Module 1

This is a single, concatenated file, suitable for printing or saving as a PDF for offline viewing. Please note that some animations or images may not work.

Module 1 Study Guide and Deliverables

Topics:

- Introduction to Android
- Introduction to Android Application Development Using the Software Engineering Approach
- Introduction to Kotlin

Readings:

- Online lectures
- Notes and references cited in lectures
- Chapters 1-4 and Appendix ii from the "Head First" textbook (or corresponding chapters related to topics covered in this module in other textbooks)

Discussions:

- Discussion 1
  - Initial post due **Tuesday, July 12 at 6:00am ET**
  - Respond to threads posted by others due **Thursday, July 14 at 6:00am ET**

Labs:

- Lab 1 due **Sunday, July 10 at 6:00am ET**

Assignments:

- Project Assignment 1 due **Tuesday, July 12 at 6:00am ET**

Assessments:

- Quiz 1 due **Tuesday, July 12 at 6:00am ET**

Live Classrooms:

- **Tuesday, July 5 from 6:00-8:00pm ET**
- **Monday, July 11 from 6:00-8:00pm ET**

Learning Outcomes

By the end of this module, you will be able to:

- Describe the history, versions and issues of Android.
- Explain the Android platform, including its hardware, kernel and framework.
- Describe the basic components of an Android application.
- Describe the life cycle of an activity component in an Android application.
- Explain various activities in a SDLC (Software Development Life Cycle).
- Describe basic values in the Agile Manifesto.
- Create a very simple application as you get familiar with AS (Android Studio).
- Describe similarities and differences between Kotlin and Java.
- Explain simple code snippets in Kotlin.
- Write the project proposal for the semester long project.

Introduction

Enhanced with hardware and software capabilities, mobile devices keep evolving from single-purpose to general-purpose computing platforms. Mobile devices such as tablets and smartphones have become essential devices in both personal and professional lives. While mobile users can still use web-based applications through the browser, native mobile apps give users much better user experiences. According to eMarketer, the US adults will spend, on average, more than 4 hours with mobile internet, with 88% of that time within apps. Mobile app development is extremely important today in business operations and customer services. In fact, it is becoming a booming industry.

This course will help you get familiar with mobile application development using Android as the platform. Using Android Studio, the IDE (Integrated Development Environment) for Android development, this course will introduce you to the basics of Android programming, as well as some advanced topics. After this course, you will be able to understand and use various features provided by the Android platform to develop simple to complex Android applications.

Topic 1: Introduction to Android

Statistics

Here are some statistics from StatCounter.

PC vs. Mobile Devices

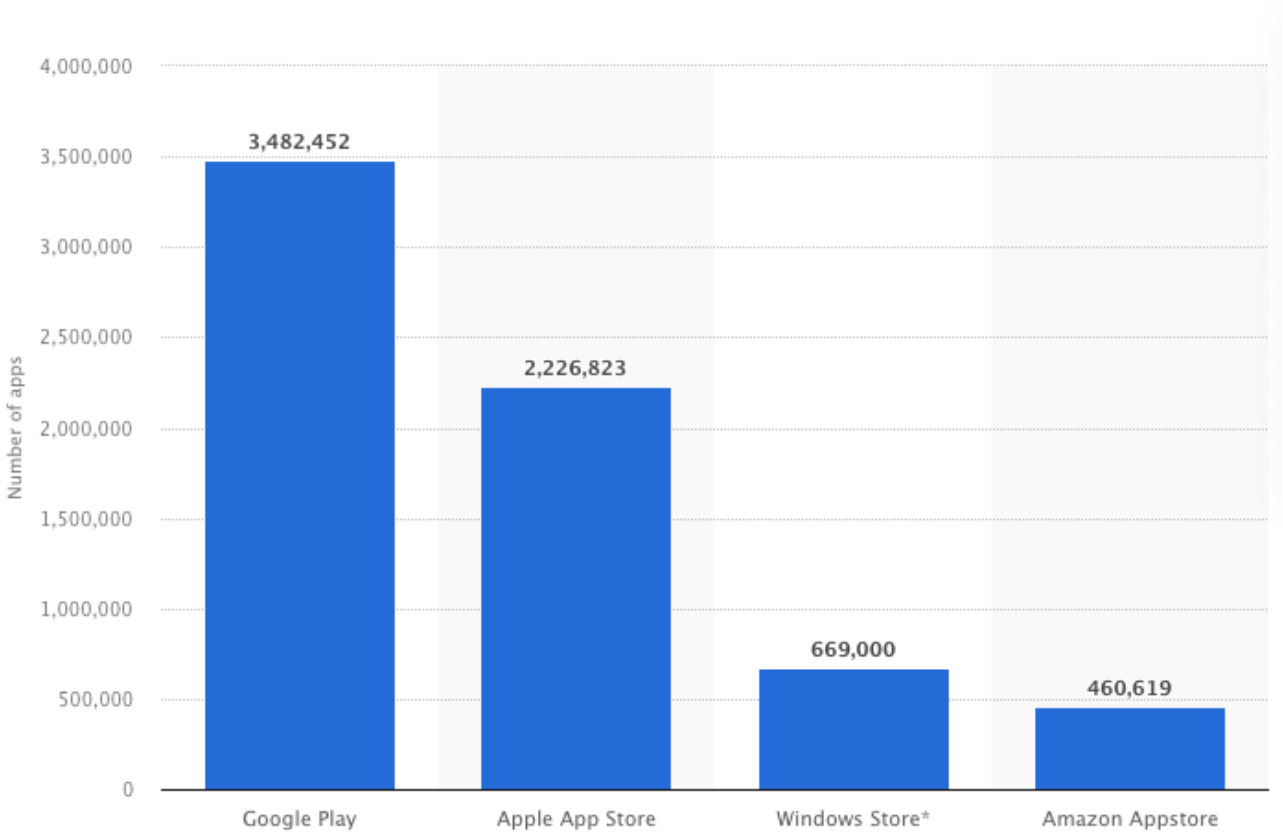
Source: StatCounter Global Stats - Platform Comparison Market Share

Smartphone OS (Operating System) Market Share

Figure 2 shows that purchases of Android phones dominate the world marketplace, with iOS phones in second place. (Source: Statcounter GlobalStats)

Source: StatCounter Global Stats - OS Market Share

Apps in App Stores



(Source: Statista 2022)

Figure 3 shows that Google play has the largest number of applications. The Amazon app store also mainly contains Android applications. In addition, there are also a number of small third-party app stores that mostly cater to Android users.

Test Yourself

Mobile users spend more time in _____.

Web applications

Native mobile applications

Test Yourself

Which platform dominates the worldwide mobile market?

Android

iOS

Windows Phone

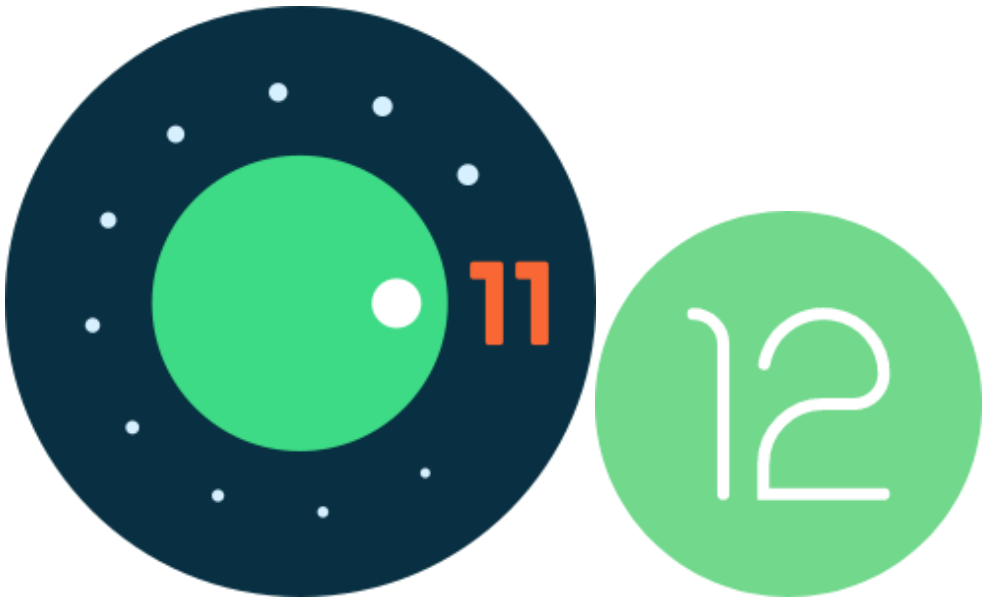
Android History and Versions

Here are some facts about Android history. You can find more information on the Android official website.

- Android Inc. was originally founded by Andy Rubin, Chris White, Nick Sears and Rich Miner in October 2003 in Palo Alto, CA.
- Google acquired Android Inc. in August, 2005.
- The OHA (Open Handset Alliance) was announced in November 2007. It is a consortium of technology companies including Google, HTC, Samsung, T-Mobile, and Qualcomm, with a goal of developing open standard for mobile devices. Currently it has a group of 84 members representing all parts of the mobile ecosystem, including mobile operators, handset manufacturers, semiconductor companies, and software companies. Together they have developed Android™, the first open and free mobile platform. More information can be found on the official OHA website.
- The OHA released an early version of Android SDK (Software Development Kit) one week later in Nov 2007.
- AOSP (Android Open Source Project) was officially released in Oct 2008, which is led by Google.
- The first publicly available Android smartphone, the T-Mobile G1 was also released in October 2008.

Android versions were named for sweet treats, with each new release starting with the next letter in the alphabet until Android 9. Since Android 10, Google dropped the sweet name and simply uses the version number now. A new Android version is released almost every year.

Android 10



Besides the Android version numbers and code names, we shall also be familiar with their mapping to the API levels as in the following table.

Android code Letter	S	R	Q	P	O	O	N	N	M	L	L	K	K
Android version	12	11	10	9	8.1	8.0	7.1	7.0	6.0	5.1	5.0	4.4W	4.4
API level	31	30	29	28	27	26	25	24	23	22	21	20	19
Android code	J	J	J	I	I	H	H	H	G	G			
Andvoid version	4.3	4.2	4.1	4.0.3	4.0	3.2	3.1	3.0	2.3.3	2.3			
API level	18	17	16	15	14	13	12	11	10	9			

As of Mar 2022, the latest version of Android is Android 12. However, not every Android user uses this latest version. Instead, Android users use all different versions from Android 4 Jelly Bean to Android 12. You can find the version distribution information in Android Studio, when you create a new project. For example, if you choose the minimum SDK to be Android 5, your app can run on approximately 98% of devices as of March, 2022. That means about 98% of devices run Android 5 or above now. If you choose a newer version such as Android 10, your app can only run on about 50% of devices. But If you choose the latest version Android 12, less than 1% of devices can run your app. Since there are numerous different manufactures for Android devices, and many manufacturers are less likely to offer or push a new OS version or cause apps to cease working if the user doesn't upgrade.

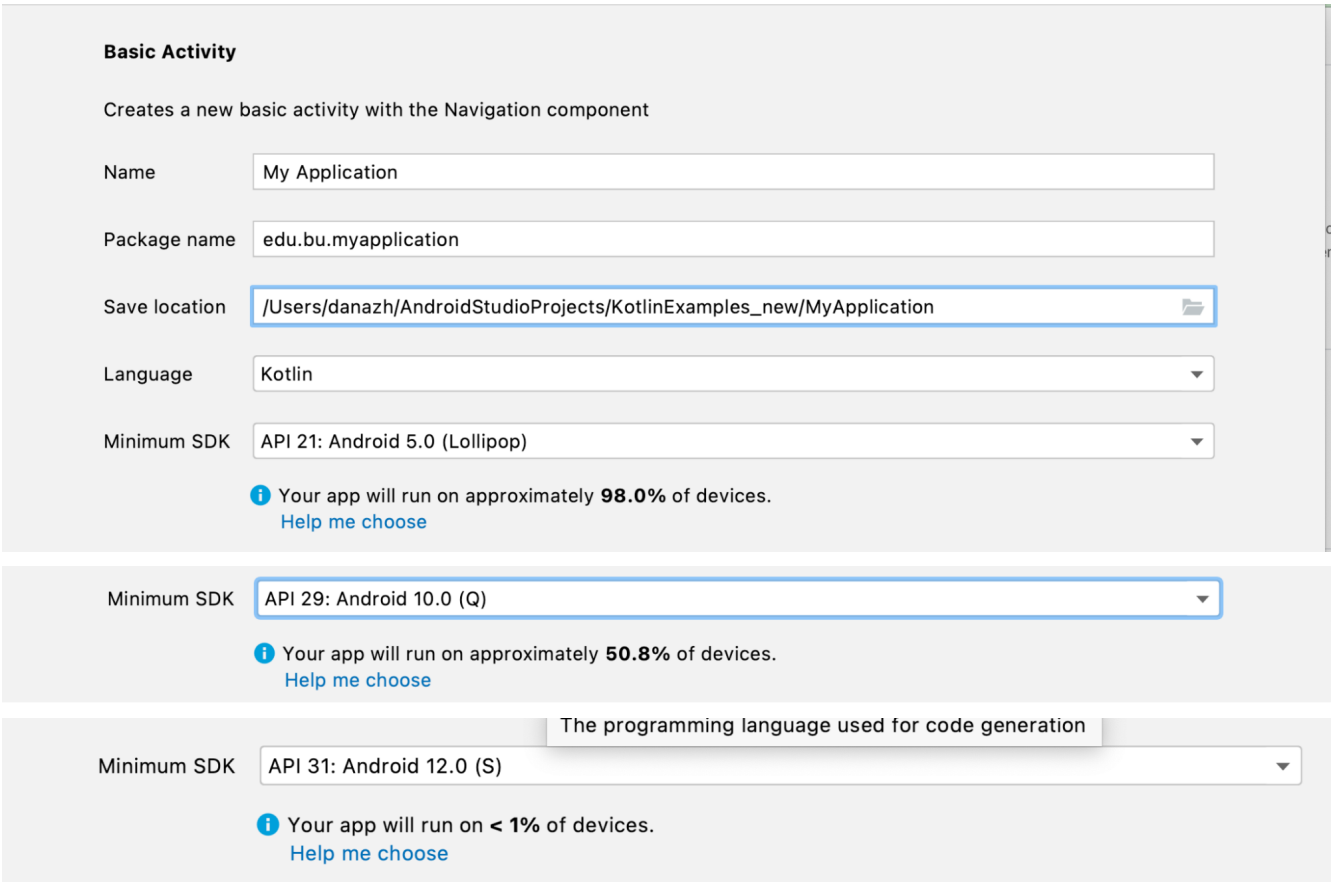


Figure 1.4: Choose Minimum SDK as 21, 29, or 31

If you click on “Help me choose”, you can see the version distribution as shown below. Most users use either Android 9, 10 or 11. In order to choose the right minimum SDK level, you need to consider both the new features your app needs to use and the compatibility with the older devices. In general, you may not choose Android 4 as there are not really security mechanisms in place in those old versions. Android 5 or 6 are better choices as it can enable your app to run on most devices, as well as the devices being more secure. Currently Android 6 (API 23) is the default choice in Android studio.

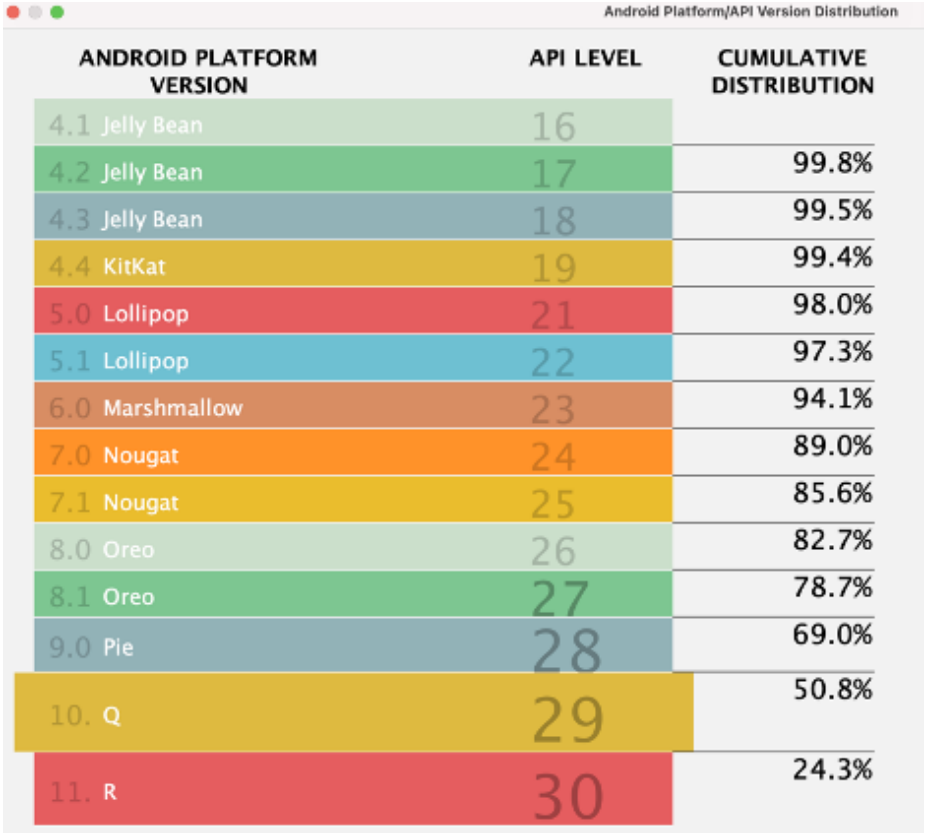


Figure 1.5 Android Version Distribution

Test Yourself

What version has API level 29?

Android 10 Q

Test Yourself

Check your Android studio to find out what the latest Android version is.

Test Yourself

Check your Android studio to find out which minimum SDK should be set if you want 90% of users to be able to use your app

Openness and Related Issues

The Android operating system was initially released by Google, and is now maintained through the AOSP (Android Open Source Project), which is administered by Google. The AOSP states that “Android is an open source software stack for a wide range of mobile devices and a corresponding open source project led by Google.” The Linux kernel code which Android is based on is licensed under GPLv2. Most Android source code is licensed under the Apache 2.0 license. Android’s open nature means that mobile phone device manufacturers can access it freely, modify it, and use the software according to the requirements of any device. This is one of the primary reasons for Android’s popularity. But be aware that customer devices may still contain some closed source software components such as bootloaders, firmware, DRM, and apps.

Fragmentation

The open nature of Android drives many different device manufacturers to build their own Android devices. Each device includes OEM-specific or carrier-specific modifications to help it distinguish itself from others for competition. Fragmentation has become a unique and significant issue for Android due to the diverging variants of the Android platform. This in particular, results in more pain for application developers to support the variety of devices. This fragmentation issue also creates great challenges for mobile forensics examiners to handle all different devices.

Compatibility

Compatibility issues are frequently coupled with fragmentation issues. Google made some effort to address this problem by providing compatibility guidelines to ensure device manufacturers comply with compatibility requirements. In particular, the Android Compatibility definition document and Android CTS (Compatibility Test Suite) are the results of this effort.

Update

The complexity of the update process is another consequence of the fragmentation issue. It also causes the biggest security threats on many Android devices, which are unable to update the system in time, especially with security fixes. Compared to iOS, the new versions of Android are adopted much more slowly. As previously

mentioned, many Android users still use older versions. Therefore, they cannot take advantage of security features in the newer Android versions. Many of these users may be vulnerable to known attacks.

The updates for applications and platforms are handled differently. An application update can be directly deployed by the developer. But the platform update needs to be done through a firmware upgrade or OTA (Over The Air) update. Google usually patches any security vulnerabilities within days or weeks of discovery. However, OEMs and carriers may take weeks or months (or never) to include these fixes in their customized builds and deliver them to end users, or may never do so. The time varies widely. The Nexus (and now Google Pixel) lines from Google are usually faster than others.

The updates for applications and platform are handled differently. An application update can be directly deployed by the developer. But the platform update needs be done through a firmware upgrade or OTA (over-the-air) update. Google usually patches any security vulnerabilities within days or weeks of discovery. However, OEMs and carriers may take weeks or months to include these fixes in their customized builds and deliver them to end users, or may never do so. The time varies widely. The Nexus (and now Google Pixel) lines from Google are usually faster than others.

Security

Android's open nature provides much more flexibility than other closed mobile platforms such as iOS. It also opens the door for more security vulnerabilities and attracts attackers to exploit. According to a recent report from Pulse Secure, the majority of mobile malware still targets Android devices.

The Android ecosystem is very complex. It has a number of different stakeholders, including Google, hardware vendors, carriers, developers, and users. The status of a mobile device, particularly the security status, is affected by the contributions of all stakeholders.

Test Yourself

True or False: All code in the Android system is open source code.

True

False

False. The customer devices may still contain some closed source software components such as bootloaders, firmware, DRM, and apps.

Test Yourself

Explain the fragmentation issue.

A variety of versions of the Android platform, combined with a mixture of hardware result in the inability for some devices to properly run certain applications. As an app developer, it is also hard to ensure all Android devices can properly run the app.

The Android Platform

Hardware

Similar to PCs, the hardware for smartphones includes CPU (central processing unit), main memory (volatile), permanent storage, and is accessed via I/O (imput/output) devices, such as the keyboard or touch screen. However, there are some major differences between the two.

On a smartphone, hardware cannot be swapped out as easily as it can be on a PC. The memory, storage, WiFi hardware, screen, keyboard, etc., are built in, and can't be upgraded and usually can't be repaired or replaced if broken. With smartphones, a "hardware upgrade" usually means replacing the entire phone.

The device is intended to run mostly on battery power. When smartphones are designed, hardware elements are chosen or designed with the battery life in mind.

The various hardware features of a smartphone are:

- Microprocessor (CPU)
    - Most are **ARM-based**
    - Speed of chip varies from 1GHz up
    - Number of cores: 2, 4, 8, 16
    - 32 or 64-bit
- Volatile (temporary) storage
    - RAM (Random Access Memory)
    - Capacity ranges from 512MB to 4GB
- Nonvolatile (long term) storage
    - NAND flash memory built in
    - Capacity ranges from 4GB to 64GB
    - Some have external SD (Secure Digital) card slots
- Display
    - Size and dimensions: 4", 5", 7", 10"
    - Different screen resolutions
- Keyboard
    - Hardware-based or touch screen
- Microphone, speaker

- Camera
    - Front and rear
    - Differ in picture resolution, video capability, etc.
- Baseband modem/radio for timing-sensitive communications
- Wireless communication technology
    - WiFi
    - Bluetooth
    - NFC (Near Field Communication)
    - IrDA (Infrared Data Association)
- Positioning System
    - In USA, GPS (Global Positioning System)
    - Outside USA, GNSS (Global Navigation Satellite System) and its various flavors: GPS, GLONASS, Galileo, Beidou
- Battery: the battery life is usually in the number of hours
- Optional sensors such as accelerometer, gyroscope
- USB port
- SIM card

In a smartphone, data is stored in the NAND flash memory. NAND stores the boot loader, OS, and user data packaged as an MCP (multi-chip package). Both internal, built-in memory and external SD cards use NAND flash memory. A number of different file systems can be used with NAND, such as EXT4 (Android) and HFS+ (iOS).

The advantages of NAND are that it has a high storage density and is resistant to shocks, bumps, temperature changes, etc., making it ideal for mobile devices. It has no mechanical platters, and thus is faster than magnetic disks.

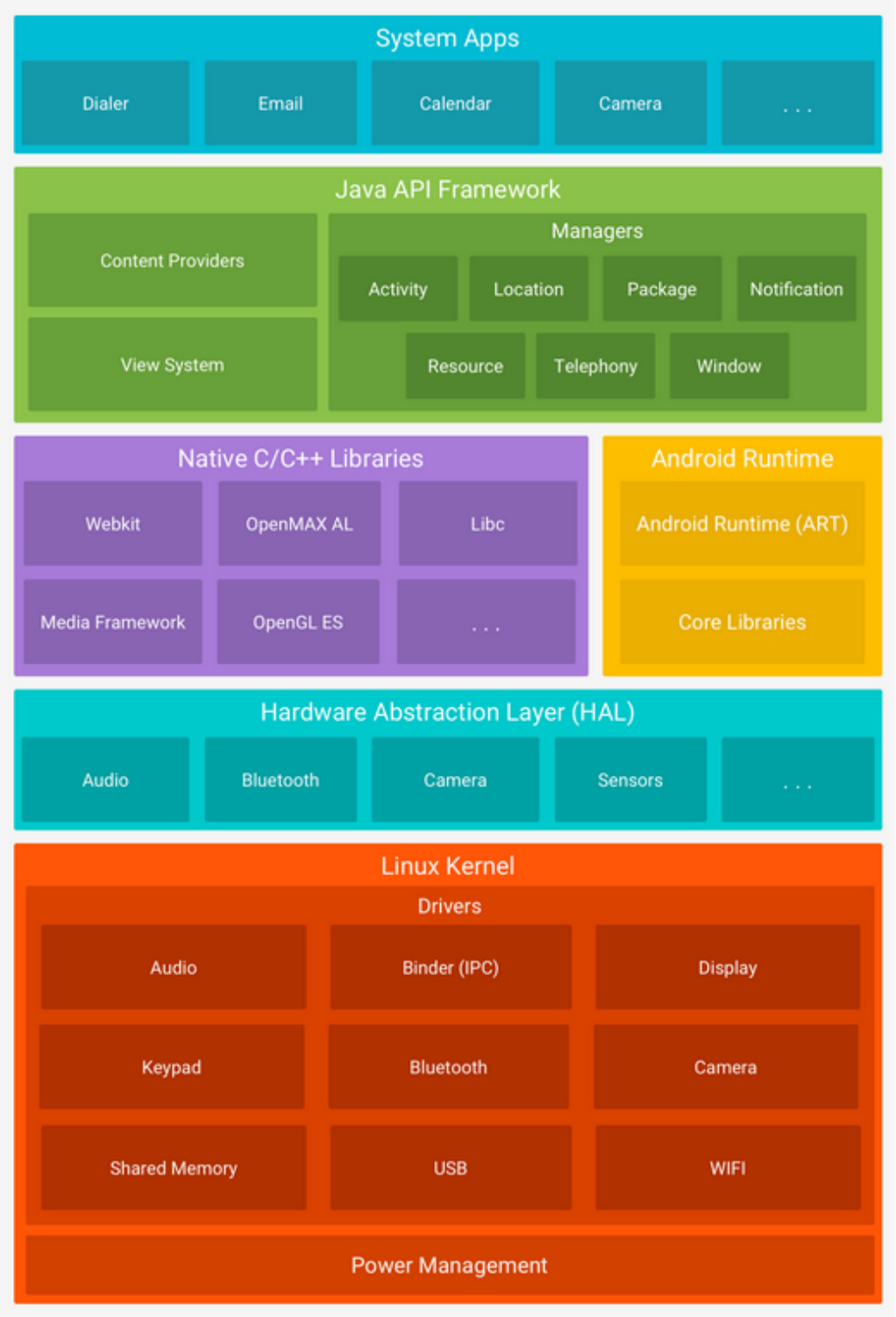
The disadvantage of NAND is that it has a high error rate and is often shipped with bad blocks and has to use ECC (error correction code) to compensate. It also “wears out” after a number of write operations.

The NAND memory is organized in pages. Several pages then are combined in the a block. There are three NAND operations:

- read(page) - Reads a page during normal operation of the device
- write(page) - Writes data to a free (or empty) page. Once a page is written, it cannot be overwritten, until it is erased. To overwrite the content of a page, it needs to find another free page and write the updated data on that new page, then mark the current page as dead.
- erase(block) - Erases the entire block of dead pages to make them free again. Once the page is made free again, it can be written to again, and the cycle continues. However, a NAND has a limited erase lifespan. A typical NAND flash can support 10,000 or more writes.

The Android OS

The following diagram shows various layers involved in the Android software stack.



(Source: Android Developers)

Kernel

The bottom layer is a Linux-based kernel. It implements the core functionality of an OS and is the lowest layer above the hardware. It manages basic system resources such as CPU, memory, storage, I/O, battery, and networking. Similar to Linux, it provides low-level process-based isolation. That is, each process is running in a

separate virtual address space. Processes cannot access each other's memory and can communicate with each other only through defined IPC (inter-process communication) channels.

Android is based on Linux, but is not exactly the same, since Android is mainly targeted to mobile devices, while Linux is more for desktop systems. Android includes some additional drivers and functionalities to address special requirements of mobile devices.

Here is a list of additional drivers:

- Binder: Facilitates the IPC mechanism. While the Linux kernel supports other traditional IPC mechanisms such as message queues, signals, sockets, and files, Android applications communicate with each other mostly through ICC (inter-component communication) supported by the Binder driver.
- As mobile devices usually have much smaller memory than desktop systems, memory management should be treated differently. The following three drivers address memory management issues in mobile devices.
    - ashmem (anonymous shared memory): Provides a file-based, reference-counted memory interface for low-memory environment.
    - pmem (persistent memory): Manages large, physically contiguous memory.
    - Viking Killer: An OOM (out of memory) killer; the basic logic is to kill the least recently used process when memory is low.
- Logger: Provides an additional logging subsystem.
- Paranoid networking: Provides finer network control by restricting certain networking features to specific group IDs.

It also includes some additional functionality to Linux such as:

- Wakelocks: A power management feature to keep a device from entering a low-power state and staying responsive.
- Yaffs2 (Yet Another Flash File System) file system.

Both Linux and Android constantly update their versions. The specific linux kernel version used depends on the actual Android device and its hardware platform. Android 1.0 is based on Linux 2.6. Android 4.4 KitK is based on Linux kernel 3.10. Android 5.0 Marshmallow uses Linux kernel 3.18. Android 8 Oreo requires Linux kernel 3.18 or above. However, all SoCs productized in 2017 must launch with kernel 4.4 or newer.

Hardware Abstraction Layer (HAL)

Since Android can be used on a variety of hardware, the hardware abstraction layer (HAL) provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework. Each specific type of hardware component is implemented in a separate library module, such as camera module. The modules are loaded and called by related APIs.

Libraries and Runtime Environment

On top of the kernel and HAL, Android provides a number of native libraries such as libc, SSL and SQLite. The Linux kernel is written in C, and these Android libraries are written in C/C++. These libraries can directly request kernel services through system calls, and provide support to the above application framework layer.

Android runtime environment enables Android application to run inside a process VM (Virtual Machine), which provides better isolation and security. While Android applications are mainly written in Java, Android does not use a Java VM, but instead used Dalvik or ART (Android Runtime). Android applications are first compiled to Java bytecode, then translated to Dalvik bytecode to run in Android runtime. In particular, Dalvik and ART are designed to use less space and computation power for mobile systems.

Here are some key points about Dalvik:

- Used in early Android versions (Android versions 4.4 "KitKat" and earlier).
- Register-based VM. Uses JIT (Just-In-Time) compilation.
- Dalvik bytecode is stored in .dex (Dalvik EXecutable) or odex (Optimized Dalvik EXecutable) formats. All Java bytecode .class file are converted to a single .dex file.
- Tries to reduce the memory footprint and optimize memory usage.

Here are some key points about ART:

- Used starting with Android versions 4.4 "KitKat"
- Compatible to run DEX bytecode.
- AOT (Ahead of Time) compilation to run app faster.
- Improved garbage collection.
- Enhanced performance and battery efficiency.
- Android 7 adds a JIT compiler with code profiling to ART to improve the performance.

The Application Framework

The application framework includes a number of managers to provide key services to Android applications, including activity manager, view system, package manager, telephone manager, resource manager, location manager, and notification manager. It provides a set of APIs to develop apps for Android. It is written in Java and also executed inside VM. The common framework packages are those within the android.* namespace (e.g. android.content, android.telephony). If you want to become an Android app developer, you need to have a good understanding of this framework layer.

Android Libraries

Here are a list of packages in the Android Libraries included in the Android Framework

- android.app
- android.content, android.database, android.provider

- android.text, android.view, android.widget
- android.util, android.os, android.hardware
- android.graphics, android.opengl
- android.media
- android.net

Test Yourself

What are the unique features of the NAND flash memory used in smartphones compared to magnetic disks used in general desktop systems?

It is faster than magnetic disks because it does not have mechanical parts. However, NAND pages cannot be overwritten. Erase is performed on the whole block. It has limited life span in terms of erase cycles.

Test Yourself

The Android kernel is based on the Linux kernel but with some modifications. List some differences that result from these modifications. Why are these modifications necessary?

Android targets on mobile devices. It has limited memory and battery life.

Test Yourself

Android uses ART (or Davik in the older versions) instead of Java VM. Why? Do some research to find out the differences between ART and Java VM.

To use less memory and computation power.

Android Applications

Android applications are run at the top of the Android stack, which includes both system apps and user-installed apps. System apps are pre-installed and shipped along with the device. They may have higher privileges than user-installed applications.

Most Android applications are written in Java and executed inside the VM. The Java programs are first compiled into Java bytecode objects (.class files). Then all .class files together with any .jar files are then converted into a single Dalvik bytecode object (.dex file). Finally, the .dex file is zipped together with all data and resource files into a single .apk (Android Package) file.

If you need to use C/C++ code with Android, the Android NDK (Native Development Kit) provides support for this. Additionally, you can use Kotlin to develop Android applications. Kotlin has become an official programming language for Android. It's totally interoperable with the existing Android languages and runtime, and is more expressive, concise, and powerful.

Android Components

The key components of an Android application are described as following:

- The manifest file: The file **AndroidManifest.xml** is provided by the app developer for each app. It includes package and version information, components definition, permission declaration, external library information and shared user ID information. It will also define all the following components in the application.
- Activities: UI components that the user can interact with.
- Services: Components running in the background without UI.
- Content providers: Structured interfaces to common shared data stores. They are usually backed by a SQLite database or a direct file system path.
- Broadcast receivers: Components that defines the broadcast messages to be received and actions to be performed when they are received.
- Intent: Message objects for ICC (inter-component communication).

Here is an example of AndroidManifest.xml which declared all four different android components.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="edu.bu.metcs.cs683example">
   <uses-permission android:name="android.permission.INTERNET"/>
   <application
       android:allowBackup="true"
       android:icon="@mipmap/ic_launcher"
       android:label="@string/app_name"
       android:roundIcon="@mipmap/ic_launcher_round"
       android:supportsRtl="true"
       android:theme="@style/Theme.MyApplication">
       <activity android:name=".Activities.MyActivity"
           android:exported="true">
           <intent-filter>
               <action android:name="android.intent.action.MAIN"/>
               <category android:name=
```

```
                    "android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <provider
            android:name=".ContentProvider.MyContentProvide"
            android:authorities=".ContentProvider.MyContentProvide"
            android:permission="My_PERMISSION"/>
        <receiver android:name="Receiver.MyReceiver"
            android:exported="true">
            <intent-filter>
                <action android:name
                    ="android.intent.action.BATTERY_LOW"></action>
            </intent-filter>
        </receiver>
        <service android:name=".Service.MyService"/>
    </application>
</manifest>
```

We will discuss each component in detail in later modules. You can also find more details in the Android Developer Guides.

Test Yourself

Fill in the blanks: In the following Android components, _____ is used to define user interface. _____ is used to handle background tasks. _____ is used to handle data share. _____ is used to receive broadcast messages.

a. Activity
b. Service
c. Content Provider
d. Broadcast Receiver

Test Yourself

Fill in the blanks: Each android app needs to have a _____ which defines all components in the application. It is named _____.

Manifest file; AndroidManifest.xml

Test Yourself

Fill in the blank: _____ are used to communicate between different components.

Intents

Topic 2: Intro to Android Application Development Using the Software Engineering Approach

Intro to SDLC and Agile

SDLC (Software Development Life Cycle)

While this course mainly focuses on Android programming, we should understand that software development (or software engineering) is more than just programming. As part of this course, you are required to develop a real world Android application that should be a useful app either for yourself and/or for other users. This is a self-defined project. Every student must propose an idea, analyze the requirements and implement it. The goal is to develop a useful and high quality Android app within a semester. To successfully achieve this goal, we will first need to understand the software development life cycle which involves various activities including programming.

SDLC (Software Development Life Cycle) is a process that aims to produce high quality software within a given time and budget. It provides a structured framework to guide various activities involved in the development process. In general, these important activities (phases) are:

- Inception: research similar systems; define the scope, major functionality, and targeted customers of the software system to be developed
- Planning: estimate the cost (in terms of man hours), define work items, schedule, resources, and high-level activities; perform feasibility study and also plan configuration management
- Requirements Analysis: gather and analyze functional and nonfunctional requirements
- Software Design: design the software architecture and design detailed classes
- Implementation: programming
- Testing: include unit-testing, integration testing, and system testing
- Maintenance: defect repair and quality enhancement
- Project Management: perform throughout the life cycle of a project; include risk management, quality management, configuration management, etc.

Agile SDLC

TThere are mainly two types of SDLC models. One is the sequential model, in which activities are conducted sequentially, one by one. This is also called the waterfall model. It is simple and has been practiced for many years. It can work with small and well-defined projects. However, it is not suitable for real world applications today because it cannot cope with change and does not provide feedback until it is done. Another model is the iterative model, in which some or all activities are repeatedly conducted in order to cope with change and provide more feedback. The spiral model and unified process are two classical iterative models. In recent years, Agile SDLC has been popularized, aiming to execute and provide feedback in a short period of time.

The term “Agile” was coined in 2001 in the Agile Manifesto by a group of people who sought an alternative to the documentation-driven, heavyweight software development process. At its core, the manifesto declares four value statements representing the foundation of the agile movement:

1. **Individuals and interactions** over processes and tools
2. **Working software** over comprehensive documentation
3. **Customer collaboration** over contract negotiation
4. **Responding to change** over following a plan

There are a number of different agile processes. While they may focus on different aspects, they all use an incremental and iterative process. With short iterations, planning, requirement analysis, design, implementation and testing are repeatedly executed. We will use the same approach in our project. Specifically, we will have 6 iterations for our semester long project:

- Iteration 0: mainly focuses on inception and planning
- Iterations 1 - 4: each iteration includes requirement analysis, design, implementation, and testing
- Iteration 5: the final iteration

In each iteration, you will be required to submit a simple document as well as the source code of the project. In the final iteration, you will also be required to do a final presentation.

Test Yourself

Which of the following SDLC models use an iterative process? (Check all that are true.)

Waterfall

Waterfall does NOT use an iterative process.

Spiral

Spiral does use an iterative process.

Unified process

Unified process does use an iterative process.

Agile

Agile does use an iterative process.

Test Yourself

Restate the Agile Manifesto.

Main Phases in SDLC

ProjectPortal - An Example

To give a concrete idea of what should be done in each phase, let us use an example - ProjectPortal. ProjectPortal is an Android app used to manage all student projects done for this course, and possible other courses, as well. The initial idea is that the user (e.g. the instructor) can easily add, edit, search, and view all projects using this app.

Inception and Planning

First, we need to define the project vision in Iteration 0. We may have a lot of cool ideas. To be practical, we can classify the features into three categories: essential, desired and optional. Essential features are the important features the project must deliver at the end of the semester. Desirable features are good features we would like to finish too. Optional features are those cool features that we may implement if we have extra time. For example, we can define ProjectPortal to have the following features:

- Essential features: add a new project, delete a project, list all projects, display details of a project, search projects based on some keywords
- Desirable features: user sign up, user login, access control for the instructor, project author, and other users
- Optional features: grade a project (by the instructor), input comments by other users

In order to define the high level requirements, you need to first research related or similar applications existing in the market. In your proposal, you should clearly state your targeted customers and the similarities and differences between your app and other related apps.

Requirement Analysis

After defining the basic ideas of the project, we will need to define and analyze the requirements in more detail. User stories are widely used in the agile process to describe the requirements.

User Story

A user story is a specific, concise representation of a requirement. It should include a title, a short description, and acceptance tests associated with it.

The description of a user story commonly uses the following format:
“As a (role), I want to … (feature), so that … (value)”

Since it is hard to capture the details of a user story in the above short description, acceptance tests are very important to help define the details. When a user story is delivered, the customer will perform acceptance tests to check if the user story is completed correctly and decide whether the user story should be accepted or not. It commonly uses the following format: “Given … (setup), when… (input & action), then … (the expected result)”

Each acceptance test is a test case that clearly specifies the setup, the action to trigger the feature, the input data, and the expected output.

For example, we can define the addProject user story as the following:

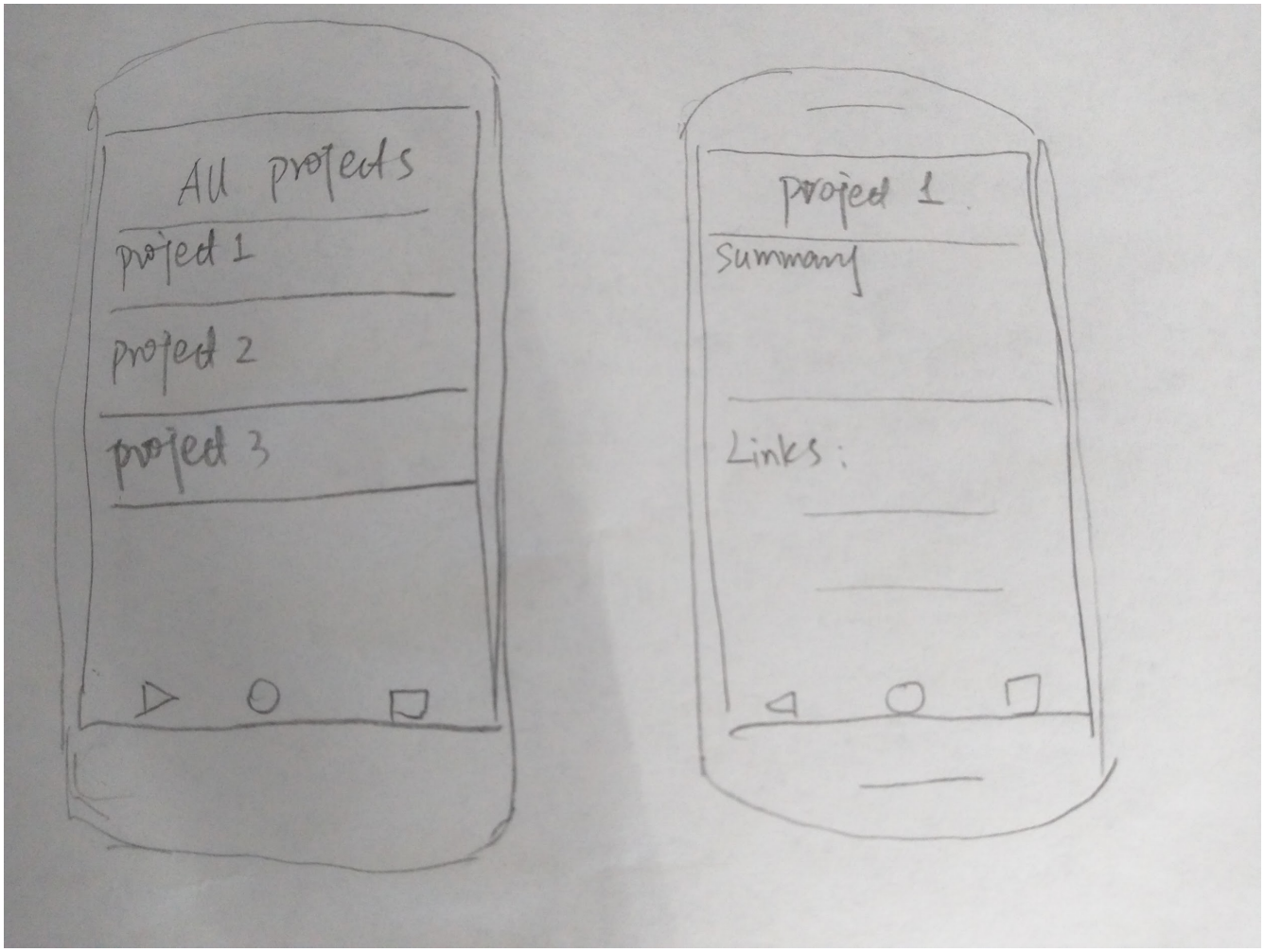
- Title
    - Add a new project
- Description:
    - As a user, I want to add a project into the system, so that I can view or search it later
- Acceptance tests:
    - Given the add button is on the screen, when the user clicks on the add button, then a new screen is displayed to let the user enter the project information.
    - Given the add project screen is displayed, when the users enters the project information including title, summary, authors, related links etc, and clicks the “submit” button, then a message, “the project is successfully added”, is displayed, and the newly added project is displayed on the screen in the project list.

Here is another example:

- Title: List all projects
- Description:
    - As a user, I want to view all projects in the system, so that I can have an overall idea about all projects.
- Acceptance test:
    - Given project 1 and project 2 are already added into the system, when the user clicks the app icon on the home screen, then project 1 and project 2 will be in the list on the screen.
    - Given project 3 is not added into the system, when the user clicks the app icon on the home screen, then project 3 is not displayed in the list on the screen.

To help visualize the application, simple mockup screens can be used. They can be either hand drawn or created using some wireframe tool. The idea is to show a basic UI design with little effort. For example, hand drawn

mockup screens for the above two user stories are shown below.



We can define a user story for each essential feature. The user stories should be prioritized. In the beginning of each iteration, we need to pick several user stories with the highest priority to implement in that iteration. To implement each user story, we need to break down each story into smaller tasks that involve the following tasks.

Design

There are two levels of design. High-level design is used to design the software architecture. Low-level design is used to design detailed classes.

MVC Software Architecture

The MVC (Model-View-Controller) architecture is commonly used today in various types of applications. The idea is to separate your software system into three subsystems:

- Model
- View
- Control

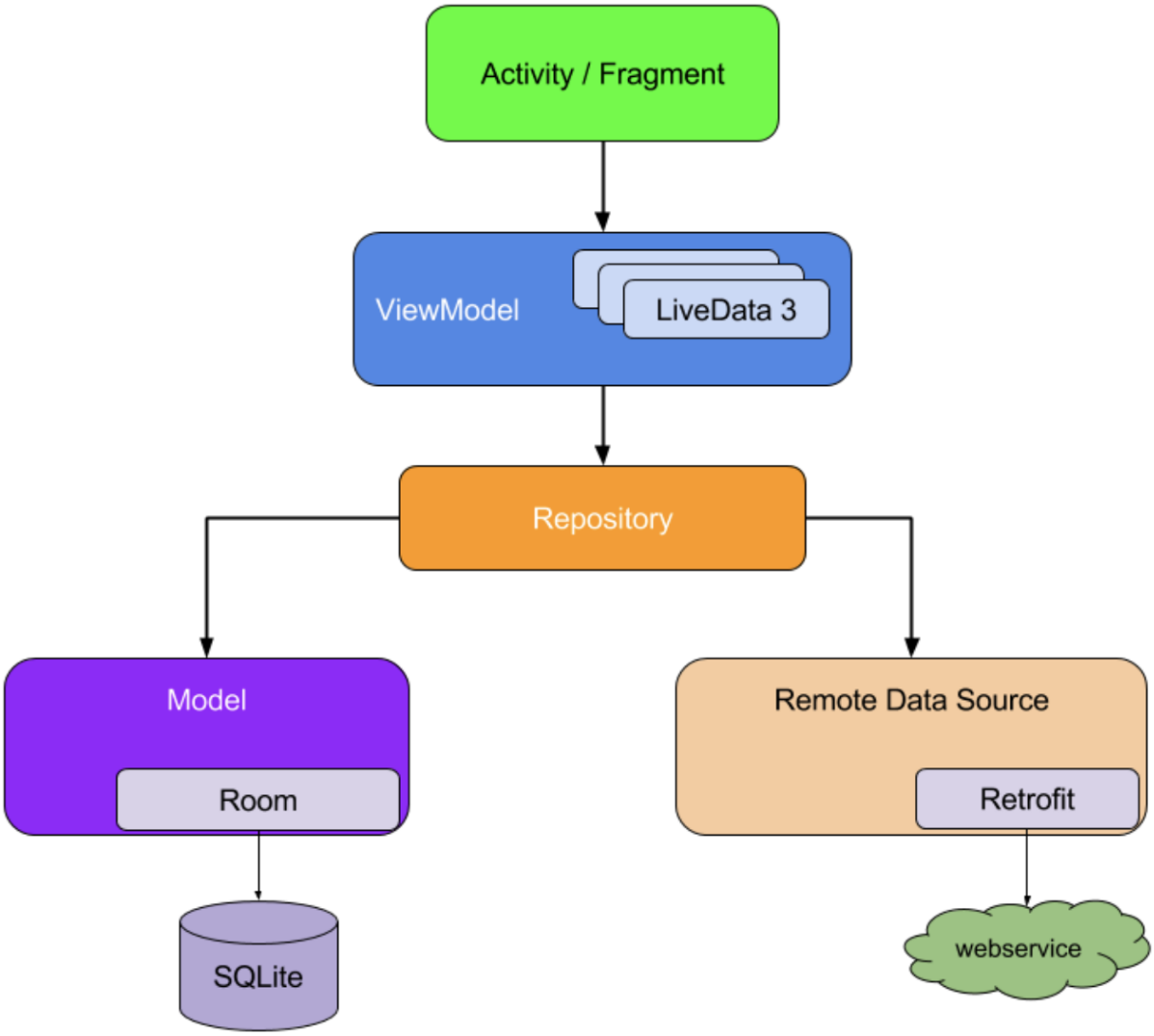
Model objects hold the application's persistent data and "business logic" that need to be maintained in the app. The model component is designed to model an application domain in the real world. For example, in the ProjectPortal app, a project should be a model class. Maybe later we would need to add "User" as a model class. Model objects are supposed to be in the lowest layer and have no knowledge of other layers such as UI. Their sole purpose is to hold and manage data and underlying business logic.

View objects are associated with the UI subsystem. They focus on drawing themselves on the screen and responding to the user input. Android provides a number of configurable view classes. The view objects are usually defined in the XML layout file.

Controller objects tie the view and model objects together. They contain "application logic". Controllers are designed to respond to various events triggered by view objects, and to manage the flow of data to and from model objects and the view layer. In Android, a controller can be thought of as a subclass of Activity, Fragment, or Service.

There are several MVC variants, such as MVP (Model View Presenter) and MVVM (Model View ViewModel). In the MVP architecture, the presenter is similar to the controller. But there is usually a one-to-one mapping between Presenter and VIew. In MVVM, the ViewModel is similar to the controller, but the main purpose of the VM is to expose data from the model to be consumed in the view. While there are different ways to describe the controller, the bottom line is that we need to separate the model from its presentation.

The MVVM architecture pattern is promoted by Google for Android application development. Google also adds another component "Repository" to support different data sources for models. Here is the MVVM diagram:



After we decide on the high-level software architecture, we can focus on define classes in each subsystem.

Implementation

Implementation is mainly the programming phase. This is also the focus of this course. To learn Android programming, you need to first be familiar with Java programming, which includes the basic syntax of Java and the object-oriented concepts implemented by Java. In particular, you should be familiar with the following object-oriented concepts:

- Encapsulation
- Inheritance
- Polymorphism

A powerful IDE (Integrated Development Environment) can be very helpful in programming. In this course, we will use Android Studio, the default Android programming IDE. We will talk about it later in more detail.

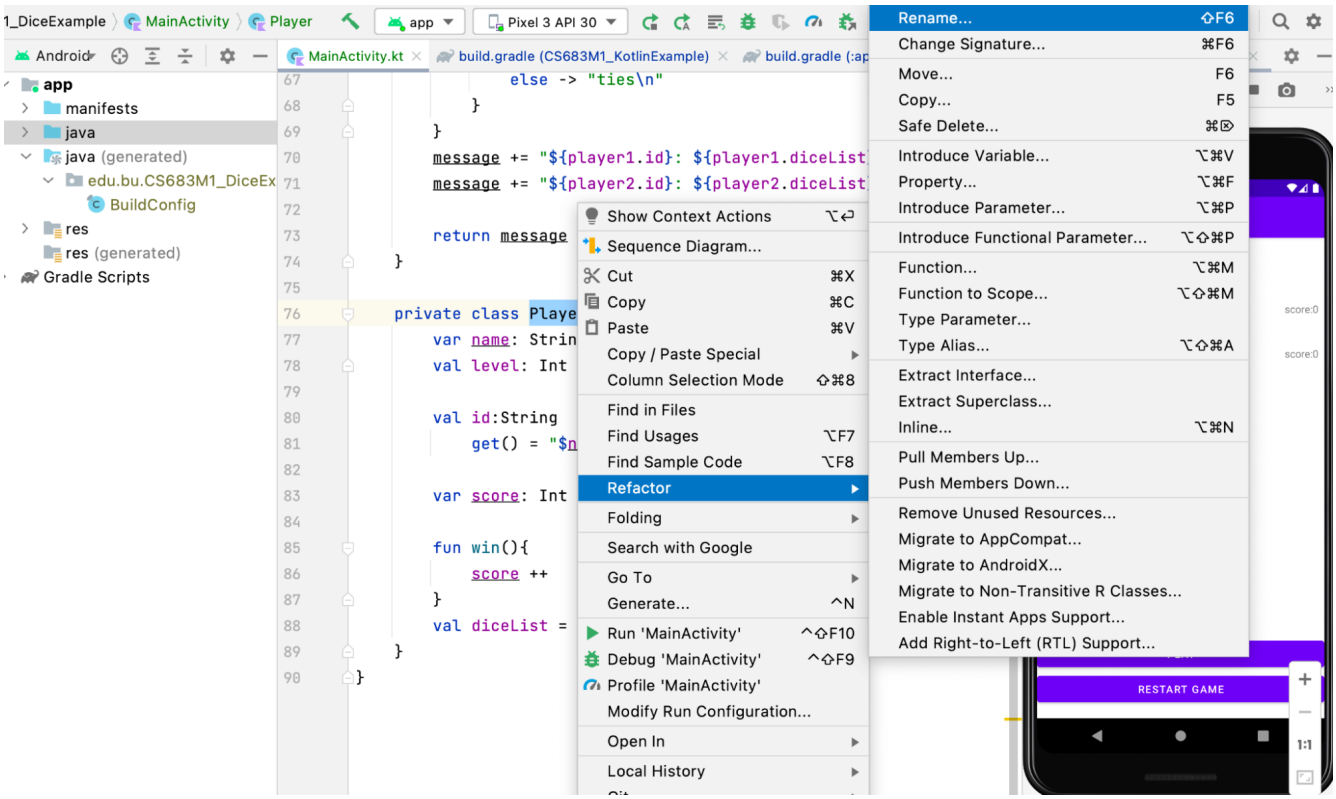
Refactoring

Just like when writing an article, our first draft of our program may not be very good even though it expresses our ideas. When you write your program, you may start in a simple and lazy way. Though it may work, the code might not be pretty. It may have a number of bad smells that causes it to be less flexible, maintainable, and extensible. Then we need to change the code structure without changing functionality. That is refactoring. Refactoring can help us clear up the code and make it neater and more elegant. We need to refactor the code at every reasonable opportunity, particularly before implementing new requirements. Here are some commonly used refactoring techniques:

- Change meaningless names to meaningful names
- Split a long method into multiple methods
- Move fields or methods from one class to another class
- Extract redundant code to a method or a class

For more details about bad smells and refactoring techniques, please visit SourceMaking.

Android Studio provides nice support for refactoring. To perform refactoring, you need to first choose the code you want to refactor and right click on it. A pop-up menu will be displayed, from which you can choose the proper refactoring action you would like to perform.



Testing

Testing is very important in the software development life cycle. Through testing, we can validate our implementation and find bugs. When performing the refactoring, it is also crucial to run the test again to check if it introduced new bugs. In general, both unit testing and integration testing need to be done. Testing can be done either manually or automatically. While manual testing is easy, it is also time consuming. Therefore, automatic testing is essential. When you create an Android application in Android Studio, it will also create two testing packages. You can write your unit test code in these two packages..

Test Yourself

What does MVC stand for?

Model View Controller

Test Yourself

Which activity(phase) is the most important or most challenging one in your opinion?

Test Yourself

Write a user story and a corresponding acceptance test for your proposed project.

Tools: Android Studio & Android SDK

Android Studio is the official IDE used to develop Android applications. The Android team recommends using it. However, since it is a special version of IntelliJ IDEA (IntelliJ IDEA is one of the most popular IDEs for Java development), customized for Android development, some people use IntelliJ IDEA instead.

- A powerful code editor
- Android SDK and other tools
- A fast and feature-rich emulator
- A flexible Gradle-based build system
- A unified environment that allows you to develop for all Android devices
- Instant Run to push changes to running apps
- Code templates and GitHub integration
- Extensive testing tools and frameworks
- Lint tools
- C++ and NDK (Native Development Kit) support

Android SDK

Android SDK (Software Development Kit) includes software libraries and APIs, reference materials, an emulator, and other tools.

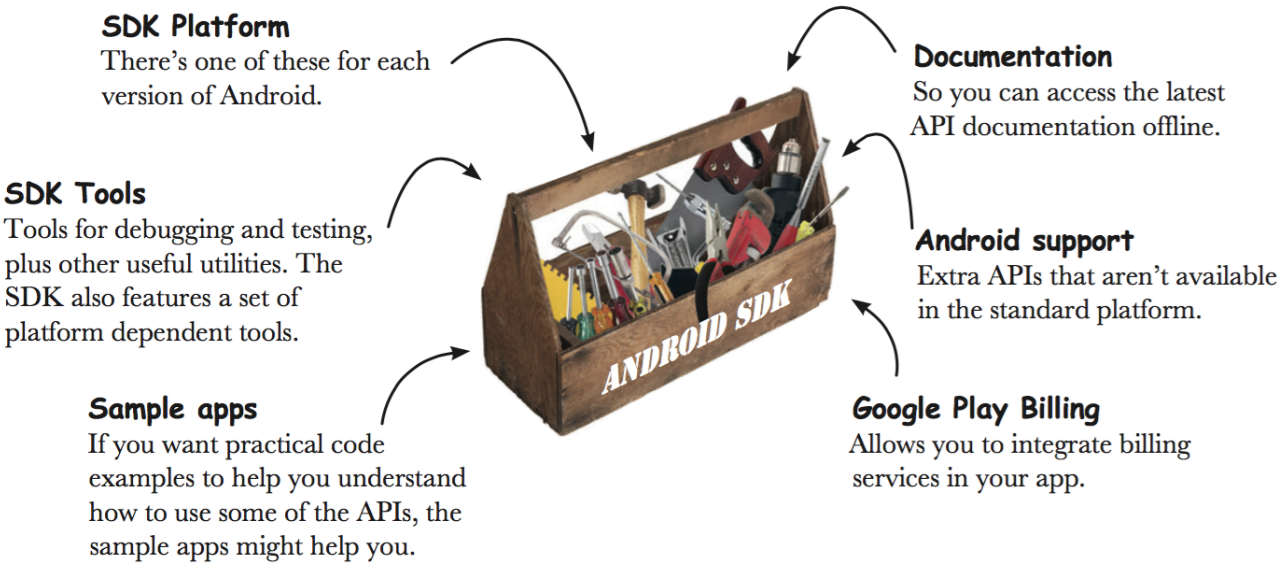


Figure: The Android SDK

(Source: Griffiths, D. & Griffiths, D. (2017). *Head First Android Development: A Brain-Friendly Guide* (2nd). O'Reilly Media)

AVD (Android Virtual Device/Emulator)

An AVD is a configuration that defines the characteristics of an Android phone, tablet, Android Wear, or Android TV device that you want to simulate in the Android Emulator.

When an AVD is created, a folder named .android is created (by default it is in your home directory), which holds all configuration and data files needed to run each avd (.avd, .ini). For more information, see Android's AVD webpage.

Gradle Build System

Similar to Ant and Maven, Gradle is an advanced build toolkit to automate and manage the build process, while allowing you to define flexible custom build configurations. It is scalable and extensible with powerful dependency management and uses Groovy build scripts.

Android Studio contains an Android plugin for Gradle to provide processes and configurable settings that are specific to building and testing Android applications. However, Gradle and the Android plugin can run independently of Android Studio.

Lint

Lint is a code-scanning tool to check your Android project source files for potential bugs. It also provides correction suggestions. The report includes a description message and a severity level for each problem identified. It can be configured in Android Studio to run whenever you build your app. It can also run manually from the command line. For more information, see Android's lint check webpage.

Test Yourself

Install Android Studio and necessary SDK tools on your computer.

Test Yourself

Create two AVDs on your computer with different configurations.

Creating a Very Basic Android App

To start Android development, first download and install Android Studio. After Android Studio is installed, you also need to install Android SDK.

If you have an Android device, you can run the application on your device, but make sure ADB debug is enabled. If you don't have an Android device, or you don't want to always run on a real device, you can run your application on a virtual device. You can use the AVD manager to create multiple virtual devices with different settings.

To create a virtual devices, open Android Studio, and click on the menu Tools -> Android -> AVD Manager to open AVD manager. Then, click on the button "+Create Virtual Device …" Choose a device definition, select the system image, and verify configuration. Click the "Finish" button, and an AVD will be created.

Now, let us create a very basic Android app in Android Studio. Open Android Studio, and choose "Start a new Android Studio project" to create a new project. First, you need to specify a template for the main activity. Android Studio provides a number of activity templates. For simplicity, we will choose "Empty Activity" now. Then, you need to provide a project name, a company domain and the project location. The package name is basically a combination of the application name and the domain name. In this case, our app package name is "edu.bu.projectportal". It uses a reverse DNS format. By default, Kotlin is chosen. You can also choose Java.

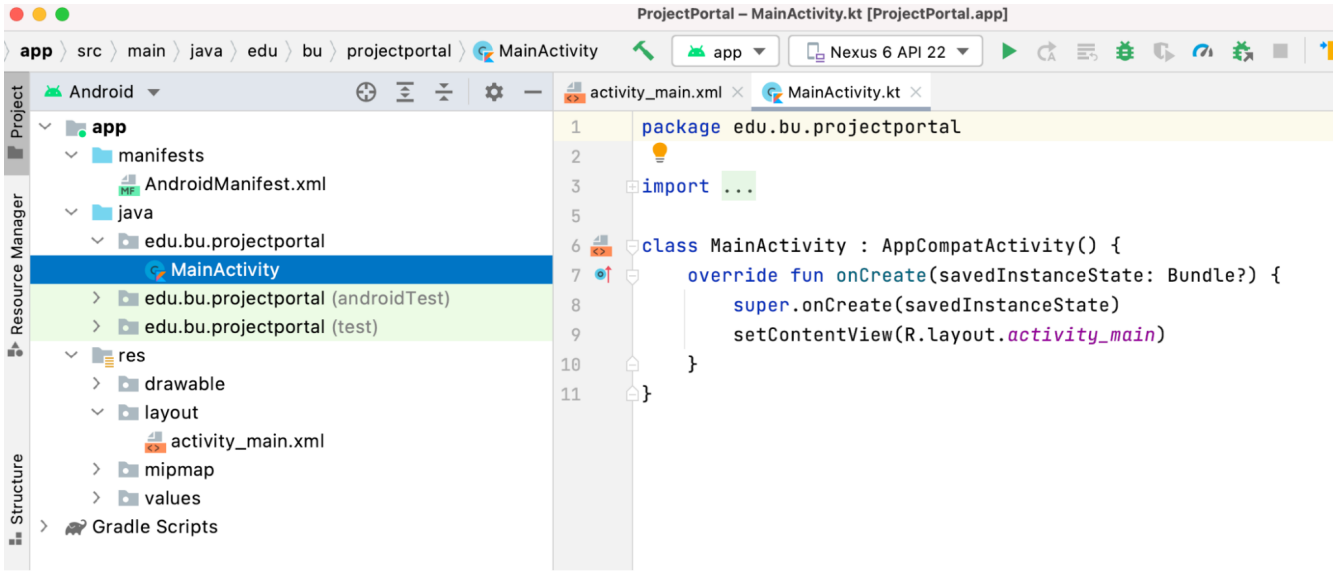
You also need to specify the target device specification. In particular, you need to specify the minimum SDK. Low API levels target more devices, but offer few API features, and are also less secure in general. Here I choose API 21, so the project should be able to run on Android 5 or above.

After you click on the **Finish** button, Android Studio will generate a project named ProjectPortal for you.

The following screenshot shows the project structure and MainActivity.kt in Android Studio.

Among all files generated, we only need to look at three files now:

- MainActivity.kt: This is the kotlin code for the MainActivity class.
- activity_main.xml: This is the corresponding XML layout file for MainActivity.
- AndroidManifest.xml: This is the manifest file for the project.



To get to know more about Android studio, please check the official website: Meet Android Studio

MainActivity.kt

This kotlin file defines the MainActivity class. For backward compatibility, it extends AppCompatActivity. Here we only override one callback function `onCreate()`, which calls setContentView() to set and inflate the layout file to the screen.

An activity is an entry point for a UI screen. A typical Android app consists of one or more UI screens. Each screen is usually associated with two files. A layout XML file defines what the screen looks like and an activity subclass Java file loads the layout file and defines how the user can interact with the screen. In this simple example, MainActivity simply specifies and loads the corresponding layout file.

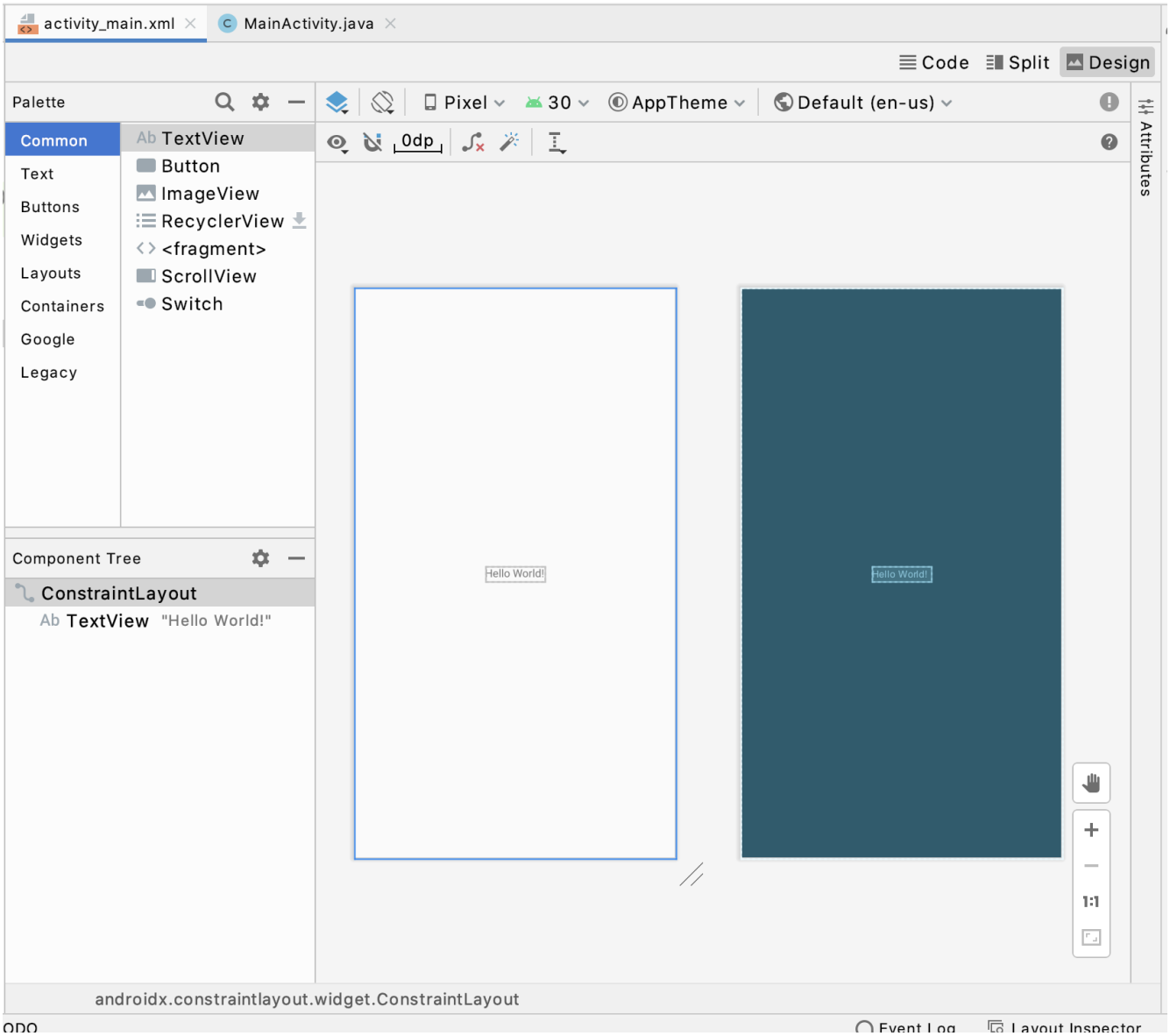
activity_main.xml

The following screenshot shows the layout XML file. There are two modes to show this file. One is the Text mode, which shows the detailed code of this XML file. The other is the Design mode, which gives you a visual interface, in which you can essentially drag and drop the widgets to change the layout. Using the Text mode, we can see this XML file has a top level component `ConstaintLayout` and an inner component `TextView`. The `TextView`

component has an attribute `android:text`, the value of which is `"Hello World!"`. Using the Design mode, we can see a single textbox displaying “Hello World!” in the middle of the screen. If you want to display something else, you can simply change this attribute.

activity_main.xml | MainActivity.java

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".MainActivity">
9
10     <TextView
11         android:layout_width="wrap_content"
12         android:layout_height="wrap_content"
13         android:text="Hello World!"
14         app:layout_constraintBottom_toBottomOf="parent"
15         app:layout_constraintLeft_toLeftOf="parent"
16         app:layout_constraintRight_toRightOf="parent"
17         app:layout_constraintTop_toTopOf="parent" />
18
19 </androidx.constraintlayout.widget.ConstraintLayout>
```



Types of Resources

Besides the layout file, there are several other types of resources stored under the /res folder. Each type of resource is stored in a separate folder.

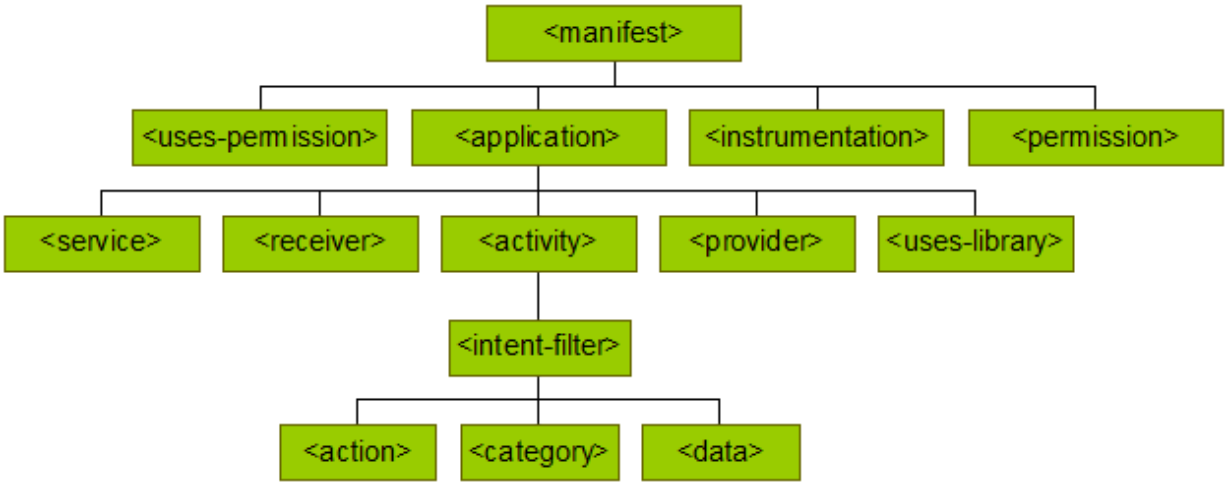
- /anim: XML representations of frame by frame animations
- /drawable: .png, .9.png (see below), and .jpg images
- /layout: XML representations of View objects
- /raw: arbitrary and un-compiled files
- /values: XML representations of strings, colors, styles, dimensions, arrays
- /xml: user-defined XML files. The XML is processed by Android Asset Processing Tool (AAPT) and compiled into class R

AndroidManifest.xml

activity_main.xml | AndroidManifest.xml | MainActivity.java

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.bu.projectportal">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="ProjectPortal"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

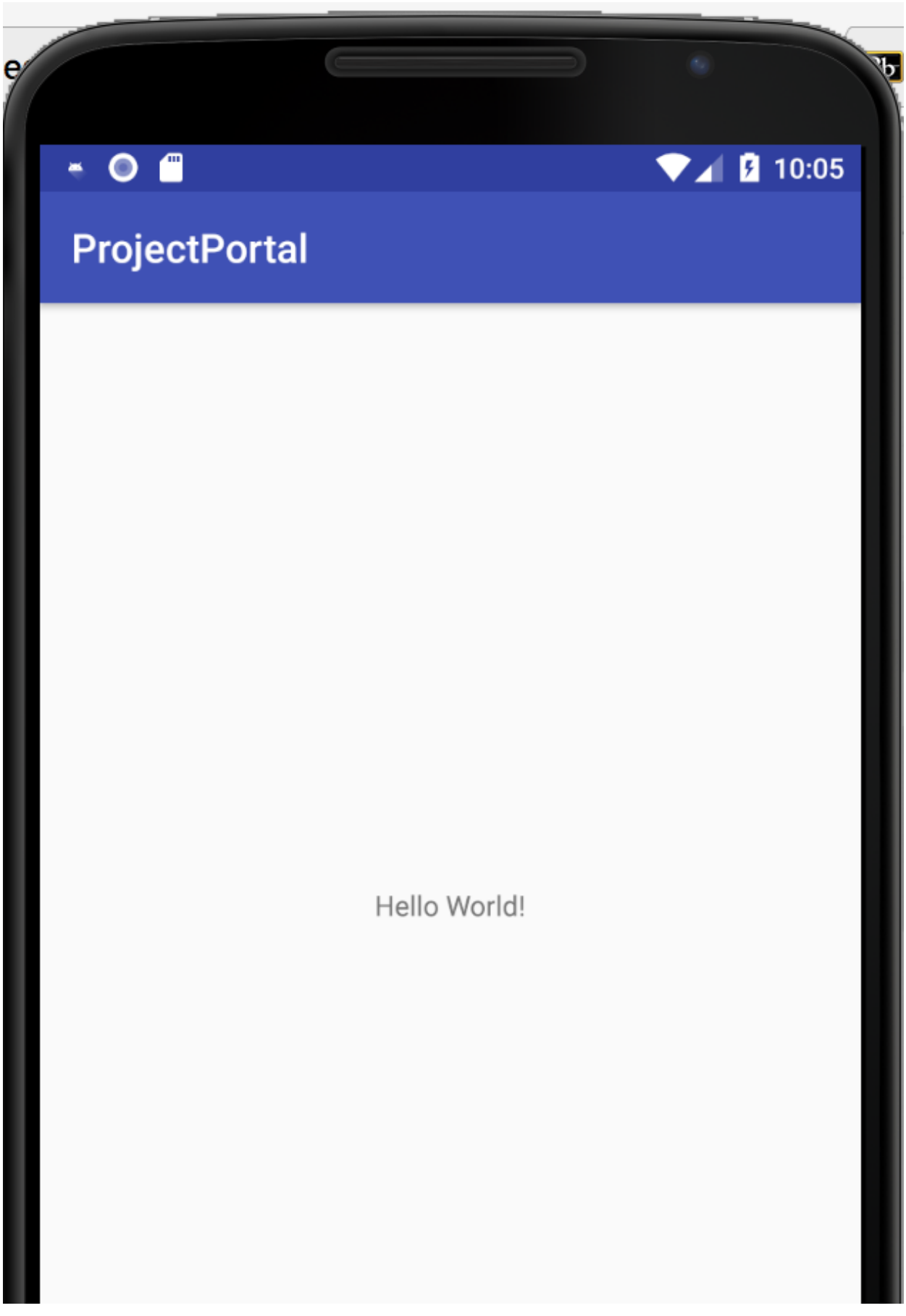
The AndroidManifest.xml file is required for every Android app. The name has to be AndroidManifest.xml and it resides in the source code root directory. It defines a lot of information about the application, including the package name and all components in the app. Each element in the manifest file is defined using a tag. Only the <manifest> and <application> elements are required, and each can occur only once. An element can contain other elements. The following diagram shows the general structure of elements in a manifest file.



Each element is described by its attributes. Except for some attributes of the root <manifest> element, all attribute names begin with an “android:” prefix. Commonly used attributes include android:name, android:icon, android:label, android:permission, etc.

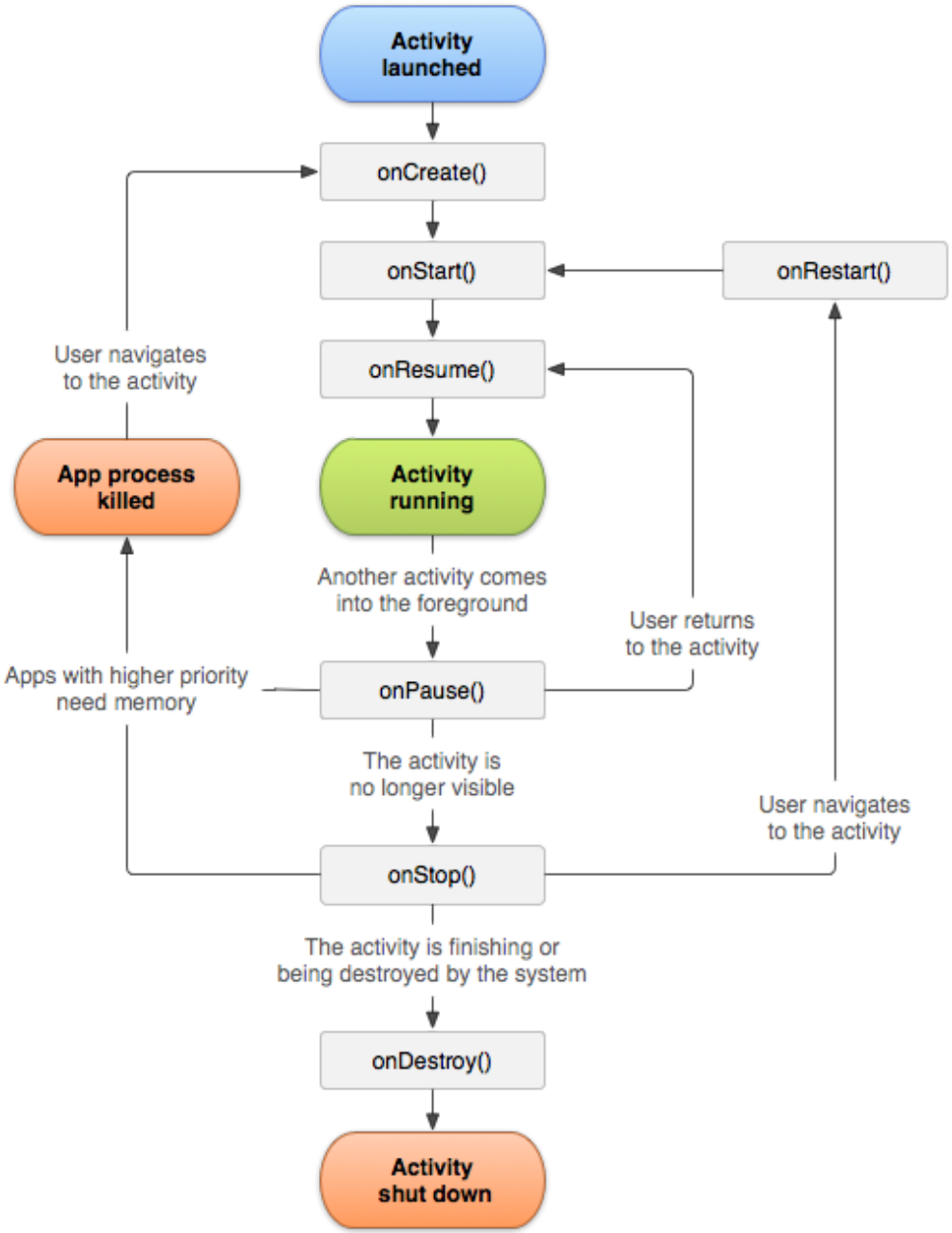
In this simple app, we can see this application only has one component, an activity named MainActivity. This activity is also associated with an intent filter with the MAIN action and the LAUNCHER category, which means that this activity will be started when the application is launched.

Click on the menu item Run->Run ‘App’, and Android Studio will compile the code, build the project apk file, install the apk file to the target AVD or physical device and execute the app. Now, this simple app only has a single screen that shows “Hello World!” as shown in the following screenshot.



Activity Callbacks and the Lifecycle

In the above simple example, we only used the `onCreate()` callback, which is called when an activity is created. There are several other callbacks. Each is associated with a different state of the activity.



(Source: Android Developers, 2018)

onCreate()	Must implement. Called when the activity is first created.
onStart()	Called when the activity is made to visible to the user (the activity enters the foreground)
onResume()	Called after onStart(). Bring the activity to the foreground. Can also called after onResume() to bring the activity to the foreground again after some interruption, such as a phone call.
onPause()	Called when the activity loses its focus because of another activity or event. Usually it is used to release the system resource that are not needed in this state.
onStop()	Called when the activity is no longer visible. Usually it is used to release the system resources that are not needed in this state.

onDestroy()	Called before the activity is destroyed. This is the final call that the activity receives. This may be called when changing the device orientation. Changing the device orientation may destroy current activity and recreate it again.
onRestart()	Called after the activity has been stopped, prior to it being started again. Always followed by onStart().

Android Logging

To have a better understanding of the activity lifecycle and these callback functions, let us add logs in each of these callbacks. Android uses a centralized logging system defined by a final class Log in the android.util package (A final class means that you cannot subclass it). It includes several static methods to log the messages. The following tables list all log methods from the most important to the least important ones.

Log.wtf()	Log an error message when a terrible failure happens
Log.e()	Log an error message
Log.w()	Log a warning message
Log.i()	Log an informative message
Log.d()	Log a debug message
Log.v()	Log a verbose message

Each log function has either 2 or 3 parameters, such as:

- Log.d(String tag, String message)
- Log. d(String tag, String message, Throwable tr);

Now add the following logging code in each callback and check the output in the Logcat. You can try to do the following and check the log output:

- Start another activity
- Bring back the previous activity
- Close the activity
- Rotate the device

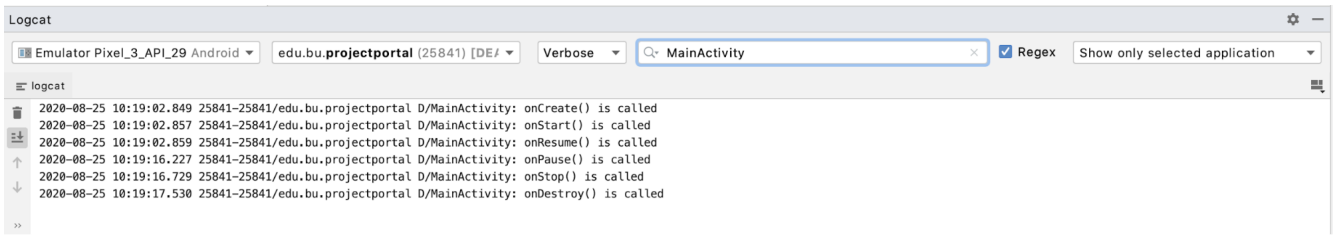
```
class MainActivity : AppCompatActivity() {
```

```
 override fun onCreate(savedInstanceState: Bundle?) {
     super.onCreate(savedInstanceState)
     setContentView(R.layout.activity_main)
     Log.d(TAG, "onCreate")
 }

 override fun onRestart() {
     super.onRestart()
     Log.d(TAG, "onRestart")
 }

 override fun onStart() {
     super.onStart()
     Log.d(TAG, "onStart")
 }

 override fun onResume() {
     super.onResume()
     Log.d(TAG, "onResume")
 }

 override fun onPause() {
     super.onPause()
     Log.d(TAG, "onPause")
 }

 override fun onStop() {
     super.onStop()
     Log.d(TAG, "onStop")
 }

override fun onDestroy() {
     super.onDestroy()
     Log.d(TAG, "onDestroy")
 }
```

```
    companion object {
        private const val TAG = "MainActivity"
    }
}
```

For more details about Log, please check Android's Log webpage.

Here is a sample log output when various actions are performed. To check the log information, you click the logcat and apply a filter to see just this log information from this application.



Test Yourself

What is the name of the manifest file?

AndroidManifest.xml

Test Yourself

What are two files in your Android projects associated with a UI screen?

An XML layout file and a Java activity class

Test Yourself

Which callback of an activity must be overridden?

onCreate()

Test Yourself

What are states of an activity? Which callbacks are called when you close an activity? Which callbacks are called when you open another activity?

6 states; onDestroy(); onPause()

Introduction to Kotlin

The default programming language used by the newer Android studio version is Kotlin. When you create a new project, the MainActivity.kt will be automatically generated, and all Kotlin configurations will be added. If you are not familiar with Kotlin and prefer to write the code in Java, the Android studio can help you convert the Java file to the Kotlin file as shown below. As Kotlin is fully interoperable with Java, you can have a project with mixed Java and Kotlin files. However, we encourage you to use Kotlin in both the lab and your project. All examples in this course will be provided in Kotlin.

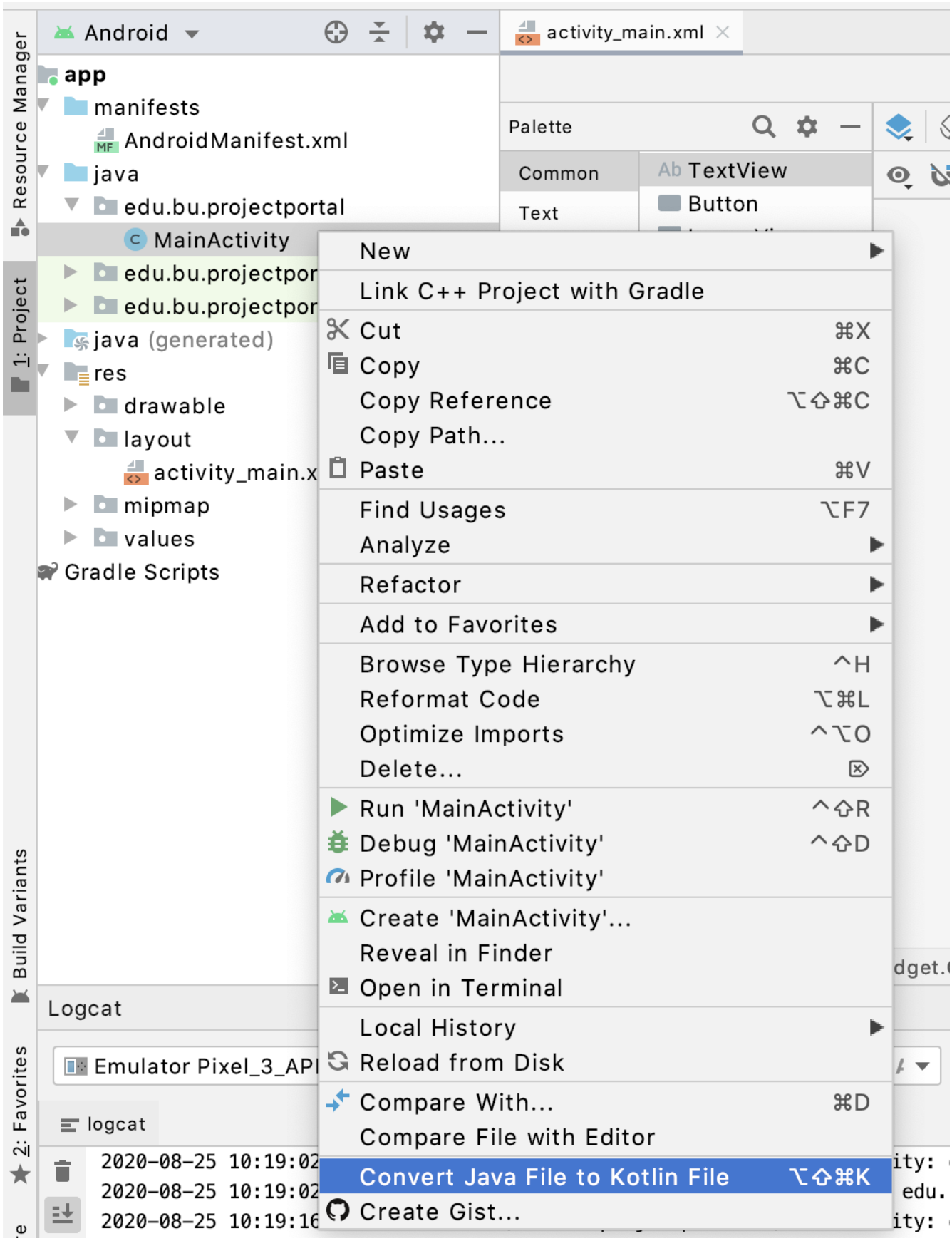


Figure 3.1 Screenshot: Convert Java to Kotlin in AS

Kotlin is a modern programming language designed by JetBrains to interoperate fully with Java. Like Java, it is a cross-platform, **statically typed**, general-purpose programming language. The JVM version of Kotlin's standard

library depends on the Java Class Library. Different from Java, it can use **type reference** which allows its syntax to be more concise. Kotlin has been chosen as the preferred language for Android app developers by Google since May 2019. It also plans to support different platforms, such as:

- Multiple mobile platforms, including both Android and iOS
- Web development for both server side framework and front end using Kotlin/JS

There are several Kotlin compiler options, Kotlin/JVM, Kotlin/JS, Kotlin/Native.

You can find more details about kotlin on the kotlinlang website. There are also many tutorials online. The goal for this module is not to fully grasp Kotlin, but get to know some basics about Kotlin to get us started. We will only focus on how to use Kotlin in Android development. If you are familiar with Java, here is the comparison to Java.

Basic Syntax

An overview of basic Kotlin syntax is given on the Kotlin official website . More details can be found at: Kotlin-Grammar.

In this section, we will highlight some basic Kotlin syntax used in basic Android apps. For example, the semicolon ";" is not needed in Kotlin. The colon ":" is used for type specification as well as inheritance.

As you can see in the MainActivity.kt, it uses many of the same keywords as Java, such as **import, package, class**. There are also some different keywords, such as

- **fun:** to define a function
- **open:** make it not final in order to change it. By default, a class is final in Kotlin. Therefore, you cannot subclass it. If you want to inherit from it, you need to make it explicitly **open**. Similarly, a function is final as well, and you cannot override it. You also need to make it explicitly open in order to override it in the subclass.
- **override:** to override an open function inherited from an open class.

Nullable and null check

Null pointer exception is one of the most common issues in Java programs. Kotlin attempts to eliminate the problem of null references by **differentiating nullable and non-null types**. When declaring a variable, it is NOT nullable by default. That is, it cannot be null. If you want a nullable variable, you need to append a ? at the end. For example, str1 is a nullable variable, and str2 is a non-nullable variable.

```
var str1:String?
var str2:String
```

To do a null check for nullable variables, we can use a safe **call operator ?**.. With a safe call operator, the operation will only be carried if the object is not null. Otherwise, it returns null, instead of throwing a Null Pointer Exception (NPE). For example, the following expression will return a null if str1 is null. Otherwise, it returns the length of str1.

```
str1?.length
```

If the object is null, and you don't want to return a null for the operation, then use the Elvis operator ?:.For example, This expression will return -1 if str1 is null.

```
Str1?.length?:-1
```

If you want to throw a NPE explicitly, you can use the **not-null assertion operator (!!)**. It converts any value to a non-null type and throws an exception if the value is null.

Try to execute the following code at Kotlin Playground. What result do you see? Are there any errors? If yes, how to fix them?

```
var str1:String? = null
var str2:String = null
println(str1?.length)
println(len = str1?.length?:-1)
println(str2.length)

str1 = "hello"
str2 = "world!"

println(str1?.length)
```

```
println(str1?.length?:-1)
println(str2!!.length)
```

Selection

The `if/else` selection statements are used in the same way as in Java.

```
if (cond)
  statement1
else
  statement2
```

The `if/else` can also be used directly in an expression, such as:

```
fun maxOf(a: Int, b: Int) = if (a > b) a else b
```

The `When` statement in Kotlin is similar to the Switch statement in Java.

```
when{
        cond1 -> statement1
        cond2 -> statement2
      ...
        else -> statement
```

It can also used directly in an expression:

```
fun describe(obj: Any): String =
   when (obj) {
       1          -> "One"
```

```
        "Hello"    -> "Greeting"
        is Long    -> "Long"
        !is String -> "Not a string"
        else       -> "Unknown"
    }
```

Loop

Similar to Java, Kotlin supports the “for” loop, the “while” loop and the “do-while” loop. It also has more concise ways to represent ranges and steps in the for loop.

```
for (i in 1..3) {
    println(i)
}
for (i in 6 downTo 0 step 2) {
    println(i)
}
```

```
var x = 0
var sum = 0
while (x <100) {
    x ++
    sum +=x
}
```

```
var x = 0
var sum = 0
do {
    x ++
```

```
    sum +=x
}while (x < 100)
```

Types

Kotlin is a static type language as Java. Each variable has a type and it cannot be changed dynamically.

It can be specified directly using ":" or inferred based on the context.

```
val i: Int
var i = 5
```

In Kotlin, there are no primitive types. Everything is an object which has both member functions and properties.

Here are some commonly used types:

- Byte, Short, Int, Long: `123, 456L, 0xa2d, 0b0102`
- Float, Double: `3.4554, 39.23f, 8.3e2`
- Boolean: `true, false`
- Char: `'1', 'a', '\n', '\uFF00'`
- String: `"hello world", "abc!\n"`
- Array: `arrayOf(1, 2, 3), IntArray(5){42}`

Try to execute the following code at Kotlin Playground. What result do you see?

```
var str = "abc!"
var str1 = "str: $str\n. It is ${str.length} char(s)\n"
var str2 =
    """ abc
    ....
    def"""
println(str)
println(str1)
println(str2)
```

```
var anArray = arrayOf("one", "two", "one")
var anIntArray = IntArray(4){1}
var anotherStrArray = Array<String>(4){"one"}
anArray.forEach{println(it)}
anIntArray.forEach{println(it)}
anotherStrArray.forEach{println(it)}
```

Collections

Here are several different collection types:

- List: a collection of elements in a specified order and each has an index.
- Set: a collection of unique elements without order.
- Map: a collection of key-value pairs with a unique key for each pair.

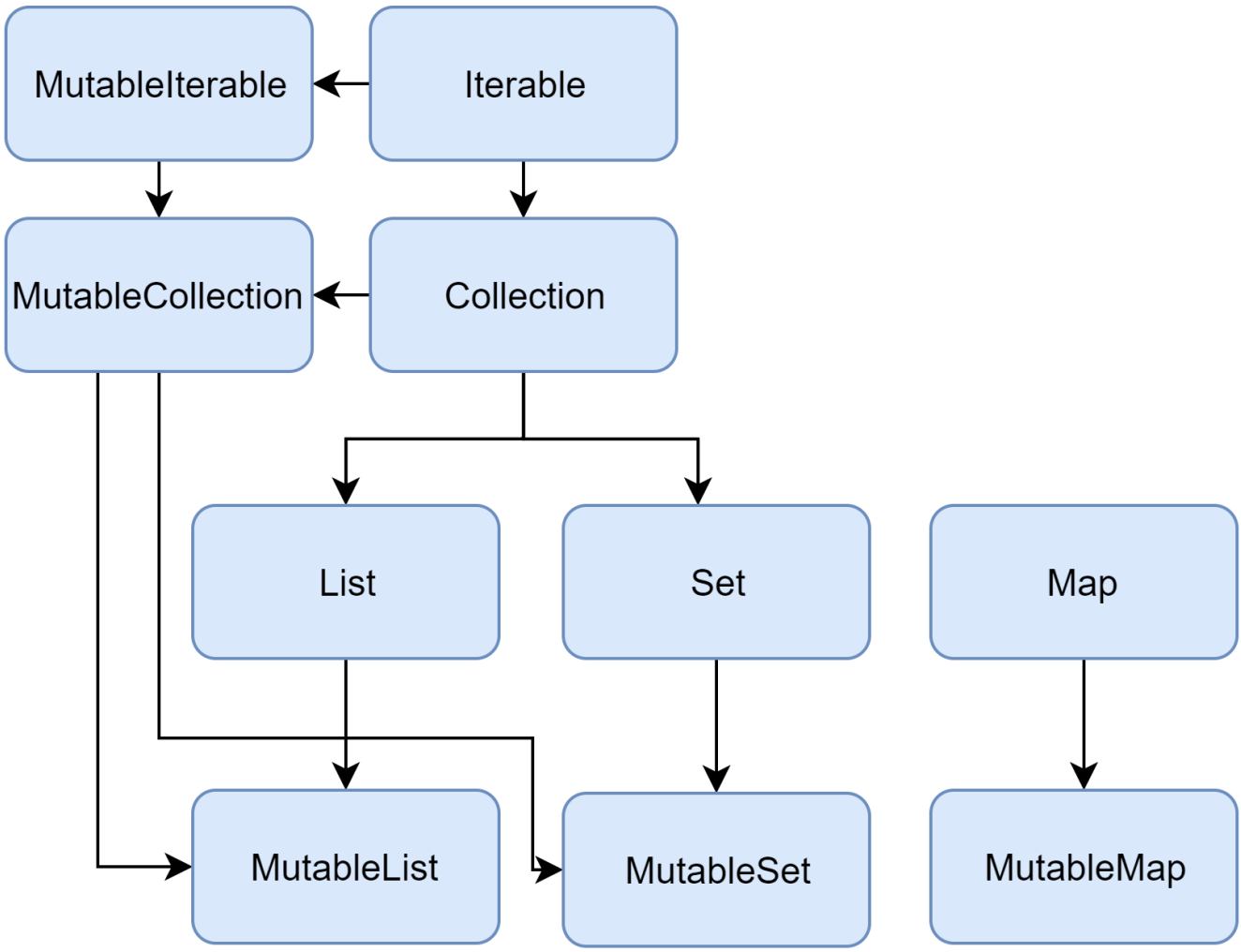


Figure 3.2 Kotlin Collections

In Kotlin, the read only collections are separated from mutable collections. You can only change the elements in a mutable collection. However, a mutable collection doesn't need to be a var. Changing elements of a collection is not the same as changing the collection object itself.

Try to execute the following code at Kotlin Playground. What result do you see? Are there any errors? If yes, how to fix them?

```
val stringList = listOf("one", "two", "one")
val stringSet = setOf("one", "two", "one")
val numbersMap = mapOf("key1" to 1, "key2" to 2, "key3" to 3)
val stringMutableList = mutableListOf("one", "two", "one")

stringList.add("three")
stringMutableList.add("three")

println(stringList)
println(stringMutableList)
println(stringSet)
numbersMap.forEach{k,v->println(k + ": " + v)}
```

Kotlin Functions

Kotlin functions are first-class citizens. Just like a variable or a value, a function can be assigned to a variable, passed as an argument or returned from another (high-order) function. It can be operated as for other non-function values. As Kotlin is statistically typed, functions need to have a type too. It is called a function type. For example, the following function type is for a function with two parameters with the type of the first parameter as A, and the type of the second parameter as B, and the return type is C.

```
(A,B) -> C
```

Can you write the types of the following function declarations?

Test Yourself

Write the type of the following function declaration.

```
fun sum(a:Int,b:Int):Int
```

Type the function here.

▶ Click to see the answer.

Test Yourself

Write the type of the following function declaration.

```
fun f()
```

Type the function here.

▶ Click to see the answer.

Test Yourself

Write the type of the following function declaration.

```
inline fun repeat(times: Int, action: (Int) -> Unit)
```

Type the function here.

▶ Click to see the answer.

Function Literals

Lambda expressions and anonymous functions are function literals. No function declaration is needed. They can be used directly in an assignment statement or as a parameter or as an expression. The return type of the lambda

expression can be inferred directly from the expression. If you need to declare the return type explicitly, you can use anonymous functions.

Here are a few examples of using lambda expressions or anonymous functions directly in an assignment statement, or an expression or in a high order function as a parameter. Try to identify the function type of the following examples.

```
max(strings, { a, b -> a.length < b.length })

val sum = { x: Int, y: Int -> x + y }

val product = items.fold(1){ acc, e -> acc * e } //trailing lambda

ints.filter { it > 0 } // this literal is of type '(it: Int) -> Boolean'

strings.filter { it.length == 5 }.sortedBy { it }.map { it.uppercase() }

ints.filter(fun(item) = item > 0)
```

Try to execute the following code at Kotlin Playground. What result do you see?

```
var anArray = arrayOf("one", "two", "one", "three", "four")
anArray.filter{it.length < 5}
                .sortedBy{it}
                .map{it.uppercase()}
                .forEach{println(it)}
```

Class

Same as in Java, the “class” keyword is used to declare a class. However, the class declaration can be much more concise in Kotlin. The initializer, getter, and setter are optional if it can be inferred. It can also have a default prime constructor, as part of the class head. One can also initialize some member fields later using the `lateinit` modifier.

```
//the Keyword constructor is usually omitted in the primary constructor)
class Person ~~constructor~~(val firstName: String, val lastName: String,
                    val isEmployed: Boolean = true){
   lateinit var jobtitle: String
   /*...*/
}
```

The **abstract** concept and usage in Kotlin is the same as Java. A member function can be abstract. A class can be abstract which has one or more abstract methods.

However, different from classes in Java, a class in Kotlin is a final class by default, which means that the class cannot be subclassed. We need to use the keyword "**open**" to make it explicitly inheritable. We also need explicit modifiers for overridable members and **overrides**. In Java, a class is inheritable by default and we need to use the keyword "**final**" to prevent inheritance.

Test Yourself

One of the basic types in Kotlin is int.

True

False

It should be Int.

Test Yourself

Which of the following expressions can throw a NPE (Null Pointer Exception) in Kotlin?

str.contains("a")

str?.contains("a")

str?.contains("a")?:"no"

str!!contains('a')

Test Yourself

Write a code snippet to create a list to hold the dice number in the dice game?

Test Yourself

What is the function type of the apply() function? (Kotlin Apply)

((T.() -> unit)->T

Conclusion

This module gave a brief overview of the Android platform and Android application development. As the most popular mobile platform, Android has its unique features and challenges. We discussed the Android ecosystem including hardware, operating systems, and applications. We briefly introduced various components in an Android application. While this course focuses on the programming aspect, we should understand that software development is more than just programming. This module introduced various phases in a software development cycle, and we will execute these phases in six iterations of our semester project using an Agile process. We also provide a brief overview of Kotlin, and particularly address some differences with Java. We will not discuss Kotlin separately in the later modules. Instead, we will use Kotlin code examples to help students learn how to develop Android apps in Kotlin.