# Dart Unit Testing

The key to high-quality software

# What is Unit Testing?

**Unit Testing** is a software testing technique where individual components or functions are tested in isolation.

# Benefits

- **Early Bug Detection**: Catch errors before they reach production
- **Code Quality**: Ensures functions work as expected
- **Documentation**: Tests serve as examples of how to use functions
- **Refactoring Safety**: Modify code with confidence
- **Regression Prevention**: Prevent old bugs from reappearing

# The Class We'll Test

```dart
class Arith {
    /// Adds two integers and returns the result.
    int add(int a, int b) { return a + b; }
    /// Subtracts the second integer from the first.
    int subtract(int a, int b) { return a - b; }
    /// Multiplies two integers and returns the result.
    int multiply(int a, int b) { return a * b; }
    /// Divides the first integer by the second.
    /// Throws an [ArgumentError] if the second integer is zero.
    double divide(int a, int b) {
        if (b == 0) {
            throw ArgumentError('Cannot divide by zero');
        }
        return a / b;
    }
}
```

# Setting Up Dart Testing

1. Add test dependency to `pubspec.yaml`:

```
dev_dependencies:
  test: ^1.24.0
```

2. Create test file structure:

```
project/
├── lib/
│   └── arith.dart
└── test/
    └── arith_test.dart
```

3. Import required packages:

```dart
import 'package:test/test.dart';
import '../lib/arith.dart';  // or correct path
```

# Basic Test Structure

## The AAA Pattern (Arrange-Act-Assert):

```
test(DESCRIPTION, Lambda-expression);
```

- DESCRIPTION contains the string that explains the nature of the test.
- Lambda expression is the code to run for testing.

```
test('should add two positive numbers correctly',
  () {
  // Arrange - Set up test data
  Arith arith = Arith();
  int a = 5;
  int b = 3;

  // Act - Execute the function being tested
  int result = arith.add(a, b);

  // Assert - Verify the result
  expect(result, equals(8));
});
```

**Remember**: Each test should be **independent** and test **one thing**!

# Expect and Assert

*Common Expect Matchers:*

```
// Equality checks
expect(result, equals(8));
expect(result, 8);  // shorthand

// Numeric comparisons
expect(result, greaterThan(5));
expect(result, lessThan(10));
expect(result, closeTo(3.33, 0.01));  // for doubles

// Type checks
expect(result, isA<int>());
expect(result, isNotNull);

// Boolean checks
expect(condition, isTrue);
expect(condition, isFalse);
```

# Group, setUp, and Test Organization

*Organizing Related Tests:*

```
void main() {
  group('Arith Class Tests', () {
    late Arith arith;
    setUp(() {
      // Runs before each test
      arith = Arith();
    });
    test('should add positive numbers', () {
      expect(arith.add(5, 3), equals(8));
    });
  });
}
```

```dart
void main() {
  group('Arith Class Tests', () {
    late Arith arith;

    setUp(() {
      // Runs before each test
      arith = Arith();
    });

    group('Addition Tests', () {
      test('should add positive numbers', () {
        expect(arith.add(5, 3), equals(8));
      });

      test('should add negative numbers', () {
        expect(arith.add(-5, -3), equals(-8));
      });
    });
  });
}
```

- We can make a group inside a group.

# Testing Edge Cases

*Important test scenarios to cover:*

```
group('Edge Cases', () {
  test('should handle zero values', () {
    expect(arith.add(5, 0), equals(5));
    expect(arith.multiply(7, 0), equals(0));
  });
  test('should handle large numbers', () {
    expect(arith.multiply(1000, 1000), equals(1000000));
  });
});
```

- **Rule**: Always test boundary conditions and unexpected inputs!

# Exception Testing

*Testing for Expected Errors:*

```
double divide(int a, int b) {
    if (b == 0) { throw ArgumentError('Cannot divide by zero');}
    return a / b;
}
```

```
group('Division Exception Tests', () {
  test('should throw ArgumentError when dividing by zero', () {
    // Act & Assert combined
    expect(
      () => arith.divide(10, 0),
      throwsA(isA<ArgumentError>()),
    );
  });
```

```
  test('should throw correct error message', () {
    expect(
      () => arith.divide(10, 0),
      throwsA(
        predicate((e) =>
          e is ArgumentError &&
          e.message == 'Cannot divide by zero'),
      ),
    );
  });
});
```

# Test Categories

1. **Happy Path Tests** - Normal expected usage

```
test('should add two positive numbers', () {
  expect(arith.add(5, 3), equals(8));
});
```

2. **Edge Case Tests** - Boundary conditions

```
test('should handle zero values', () {
  expect(arith.add(0, 0), equals(0));
});
```

3. **Error Case Tests** - Invalid inputs

```
test('should throw error for division by zero', () {
  expect(() => arith.divide(5, 0), throwsArgumentError);
});
```

# Integration Testing

## Testing Multiple Operations Together:

```
group('Integration Tests', () {
  test('should perform complex calculation correctly', () {
    // Test: (5 + 3) * 2 - 4 = 12
    int step1 = arith.add(5, 3);         // 8
    int step2 = arith.multiply(step1, 2); // 16
    int step3 = arith.subtract(step2, 4); // 12
    expect(step3, equals(12));
  });

  test('should handle calculation with division', () {
    // Test: (10 + 5) / 3 = 5.0
    int sum = arith.add(10, 5);
    double result = arith.divide(sum, 3);

    expect(result, equals(5.0));
  });
});
```

# Running Tests

## Command Line:

```
# Run all tests in the test directory
dart test

# We can specify a specific directory
dart test test-directory

# Run specific test file
dart test test/arith_test.dart

# Run with verbose output
dart test -r expanded
```