Example 1: Testing a Simple Addition Function

Step 1: Set up a new Maven project (if not already done)

1. Open IntelliJ IDEA and select **File > New > Project**.
2. Choose **Maven** as the project type and click **Next**.
3. Set a project name (e.g., `JUnit5Example`), and choose a location.
4. Finish the project setup.

Step 2: Add JUnit 5 dependencies

1. Open the `pom.xml` file in the root of the project.
2. Add the following dependencies inside the `<dependencies>` section:

```
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>5.7.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>5.7.0</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

3. IntelliJ should automatically import the dependencies. If not, right-click on the `pom.xml` and select **Maven > Reload Project**.

1. Calculator Class (Your existing class)

```
public class Calculator
{
    // Method to add two integers
    public int add(int a, int b)
    {
        return a + b;
         // Return the sum of a and b
    }
}
```

The `Calculator` class has a method `add(int a, int b)` that takes two integers, adds them, and returns the result.

2. Main Class to Implement the Calculator Class Methods

```
public class Main
{
    public static void main(String[] args)
  {
        // Step 1: Create an instance of the Calculator class
          Calculator calculator = new Calculator();
                // Create the calculator object

        // Step 2: Call the add method and store the result
          int sum = calculator.add(5, 3);
               // Add 5 and 3 using the add method

        // Step 3: Print the result
           System.out.println("The sum of 5 and 3 is: " + sum);
              // Output the result
    }
}
```

3. Detailed Comments in Code (Step-by-Step)

1. **Step 1: Create an instance of the `Calculator` class**
   - In the `main` method, we create an object `calculator` of the `Calculator` class. This object will allow us to call the methods of the `Calculator` class.
2. **Step 2: Call the `add` method**
   - Using the `calculator` object, we call the `add()` method, passing `5` and `3` as arguments. The result of the addition is stored in the variable `sum`.
3. **Step 3: Print the result**
   - Finally, we print the result using `System.out.println()`, which outputs the sum of 5 and 3 (8) to the console.

4. Object Diagram

This diagram illustrates the relationship between the objects in this scenario.

```
+----------------+       +----------------------+
|     Main       |       |    Calculator        |
+----------------+       +----------------------+
| - calculator   |------>| + add(int, int): int |
+----------------+       +----------------------+
                         |                      |
                         +----------------------+
```

- The `Main` class has a reference to a `Calculator` object (`calculator`).
- The `Calculator` class contains the method `add(int a, int b)`.

5. Algorithm (Step-by-Step)

Here is the algorithm for the above implementation:

1. **Start the Program**:
   - The `main` method is called to start the program.
2. **Create a Calculator Object**:
   - A `Calculator` object named `calculator` is instantiated. This object will be used to call methods from the `Calculator` class.
3. **Call the `add` Method**:
   - The `add()` method of the `calculator` object is called with the integers `5` and `3` as arguments.
   - The method adds these two numbers and returns the result, which is `8`.
4. **Store the Result**:
   - The result of the addition (`8`) is stored in the variable `sum`.
5. **Print the Result**:
   - The program prints the result to the console: `The sum of 5 and 3 is: 8`.

6. Execution Flow

- **Object Instantiation**: The `Calculator` object is created first. The constructor of the `Calculator` class is implicitly called.
- **Method Call**: The `add` method is called on the `calculator` object, performing the addition.
- **Result Display**: The result of the addition is displayed on the screen.

Step 4: Write a JUnit 5 test

1. In the `src/test/java` folder, create a test class `CalculatorTest.java`.

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class CalculatorTest {

    @Test
    public void testAddition() {
        Calculator calculator = new Calculator();
        int result = calculator.add(2, 3);
        assertEquals(5, result, "The addition result should be 5");
    }
}
```

Step 5: Run the test

1. Right-click on the `CalculatorTest.java` file and choose **Run 'CalculatorTest'**.
2. The test should pass, and you should see the result in the Run console.

Example 2: Testing String Manipulation (JUnit 5)

Step 1: Create a new Java class

1. In the `src/main/java` folder, create a class `StringManipulator.java`.

```
public class StringManipulator
{
    public String reverse(String input)
    {
        return new StringBuilder(input).reverse().toString();
    }
}
```

Step 2: Write a JUnit 5 test

1. In the `src/test/java` folder, create a test class `StringManipulatorTest.java`.

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class StringManipulatorTest
 {

    @Test
    public void testReverse()
   {
        StringManipulator manipulator = new StringManipulator();
        String result = manipulator.reverse("hello");
        assertEquals("olleh", result, "The reversed string should be
'olleh'");
    }
}
```

Step 3: Run the test

1. Right-click on the `StringManipulatorTest.java` file and choose **Run 'StringManipulatorTest'**.
2. The test should pass, and you will see the result in the Run console.

Additional Notes

- JUnit 5 uses annotations like `@Test`, `@BeforeEach`, `@AfterEach`, `@BeforeAll`, and `@AfterAll` to specify the test lifecycle.
- IntelliJ IDEA also supports running all tests at once using the **Run All Tests** option.
- Make sure your test methods are `public` and `void`.

By following these steps, you should be able to create and run JUnit 5 tests in IntelliJ IDEA easily!

Example 1: Testing Multiplication of Two Numbers

Step 1: Create the class to be tested

1. In the `src/main/java` folder, create a class `MathOperations.java`.

```java
public class MathOperations {
    public int multiply(int a, int b) {
        return a * b;
    }
}
```

Step 2: Create a JUnit test for multiplication

1. In the `src/test/java` folder, create a test class `MathOperationsTest.java`.

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class MathOperationsTest {

    @Test
    public void testMultiply() {
        MathOperations mathOps = new MathOperations();
        int result = mathOps.multiply(3, 4);
        assertEquals(12, result, "3 * 4 should be 12");
    }
}
```

Step 3: Run the test

1. Right-click on the `MathOperationsTest.java` file and choose **Run 'MathOperationsTest'**.
2. The test should pass, and you will see the result in the Run console.

Example 2: Testing Subtraction

Step 1: Create the class to be tested

1. In the `src/main/java` folder, create a class `MathOperations.java`.

```java
public class MathOperations {
    public int subtract(int a, int b) {
        return a - b;
    }
}
```

Step 2: Create a JUnit test for subtraction

1. In the `src/test/java` folder, create a test class `MathOperationsTest.java`.

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class MathOperationsTest {

    @Test
    public void testSubtract() {
        MathOperations mathOps = new MathOperations();
        int result = mathOps.subtract(10, 4);
        assertEquals(6, result, "10 - 4 should be 6");
    }
}
```

Step 3: Run the test

1. Right-click on the `MathOperationsTest.java` file and choose **Run 'MathOperationsTest'**.
2. The test should pass, and you will see the result in the Run console.

Example 3: Testing String Length

Step 1: Create the class to be tested

1. In the `src/main/java` folder, create a class `StringUtils.java`.

```
public class StringUtils {
    public int getLength(String str) {
        return str.length();
    }
}
```

Step 2: Create a JUnit test for string length

1. In the `src/test/java` folder, create a test class `StringUtilsTest.java`.

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class StringUtilsTest {

    @Test
    public void testGetLength() {
        StringUtils stringUtils = new StringUtils();
        int result = stringUtils.getLength("Hello");
        assertEquals(5, result, "The length of 'Hello' should be 5");
    }
}
```

Step 3: Run the test

1. Right-click on the `StringUtilsTest.java` file and choose **Run 'StringUtilsTest'**.
2. The test should pass, and you will see the result in the Run console.

Example 4: Testing Concatenation of Strings

Step 1: Create the class to be tested

1. In the `src/main/java` folder, create a class `StringUtils.java`.

```java
public class StringUtils {
    public String concatenate(String str1, String str2) {
        return str1 + str2;
    }
}
```

Step 2: Create a JUnit test for string concatenation

1. In the `src/test/java` folder, create a test class `StringUtilsTest.java`.

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class StringUtilsTest {

    @Test
    public void testConcatenate() {
        StringUtils stringUtils = new StringUtils();
        String result = stringUtils.concatenate("Hello", "World");
        assertEquals("HelloWorld", result, "The concatenation of 'Hello'
and 'World' should be 'HelloWorld'");
    }
}
```

Step 3: Run the test

1. Right-click on the `StringUtilsTest.java` file and choose **Run 'StringUtilsTest'**.
2. The test should pass, and you will see the result in the Run console.

Example 5: Testing the Absolute Value of a Number

Step 1: Create the class to be tested

1. In the `src/main/java` folder, create a class `MathOperations.java`.

```java
public class MathOperations {
    public int absoluteValue(int number) {
        return Math.abs(number);
    }
}
```

Step 2: Create a JUnit test for absolute value

1. In the `src/test/java` folder, create a test class `MathOperationsTest.java`.

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class MathOperationsTest {

    @Test
    public void testAbsoluteValue() {
        MathOperations mathOps = new MathOperations();
        int result = mathOps.absoluteValue(-5);
        assertEquals(5, result, "The absolute value of -5 should be 5");
    }
}
```

Step 3: Run the test

1. Right-click on the `MathOperationsTest.java` file and choose **Run 'MathOperationsTest'**.
2. The test should pass, and you will see the result in the Run console.

Recap of Steps for Each Example:

1. **Create the class**: Create a class containing the method you want to test.
2. **Write the test**: Create a test class that contains test methods using the `assertEquals` method.
3. **Run the test**: Right-click on the test class and choose "Run" to execute the test.

Notes:

- `assertEquals(expected, actual)` is used to compare the expected value with the actual result.
- Make sure the test methods are annotated with `@Test` and are `public` and `void`.
- You can run individual tests or all tests in your test class.