Objective

To gain proficiency in setting up a Java project using Gradle from the terminal, implementing unit tests using JUnit 5 assertions, and following the TDD (Test-Driven Development) workflow.

Phase 1: The Terminal Powerhouse (Gradle Setup)

NOTE: Use the terminal or your IDE of choice. **If using IDE skip Phase 1, since your IDE likely creates the Test setup automatically.**

Goal: Initialize a project without an IDE to understand the underlying build structure.

1. **Open your Terminal** (Command Prompt, PowerShell, or Terminal).

Create a workspace:
mkdir LehmanJUnitLab
cd LehmanJUnitLab

2. **Initialize Gradle:**
   gradle init

   Select `2: application`

   Select `1: Java`

   Select `no` for multiple subprojects

   Select `1: Groovy` for build script DSL

   Select `4: JUnit Jupiter` (This is JUnit 5!)

3. **Verify the setup:** Run `./gradlew test` (or `gradlew test` on Windows). It should pass with the default boilerplate code.

Phase 2: The Red-Green-Refactor Cycle (TDD)

Goal: Practice the TDD philosophy: *Design the test before the implementation.*

Task: We need a class called `LehmanGradeBook` that calculates if a student passes based on a numeric grade.

Step 1 (RED): Navigate to `app/src/test/java/.../AppTest.java`. Delete the boilerplate and add this test:
@Test

```
@DisplayName("Grade 70 should return true for passing")
void testPassingGrade() {
    LehmanGradeBook gb = new LehmanGradeBook();
    assertTrue(gb.isPassing(70), "A grade of 70 should pass.");
}
```

1. *Try to run `./gradlew test`. It will fail to compile because `LehmanGradeBook` doesn't exist yet.*

Step 2 (GREEN): Create `LehmanGradeBook.java` in `app/src/main/java/.../` and implement the minimum:

```
public class LehmanGradeBook {
    public boolean isPassing(int grade) {
        return grade >= 70;
    }
}
```

2. *Run `./gradlew test` again. You should see "BUILD SUCCESSFUL".*
3. **Step 3 (REFACTOR):** Look at your code. Is there any redundancy? Can the logic be cleaner? (In this simple case, maybe not, but always check!)

Phase 3: Assertions & Edge Cases

Goal: Use different JUnit 5 assertions to verify complex behavior.

Task: Add a method `char getLetterGrade(int score)` to your class.

1. **Write tests for multiple outcomes:**
    - Use `assertEquals('A', gb.getLetterGrade(95))`
    - Use `assertEquals('F', gb.getLetterGrade(50))`
2. **Boundary Testing:** Write a test for exactly `90`, `80`, and `70`.

Phase 4: Testing for Exceptions

Goal: Ensure your code handles "bad" data properly.

Task: If a user enters a grade over 100 or below 0, the code should throw an `IllegalArgumentException`.

Write the test first (TDD):

```
@Test
void testInvalidGradeThrowsException() {
    LehmanGradeBook gb = new LehmanGradeBook();
    assertThrows(IllegalArgumentException.class, () -> {
```

```
        gb.isPassing(150);
    });
}
```

1. **Update your implementation** to throw the exception and make the test pass.

Deliverables

- Ensure all tests pass via `./gradlew test`.