Recording in Progress

This class is being recorded

Please turn off your video and/or video if you do not wish to be recorded
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 gLinux)
- 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

import ...

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

Can interconnect multiple emulators
Advanced Features

Many more options

See:

Debugger

Tool for examining the internal state of a running application
TheAnswer

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
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)

    // Get a reference to a TextView in the content view
    val answerView = findViewById<TextView>(R.id.answer_view)
    val value = findAnswer()
    val output = if (value == answer) answerView.text = answer
                 else "We may never know"

    // Set desired text in answerView Textview
    answerView.text = output

}

private fun findAnswer(): Int {
    Log.d(TAG, msg = "Entering findAnswer()"

    // Incorrect behavior
    return answers.firstOrNull { it == answer }

    // Correct behavior
    return answers.firstOrNull { it == answer }
}
package course.examples.theanswer

import ...

class TheAnswer : Activity() {
    companion object {
        private val answers = intArrayOf(42, -10, 0, 100, 1888)
        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)

        setContentView(R.layout.answer_layout)

        // Get a reference to a TextView in the content view
        val answerView = findViewById<TextView>(R.id.answer_view)
        val value = findViewById()
        if (value == answer) answer.value.toString() else "We may never know"

        // Set desired text in answerView TextView
        answerView.text = output
    }

    private fun findAnswer(): Int? {
        Log.d(TAG, msg) "Entering findAnswer()"
        // Incorrect behavior
        return answers.firstOrNull { it == answer }
        // Correct behavior
        return answers.firstOrNull { it == answer }
    }
}
Logcat

Write and review log messages
Apps use Log class to write messages to log
Developer can search and filter log messages
// Set up the application's user interface (content view)
setContentView(R.layout.answer_layout)

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

// Get user input
val input = answerView.text.toString()

// Set desired text in answerView TextView
answerView.text = output

private fun findAnswer(): Int? {
    Log.d(TAG, msg "Entering findAnswer()")
    // Incorrect behavior
    return answers.firstOrNull { it == answer }
    // Correct behavior
    return answers.firstOrNull { it == answer }
}
CPU Profiler

Logs execution sequences and timing taken from a running application

Graphically displays method traces and metrics
package theanswer

import java.util.List

class TheAnswer {

    private static final List<String> answers = Arrays.asList("1", "2", "3", "4");

    public String findAnswer(String question) {
        for (String answer : answers) {
            if (question.equals(answer)) {
                return answer;
            }
        }
        return null;
    }

    public static void main(String[] args) {
        TheAnswer theAnswer = new TheAnswer();
        String question = "What is the meaning of life?";
        String answer = theAnswer.findAnswer(question);
        System.out.println("The answer is: " + answer);
    }
}
package course.examples.theanswer

import ...

class TheAnswer : Activity()
  companion object {
    private val val answers = intArrayOf(42, -10, 0, 100, 1888)
    private const val answer = 42
    private const val TAG = "TheAnswer"
  }

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_theanswer)
    val answerView = findViewById<TextView>(R.id.answer)
    val value = val answers.random()
    if (value == answer) {
      answerView.text = "Correct!"
      return
    }
    // Set design answerView.text = "Incorrect"
  }

private fun findLog(dTAG, message: String) {
  Log.d(dTAG, message)
}

// Incorrect
// Correct
package theanswer

import java.util.List

public class TheAnswer {
  public static void main(String[] args) {
    String answer = "42, -16, 0, 100, 1888, ";
    int x = Integer.parseInt(answer);
    int y = x / 2;
    System.out.println("The answer is: "+y);
  }
}

class TheAnswer {
  public static void main(String[] args) {
    String answer = "42, -16, 0, 100, 1888, ";
    int x = Integer.parseInt(answer);
    int y = x / 2;
    System.out.println("The answer is: "+y);
  }
}

public class TheAnswer {
  public static void main(String[] args) {
    String answer = "42, -16, 0, 100, 1888, ";
    int x = Integer.parseInt(answer);
    int y = x / 2;
    System.out.println("The answer is: "+y);
  }
}

public class TheAnswer {
  public static void main(String[] args) {
    String answer = "42, -16, 0, 100, 1888, ";
    int x = Integer.parseInt(answer);
    int y = x / 2;
    System.out.println("The answer is: "+y);
  }
}

public class TheAnswer {
  public static void main(String[] args) {
    String answer = "42, -16, 0, 100, 1888, ";
    int x = Integer.parseInt(answer);
    int y = x / 2;
    System.out.println("The answer is: "+y);
  }
}

public class TheAnswer {
  public static void main(String[] args) {
    String answer = "42, -16, 0, 100, 1888, ";
    int x = Integer.parseInt(answer);
    int y = x / 2;
    System.out.println("The answer is: "+y);
  }
}
Layout Inspector

Shows the runtime organization of the user interface
package com.example.thearnswer

import ...

class TheAnswer {

    companion object {
        private val answers = listOf(42, -10, 0, 100, 1800)
    }

    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)

        // Get a reference to a TextView in the content view
        val answerView = findViewById<TextView>(R.id.answer_view)
        val value = findAnswer()
        val output = if (value == answer) answer.toString() else "We may never know"
        // Set desired text in answerView TextView
        answerView.text = output
    }

    private fun findAnswer(): Int? {
        Log.d(TAG, msg "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
Next

Application Fundamentals
Example Applications

HelloWorld

TheAnswer