



---

# Android Calculator Architecture

---

## Capstone Project Part - B



Name of Testers:

1. Hrushikesh
2. Harshvardhan
3. Anupam
4. Krishna
5. Manish
6. Asha
7. Tahir

Trainer:

Prof. Abhishek Sengupta

June 7, 2024

## Revision Sheet

| Version | Date       | Author        | Description of change  |
|---------|------------|---------------|--|
| 1.1     | 03/06/2024 | Hrushikesh    | Introduction, Layered Architecture for Calculator                          |
| 1.2     | 03/06/2024 | Harsh Vardhan | Test Strategy: Test Objectives, Test Assumptions, Test Principles          |
| 1.3     | 03/06/2024 | Anupam        | Test Acceptance Criteria   |
| 1.4     | 04/06/2024 | Krishna       | Test Deliverables  |
| 1.5     | 04/06/2024 | Tahir         | Test Driven Development: Overview, History, Process                        |
| 1.6     | 05/06/2024 | Asha          | Requirement Traceability Matrix  |
| 1.7     | 05/06/2024 | Manish        | Risk Management, Diagnostic Error Codes for Android Calculator, Conclusion |
| 1.8     | 06/06/2024 | Group-1       | Final Draft  |

# Table of content

## 1. Introduction

- 1.1 Purpose
- 1.2 project Overview
- 1.3 Audience

## 2. Test Strategy

- 2.1 Test Objectives
- 2.2 Test Assumptions
- 2.3 Test Principles
- 2.4 Data Approach
- 2.5 Scope and Levels of Testing
  - 2.5.1 Unit Testing
    - Application Layer
    - Runtime Environment
  - 2.5.2 Integration Testing

## 3. Test Acceptance Criteria

## 4. Test Deliverables

## 5. Test Driven Development

- 5.1 What is Test-Driven Development?
- 5.2 History of Test-Driven Development.
- 5.3 Test-Driven work in Test-Driven Development
- 5.4 Process of Test-Driven Development
- 5.5 TDD vs Traditional Testing
  - 5.5.1 Approach

- 5.5.2 Testing Scope
- 5.5.3 Iterative
- 5.5.4 Debugging
- 5.5.5 Documentation
- 5.6 Pros and Cons of TDD

## 6. Requirement Traceability Matrix

- 6.1 What is Requirement Traceability Matrix?
- 6.2 Why is Requirement Traceability Matrix Important?
- 6.3 Parameters of Requirement Traceability Matrix
- 6.4 Types of Traceability Matrix
  - 6.4.1 Forward Traceability Matrix
  - 6.4.1 Backward Traceability Matrix
  - 6.4.3 Bi-Directional Traceability
- 6.5 Traceability matrix for a QA process for Android Calculator

## 7. Risk Management

- 7.1 Identify Potential Risks
- 7.2 Migration Strategies

## 8. Layered Architecture for Calculator

## 9. Diagnostic Error Codes for Android Calculator

## 10. Conclusion

# 1. Introduction to Android Calculator

## 1.1 Purpose:

The purpose of the Android calculator QA process is multifaceted. Primarily, it aims to ensure the reliability, accuracy, and user-friendliness of the calculator application. By conducting thorough testing, the QA team verifies that the calculator performs arithmetic operations correctly, handles user inputs accurately, and delivers an intuitive user experience. Additionally, the QA process aims to identify and mitigate any potential issues or defects that may impact the functionality or usability of the app, thereby enhancing its overall quality and reliability.

## 1.2 Project Overview:

The Android calculator app serves as an essential tool for users across various demographics. Its primary function is to provide basic arithmetic functionalities, including addition, subtraction, multiplication, and division, through a user-friendly interface. The calculator app caters to a wide range of users, from students and professionals to individuals requiring quick calculations in their daily lives. With its intuitive design and ease of use, the Android calculator app aims to simplify mathematical tasks and enhance user productivity.

## 1.3 Audience:

The target audience for the Android calculator app encompasses end-users who rely on the application for their everyday calculations. This includes students studying mathematics, professionals working in fields such as finance or engineering, and individuals performing routine calculations in their personal or professional lives. The QA process ensures that the calculator app delivers a seamless experience across various Android devices, catering to the diverse needs and preferences of its user base. By addressing the requirements and expectations of its audience, the calculator app aims to become a trusted and indispensable tool for users across different contexts.

## 2. Test Strategy

### 2.1 Test Objectives:

The primary objectives of the test strategy for the Android calculator app are to validate arithmetic operations, ensure correct UI functionality, and detect potential bugs or issues.

- Validating arithmetic operations involves verifying the accuracy of addition, subtraction, multiplication, and division functionalities.
- Ensuring correct UI functionality includes testing user interactions, button responsiveness, input validation, and error handling.
- Detecting potential bugs or issues involves identifying and resolving any defects that may impact the functionality, performance, or user experience of the calculator app.

### 2.2 Test Assumptions:

The test strategy makes certain assumptions to facilitate effective testing:

- Availability of adequate testing resources, including testing devices, testing environments, and testing tools.
- Access to different Android devices for compatibility testing to ensure the calculator app functions correctly across various screen sizes, resolutions, and Android versions.
- Collaboration with development teams to address any issues or defects identified during testing and to ensure timely resolution.

### 2.3 Test Principles:

The test strategy is guided by several principles to ensure the effectiveness and efficiency of the testing process:

- Thorough testing coverage: The test strategy aims to achieve comprehensive test coverage, addressing all functional and non-functional requirements of the calculator app.

- Adherence to testing standards: Testing activities adhere to established testing standards and best practices to ensure consistency, reliability, and repeatability.
- Continuous improvement: The test strategy emphasizes continuous improvement through feedback, lessons learned, and process refinement to enhance the quality and efficiency of testing activities.
- Collaboration between QA and development teams: Close collaboration between QA and development teams fosters communication, knowledge sharing, and alignment of goals, leading to better outcomes and higher-quality software.

## 2.4 Data Approach:

The test strategy includes a data-driven approach to testing, utilizing various numerical inputs for arithmetic operations, edge cases, and boundary conditions to validate the calculator's robustness.

- Test data encompasses both typical and atypical scenarios to ensure comprehensive coverage of the calculator's functionality.
- Edge cases and boundary conditions are included to verify the calculator's behavior under challenging circumstances and to identify any potential vulnerabilities or limitations.

## 2.5 Scope and Levels of Testing:

### 2.5.1 Unit Testing:

Unit testing focuses on testing individual components of the calculator app, including:

- Arithmetic operations: Each arithmetic operation (addition, subtraction, multiplication, division) is tested independently to verify its correctness and accuracy.
- Application layer functionalities: Unit tests validate the functionality of different modules and components within the application layer, ensuring they perform as expected.
- Runtime environment compatibility: Unit tests assess the compatibility of the calculator app with different Android versions and device configurations, ensuring consistent behavior across various environments.

# Testing the Calculator App

## Testing Approach and Tools

Testing ensures that the calculator app functions correctly and handles edge cases appropriately. The testing process will involve unit testing to verify individual components and integration testing to ensure that all components work together as expected.

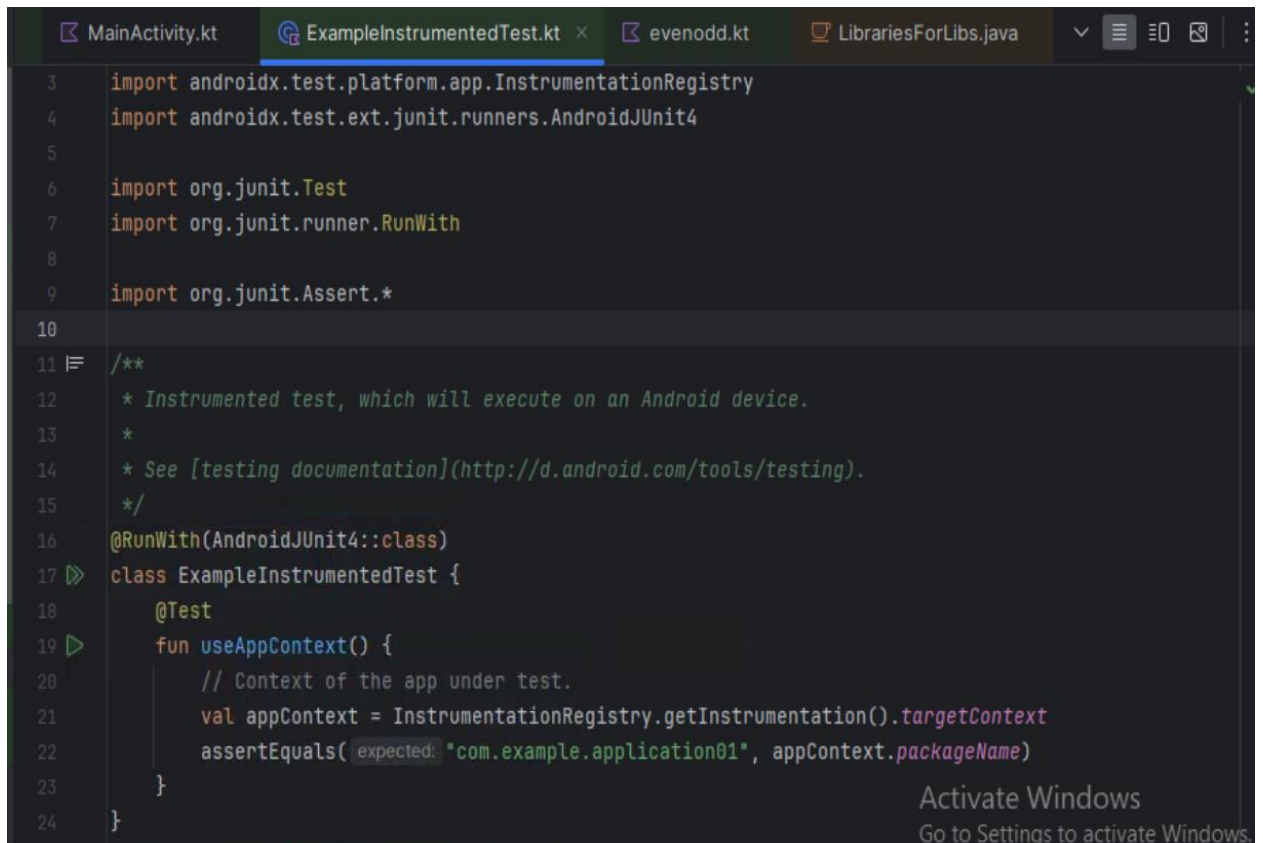
### Tools Used for Testing:

- JUnit: A framework for writing and running unit tests in Java and Kotlin.
- Espresso: A testing framework for writing concise and reliable UI tests for Android applications.

### Sample Test Cases for Android Studio

The below picture represents the sample test code built using Junit and Kotlin in android studio. Further snippets represent unit cases for different functionalities such as addition, subtraction, multiplication etc. To test the cases for integration testing we have chosen integer combinations of different types including two positive integers as well as integers of different annotations in a single function so that it can be verified that the test results remain the same for all cases.





```
3 import androidx.test.platform.app.InstrumentationRegistry
4 import androidx.test.ext.junit.runners.AndroidJUnit4
5
6 import org.junit.Test
7 import org.junit.runner.RunWith
8
9 import org.junit.Assert.*
10
11 /**
12  * Instrumented test, which will execute on an Android device.
13  *
14  * See [testing documentation](http://d.android.com/tools/testing).
15  */
16 @RunWith(AndroidJUnit4::class)
17 class ExampleInstrumentedTest {
18     @Test
19     fun useAppContext() {
20         // Context of the app under test.
21         val appContext = InstrumentationRegistry.getInstrumentation().targetContext
22         assertEquals("expected: \"com.example.application01\", appContext.packageName)
23     }
24 }
```

## Subtraction Test Case:

The first test method verifies whether the subtraction operation for positive numbers in the Android calculator app's calculatorFunction class returns the correct result (-10). If the actual result matches the expected result, the test will pass; otherwise, it will fail, indicating a potential issue with the subtraction functionality for positive numbers.

The second test method verifies whether the subtraction operation for negative numbers in the Android calculator app's calculatorFunction class returns the correct result (10). If the actual result matches the expected result, the test will pass; otherwise, it will fail, indicating a potential issue with the subtraction functionality.

```

@Test
fun subtraction_Of_Both_Positive_Num() {
    Assert.assertEquals(calculatorFunction.subtraction( num1: 10, num2: 20), actual: -10)
}

@Test
fun subtraction_Of_Both_Negative_Num() {
    Assert.assertEquals(calculatorFunction.subtraction( num1: -10, num2: -20), actual: 10)
}

```

## Addition Test

```

@Test
fun addition_Of_Positive_And_Negative_Num() {
    Assert.assertEquals(calculatorFunction.addition( num1: 10, num2: -20), actual: -10)
}

@Test
fun addition_Of_Both_Negative_num() {
    Assert.assertEquals(calculatorFunction.addition( num1: -10, num2: -20), actual: -30)
}

```

## Multiplication Test Case

```

@Test
fun multiplication_Of_Both_Positive_Num() {
    Assert.assertEquals(calculatorFunction.multiplication( num1: 10, num2: 20), actual: 200)
}

@Test
fun multiplication_of_Positive_And_Negative_Num() {
    Assert.assertEquals(calculatorFunction.multiplication( num1: 10, num2: -3), actual: -30)
}

```

## Division Test Case:

```
@Test
fun division_Of_Both_Positive_Num() {
    Assert.assertEquals(calculatorFunction.divide(num1: 30, num2: 15), actual: 2)
}

@Test
fun division_Of_Positive_And_Negative_Num() {
    Assert.assertEquals(calculatorFunction.divide(num1: 40, num2: -20), actual: -2)
}
```

## Test Results After the Execution of all Test cases pass

The screenshot displays the 'Run' window of an IDE, showing the results of a test execution. The 'Test Results' tab is active, displaying a list of tests under the 'CalculatorTest' suite. All tests passed, with a total of 10 tests passed in 7ms. The 'Test Results' list includes:

- division\_Of\_Both\_Positive\_Num (5ms)
- division\_Of\_Positive\_And\_Negative\_Num (0ms)
- addition\_Of\_Negative\_And\_Positive\_Num (0ms)
- addition\_Of\_Both\_Negative\_num (0ms)
- addition\_Of\_Two\_Positive\_Num (1ms)
- multiplication\_Of\_Positive\_And\_Negative\_Num (1ms)
- subtraction\_Of\_Both\_Positive\_Num (0ms)
- addition\_Of\_Positive\_And\_Negative\_Num (0ms)
- subtraction\_Of\_Both\_Negative\_Num (0ms)
- multiplication\_Of\_Both\_Positive\_Num (0ms)

The 'Output' tab shows the following build and test logs:

```
> Task :app:javaRecompileDebugUnitTest UP-TO-DATE
> Task :app:processDebugJavaRes UP-TO-DATE
> Task :app:compileDebugUnitTestKotlin
> Task :app:compileDebugUnitTestJavaWithJavac NO-SOURCE
> Task :app:processDebugUnitTestJavaRes UP-TO-DATE
> Task :app:testDebugUnitTest
BUILD SUCCESSFUL in 3s
22 actionable tasks: 2 executed, 20 up-to-date

Build Analyzer results available
20:09:09: Execution finished 'app:testDebugUnitTest --tests "com.example.application01
.CalculatorTest"'.
Activate Windows
Go to Settings to activate Windows.
```

### 2.5.2 Integration Testing:

Integration testing validates the interaction between different modules and ensures seamless functionality across the entire application.

- Integration tests verify that individual components work together harmoniously and that data flows correctly between different parts of the calculator app.
- Integration testing also includes end-to-end testing to validate the calculator's functionality from user input to output, ensuring a smooth and error-free user experience.

## 3. Test Acceptance Criteria

The test acceptance criteria for the Android calculator app outline the specific conditions and requirements that must be met for the application to be considered acceptable for release. These criteria encompass various aspects of the app's functionality, usability, and compatibility to ensure a high-quality user experience.

### - Arithmetic Operations Accuracy:

- All arithmetic operations, including addition, subtraction, multiplication, and division, must produce accurate results according to mathematical principles.
- Test cases are designed to cover a wide range of input values, including positive and negative numbers, decimals, and large numbers, to validate the correctness of the calculations.

### - Intuitive and Responsive User Interface:

- The user interface (UI) of the calculator app must be intuitive, easy to use, and responsive to user inputs.
- Test cases evaluate the responsiveness of UI elements such as buttons, input fields, and calculations, ensuring that users can interact with the app effortlessly.

- Seamless Functionality Across Devices and Android Versions:

- The calculator app must function seamlessly across different screen sizes, resolutions, and Android versions to accommodate a diverse range of devices.

- Compatibility testing is conducted on various Android devices, including smartphones and tablets, running different versions of the Android operating system (OS), ensuring consistent behavior and performance across the board.

- Error Handling and Validation:

- The app must handle errors gracefully and provide informative error messages to users in case of invalid inputs or calculation errors.

- Test cases verify that error messages are displayed correctly, and users are guided on how to rectify the issue, enhancing the overall usability and user satisfaction.

- Accessibility and Usability:

- The calculator app should be accessible to users with different abilities, ensuring compliance with accessibility standards and guidelines.

- Usability testing is conducted to assess the ease of use and accessibility features of the app, including font size options, color contrast, and screen reader compatibility.

- Performance and Stability:

- The app must demonstrate stable performance under normal usage conditions, with minimal lag or slowdowns during arithmetic operations.

- Performance testing evaluates the app's responsiveness, load times, and resource usage, ensuring that it performs optimally even under heavy usage or resource constraints.

- Security and Privacy:

- The calculator app must adhere to security best practices to protect user data and ensure the privacy of sensitive information.

- Security testing assesses the app's vulnerability to common security threats such as data breaches or unauthorized access, ensuring that user data remains secure and confidential.

By defining clear and specific acceptance criteria, the QA team ensures that the Android calculator app meets the expectations and requirements of its users, delivering a reliable, user-friendly, and high-quality application.

## 4. Test Deliverables

In the context of QA for an Android calculator developed using Android Studio, test deliverables play a critical role in ensuring the app's functionality, reliability, and user experience meet predefined standards. The comprehensive test plan serves as a blueprint, delineating test objectives, methodologies, and schedules tailored specifically to the intricacies of the calculator application. Within this plan, meticulous attention is given to assessing various functionalities, including basic arithmetic operations, memory functions, scientific calculations, and user interface interactions.

Detailed test cases form the backbone of the QA process, encompassing a myriad of scenarios to rigorously validate the calculator's performance across different input combinations, edge cases, and device configurations. Each test case meticulously outlines the steps to be executed, the expected outcomes, and the criteria for determining pass or fail results. From verifying simple addition and subtraction operations to scrutinizing complex mathematical computations, these test cases ensure thorough coverage of all features and functionalities.

Throughout the testing phase, diligent documentation is maintained via test reports, capturing comprehensive records of test results, observed behaviors, and any encountered issues or anomalies. These reports serve as invaluable artifacts for tracking the app's testing progress, facilitating collaboration among QA team members, and informing stakeholders of the application's readiness for release.

Finally, the culmination of the testing process is marked by the preparation of final sign-off documentation, signaling the app's readiness for deployment. This documentation encapsulates the collective assurance of the QA team, affirming that all identified issues have been addressed, and the application meets the requisite quality standards for release to end-users.

## 5. Test Driven Development

Let's delve deeper into each point in the context of using Test-Driven Development (TDD) for developing an Android calculator application using Android Studio as a QA engineer:

### 5.1 What is Test-Driven Development?

- In the development of the Android calculator app, TDD entails writing tests before implementing the actual code. This means defining test cases that specify the expected behavior of different calculator functionalities, such as addition, subtraction, multiplication, division, and memory operations. These tests serve as a blueprint for guiding the development process and ensuring that the implemented features align with the desired behavior outlined in the tests.

### 5.2 History of Test-Driven Development:

- Originating from the agile software development community, TDD has gained recognition for its efficacy in enhancing code quality and reducing bugs. In the context of developing the Android calculator, TDD's adoption reflects a commitment to achieving robust and reliable software by iteratively refining code through rigorous testing.

### 5.3 Test-Driven Work in Test-Driven Development:

- As a QA engineer utilizing TDD for the Android calculator, your role involves crafting failing tests that articulate the expected behavior of calculator functionalities. Subsequently, you collaborate with developers to incrementally implement code that fulfills these test criteria. Through this iterative process, the code undergoes continuous refinement and optimization, guided by the feedback provided by failing and passing tests.

#### 5.4 Process of Test-Driven Development:

- The TDD process for the Android calculator comprises writing failing tests to specify desired behavior, followed by writing the minimum amount of code necessary to pass those tests. Finally, you engage in refactoring the code to enhance its design and maintainability, ensuring that all tests continue to pass throughout the process.

#### 5.5 TDD vs Traditional Testing

- In contrast to traditional testing approaches, TDD for the Android calculator prioritizes writing tests before code implementation, fostering a proactive approach to quality assurance. While traditional testing encompasses various levels such as unit, integration, and system testing, TDD predominantly focuses on unit testing individual components of the calculator app. Additionally, TDD's iterative nature facilitates early bug detection and easier debugging, contributing to the app's overall reliability and stability.

#### 5.6 Pros and Cons of TDD:

- Embracing TDD for the Android calculator offers numerous advantages, including improved code quality, reduced bugs, better design, faster feedback loop, and increased developer confidence. However, it's important to acknowledge potential challenges such as an initial learning curve and perceived increase in development time. Nevertheless, these drawbacks are outweighed by the long-term benefits of TDD in fostering a culture of quality and reliability in software development.

## 6. Requirement Traceability Matrix

- 6.1 What is Requirement Traceability Matrix?
  - A Requirement Traceability Matrix (RTM) is a document used to track and manage the relationship between requirements and test cases throughout the software development lifecycle.
  - It serves as a mapping tool, linking each requirement to its corresponding test case(s) to ensure comprehensive test coverage and validation of all specified functionalities.



- 6.2 Why is Requirement Traceability Matrix Important?
  - RTM ensures alignment between the project's requirements and the test cases designed to verify them, helping to validate that all requirements are adequately tested.
  - It provides traceability, enabling stakeholders to track the status of requirements and their associated test cases, ensuring transparency and accountability throughout the project.
  - RTM facilitates impact analysis by identifying the potential impact of changes to requirements on existing test cases and vice versa, aiding in decision-making and change management processes.
  
- 6.3 Parameters of Requirement Traceability Matrix
  - The RTM typically includes parameters such as requirement ID, requirement description, test case ID, test case description, status, and comments.
  - Requirement ID and test case ID serve as unique identifiers for easy reference and cross-referencing.
  - Description provides detailed information about each requirement and test case, ensuring clarity and understanding.
  - Status indicates the current status of each requirement and test case (e.g., pass, fail, in progress), helping to track progress and identify areas needing attention.
  - Comments allow for additional notes or explanations to provide context and insights into the status or implementation of requirements and test cases.
  
- 6.4 Types of Traceability Matrix
  - 6.4.1 Forward Traceability Matrix: Maps requirements to test cases, ensuring that each requirement has corresponding test coverage. It helps ensure that all requirements are adequately validated through testing.
  - 6.4.2 Backward Traceability Matrix: Maps test cases back to requirements, ensuring that all test cases are linked to specific requirements. It helps ensure that all test cases are aligned with project requirements and objectives.
  - 6.4.3 Bi-Directional Traceability: Combines forward and backward traceability, providing a comprehensive view of the relationship between requirements and test cases. It ensures traceability in both directions, facilitating better understanding and management of project scope and objectives.

- 6.5 Traceability Matrix for a QA Process for Android Calculator
  - For the Android calculator QA process, the traceability matrix would map each requirement related to calculator functionalities (e.g., addition, subtraction, multiplication, division) to corresponding test cases.
  - Each test case would be designed to validate a specific requirement, ensuring comprehensive coverage of all calculator functionalities and ensuring that the app meets user expectations and quality standards.

| Test Case ID | Requirement ID | Description   | Output |
|--------------|----------------|---|--------|
| TC001        | REQ001         | Verify that the calculator can perform addition correctly                           | PASS   |
| TC002        | REQ002         | Verify that the calculator can perform subtraction correctly                        | PASS   |
| TC003        | REQ003         | Verify that the calculator can perform multiplication correctly                     | PASS   |
| TC004        | REQ004         | Verify that the calculator can perform division correctly                           | PASS   |
| TC005        | REQ005         | Verify that the calculator can handle negative numbers                              | PASS   |
| TC006        | REQ006         | Verify that the calculator can handle decimal numbers                               | PASS   |
| TC007        | REQ007         | Verify that the calculator can handle large numbers                                 | PASS   |
| TC008        | REQ008         | Verify that the calculator displays answers correctly in UI                         | PASS   |
| TC009        | REQ009         | Verify that the calculator clears the input field after a calculation is performed  | PASS   |
| TC010        | REQ010         | Verify that the calculator provides error messages for invalid inputs or operations | PASS   |

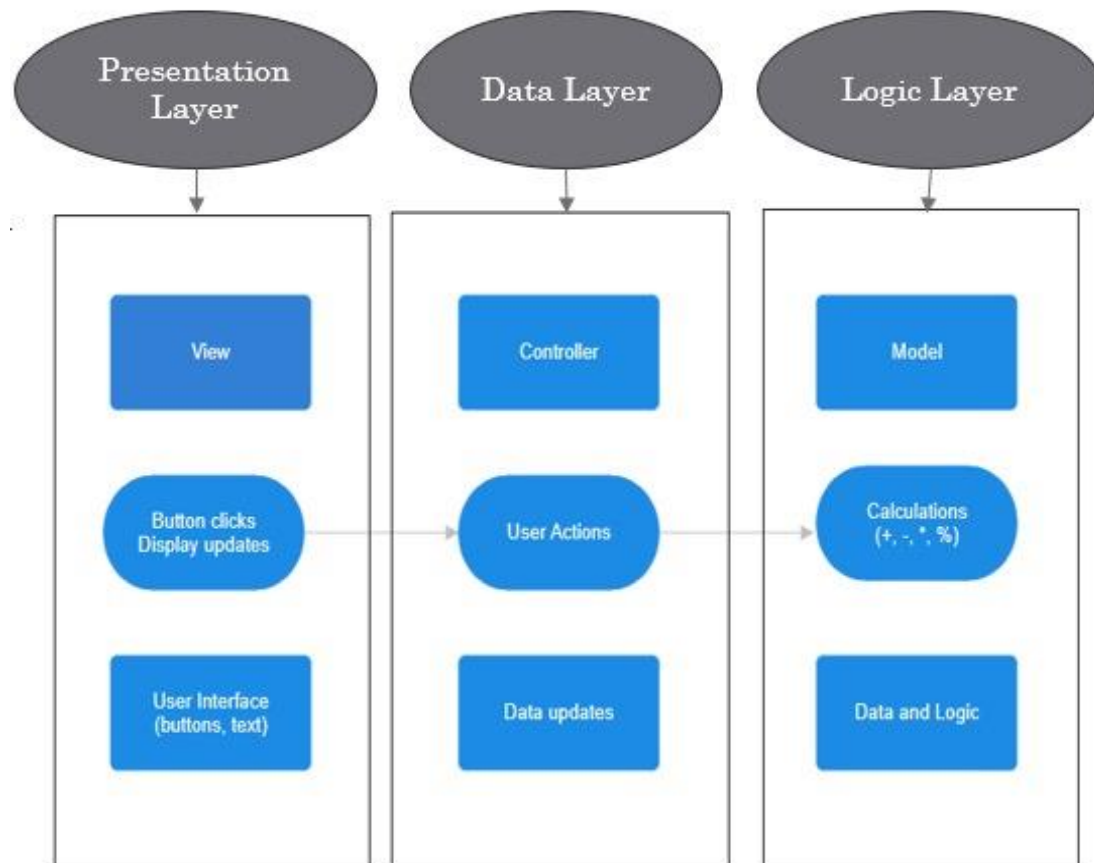
## 7. Risk Management

- 7.1 Identify Potential Risks:
  - The risk management process involves identifying potential risks that could impact the successful development, deployment, and usage of the Android calculator app.
  - Risks may include calculation inaccuracies, compatibility issues across different Android versions and devices, UI inconsistencies, performance issues, security vulnerabilities, and user data privacy concerns.
- 7.2 Migration Strategies:
  - Once risks are identified, migration strategies are developed to mitigate or manage these risks effectively.
  - Strategies may include proactive risk identification and assessment, continuous testing throughout the development lifecycle, collaboration between QA and development teams, version control, backup and recovery plans, and contingency measures.

## 8. Layered Architecture for Calculator

- This section would detail the layered architecture of the Android calculator app, outlining the different layers and their responsibilities:
  - Presentation Layer (UI): Responsible for the user interface, including layout, design, and interaction with users. Represents the user interface (UI) elements of the calculator. This includes the buttons for numbers, operators, and functions like clear, delete, etc. The view displays the current expression being built and the final result calculated by the model. Listens for user interactions like button clicks and updates the model accordingly.
  - Logic Layer: Contains the core logic and algorithms for performing mathematical calculations and other functionalities. Acts as the intermediary between the view and the model. Receives user input from the view (button clicks). Updates the model with the new data or operation. Retrieves the results from the model and updates the view (display) with the latest information. The controller essentially translates user actions into operations for the model and updates the UI based on the model's output.

- Data Layer: Manages data storage and retrieval, if applicable, such as storing user preferences or calculation history. Handles the data and logic behind the calculations. This can involve classes representing numbers, operators, and the overall expression being built. The model performs the actual calculations based on the user input and stores the intermediate and final results.



- The explanation would describe how each layer interacts with one another and their respective responsibilities in the overall functioning of the calculator application, ensuring clarity and understanding.

## 8. Flow Chart of Calculator Functionality

- In essence, this flowchart represents a fundamental model of how a basic calculator operates. It captures the user's input, processes it mathematically, and delivers the final outcome on the screen.

- Start: The process begins here.

- User inputs numbers & operations: The user interacts with the calculator by pressing buttons to enter numbers and desired operations (addition, subtraction, etc.).

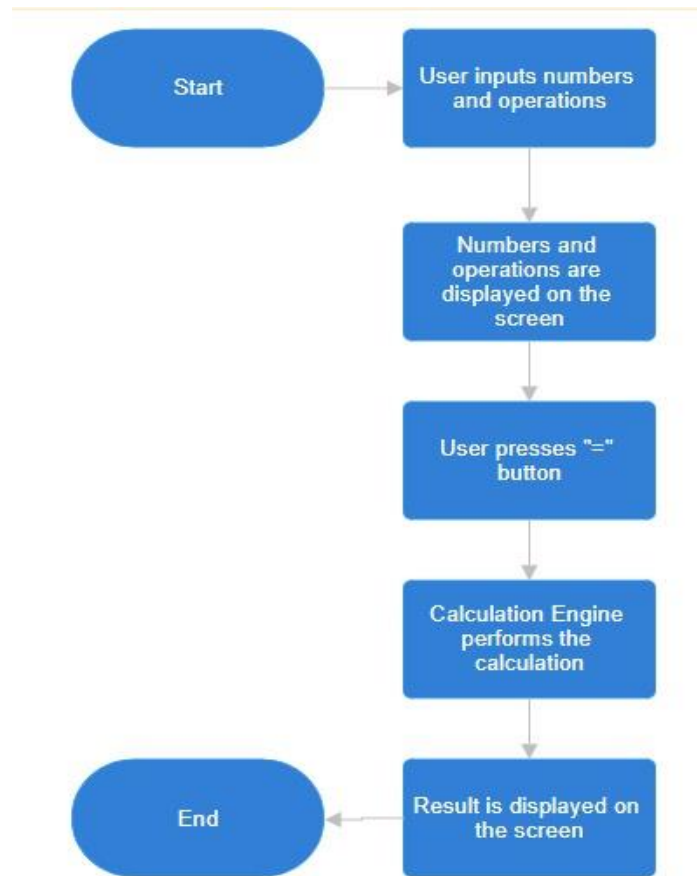
- As the user inputs numbers and operations, they are simultaneously displayed on the calculator's screen, providing a real-time view of the expression being built.

- User presses "=" button: Once the user has entered the complete expression, they press the "=" button to initiate the calculation.

- Calculation Engine performs the calculation: Upon receiving the equal sign command, the calculator's processing unit, referred to as the Calculation Engine in the flowchart, takes over. It retrieves the entered expression, processes it according to the order of operations (PEMDAS - Parentheses, Exponents, Multiplication and Division from left to right, Addition and Subtraction from left to right), and calculates the final result.

- Result is displayed on the screen: The calculated answer is then presented on the calculator's display for the user to view.

- End: The process reaches its conclusion after the result is shown on the screen. The calculator essentially waits for further user input to begin a new calculation cycle.



## Project Structure in Android Studio

### Key Directories and Files

- **app/src/main/java:**

Contains the Java code for the Android calculator app.  
Includes classes and packages responsible for implementing app functionality.

- **app/src/main/res:**

Stores resources used by the app, such as XML layout files, images, and strings.  
Organized into subdirectories for different types of resources (e.g., layout, drawable, values).

- Gradle Files:**

- app/build.gradle:**

Configuration file for the app module, including dependencies and build settings.

- Project-level Gradle files:**

- settings.gradle and build.gradle:**

Configure project-wide settings and dependencies.

Manage the overall build process for the Android calculator project.

## 9. Diagnostic Error Codes for Android Calculator

Diagnostic error codes for the Android calculator app play a crucial role in communicating specific errors or issues to users effectively. Here's how the error code system can be structured:

### 1. Error Code 001: Invalid Input

- Description: This error code indicates that the user has entered invalid input, such as non-numeric characters or unsupported symbols.

- Action: The user should double-check their input and ensure it consists of valid numeric values and supported symbols.

### 2. Error Code 002: Division by Zero

- Description: This error code occurs when the user attempts to divide a number by zero, which is mathematically undefined.

- Action: The user should revise their calculation and avoid dividing by zero to obtain a valid result.

### 3. Error Code 003: Overflow

- Description: This error code indicates that the result of the calculation exceeds the maximum supported value, resulting in an overflow.

- Action: The user should consider using a smaller input or splitting the calculation into multiple steps to avoid exceeding the maximum value.

#### 4. Error Code 004: Underflow

- Description: This error code occurs when the result of the calculation is smaller than the minimum supported value, resulting in an underflow.
- Action: The user should review their input and ensure it is within the supported range to obtain a valid result.

#### 5. Error Code 005: Unsupported Operation

- Description: This error code indicates that the user has attempted to perform an unsupported operation, such as taking the square root of a negative number.
- Action: The user should verify their calculation and ensure it adheres to the supported operations provided by the calculator app.

#### 6. Error Code 006: Calculation Timeout

- Description: This error code occurs when the calculation takes longer than the predefined timeout period, indicating a potential performance issue.
- Action: The user should consider simplifying their calculation or breaking it down into smaller steps to improve performance.

#### 7. Error Code 007: Memory Allocation Failure

- Description: This error code indicates that the calculator app encountered a memory allocation failure while performing the calculation.
- Action: The user should close any unnecessary apps running in the background to free up memory and retry the calculation.

#### 8. Error Code 008: Unknown Error

- Description: This error code is a generic placeholder for any unforeseen errors or issues encountered during the calculation process.
- Action: The user should try restarting the calculator app or contacting technical support for further assistance in resolving the issue.



## 10. Conclusion

In conclusion, the QA process for the Android calculator app serves a pivotal role in ensuring its reliability, accuracy, and user-friendliness. By conducting thorough testing and adhering to best practices, the QA team validates arithmetic operations, UI functionality, and detects potential bugs or issues. Through Test-Driven Development (TDD), developers and QA engineers collaboratively iterate on code implementation guided by failing tests, leading to improved code quality and reduced bugs.

The Test Strategy outlines objectives, assumptions, principles, and approaches for comprehensive testing, encompassing unit and integration testing to validate individual components and overall app functionality. Requirement Traceability Matrix ensures alignment between project requirements and test cases, while Risk Management strategies mitigate potential risks throughout the development lifecycle.

Diagnostic error codes enhance user experience by effectively communicating specific errors or issues, ensuring clarity and guidance in case of calculation errors or unexpected behaviors. Overall, the QA process, coupled with TDD principles and robust testing strategies, ensures the Android calculator app meets the expectations and requirements of its diverse user base, delivering a reliable, user-friendly, and high-quality application.

**THANK YOU**