CMSC436: Programming
Handheld Systems
Android Development Environment
The Android Platform

A multi-layered software stack for building and running mobile applications
The Android Development Environment

Your workbench for writing Android applications

See:

https://developer.android.com/studio/intro/
Today’s Topics

- Downloading Android SDK
- Using the Android Studio IDE
- Using the Android emulator
- Debugging Android applications
- Other tools
Prerequisites

Supported Operating Systems:

- Microsoft Windows 7/8/10 (32- or 64-bit)
- Mac OS X 10.10 (Yosemite) up to 10.14 (Mojave)
- GNOME or KDE desktop (tested on Ubuntu 14.04 LTS, Trusty Tahr)
- Chrome OS
General Prerequisites

4 GB RAM min, 8 GB RAM rec

2-4 GB+ for Android SDK, emulator system images, and caches

1280 x 800 min screen resolution
Getting Started

Download & install Android Studio

See: https://developer.android.com/studio/
Android Studio

Android platform
Android Studio IDE
Key development tools
System image for emulator
Hello World!
package course.examples.helloworld

import android.app.Activity
import android.os.Bundle

class MainActivity : Activity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
The Android Emulator

Runs virtual devices
package course.examples.helloworld;
The Android Emulator

Pros

Doesn’t require an actual phone
Hardware is reconfigurable
Changes are non-destructive
The Android Emulator

Cons

- Slower than an actual device
- Some features unavailable
  - e.g., no support for Bluetooth, USB connections, NFC, etc.
- Performance / user experience can be misleading
Advanced Features

Can emulate many different device/user characteristics, such as:

- Network speed/latencies
- Battery power
- Location coordinates
Advanced Features

Change network speeds
Advanced Features

Emulate incoming phone calls & SMS messages
Nougat is sweet!
Nougat is sweet!
The Android Emulator

Can interconnect multiple emulators
Advanced Features

Many more options

See:

Debugger

Tool for examining the internal state of a running application
The answer to life, the universe and everything is:

We may never know
class TheAnswer : Activity() {
    companion object {
        private val answers = intArrayOf(42, -10, 0, 100, 1000)
        private const val answer = 42
        private const val TAG = "TheAnswer"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        // Required call through to Activity.onCreate()
        // Restore any saved instance state
        super.onCreate(savedInstanceState)

        // Set up the application's user interface (content view)
        setContentView(R.layout.answer_layout)
        val value = findAnswer()
    }
}
val output = if (value != null) answer.toString() else getString(R.string.never_know_string)

// Get a reference to a TextView in the content view
val answerView = findViewById<TextView>(R.id.answer_view)
// Set desired text in answerView TextView
answerView.text = output

private fun findAnswer(): Int? {
    Log.d(TAG, "Entering findAnswer()")
    // Incorrect behavior
    return answers.firstOrNull { it == -answer }
    // Correct behavior
    // return answers.firstOrNull { it == answer }
}
The answer to life, the universe and everything is:

42
Development Tools

Android Studio provides numerous tools for monitoring application behaviors.
Example Tools

Device File Explorer
Logcat
CPU Profiler
Layout Inspector
Device File Explorer

View, copy, and delete files on your device

Often used to examine and verify file creation and transfer
private int findAnswer() {
    for (int val : answers) {
        if (val == answer)
            return val;
    }
    return -1;
}
import ... ;

class TheAnswer extends Activity {

    private static final int[] answers = {42, -10, 42};

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        // Required call through to Activity.onCreate
        // Restore any saved instance state
        super.onCreate(savedInstanceState);

        // Set up the application's user interface
        setContentView(R.layout.answer_layout);

        // Get a reference to a TextView in the content view
        TextView answerView = findViewById(R.id.answer_view);

        int val = findAnswer();
        String output = (val == answer) ? String.valueOf(val) : String.valueOf(-1);

        // Set desired text in answerView TextView
        answerView.setText(output);

    }

    private int findAnswer() {

        for (int val : answers) {
            if (val == answer) return val;
        }

        return -1;
    }
}
Logcat

Write and review log messages
Apps use Log class to write messages to log
Developer can search and filter log messages
setContentView(R.layout.answer_layout);

// Get a reference to a TextView in the content view
TextView answerView = findViewById(R.id.answer_view);

int val = findAnswer();
String output = (val == answer) ? String.valueOf(answer) : "We may never know";

// Set desired text in answerView
answerView.setText(output);

private int findAnswer() {
    Log.d(TAG, msg:"Entering findAnswer()");
    for (int val : answers) {
        if (val == answer)
            return val;
    }
    Log.e(TAG, msg:"Unexpected behavior");
    return -1;
}
setContentView(R.layout.answer_layout);

// Get a reference to a TextView in the content view
TextView answerView = findViewById(R.id.answer_view);

int val = findAnswer();
String output = (val == answer) ? String.valueOf(answer) : "We may never know";

// Set desired text in answerView TextView
answerView.setText(output);

private int findAnswer() {
    Log.d(TAG, msg: "Entering findAnswer()"");
    for (int val : answers) {
        if (val == answer)
            return val;
    }
    Log.e(TAG, msg: "Unexpected behavior");
    return -1;
}
CPU Profiler

Logs execution sequences and timing taken from a running application

Graphically displays method traces and metrics
package course.examples.thanitize;

import ...

public class TheAnswer extends Activity {

    private static final int[] answers = { 42, -10, 0, 100, 1000 };  
    private static final int answer = 42;  
    private static String TAG = "TheAnswer";

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        // Required call through to Activity.onCreate()
        super.onCreate(savedInstanceState);

        // Set up the application's user interface (content view)
        setContentView(R.layout.answer_layout);

        // Get a reference to a TextView in the content view
        TextView answerView = findViewById(R.id.answer_view);

        int val = findAnswer();
        String output = (val == answer) ? String.valueOf(answer) : getString(R.string.never_kno

        // Set desired text in answerView TextView
        answerView
            .setText(output);
    }

    private int findAnswer() {
        Log.d(TAG, msg: "Entering findAnswer()";  
        for (int val : answers) {

        }
    }
}
Layout Inspector

Shows the runtime organization of the user interface
package course.examples.helloworld

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

public class MainActivity
        extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
The answer to life, the universe and everything is:

We may never know
Next

Application Fundamentals
Example Applications

HelloWorld

TheAnswer