Mobile Application Development

Mobile Application Development fundamentals

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
Discover Your Future

Lecture Plan

- Mobile Application Development fundamentals
- Mobile Platforms
- Introduction to Android Operating System
- Main Components of Android Application
- Android Interface Design Concepts
- Data handling in Mobile App Development
- Sensors and Media Handling in Android Applications
- Kotlin Language to develop Android Mobile Apps
- Android Application Testing and security aspects

Assessment Criteria

Continuous Assessments

Midterm Examination (MCQ) - 20 %

- Mini Project (Group Project) - 30 %
    1. Project Phase 1 – 10%
    2. Project Phase 2 – 20%

End Semester Examination

- Final Examination (Online) - 50 %

Lecturer Panel

- Lecturer in-charge of the module
  Ms. Disni Sriyaratna (disni.s@sliit.lk)
- Malabe
  Mr. Nelum Chathuranga Amarasena (nelum.a@sliit.lk)
  Mr. Thusithanjana Thilakarathna (thusithanjana.t@sliit.lk)
- Metro
  Mr. S.M.B. Harshanath (harshanath.s@sliit.lk)
- Kandy
  Ms.Gihani Gunarathna (gihani.g@sliit.lk)
  Ms.Nilanka Singhagosha (nilanka.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:

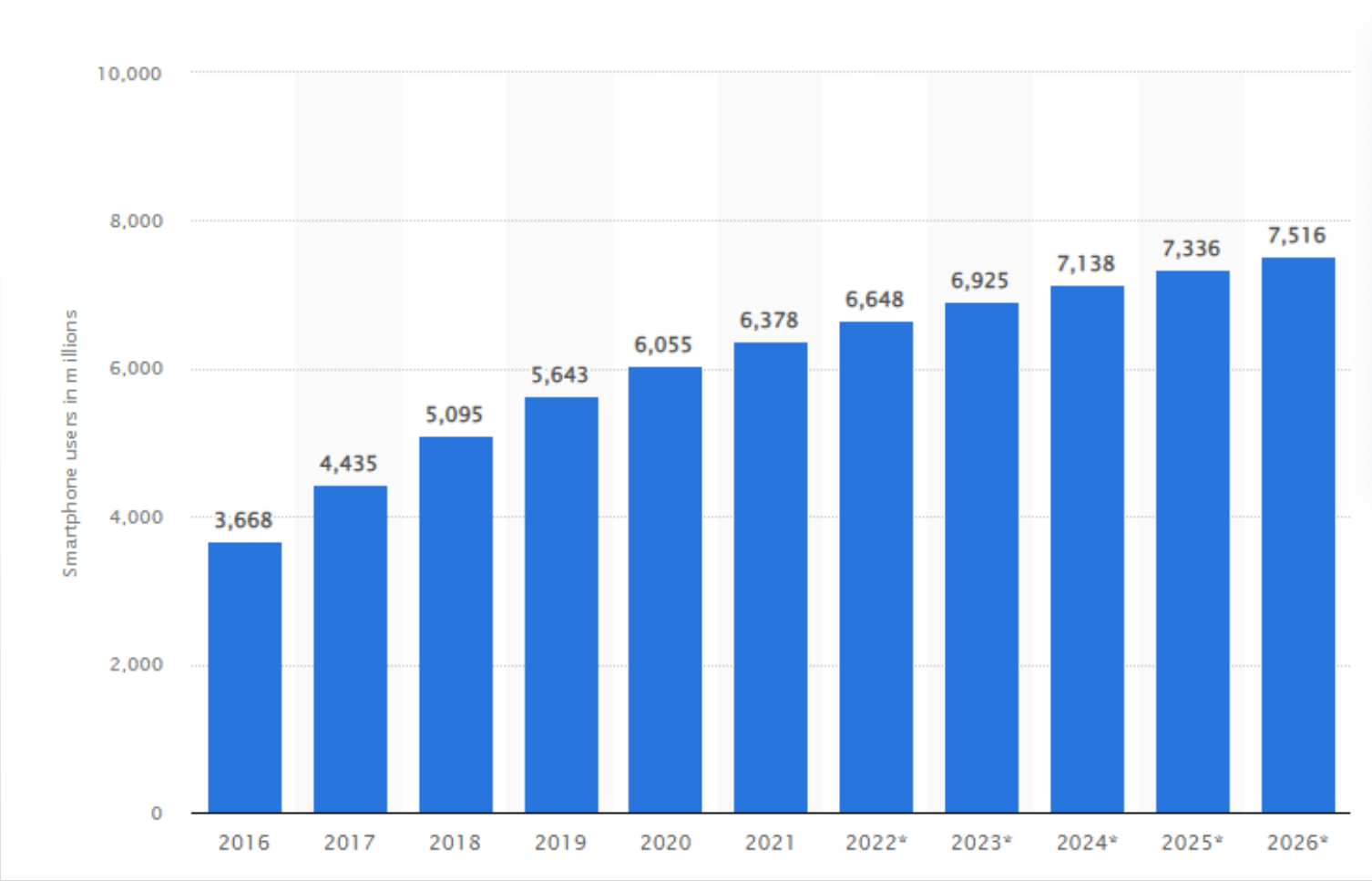
- Understand the fundamentals of mobile Application Development.

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/

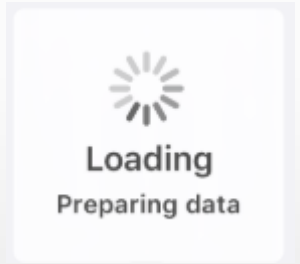
Mobile Application Development

- 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

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"

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

Groups formation to the project

- Register your 4 member groups into given links in courseweb.
- First phase Evaluation –> 7th Academic Week
- Final Evaluation -> 12th Academic Week

NOTE: The above dates are fixed. No extensions will be given.

Thank You

Mobile Application Development

Mobile Platforms

SLIIT
Discover Your Future

Lecture Plan

- Mobile Application Development fundamentals
- Mobile Platforms
- Introduction to Android Operating System
- Main Components of Android Application
- Android Interface Design Concepts
- Data handling in Mobile App Development
- Sensors and Media Handling in Android Applications
- Kotlin Language to develop Android Mobile Apps
- Android Application Testing and security aspects

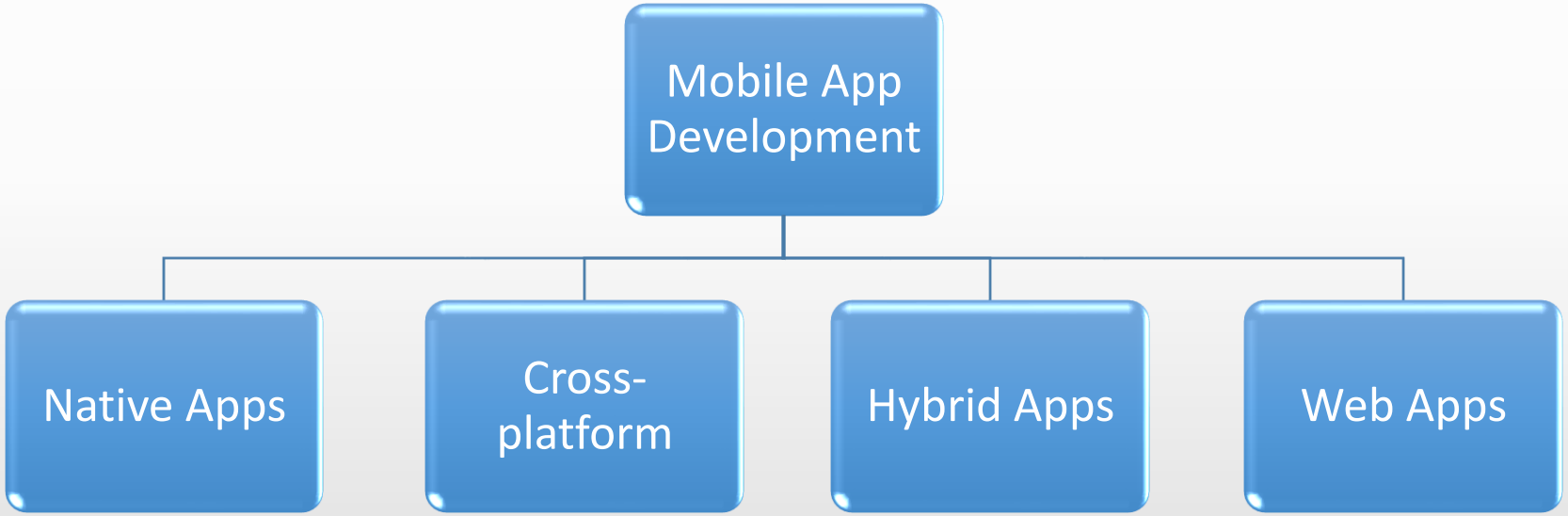
Learning Outcomes of the Lecture

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

- Comprehend native mobile operating systems.
- Describe cross-platform mobile development.
- Describe Hybrid mobile development.

SLIIT
Discover Your Future

Mobile Application Development



SLIIT
Discover Your Future

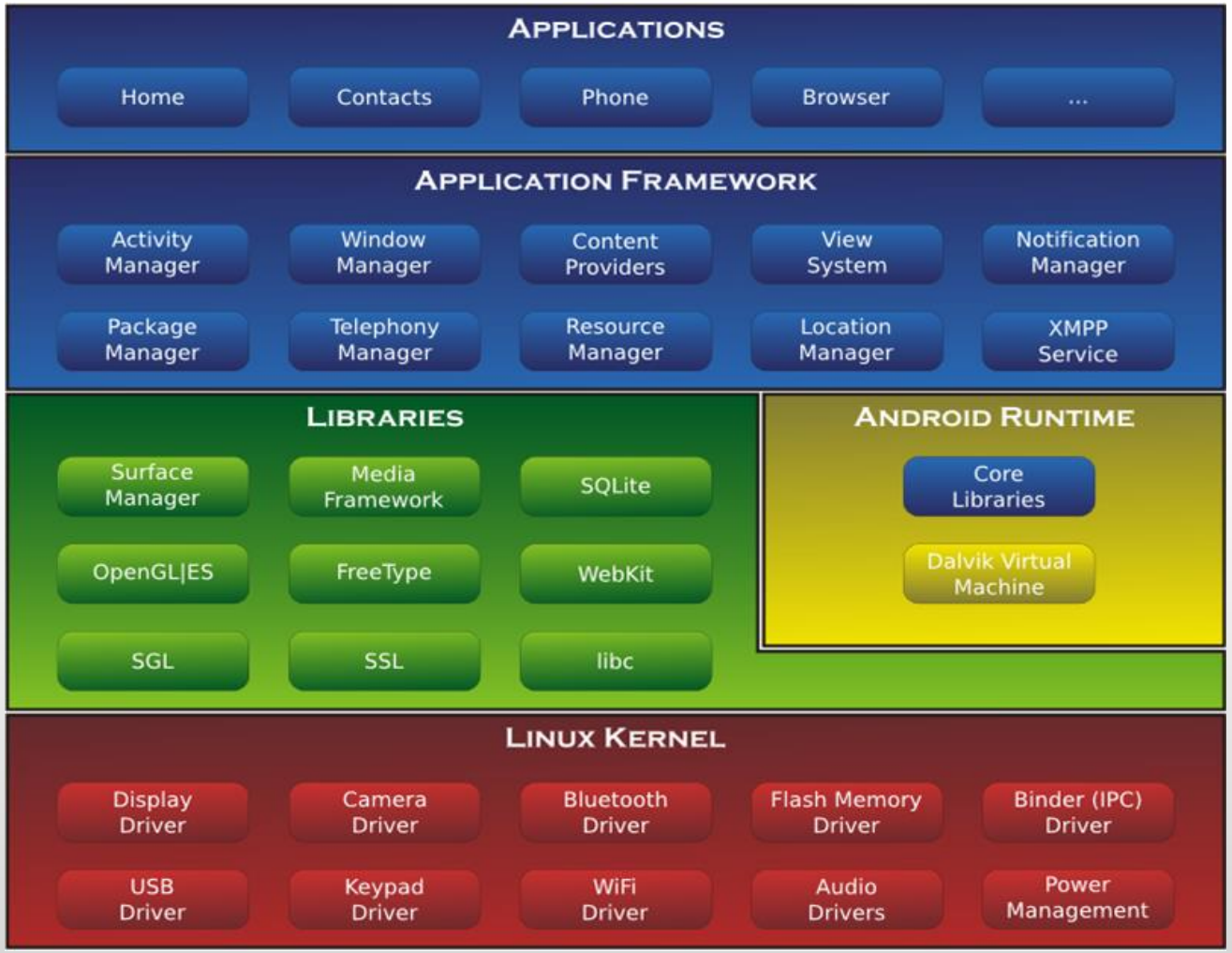
Native Mobile Application

- 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 Architecture



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

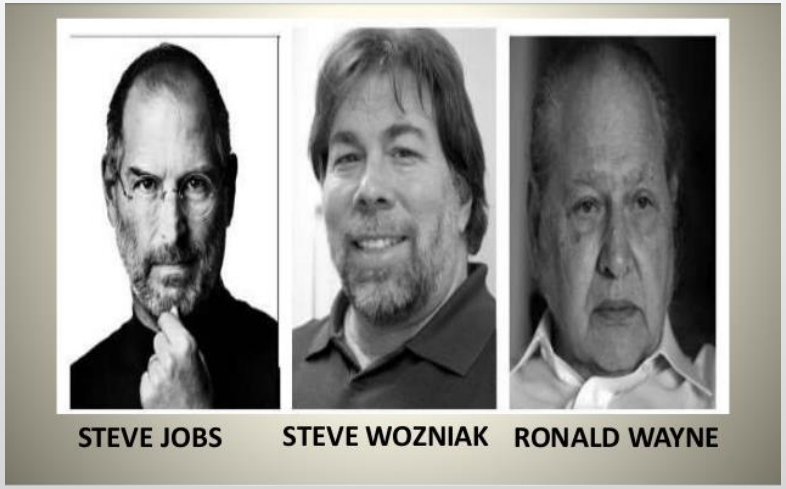
Android Devices

Development Environments

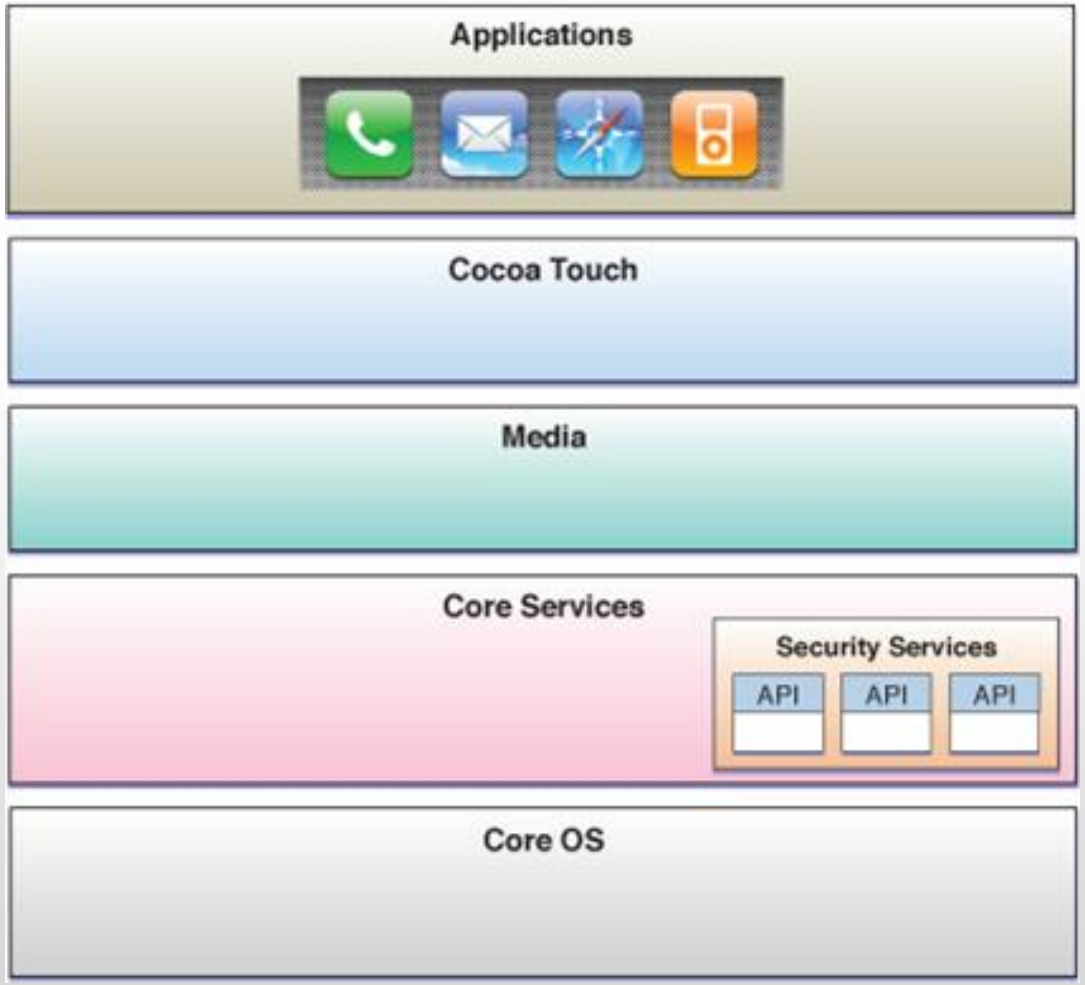
- Android Studio
- Eclipse
- Apache Cordova
- App Inventor for Android
- C++ Builder
- Blue J
- FlashDevelop
- Titanium

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 Architecture



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

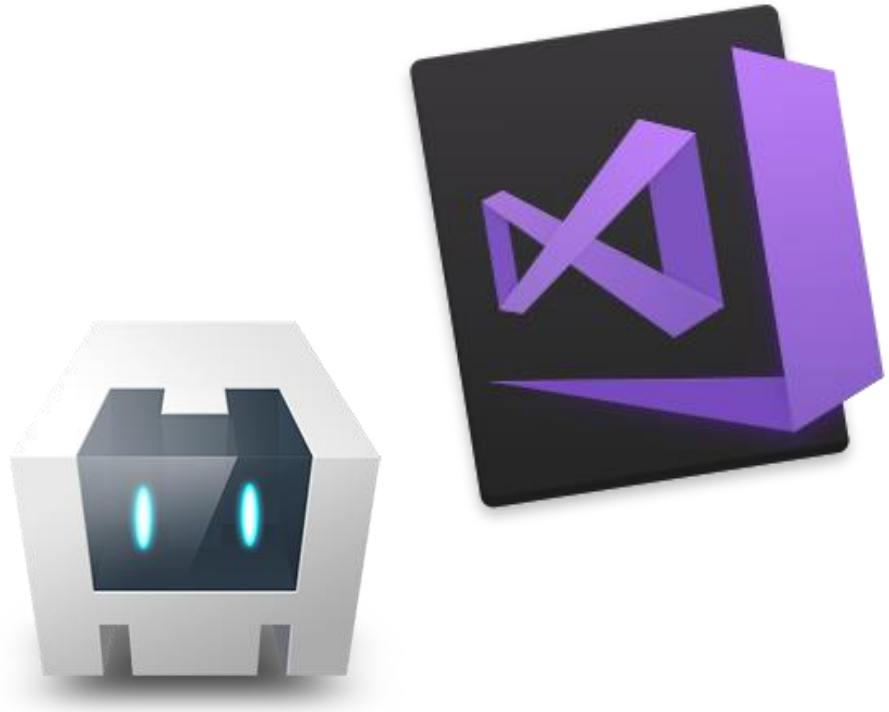
Devices using windows mobile operating system

- 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

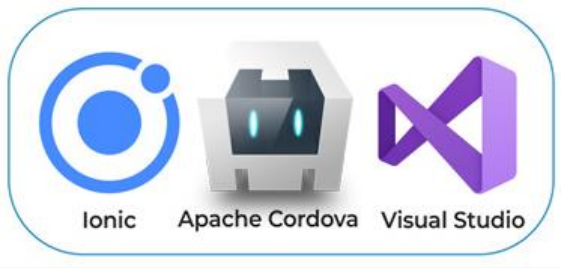
- Visual Studio
- Apache Cordova



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)

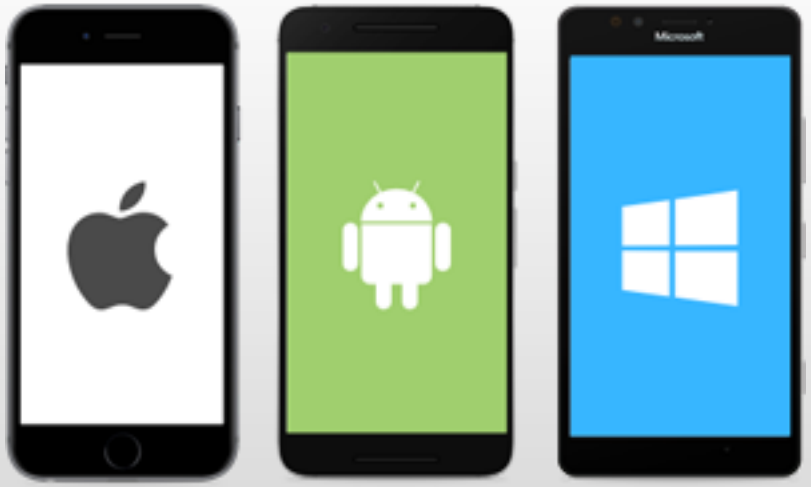
Tools:



Examples

Cross-platform mobile application development

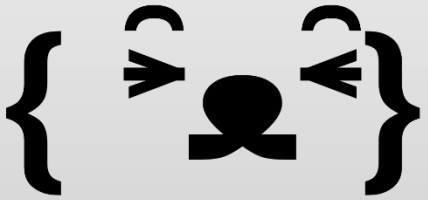
Cross-platform mobile application development refers to the development of mobile apps that can be used on multiple mobile platforms.



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

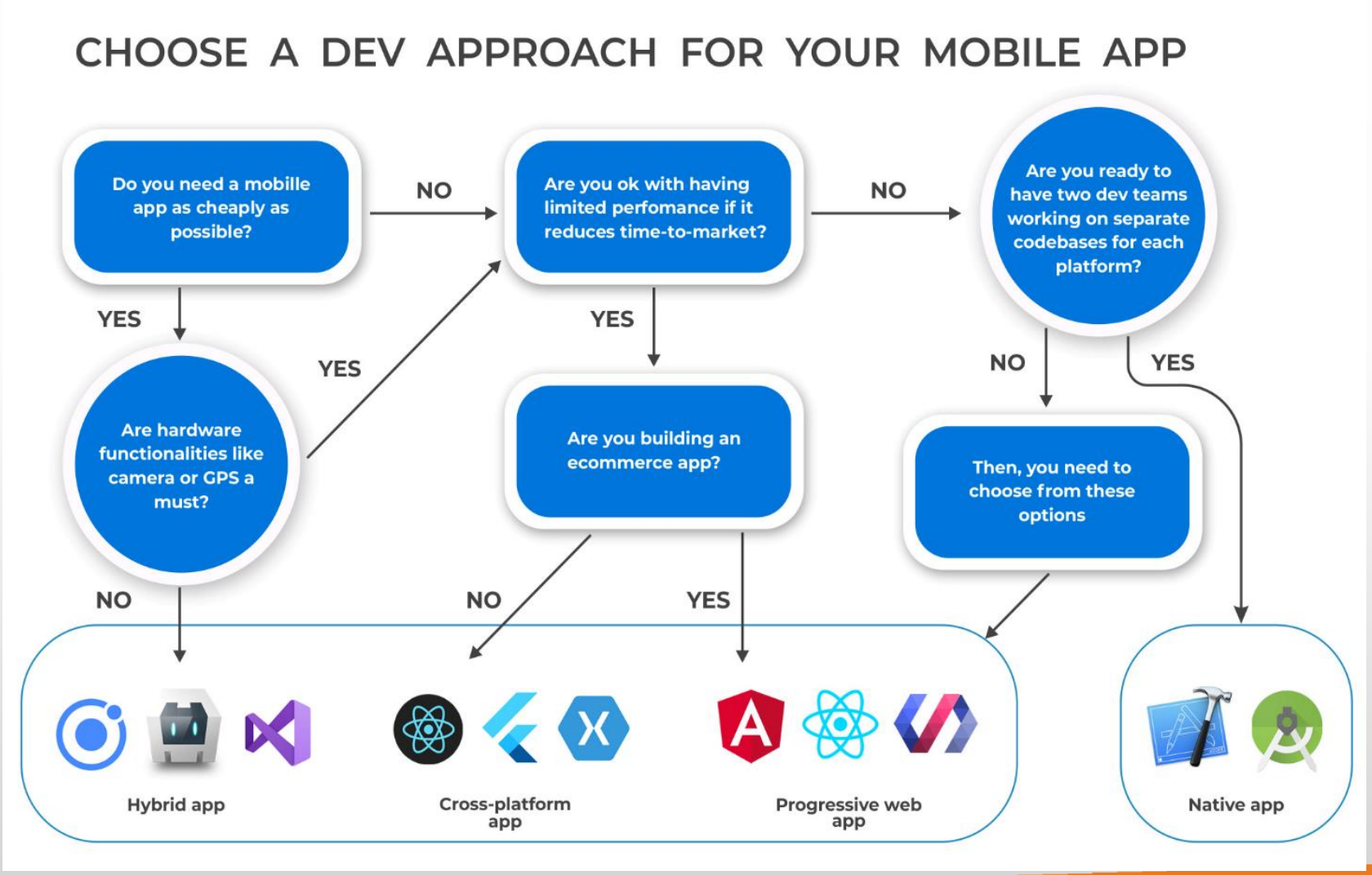
Disadvantages

- 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



Thank You

SLIIT
Discover Your Future

Mobile Application Development – IT2010

Lecture 3 - Mobile Interface Design Concepts and UI/UX Design Fundamentals

"Practice safe design: Use a concept."

-Petrula Vrontikis-
Graphic Designer and Lynda Author

Learning outcomes of the lecture

At the end of this Lecture students will be able to

- Define the terms UI and UX
- List the principles of user interface design
- Categorize different UI components in Android
- Recognize the important of UI Evaluation

Mobile Interface Design

- Interface design add meaning and value to the application
- Design will become attractive if universality design principles are applied.
- Designers face thread when many requirements are unclear and uncertain.

Are the above mentioned two words express the same idea???

Mobile UX – 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.

- Includes all aspects of the end-user's interaction with the company, and its products/services.

Cont'd… - Why UX in mobile?

UX of a mobile application influences how users observe it

Ex:

- Does app provides them value?
- Is the app easy to use?
- Does it help them to fulfill their goal?

Mobile UI – User Interface

- User interface is everything that the user can see and interact with
- In simply terms, “the design of the user interface is not the appearance of a product, but how it works”
- Interface design is the first thing that users will see, therefore, it directly effects the user's view

Cont'd… - Why UI in mobile?

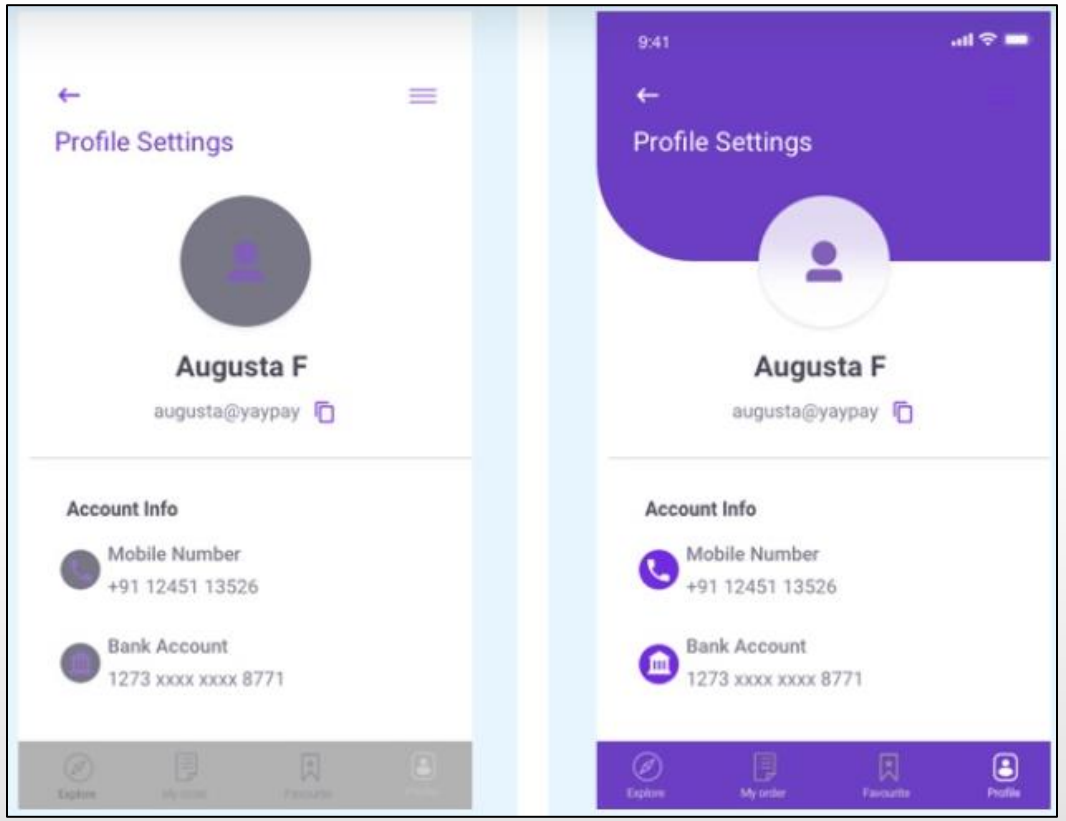
Visual elements greatly impacts an emotional connection with the user

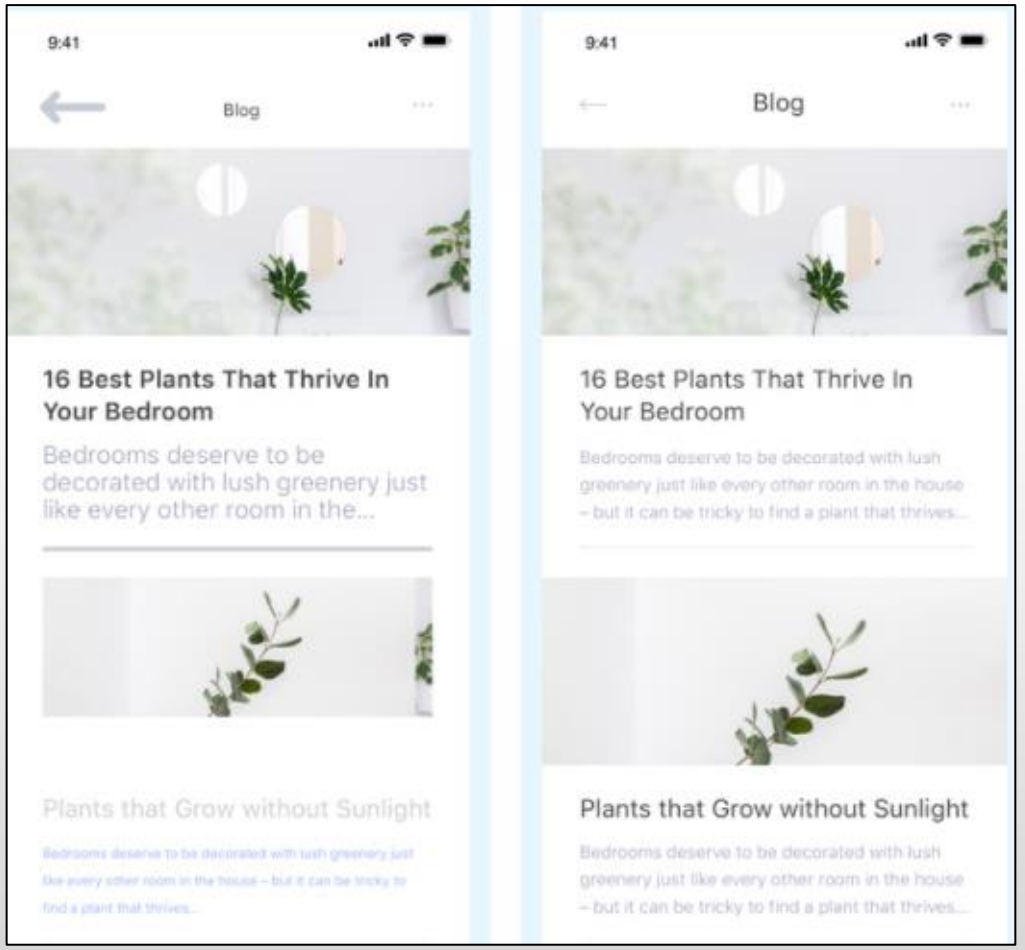
Ex:

- Does the color attract the user?
- Are the elements are placed properly?

Principles of Mobile User Interface Design

Which one is better?





Principles of Mobile Interface Design

Mobile Mindset

- Be Focused
- Be Unique
- Be Charming
- Be Considerate

Mobile Contexts

- Bored
- Busy
- Lost

Global Guidelines

- Responsiveness
- Polish
- Thumbs
- Targets
- Contents
- Controls
- Scrolling

Principles of Mobile Interface Design

Navigation Models

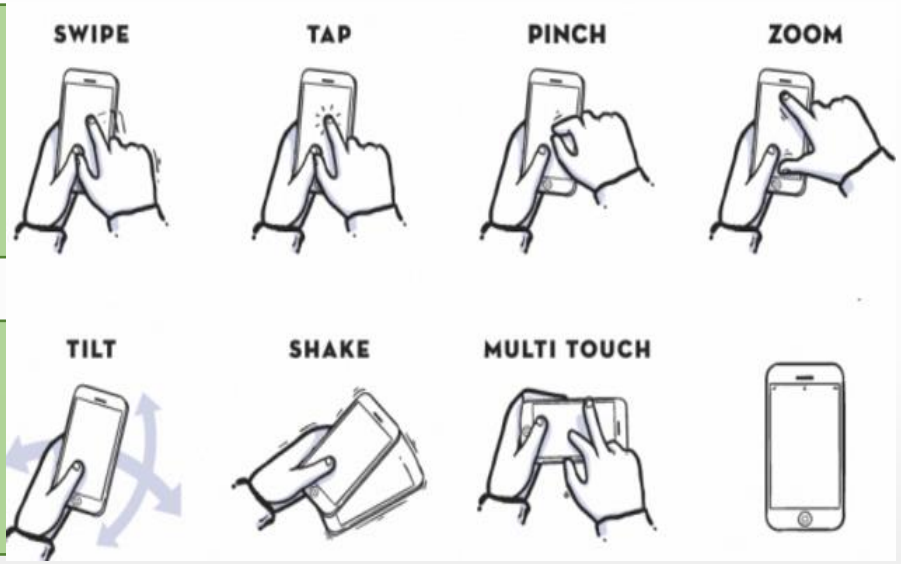
- None
- Tab bar
- Drill down

User Inputs

- Keyboard variations
- Auto correction
- Device Orientation

Gestures

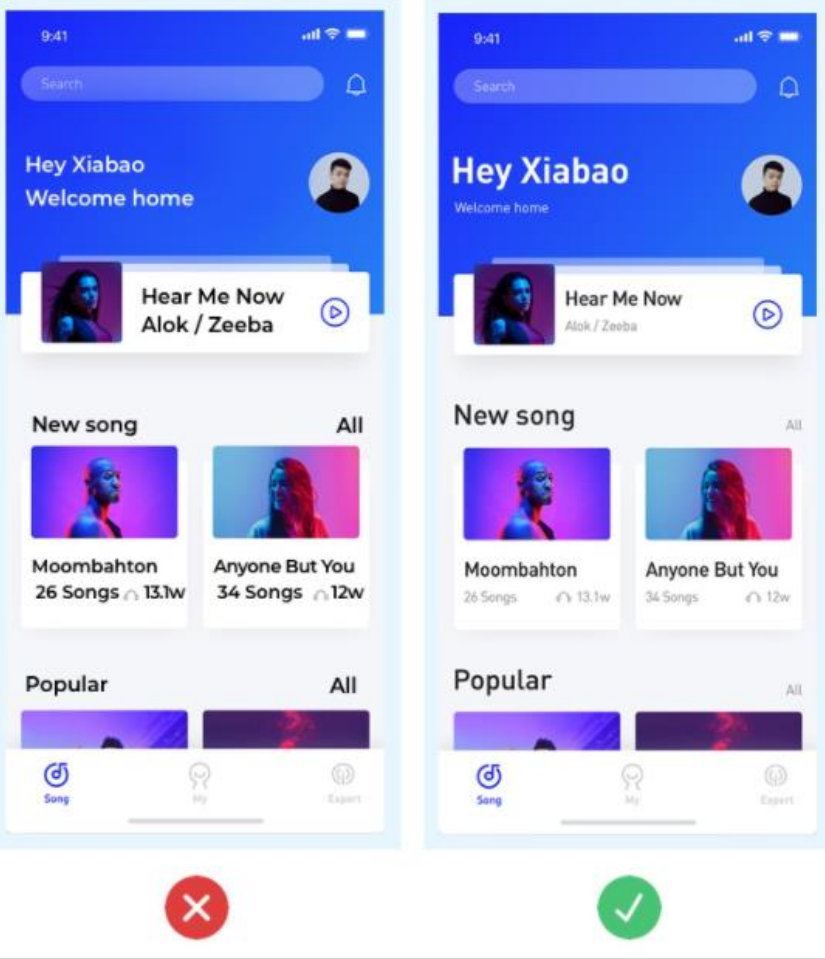
- Invisible
- Two hands
- Nice to have
- No replacement

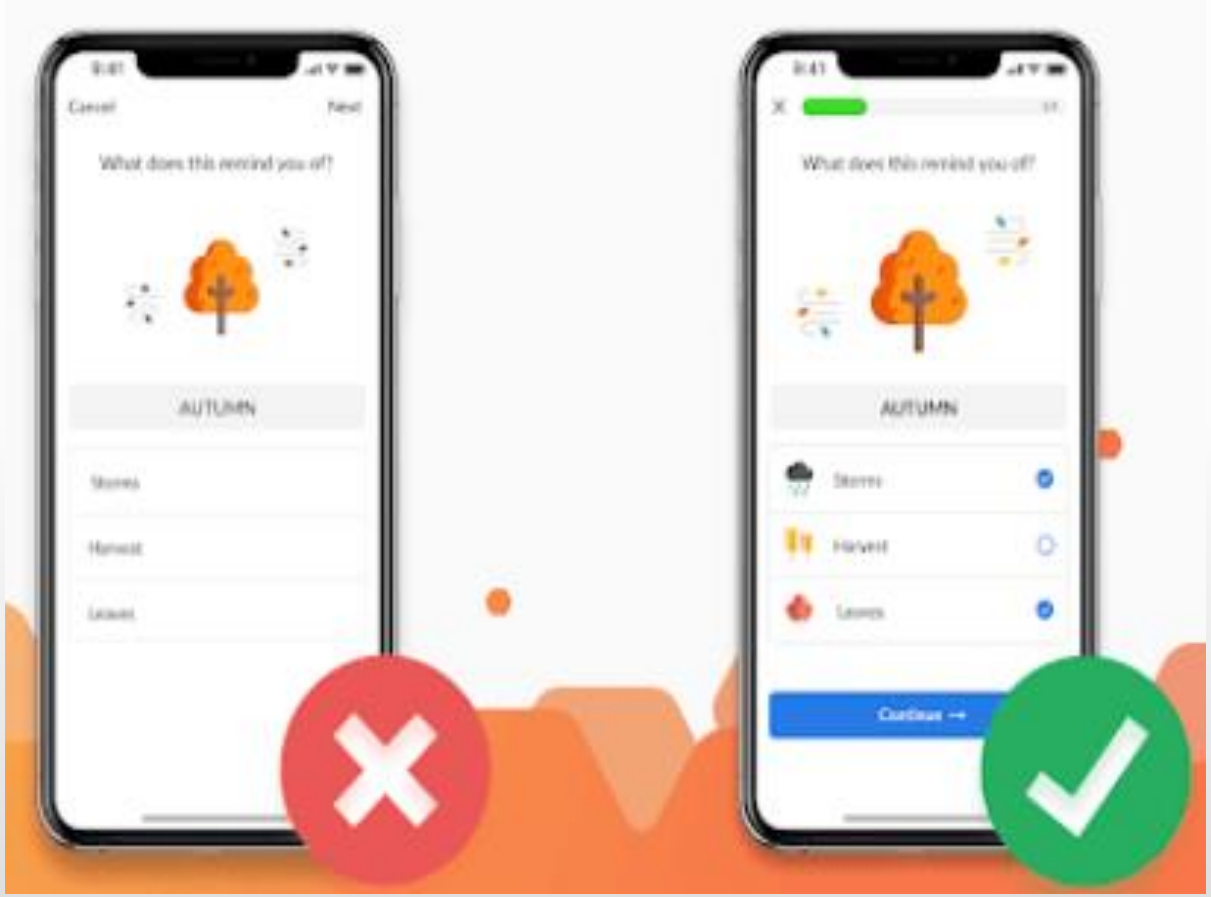


Principles of Mobile Interface Design

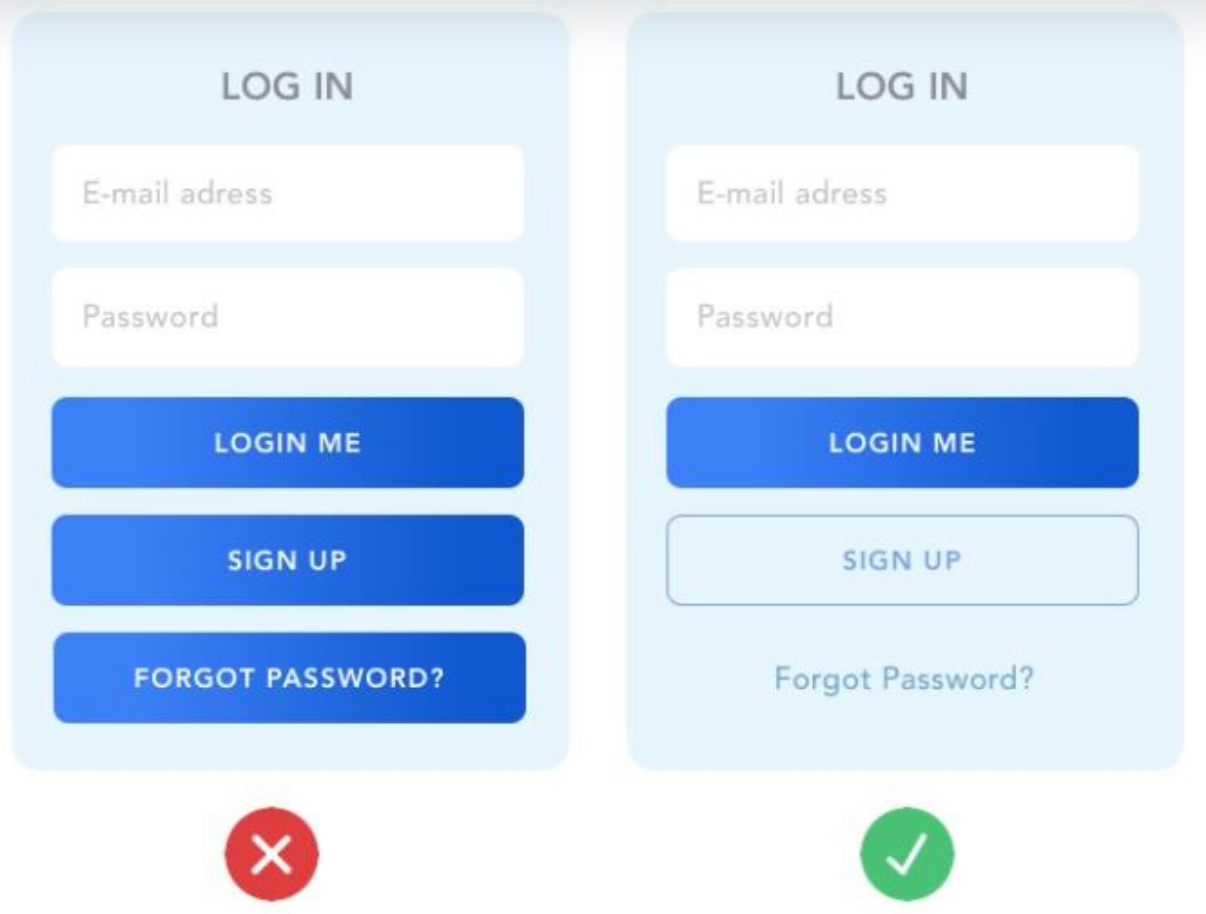
- Orientation
- Communication
  - Provide feedback
  - Model alerts
  - Confirmations
- Launching
- First Impressions
  - Icon
  - First Launch

Text Hierarchy



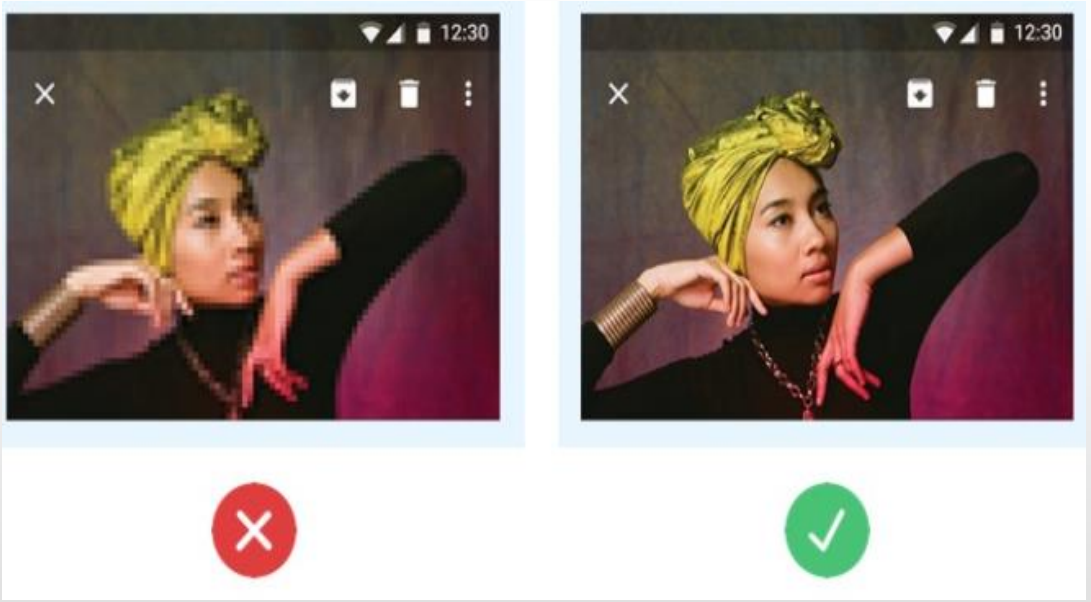


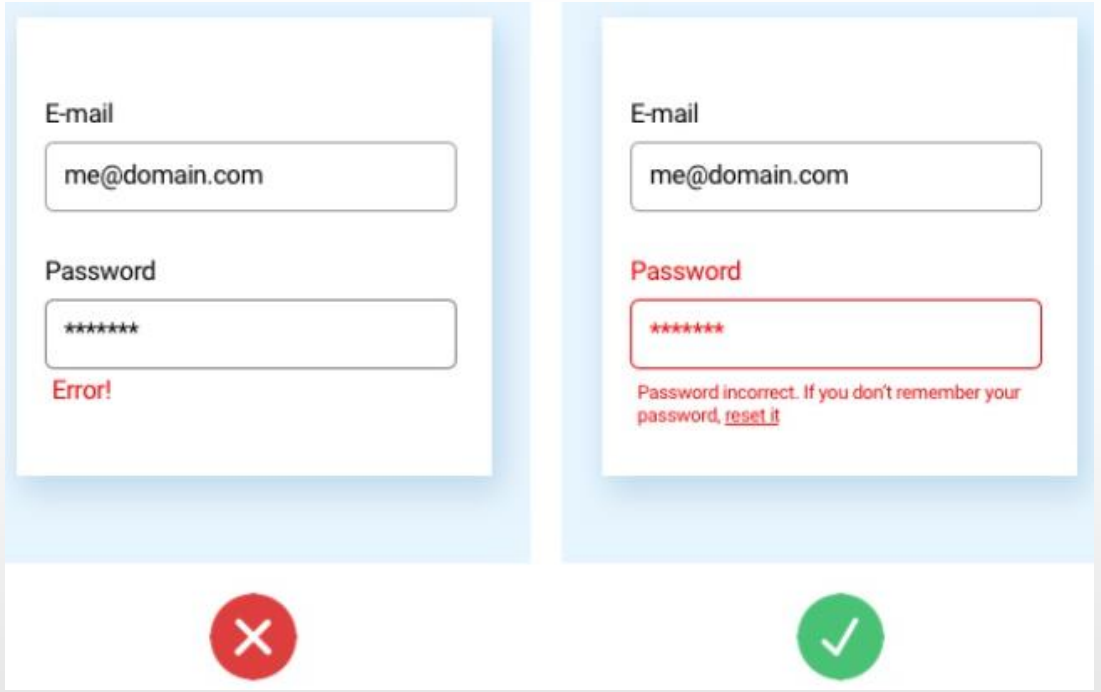
Attractive



- Little distinction between primary and secondary buttons.
  - ✓Different visual weight
  - ✓Strong colors

Low quality images





Confusing forms

Cont'd…

Principles of Mobile Interface Design:

Reference: https://www.youtube.com/watch?v=XS0Qd7hLPhw

Mobile UI Components

Mobile UI components Based on Android

- Android provides a variety of pre-built UI components such as,
  - **Layouts**
  - **Notification**
  - **Menus**
  - **Dialogs**
  - **Toast**

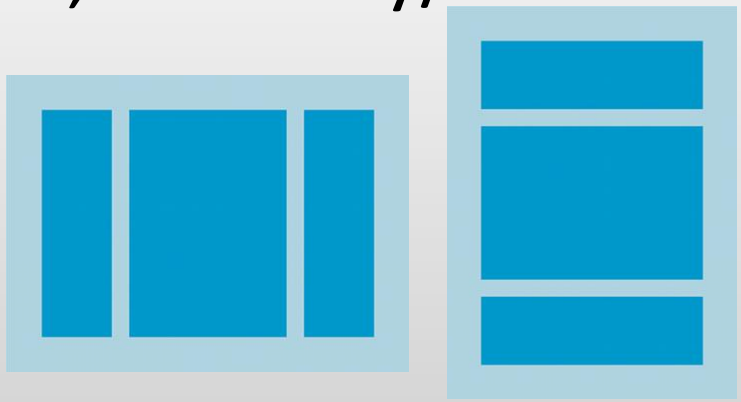
    and etc.
  - **Other common elements** (Buttons, Text fields and etc,)

Cont'd…

- A layout defines the structure for a user interface in your app

  - **Linear Layout**

    This layout aligns all children in a single direction, vertically/horizontal

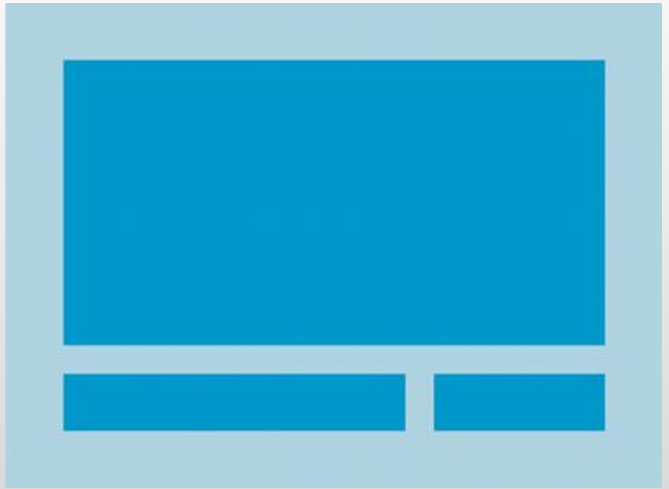


Cont'd…

- **Relative Layout**

  displays child views in relative positions to,

  - Sibling elements
  - Parent



Reference: https://developer.android.com/images/ui/relativelayout.png

Cont'd...

- **Constraint Layout**
  - This layout provide feature to position and size widgets in a flexible way
  - Works similar to relative layout but more flexible than that.

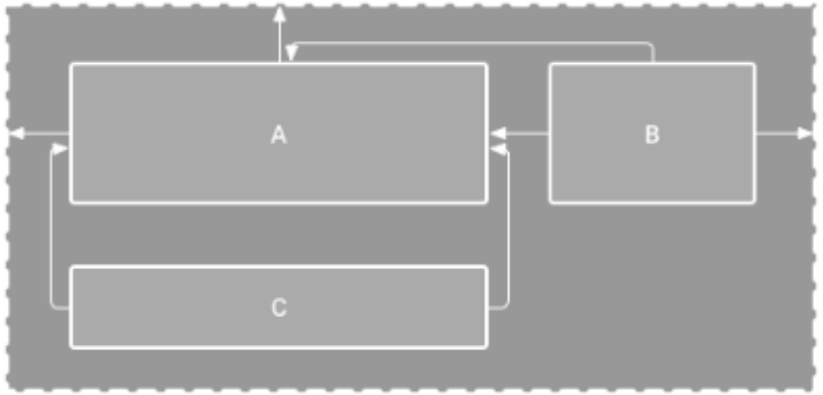


Figure 1. The editor shows view C below A, but it has no vertical constraint

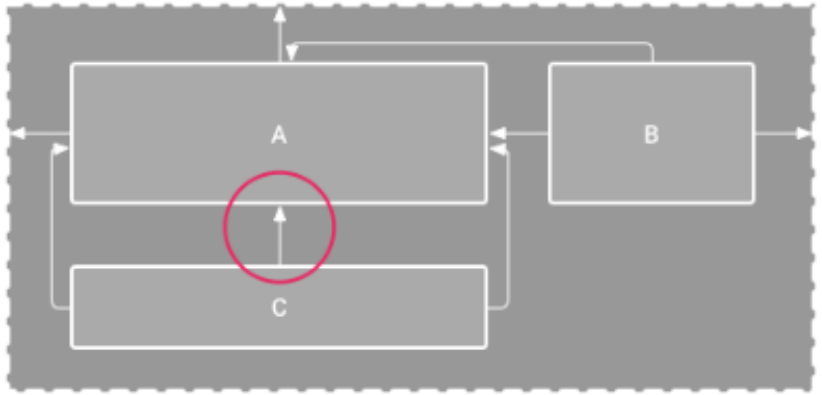


Figure 2. View C is now vertically constrained below view A

Reference: https://developer.android.com/training/constraint-layout/

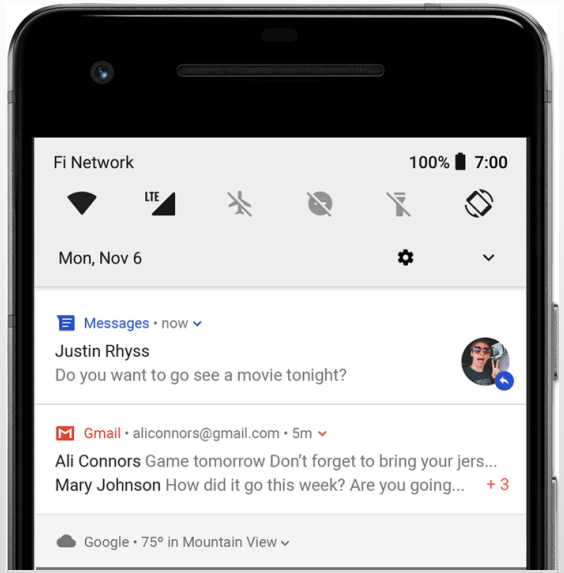
Cont'd…

Other available layouts,

- **Adapter View**
- **Grid View**
- **Table Layout**
- **Absolute Layout**
- **Frame Layout**

Cont'd... Notification

- A message displays outside the app's UI to provide the user with,
  - Reminders
  - Communication from other people
  - Timely information from the app



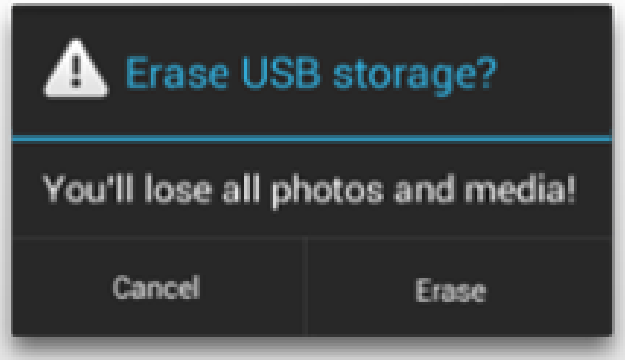
- Users can tap the notification to open an app/take an action directly from the notification

Reference:
https://developer.android.com/images/ui/notifications/notification-drawer_2x.png

Cont'd… Dialogs

- Small window that prompts the user to make a decision before they can proceed.

- Dialog box does not fill the screen



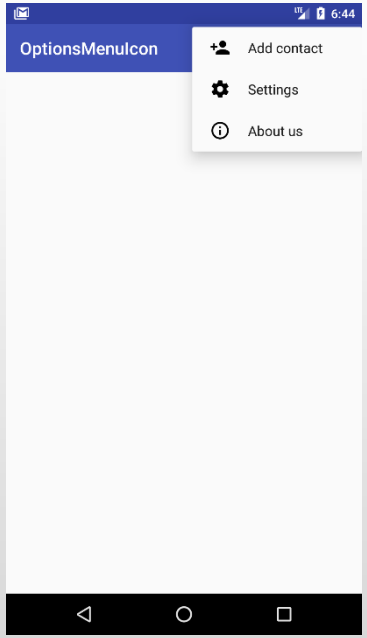
- Consists of subclasses
  - AlertDialog
  - DatePickerDialog/TimePickerDialog

Reference: https://developer.android.com/images/ui/dialogs.png

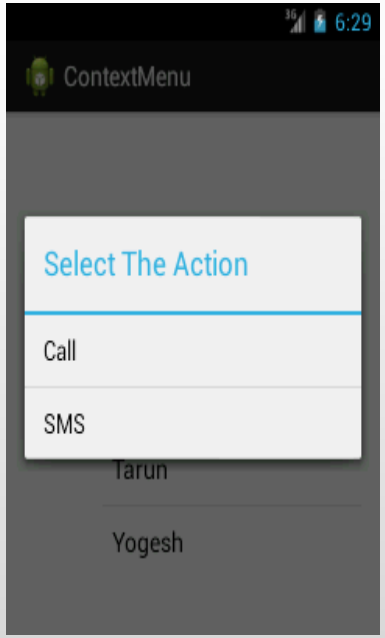
Cont'd… Menus

- This is a common component in many application, there are three standard menus,

Options menu



Context menu

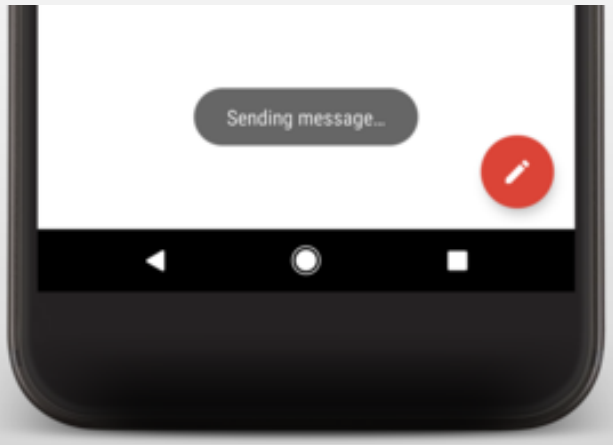


Popup menu

Reference:
http://wptrafficanalyzer.in/blog/wp-content/uploads/2012/07/popup_menu_demo.png
https://www.codingdemos.com/wp-content/uploads/2017/10/Android-Options-Menu-Icon.png

Cont'd... Toasts

- Provides simple feedback about an operation.
- Only uses the space required for the message while the current activity remains visible and interactive.
- Toasts automatically disappear after a timeout.



Reference: https://developer.android.com/images/toast.png

References

1. https://clearbridgemobile.com/mobile-app-design-fundamentals-user-experience-user-interface/

2. https://developer.android.com

3. https://www.tutorialspoint.com

Thank You

Lecture 4/5

Main Components of Android Application

Lecture Plan

- Mobile Mindset
- Mobile Platforms and Application Development fundamentals
- Introduction to Android Operating System
- Android Interface Design Concepts
- Main Components of Android Application
- Sensors and Media Handling in Android Applications
- Data Handling in Android Applications
- Android Web Services
- Kotlin Language to develop Android Mobile Apps
- Android Application Testing and security aspects

Android Core Building Blocks

Android Core Building Blocks

- Make it easy to break them down into conceptual units.
- So that you can work on them independently, and then easily put them back together into a complete package.

Eg: Basic Features

- Login screen
- Main screen
- Play screen

Background tasks

- Network connection
- Caching data

Activity Manager

- ActivityManager class gives information about, and interacts with activities, services, and the containing processes.

- Responsible for creating, destroying and managing activities.

Activity Manager cont..

Eg:

- When the user starts an application for the first time, the Activity Manager will create its activity and put it onto the screen.
- Later, when the user switches screens, it will move that previous activity to a holding place.
- This way, if the user wants to go back to an older activity, it can be started more quickly.
- Older activities that the user hasn't used in a while will be destroyed in order to free more space for the currently active one.
- This mechanism is designed to help improve the speed of the user interface and thus improve the overall user experience.

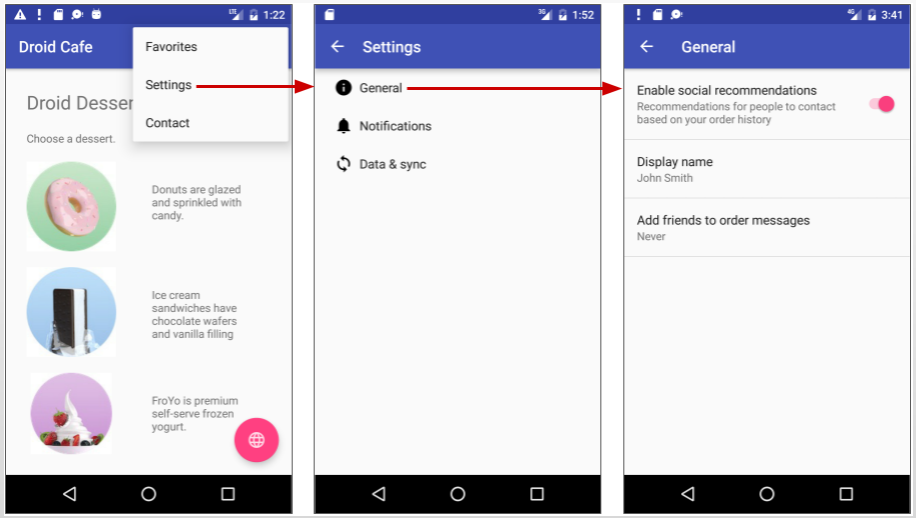
Activities

"A single screen that the user sees on the device at one time. "

- Most visible part of your application
- Just like a website has to provide some sort of navigation among various pages, an Android app should do the same.

Activities cont..

- Launching single activity in Android involves:
  - ➢Creating new Linux process.
  - ➢Allocating memory for al the UI objects.
  - ➢Retrieving the XML Layouts.
  - ➢Setting up the whole screen.



How to create an Activity

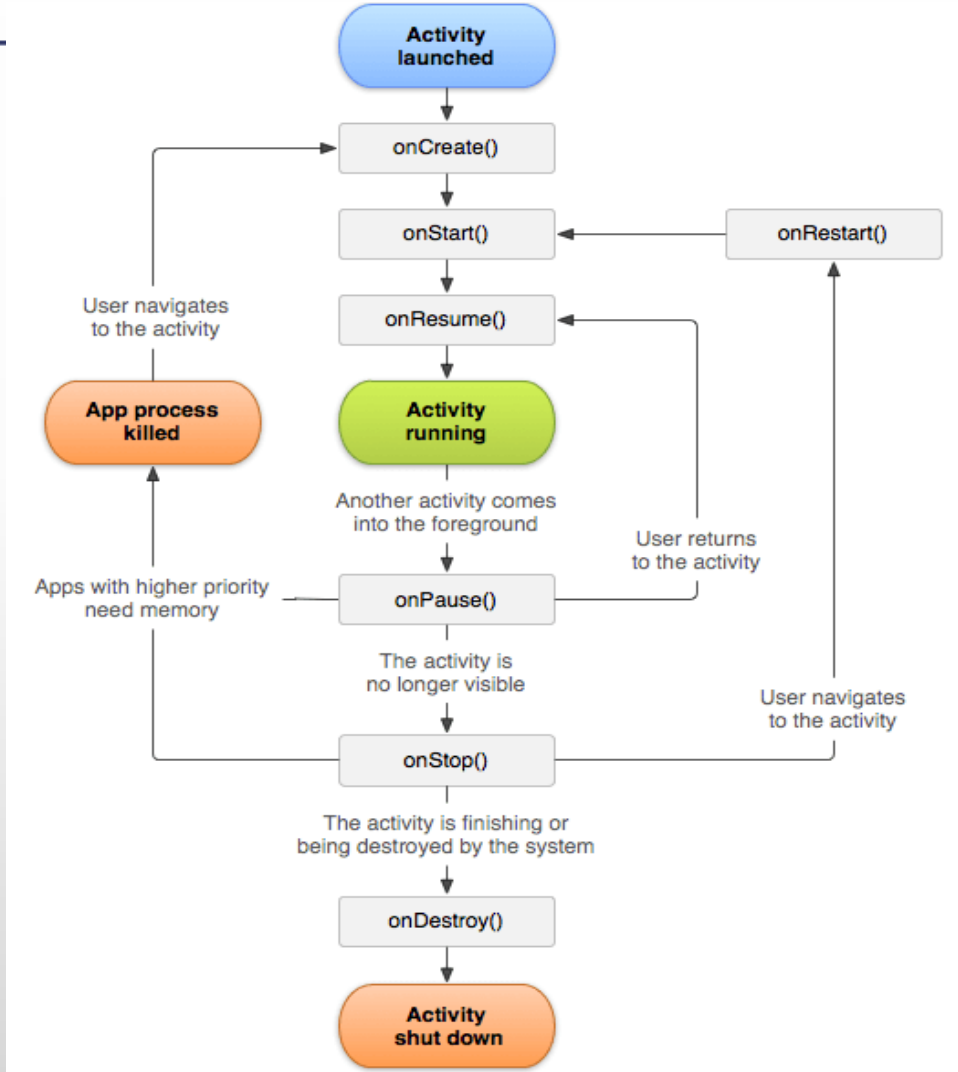
```
package com.example.mad.helloapp;

import android.app.Activity;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends Activity {

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

Activity class loads it's UI component main.xml file

Activity in Manifest file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="net.learn2develop.Activities"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".MainActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="9" />
</manifest>
```

Activity Life Cycle



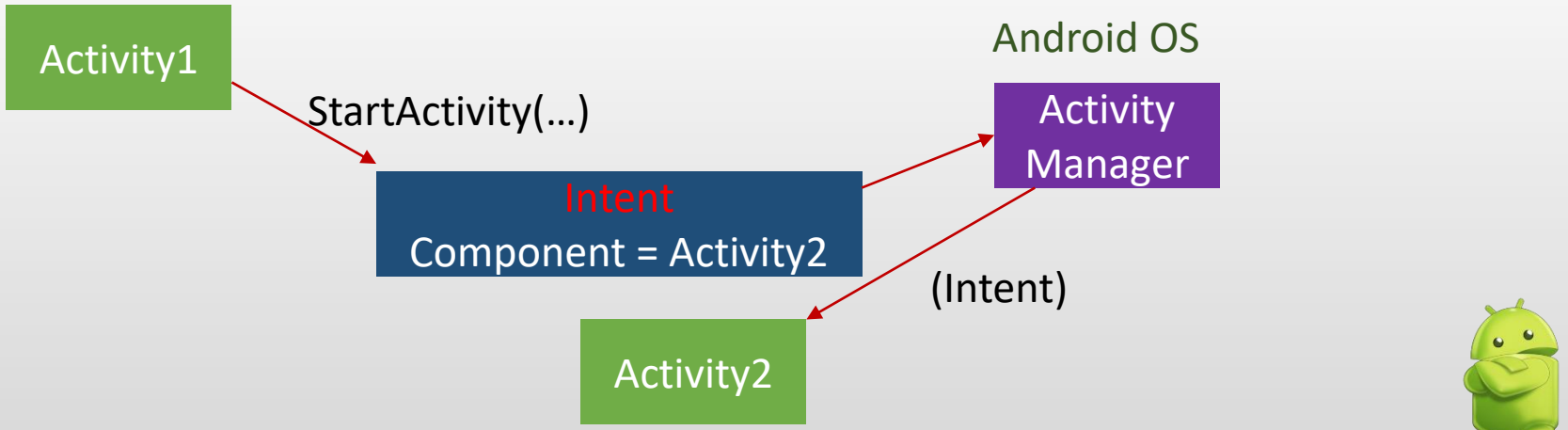
Reference: https://www.javatpoint.com/images/androidimages/Android-Activity-Lifecycle.png

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

Intents

- The message that is passed between activities.
- Intents are asynchronous (the code that sends them doesn't have to wait for them to be completed).
- Allowing us to pass data between activities.



Intents

```
Intent myIntent = new Intent(this, MainMenu.class);
startActivity(myIntent);
```

Note:

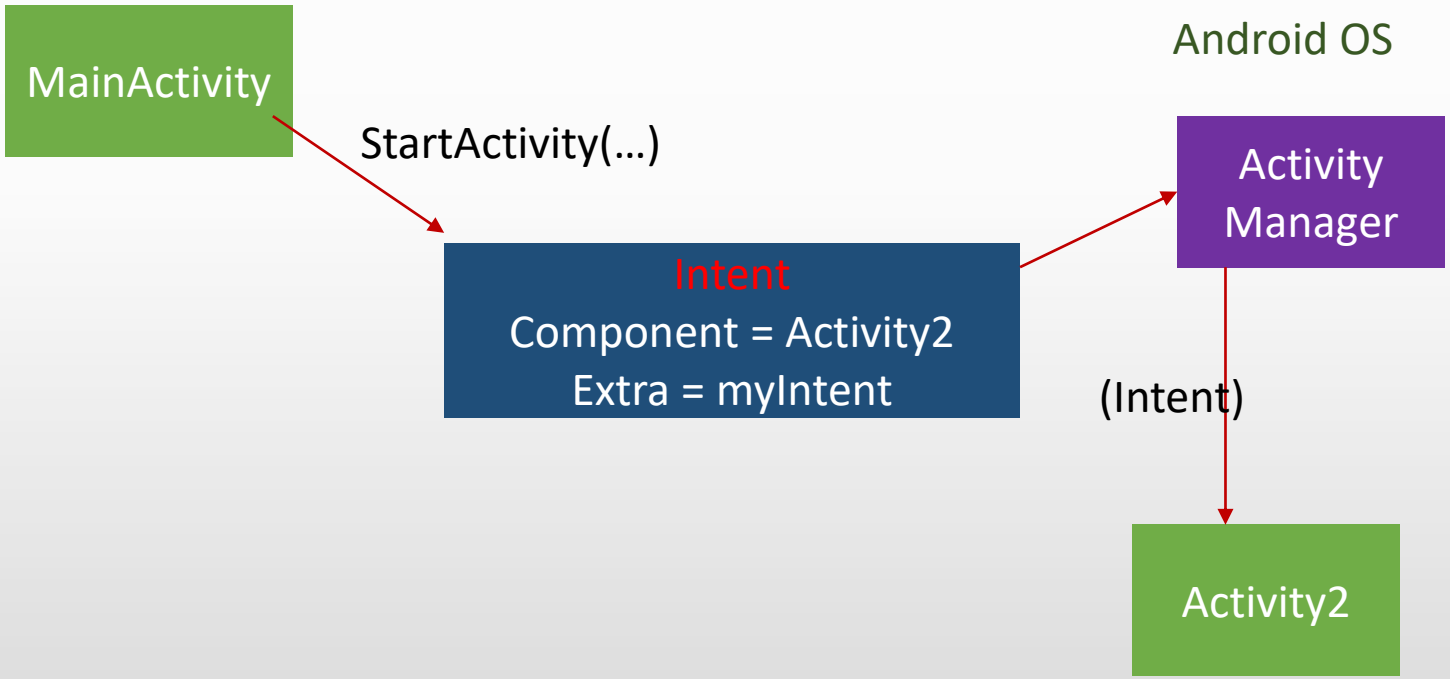
Intent class has a constructor that has two parameters.

1 – reference to the current activity (this)

2 – name of the activity we want to open

Intent Extras

- Passing data between Activities.



Intent Extras

```
public class MainActivity extends Activity {

   public String myExtra = "text";
   Button button1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        button1 = (Button)findViewById(R.id.button1);
        button1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(MainActivity.this, Activity2.class);
                intent.putExtra("MAIN_EXTRA", myExtra);
                startActivity(intent);
            }
        });
    }
}
```

```
public class Activity2 extends AppCompatActivity {

    String takeExtra;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_2);
        Intent myIntent = new Intent();
        takeExtra = myIntent.getStringExtra("MAIN_EXTRA");
    }
}
```

Explicit Intents

- Specifies the component to be invoked from the current activity.
- Can also pass the information from one activity to another.

Implicit Intents

- Doesn't specify the component.
- The sender specifies the type of receiver.

Eg:

if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

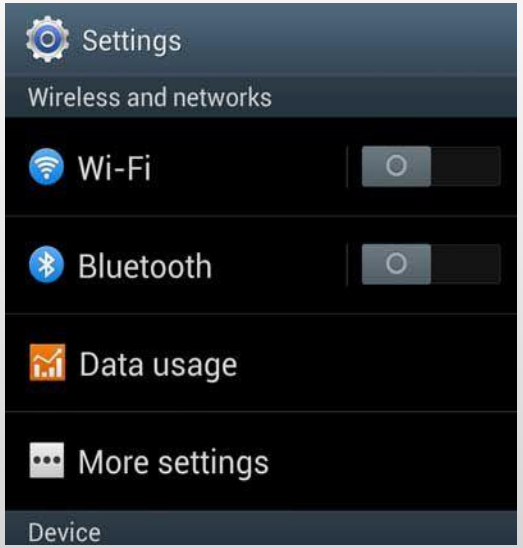
```
Intent intent=new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.google.com"));
startActivity(intent);
```

Android Services

- Services run in the background and don't have any user interface components.
- They can perform the same actions as activities, but without any user interface.
- Services are useful for actions that we want to perform for a while, regardless of what is on the screen.

Eg: - playing music in music player even as you are flipping between other applications

- Handle network transactions



Declare the services in Manifest file

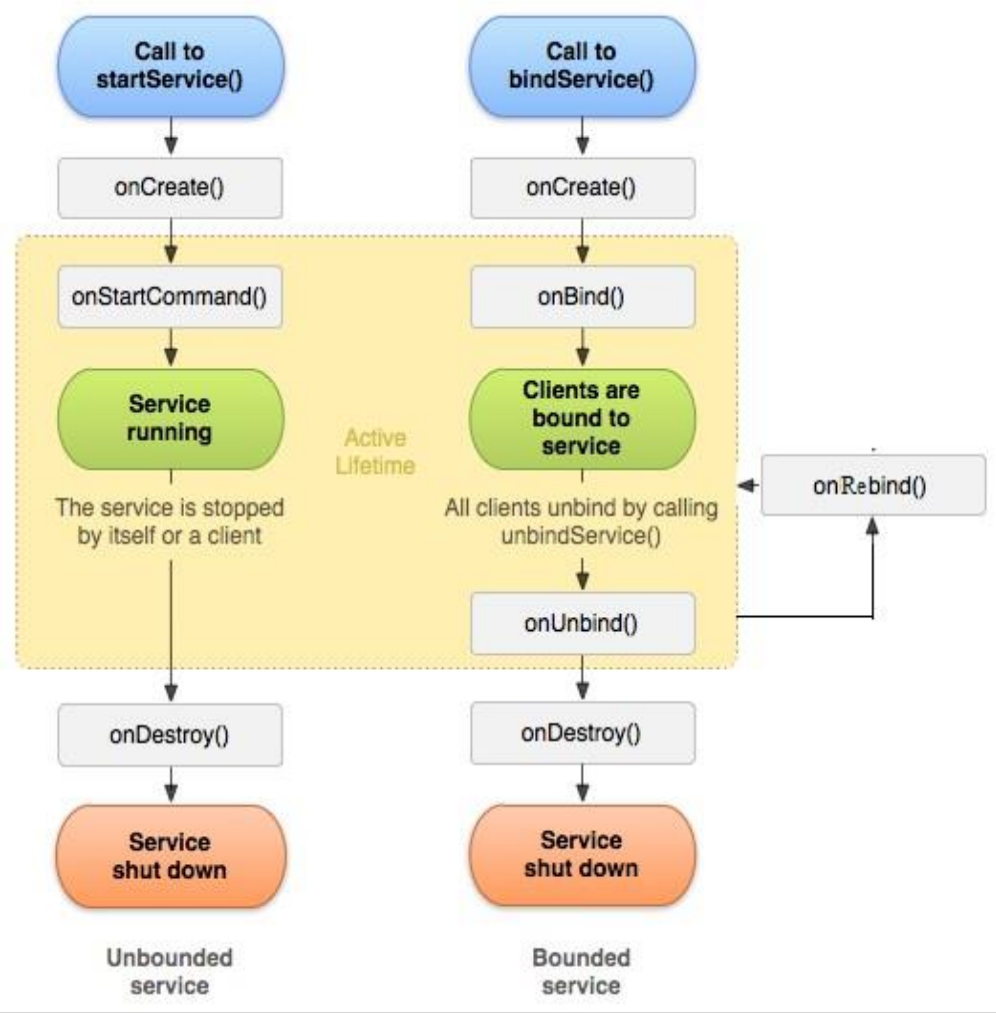
- Declare all services in your application's manifest file.

Note: Add a <service> element as a child of the <application> element.

```
<manifest ... >
  ...
  <application ... >
      <service android:name=".ExampleService" />
      ...
  </application>
</manifest>
```

Android Service Life Cycle

- Two forms of a service
  1. Started
  2. Bound



Android Service Life Cycle cont..

Started Service

- Tells the system *about* something it wants to be doing in the background.

- By calling 'Context.startService()', it asks the system to schedule work for the service, to be run until the service or someone else explicitly stop it.

Android Service Life Cycle cont..

Bound Service

- Application can be exposed some of its functionality to other applications.
- By calling 'Context.bindService()', allows a long-standing connection to be made to the service in order to interact with it.

Broadcast Receivers

- Allows you to register for system or application events. All registered receivers for an event are notified by the Android runtime once this event happens.

- Simply, Broadcast messages from other application or from the system itself.
  - Eg:
    - User can see when it reduces brightness after the battery runs under 20%.
    - Let other applications know that some data has been downloaded to the device and available for them to use.

Broadcast Receivers cont..

- Two important steps:

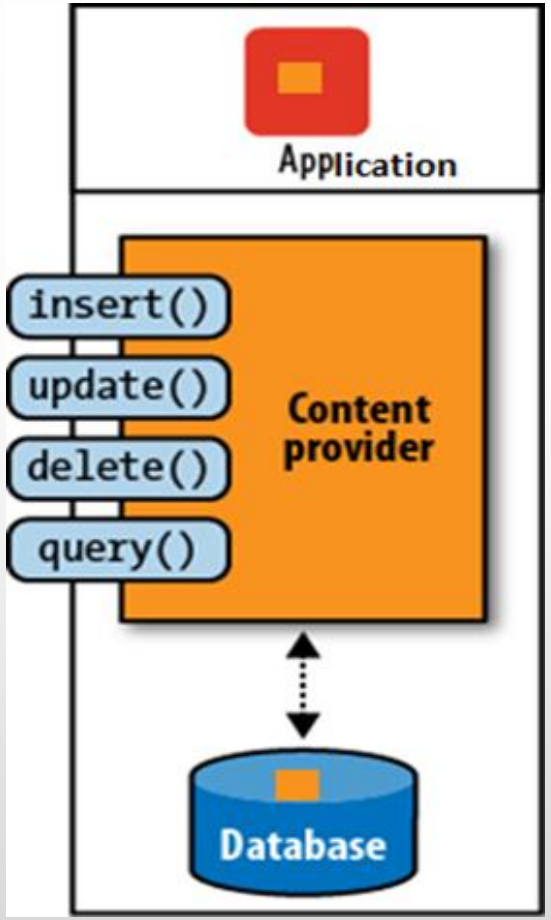
1. Creating the Broadcast Receiver

Extend the BroadcastReceiver class and do the implementation in onReceive() method.

2. Registering Broadcast Receiver

Content Providers

- Interfaces for sharing data between applications
- Behaves like a database.
- Data can be stored in a database, in files, or over a network.
- Let you centralize content in one place and have main different applications access it as needed.
- Implemented as a subclass of ContentProvider class.



Content Providers

- Manages several kinds of data such as audio, video, images, and personal contact information.

- Can be used to manage both structured data(Eg: SQLite relational databases) and unstructured data (Eg: image files).

Content Providers -Advantages

- Control the permission for accessing data.

- Abstract away the details for accessing different data sources in your application.

  Eg:

  your application might store structured records in a SQLite database, as well as video and audio files. You can use a content provider to access all of this data, if you implement this development pattern in your application

Create Content Provider

1. Create a Content Provider class that extends the *ContentProviderbaseclass.*

2. Define your content provider URI address which will be used to access the content.

3. Create your own database to keep the content. Usually, Android uses SQLite database and framework needs to override *onCreate()* method which will use SQLite Open Helper method to create or open the provider's database. When your application is launched, the *onCreate()* handler of each of its Content Providers is called on the main application thread.

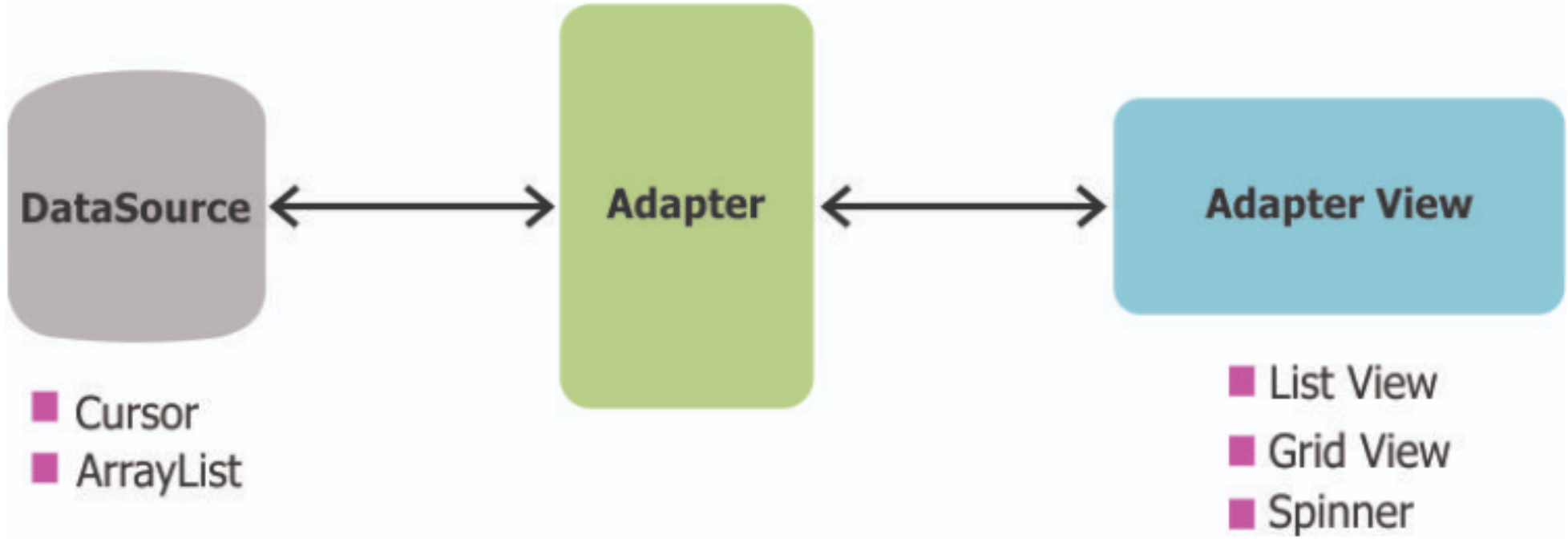
Create Content Provider cont..

4. Implement Content Provider queries to perform different database specific operations.

5. Register your Content Provider in your activity file using <provider> tag.

Overridden methods for Content Providers

- **onCreate()** - Called when the provider is started.
- **query()** - Receives a request from a client. The result is returned as a Cursor object.
- **insert()** - Inserts a new record into the content provider.
- **delete()** - Deletes an existing record from the content provider.
- **update()** - Updates an existing record from the content provider.
- **getType()** - Returns the MIME type of the data at the given URI.


DataSource
Adapter
Adapter View
Cursor
ArrayList
List View
Grid View
Spinner

Thank you!

SLIIT
Discover Your Future

Mobile Application Design and Development

Lecture 4 - Introduction to Android Operating System

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

-Andy Rubin-
Co-funder of Android

Learning Outcomes of the Lecture

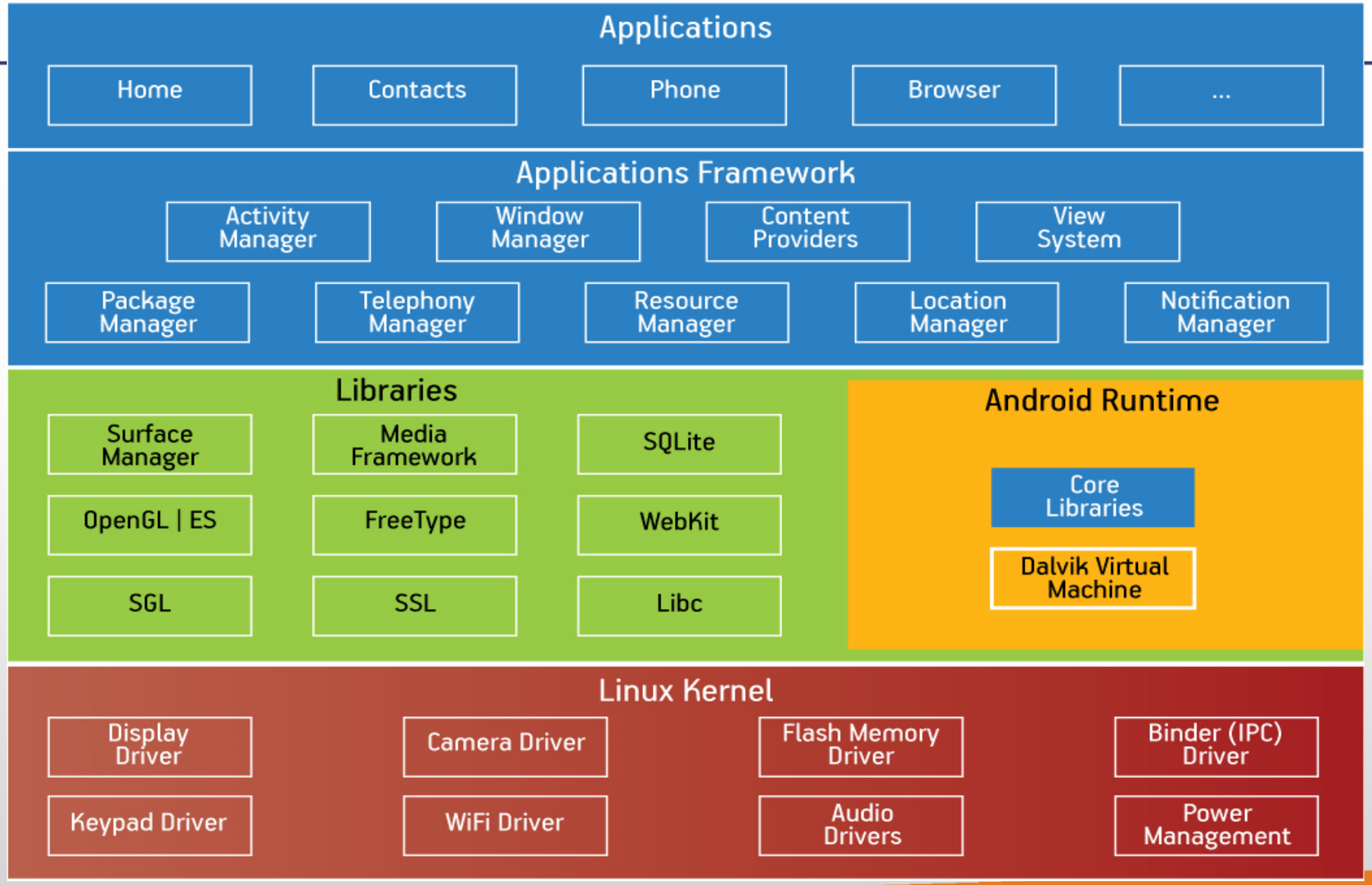
At the end of this Lecture students will be able to

- 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 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.

Cont'd...

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 Run Time

Today, the Android use ART because,

- Compilation Approach
- Boot time
- Space usage
- Fast

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

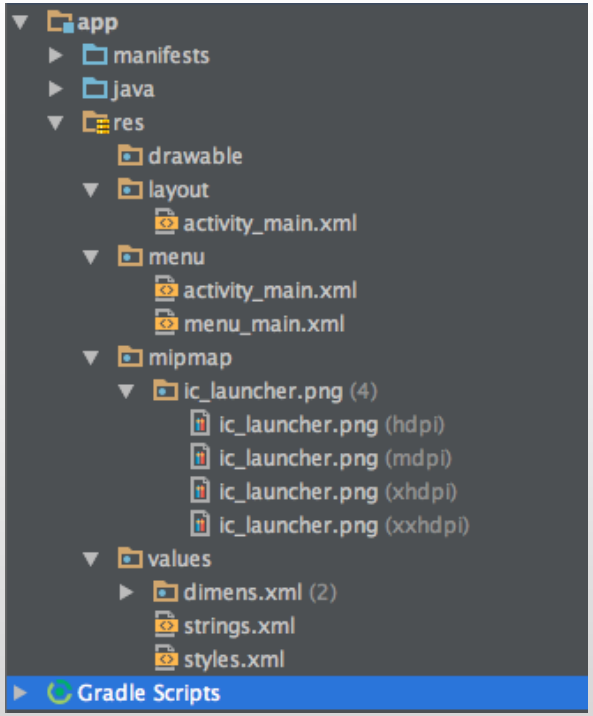
Mobile Application Development Life cycle

Cont'd

Video Reference: 6 Steps of Mobile App Development Lifecycle.mp4

Reference: https://www.youtube.com/watch?v=z3NsfhqAmnA

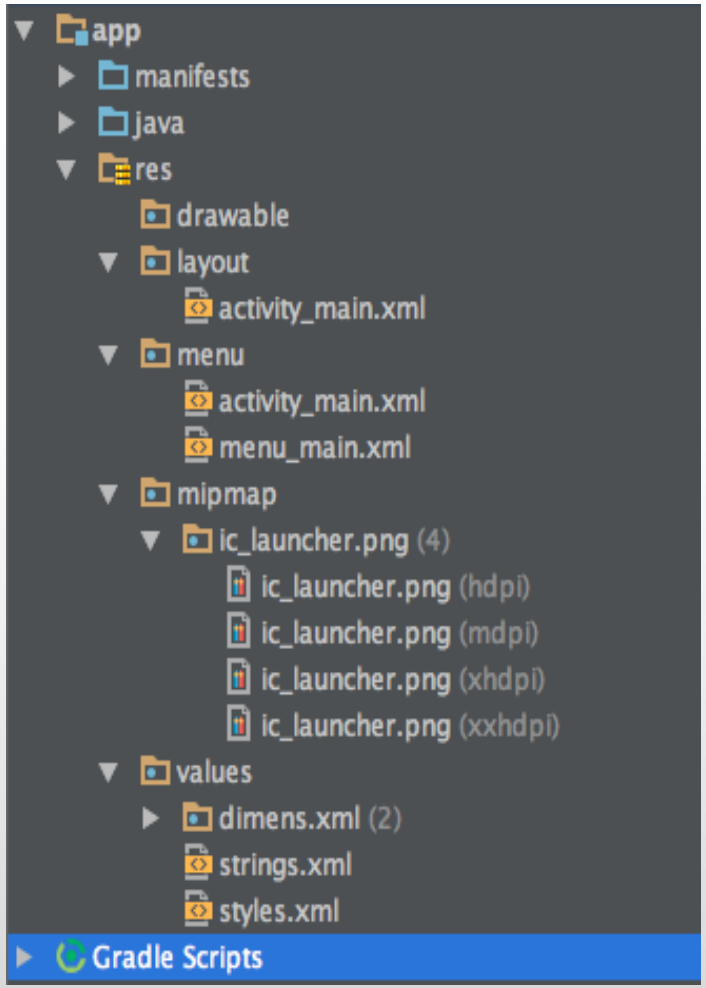
Android Application Project Structure



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



Reference: https://developer.android.com/images/tools/projectview-p2.png

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

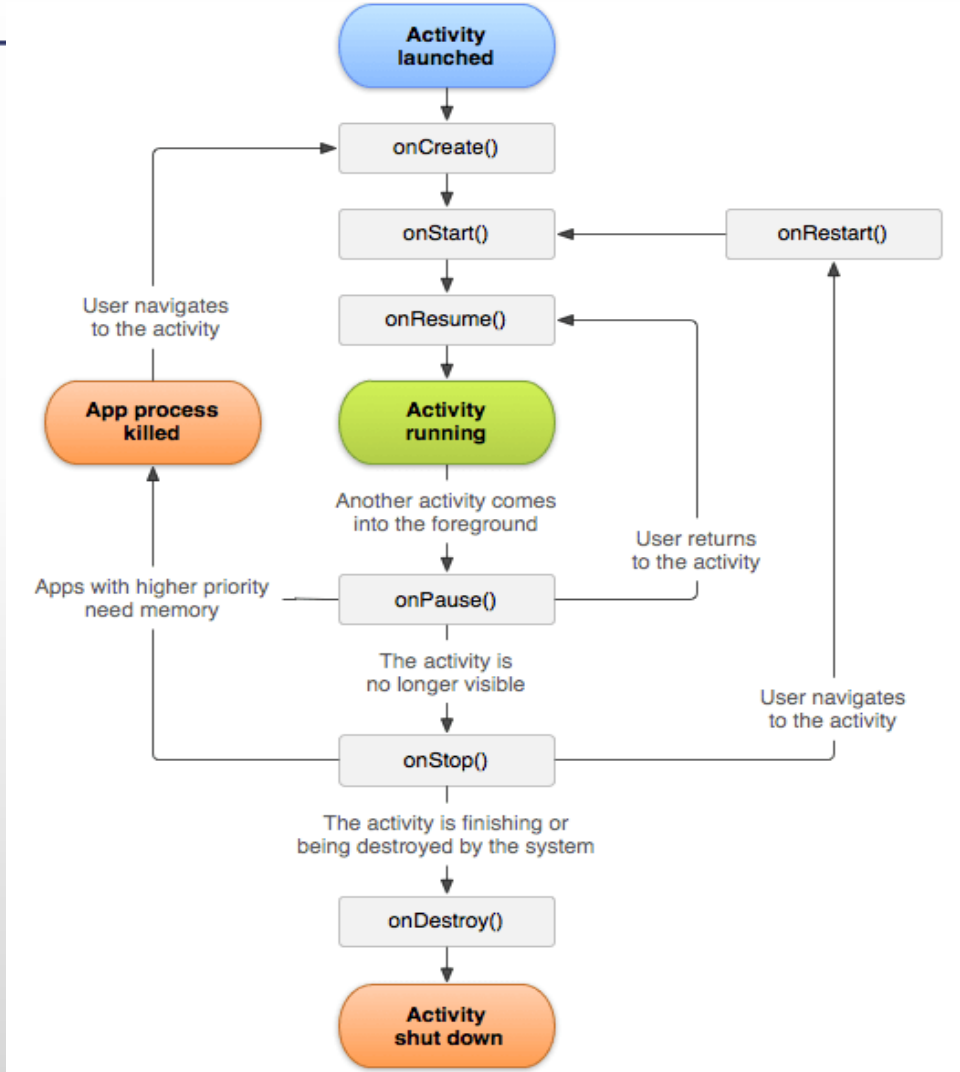
- Resource file that contains resource IDs for all the resources of res/ directory.
- This is an abstraction between different resources (XML file, any UI component [icon], audio & etc. and the java file)
- Auto generated file by AAPT (Android Asset Packaging Tool).

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

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

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!!!

Lecture 6

Data Handling in Mobile Platforms

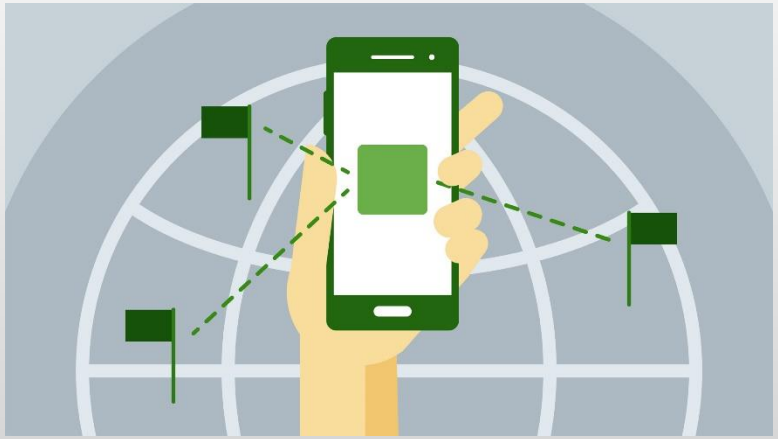
- Mobile Mindset
- Mobile Platforms and Application Development fundamentals
- Introduction to Android Operating System
- Android Interface Design Concepts
- Main Components of Android Application
- Sensors and Media Handling in Android Applications
- Data Handling in Android Applications
- Kotlin Language to develop Android Mobile Apps
- Android Application Testing and security aspects

Learning Outcomes

- At the end of this Lecture, students should be able to
  - ✓ Identify the persistence techniques in Android Applications.
  - ✓ Understand the database handling of mobile technology using SQLite as the database tool.
  - ✓Develop mobile applications using SQLite as the database.

Persistence techniques in Android Applications

- Android provides several options to save your app data.
- Your solution depends on:
  - How much space your data requires
  - What kind of data you need to store
  - Data should be private to your app or accessible to the other apps



File storage options

1. **Internal file storage** - Store app-private files on the device file system.

2. **External file Storage** - Store files on the shared external file system. This is usually for shared user files, such as photos.

3. **Shared preferences** - Store private primitive data in key-value pairs.

4. **Database** - Store structured data in a private database.

Internal Storage

- By default, files saved to internal storage are private to your app.
- The system provides a private directory on the file system for each file.
- When the user uninstalls your app, the files saved on the internal storage are removed.

External Storage

- A storage space that users can mount to a computer as an external storage device.

- Physically removable.

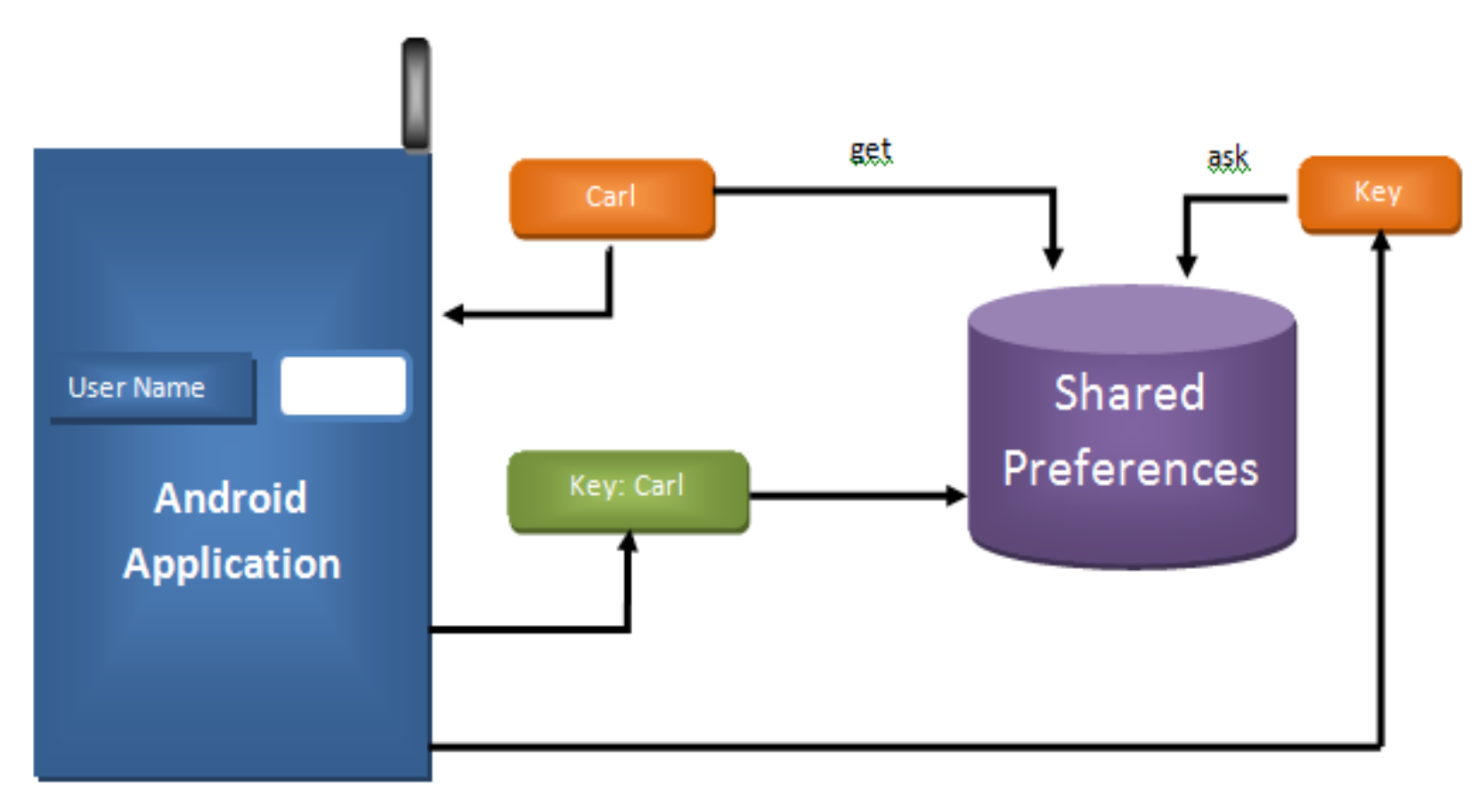
  Eg: SD card

- Can be modified by the user when they enable USB mass storage to transfer files on a computer.

Shared preferences

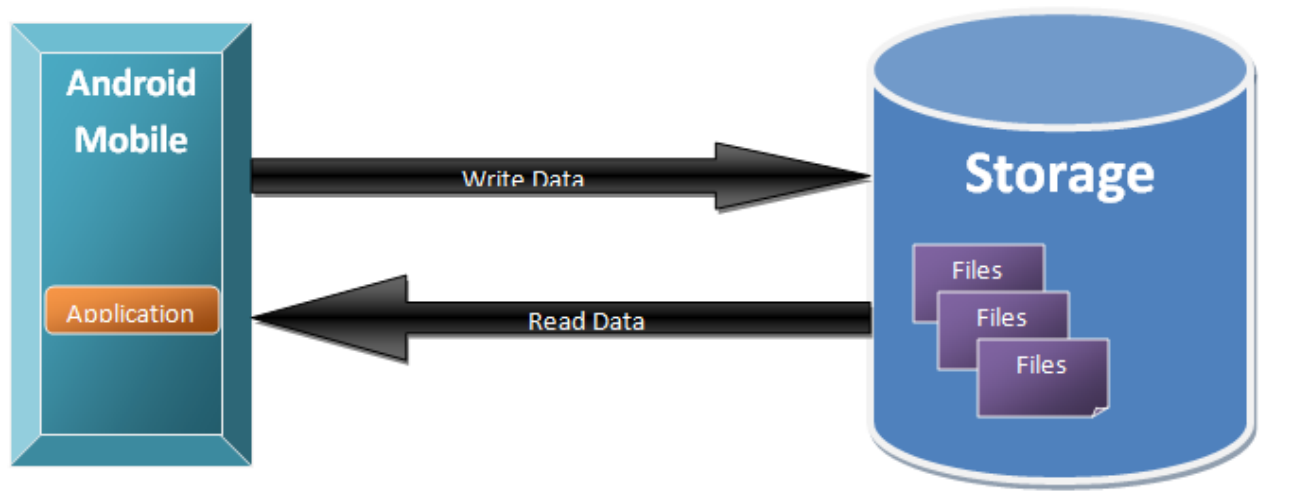
- If you don't need to store a lot of data and it doesn't require structure, you can use Shared preferences.
- The APIs allow you to read and write persistent key-value pairs of primitive data types such as Booleans, floats, ints, longs, and strings.
- The key-value pairs are written to XML files that 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.

Shared preferences



Databases

- Android provides full support for SQLite databases.
- Any database you create is accessible only by your app.



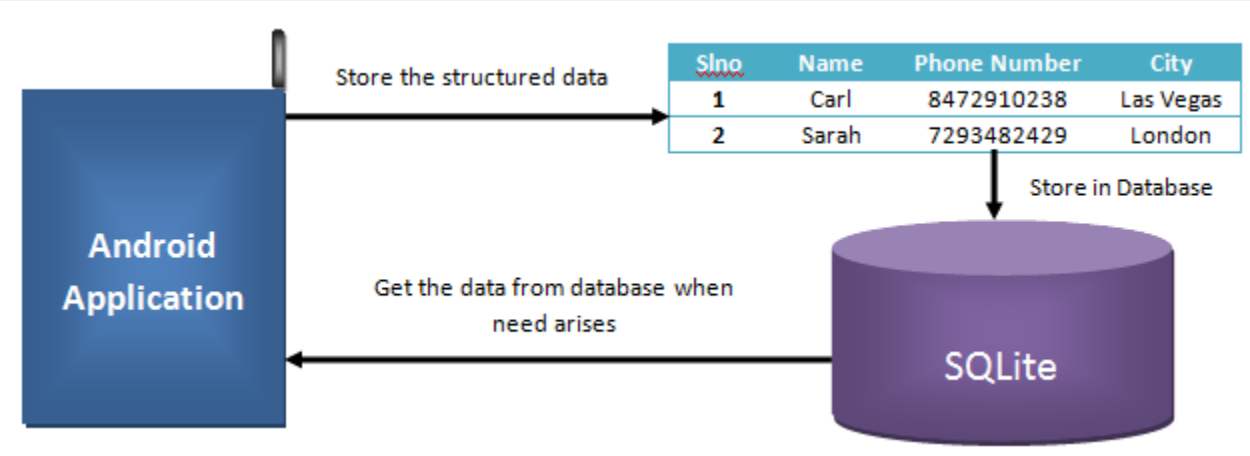
Popular Mobile App Databases

- MySQL
- PostgreSQL
- Redis
- MongoDB
- Memcached
- MariaDB
- SQLite
- InfluxDB
- RethinkDB
- Riak DB
- CouchDB
- Couchbase
- ArangoDB

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.



Why SQLite?

- Serverless (does not require a separate server / ODBC or JDBC queries or system to operate). So. It's a local database.
- Zero-configuration (no setup or administration needed).
- Stored in a single cross-platform disk file.
- Very small and light-weight (250 KB- 400KB).
- Self-contained (no external dependencies and embeddable.
- Includes all basic functionalities.

Advantages of SQLite

- **Portable –** Uses only ANSI-standard C and VFS, file format is cross platform (little vs big endian, 32 vs 64 bit)
- **Reliable –** Has 100% test coverage, open source code and bug database, tranactions are ACID even if power fails
- **Small –** 300kb library, runs in 16kb stack and 100kb heap

Disadvantages of SQLite

- **High concurrency –** Reader / writer loacks on the entire file
- **Huge database –** Db file can’t exceed file system limit or 2TB
- **Access control –** there isn’t any

Create Database

- The SQLiteDatabse and SQLiteOpenHelper libraries are required.
- They include necessary information for the Db handling.

```
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
```

- Create a java class with above two imports and extend the SQLiteOpenHelper class. Implement thenecessary constructors and required onCreate() and onUpgrade() methods.

Create Columns for a Table

- Create a final class to define all tables for the database.
- Create an inner class to define columns.

```
import android.provider.BaseColumns;

public final class UsersMaster {
    private UsersMaster() {}

    /* Inner class that defines the table contents */
    public static class Users implements BaseColumns {
        public static final String TABLE_NAME = "users";
        public static final String COLUMN_NAME_USERNAME = "username";
        public static final String COLUMN_NAME_PASSWORD = "password";
    }
}
```

Create Database

```
public class DBHelper extends SQLiteOpenHelper {

    public static final String DATABASE_NAME = "UserInfo.db";

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

    @Override
    public void onCreate(SQLiteDatabase db) {
        String SQL_CREATE_ENTRIES =
                "CREATE TABLE " + UsersMaster.Users.TABLE_NAME + " (" +
                        UsersMaster.Users._ID + " INTEGER PRIMARY KEY," +
                        UsersMaster.Users.COLUMN_NAME_USERNAME + " TEXT," +
                        UsersMaster.Users.COLUMN_NAME_PASSWORD + " TEXT)";
        // Use the details from the UsersMaster and Users classes we created. Specify the primary key from the BaseColumns

        db.execSQL(SQL_CREATE_ENTRIES);  // This will execute the contents of SQL_CREATE_ENTRIES
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

    }
```

Insert Data

```
public void addInfo(String userName, String password) {
    // Gets the data repository in write mode
    SQLiteDatabase db = getWritableDatabase();

    // Create a new map of values, where column names the keys
    ContentValues values = new ContentValues();
    values.put(Users.COLUMN_NAME_USERNAME, userName);
    values.put(Users.COLUMN_NAME_PASSWORD, password);

    // Insert the new row, returning the primary key value of the new row
    long newRowId = db.insert(Users.TABLE_NAME, nullColumnHack: null, values);
}
```

Read data from the Database

```
public List readAllInfo()
{
    SQLiteDatabase db = getReadableDatabase();

    // define a projection that specifieswhich columns from the database
    // you will actually use after this query
    String[] projection = {
            Users._ID,
            Users.COLUMN_NAME_USERNAME,
            Users.COLUMN_NAME_PASSWORD
    };
    //Filter results WHERE "userName" = 'SLIIT USER'
   // String selection = Users.COLUMN_NAME_USERNAME + " = ?";
    //String[] selectionArgs = {""};

    // How you want the results sorted in the resulting cursor
    String sortOrder = Users.COLUMN_NAME_USERNAME + " DESC";

    Cursor cursor = db.query(
            Users.TABLE_NAME,             // the table to query
            projection,                   // the columns to return
            selection: null,              // the columns for the WHERE clause
            selectionArgs: null,          // the values for the WHERE clause
            groupBy: null,               // don't group the rows
            having: null,                // don't filter by row groups
            sortOrder                    // the sort order
    );
```

Read data from the Database cont..

```
    List userNames = new ArrayList<>();
    List passwords = new ArrayList<>();

    while(cursor.moveToNext()){
        String username = cursor.getString( cursor.getColumnIndexOrThrow(Users.COLUMN_NAME_USERNAME));
        String password = cursor.getString( cursor.getColumnIndexOrThrow(Users.COLUMN_NAME_PASSWORD));
        userNames.add(username);
        passwords.add(password);
    }
    cursor.close();
    return userNames;
}
```

Update a record

```
public void updateInfo(String userName, String password) {
    SQLiteDatabase db = getReadableDatabase();

    //New value for one column
    ContentValues values = new ContentValues();
    values.put(Users.COLUMN_NAME_PASSWORD, password);

    //Which row to update, based on the title
    String selection = Users.COLUMN_NAME_USERNAME + " LIKE ?";
    String[] selectionArgs = {userName};

    int count = db.update(
            Users.TABLE_NAME,
            values,
            selection,
            selectionArgs
    );
}
```

Delete a record

```
//This will delete a particular user from the table
public void deleteInfo(String userName){
    SQLiteDatabase db = getReadableDatabase();
    //Define 'where' part of query
    String selection = Users.COLUMN_NAME_USERNAME + " LIKE ?";
    //Specify arguments n placeholder order
    String[] selectionArgs = { userName };
    //Issue SQL statement
    db.delete(Users.TABLE_NAME, selection, selectionArgs);

}
```

SLIIT
Discover Your Future

Mobile Application Development

Mobile Application Testing

- Mobile Mindset
- Mobile Platforms and Application Development fundamentals
- Introduction to Android Operating System
- Android Interface Design Concepts
- Main Components of Android Application
- Sensors and Media Handling in Android Applications
- Data Handling in Android Applications
- Android Application Testing and security aspects
- Kotlin Language to develop Android Mobile Apps

Learning Outcomes of the Lecture

At the end of this lecture students should be able to:

- Identify the purpose of Mobile Application Testing
- Understand the Testing Types for Mobile Application
- Write test cased for Android Mobile Application

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

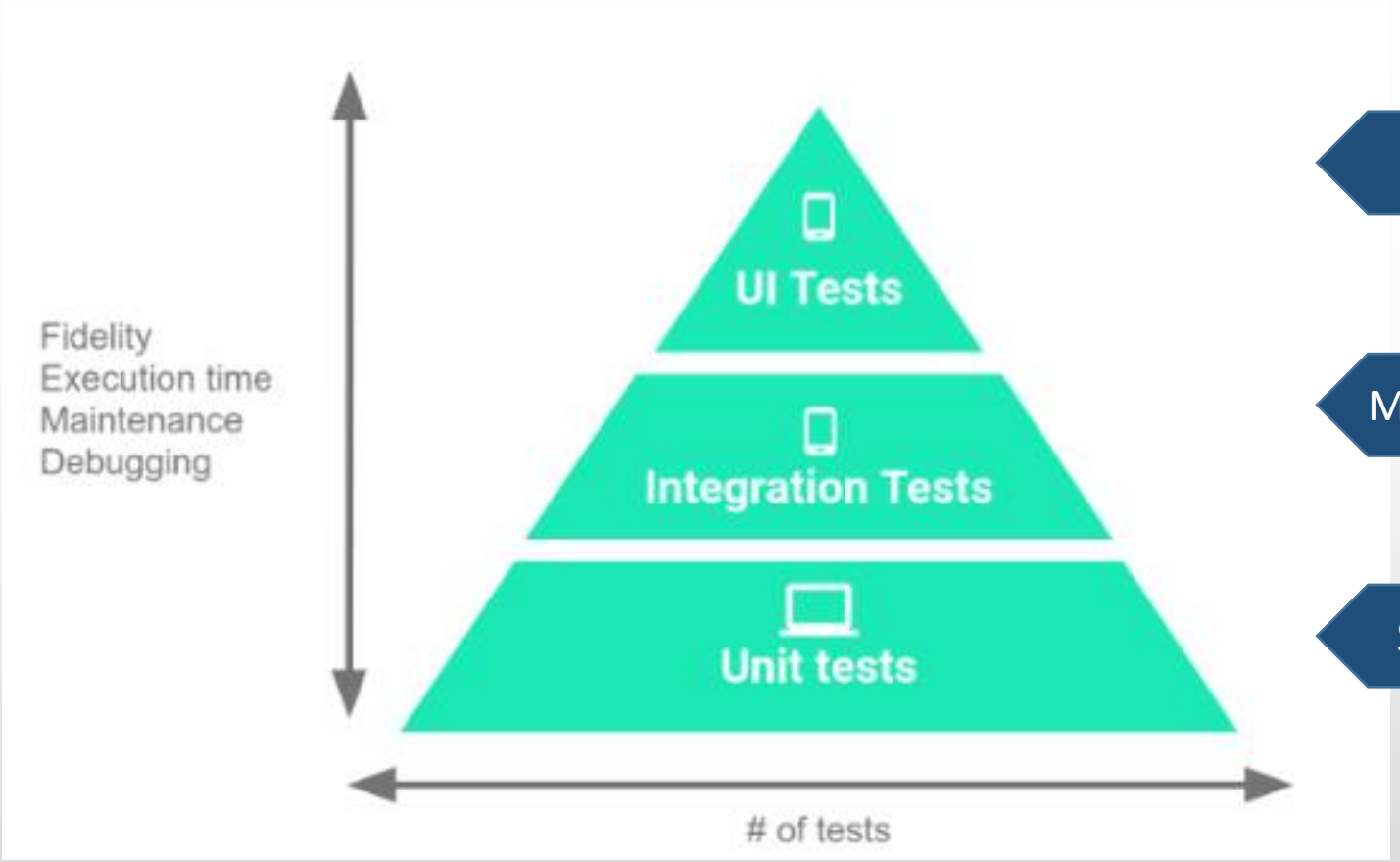
Eg: - Data cable removal during data transfer process.

- 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.

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 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.

Setting dependencies

✓Specify the plugin to the root file.

```
buildscript {
  dependencies {
    classpath "de.mannodermaus.gradle.plugins:android-junit5:1.3.2.0"
  }
}
```

✓Specify the test library dependencies in the app module's *build.gradle* file.

```
testImplementation "org.junit.jupiter:junit-jupiter-api:5.3.2"
testRuntimeOnly "org.junit.jupiter:junit-jupiter-engine:5.3.2"
testImplementation "org.junit.jupiter:junit-jupiter-params:5.3.2"
testRuntimeOnly "org.junit.vintage:junit-vintage-engine:5.3.2"
```

Sample Calculator for local Unit tests

MainActivity.java

```
protected int multiplyNumbers(int x, int y) {
    return x*y;
}

protected int subNumbers(int x, int y) {
    return x - y;
}

protected int addNumbers(int x, int y) {
    return x + y;
}
```

MainActivityTest.java

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class ExampleUnitTest {
    private  MainActivity mainActivity;
    @BeforeEach
    public void setup(){
        mainActivity = new MainActivity();
    }
    @Test
    public void testAddNumbers(){
        int result = mainActivity.addNumbers( x: 4,  y: 6);
        assertEquals( expected: 10, result);
    }
    @Test
    public void testSubNumbers(){
        int result = mainActivity.subNumbers( x: 4, y: 6);
        assertEquals( expected: -2, result);
    }
    @Test
    public void testMultNumbers(){
        int result = mainActivity.multiplyNumbers( x: 4, y: 6);
        assertEquals( expected: 24, result);
    }
}
```

Testing annotations in jUnit5

Annotation	Description
`@Test`	Denotes that a method is a test method. Unlike JUnit 4's `@Test` annotation, this annotation does not declare any attributes, since test extensions in JUnit Jupiter operate based on their own dedicated annotations. Such methods are *inherited* unless they are *overridden*.
`@ParameterizedTest`	Denotes that a method is a parameterized test. Such methods are *inherited* unless they are *overridden*.
`@RepeatedTest`	Denotes that a method is a test template for a repeated test. Such methods are *inherited* unless they are *overridden*.
`@TestFactory`	Denotes that a method is a test factory for dynamic tests. Such methods are *inherited* unless they are *overridden*.
`@TestTemplate`	Denotes that a method is a template for test cases designed to be invoked multiple times depending on the number of invocation contexts returned by the registered providers. Such methods are *inherited* unless they are *overridden*.
`@TestMethodOrder`	Used to configure the test method execution order for the annotated test class; similar to JUnit 4's `@FixMethodOrder`. Such annotations are *inherited*.
`@TestInstance`	Used to configure the test instance lifecycle for the annotated test class. Such annotations are *inherited*.

Testing annotations in jUnit5

`@DisplayName`	Declares a custom display name for the test class or test method. Such annotations are not *inherited*.
`@DisplayNameGeneration`	Declares a custom display name generator for the test class. Such annotations are *inherited*.
`@BeforeEach`	Denotes that the annotated method should be executed *before* **each** `@Test`, `@RepeatedTest`, `@ParameterizedTest`, or `@TestFactory` method in the current class; analogous to JUnit 4's `@Before`. Such methods are *inherited* unless they are *overridden*.
`@AfterEach`	Denotes that the annotated method should be executed *after* **each** `@Test`, `@RepeatedTest`, `@ParameterizedTest`, or `@TestFactory` method in the current class; analogous to JUnit 4's `@After`. Such methods are *inherited* unless they are *overridden*.
`@BeforeAll`	Denotes that the annotated method should be executed *before* **all** `@Test`, `@RepeatedTest`, `@ParameterizedTest`, and `@TestFactory` methods in the current class; analogous to JUnit 4's `@BeforeClass`. Such methods are *inherited* (unless they are *hidden* or *overridden*) and must be `static` (unless the "per-class" test instance lifecycle is used).
`@AfterAll`	Denotes that the annotated method should be executed *after* **all** `@Test`, `@RepeatedTest`, `@ParameterizedTest`, and `@TestFactory` methods in the current class; analogous to JUnit 4's `@AfterClass`. Such methods are *inherited* (unless they are *hidden* or *overridden*) and must be `static` (unless the "per-class" test instance lifecycle is used).
`@Nested`	Denotes that the annotated class is a non-static nested test class. `@BeforeAll` and `@AfterAll` methods cannot be used directly in a `@Nested` test class unless the "per-class" test instance lifecycle is used. Such annotations are not *inherited*.

Testing annotations in jUnit5

`@Tag`	Used to declare tags for filtering tests, either at the class or method level; analogous to test groups in TestNG or Categories in JUnit 4. Such annotations are *inherited* at the class level but not at the method level.
`@Disabled`	Used to disable a test class or test method; analogous to JUnit 4's `@Ignore`. Such annotations are not *inherited*.
`@Timeout`	Used to fail a test, test factory, test template, or lifecycle method if its execution exceeds a given duration. Such annotations are *inherited*.
`@ExtendWith`	Used to register extensions declaratively. Such annotations are *inherited*.
`@RegisterExtension`	Used to register extensions programmatically via fields. Such fields are *inherited* unless they are *shadowed*.
`@TempDir`	Used to supply a temporary directory via field injection or parameter injection in a lifecycle method or test method; located in the `org.junit.jupiter.api.io` package.

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.