**JUNIT**

* **JUnit** is an open-source **unit testing framework** for the Java programming language.
* It is used to write and run repeatable test cases to ensure that your code works as expected.
* **Unit Testing** means testing ***small parts (units)*** of your code—usually one method at a time—to make sure it works correctly.

**e.g.**

Imagine you’re making a calculator. Instead of waiting to check the whole calculator at the end, you first test:

* ***Does addition work?***
* ***Does subtraction work?***
* ***What happens if you divide by zero?***

Each of these small checks is a **unit test.**

* A **unit** is the smallest part of your program, that can be tested on its own.

Usually:

* A **method** in a class is considered a unit.
* We test the logic of that method, **without running the full application**.
* Uses annotations like **@Test, @BeforeEach, @AfterEach**, etc.
* **Annotations** are set of instructions, used to inform JVM to perform some tasks (**process/ generate/ validate the code**).
* They act as a **metadata** added to code to *give extra information to compiler or runtime,* efficient for reducing boilerplate code.**(Lombok annotation)**

🌟 **Advantages of Junit**

| **Advantage** | **Description** |
| --- | --- |
| **Fast Feedback** | Quick execution of tests ensures faster feedback on code changes |
| **Early Bug Detection** | Helps catch bugs early in the development cycle |
|  |  |
| **Readable and Maintainable** | Clear and structured test code with annotations |
|  |  |
| **Open Source** | Free and widely supported by the community |

* **Steps to include JUNIT API**

**Step 1:**  Go to mvn repository and search for **JUNIT Jupitor**

**Step 2:** Select **Junit Jupiter API** , and select appropriate version

**Step 3:** Copy the **dependency tag,**  and paste it inside the <dependencies> tags in **pom.xml**

* ***Writing the test cases:-***
* **Source: 📦 src/test/java**
* The **src/test/java directory** is used to write and organize your *test cases* (e.g., JUnit test classes) separately from your application code, which lives in **src/main/java.**
* **Junit test classes( testing implementations)** will be present here.
* This ensures:
* **Clean separation** of production and test code
* **Automatic detection** of test files by build tools (Maven, Gradle)
* **Maintainability** and better project structure
* ***Steps to create a JUnit Test Class:***

**Step 1:** Right click on the class containing *application code* present in **src/main/java,**  and create new **JUnit test case.**

new🡪other🡪Junit test case.

**Step 2:** Write logics to perform unit testing in methods.

e.g.:  
*//@Test is used to signal that the annotated method is a testmethod.*

***@Test***

void **testAdd**() {

double a=10,b=20,

expectedSum=a+b;

*assertEquals*(expectedSum, calculator.add(a,b));

}

**Assertion Statements**

* We use assert statements to check if the output of your code matches the expected result.
* To validate whether a piece of code behaves as expected during a test.

**📦 Why Use assert?**

* To compare **expected vs actual** results
* To confirm whether a method **passes or fails** a test case
* To **automate** testing – no need to manually check outputs

**🔧 Commonly Used Assert Methods in Junit**

| **Method** | **Purpose** |
| --- | --- |
| **assertEquals**(expected, actual) | Passes if both values are equal |
| **assertNotEquals**(expected, actual) | Passes if values are different |
| **assertTrue**(condition) | Passes if the condition is true |
| **assertFalse(**condition) | Passes if the condition is false |
| **assertNull**(object) | Passes if object is null |
| **assertNotNull**(object) | Passes if object is not null |
| **assertThrows**(Exception.class, () -> code) | Passes if the code throws the expected exception |

**e.g.:-**

***@Test***

void testDiv() {

double a=10,b=0,

expectedDiv=a/b;

*assertEquals*(expectedDiv, calculator.div(a,b));

//not give ArithmeticException

//bcz double/double is giving Infinity

}

***@Test***

void testDivInt() {

int a=10,b=0;

//Assert that execution of the supplied executable throws an exception of the expectedType and return the exception.

*assertThrows*(ArithmeticException.class, ()->{

calculator.divInt(a,b);

});

}

⚠️ **Disadvantages of JUnit**

| **Disadvantage** | **Description** |
| --- | --- |
| **Only for Java** | Can't be used with other programming languages |
| **Limited UI Testing** | Not suitable for GUI or front-end tests; needs additional tools |