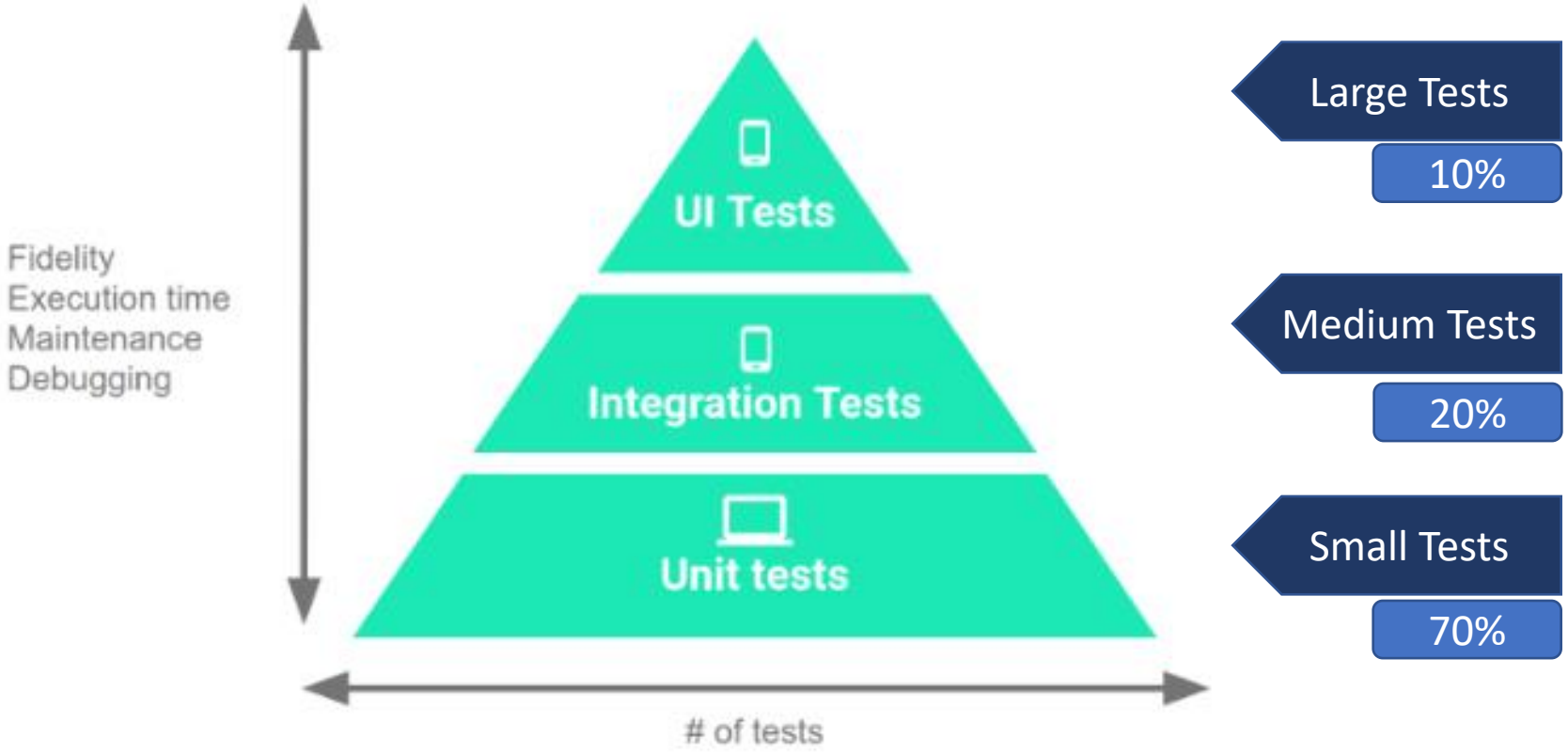
Eg: - Data cable removal during data transfer process.
- Network outage during the transaction posting phase.
- Network recovery after an outage.
- Battery removal or power ON/Off when it is in the transactional phase.

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Mobile Testing tools

- Kobiton
- TestProject
- Squish By FroLogic
- TestingBot
- Apptim
- Headspin
- Appium (iOS/Android Testing tool)
- Selendroid
- MonkeyRunner
- Calabash
- KIF
- Testroid
- Robotium - Android
- Robo-electric - Android

Writing Tests



Testing Pyramid

Write Small Tests

- Highly focused to Unit tests.

Local Unit tests	Instrumented Unit tests
• Use AndroidX Test APIs	• Can be done on a physical device or emulator
• Can be used Robolectric for tests that always run on a JVM-powered development machine	• AndroidX test makes use of following threads. - **Main threa**d (UI thread/ activity thread) -> occur UI interactions and activity lifecycle events - **Instrumentation thread** -> most of the tests are run under this. When the test suit begins, the AndroidJUnitTest class starts this thread.
• Roboelectric supports: - Component lifecycles - Event loops - All resources	

Write Small Tests

- Highly focused to Unit tests.

Local Unit tests	Instrumented Unit tests
• Use AndroidX Test APIs	• Can be done on a physical device or emulator
• Can be used Robolectric for tests that always run on a JVM-powered development machine	• AndroidX test makes use of following threads. - **Main threa**d (UI thread/ activity thread) -> occur UI interactions and activity lifecycle events - **Instrumentation thread** -> most of the tests are run under this. When the test suit begins, the AndroidJUnitTest class starts this thread.
• Roboelectric supports: - Component lifecycles - Event loops - All resources	

Write Medium Tests

- Validate the collaboration and interaction of a group of units.

Eg:

- ✓Interactions between a view and view model (testing a Fragment object, validating a layout XML, evaluation a data binding logic of a ViewModel object)
- ✓Testing od app's repository layer – verify different data sources and data access objects interact as expected.

- Use methods from the Espresso Intents library.

Write Large Tests

- Validate end-to-end workflows that guide users through multiple modules and features.

Configuring the environment in Android Studio

✓Organize test directories based on execution environment

Android Studio contains two directories to locate tests.

1. androidTest – Contains the tests that run on real or virtual devices.

- Tests include integration tests, end-to-end tests and other tests where JVM alone cannot validate the app's functionality.

2. test – Contains the tests that run on the local machine such as unit tests.

Testing annotations in jUnit4

JUnit 4	Description
`import org.junit.*`	Import statement for using the following annotations.
`@Test`	Identifies a method as a test method.
`@Before`	Executed before each test. It is used to prepare the test environment (e.g., read input data, initialize the class).
`@After`	Executed after each test. It is used to cleanup the test environment (e.g., delete temporary data, restore defaults). It can also save memory by cleaning up expensive memory structures.
`@BeforeClass`	Executed once, before the start of all tests. It is used to perform time intensive activities, for example, to connect to a database. Methods marked with this annotation need to be defined as `static` to work with JUnit.
`@AfterClass`	Executed once, after all tests have been finished. It is used to perform clean-up activities, for example, to disconnect from a database. Methods annotated with this annotation need to be defined as `static` to work with JUnit.

Testing annotations in jUnit4

`@AfterClass`	Executed once, after all tests have been finished. It is used to perform clean-up activities, for example, to disconnect from a database. Methods annotated with this annotation need to be defined as `static` to work with JUnit.
`@Ignore` or `@Ignore("Why disabled")`	Marks that the test should be disabled. This is useful when the underlying code has been changed and the test case has not yet been adapted. Or if the execution time of this test is too long to be included. It is best practice to provide the optional description, why the test is disabled.
`@Test (expected = Exception.class)`	Fails if the method does not throw the named exception.
`@Test(timeout=100)`	Fails if the method takes longer than 100 milliseconds.

Methods to assert test results

Statement	Description
fail([message])	Let the method fail. Might be used to check that a certain part of the code is not reached or to have a failing test before the test code is implemented. The message parameter is optional.
assertTrue([message,] boolean condition)	Checks that the boolean condition is true.
assertFalse([message,] boolean condition)	Checks that the boolean condition is false.
assertEquals([message,] expected, actual)	Tests that two values are the same. Note: for arrays the reference is checked not the content of the arrays.
assertEquals([message,] expected, actual, tolerance)	Test that float or double values match. The tolerance is the number of decimals which must be the same.

Methods to assert test results

assertNull([message,] object)	Checks that the object is null.
assertNotNull([message,] object)	Checks that the object is not null.
assertSame([message,] expected, actual)	Checks that both variables refer to the same object.
assertNotSame([message,] expected, actual)	Checks that both variables refer to different objects.

Thank You!