**Lab Exercise - Boundary Value Analysis using JUnit**

**Objective:**

To implement **Boundary Value Analysis (BVA)** using **JUnit** in Java. We will test a function that validates an **age input (1-100)**, ensuring it falls within a valid range.

**Scenario:**

We are testing an **age validation function** that accepts an integer input between **1 and 100**. The function returns:

* "Valid" if the age is **between 1 and 100 (inclusive)**
* "Invalid" if the age is **less than 1 or greater than 100**

**Boundary Values for Testing:**

* **Lower Boundary:** 0, 1, 2
* **Upper Boundary:** 99, 100, 101

**Steps to Perform:**

1. Create a **Java class** with an isValidAge(int age) function.
2. Use **JUnit** to test boundary values.
3. Validate results using assertions.

**Implementation**

**Step 1: Implement the Age Validation Function**

Create a Java class **AgeValidator.java**:

public class AgeValidator {

public static String isValidAge(int age) {

if (age >= 1 && age <= 100) {

return "Valid";

} else {

return "Invalid";

}

}

}

**Step 2: Create a JUnit Test Class**

Create a test class **AgeValidatorTest.java**:

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class AgeValidatorTest {

@Test

public void testLowerBoundary() {

assertEquals("Invalid", AgeValidator.isValidAge(0)); // Below valid range

assertEquals("Valid", AgeValidator.isValidAge(1)); // Lower boundary

assertEquals("Valid", AgeValidator.isValidAge(2)); // Just above lower boundary

}

@Test

public void testUpperBoundary() {

assertEquals("Valid", AgeValidator.isValidAge(99)); // Just below upper boundary

assertEquals("Valid", AgeValidator.isValidAge(100)); // Upper boundary

assertEquals("Invalid", AgeValidator.isValidAge(101)); // Above valid range

}

}

**Step 3: Run the JUnit Tests**

* Use **JUnit 4 or JUnit 5** in an IDE like **Eclipse, IntelliJ, or VS Code**.
* Run the test cases.
* Ensure all assertions pass.

**Expected Output (JUnit Test Results)**

✔ **testLowerBoundary()** → **Passed**  
✔ **testUpperBoundary()** → **Passed**

**Key Takeaways**

✅ **Boundary Value Analysis (BVA)** helps test extreme values near limits.  
✅ **JUnit Assertions** validate expected vs actual output.  
✅ Helps in **finding edge-case bugs** in input validation.

**Hands-On-2**

**Objective:**

To implement **Boundary Value Analysis (BVA)** using **JUnit** in Java by testing a function that **categorizes exam scores** into different grade levels.

**Scenario:**

We will use the same grading system as before, where scores determine the grade category:

* "Fail" → **0-39**
* "Pass" → **40-59**
* "Merit" → **60-79**
* "Distinction" → **80-100**
* "Invalid" → **Less than 0 or greater than 100**

**Boundary Values for Testing:**

| **Partition** | **Lower Boundary** | **Upper Boundary** | **Expected Output** |
| --- | --- | --- | --- |
| Below Range | -1 | 0 | "Invalid", "Fail" |
| Fail | 0 | 39, 40 | "Fail", "Pass" |
| Pass | 40 | 59, 60 | "Pass", "Merit" |
| Merit | 60 | 79, 80 | "Merit", "Distinction" |
| Distinction | 80 | 100, 101 | "Distinction", "Invalid" |
| Above Range | 101 | 110 | "Invalid" |

**Steps to Perform:**

1. Modify the **JUnit test class** to cover **boundary values**.
2. Add **test cases for each boundary**.
3. Run the tests to verify the implementation.

**Implementation**

**Step 1: Implement the Grading Function**

The **GradingSystem.java** class remains unchanged:

public class GradingSystem {

public static String getGrade(int score) {

if (score < 0 || score > 100) {

return "Invalid";

} else if (score <= 39) {

return "Fail";

} else if (score <= 59) {

return "Pass";

} else if (score <= 79) {

return "Merit";

} else {

return "Distinction";

}

}

}

**Step 2: Create a JUnit Test Class for Boundary Value Analysis**

Create a new test class **GradingSystemBVA.java**:

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class GradingSystemBVA {

@Test

public void testBelowRange() {

assertEquals("Invalid", GradingSystem.getGrade(-1));

assertEquals("Fail", GradingSystem.getGrade(0));

}

@Test

public void testFailBoundary() {

assertEquals("Fail", GradingSystem.getGrade(1));

assertEquals("Fail", GradingSystem.getGrade(38));

assertEquals("Fail", GradingSystem.getGrade(39));

assertEquals("Pass", GradingSystem.getGrade(40)); // Transition to "Pass"

}

@Test

public void testPassBoundary() {

assertEquals("Pass", GradingSystem.getGrade(41));

assertEquals("Pass", GradingSystem.getGrade(58));

assertEquals("Pass", GradingSystem.getGrade(59));

assertEquals("Merit", GradingSystem.getGrade(60)); // Transition to "Merit"

}

@Test

public void testMeritBoundary() {

assertEquals("Merit", GradingSystem.getGrade(61));

assertEquals("Merit", GradingSystem.getGrade(78));

assertEquals("Merit", GradingSystem.getGrade(79));

assertEquals("Distinction", GradingSystem.getGrade(80)); // Transition to "Distinction"

}

@Test

public void testDistinctionBoundary() {

assertEquals("Distinction", GradingSystem.getGrade(81));

assertEquals("Distinction", GradingSystem.getGrade(99));

assertEquals("Distinction", GradingSystem.getGrade(100));

assertEquals("Invalid", GradingSystem.getGrade(101)); // Out of range

}

@Test

public void testAboveRange() {

assertEquals("Invalid", GradingSystem.getGrade(110));

}

}

**Step 3: Run the JUnit Tests**

* Use **JUnit 4 or JUnit 5** in **Eclipse, IntelliJ, or VS Code**.
* Run the test cases.
* Ensure all assertions pass.

**Expected Output (JUnit Test Results)**

✔ **testBelowRange()** → **Passed**  
✔ **testFailBoundary()** → **Passed**  
✔ **testPassBoundary()** → **Passed**  
✔ **testMeritBoundary()** → **Passed**  
✔ **testDistinctionBoundary()** → **Passed**  
✔ **testAboveRange()** → **Passed**

**Key Takeaways**

✅ **Boundary Value Analysis (BVA)** tests **edge cases** to ensure correct transitions between partitions.  
✅ **JUnit assertions** help validate expected vs actual outputs.  
✅ Combined with **Equivalence Partitioning (EP)**, this ensures **comprehensive test coverage**.