**CHAPTER 3**

**ANSWERS TO EXERCISE 1**

**ANSWER 3.10**

**Comparison of if Single-Selection Statement and while Repetition Statement in Java**

**Similarities:**

* **Conditional Execution:** Both statements rely on a conditional expression to determine their execution.
* **Boolean Expression:** The conditional expression in both cases is a Boolean expression that evaluates to either true or false.

**Differences:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **if Single-Selection**  **Statement** | **while Repetition Statement** |
| **Execution:** | Executes a block of code once if the condition is true. | Executes a block of code repeatedly as long as the condition remains true. |
| **Flow of Control:** | Linear flow. Once the block is executed (or not), control moves to the next statement**.** | Looping flow. The control jumps back to the beginning of the loop to re-evaluate the condition after each iteration. |
| **Number of Executions:** | At most once. | Zero or more times. |

**ANSWER 3.11**

When you divide two integer values in Java, the result is also an integer. This means that any fractional part of the result is truncated, or discarded.

In this case, the actual result of the division is 3.3333..., but since both operands are integers, the fractional part is discarded, and only the integer part (3) is assigned to the result variable.

**To avoid this truncation and obtain a more precise result, you can use one of the following approaches:**

**1. Casting to a Floating-Point Type:**

* Convert at least one of the operands to a floating-point type (like float or double) before the division:

By casting 10 to a double, the division is performed as floating-point division, and the result will include the fractional part.

**2. Using Floating-Point Literals:**

* Directly use floating-point literals for one or both operands:

Adding the decimal point to 10 makes it a double literal, ensuring floating-point division.

**Remember:**

* Integer division is often useful for specific calculations, such as finding remainders or performing discrete operations.
* When precision is crucial, using floating-point division is essential.
* Be aware of potential rounding errors and limitations in floating-point arithmetic, especially when dealing with very large or very small numbers.

**ANSWER 3.12**

**Combining Control Statements in Java**

Control statements in Java can be combined to create more complex and flexible control flow within your programs. Here are the two primary ways to combine them:

**1. Nesting:**

* This involves placing one control statement within another.
* It's commonly used to create hierarchical structures in your code.

**2. Sequential Execution:**

* This involves placing control statements one after another.
* The execution flow progresses from one statement to the next.
* Here, a **while** loop is followed by an **if** statement within its body. The loop continues to execute as long as the **condition** is true, and within each iteration, the **if** statement checks the **innerCondition** to determine further actions002E.
* By effectively combining these techniques, you can create intricate control flow structures to solve various programming problems. Remember to consider readability and maintainability when nesting or sequentially combining control statements.

**ANSWER 3.13**

For calculating the sum of a fixed number of integers (like the first 100 positive integers), a **definite loop** is the most suitable choice. A definite loop, such as a for loop, executes a specific number of times.

For calculating the sum of an arbitrary number of positive integers, an **indefinite loop** is more appropriate. An indefinite loop, like a while or do-while loop, continues to execute until a certain condition is met.

**OR**

For a fixed number of iterations, like summing the first 100 integers, a **definite loop** like a for loop is ideal. You can initialize a counter, set a condition to iterate 100 times, and increment the counter in each iteration.

For an arbitrary number of iterations, an **indefinite loop** like a while or do-while loop is suitable. You can continue the loop until a specific condition, such as user input being zero, is met.

**ANSWER 3.14**

Both pre-increment and post-increment operators increment a variable by 1. The key difference lies in when the increment occurs relative to the use of the variable's value in an expression.

**Pre-increment (++)**

* Increments the variable **before** its value is used in the expression.

**Post-increment (++)**

* Increments the variable **after** its value is used in the expression.

**In essence:**

* **Pre-increment:** Increment first, then use.
* **Post-increment:** Use first, then increment.

**ANSWER 3.15**

**a) Missing Braces:**

Java

if (age >= 65) {

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

} else {

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

}

**b) Uninitialized Variable:**

Java

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

while (x <= 10) {

total += x;

++x;

}

**c) Missing Braces and Infinite Loop:**

Java

while (x <= 100) {

total += x;

++x;

}

**d) Infinite Loop and Incorrect Increment:**

Java

while (y > 0) {

System.out.println(y);

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

}

**PSEUDOCODE**

**ANSWER 3.21**

Initialize counter to 1

Initialize largest to the first input number

While counter is less than or equal to 10

Input number

If number is greater than largest

Set largest to number

Increment counter

**ANSWER 3.23**

Initialize counter to 1

Initialize largest1 to the first input number

Initialize largest2 to the smaller of the first two input numbers

While counter is less than or equal to 10

Input number

If number is greater than largest1

Set largest2 to largest1

Set largest1 to number

Else if number is greater than largest2

Set largest2 to number

Increment counter