**1. Second Code Segment**

**. Error:** The semicolon after while (i <= 10); creates an empty loop body, causing an infinite loop at i = 1. The ++i is outside the loop and never executes.

**. Correction:** Remove the semicolon to include ++i in the loop body.

**2. Second Code Segment**

**Error:** Using != with floating-point numbers (double k) is unreliable due to precision issues. k may never exactly equal 1.0 (e.g., it might be 0.9999999), causing an infinite loop.

**Correction:** Use <= 1.0 to ensure the loop stops when k reaches or exceeds 1.0.

**3. Second Code Segment**

**Error:** Missing break after case 1, causing fall-through to case 2’s output when n = 1. Both messages print, which is unintended.

Correction: Add break after case 1.

**4. Second Code Segment**

**Error:** The condition n < 10 prints 1 to 9, not 1 to 10 as intended.

**Correction:** Change to n <= 10 to include 10.

**Exercise 2:**

**4.5**

* **Initialization**: Set the loop counter to a starting value (e.g., int i = 0).
* **Condition:** Test the counter to determine if the loop continues (e.g., i < 10).
* **Body:** Execute statements each iteration (e.g., System.out.println(i)).
* **Increment:** Update the counter (e.g., i++) to eventually terminate the loop.

**4.6**

* **Similarities:** 
  + Both repeat a block of code based on a condition.
  + Both can achieve the same counter-controlled repetition.
* **Differences:** 
  + while: More flexible, requires manual initialization and increment (e.g., int i = 0; while (i < 10) { ... i++; }).
  + for: Compact, combines initialization, condition, and increment in one line (e.g., for (int i = 0; i < 10; i++) { ... }).
  + while is better for indefinite loops; for suits known iteration counts.

**4.7**

Situation: Validating user input (e.g., asking for a positive number).

* Why: do…while ensures the body executes at least once before checking the condition, guaranteeing the user is prompted even if the first input is invalid. A while loop might skip the body if the condition is false initially, requiring extra setup.
* Example: do { num = input.nextInt(); } while (num <= 0); vs. while needing prior initialization.

**4.8**

* **Similarities:** 
  + Both alter loop flow.
  + Used within loops (for, while, do…while).
* **Differences:** 
  + break: Exits the loop entirely, moving to the next statement after the loop.
  + continue: Skips the current iteration, proceeding to the next iteration.
  + Example: In a loop from 1 to 10, if (i == 5) break; stops at 5; if (i == 5) continue; skips 5 but continues to 10.

**4.9**

**a)**

**Errors:** For should be for, increment i++ should be i-- to count down, missing braces for clarity.

* **Corrected:**

for (int i = 100; i >= 1; i--) {

System.out.println(i);

}

**b)**

* **Error:** Missing break after case 0, causing fall-through to print both messages for even numbers.
* **Corrected:**

switch (value % 2) {

case 0:

System.out.println("Even integer");

break;

case 1:

System.out.println("Odd integer");

}

**c)**

* **Error:** i += 2 increases i, causing an infinite loop (19, 21, …); should be i -= 2 to decrease.
* **Corrected:**

for (int i = 19; i >= 1; i -= 2)

System.out.println(i);

**d)**

* **Errors:** While should be while (case-sensitive), condition excludes 100 (should be <= 100).
* Corrected:

int counter = 2;

do {

System.out.println(counter);

counter += 2;

} while (counter <= 100);

**4.10**

* **Answer:** Prints 10 rows of 5 @ symbols each (50 @s total).
  + Outer loop (i): 10 iterations for rows.
  + Inner loop (j): 5 iterations for @s per row.
  + Output:
    - @@@@@
    - @@@@@
    - ...
    - (10 lines total)