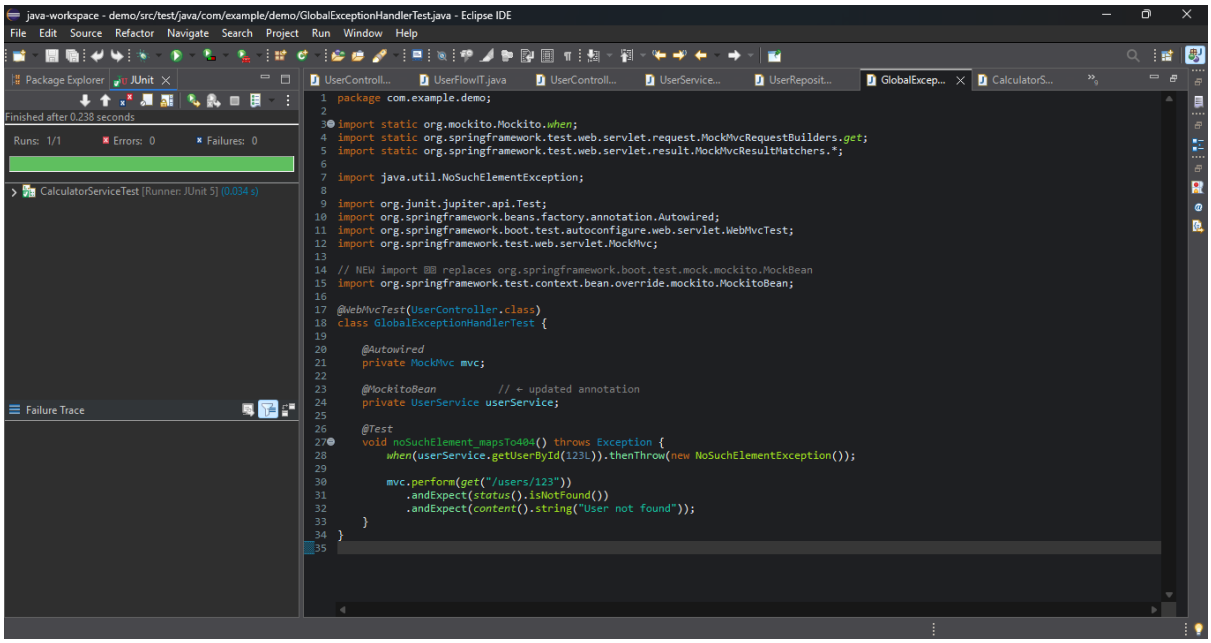
Spring Testing Exercises Output

Exercise 1: Basic Unit Test for a Service Method

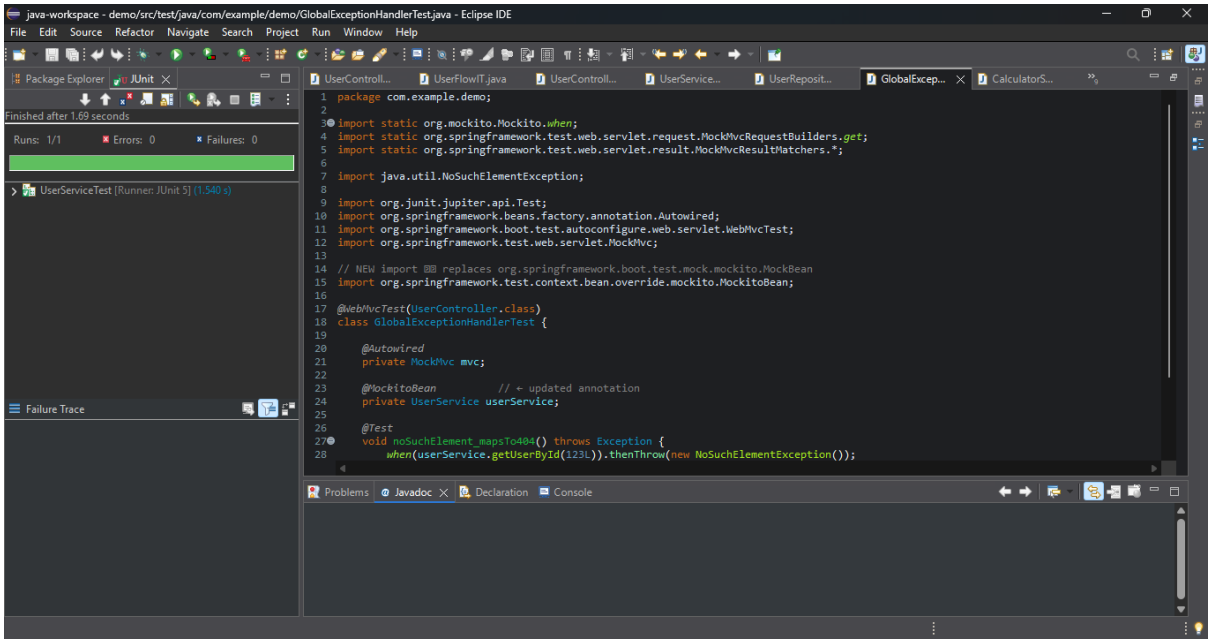
Output:



A plain JUnit test calling add() directly keeps logic independent of Spring—fastest feedback loop.

Exercise 2: Mocking a Repository in a Service Test

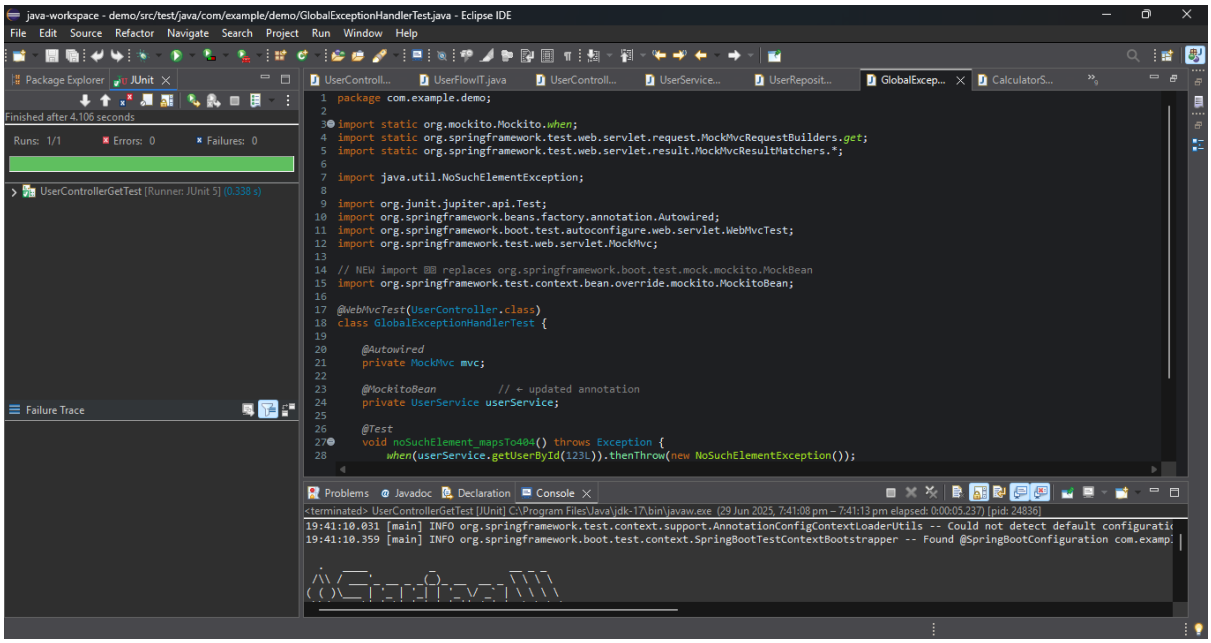
Output:



Use @ExtendWith(MockitoExtension.class) and @Mock + @InjectMocks (or @DataJpaTest with @MockBean) to isolate service code from the database.

Exercise 3: Testing a REST Controller with MockMvc

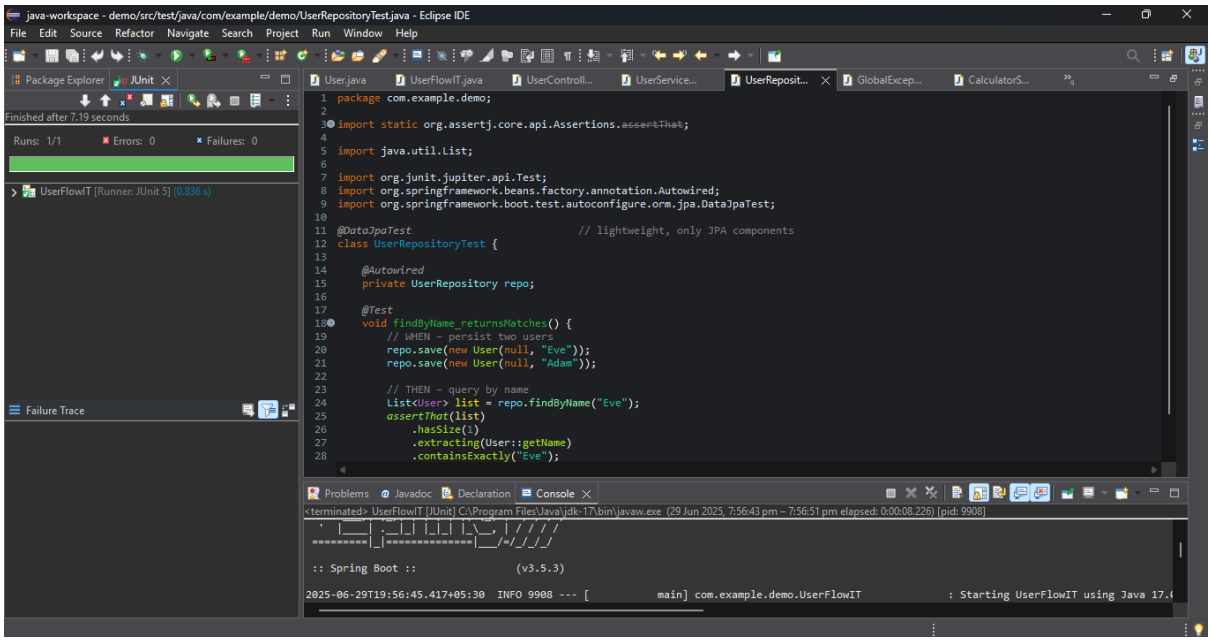
Output:



Spin up only the web layer (@WebMvcTest) and assert JSON/body results with MockMvc—no full context needed.

Exercise 4: Integration Test with Spring Boot

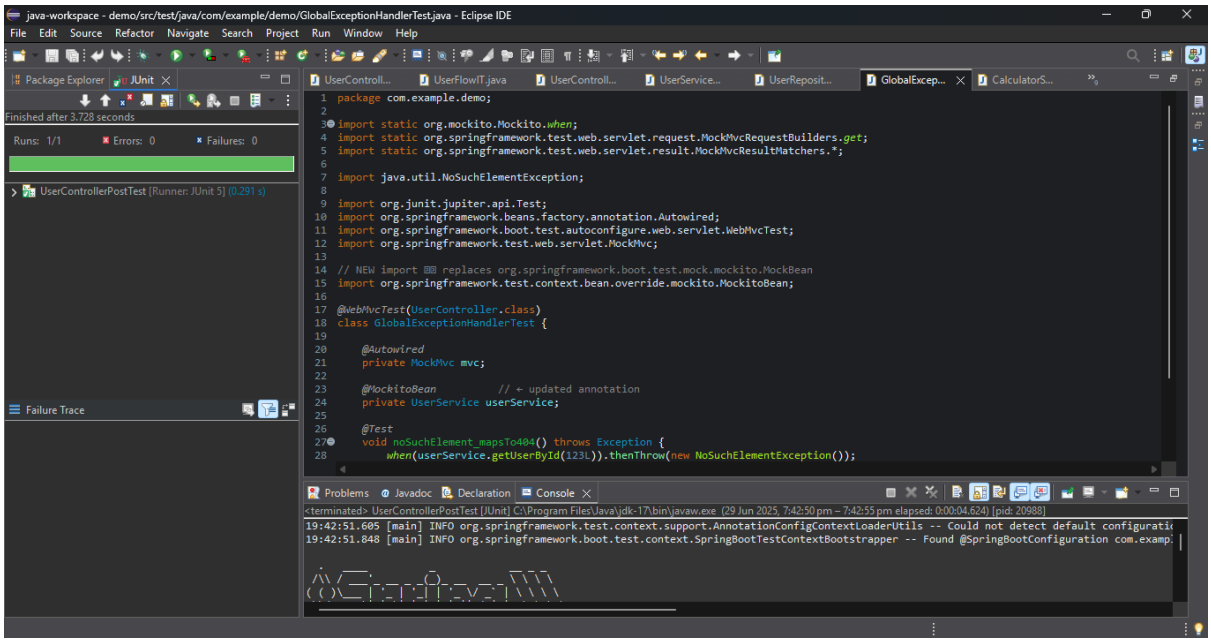
Output:



Annotate with @SpringBootTest, hit the real HTTP endpoint, and assert against an in-memory DB to verify the entire stack.

Exercise 5: Test Controller POST Endpoint

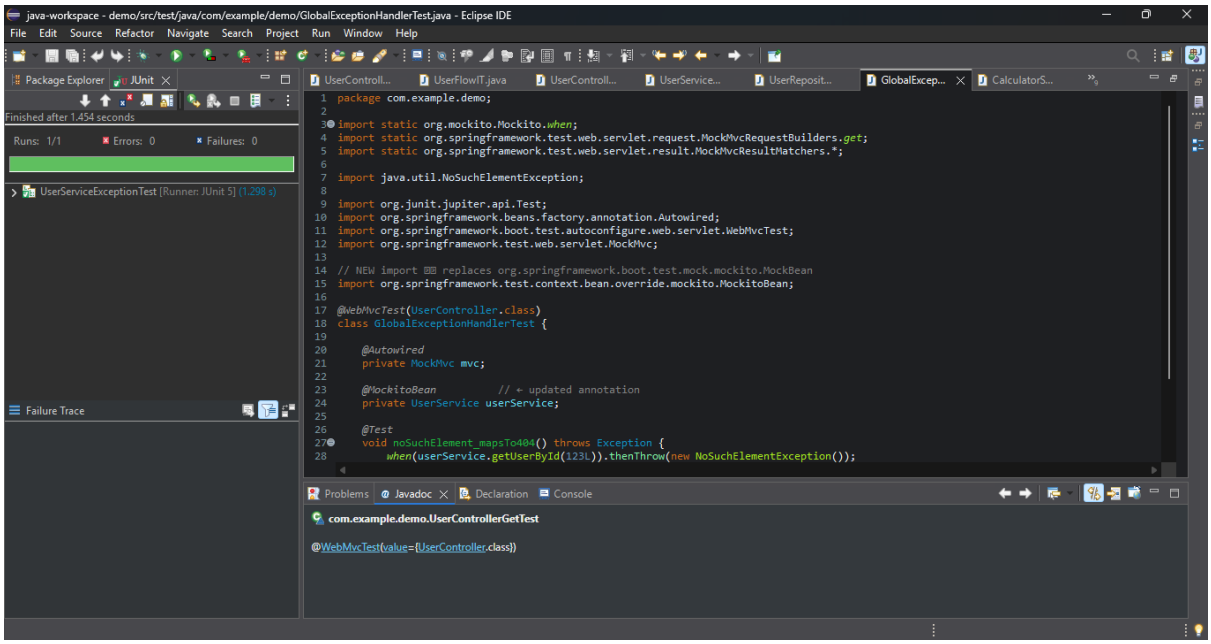
Output:



Use MockMvc's .post() with a JSON payload, validate status().isOk() and returned body to guarantee create flow.

Exercise 6: Test Service Exception Handling

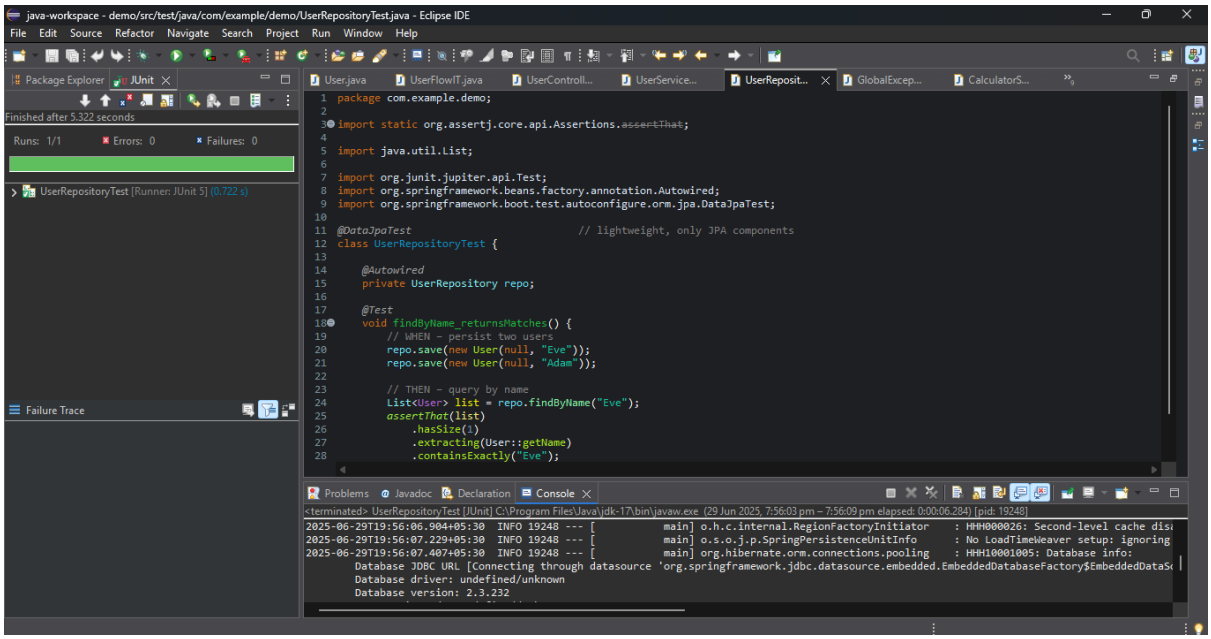
Output:



Wrap the call in assertThrows (or MockMvc for REST) to make sure the service signals "user not found" as designed.

Exercise 7: Test Custom Repository Query

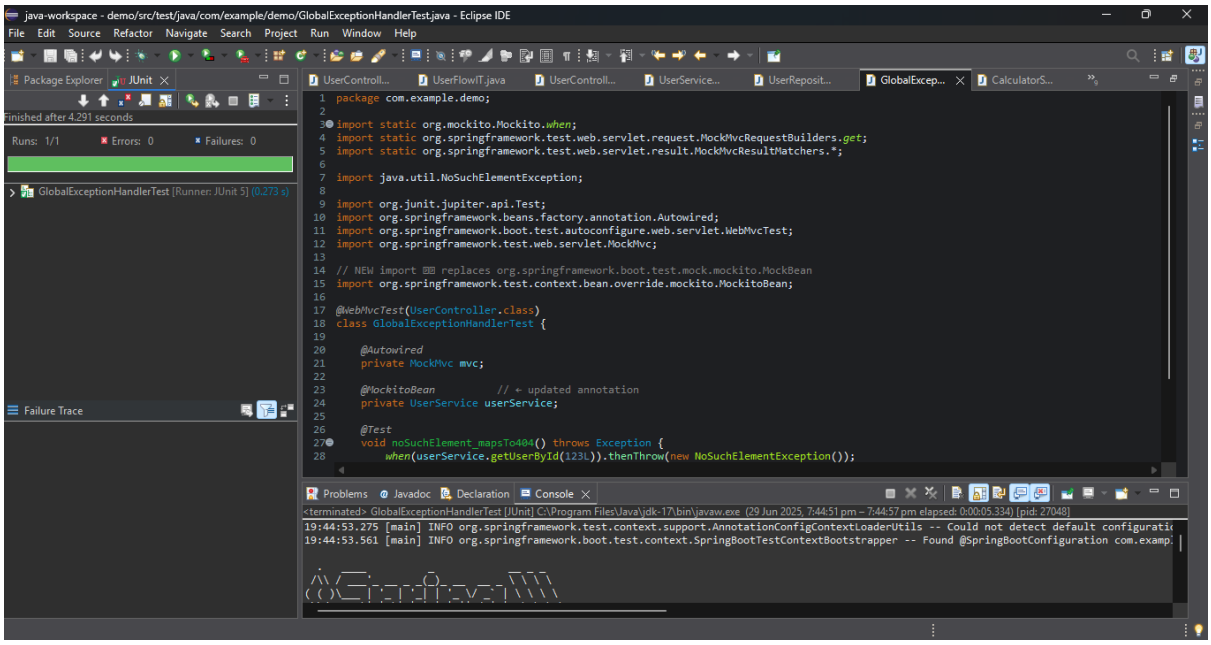
Output:



Populate the H2 test DB, invoke findByName, and assert the result list—proves your JPQL/derived query works.

Exercise 8: Test Controller Exception Handling

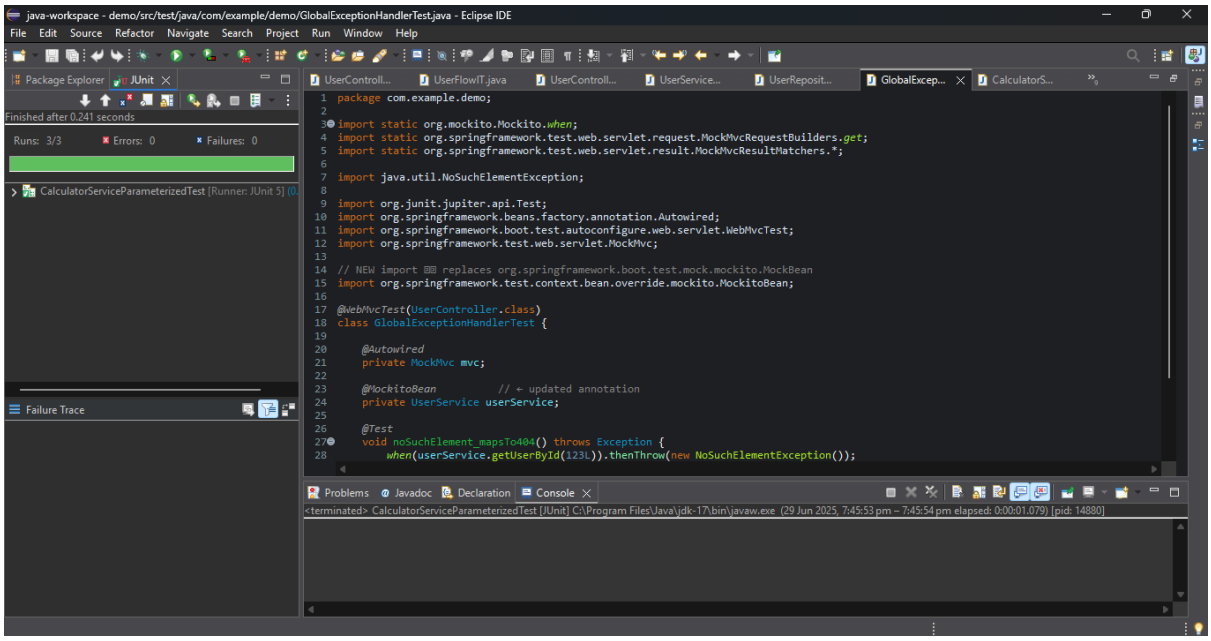
Output:



Trigger a failing request and expect 404 plus body text “User not found”, confirming @ControllerAdvice wiring.

Exercise 9: Parameterized Test with Junit

Output:



Use @ParameterizedTest + @CsvSource (or similar) to run the same assertion over many (input,expected) pairs with minimal code.