

Unit Testing in Android with GitHub Action

Teaching Assistants

Boston University

Apr. 15th, 2025

**Android
Developers**



Outline

Previously: Google Test for C++

Today: Unit Testing in Android

Unit Testing Constraints

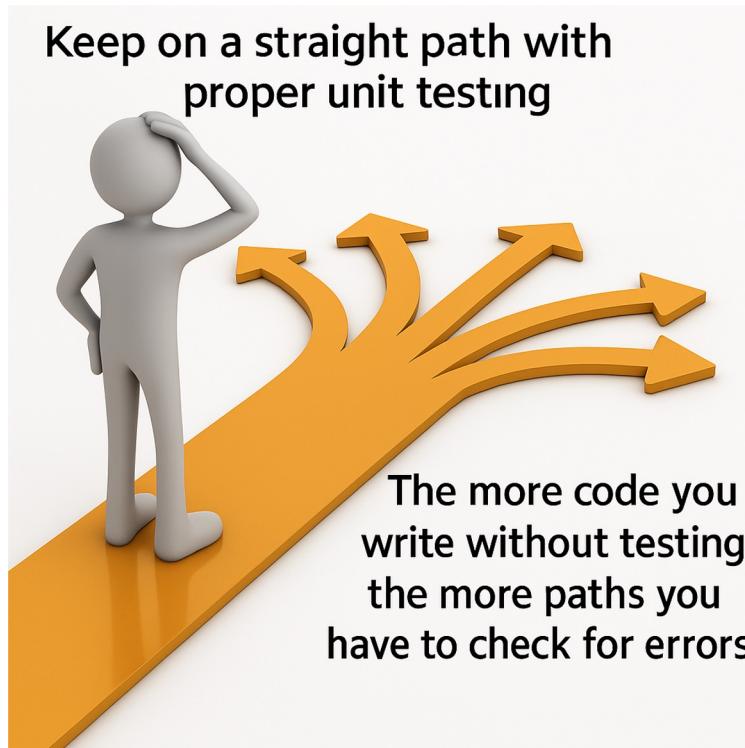
Test Suits with GitHub Actions

Code Merging after Passing Unit Testing

[Weather Demo](#) Unit Testing

Takeaways

Part 1 — Previously: Google Test (gtest) for C++



Ensuring that 'the thing' does what it is supposed to do

Credits: [Introduction to Google Test: An Open Source C/C++ Unit-Testing Framework](#)

Part 1 — Previously: Google Test (gtest) for C++

An Open Source C/C++ Unit-Testing Framework

Ensuring that 'the thing' does what it is supposed to do

Tests should be independent and repeatable.

Tests should be well organized and reflect the structure of the tested code.

Tests should be portable and reusable.

Some example codes: [Android-Unit-Testing/tree/main/GoogleTest/tests](#)

Part 1 — Previously: Google Test (gtest) for C++

The screenshot shows an IDE interface with a C++ file named `hw3_problem3_test.cpp`. The code implements unit tests using Google Test. The code includes two test cases: `InsertInOrder` and `InsertInOrder`. The first test case, `InsertInOrder, EmptyListLinkedList`, initializes a `Node*` `head` to `NULL`, inserts a value of `10` into it, and then checks if the head's value is `10` and its next pointer is `NULL`. The second test case, `InsertInOrder, SingleValueInsertAfter`, creates a single-node linked list with value `5`, inserts a value of `10` after it, and then checks if the head's value is `5`, its next node's value is `10`, and its next pointer's next pointer is `NULL`. Both test cases use `EXPECT_EQ` to make assertions.

```
#include <gtest/gtest.h>
#include "../hw3_problem3.h"

TEST(InsertInOrder, EmptyListLinkedList) {
    Node* head = NULL;
    head = insertInOrder(head, 10);

    EXPECT_EQ(10, head->value);
    EXPECT_EQ(NULL, head->next);
}

TEST(InsertInOrder, SingleValueInsertAfter) {
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->value = 5;
    head->next = NULL;

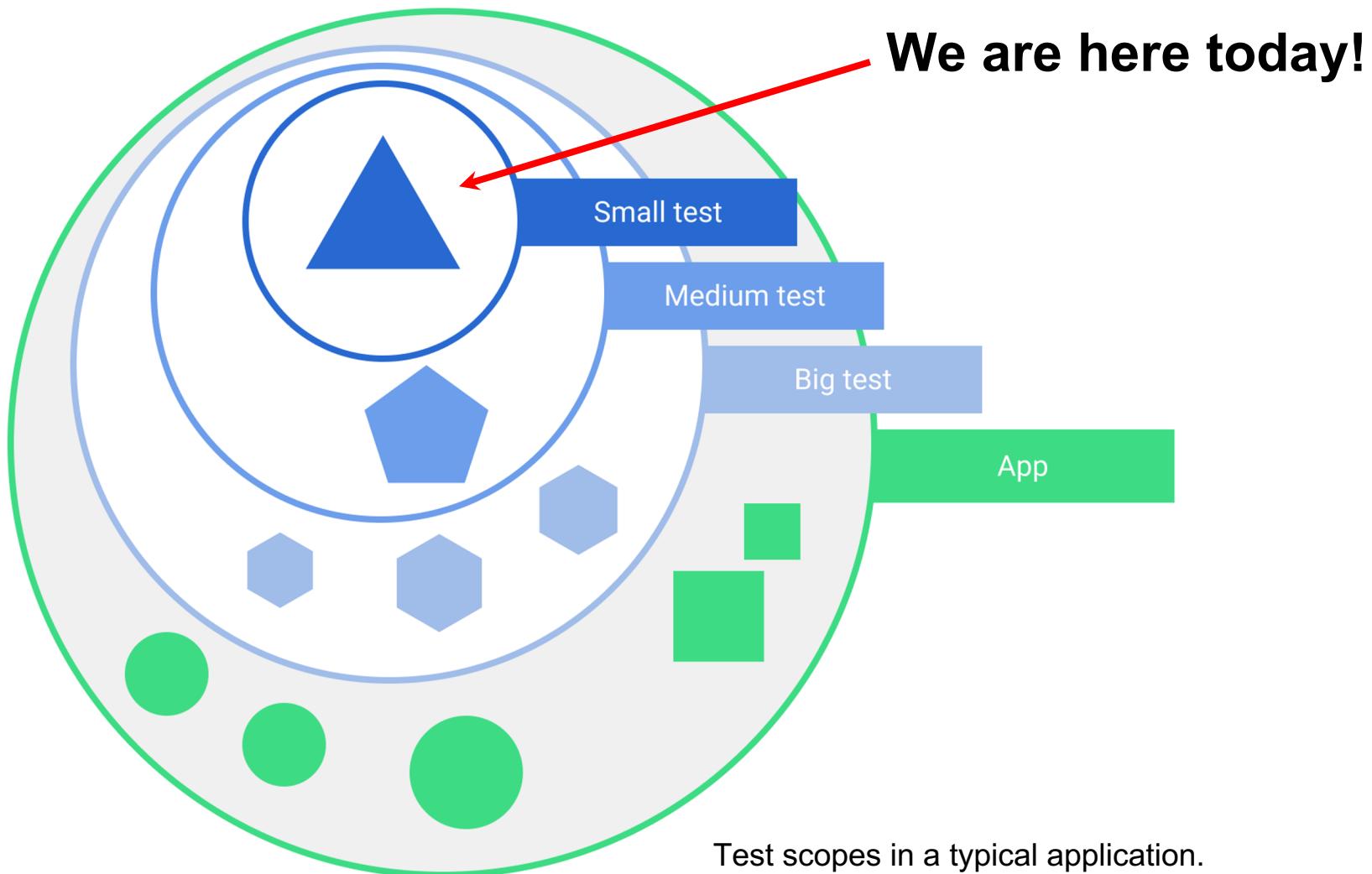
    head = insertInOrder(head, 10);

    EXPECT_EQ(5, head->value);
    EXPECT_EQ(10, head->next->value);
    EXPECT_EQ(NULL, head->next->next);

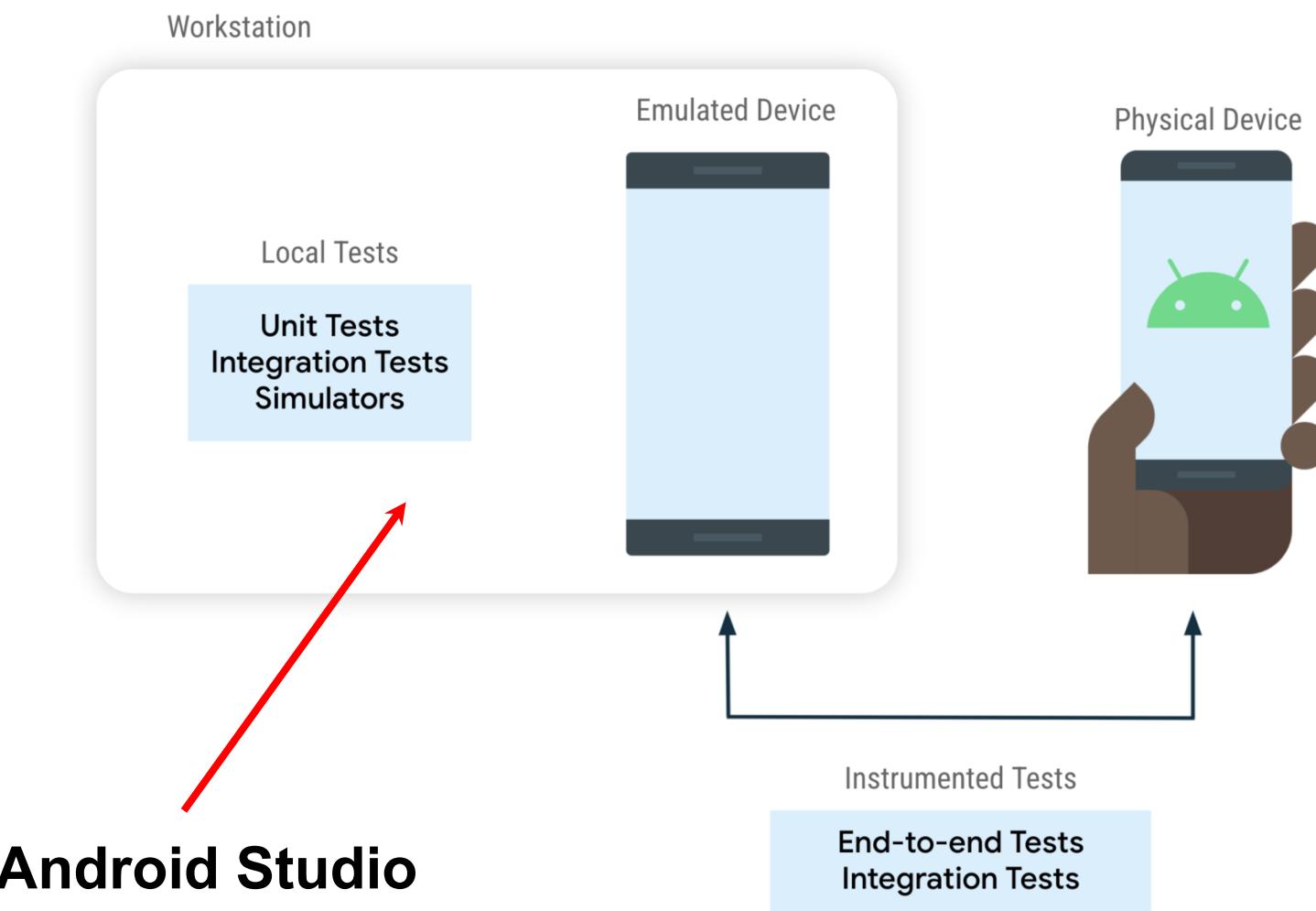
    free(head);
}
```

Credits: <https://google.github.io/googletest/primer.html>

Part 2 — Today: Unit Testing in Android



Part 2 — Today: Unit Testing in Android



Part 2 — Today: Unit Testing in Android



A type of **software testing** where individual

units/components of a software are tested

Done during the **development of an application**

The **objective** of Unit Testing is to verify its correctness

Usually performed by the developer

Tools that we use in Android: **JUnit**

Part 2 — Today: Unit Testing in Android



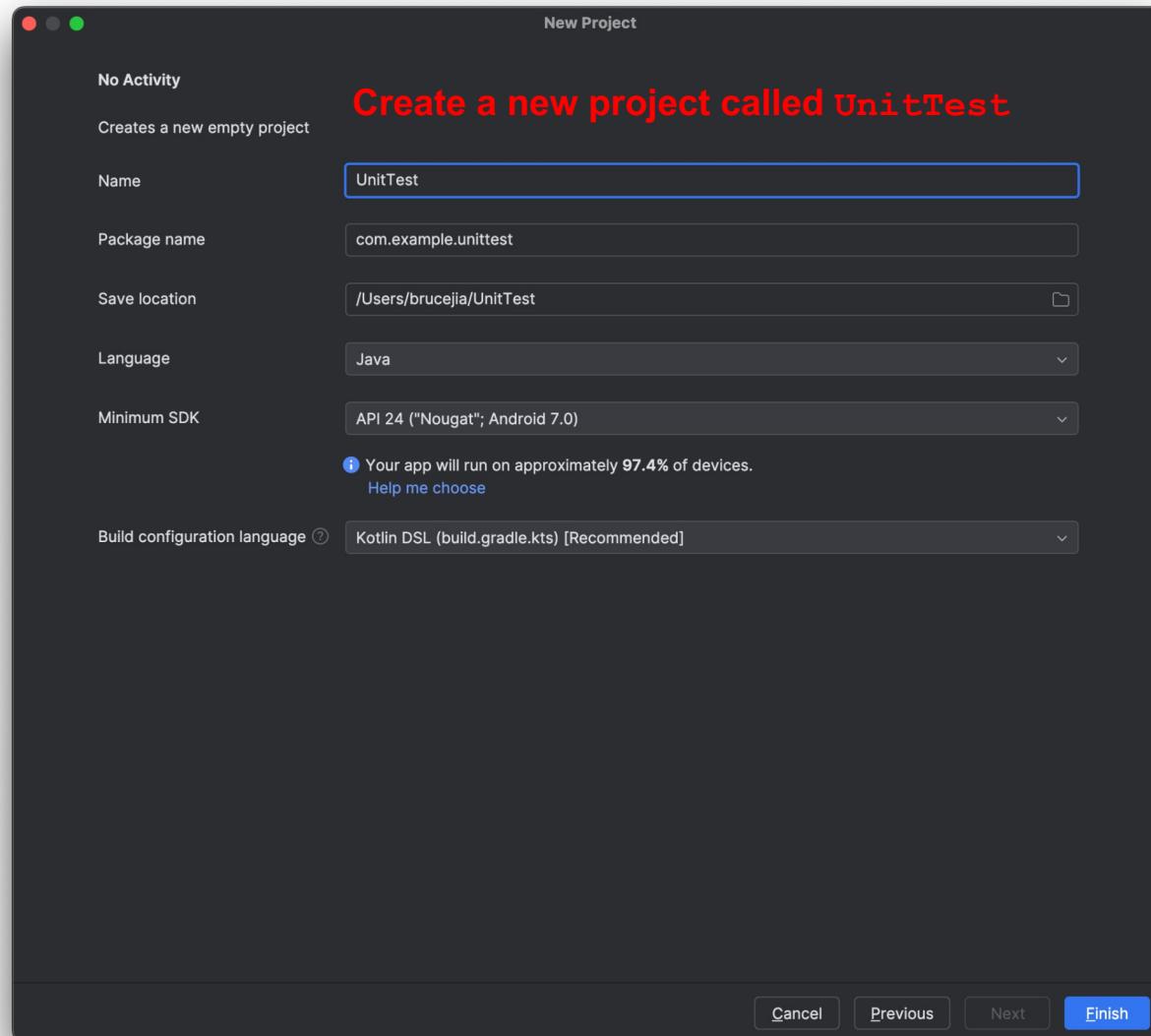
Tools that we use in Android: **JUnit4**

A programmer-oriented testing framework for Java.

By default, we will use Gradle Build Tool

**Gradle Build Tool accelerates
developer productivity**

Part 2 — Today: Unit Testing in Android



Part 2 — Today: Unit Testing in Android

The screenshot shows the Android Studio interface with the project 'UnitTest' open. The left sidebar displays the project structure under 'Android'. The 'app' module is selected, showing its contents: 'manifests', 'java' (containing 'com.example.unittest' which has 'Calculator', 'com.example.unittest (androidTest)', 'com.example.unittest (test)', 'java (generated)', 'res', and 'res (generated)'); 'Gradle Scripts' (with 'build.gradle.kts (Project: UnitTest)', 'build.gradle.kts (Module :app)', 'proguard-rules.pro', 'gradle.properties', 'gradle-wrapper.properties', 'libs.versions.toml', 'local.properties', and 'settings.gradle.kts'); and 'Version control' (with 'Medium Phone API 36'). The top bar shows 'Unit Test' and 'Version control'. A message in the center says 'You can use the Project Structure dialog to view and edit your project configuration'. On the right, there are 'Open (⌘O)' and 'Hide notification' buttons.

The main code editor window shows the 'build.gradle.kts (Module :app)' file:

```
plugins {
    alias(libs.plugins.android.application)
}

android {
    namespace = "com.example.unittest"
    compileSdk = 35

    defaultConfig {
        applicationId = "com.example.unittest"
        minSdk = 24
        targetSdk = 35
        versionCode = 1
        versionName = "1.0"

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            isMinifyEnabled = false
            proguardFiles(
                getDefaultProguardFile("name: proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_11
        targetCompatibility = JavaVersion.VERSION_11
    }
}

dependencies {
    implementation(libs.appcompat)
    implementation(libs.material)
    testImplementation(libs.junit)
    androidTestImplementation(libs.ext.junit)
    androidTestImplementation(libs.espresso.core)
}
```

A red arrow points from the text 'Under the build.gradle.kts (Module :app)' to the line 'testImplementation(libs.junit)' in the code, which is highlighted with a red box.

Under the build.gradle.kts (Module :app)

Part 2 — Today: Unit Testing in Android

The screenshot shows the Android Studio interface. On the left, the Project Navigational Bar displays the project structure under the 'Android' tab. A red box highlights the 'Calculator' class file located in the 'src/androidTest/java/com/example/unittest' directory. A red arrow points from this highlighted file to the code editor on the right. The code editor shows the 'Calculator.java' source code:

```
1 package com.example.unittest;
2
3 public class Calculator {
4     @
5         public int evaluate(String expression) {
6             // Basic implementation for demo purposes.
7             String[] tokens = expression.split( regex: "\\"+");
8             int sum = 0;
9             for (String token : tokens) {
10                 sum += Integer.parseInt(token);
11             }
12             return sum;
13         }
14 }
```

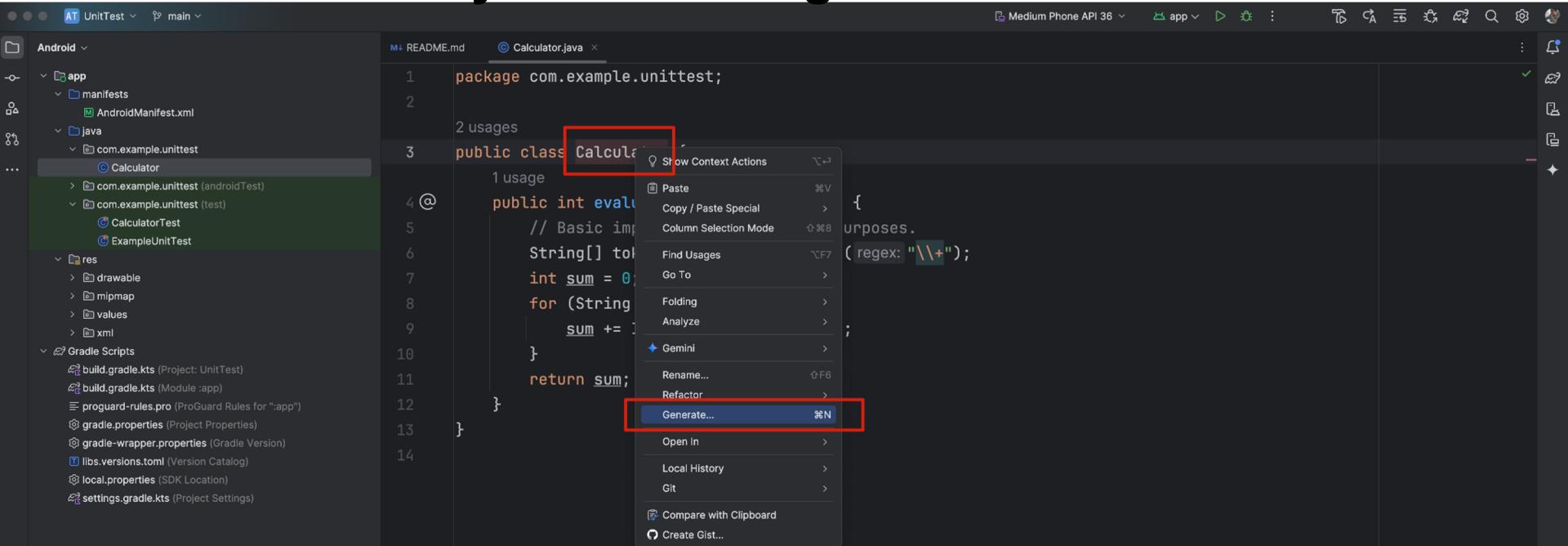
Create a JAVA class Calculator.java under the src/androidTest/java/com/example/unittest folder

Part 2 — Today: Unit Testing in Android

```
package com.example.unittest;

public class Calculator {
    public int evaluate(String expression) {
        // Basic implementation for demo purposes.
        String[] tokens = expression.split("\\\\+");
        int sum = 0;
        for (String token : tokens) {
            sum += Integer.parseInt(token);
        }
        return sum;
    }
}
```

Part 2 — Today: Unit Testing in Android



right click on the Class name and choose
"Generate -> Test" and the boilerplate of the test
will be generated in the correct place

Part 2 — Today: Unit Testing in Android

The screenshot shows the Android Studio interface with the following details:

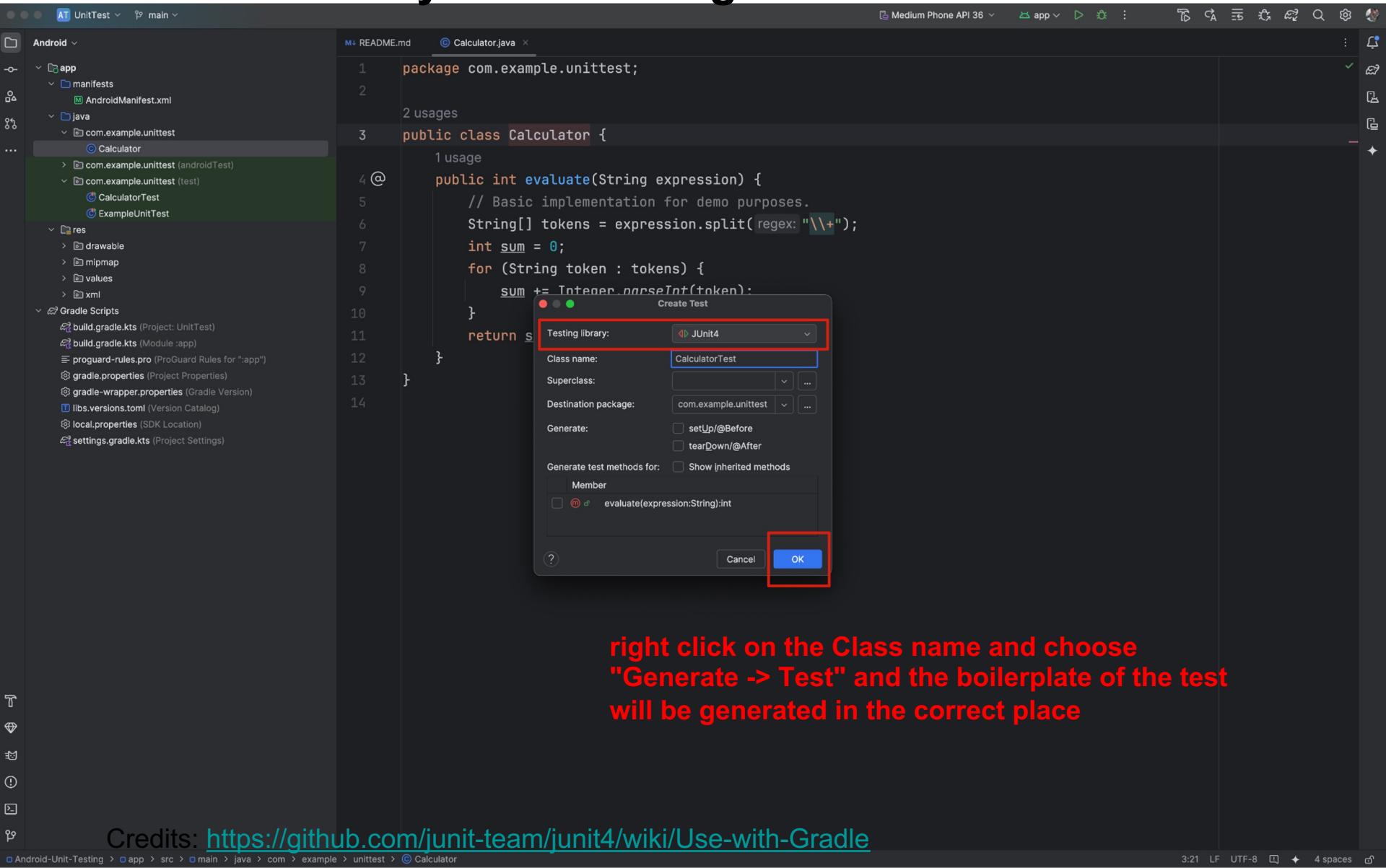
- Project Structure:** The left sidebar shows the project structure under "Android". It includes the "app" module with "manifests", "java" (containing "Calculator.java"), and "res" (containing "drawable", "mipmap", "values", and "xml"). Below "app" are "Gradle Scripts" with files like "build.gradle.kts" and "gradle-wrapper.properties".
- Code Editor:** The main editor window displays "Calculator.java". The code is as follows:

```
1 package com.example.unittest;
2
3 public class Calculator {
4     @
5         public int ev
6             // Basic
7             String[]
8             int sum =
9                 for (String token : s
10                sum += Integer.parseInt(token);
11
12            return sum;
13        }
14    }
```

- Context Menu:** A context menu is open over the word "Calculator" at line 3. The menu has a title "Generate" and contains several options: "Unit Test Scenarios", "Constructor", "toString()", "Override Methods...", "Test...", "Copy", and "Copyright". The "Test..." option is highlighted with a red box.
- Annotations:** Red text annotations are overlaid on the right side of the image:
 - "right click on the Class name and choose
"Generate -> Test" and the boilerplate of the test
will be generated in the correct place"
- Bottom Status Bar:** The status bar at the bottom shows the path "Android-Unit-Testing > app > src > main > java > com > example > unittest > Calculator" and the system status "3:18 LF UTF-8 4 spaces".

Credits: <https://github.com/junit-team/junit4/wiki/Use-with-Gradle>

Part 2 — Today: Unit Testing in Android



Part 2 — Today: Unit Testing in Android

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The left sidebar shows the project structure under "Android". It includes the "app" module, which contains "manifests", "java" (with "Calculator" and "com.example.unittest" packages), "res", and "Gradle Scripts". The "com.example.unittest" package is expanded, showing "CalculatorTest" and "ExampleUnitTest". A red box highlights the "CalculatorTest.java" file.
- Code Editor:** The main editor area displays the content of "CalculatorTest.java".

```
1 package com.example.unittest;
2
3 import static org.junit.Assert.assertEquals;
4 import org.junit.Test;
5
6 public class CalculatorTest {
7     @Test
8     public void evaluatesExpression() {
9         Calculator calculator = new Calculator();
10        int sum = calculator.evaluate(expression: "1+2+3");
11        assertEquals(expected: 6, sum);
12    }
13 }
14 }
```
- Annotations:** A red box highlights the "CalculatorTest.java" tab in the top bar. A large red box highlights the "CalculatorTest" package in the project tree.
- Text Overlay:** The text "The testing codes will be automatically generated!" is displayed in red at the bottom right of the code editor area.
- Bottom Bar:** The bottom bar shows the credits: "Credits: <https://github.com/junit-team/junit4/wiki/Use-with-Gradle>". It also includes icons for font size, line height, and other settings.

Part 2 — Today: Unit Testing in Android

Open your terminal and go to the project folder

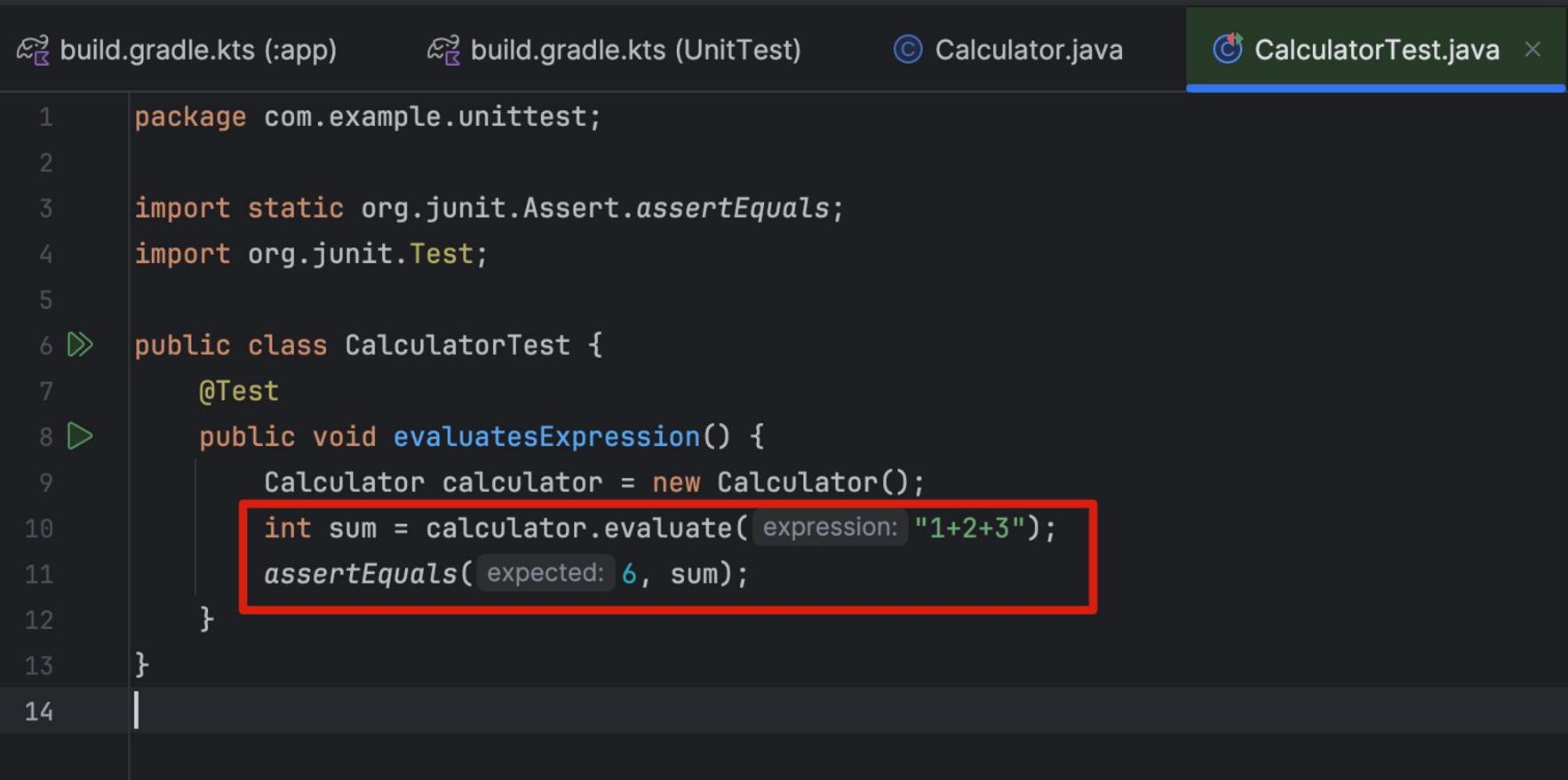
```
$ cd Desktop/UnitTest/
```

Then run the unit testing

```
$ ./gradlew test
```

Check the code if you need it: [Android-Unit-Testing](#)

Part 2 — Today: Unit Testing in Android



The screenshot shows a code editor with four tabs at the top: build.gradle.kts (:app), build.gradle.kts (UnitTest), Calculator.java, and CalculatorTest.java. The CalculatorTest.java tab is active, highlighted with a blue bar. The code in the editor is as follows:

```
1 package com.example.unittest;
2
3 import static org.junit.Assert.assertEquals;
4 import org.junit.Test;
5
6 public class CalculatorTest {
7     @Test
8     public void evaluatesExpression() {
9         Calculator calculator = new Calculator();
10        int sum = calculator.evaluate( expression: "1+2+3");
11        assertEquals( expected: 6, sum);
12    }
13 }
14
```

A red rectangular box highlights the line of code: `int sum = calculator.evaluate(expression: "1+2+3");`. This line is part of a test method that calls the `evaluate` method on a `Calculator` object with the expression `"1+2+3"` and asserts that the result is `6`.

Part 2 – Today: Unit Testing in Android

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The left sidebar shows the project structure under "WeatherDemo". The "app" module is selected, and the "java" directory contains "WeatherAPIServiceTest" and "ExampleUnitTest".
- Code Editor:** The main editor window displays the content of `WeatherAPIServiceTest.java`. A red arrow points from the top right towards the context menu.
- Context Menu:** A context menu is open at the bottom right of the code editor, listing options like "Run Context Actions", "Paste", "Copy / Paste Special", etc. A red box highlights the "Run 'WeatherAPIServiceTest'" option, which is also indicated by a red arrow.
- Toolbar:** The top right of the screen shows the toolbar with icons for search, refresh, and settings.
- Status Bar:** The bottom right corner shows the status bar with "74:1 LF UTF-8" and "4 spaces".

```
public class WeatherAPIServiceTest {
    static final String HEALTHY_JSON =
        "windchill_c": 23.9,
        "windchill_f": 75,
        "heatindex_c": 26.1,
        "heatindex_f": 78.9,
        "dewpoint_c": 21.5,
        "dewpoint_f": 70.7,
        "vis_km": 10,
        "vis_miles": 6,
        "uv": 6.1,
        "gust_mph": 6.7,
        "gust_kph": 10.8
    };

    @Test
    public void parseJson_success(){
        WeatherData result = weatherAPIService.parseJSONResponse("boston", HEALTHY_JSON);
        assertEquals("boston", result.city);
        assertEquals(75, result.degreesFahrenheit, delta: 0.001);
    }

    @Test(expected = IllegalStateException.class)
    public void parseJson_failure(){
        WeatherData result = weatherAPIService.parseJSONResponse("boston", "");
    }

    @Test
    public void makeAPICall_success() throws IOException {
        OkHttpClient mockHttpClient = Mockito.mock(OkHttpClient.class);
        weatherAPIService.httpClient = mockHttpClient;

        WeatherCallback mockCallback = Mockito.mock(WeatherCallback.class);
        Call mockCall = Mockito.mock(Call.class);
        Response mockHTTPResponse = Mockito.mock(Response.class);

        Mockito.when(mockHttpClient.newCall(Mockito.any(Request.class))).thenReturn(mockCall);
        Mockito.when(mockCall.execute()).thenReturn(mockHTTPResponse);

        Mockito.when(mockHTTPResponse.isSuccessful()).thenReturn(true);

        ResponseBody mockResponseBody = Mockito.mock(ResponseBody.class);
        Mockito.when(mockHTTPResponse.body()).thenReturn(mockResponseBody);

        Mockito.when(mockResponseBody.string()).thenReturn(HEALTHY_JSON);
    }
}
```

Part 2 — Today: Unit Testing in Android

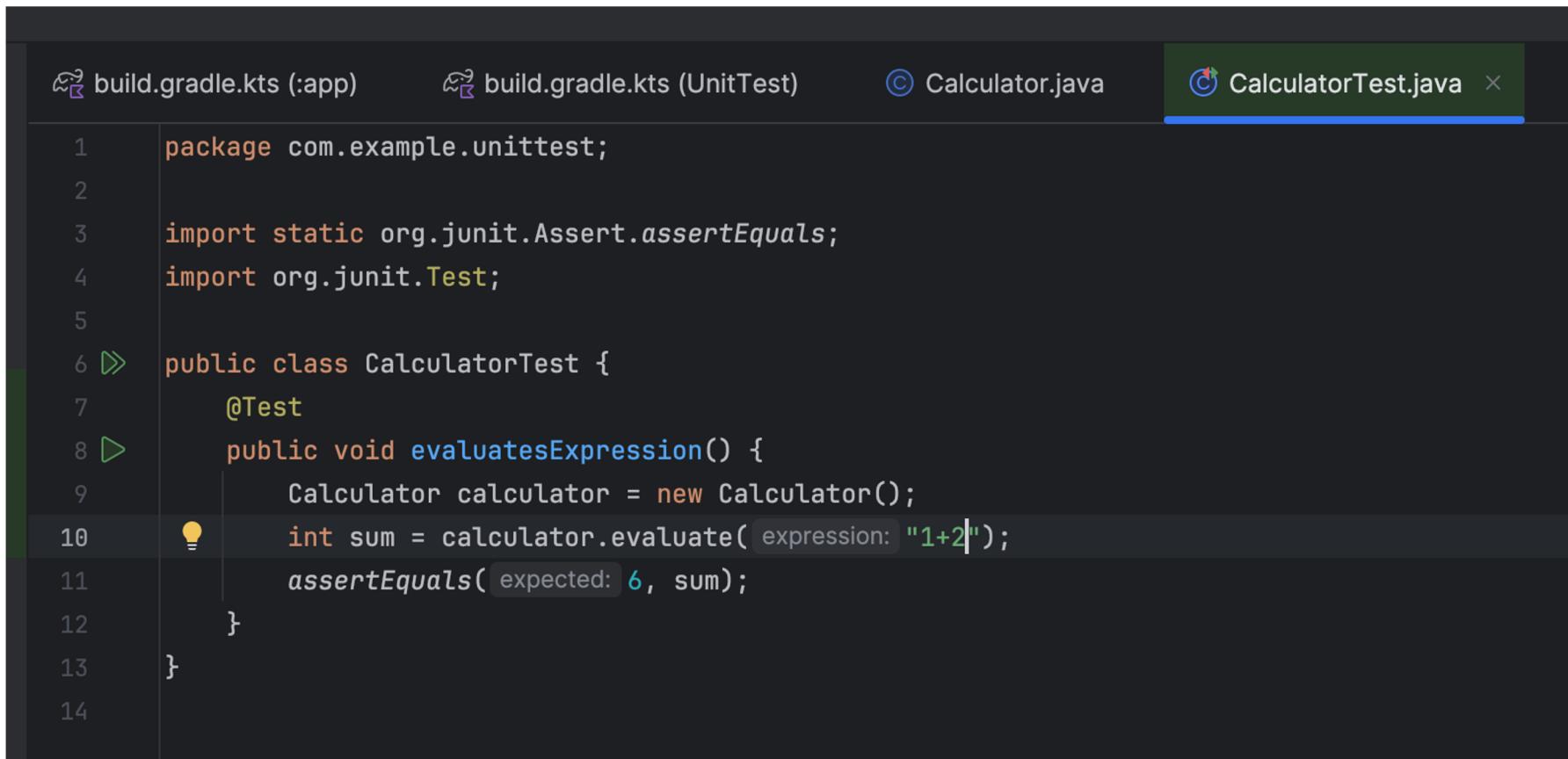
A screenshot of a terminal window titled ".147:~/Desktop/UnitTest". The window has a dark background with a blurred image of a building with a dome in the background. The terminal output shows:

```
base ~/Desktop/UnitTest (1.153s)
./gradlew test

BUILD SUCCESSFUL in 972ms
40 actionable tasks: 2 executed, 38 up-to-date
```

The bottom of the terminal shows the GitHub Actions interface with the status "In Progress".

Part 2 — Today: Unit Testing in Android



The screenshot shows a dark-themed IDE interface with several tabs at the top:

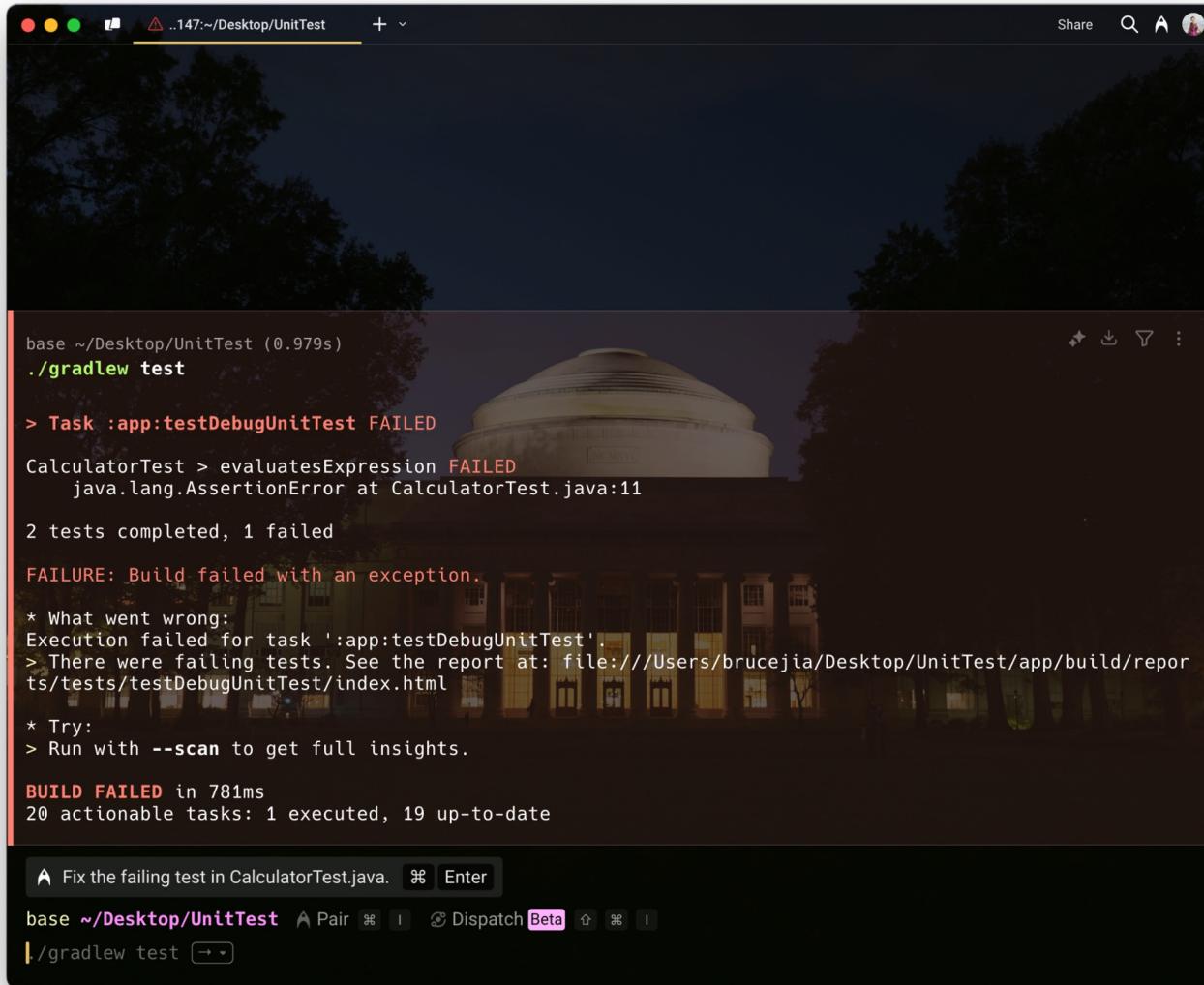
- build.gradle.kts (:app)
- build.gradle.kts (UnitTest)
- Calculator.java
- CalculatorTest.java (selected tab, indicated by a blue border)

The code editor displays the following Java code:

```
1 package com.example.unittest;
2
3 import static org.junit.Assert.assertEquals;
4 import org.junit.Test;
5
6 public class CalculatorTest {
7     @Test
8     public void evaluatesExpression() {
9         Calculator calculator = new Calculator();
10        int sum = calculator.evaluate( expression: "1+2" );
11        assertEquals( expected: 6, sum );
12    }
13 }
14 }
```

The code uses JUnit annotations (@Test) and the Assert.assertEquals method to verify the output of the Calculator class's evaluate method. A yellow lightbulb icon is visible near the 10th line, likely indicating a warning or suggestion.

Part 2 — Today: Unit Testing in Android



A screenshot of a terminal window titled '..147:~/Desktop/UnitTest'. The terminal displays the output of a Gradle test run:

```
base ~/Desktop/UnitTest (0.979s)
./gradlew test

> Task :app:testDebugUnitTest FAILED
CalculatorTest > evaluatesExpression FAILED
    java.lang.AssertionError at CalculatorTest.java:11

2 tests completed, 1 failed

FAILURE: Build failed with an exception.

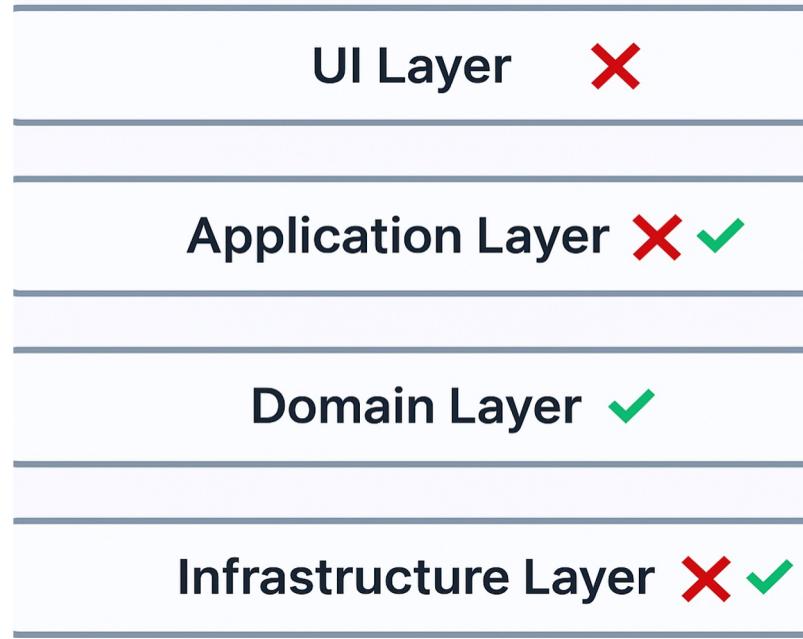
* What went wrong:
Execution failed for task ':app:testDebugUnitTest'.
> There were failing tests. See the report at: file:///Users/brucejia/Desktop/UnitTest/app/build/reports/tests/testDebugUnitTest/index.html

* Try:
> Run with --scan to get full insights.

BUILD FAILED in 781ms
20 actionable tasks: 1 executed, 19 up-to-date
```

The terminal has a dark background with a blurred image of a classical building in the background. A tooltip at the bottom left says 'Fix the failing test in CalculatorTest.java.' with an 'Enter' key icon.

Part 3 — Unit Testing Constraints



Unit Test is not aware of **Nullability**

Don't use **Singleton**

Static Methods has hidden dependencies, so can not be substituted

Part 4 — Test Suits with GitHub Actions (Android CI)

SuperBruceJia / Android-Unit-Testing

Code Issues Pull requests Actions Projects Security Insights Settings

Search Type to search

1

Choose a workflow

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow to get started.

Skip this and [set up a workflow yourself](#) →

Search workflows

Suggested for this repository

- Jekyll using Docker image**
By GitHub Actions
Package a Jekyll site using the `jekyll/buildler` Docker image.
[Configure](#) HTML ●
- C/C++ with Make**
By GitHub Actions
Build and test a C/C++ project using Make.
[Configure](#) C ●
- CMake based, multi-platform projects**
By GitHub Actions
Build and test a CMake based project on multiple platforms.
[Configure](#) C ●
- Python Package using Anaconda**
By GitHub Actions
Create and test a Python package on multiple Python versions using Anaconda for package management.
[Configure](#) Python ●
- Android CI**
By GitHub Actions
Build an Android project with Gradle.
[Configure](#) Java ●
- CMake based, single-platform projects**
By GitHub Actions
Build and test a CMake based project on a single-platform.
[Configure](#) C ●

Deployment

[View all](#)

- Deploy Python app to Azure Functions App** A
- Deploy Java app to Azure Functions App** A
- Deploy a Python app to an Azure Web App** A
- Deploy a Java .jar app to an Azure Web App** A

Part 4 — Test Suits with GitHub Actions (Android CI)

The screenshot shows a GitHub Actions build log for a 'build' job. The job failed 1 minute ago in 1m 16s. The log output is as follows:

```
build
failed 1 minute ago in 1m 16s
Search logs
Build with Gradle
1m 11s

16
17 For more details see https://docs.gradle.org/8.11.1/release-notes.html
18
19 Starting a Gradle Daemon (subsequent builds will be faster)
20 [Incubating] Problems report is available at: file:///home/runner/work/Android-Unit-Testing/Android-Unit-Testing/build/reports/problems/problems-report.html
21
22 FAILURE: Build failed with an exception.
23
24 * Where:
25 Build file '/home/runner/work/Android-Unit-Testing/Android-Unit-Testing/app/build.gradle.kts' line: 1
26
27 * What went wrong:
28 An exception occurred applying plugin request [id: 'com.android.application', version: '8.9.1']
29 > Failed to apply plugin 'com.android.internal.application'.
30   > Android Gradle plugin requires Java 17 to run. You are currently using Java 11.
31   Your current JDK is located in /usr/lib/jvm/temurin-11-jdk-amd64
32   You can try some of the following options:
33     - changing the IDE settings.
34
35     - changing the JAVA_HOME environment variable.
36 Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.
37
38 You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.
39   - changing 'org.gradle.java.home' in 'gradle.properties'.
40
41 For more on this, please refer to https://docs.gradle.org/8.11.1/userguide/command\_line\_interface.html#sec:command\_line\_warnings in the Gradle documentation.
42
43 * Try:
44 > Run with --stacktrace option to get the stack trace.
45 > Run with --info or --debug option to get more log output.
46 > Run with --scan to get full insights.
47 > Get more help at https://help.gradle.org.
48
49 BUILD FAILED in 1m 9s
50 Error: Process completed with exit code 1.
```

A red box highlights the error message starting with 'An exception occurred applying plugin request [id: 'com.android.application', version: '8.9.1']'. Above this box, the text 'We need Java 17 to run the Gradle plugin' is overlaid in red. The bottom of the log shows the build was completed with an exit code of 1.

Summary
Jobs
build
Run details
Usage
Workflow file

We need Java 17 to run the Gradle plugin

Part 4 — Test Suits with GitHub Actions (Android CI)

SuperBruceJia / Android-Unit-Testing

Type to search

Code Issues Pull requests Actions Projects Security Insights Settings

0011c08 · Android-Unit-Testing/.github/workflows/android.yml

View Runs Go to file

SuperBruceJia Update android.yml ✓ 0011c08 · 8 minutes ago History

You must be on a branch to make or propose changes to this file

Code Blame 26 lines (21 loc) · 472 Bytes Code 55% faster with GitHub Copilot

Raw

```
1 name: Android CI
2
3 on:
4   push:
5     branches: [ "main" ]
6   pull_request:
7     branches: [ "main" ]
8
9 jobs:
10   build:
11
12     runs-on: ubuntu-latest
13
14     steps:
15       - uses: actions/checkout@v4
16       - name: set up JDK 17
17         uses: actions/setup-java@v4
18         with:
19           java-version: '17'
20           distribution: 'temurin'
21           cache: gradle
22
23       - name: Grant execute permission for gradlew
24         run: chmod +x gradlew
25       - name: Build with Gradle
26         run: ./gradlew build
```

We need to change Java version to 17 here



GitHub

“Take care of everything for you”

Part 5 — Code Merging after Passing Unit Testing

The screenshot shows the GitHub repository settings page for 'Android-Unit-Testing'. A red box labeled '1' highlights the 'Settings' tab in the top navigation bar. A red box labeled '2' highlights the 'Rules' section in the left sidebar under the 'Code and automation' heading. A red box labeled '3' highlights the 'Code Merging Rule' card in the 'Rulesets' list.

Since there has been a rule set created already that enforces that nothing can be pushed directly to the main branch - everything has to go through a pull request and approval process before being merged.

You can update the existing rule!

Part 5 — Code Merging after Passing Unit Testing

The screenshot shows the GitHub repository settings page for 'Android-Unit-Testing'. The left sidebar is collapsed. The main area is titled 'Rulesets / New branch ruleset'. A red box highlights the 'Rulesets' section under 'Code and automation'. A red arrow points from the 'enforcement=disabled' note in the 'Protect your most important branches' section to the 'Enforcement status' dropdown, which is set to 'Active'. Below this is a 'Bypass list' section with a red box around it, showing an empty list.

Settings · Ruleset · Android-Unit-Testing · +

https://github.com/SuperBruceJia/Android-Unit-Testing/settings/rules/new?target=branch&enforcement=disabled

SuperBruceJia / Android-Unit-Testing

Code Issues Pull requests Actions Projects Security Insights Settings

General

Access

Collaborators

Code and automation

Branches

Tags

Rules

Rulesets

Actions

Webhooks

Codespaces

Pages

Rulesets

Code Merging Rule

Active

Bypass list

+ Add bypass

Bypass list is empty

Targets

Which branches do you want to make a ruleset for?

Target branches

Part 5 — Code Merging after Passing Unit Testing

This has been set for all of you

pull requests cannot be merged until all the testing has passed

The screenshot shows the 'Merge' section of a GitHub pull request settings page. It includes the following configurations:

- Require signed commits**: Commits pushed to matching refs must have verified signatures.
- Require a pull request before merging**: Require all commits be made to a non-target branch and submitted via a pull request before they can be merged. This setting is highlighted with a blue border.
- Dismiss stale pull request approvals when new commits are pushed**: New, reviewable commits pushed will dismiss previous pull request review approvals.
- Require review from Code Owners**: Require an approving review in pull requests that modify files that have a designated code owner.
- Require approval of the most recent reviewable push**: Whether the most recent reviewable push must be approved by someone other than the person who pushed it.
- Require conversation resolution before merging**: All conversations on code must be resolved before a pull request can be merged.
- Request pull request review from Copilot**: Automatically request review from Copilot for new pull requests, if the author has access to Copilot code review. This setting is also highlighted with a blue border.
- Allowed merge methods**: Set to 'Merge, Squash, Rebase'.
- Require status checks to pass**: Choose which status checks must pass before the ref is updated. When enabled, commits must first be pushed to another ref where the checks pass. This setting is highlighted with a red border.
- Require branches to be up to date before merging**: Whether pull requests targeting a matching branch must be tested with the latest code. This setting will not take effect unless at least one status check is enabled.
- Do not require status checks on creation**: Allow repositories and branches to be created if a check would otherwise prohibit it.
- Status checks that are required**: A table showing a single row for 'action' with 'Any source' and a delete icon.
- Block force pushes**

Part 6 — Weather Demo Unit Testing

We need these two dependencies in the `build.gradle` file:

```
testImplementation("org.mockito:mockito-core:5.11.0")
testImplementation("androidx.arch.core:core-testing:2.2.0")
```



The screenshot shows the Android Studio code editor with the `build.gradle.kts` file open. On the left, the project structure sidebar shows files like `mipmap`, `values`, `xml`, `res (generated)`, and Gradle Scripts. The `build.gradle.kts` file is selected. The code editor displays the following content:

```
31
32 }
33
34 dependencies {
35     testImplementation("org.mockito:mockito-core:5.11.0")
36     testImplementation("androidx.arch.core:core-testing:2.2.0")
37         implementation(libs.appcompat)
38         implementation(libs.material)
39         implementation(libs.activity)
40         implementation(libs.constraintlayout)
41         testImplementation(libs.junit)
42         implementation("com.squareup.okhttp3:okhttp:4.12.0")
43         androidTestImplementation(libs.ext.junit)
44         androidTestImplementation(libs.espresso.core)
45 }
```

A red rectangular box highlights the two `testImplementation` lines at the top of the `dependencies` block. The code editor interface includes standard icons for file operations (New, Open, Save, etc.) and navigation.

Credits: <https://github.com/BU-EC327-Spring2025/weather-demo/blob/main/app/build.gradle.kts#L43-L56>

Part 6 – Weather Demo Unit Testing

import necessary packages and dependencies:

For Mockito framework:

```
import org.junit.Before;  
import org.junit.Test;  
import org.mockito.Mockito;
```

For @RunWith(AndroidJUnit4.class) framework:

```
import androidx.test.ext.junit.runners.AndroidJUnit4;  
import org.junit.Test;  
import org.junit.runner.RunWith;
```

Credits:

- [1]<https://github.com/BU-EC327-Spring2025/weather-demo/blob/main/app/src/test/java/com/example/weatherdemo/weatherapis/WeatherAPIServiceTest.java>
- [2]<https://github.com/BU-EC327-Spring2025/weather-demo/blob/main/app/src/androidTest/java/com/example/weatherdemo/DependencyManagerTest.java>

Part 6 – Weather Demo Unit Testing



```
WeatherAPIServiceTest.java
23 public class WeatherAPIServiceTest {
24     2 usages
25     static final String HEALTHY_JSON =
26         """
27         {
28             "location": {
29                 "name": "Boson",
30                 "region": "West Java",
31                 "country": "Indonesia",
32                 "lat": -6.975,
33                 "lon": 106.9983,
34                 "tz_id": "Asia/Jakarta",
35                 "localtime_epoch": 1744683387,
36                 "localtime": "2025-04-15 09:16"
37             },
38             "current": {
39                 "last_updated_epoch": 1744683300,
40                 "last_updated": "2025-04-15 09:15",
41                 "is_day": 1,
```

Credits: <https://github.com/BU-EC327-Spring2025/weather-demo/blob/main/app/src/test/java/com/example/weatherdemo/weatherapis/WeatherAPIServiceTest.java>

Part 6 – Weather Demo Unit Testing

just testing json parsing

```
WeatherAPIServiceTest.java
23  public class WeatherAPIServiceTest {
75      |     8 usages
76      |     WeatherAPIService weatherAPIService;
77
78    >     @Before
79    >     public void setUp() { weatherAPIService = new WeatherAPIService(skipInitialization: true); }
80
81
82    @Test
83    public void parseJson_success(){
84        WeatherData result = weatherAPIService.parseJSONResponse(city: "boston", HEALTHY_JSON);
85        assertEquals(expected: "boston", result.city);
86        assertEquals(expected: 75, result.degreesFahrenheit, delta: 0.001);
87    }
88
89    @Test(expected = IllegalStateException.class)
90    public void parseJson_failure(){
91        WeatherData result = weatherAPIService.parseJSONResponse(city: "boston", result: "<corrupt-json>");
92    }
93
```

Part 6 — Weather Demo Unit Testing

```
WeatherAPIServiceTest.java ✘
23  public class WeatherAPIServiceTest {
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116 }
```

mock out httpClient and mainThreadHandler and cover the code in WeatherAPIService around the logic of making that API call and handling its result

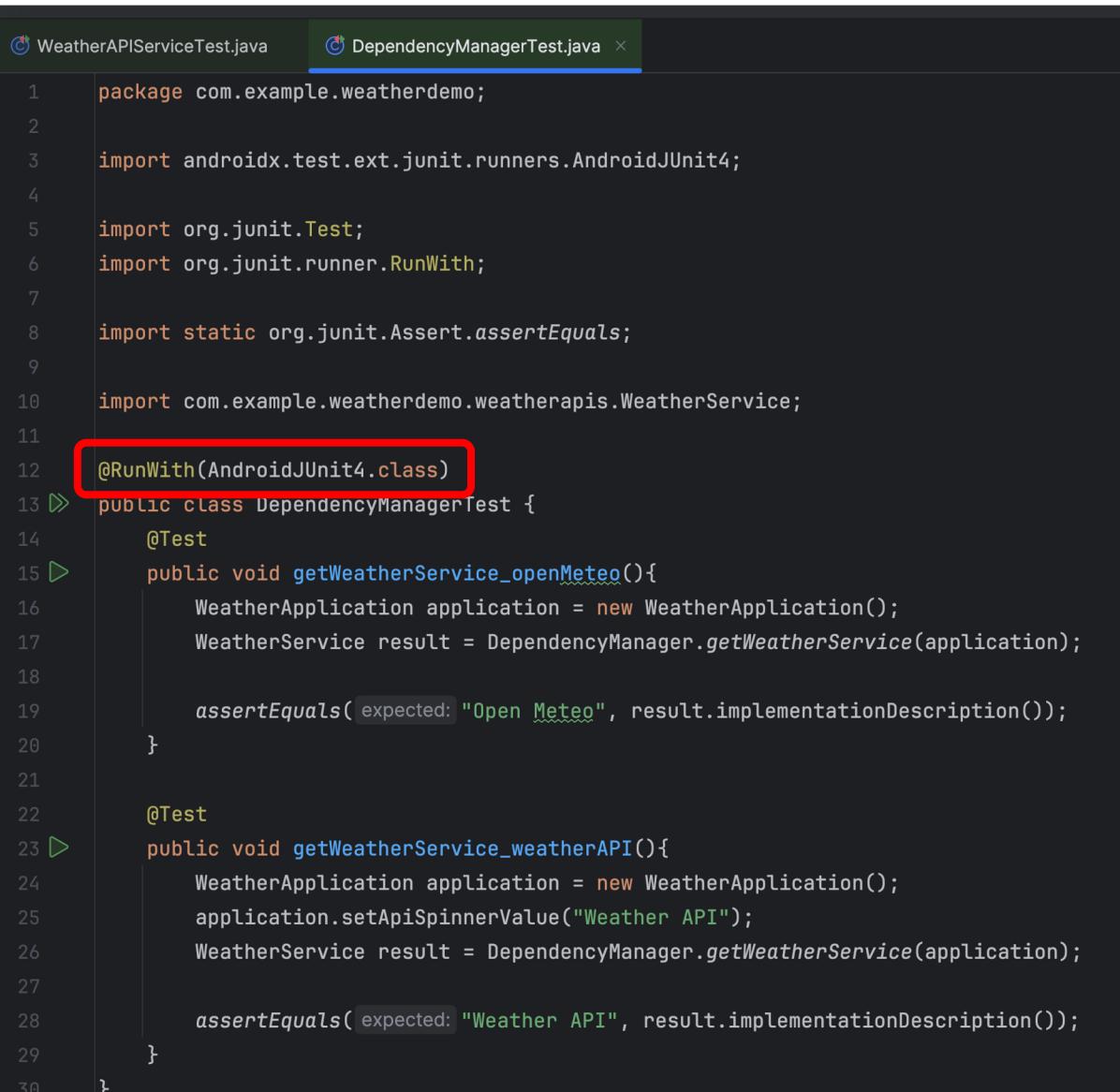
standard Java JUnit + Mockito unit test. There is nothing Android specific there

Part 6 — Weather Demo Unit Testing

```
@Test  
public void makeAPICall_failure() throws IOException {  
    OkHttpClient mockHttpClient = Mockito.mock(OkHttpClient.class);  
    Handler mockHandler = Mockito.mock(Handler.class);  
  
    weatherAPIService.httpClient = mockHttpClient;  
    weatherAPIService.mainThreadHandler = mockHandler;  
  
    WeatherCallback mockCallback = Mockito.mock(WeatherCallback.class);  
    Call mockCall = Mockito.mock(Call.class);  
    Response mockHTTPResponse = Mockito.mock(Response.class);  
  
    Mockito.when(mockHttpClient.newCall(Mockito.any(Request.class))).thenReturn(mockCall);  
    Mockito.when(mockCall.execute()).thenReturn(mockHTTPResponse);  
  
    Mockito.when(mockHTTPResponse.isSuccessful()).thenReturn(false);  
  
    weatherAPIService.makeAPICall(city: "test-city", mockCallback);  
  
    Mockito.verify(mockHandler).post(Mockito.any(Runnable.class));  
}
```

mock out
httpClient and
mainThreadHandler and cover the
code in
WeatherAPIService around the logic
of making that
API call and
handling its result
standard Java
JUnit + Mockito
unit test. There is
nothing Android
specific there

Part 6 – Weather Demo Unit Testing



```
1 package com.example.weatherdemo;
2
3 import androidx.test.ext.junit.runners.AndroidJUnit4;
4
5 import org.junit.Test;
6 import org.junit.runner.RunWith;
7
8 import static org.junit.Assert.assertEquals;
9
10 import com.example.weatherdemo.weatherapis.WeatherService;
11
12 @RunWith(AndroidJUnit4.class)
13 public class DependencyManagerTest {
14     @Test
15     public void getWeatherService_openMeteo(){
16         WeatherApplication application = new WeatherApplication();
17         WeatherService result = DependencyManager.getWeatherService(application);
18
19         assertEquals( expected: "Open Meteo", result.implementationDescription());
20     }
21
22     @Test
23     public void getWeatherService_weatherAPI(){
24         WeatherApplication application = new WeatherApplication();
25         application.setApiSpinnerValue("Weather API");
26         WeatherService result = DependencyManager.getWeatherService(application);
27
28         assertEquals( expected: "Weather API", result.implementationDescription());
29     }
30 }
```

- for anything that touches an Android UI component, or an Android activity, or an Android main thread, we need to use `@RunWith(AndroidJUnit4.class)`

- these tests will be significantly slower because they will need to start the emulator and will get executed on the emulator instead of just in the JVM

Part 6 – Weather Demo Unit Testing

Run a **specific** Instrumented Test Class

```
$ ./gradlew connectedAndroidTest -  
Pandroid.testInstrumentationRunnerArguments.class=com.example.weatherdemo.DependencyManagerTest
```

Run a **specific** Instrumented Test Class

```
$ ./gradlew connectedAndroidTest -  
Pandroid.testInstrumentationRunnerArguments.class=com.example.weatherdemo.DependencyManagerTest#methodName
```

Run **All** Instrumented Tests

```
$ ./gradlew connectedAndroidTest test
```

Clean all build artifacts and test results

```
$ ./gradlew clean
```

Clean and then **run** all tests

```
$ ./gradlew clean test connectedAndroidTest
```

Unit Testing in Android with GitHub Action

```
base ~/Desktop/weather-demo git:(main) (10.827s)
./gradlew connectedAndroidTest test

> Task :app:compileDebugJavaWithJavaC
Note: /Users/brucejia/Desktop/weather-demo/app/src/main/java/com/example/weatherdemo/WeatherDetailsActivity.ja
va uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpat
h has been appended

> Task :app:testDebugUnitTest
ExampleUnitTest > addition_isCorrect PASSED
WeatherAPIServiceTest > makeAPICall_failure PASSED
WeatherAPIServiceTest > parseJson_success PASSED
WeatherAPIServiceTest > makeAPICall_success PASSED
WeatherAPIServiceTest > parseJson_failure PASSED

> Task :app:compileReleaseJavaWithJavaC
Note: /Users/brucejia/Desktop/weather-demo/app/src/main/java/com/example/weatherdemo/WeatherDetailsActivity.ja
va uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

> Task :app:connectedDebugAndroidTest
Starting 3 tests on Medium_Phone_API_36(AVD) - 16

Finished 3 tests on Medium_Phone_API_36(AVD) - 16
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpat
h has been appended

> Task :app:testReleaseUnitTest
ExampleUnitTest > addition_isCorrect PASSED
WeatherAPIServiceTest > makeAPICall_failure PASSED
WeatherAPIServiceTest > parseJson_success PASSED
WeatherAPIServiceTest > makeAPICall_success PASSED
WeatherAPIServiceTest > parseJson_failure PASSED
[Incubating] Problems report is available at: file:///Users/brucejia/Desktop/weather-demo/build/reports/proble
ms/problems-report.html

BUILD SUCCESSFUL in 10s
83 actionable tasks: 83 executed

⚠ Fix deprecation warnings in WeatherDetailsActivity.java. ⌘ Enter
base ~/Desktop/weather-demo git:(main) ✘ Pair ⌘ I Dispatch Beta ⌘ ⌘ I
./gradlew clean [→ ▾]
```

```
InstrumentationRunnerArguments.class=com.example.weatherdemo.DependencyManagerTest

> Task :app:compileDebugJavaWithJavac
Note: /Users/brucejia/Desktop/weather-demo/app/src/main/java/com/example/weatherdemo/WeatherDetailsActivity.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

> Task :app:connectedDebugAndroidTest
Starting 2 tests on Medium_Phone_API_36(AVD) - 16

Finished 2 tests on Medium_Phone_API_36(AVD) - 16
[Incubating] Problems report is available at: file:///Users/brucejia/Desktop/weather-demo/build/reports/problems/problems-report.html

BUILD SUCCESSFUL in 8s
59 actionable tasks: 59 executed
```

A Check for deprecated API usage in the code. ⌘ Enter

```
base ~/Desktop/weather-demo git:(main) ~ A Pair ⌘ | ⚙ Dispatch Beta ⌘ |
```

```
base ~/Desktop/weather-demo git:(main) (8.823s)
e./gradlew connectedAndroidTest -Pandroid.testInstrumentationRunnerArguments.class=com.example.weatherdemo.DependencyManagerTest

> Task :app:connectedDebugAndroidTest
Starting 2 tests on Medium_Phone_API_36(AVD) - 16

Finished 2 tests on Medium_Phone_API_36(AVD) - 16
[Incubating] Problems report is available at: file:///Users/brucejia/Desktop/weather-demo/build/reports/problems-report.html

BUILD SUCCESSFUL in 8s
59 actionable tasks: 59 executed

base ~/Desktop/weather-demo git:(main) (0.87s)
./gradlew clean

BUILD SUCCESSFUL in 677ms
1 actionable task: 1 executed

base ~/Desktop/weather-demo git:(main) ~ Pair ⌘ I Dispatch Beta ⌘ I
clear → ▾
```

Part 6 – Weather Demo Unit Testing

The screenshot shows the Android Studio interface with the project structure on the left and the build.gradle.kts file open in the main editor. The file contains Gradle configuration for an Android application. Two specific sections are highlighted with colored boxes:

- A blue box highlights the `android { ... }` block, which defines the application's namespace, compileSdk version, defaultConfig (applicationId, minSdk, targetSdk, versionCode, versionName), and testInstrumentationRunner.
- A red box highlights the `testOptions { ... }` block, which configures test options for unit tests, including test logging events and standard stream settings.

```
5 android {
6     namespace = "com.example.weatherdemo"
7     compileSdk = 35
8
9     defaultConfig {
10         applicationId = "com.example.weatherdemo"
11         minSdk = 26
12         targetSdk = 35
13         versionCode = 1
14         versionName = "1.0"
15
16         testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
17     }
18
19     buildTypes {
20         release {
21             isMinifyEnabled = false
22             proguardFiles(
23                 getDefaultProguardFile(name: "proguard-android-optimize.txt"),
24                 "proguard-rules.pro"
25             )
26         }
27     }
28     compileOptions {
29         sourceCompatibility = JavaVersion.VERSION_11
30         targetCompatibility = JavaVersion.VERSION_11
31     }
32     testOptions {
33         unitTests.all {
34             it.testLogging {
35                 events("passed", "skipped", "failed")
36                 showStandardStreams = true
37             }
38         }
39     }
40 }
```

At the bottom of the screen, the navigation bar shows the path: weather-demo > app > build.gradle.kts > android.

Part 6 – Weather Demo Unit Testing

You can add your `testOptions { }` inside the `android { }`

```
android {  
    ...  
  
    testOptions {  
        unitTests.all {  
            it.testLogging {  
                events("passed", "skipped", "failed")  
                showStandardStreams = true  
            }  
        }  
    }  
}
```

Part 6 – Weather Demo Unit Testing

The screenshot shows the Android Studio interface with the project structure and code editor for a Weather Demo application.

Project Structure: The left sidebar shows the project structure under "Android". The "app" module contains Java files for the main application and test packages: com.example.weatherdemo, com.example.weatherdemo.androidTest, and com.example.weatherdemo.test. The "build.gradle.kts" file is open in the code editor.

Code Editor: The code editor displays the "build.gradle.kts" file for the "app" module. It includes configuration for Java version compatibility, test options, and logging settings. A red arrow points from the "showStandardStreams = true" line in the "testOptions" block down to the "Test Results" section.

Test Results: The bottom section shows the "Test Results" tab with the output of the unit tests. The results indicate 3 of 3 tests passed in 506 ms. The log output shows the tasks executed during the build and the successful execution of three test cases: "testFetchWeatherSuccess", "testFetchWeatherFailure", and "testFetchWeatherStatusUpdate". A red box highlights the test results output.

```
build.gradle.kts (app) - build.gradle.kts (WeatherDemo)
```

```
Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properly.
```

```
5 android {  
6     buildTypes {  
7         ...  
8     }  
9     compileOptions {  
10        sourceCompatibility = JavaVersion.VERSION_11  
11        targetCompatibility = JavaVersion.VERSION_11  
12    }  
13    testOptions {  
14        unitTests.all {  
15            it.testLogging {  
16                events("passed", "skipped", "failed")  
17                showStandardStreams = true  
18            }  
19        }  
20    }  
21}
```

```
Tests passed: 3 of 3 tests – 506 ms
```

```
> Task :app:processDebugManifestForPackage UP-TO-DATE  
> Task :app:processDebugResources UP-TO-DATE  
> Task :app:compileDebugJavaWithJavac UP-TO-DATE  
> Task :app:bundleDebugClassesToRuntimeJar UP-TO-DATE  
> Task :app:bundleDebugClassesToCompileJar UP-TO-DATE  
> Task :app:preDebugUnitTestBuild UP-TO-DATE  
> Task :app:javaPreCompileDebugUnitTest UP-TO-DATE  
> Task :app:compileDebugUnitTestJavaWithJavac UP-TO-DATE  
> Task :app:processDebugJavaRes NO-SOURCE  
> Task :app:processDebugUnitTestJavaRes NO-SOURCE  
> Task :app:testDebugUnitTest  
WARNING: A Java agent has been loaded dynamically (/Users/brucejia/.gradle/caches/modules-2/files-2.1/net.bytebuddy/byte-buddy-agent/1.14.12/be4984cb6fd1ef1d1f218a648889dfda44b8a15/byte-buddy-agent.jar)  
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning  
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information  
WARNING: Dynamic loading of agents will be disallowed by default in a future release  
WeatherServiceTest > testFetchWeatherSuccess PASSED  
WeatherServiceTest > testFetchWeatherFailure PASSED  
WeatherServiceTest > testFetchWeatherStatusUpdate PASSED  
BUILD SUCCESSFUL in 939ms  
20 actionable tasks: 1 executed, 19 up-to-date
```

```
Build Analyzer results available  
15:10:11: Execution finished ':app:testDebugUnitTest --tests "WeatherServiceTest"'.
```

Part 7 — Takeaway

Use **Google Test** for **C++** codes

Use **JUnit with Gradle** for **Android Java/Kotlin** codes

Use **Android CI with GitHub Actions** for **Android**

Testing

Setting **Rules** on GitHub for **code merging**

Use **Mockito** to replace any **network**, **file system**, or
async behavior with canned responses

Use `@RunWith(AndroidJUnit4.class)` for **Android UI components**, **Android activity**, or **Android main thread**

Use `testOptions` for testing **event report**

Part 7 — Takeaway (Credits and Links)

Google Test for C++ codes

[Android-Unit-Testing/tree/main/GoogleTest/tests](#)

Use JUnit with Gradle for Android Java/Kotlin codes

[1] [Android-Unit-Testing](#)

[2] [weatherdemo/weatherapis/WeatherAPIServiceTest.java](#)

Use Mockito to replace any network, file system, or async behavior with canned responses

[weatherdemo/weatherapis/WeatherAPIServiceTest.java](#)

Use @RunWith(AndroidJUnit4.class) for Android UI components, Android activity, or Android main thread

[weatherdemo/DependencyManagerTest.java](#)

Use testOptions for testing event report

[weather-demo/blob/main/app/build.gradle.kts](#)

Thank you very much for your attention!