Android Fundamentals

Introduction

What is Android?

- “Android is a software platform that delivers a complete set of softwares for mobile devices, including an operating system, a middle-ware and key mobile applications” (Google).

The Android SDK

- The Android platform is available to Java programmers through a software development kit, known as the Android SDK.
- The Android SDK supports most of the Java platform Standard Edition (Java SE) except for the Abstract Windowing Toolkit (AWT) and Swing.
- Instead of the AWT and Swing, the Android SDK has its own extension modern UI framework.

The Android Runtime

- The Android platform has its own optimized Virtual Machine. Till Android KitKat it was known as Dalvik. As of Android L it is replaced with ART.
  https://source.android.com/devices/tech/dalvik/art.html

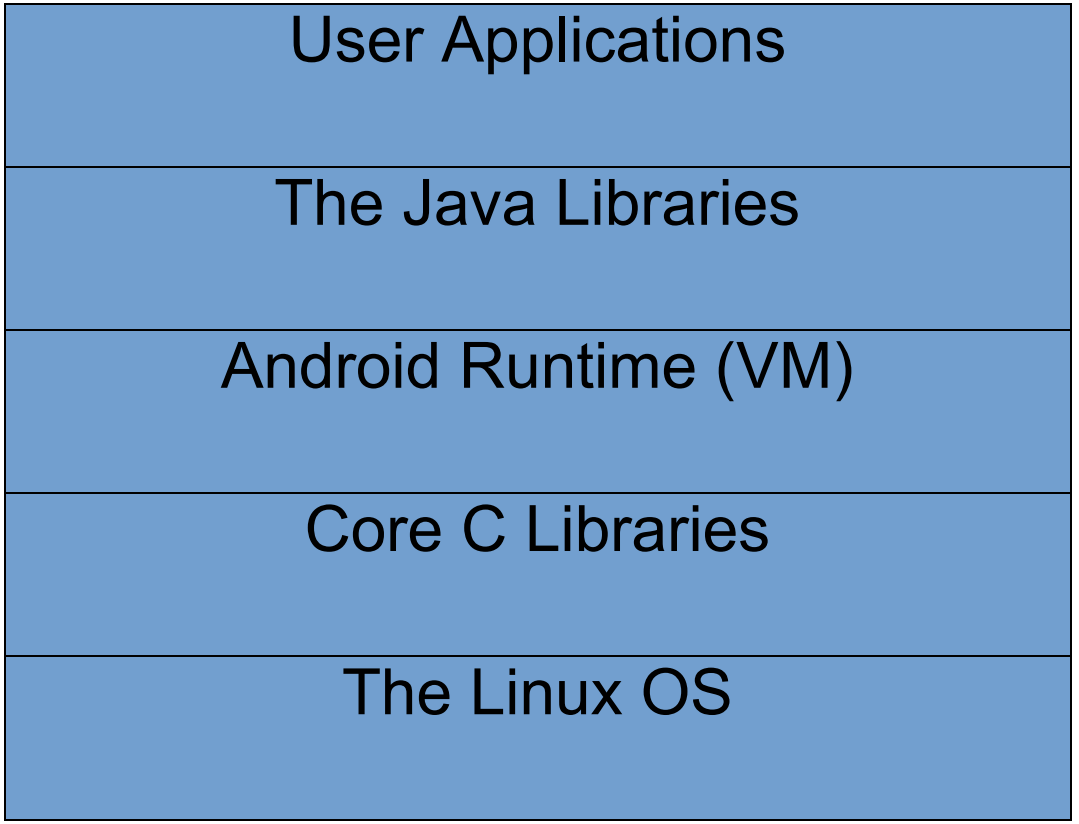
The Android Runtime

- The main user related improvements ART provides are Ahead-of-time (AOT) compilation that takes place when the application is installed, tighter install-time verification and improved garbage collection.

The Android Runtime

- The improvements ART provides for developers include the support for a sampling profiler without the overhead known in traceview, the support for more debugging features and improved diagnostic details when exceptions take place.

The Android Software Stack



Android Development Tools

- The Android Studio includes the IntelliJ IDE together with the required plugin installed into it. The Android Studio should be configured to work with the Android SDK.
- You can alternatively sill use the Eclipse IDE with the ADT plugin. You can get the Android Studio installed on your desktop ready for use in one simple download that includes the IDE, the ADT plugin and the Android SDK.
  http://developer.android.com/sdk/installing/studio.html

Android User Interface

- The Android UI, Microsoft Sliverlight, Mozilla XML User Interface (XUL) and JavaFX belong to the fourth generation UI frameworks, in which the UI is declarative and independently themed.
- Developing a Java application for the Android platform we declare the user interface in XML files.
- Screens are usually activities composed of fragments. Each fragment has its view.

Installing the Development Tools

- The first steps include installing the Java Development Kit (JDK), the IDE and the Android SDK. Once installed, there is a need to install the required additional plugin into the IDE.
- When installing the Android Studio you get it installed bundled with the plugin, configured and ready for use in one simple installation.

Federation of Components

- The android platform is kind of a federation of various different components of the following types: activities, services, content providers and content providers.

The Android Activities

- The android activity is a user interface single screen in our application. Each android activity usually contains one or more views.

The Android Content Providers

- Through the 'Content Provider' mechanism we can share data without exposing the underlying storage and the underlying implementation. Through 'Content Provider' our application can use other applications' data and other applications can use ours.

The Android Services

- The android service is similar in its nature to services we know from the windows operating system.
- Local services are accessible by activities that belong to the same application to which the service belongs. Remote services are accessible by activities that belong to other applications as well.
- We can use already existing services as well as creating our own by declaring a class that extends Service.

The Android Intents

- The Android SDK presents a new and unique concept. The Intent defines an “intention” to do a specific work (e.g. send a message, start a service, launch an activity, dial a number, answer a phone call etc.).
- The intents might be initiated by your applications as well as by the system (e.g. an intention that represents an arriving message).

The Android Intents

- The android intent defines an intention to do some work. We can use intents to perform various tasks, such as 'broadcasting a message', 'starting a service', 'launching an activity', 'displaying a web page' etc.

The Android Views

- The android view is kind of a canvas self drawn on the screen.

The Android Configuration File

- The `AndroidManifest.xml` file is the basic configuration file that defines our application. It plays a similar role to `web.xml` in Java EE web application.
- This file provides the essential information about the application to the android system. The android needs that information before it can execute the application.

The Android Configuration File

- Every application must have the `AndroidManifest.xml` file. The name cannot be different. The file must be within the root directory.

The Android Configuration File

- The `AndroidManifest.xml` names the primary Java package of the application. This package name serves as a unique identifier for the application.

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

    <application ... >

    ...

    </application>

</manifest>
```

The Android Configuration File

- The `AndroidManifest.xml` describes the application's components (activities, services, broadcast receivers, and content providers), names the classes that implement each one of the components and publishes their capabilities, including the Intent messages they can handle.

The Android Configuration File

- The `AndroidManifest.xml` lists the required permissions the user should provide in order to use the application.

```
<uses-permission android:name="string"
                 android:maxSdkVersion="integer" />
```

The `<uses-permission>` should be a child element of the `manifest` element.

The Android Configuration File

- We can use a variant of the `uses-permission` element that will take effect if (and only if) the application is running on API with the specified level (or higher).

```
<uses-permission-sdk-23 android:name="string"
                        android:maxSdkVersion="integer" />
```

The Android Configuration File

- The `AndroidManifest.xml` specifies the minimum Android API version number required for running the app.

  The `<uses-sdk>` element should be within the `<manifest>` element.

```
<uses-sdk android:minSdkVersion="integer"
   android:targetSdkVersion="integer"
   android:maxSdkVersion="integer" />
```

The Android Configuration File

- The `AndroidManifest.xml` lists the libraries the application must be linked with. The `<uses-library>` element should be child of the `<application>` element.

```
<uses-library
      android:name="string"
      android:required=["true" | "false"] />
```

The Android Configuration File

- Using the `uses-feature` element the device is informed about the set of hardware and software features on which the application depends. It is possible to specify whether the application requires and cannot function without the declared feature or whether it prefers to have that feature but can work without it.

The Android Configuration File

- The `uses-feature` element should be child of the `manifest` element.

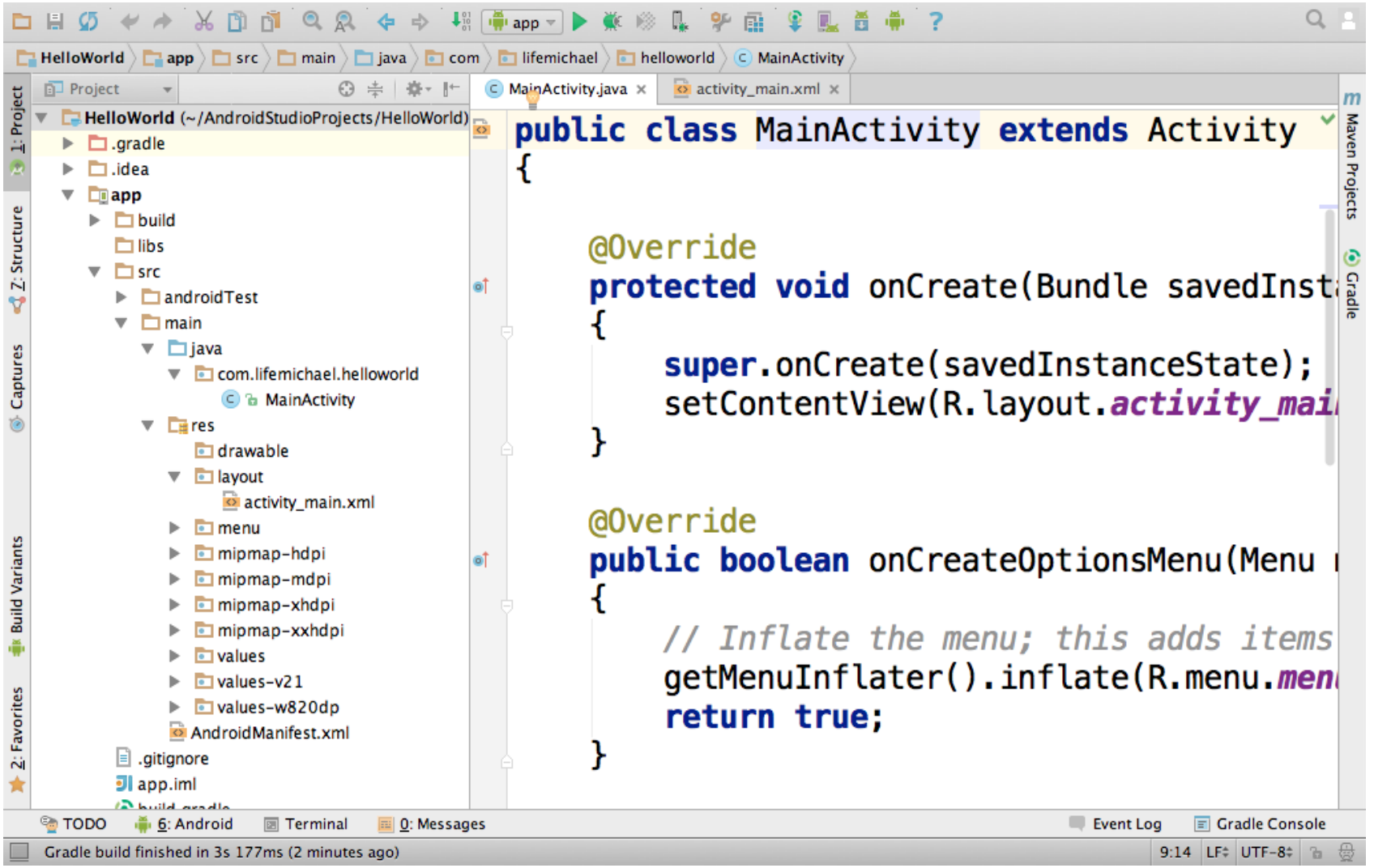
```
<uses-feature
      android:name="string"
      android:required=["true" | "false"]
      android:glEsVersion="integer" />
```

Hello World

- Start a new project your IDE. It should be an Android project.

  Make sure you choose a meaningful application name. This is the name that will be shown in the application's title bar.

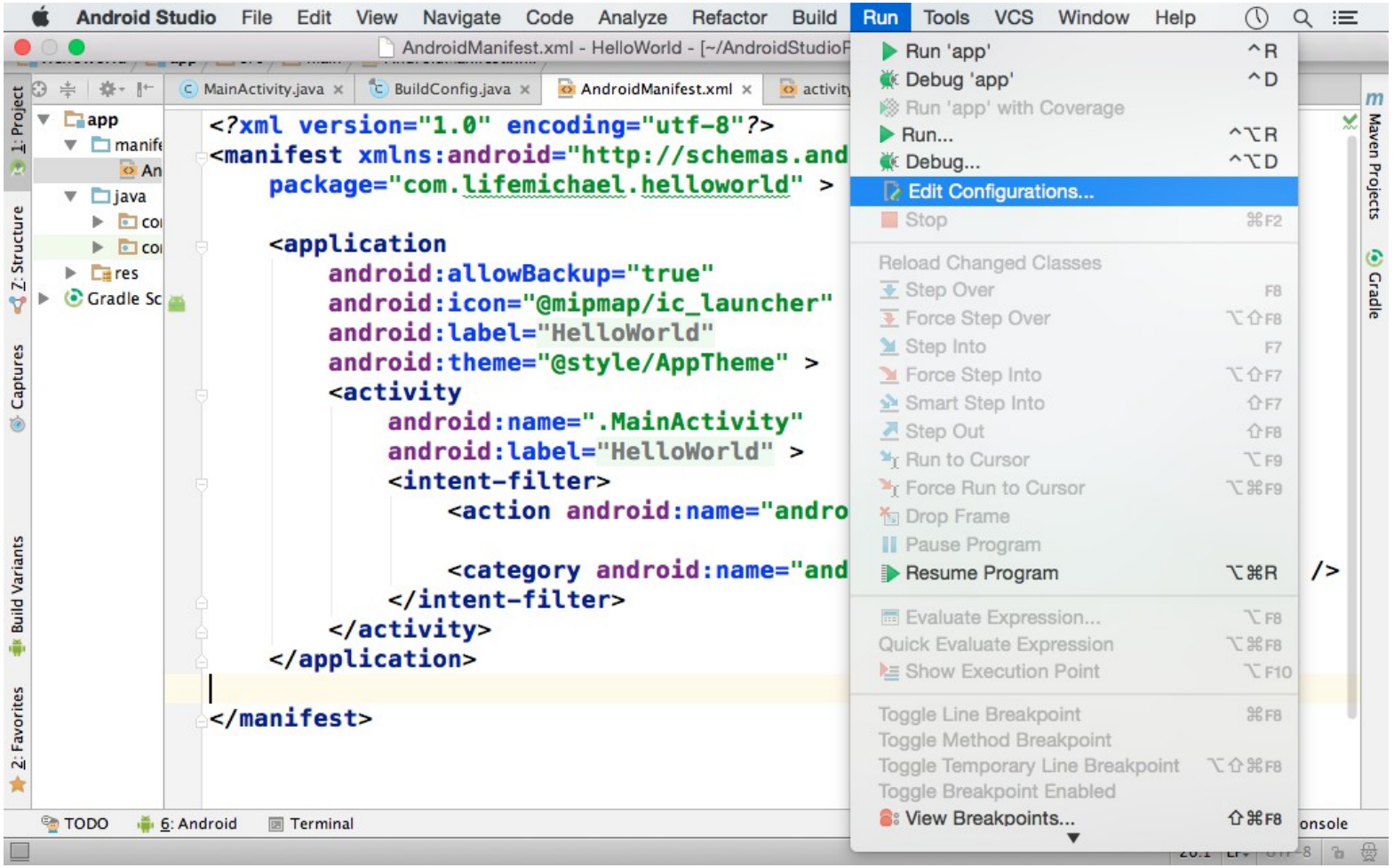
Hello World



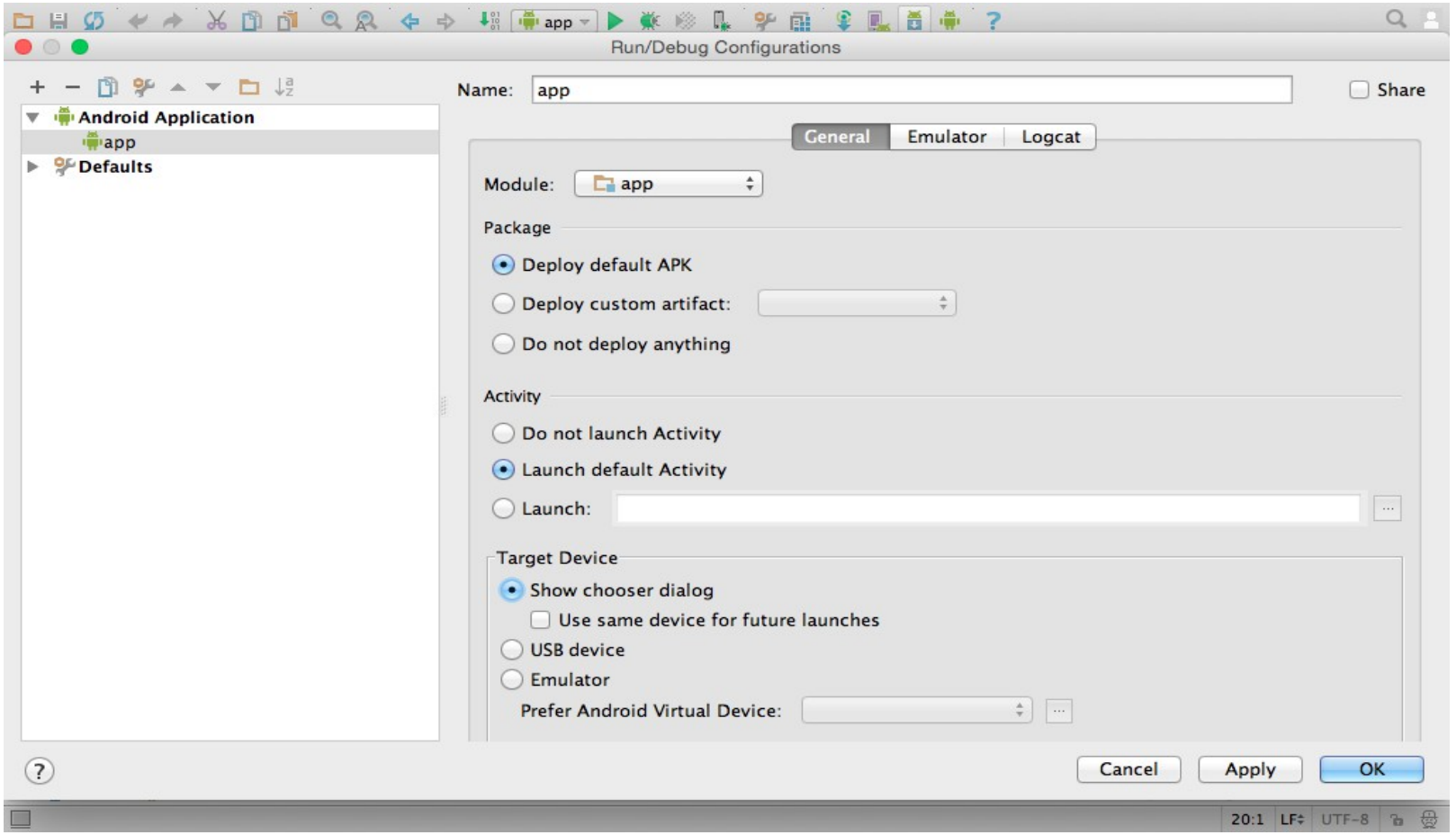
Hello World

- In order to run the application we should first make sure we have a compatible android device. Either a virtual one (AVD) or a device connected with our computer.

Hello World



Hello World



The Android Asset Packaging Tool

- This tool is used to pack all files that compose an android application into one binary file.

The Entry Point Activity

- The android manifest file defines an entry point activity, also known as the 'Top Level Activity', marked using an intent-filter element that looks as the following:

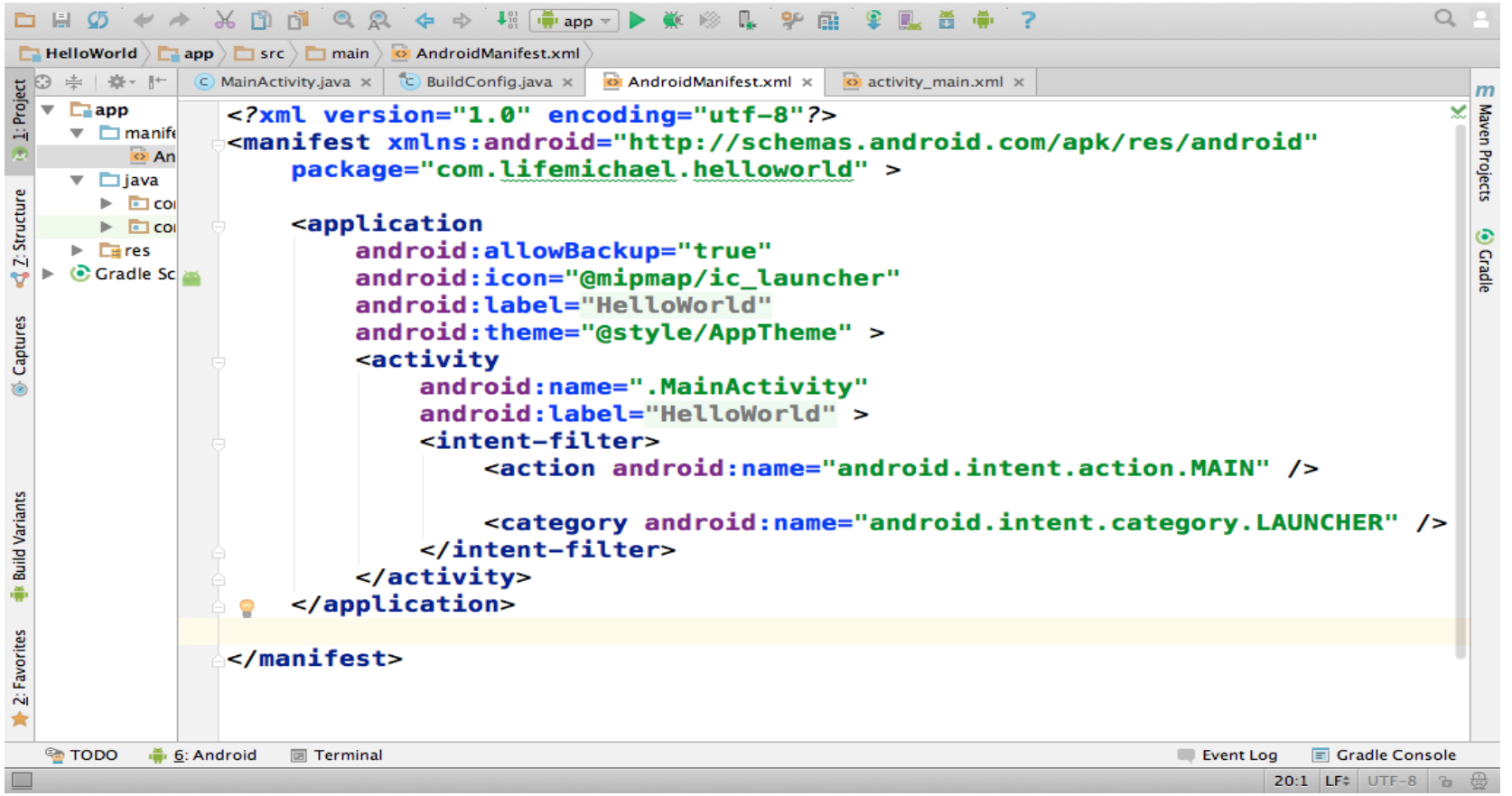
```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

When the user tries to run an application, the mobile telephone reads the application manifest file and starts running the activity \ activities that include the MAIN action with the LAUNCHER category.

The Entry Point Activity

- Once the environment identifies the entry point it starts running the activity by calling the `onCreate()` method on that activity.

The Entry Point Activity



The Intent

- Activities are usually started with an intent. Each activity is defined using the `<activity>` element, that may include the `<intent-filter>` element defining the intent that starts it.

Calling Other Activities

- When a given activity starts it can call another one. Different methods allow calling another activity.

```
startActivity(Intent intent)
startActivityForResult(Intent intent, int requestCode)
startActivityFromChild(Activity child, Intent intent,
    int requestCode)
startActivity(Intent)
startActivityForResult(Intent, int)
startActivityIfNeeded(Intent intent, int requestCode)
startNextMatchingActivity(Intent intent)
```

The Activities Stack

- The activities have a life cycle and all activities are maintained on an activity stack. The running activity is on top. When a running activity starts another one then the running activity moves down the stack and a new activity moves to the top.

Paused & Stopped Activities

- Activities lower in the stack can be called 'paused' or 'stopped' activities. A paused activity is partially (or fully) visible. A stopped activity is not visible.
- The execution environment is allowed to kill paused and stopped activities if their resources are needed by other activities.

The SQLite Database

- The android platform includes the SQLite database. The SQLiteDatabase class describes this database.

System Management

- The life cycle of an android application is managed by the system, based on the user's needs and the available resources.

  The system own judgment might cause strange situations (e.g. the user might ask to start running the web browser and the system might prevent that). Current running activities will get the highest priority. Paused or stopped activities (on the other hand) are getting a lower priority and the system might shut them down if it needs their resources.

Separated Processes

- Each application on the android platform runs in a separated process. Each process is running with its own virtual machine.
- The isolation of each application into a separated process allows the system to allocate different priorities to each one of them.

  This way, the execution of high priority applications as getting an incoming phone call is ensured.

Component & Integration Architecture

- The user can move from one application to another. The system is responsible for saving the state of each application allowing the user to return back to it at a later stage.
- The system saves meta data on every activity before starting another.
- When memory becomes an issue the system is allowed to independently shut down a running activity and resume it when necessary.

The `System` Class

- This class in one of the Java classes available on the android platform.
- Using the `getProperties()` static function that was defined in this class we can get detailed information about the android device on which our application is running.

The `android.os.Build` Class

- This class includes lots of public static fields we can refer in order to get detailed information about the build on which our program is executed.
- The information we get using this class is the same information we can get through the system properties.

The `getSystemService` Method

- This method was defined in Context. When we define a new class that extends Activity we can invoke this method in order to get a facade implementation that provides us with a simple access to various remote services of the platform.

```
...
TelephonyManager mngr = (TelephonyManager)context.
    getSystemService(Context.TELEPHONY_SERVICE);
String number = mngr.getLine1Number();
...
```

The `PackageManager` Class

- When calling the `getPackageManager()` function that was defined in `Context` we will get a reference for a `PackageManager` object.
- The `PackageManager` object can provide us with detailed information about the applications installed on our device.

The `R` Class

- This is an auto generated class that includes inner static classes for holding static variables assigned with the ids of each and every resource our application includes.

The `BuildConfig` Class

- This is an auto generated class that includes the definition for the final static variable `DEBUG`.

The `ApplicationInfo` Class

- Calling the `getApplicationInfo()` method that was defined in `Context` we can get an `ApplicationInfo` object through which we can get detailed information about the application.

The `Display` Class

- Calling the `getWindowManager()` method that was defined in `Activity` we will get a `WindowManager` object on which we can invoke the `getDefaultDisplay()` method that returns a reference for a `Display` object that represents the display our device uses.