# Faculty of Computing

**SE-314: Software Construction**

**Class: BESE 13AB**

# Lab 02: Code Coverage

# CLO-02: Apply patterns, frameworks, and techniques for Software Construction. CLO-04: Use modern tools such as Eclipse, NetBeans etc. for software construction.

**Date: 16th Sep 2024**

**Time: 10:00 AM** **- 12:50 PM   
 02:00 PM – 04:50 PM**

|  |  |
| --- | --- |
| NAME | MUHAMMAD SAAD UMER |
| CLASS | BESE 13-A |
| CMS | 408485 |

# Instructor: Dr. Mehvish Rashid

# Lab Engineer: Mr. Aftab Farooq

### 

**Introduction:**

# Lab 02: Code Coverage

Students will have hands-on experience of coverage and unit testing tools such as EclEmma and JUnit.

### **Lab Objectives:**

The objective of this lab is to get a practical understanding and knowledge of the unit testing and coverage tools. After the completion of this lab, students will be able to apply unit testing tools.

### **Helping Material:**

https://ocw.mit.edu/ans7870/6/6.005/s16/getting-started/ https://github.com/junit-team/junit4/wiki/Download-and-Install <https://github.com/junit-team/junit4/wiki/Getting-started>

<https://www.codejava.net/testing/junit-tutorial-for-beginner-with-eclipse>

**Lab Tasks**

### **Task 1:**

Then create a new Java class called Hailstone.java containing this code:

public class Hailstone {

public static void main(String[] args) {

int n = 3;

while (n != 1) {

if (n % 2 == 0) {

n = n / 2;  
}

else {

n = 3 \* n + 1;  
}

}

}  
}

Run this class with EclEmma code coverage highlighting turned on, by choosing Run / Coverage As / Java Application.

By changing the initial value of n, you can observe how EclEmma highlights different lines of code differently.

When n=3 initially, what color is the line n = n/2 after execution? When n=16 initially, what color is the line n = 3 \* n + 1 after execution?

What initial value of n would make the line while (n != 1) yellow after execution?

**Task 2:**

For the below Vowels class, construct JUnit tests considering the following Testing strategy:

//

// Testing strategy:

//

//

// Partition for vowelsIn(letters) -> result:

//

// letters.length: 0, 1, >1

// result.length: 0, 1, >1

// letters contains repeated vowels or doesn't

//

// covers letters.length=0, result.length=0, no repeated vowels

public class Vowels {

/\*\*

* @param letters a string of lowercase English letters
* @return the set of vowels found in letters

\*/

public static String getVowelsIn(String letters) { final String VOWELS = "aeiou";

String vowelsFound = "";

for (int i = 0; i < VOWELS.length(); ++i) { char vowel = VOWELS.charAt(i);

if (letters.indexOf(vowel) != -1) { vowelsFound += vowel;

}

}

return vowelsFound;

}

}

**Task 3: Improve Code Coverage for Hailstone.java and Vowels.java**

**Objective:** The objective of this task is to ensure that all possible execution paths (branches) of the code are covered by tests. This will allow you to get hands-on experience with achieving high code coverage and understanding how different tests impact coverage.

**Step 1: Achieving 100% Code Coverage for Hailstone.java**

1. **Analyze Code Coverage**: After running the initial version of Hailstone.java with EclEmma, you will notice some lines may not be fully covered. For instance, the conditions inside the if-else statements (such as n = n / 2 and n = 3 \* n + 1) may be partially covered, depending on the initial value of n.
2. **Task**: Modify the Hailstone.java class so that you cover all possible branches of the code. Write a few test cases with different initial values of n to ensure all branches (both the if and else conditions) are executed at least once.
   * Try different starting values of n (e.g., 2, 3, 5, 16) to observe the coverage in different cases.
   * Aim to achieve 100% line and branch coverage using EclEmma.
3. **Question**: After running the tests, identify the value of n that ensures all branches in the if-else block are fully covered (marked green in EclEmma). What combination of initial values is required to cover all conditions?  
     
   **Step 2: Achieving High Code Coverage for Vowels.java**
4. **Analyze Code Coverage**: The Vowels.java class might not achieve full code coverage using a few basic test cases. For example, cases where the string contains repeated vowels or no vowels at all may not be covered.
5. **Task**: Add additional JUnit tests to increase the coverage of Vowels.java. Create test cases for the following conditions:

- Empty string (already suggested).

- A string with no vowels (e.g., "bcdfg").

- A string containing all vowels.

- A string with repeated vowels (e.g., "aaaaaaa").

- A string with a mix of vowels and consonants (e.g., "education").

1. **Question**: After running the tests with EclEmma, which cases helped improve the code coverage for the getVowelsIn() method? What additional edge cases did you cover to achieve higher coverage?

### **Answer:**

|  |
| --- |
| **Solution** |
| **Task1: Provide screen shots of attained coverage by Ecl Emma tool for each question.**  **1. When n = 3: (Green)**    **2. When n = 16 (Red)**    **3. n = 1, would make the color yellow:**    **Task2: Provide JUnit Test File containing test cases according to given Test Strategy.**  **package lab\_2;**  **import static org.junit.jupiter.api.Assertions.assertEquals;**  **import org.junit.jupiter.api.Test;**  **public class VowelsTest {**  **// covers letters.length = 0, result.length = 0, no repeated vowels**  **@Test**  **public void testEmptyString() {**  **String letters = "";**  **String result = Vowels.getVowelsIn(letters);**  **assertEquals("", result, "Empty string should return an empty result.");**  **}**  **// covers letters.length = 1, result.length = 0**  **@Test**  **public void testSingleConsonant() {**  **String letters = "b";**  **String result = Vowels.getVowelsIn(letters);**  **assertEquals("", result, "Single consonant should return an empty result.");**  **}**  **// covers letters.length = 1, result.length = 1, no repeated vowels**  **@Test**  **public void testSingleVowel() {**  **String letters = "a";**  **String result = Vowels.getVowelsIn(letters);**  **assertEquals("a", result, "Single vowel should return that vowel.");**  **}**  **// covers letters.length > 1, result.length = 0, no repeated vowels**  **@Test**  **public void testNoVowels() {**  **String letters = "bcdfg";**  **String result = Vowels.getVowelsIn(letters);**  **assertEquals("", result, "String with no vowels should return an empty result.");**  **}**  **// covers letters.length > 1, result.length = 1, no repeated vowels**  **@Test**  **public void testOneVowel() {**  **String letters = "hello";**  **String result = Vowels.getVowelsIn(letters);**  **assertEquals("e", result, "String with one vowel should return that vowel.");**  **}**  **// covers letters.length > 1, result.length > 1, no repeated vowels**  **@Test**  **public void testMultipleVowels() {**  **String letters = "education";**  **String result = Vowels.getVowelsIn(letters);**  **assertEquals("aeiou", result, "String with multiple vowels should return all vowels in order.");**  **}**  **// covers letters.length > 1, result.length = 1, contains repeated vowels**  **@Test**  **public void testRepeatedVowel() {**  **String letters = "banana";**  **String result = Vowels.getVowelsIn(letters);**  **assertEquals("a", result, "String with repeated vowel should return that vowel only once.");**  **}**  **// covers letters.length > 1, result.length > 1, contains repeated vowels**  **@Test**  **public void testMixedRepeatedVowels() {**  **String letters = "aeiouaeiou";**  **String result = Vowels.getVowelsIn(letters);**  **assertEquals("aeiou", result, "String with all vowels repeated should return each vowel once.");**  **}**  **}**    **Task 3: Improve Code Coverage for Hailstone.java and Vowels.java**  **Step#1 (Hailstone)**    **Value of n = 3**    **Changes:**   * **Static methods are made non-static class methods. i.e. generateSequence()** * **Called by the separate main file.** * **Object construction**   **Step#2 (Vowels)**      **Changes:**   * **Static methods are made non-static class methods.** * **Called by the test file.** * **Object Construction**   **Test Cases (Including Additional Edge Cases):**   **Test an empty string**: Ensure that an empty string returns an empty result.   **Test a single consonant**: A string with one consonant should return an empty result.   **Test a single vowel**: A string with one vowel should return that vowel.   **Test a string with no vowels**: A string containing only consonants should return an empty result.   **Test a string with one vowel**: A string with a single vowel among consonants should return that vowel.   **Test a string with multiple vowels**: A string with multiple distinct vowels should return all the vowels in order.   **Test a string with repeated vowels**: A string with a repeated vowel should return that vowel only once.   **Test a string with mixed repeated vowels**: A string containing all vowels, each repeated, should return each vowel only once. |

### **Deliverables:**

Compile a single word document by filling in the solution part and submit this Word file on LMS. In case of any problems with submissions on LMS, submit your Lab assignments by emailing it to [aftab.farooq@seecs.edu.pk](mailto:aftab.farooq@seecs.edu.pk).