# GoogleTest Fixtures & Main – Deep Dive Q&A;

## Why should all fixture variables be in protected?

Fixture variables should be `protected` because:

- They must be accessible to the generated test classes (which inherit from your fixture).

- If they were `private`, tests wouldn’t compile because the generated derived class couldn’t access them.

- If they were `public`, they would expose internals unnecessarily to all code.

■ `protected` is the right balance: accessible to tests, hidden from external code.

## Why are SetUp and TearDown virtual?

They are declared `virtual` in `::testing::Test` so gtest can call your overridden versions polymorphically.

- At runtime, gtest only knows it has a `::testing::Test*`.

- Thanks to `virtual`, it will correctly call your derived `SetUp()` and `TearDown()` methods.

This ensures your fixture setup/teardown runs for every test.

## What does ::testing::Test mean?

- `::testing` is the gtest namespace.

- `Test` is the base fixture class provided by gtest.

- It defines empty `SetUp()` and `TearDown()` virtual methods that you can override.

- It also provides hooks like `SetUpTestSuite()` and `TearDownTestSuite()`.

## Why do we inherit from ::testing::Test?

We inherit from `::testing::Test` to integrate with the gtest framework.

- Gtest macros (`TEST_F`, etc.) generate new test classes that derive from your fixture.

- These test classes are also subclasses of `::testing::Test`, so gtest knows how to construct them, run `SetUp()`, run the test body, then `TearDown()`.

Without inheriting from `::testing::Test`, gtest wouldn’t know the lifecycle hooks to call.

## What happens under the hood with TEST_F?

Writing:

```
TEST_F(CalculatorTest, Add) {
```

EXPECT_EQ(calc->add(2,3), 5);

}

expands roughly to:

```
class CalculatorTest_Add_Test : public CalculatorTest {
void TestBody() override {
EXPECT_EQ(calc->add(2,3), 5);
}
};
```

Gtest execution flow:

1. Construct CalculatorTest_Add_Test
2. Call SetUp()
3. Call TestBody() (your test)
4. Call TearDown()
5. Destroy object

This works because CalculatorTest inherits from ::testing::Test.