Exercise 3.10 – 3.15 Answer

A. Similarities:

-Conditional Execution: Both if and while statements rely on a boolean condition to determine whether a block of code should be executed.

-Control Flow: They both control the flow of execution in a program, allowing for branching or looping based on conditions.

B. Differences:

Purposes:

- if Statement: Executes a block of code at most once if the condition is true. It's used for decision-making or selection.

-while Statement: Executes a block of code repeatedly as long as the condition is true. It's used for iteration or looping.

Execution Count:

- if Statement: The associated block of code is executed either zero times (if the condition is false) or one time (if the condition is true).

- while Statement: The associated block of code can be executed zero or more times, depending on the condition.

Flow Control:

- if Statement: After the block of code (if executed), the program continues with the next statement after the if block.

-while Statement: After the block of code is executed, the condition is re-evaluated, and the block is executed again if the condition is still true. This creates a loop.

3.11. Integer Division in Java:

-- Truncation: When you divide one integer by another in Java, the result is also an integer. The fractional part of the result is truncated (discarded). This means the result is rounded down to the nearest whole number.

- Example: 7 / 3 in Java will result in 2, not 2.333....

Avoiding Truncation:

To avoid integer truncation and get a floating-point result (with the fractional part), you need to ensure that at least one of the operands is a floating-point type (e.g., float or double). Here's how you can do it:

A.Cast one operand to a float or double:

int numerator = 7;

int denominator = 3;

double result = (double) numerator / denominator; // result will be 2.333...

B.Use a floating-point literal:

int numerator = 7;

int denominator = 3;

double result = numerator / 3.0; // result will be 2.333...

3.12. Control statements can be combined in two primary ways:

A.Nesting: Placing one control statement inside another. This allows for more complex decision-making and looping structures.

-Example (Nested if):

if (condition1) {

if (condition2) {

// Code to execute if both condition1 and condition2 are true

}

}

Example (Nested while):

while (condition1) {

while (condition2) {

// Code to execute while both condition1 and condition2 are true

}

}

B. Sequential Composition: Placing control statements one after another in a sequence. This allows for a series of actions to be executed in order.

-Example (if followed by while):

if (condition1) {

// Code to execute if condition1 is true

}

while (condition2) {

// Code to execute while condition2 is true

}

3.13 What type of repetition would be appropriate for calculating the sum of the first 100 positive integers? What type would be appropriate for calculating the sum of an arbitrary number of positive integers? Briefly describe how each of these tasks could be performed.

A. Sum of the First 100 Positive Integers:

-Appropriate Repetition: A for loop is ideal because you know the exact number of iterations (100).

-Code Example:

int sum = 0;

for (int i = 1; i <= 100; i++) {

sum += i;

}

System.out.println("Sum of first 100 integers: " + sum);

B.Sum of an Arbitrary Number of Positive Integers:

-Appropriate Repetition: A while loop is better because you don't know in advance how many integers will be entered. You'll typically use a sentinel value (e.g., -1) to indicate the end of input.

-Code Example (using a sentinel value):

(CODE IS IN NOTEPAD ++)

3.14 What is the difference between preincrementing and postincrementing a variable?

-Preincrementing (++variable):

-The variable's value is incremented before it is used in the expression.

Example:

int x = 5;

int y = ++x; // x becomes 6, then y is assigned the value of x (which is 6)

// Now x is 6 and y is 6

Postincrementing (variable++):

--The variable's value is incremented after it is used in the expression.

-Example:

int a = 5;

int b = a++; // b is assigned the value of a (which is 5), then a is incremented

// Now a is 6 and b is 5

3.15 Identify and correct the errors in each of the following pieces of code.

a) if (age >= 65); System.out.println("Age is greater than or equal to 65"); else System.out.println("Age is less than 65");

--Errors:

1.Semicolon after if (age >= 65): This semicolon terminates the if statement prematurely. The System.out.println statement that follows will always execute, regardless of the condition.

2.Missing else block braces: The else block should have braces {} to enclose the statement it controls. Without braces, only the first statement after else is considered part of the else block.

--Corrected Code:

if (age >= 65) {

System.out.println("Age is greater than or equal to 65");

} else {

System.out.println("Age is less than 65");

}

b) int x = 1, total; while (x <= 10) { total += x; ++x; }

--Errors:

1.Uninitialized total variable: The total variable is declared but not initialized before it's used in the while loop. This will lead to a compilation error.

2.Missing semicolon after total += x: This line needs a semicolon to terminate the statement.

--Corrected Code:

int x = 1, total = 0; // Initialize total to 0

while (x <= 10) {

total += x;

++x;

}

c) while (x <= 100) total += x; ++x; d)

--Errors:

1.Missing braces for the while loop: The loop body should be enclosed in curly braces {}. Without braces, only the first statement after the while condition is considered part of the loop.

2.d) at the end: The d) at the end is syntactically incorrect. It's unclear what it's supposed to do and should be removed.

3.Uninitialized total and x: Neither total nor x is initialized before use, which will cause errors.

--Corrected Code (assuming total and x are meant to be integers):

int x = 0; // Initialize x

int total = 0; // Initialize total

while (x <= 100) {

total += x;

++x;

}

d) while (y > 0) { System.out.println(y); ++y; }

--Errors:

1.Infinite loop if y is initially positive: If y starts as a positive number, the loop will run forever because y is incremented, making it larger and never reaching a value less than or equal to 0.

2.Potential for y to become very large: If the loop runs for a long time, y could exceed the maximum value for its data type, potentially leading to unexpected behavior.

--Corrected Code (assuming you want to decrement y):

while (y > 0) {

System.out.println(y);

--y; // Decrement y instead of incrementing

}