1. What is the purpose of a loop structure?

A loop structure allows a program to **repeat a set of instructions** multiple times without having to write the same code over and over.
It's useful when you need to process data or perform actions until a condition is met — for example, printing numbers 1 to 10 or asking for input until it's valid.

2. Explain the difference between a while statement and a do-while statement.

while loop: Tests the condition *before* running the loop body. If the condition is false at the start, the loop may **never run**.

```
while (condition) {
    // code
}
```

-

do-while loop: Tests the condition *after* running the loop body, so it **always runs at least once**.

```
do {
    // code
} while (condition);
```

3. An input validation loop... In which review did you write code for one?

You likely wrote an input validation loop in the **chapter section where you learned about while or do-while loops**, especially when checking for valid input (like making sure a number isn't negative or within a range).
Example: The **Input Validation Loop Review** where you verified user input before continuing.

4. a) What is an infinite loop

An **infinite loop** is a loop that **never ends** because its condition is always true or it never updates in a way that makes the condition false.

b) Two errors that can lead to an infinite loop:

1. Forgetting to update the loop control variable

2. Writing a condition that never becomes false (e.g., `while (true)` without a `break`)

c) What is meant by overflow?
Overflow happens when a variable's value goes beyond the maximum limit it can hold (for example, an `int` exceeding 2,147,483,647).

5. How many times will the do-while loop execute?

```
int x = 0;
do {
    x = x + 2;
} while (x < 120);
```

- The loop adds 2 each time, starting from 0.
- It stops when `x` becomes **120 or greater**.
- To reach 120: `0, 2, 4, …, 118, 120` → It executes **60 times**.

6. What initial value of x would make the loop infinite?

```
do {
    x = x + 3;
} while (x < 120);
```

If `x` starts with a **negative value that decreases or never increases toward 120**, it would be infinite.
However, since the loop *adds 3 each time*, the only way it could be infinite is if **the condition or variable never changes properly** — for example:

```
do {
    x = x - 3;  // decreases instead of increases
} while (x < 120);
```

So if `x` starts below 120 and the loop decreases, it would run forever.

7. Compare and contrast counters and accumulators.

Feature	Counter	Accumulator
Purpose	Counts how many times something happens	Adds (accumulates) values together
Change per iteration	Increases by a fixed amount (often +1)	Increases by varying amounts (user input, totals, etc.)
Uses	Loop iteration counting, tracking repetitions	Calculating totals, averages, sums

Examples:

- Counter: Counting the number of students in a loop.
- Accumulator: Adding up test scores.

8. Write a for statement that sums the integers from 3 to 10, inclusive.

```
int sum = 0;
for (int i = 3; i <= 10; i++) {
    sum = sum + i;
}
System.out.println("Sum = " + sum);
```

Output will be **52**.

9. List two factors to consider when choosing a loop structure.

1. **Whether the number of iterations is known**
    - Use a **for loop** if you know how many times to repeat.
    - Use a **while or do-while** if you don't know in advance.
2. **When the condition should be tested**

- Before the loop (use `while`)
- After the first run (use `do-while`)

11. String question

```
String x = "my string.";
```

a) `x.length()` → **10** (there are 10 characters, including the space and the period)
b) `x.substring(0, 3)` → **"my "** (characters from index 0 up to, but not including, 3)