CodInClub

Powered By BridgeLabz

1️⃣ Basic JUnit Test: Testing a Calculator Class

Problem:

Create a `Calculator` class with methods `add(int a, int b)`, `subtract(int a, int b)`, `multiply(int a, int b)`, and `divide(int a, int b)`. Write JUnit test cases for each method.

👉 Bonus: Test for division by zero and handle exceptions properly.

2️⃣ Testing String Utility Methods

Problem:

Create a `StringUtils` class with the following methods:

- **`reverse(String str)`: Returns the reverse of a given string.**
- **`isPalindrome(String str)`: Returns `true` if the string is a palindrome.**
- **`toUpperCase(String str)`: Converts a string to uppercase.**

Write JUnit test cases to verify that these methods work correctly.

3️⃣ Testing List Operations

Problem:

Create a `ListManager` class that has the following methods:

- **`addElement(List<Integer> list, int element)`: Adds an element to a list.**
- **`removeElement(List<Integer> list, int element)`: Removes an element from a list.**
- **`getSize(List<Integer> list)`: Returns the size of the list.**

Write JUnit tests to verify that:

✅ **Elements are correctly added.**

✅ **Elements are correctly removed.**

✅ **The size of the list is updated correctly.**

4️⃣ Testing Exception Handling

Problem:

Create a method `divide(int a, int b)` that throws an `ArithmeticException` if `b` is zero. Write a JUnit test to verify that the exception is thrown properly.

5️⃣ Testing @BeforeEach and @AfterEach Annotations

Problem:

Create a class `DatabaseConnection` with a method `connect()` and `disconnect()`.

- **Use `@BeforeEach` to initialize a database connection before each test.**
- **Use `@AfterEach` to close the connection after each test.**

Write JUnit test cases to verify that the connection is established and closed correctly.

6️⃣ Testing Parameterized Tests

Problem:

Create a method `isEven(int number)` that returns `true` if a number is even.

- **Use `@ParameterizedTest` to test this method with multiple values like `2, 4, 6, 7, 9`.**

7️⃣ Performance Testing Using @Timeout

Problem:

Create a method `longRunningTask()` that sleeps for 3 seconds before returning a result.

- **Use `@Timeout(2)` in JUnit to fail the test if the method takes more than 2 seconds.**

8️⃣ Testing File Handling Methods

Problem:

Create a class `FileProcessor` with the following methods:

- **`writeToFile(String filename, String content)`: Writes content to a file.**
- **`readFromFile(String filename)`: Reads content from a file.**

Write JUnit tests to check if:

✅ The content is written and read correctly.

✅ The file exists after writing.

✅ Handling of `IOException` when the file does not exist.

📝 Advanced JUnit Practice Problems

1️⃣ Testing Banking Transactions

📌 Problem:

Create a `BankAccount` class with:

- **`deposit(double amount)`: Adds money to the balance.**

- `withdraw(double amount)`: Reduces balance.
- `getBalance()`: Returns the current balance.

✅ Write JUnit tests to check correct balance updates.
✅ Ensure withdrawals fail if funds are insufficient.

2️⃣ Testing Password Strength Validator

📌 Problem:

Create a `PasswordValidator` class with:

- Password must have at least 8 characters, one uppercase letter, and one digit.

✅ Write JUnit tests for valid and invalid passwords.

3️⃣ Testing Temperature Converter

📌 Problem:

Create a `TemperatureConverter` class with:

- `celsiusToFahrenheit(double celsius)`: Converts Celsius to Fahrenheit.
- `fahrenheitToCelsius(double fahrenheit)`: Converts Fahrenheit to Celsius.

✅ Write JUnit tests to validate conversions.

4️⃣ Testing Date Formatter

📌 Problem:

Create a `DateFormatter` class with:

- `formatDate(String inputDate)`: Converts `yyyy-MM-dd` format to `dd-MM-yyyy`.

✅ **Write JUnit test cases for valid and invalid dates.**

5️⃣ Testing User Registration

📌 **Problem:**

Create a `UserRegistration` class with:

- `registerUser(String username, String email, String password)`.
- **Throws `IllegalArgumentException` for invalid inputs.**

✅ **Write JUnit tests to verify valid and invalid user registrations.**