**3.10**

Compare and contrast the if single-selection statement and the while repetition statement. How are these two statements similar? How are they different?

* **Similarities:** 
  + Both are control statements that alter the flow of a program based on conditions.
  + Both use a boolean expression (e.g., **condition**) to determine whether to execute their body.
  + Both can execute a block of code (enclosed in **{}**) or a single statement.
* **Differences:** 
  + **if**: Executes its body once if the condition is true; it’s a single-selection statement for decision-making.
  + **while**: Repeatedly executes its body as long as the condition remains true; it’s a repetition statement for looping.
  + if doesn’t inherently repeat, while **while** is designed for iteration until the condition becomes false.

**3.11**

Explain what happens when a Java program attempts to divide one integer by another. What happens to the fractional part of the calculation? How can you avoid that outcome?

* **What Happens:** When dividing two integers (e.g., **5 / 2**), Java performs integer division, resulting in an integer quotient (e.g., 2).
* **Fractional Part:** The fractional part (e.g., **0.5** in **5 / 2**) is discarded (truncated), not rounded.
* **Avoiding It:** To retain the fractional part, cast one or both operands to a floating-point type (e.g., **double** or **float**) before division: **(double) 5 / 2** yields **2.5**

**3.12**

Describe the two ways in which control statements can be combined.

* **Nesting:** Placing one control statement inside another (e.g., an **if** inside a while or vice versa). Example: Checking a condition within a loop.
* **Sequencing:** Using control statements one after another (e.g., an **if** followed by a while). Example: Decision-making followed by repetition.

**3.13**

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

* **First 100 Positive Integers:** 
  + **Type:** for loop (definite repetition, known number of iterations).
  + **How:** Initialize a sum to 0, loop from 1 to 100, adding each number to the sum (e.g., for (int i = 1; i <= 100; i++) sum += i;).
* **Arbitrary Number of Positive Integers**:
  + **Type:** while loop (indefinite repetition, unknown number of inputs).
  + How: Use a sentinel-controlled while loop; prompt the user for numbers, add each to a sum, and stop when a sentinel (e.g., -1) is entered.

**3.14**

What is the difference between preincrementing and postincrementing a variable?

* **Preincrement (**++x**):** Increments the variable before its value is used in the expression (e.g., y = ++x sets x to x + 1, then assigns the new x to y).
* **Postincrement (**x++**):** Increments the variable after its value is used in the expression (e.g., y = x++ assigns the current x to y, then increments x).
* **Example:** If x = 5, y = ++x makes y = 6, x = 6; but y = x++ makes y = 5, x = 6.

**3.15**

**A)**

**Identify and correct the errors in each piece of code.**

if (age >= 65); // Error: Semicolon after if makes it empty; else becomes detached

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

else System.out.println("Age is less than 65)"; // Error: Unmatched else

**Corrected:**

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; // Error: total not initialized

while (x <= 10) {

total += x; // Error: Using uninitialized total

++x;

}

**Corrected:**

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

while (x <= 10) {

total += x;

++x;

}

**C)**

while (x <= 100) // Error: No braces, only total += x is looped; x undefined

total += x;

++x; // Executes once after loop

**Corrected:**

int x = 0, total = 0; // Define and initialize x and total

while (x <= 100) {

total += x;

++x;

}

**D)**

while (y > 0) { // Error: y undefined; ++y causes infinite loop if y > 0

System.out.println(y);

++y; // Should be --y to decrease y

**Corrected:**

int y = 10; // Define and initialize y

while (y > 0) {

System.out.println(y);

--y; // Decrease y to avoid infinite loop

}