Answer to exercise 4 – 4.10

(A)

* **Error:** The semicolon (;) after the while condition creates an empty loop body. This means the loop will execute without doing anything until i becomes 11, and then the increment ++i (outside the loop) will execute once.
* **Correction:** Remove the semicolon after the while condition and enclose the increment ++i within the loop's curly braces {}.

**Correct code:**

i = 1;

while (i <= 10) {

++i;

}

(B)

* **Error:** Floating-point numbers (like 0.1) are not stored precisely in computers. Due to rounding errors, the loop might not terminate when k exactly equals 1.0. It might skip over 1.0 and lead to an infinite loop or unexpected behavior.
* **Correction:** Use a comparison with a tolerance value (epsilon) to check if k is "close enough" to 1.0.

**Correct code:**

for (double k = 0.1; Math.abs(k - 1.0) > 1e-9; k += 0.1) {

System.out.println(k);

}

(C)

* **Error:** The case 1 statement is missing a break statement. Without a break, execution will "fall through" to the case 2 statement even if n is 1.
* **Correction:** Add a break statement after the case 1 code.

Correct code:

switch (n) {

case 1:

System.out.println("The number is 1");

break;

case 2:

System.out.println("The number is 2");

break;

default:

System.out.println("The number is not 1 or 2");

break;

}

(D)

* **Error:** The loop condition n < 10 will cause the loop to stop when n is 9. The n++ inside the println statement means n is incremented after its value is printed. Therefore, the last value printed will be 9, not 10.
* **Correction:** Change the loop condition to n <= 10 or increment n before printing.

Correct code:

n = 1;

while (n <= 10) {

System.out.println(n++);

}

**(4.5)**

The four basic elements of counter-controlled repetition (loops) are:

1. **Initialization:** Setting the loop control variable to its starting value (e.g., i = 0).
2. **Condition Test:** Evaluating a boolean expression that determines whether the loop should continue (e.g., i < 10).
3. **Loop Body:** The block of code that is executed repeatedly as long as the condition is true.
4. **Increment/Decrement:** Modifying the loop control variable (usually by incrementing or decrementing it) after each iteration (e.g., i++).

**(4.6 )**

**Similarities:**

* + Both are used for repetitive execution of a block of code based on a condition.
  + Both can be used to achieve the same results, although one might be more concise or clear in certain situations.
* **Differences:**
  + **Structure:**
    - for loops typically combine initialization, condition testing, and increment/decrement into a single line.
    - while loops separate the initialization and increment/decrement from the condition test.
  + **Use Cases:**
    - for loops are generally preferred when the number of iterations is known or can be easily determined.
    - while loops are better suited when the number of iterations is unknown or dependent on a condition that might change during execution.

**(4.7)**

A do...while loop is more appropriate when you need to execute the loop body at least once, regardless of the initial condition. This is because the condition is checked *after* the loop body is executed.

* **Example:** Reading user input until a valid value is entered.

Java

int input;

do {

System.out.print("Enter a positive number: ");

input = scanner.nextInt();

} while (input <= 0); // Loop continues if input is not positive

In this case, you want to ask the user for input at least once, even if their first attempt is invalid.

**(4.8)**

**Similarities:**

* + Both are used to alter the normal flow of control within loops.
* **Differences:**
  + **break:** Terminates the loop immediately, and execution continues with the statement following the loop.
  + **continue:** Skips the remaining statements in the current iteration of the loop and proceeds to the next iteration.

**(4.9)**

**a) For (i = 100, i >= 1, i++) System.out.println(i);**

* **Error:** The increment i++ will cause i to increase, not decrease. This will lead to an infinite loop.
* **Correction:** Change i++ to i-- to decrement i.

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

System.out.println(i);

}

**(b)**

* **Error:** The case 0 statement is missing a break statement. Without it, execution will "fall through" to case 1 even if value is even.
* **Correction:** Add a break statement after case 0.

Correct code:

switch (value % 2) {

case 0:

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

break;

case 1:

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

break;

}

**(c)**

* **Error:** The loop will print all odd numbers, but it will increment i by 2, which is correct for getting odd numbers, but it will not start at 19 and go *down* to 1.
* **Correction:** Change i += 2 to i -= 2.

Correct code:

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

System.out.println(i);

}

**(d)**

* **Error:** The loop condition counter < 100 will cause the loop to stop when counter is 100, but the last value printed will be 98.
* **Correction:** Change the loop condition to counter <= 100.

Java

int counter = 2;

do {

System.out.println(counter);

counter += 2;

} while (counter <= 100);

(4.10)

**Output:**

The program will print a pattern of '@' characters. The outer loop controls the number of lines, and the inner loop controls the number of '@' characters printed on each line.

The output will look like this:

@@@@@

@@@@@

@@@@@

@@@@@

@@@@@

@@@@@

@@@@@

@@@@@

@@@@@

@@@@@