Mobile Programming

Course 1

Qiong Liu

qiong.liu@cyu.fr
CY Tech, CY Cergy Paris University

2024 - 2025

Contents

1. Pre - information

2. Introduction to Android Development

3. Installing Android Studio

4. Backup: Java language principals in Android

Contents

1. Pre - information
2. Introduction to Android Development
3. Installing Android Studio
4. Backup: Java language principals in Android

Course Schedule

- 10 sessions in total
  - 5 Lectures
  - 4 Labs
  - 1 Project
- Grades:
  - Lab 1: 15%
  - Lab 2: 15%
  - Lab 3: 15%
  - Lab 4: 15%
  - Project: 40%
- Materials:
  - I will update the slides and TPs at `https://www.qiongliu.info/teaching/2024-MP`.
  - Reference: `http://developer.android.com`

Contents

1 Pre - information

2 Introduction to Android Development

3 Installing Android Studio

4 Backup: Java language principals in Android

What is Android?

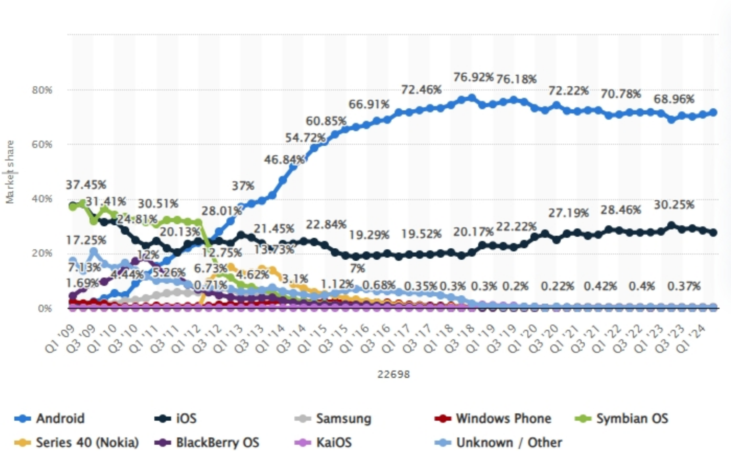
- Android is an open-source mobile operating system developed by Google.
- Based on the Linux kernel, designed for touchscreen devices (smartphones, tablets).
- First launched in 2008, it has become the world's most popular mobile OS.
- Current version: Android 14 (as of 2024).
- Originally developed by Andy Rubin, Android primarily supports smartphones.

History on Android

- Jul. 2005: Google acquires Android (startup)
- Nov. 2007: Android is developed by a consortium of developers known as the Open Handset Alliance
- Oct. 2008: Android goes open source
- Dec. 2010: Android 2.3 last smartphone-only version
- Jan. 2011: Android 3.x for tablets
- Oct. 2011: Android 4.x unified version (smartphone + tablet)
- Nov. 2014: Android 5
- Oct. 2015: Android 6
- . . .
- Oct. 2023: Android 14
- Oct. 2024: Android 15

Android Global Market Share

- Android holds about **68.96%** of the global smartphone market.
- Used by millions of devices worldwide including brands like Samsung, Xiaomi, and Google.



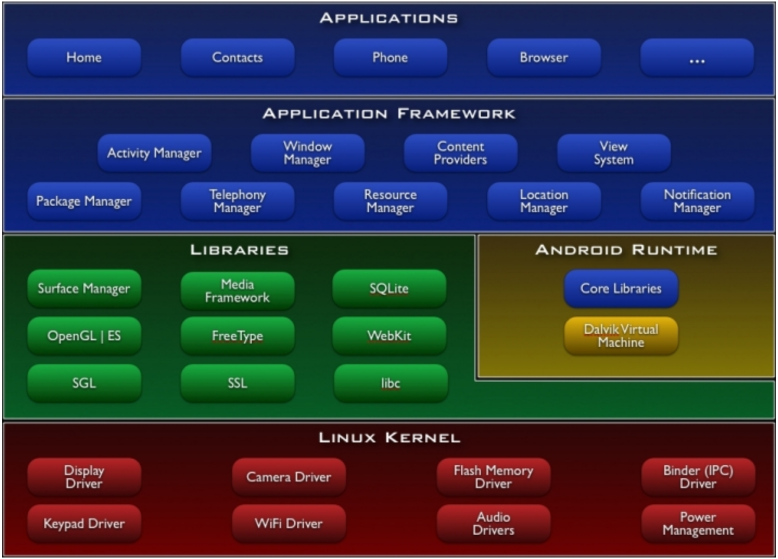
Key Features of Android (part 1)

- Open Source: Free to use and customize.
- Multitasking
  - Android allows multiple apps to run simultaneously
  - Permission management (via manifests) to control app access (e.g., GPS, camera)
- Rich App Ecosystem
  - The Google Play Store offers over 3 million apps.
  - Android's open nature allows custom app stores beyond Google Play
- Versatile Hardware Support
  - Runs on smartphones, tablets, smart TVs, smartwatches, and vehicles (Android Auto).
- Google Play Services
  - Provides APIs for integrating Google services like Maps, Drive, and Analytics.
  - Simplifies development of feature-rich apps with built-in services.
- . . .

Key Features of Android (part 2)

- Programming and APIs
  - Official languages: Java, Kotlin (Native code in C/C++ is also possible)
- Android SDK (Software Development Kit)
  - Tools for development: Compiler, debugger, and device emulator
- Storage and Databases
  - File system access for local storage
  - Built-in support for SQLite databases
- . . .

System Architecture



Android System Architecture Overview

- Applications: User-facing apps like Contacts, Phone, Browser.
- Application Framework: Provides APIs for managing activities, windows, notifications, content providers, and more.
- Libraries: Core libraries like OpenGL, SQLite, WebKit, and Media Framework for multimedia handling.
- Android Runtime (Dalvik/ART): Executes application bytecode using the Dalvik Virtual Machine or Android Runtime (ART).
- Linux Kernel: Provides core system services like memory management, drivers, power management, and IPC through the Binder framework.

Android Studio Development History

- Android Studio is an Android application development tool released by Google in May 2013.
- It is based on IntelliJ IDEA, which makes it easier and faster to use compared to Eclipse.

Android Studio Version	Release Date
Android Studio 4.0	May 2020
Android Studio 3.0	October 2017
Android Studio 2.0	April 2016
Android Studio 1.0	May 2013
Android Studio Arctic Fox (2020.3.1)	July 2021
Android Studio Chipmunk (2021.2.1)	May 2022
Android Studio Dolphin (2021.3.1)	September 2022
Android Studio Flamingo (2022.2.1)	April 2023
Android Studio Giraffe (2023.3.1)	August 2023

Language Comparison in Android Development

- Java ($60\% \sim 70\%$):
  - Primary language for Android development.
  - Strong community support, widely used in Android.
  - Statically typed, object-oriented, and well-documented.
- Kotlin ($30\% \sim 40\%$):
  - Official language for Android development since 2017.
  - More concise and expressive compared to Java.
  - Interoperable with Java, can be used alongside it in the same project.
- C++ ($< 5\%$):
  - Used in Android Native Development Kit (NDK) for high-performance needs.
  - Primarily for games or apps requiring extensive computational tasks.
- Which Language to Use?
  - **Kotlin** a rising alternative, but both are officially supported.
  - **Java** for established projects and compatibility.
  - **C++** for performance-critical sections (via NDK).
- For This Course: We will use **Java** to cover core Android concepts.

Overview of Android Components

What is an Android App?

- An Android app is made up of several components that work together to create a functional user experience.
- The key components:
  - Activity: A single screen with a user interface.
  - Fragment: A modular section of an activity, a portion of the UI.
  - Service: Performs background operations without a user interface.
  - Broadcast Receiver: Listens for and responds to system-wide broadcast messages.
  - Content Provider: Manages data sharing between applications.

Activity

Activity

An activity is a core Android component that represents a single screen of the user interface.

- It acts like a window where the user can interact with the app.
- Each activity is associated with a layout that defines the user interface.
- Apps can have multiple activities, but one is designated as the MainActivity, the default entry point in an Android app.

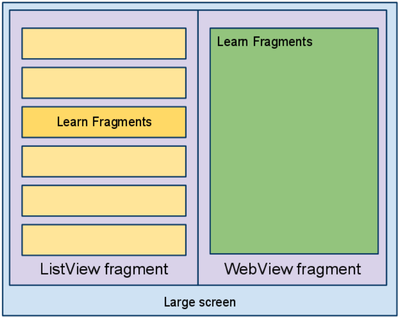
Activity

Activity Lifecycle: Activities follow a lifecycle, which includes important methods like:

- `onCreate()` – Initializes the activity.
- `onStart()` – Called when the activity becomes visible.
- `onResume()` – Called when the activity starts interacting with user.
- `onPause()` – the activity is partially obscured by another activity.

Fragments

- A Fragment is a reusable portion of your app's UI.
- Fragments have their own lifecycle, can handle input events, and define/manage their own layout.
- Unlike an Activity, a Fragment cannot exist independently. It must be hosted by an Activity or another Fragment.
- Fragments allow for dynamic UI changes within an app.



What is an Android Service? (UberEats Example)

What is a Service?

- A Service in Android is a component that runs in the background to perform long-running tasks without a user interface.
- Services do not directly interact with the user but continue working even when the app is not in the foreground.

UberEats Example:

- Receiving real-time updates about your food order status (e.g., when it's being prepared or on the way) even if the app is closed.
- Notifying you when your delivery is near or completed, without the need for the app to be open.

Broadcast Receiver

What is a Broadcast Receiver?

- Listen from and respond to system-wide broadcast messages.
- It allows your app to react to events that happen outside of the app, such as system events or notifications from other apps.
- Broadcast messages are sent by the system or other apps (e.g., battery low, incoming messages).

Uber Eats Example:

- Detect your GPS
- Detect network connectivity changes: UberEats can detect when your device connects to or disconnects from Wi-Fi.

Content Provider

What is a Content Provider?

- A component in Android that manages access to a structured set of data.
- It allows apps to share data with other apps in a controlled and secure way.

Uber Eats Example:

- When you add or use your bank card in Uber Eats, the app might need authorization from your banking app.
- The banking app manages sensitive payment data, and the Content Provider ensures the data is shared securely between UberEats and your bank.
- The banking app authorizes the transaction, then sends the response back to Uber Eats, completing the payment process.

Contents

1. Pre - information
2. Introduction to Android Development
3. Installing Android Studio
4. Backup: Java language principals in Android

Development Machine Requirements

- Hardware Requirements:
  - At least 8GB of RAM (more is better).
  - CPU requirement of 1.5GHz or higher (the faster, the better).
  - Minimum 10GB of available disk space (the more, the better).
  - Wireless network card and USB ports are required.
- Operating System Requirements:
  - Windows:
    - Must be a 64-bit version (32-bit is not supported).
    - Windows 10 or 11 recommended.
  - macOS (for iOS):
    - macOS 10.14 Mojave or later (macOS 12 Monterey recommended).
    - Xcode (latest version) required for building iOS apps.
    - Minimum 4GB of RAM (8GB recommended).
  - Linux:
    - 64-bit distributions only (32-bit not supported).
    - Tested on Ubuntu 18.04 LTS, Fedora 29, and Debian 9.
    - Make sure ‘g++‘, ‘make‘, and ‘Java Development Kit (JDK 8 or later)‘ are installed.

Windows System Requirements

- Check System Architecture (32-bit or 64-bit):
  - Go to `Settings`→`System`→`About`.
  - Check `System type` for →`64 bit operating system`.
- Check RAM and CPU:
  - Press ‘Ctrl + Shift + Esc‘ to open `Task Manager`.
  - Under `Performance` tab, view CPU speed (1.5GHz+ recommended) and `Installed RAM` (8GB+).
- Check Disk Space:
  - Open `File Explorer` and check available space under `This PC` (minimum 10GB free).
- Check Windows Version:
  - Press ‘Windows + R‘, type ‘`winver`‘, and hit Enter.
  - Must be **Windows 7 or later** (Windows 10 or 11 recommended).

macOS System Requirements

- Check macOS Version:
  - Click `Apple Menu`→ **About This Mac**.
  - Must be `macOS 10.14 Mojave` or later (macOS 12 Monterey recommended).
- Check RAM and CPU:
  - In `About This Mac`, view `Memory` for RAM (8GB minimum, 16GB recommended).
  - Check `Processor` for CPU speed (1.5GHz minimum).
- Check Disk Space:
  - In `About This Mac`, go to `Storage` to check available space (10GB minimum).
- Check for Xcode (iOS Developers):
  - Open `Terminal` and run: `xcode-select --version`.
  - If Xcode is not installed, download it from the Mac App Store.

Linux System Requirements

- Check System Architecture (64-bit):
  - Open **Terminal** and run: ‘`uname -m`‘.
  - Output must be `x86_64` (64-bit).
- Check RAM and CPU
  - Run: ‘`free -h`‘ to check RAM (8GB minimum).
  - Run: `lscpu` to check CPU speed (1.5GHz+ recommended).
- Check Disk Space
  - Run: ‘`df -h`‘ and check available space under ‘/‘ (10GB minimum).
- Check for Required Packages (g++, make, JDK)
  - Run: ‘`sudo apt install g++ make openjdk-21-jdk`‘ to install necessary packages.
- Check Linux Distribution Version
  - Run: ‘`lsb_release -a`‘ to check distribution (Ubuntu 18.04+ or Fedora 29+ supported).

What to Do if System Doesn't Meet Requirements

- Optimize Performance: Close background apps, use physical Android devices, or lightweight emulators.
- Upgrade OS: Switch to 64-bit Windows, latest macOS, or a supported Linux distribution.
- Use Virtual Machine: Set up a 64-bit Linux VM for Android development.
- Alternative IDEs: Try IntelliJ IDEA with Android Plugin, Visual Studio Code, or cross-platform tools like Flutter.

Step 1: Pre-installation – JDK

Install JDK and configure environment

1. Download and install **JDK** from `https://www.oracle.com/java/technologies/downloads/`
2. Configure `JAVA_HOME` and `Path`
   - `JAVA_HOME`: let your OS knows where the JDK is installed;
   - `Path`: ensures that Java commands are recognized globally in the system, so you don't have to specify the full path every time.

Step 2: JDK Environment Configuration (Windows)

Windows Steps:

1. Windows+R to open command prompt, type `sysdm.cpl`, go to the **Advanced** tab, and click **Environment Variables**.
2. Set `JAVA_HOME`:
   - Under System variables, click **New**.
   - Variable Name: `JAVA_HOME`, Variable Value: JDK installation path (e.g., `C:\Program Files\Java\jdk-21`).
3. Update the `Path` Variable:
   - Find the Path variable, click **Edit**, then add `%JAVA_HOME%\bin`.
4. Verify installation: Run `java -version` and `javac -version` in Command Prompt.

Step 2: JDK Environment Configuration (Linux)

Linux Steps:

1. Install the JDK:
   - `sudo apt update`
   - `sudo apt install openjdk-21-jdk`
2. Verify the installation:
   - Run `java -version`.
3. Locate the JDK Installation Path:
   - Run `sudo update-alternatives --config java`.
4. Configure `JAVA_HOME`:
   - Edit shell configuration file: `nano ~/.bashrc`
   - Add `export JAVA_HOME=/usr/lib/jvm/java-21-openjdk-amd64` and `export PATH=$JAVA_HOME/bin:$PATH`.
5. Apply changes: Run `source ~/.bashrc`
6. Verify configuration: Run `echo $JAVA_HOME`.

Step 3: Install Android Studio

1. Download and install Android Studio from `https://developer.android.com/studio`
2. Open Android Studio → New → Project → Empty View Activity
3. Name the new project "Lecture1"
4. Choose Java as the language

Execution

Run on the Emulator

- Select View → Tool Windows → Device Manager from the main menu bar, and then click Create device.

Run on a real device

- Connect the device to your machine with a USB cable
- Is your device supported? If the default USB driver does not work, you might need to install a special ADB driver.
- Ensure that USB debugging is enabled in the device settings
- Click Run from the toolbar.
- Android Studio installs the app on your connected device and starts it.

Project Architecture

Project Architecture:

- **manifests/** AndroidManifest.xml : Fundamental configuration of the application (permissions, feature requirements, main Activity, . . .)
- **java/** Source files
- **res/**: Non-code application resources (images, strings, layout files, etc.).
  - drawable: drawable objects (such as bitmaps)
  - layout: XML files that define the user interface
  - mipmap: The mipmap (multiple-density map) folder is used to store different versions of the same image at various resolutions or densities.
  - values: various XML files that contain resources, such as string and color definitions.
  - xml: The xml folder is used to store various XML files used in your Android application.

Gradle: Build automation tool (like Ant, Maven), creates the apk file.

Project Architecture

- **Project Root Files:**
  - `.gradle` (Project cache – *do not modify*)
  - `.idea` (IDE config – *do not modify*)
  - `gradle` (Build system – *do not modify*)
  - `.gitignore` (Git config – can modify for version control)
  - External Libraries (Managed by Gradle – *do not modify*)
- **App Module:**
  - `src/main/java` (Java code – students modify)
  - `src/main/res` (Resources – students modify)
  - `build.gradle` (Module-level config – can modify for dependencies)
  - `build/` (Auto-generated files – *do not modify*)

API Overview (1/2)

Application Programming Interfaces (APIs) in Android:

- APIs provide predefined methods and classes for accessing system resources and hardware.
- In Java, APIs are imported using the `import` statement.
- Commonly used Android APIs:
  - J2SE (Java Standard Edition APIs):
    - `import java.util.*;` (e.g., `ArrayList`, `HashMap`)
    - `import java.io.*;` (e.g., file operations, input/output)
    - `import java.lang.*;` (e.g., `String`, `Math`)
  - UI (User Interface):
    - `import android.widget.*;` (e.g., `Button`, `TextView`)
    - `import android.view.*;` (e.g., layouts, touch events)
    - `import android.graphics.*;` (e.g., drawing, handling bitmaps)

API Overview (2/2)

- Phone, SMS, Web, Camera:
  - `import android.telephony.*;`
  - `import android.telephony.SmsManager;`
  - `import android.webkit.WebView;`
  - `import android.hardware.*;` (e.g., controlling camera, sensors)
- Database, Multimedia, HTTP:
  - `import android.database.*;` (e.g., accessing SQLite databases)
  - `import android.media.*;` (e.g., playing music, videos, recording audio)
  - `import org.apache.http.client.*;` (for HTTP requests, now deprecated)

Creating an Android Application

Android application = Java (code) + resources (XML, images, etc.)

Key steps in building an Android app:

1. `javac`: Compiles Java source code into bytecode
   - Converts `.java` files into `.class` bytecode
2. `dx`: Converts bytecode to Dalvik executable
   - Compresses `.class` files into `.dex` format (Dalvik Executable)
3. `aapt`: Packages `.dex` files + resources into an APK
   - Creates the final `.apk` (Android Package) containing code and resources
4. `adb`: Deploys the APK onto a device
   - Installs the APK on a physical device or emulator for testing

Simplified process:

- Just one click in Android Studio:
- Select `Build` → `Build Bundle(s) / APK(s)`.

Resources

Resources

- An Android application is more than just code
- For every resource that you include in your Android project, the build tool defines a unique integer ID, which you can use to reference the resource from within the code (class “R”)
- Resources must be lowercase

Advantages:

- MVC: Model, View, and Controller.
- update your application without modifying code
- customize your application

Resources in Android (1/2)

Where are resources stored?

In Android, all resources are stored under the `res` directory. Different types of resources are organized into specific subdirectories:

- **res/layout/**:
  - Contains XML layout files that define the user interface.
  - Example: `activity_main.xml`
- **res/drawable/**:
  - Stores images, vector graphics, or XML-based graphic elements.
  - Example: `logo.png`, `rounded_button.xml`
- **res/values/**:
  - Stores constants like strings, colors, dimensions, and styles.
  - Example: `strings.xml`, `colors.xml`, `styles.xml`

Resources in Android (2/2)

More resource directories:

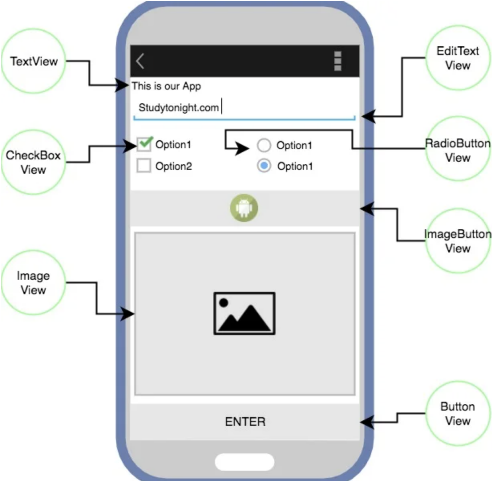
- **res/mipmap/**:
  - Contains app icons in various resolutions for different screen densities.
  - Example: `ic_launcher.png`
- **res/raw/**:
  - Stores raw, unprocessed files (e.g., audio, video).
  - Example: `sound.mp3`, `data.txt`
- **res/menu/**:
  - Contains XML files that define menus in the app.
  - Example: `main_menu.xml`

Activities

- Provides a screen with which users can interact (displays Views and handles Events).
- An application usually consists of multiple activities
- Each activity has a window for its user interface
- One activity is specified as the "main" activity, which is presented to the user when launching the application
- Each activity can start another activity. Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack ("back stack").
- When the user is done with the current activity and presses the Back button, it is popped from the stack (and destroyed) and the previous activity resumes.

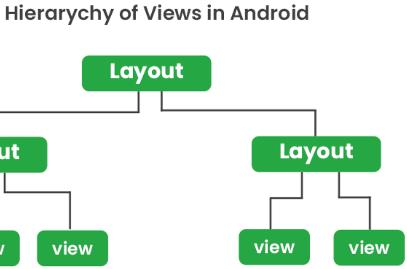
View

- The user interface for an activity is provided by a hierarchy of views—objects derived from the View class.
- Each view controls a particular rectangular space within the activity's window and can respond to user interaction.



Layout

- Layouts are special views derived from ViewGroup that provide a layout model
- XML layout file (saved as a resource)
- Some interesting layout managers:
  - ConstraintLayout
  - LinearLayout
  - GridLayout



Now you open your MainActivity.

Creating a New App Page

- The complete process of creating a page includes three steps:
  - Create an XML file in the `layout` directory.
  - Create the corresponding Java code for the XML file.
  - Register the page configuration in `AndroidManifest.xml`.

Main Activity

We always change `.java` file and `.xml` together.

- `MainActivity.java` (/app/src/main/java/com.example.lecture1/):
  - Handles the logic and behavior of the app.
  - Defines what happens when the user interacts with the app.
- `activity_main.xml`(/app/src/res/layout/):
  - Defines the layout and UI components (buttons, text, etc.).
  - Uses XML to design how the app looks.
- Interaction: After linking, Java code can modify or interact with UI components defined in XML.

Defining UI in activity_main.xml

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/myTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />

    <Button
        android:id="@+id/myButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me" />
</LinearLayout>
```

Button Click

1) Add the android: onClick attribute to the "Button" element in your XML layout.

```
<Button
    android:id="@+id/mybutton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/clickme"
    android:onClick="clicked" />
```

2) Within the Activity, the following method handles the click event:

```
    public void clicked(View view) {
    // do something in response to button click
        view.setEnabled(false);
    }
```

Exercise

1. Create an Button. When you click it, it will display how many times the button is clicked.
2. Change the associated action to be generate a random number

Reference

http://developer.android.com

Contents

1 Pre - information

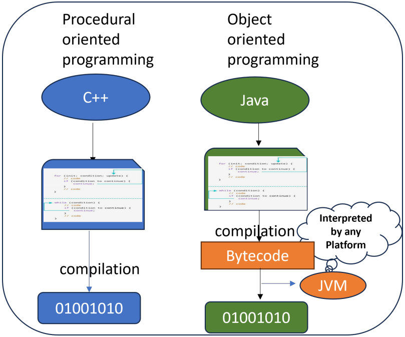
2 Introduction to Android Development

3 Installing Android Studio

4 Backup: Java language principals in Android

POP vs OOP

A simple case for JAVA advantage:


Procedural oriented programming
Object oriented programming
C++
Java
compilation
compilation
Bytecode
Interpreted by any Platform
JVM
01001010
01001010

```
for (init; condition; update) {
    // code
    if (condition to continue) {
      continue;
    }
    // code
}

while (condition) {
    // code
    if (condition to continue) {
      continue;
    }
    // code
}
```

```
for (init; condition; update) {
    // code
    if (condition to continue) {
      continue;
    }
    // code
}

while (condition) {
    // code
    if (condition to continue) {
      continue;
    }
    // code
}
```

Figure: Difference between POP and OOP

The characteristics of Java Virtual Machine (JVM):

- Platform Independence
- Memory Management
- Bytecode Execution
- Security
- Managed Execution Environment

Object-Oriented Principles

Four Key Principles:

Abstraction

Hiding unnecessary details and showing only essential features.

Encapsulation

Encapsulating data and methods together; restricting access to internal details.

Inheritance

Creating new classes based on existing ones, promoting code reuse.

Polymorphism

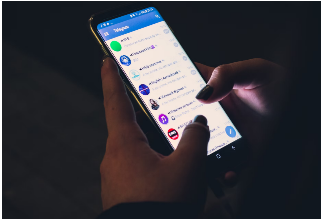
One action or method can behave differently based on the object.

Abstraction

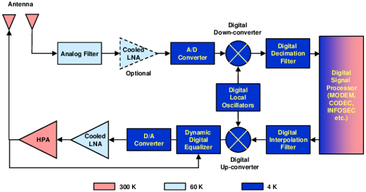
Abstraction hide unnecessary details and showing only essential features

- What method to call?
- What parameters to input?

E.g.1, how to define "Abstraction"



Abstraction: Send a message by click the button.


Antenna
Analog Filter
Cooled LNA
Optional
A/D Converter
Digital Down-converter
Digital Decimation Filter
Digital Signal Processor (MODEM, CODEC, INFOSEC etc.)
Digital Local Oscillators
HPA
Cooled LNA
D/A Converter
Dynamic Digital Equalizer
Digital Up-converter
Digital Interpolation Filter
300 K
60 K
4 K

Text message --> coding --> DA converter --> antenna

Using Abstraction in Android Code

```java
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Abstraction: The button is set up to send a message
        Button sendButton = findViewById(R.id.sendButton);
        sendButton.setOnClickListener(v -> sendMessage());
    }
    // Abstraction: hides the details of how the message is sent
    private void sendMessage() {
        Log.d("MainActivity", "Message sent!");
    }
}
```

- The user clicks the button, and the message is sent without needing to understand the internal process.
- `sendMessage()` abstracts the actual logic of sending a message.
- This hides unnecessary complexity, allowing developers to focus on higher-level interactions.

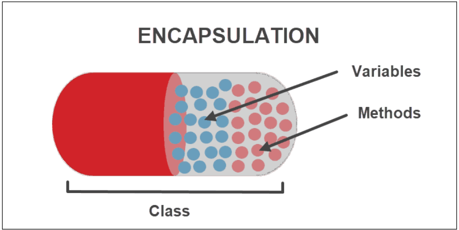
Encapsulation.

Encapsulation

Encapsulation refers to the bundling of data with the attributes/features/properties or methods that operate on the data.

Example

- An Android Activity as a capsule that contains data (user input, state) and methods to operate on that data.
- The internal details of how the data is stored or processed are hidden from the user or other classes.
- We interact with the Activity via public methods that control how the data is used or displayed.



Encapsulation Example in Android Code

```java
public class User {
    // Private data: cannot be accessed directly from outside
    private String name;
    private int age;

    // Constructor to initialize the User object
    public User(String name, int age) {
        this.name = name;
        this.age = age;
    }
    // Public method to modify the private data
    public void setName(String name) {
        this.name = name;
    }
}
```

Key Points:

- Private fields: The ‘name‘ and ‘age‘ are hidden from outside access.

Inheritance in Android

What is Inheritance?

- Inheritance defines a relationship between classes.
- A subclass inherits properties and behaviors (attributes and methods) from a superclass.
- In Android, many components (e.g., `Activity`, `View`) inherit common functionality from their superclasses.
- Inheritance allows Android components to reuse code and extend functionality.

Key Properties of Inheritance in Android:

- The subclass inherits all attributes of the superclass (e.g., layout handling from `Activity`).
- The subclass inherits all methods of the superclass (e.g., `onCreate()`, in `AppCompatActivity`).
- An Android component like `MainActivity` is a specialized version of `AppCompatActivity`.

Examples of Inheritance

When does a class D inherit from class B?

Inheritance Rules

- The set of attributes of B is included in the set of attributes of D.
- The set of methods of B is included in the set of methods of D.

Examples:

- Superclass: `AppCompatActivity`, Subclass: `MainActivity`
- Attributes: the data or variables stored in an object.
  - `e.g., title of the activity or the contentView that defines the layout.`
- Methods: the functions that objects of a class can perform.
  - `e.g., If MainActivity inherits from AppCompatActivity, methods onCreate() and onStart() are also included in MainActivity.`

Polymorphism in Android

"One function, many forms"

Definition

Polymorphism refers to the ability of a method to perform a single action in different ways depending on the object.

Examples:

- Superclass: `View`, Subclasses: `Button`, `TextView`
- Attributes: Both `Button` and `TextView` share common attributes from the superclass `View`.
  - *e.g., `width`, `height`, `text`.*
- Methods: The `setOnClickListener()` method behaves differently for each subclass.
  - *e.g., In `Button`, `setOnClickListener()` triggers a button press. In `TextView`, it handles a text click.*

Polymorphism in Android with Views

```java
// Superclass View
class View {
    public String getType() {
        return "Generic View"; // Default view
    }
}

// Subclass ButtonView
class Button extends View {
    @Override
    public String getType() {
        return "Button View";
    }
}

// Subclass TextView
class TextView extends View {
    @Override
    public String getType() {
        return "Text View";
    }
}

// Demonstration
public class TestPolymorphism {
    public static void main(String[] args) {
        View myButton = new Button();
        View myTextView = new TextView();
        System.out.println(myTextView.getType());
    }
}
```

Polymorphism in Action

When `getType()` is called on `myButton` and `myTextView`, it returns the type corresponding to the specific subclass (Button or TextView).