**WEEK 2 MANDATORY HANDS-ON**

**Superset ID: 6365267**

**MODULE 4: UNIT TESTING**

**Q1: Differentiation between Unit Testing and Functional Testing**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Unit Testing** | **Functional Testing** |
| Focus | Smallest unit (method/class) | End-to-end system functionality |
| Scope | Narrow, internal logic | Broad, user-level flows |
| Dependencies | Mocked/Stubs used | Uses actual components or interfaces |
| Tool Examples | NUnit, xUnit | Selenium, Postman |
| Performed by | Developers | QA/Testers |

**Q2: Types of Testing**

* **Unit Testing** – Testing the smallest isolated code units (methods/classes)
* **Functional Testing** – Testing application features against requirements
* **Automated Testing** – Using scripts to perform repetitive tests (Unit/UI/API)
* **Performance Testing** – Measuring response time, load, and scalability

**Q3: Benefits of Automated Testing**

* Faster feedback on code changes
* Prevents regression bugs
* Enables CI/CD pipelines
* Increases confidence in refactoring

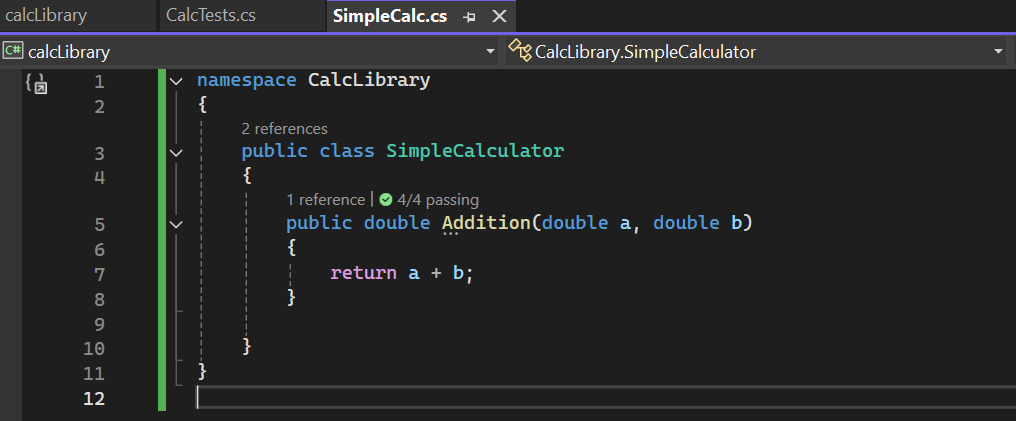
**Q4: Loosely Coupled & Testable Design**

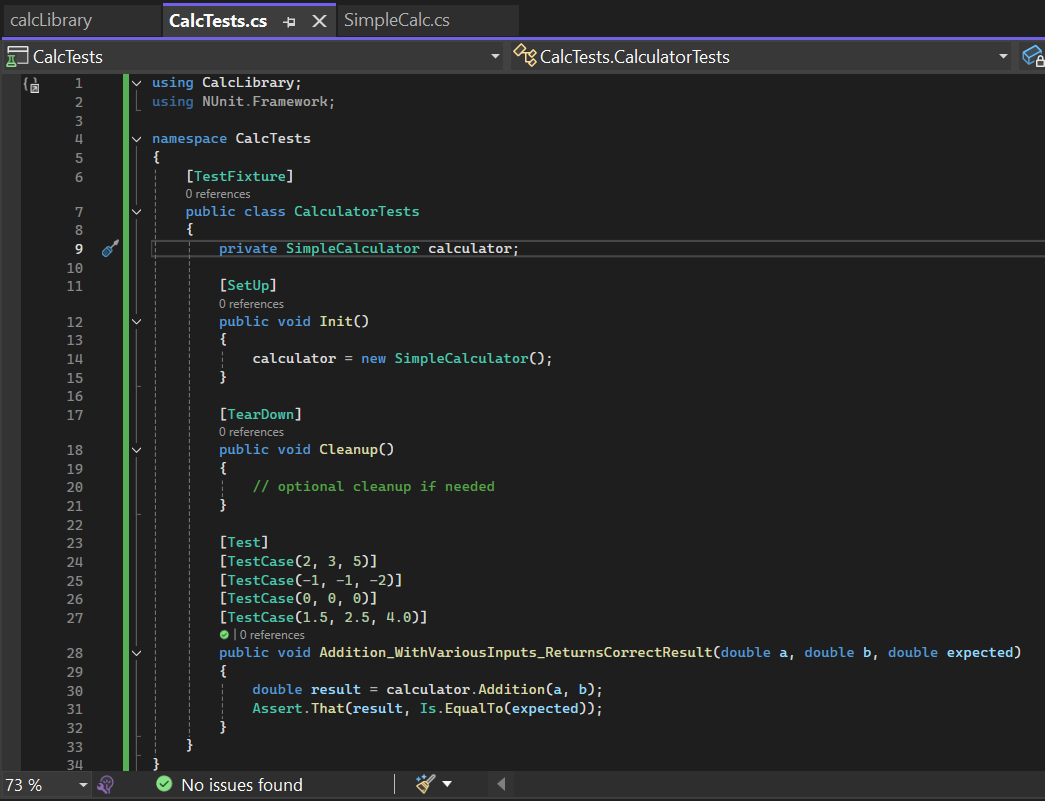
* A loosely coupled system avoids hard-coded dependencies.
* Use interfaces, dependency injection, and mocks.
* Makes your code testable, flexible, and reusable.

For instance,

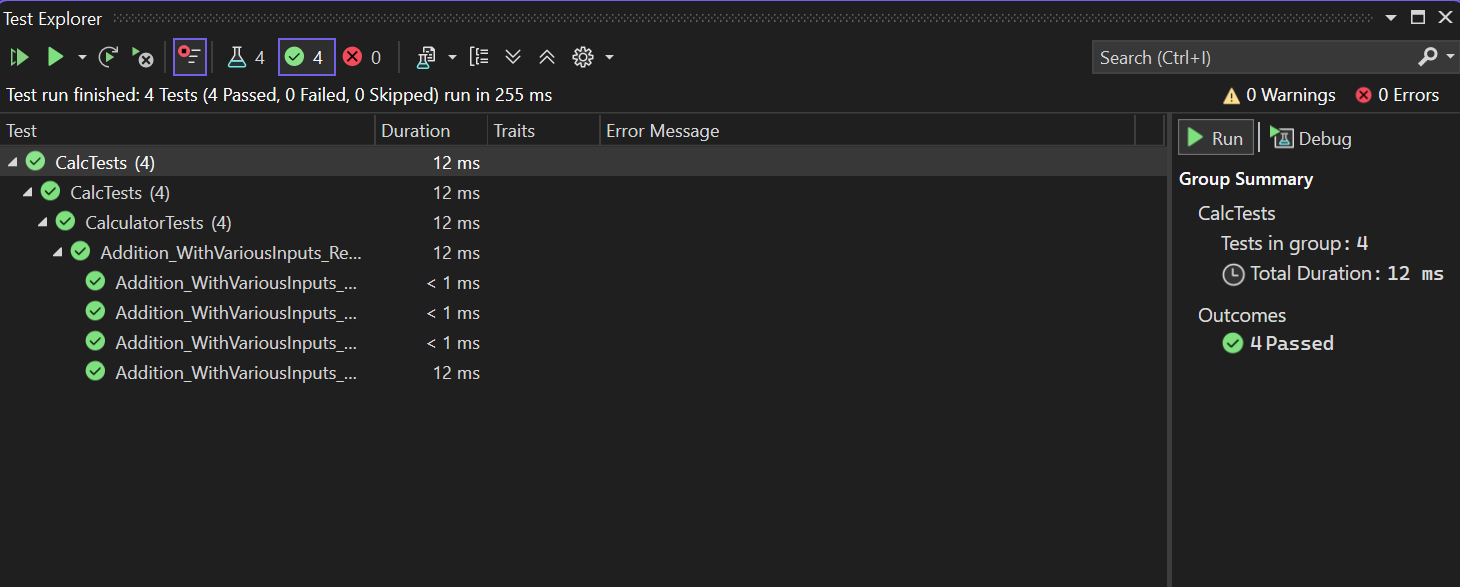
1. var db = new Database();  **// tightly coupled**
2. public CalculatorService(IDatabase db) {
   1. \_db = db; **// pass dependency externally (mockable)**
   2. }

**Unit testing code**

****

****

**Running tests:**

****