**Looping constructs in java**

**What is a loop in java?**

In Java, a loop is a programming construct that allows you to execute a block of code repeatedly as long as a certain condition is true.

Loops are essential for automating repetitive tasks and iterating over collections of data.

There are several types of loops in Java, such as the for loop, while loop, do-while loop, for-each loop, and enhanced for loop.

The basic idea of a loop is to repeatedly execute a block of code until a specific condition is met or until all elements in a collection have been processed.

Loops help to streamline code by avoiding repetitive coding patterns and making it easier to manage iterations.

**List of loops available in java**

In Java, there are several looping constructs available to facilitate repetitive execution of code. Here's a list of the main looping constructs in Java:

1. while

2. do while

3. for

4. for-each (or) enhanced for loop

5. forEach

**while loop**

In Java, a while loop is a control flow statement that allows you to execute a block of code repeatedly as long as a specified condition is true.

The syntax of a while loop in Java is as follows:

while (condition) {

// code to be executed

}

The **condition** is a boolean expression that is evaluated before each iteration of the loop.

If the condition evaluates to **true**, the code inside the loop is executed.

After the execution of the loop body, the condition is evaluated again.

If it still evaluates to **true**, the loop continues to execute; otherwise, the loop terminates, and control passes to the next statement after the loop.

Here's a simple example of a while loop in Java that prints numbers from 1 to 5:

int i = 1;

while (i <= 5) {

System.out.println(i);

i++;

}

In this example:

* We initialize the variable **i** with the value **1**.
* The condition **i <= 5** is evaluated before each iteration. As long as **i** is less than or equal to **5**, the loop continues.
* Inside the loop, the current value of **i** is printed, and then **i** is incremented by **1**.
* The loop continues executing until **i** is no longer less than or equal to **5**.

Once **i** becomes **6**, the condition **i <= 5** becomes **false**, and the loop terminates.

It's important to ensure that the condition inside a while loop will eventually become **false** to prevent infinite loops, which can cause the program to hang or become unresponsive.

**do while loop**

In Java, a do-while loop is similar to a while loop, but it guarantees that the loop body is executed at least once, even if the condition is initially false.

The syntax of a do-while loop in Java is as follows:

do {

// code to be executed

} while (condition);

The loop body is executed first, and then the condition is evaluated.

If the condition evaluates to **true**, the loop body is executed again, and the process repeats.

If the condition evaluates to **false**, the loop terminates, and control passes to the next statement after the loop.

Here's a simple example of a do-while loop in Java that prints numbers from 1 to 5:

int i = 1;

do {

System.out.println(i);

i++;

} while (i <= 5);

In this example:

* The variable **i** is initialized with the value **1**.
* The loop body prints the current value of **i** and then increments **i** by **1**.
* After executing the loop body, the condition **i <= 5** is evaluated.
* Since **i** is less than or equal to **5**, the loop continues to execute.
* This process repeats until **i** becomes **6**, at which point the condition **i <= 5** becomes **false**, and the loop terminates.

Unlike the while loop, where the condition is evaluated before the loop body is executed, the do-while loop ensures that the loop body is executed at least once before checking the condition.

This can be useful in scenarios where you need to execute certain code before testing a condition for continuation.

**for loop**

A for loop in Java is a control flow statement used to repeatedly execute a block of code for a specified number of times.

It's particularly useful when you know the exact number of iterations you want to perform.

The syntax of a for loop in Java is as follows:

for (initialization; condition; update) {

// code to be executed

}

Here's what each part of the for loop means:

* **initialization**: This part is executed once before the loop starts. It initializes a loop control variable, typically used as a counter. The initialization is optional and can be omitted if the loop control variable is already initialized before the loop.
* **condition**: This part specifies the condition for continuing the loop. The loop continues to execute as long as the condition evaluates to true. If the condition evaluates to false, the loop terminates.
* **update**: This part is executed after each iteration of the loop, typically used to update the loop control variable. It's commonly used to increment or decrement the loop control variable. Like the initialization, the update is optional and can be omitted if the loop control variable is updated within the loop body.
* **code to be executed**: This is the block of code that is executed repeatedly as long as the condition is true.

Here's a simple example of a for loop in Java that prints numbers from 1 to 5:

for (int i = 1; i <= 5; i++) {

System.out.println(i);

}

In this example:

* The loop control variable **i** is initialized with the value **1**.
* The loop continues as long as **i** is less than or equal to **5**.
* The loop body prints the value of **i**.
* After executing the loop body, **i** is incremented by **1**.
* This process repeats until **i** becomes **6**, at which point the condition **i <= 5** becomes false, and the loop terminates.

The for loop provides a concise way to express repetitive tasks in Java, especially when the number of iterations is known beforehand.

**for-each loop (or) enhanced for loop**

A for-each loop in Java, also known as an enhanced for loop, is a special type of loop that simplifies iterating over elements in an array or a collection.

It is particularly useful when you want to iterate over all elements of an array or a collection without the need for explicit index manipulation.

The syntax of a for-each loop in Java is as follows:

for (type element : array) {

// code to be executed for each element

}

Here's what each part of the for-each loop means:

* **type**: This specifies the data type of elements in the array or collection.
* **element**: This is a variable that represents each element in the array or collection.
* **array**: This is the array or collection over which the loop iterates.

Here's a simple example of using a for-each loop to iterate over elements of an array:

int[] numbers = {1, 2, 3, 4, 5};

for (int number : numbers) {

System.out.println(number);

}

In this example:

* **int** is the data type of elements in the array **numbers**.
* **number** is the variable that represents each element in the array.
* **numbers** is the array over which the loop iterates.
* The loop body prints each element of the array.

For-each loops can also be used with collections such as ArrayList, LinkedList, etc.

Here's an example of using a for-each loop with an ArrayList:

List<String> names = new ArrayList<>();

names.add("Alice");

names.add("Bob");

names.add("Charlie");

for (String name : names) {

System.out.println(name);

}

In this example, the for-each loop iterates over each element (String) in the **names** ArrayList and prints them.

The for-each loop provides a concise and readable way to iterate over elements of arrays and collections in Java, making the code more expressive and less error-prone.

**forEach**

In Java, the **forEach** loop is a feature introduced in Java 8 that allows for concise iteration over elements in collections, arrays, or any object implementing the **Iterable** interface.

It's part of the Stream API, providing a more functional approach to iterating over elements.

The **forEach** loop simplifies the process of iterating over elements by abstracting away the underlying implementation details and providing a clean syntax. It's commonly used with lambda expressions or method references to specify the action to be performed on each element.

The syntax of the **forEach** loop in Java is as follows:

iterable.forEach(element -> {

// code to be executed for each element

});

Here's an example of using the **forEach** loop to iterate over elements of an ArrayList:

List<String> names = new ArrayList<>();

names.add("Alice");

names.add("Bob");

names.add("Charlie");

**names.forEach(name -> System.out.println(name));**

In this example, the **forEach** loop iterates over each element in the **names** ArrayList and prints them using a lambda expression.

You can also use method references with the **forEach** loop.

Here's an example:

List<String> names = new ArrayList<>();

names.add("Alice");

names.add("Bob");

names.add("Charlie");

**names.forEach(System.out::println);**

In this example, **System.out::println** is a method reference to the **println** method of the **PrintStream** class.

It's equivalent to using a lambda expression like **name -> System.out.println(name)**.

The **forEach** loop provides a concise and expressive way to iterate over elements in collections or arrays, making the code more readable and less error-prone.

It's especially useful when combined with lambda expressions or method references to specify the action to be performed on each element.

**Some examples**

Example 1: While Loop

**Write a program that prints numbers from 1 to 10 using a while loop.**

Solution:

public class WhileLoopExample {

public static void main(String[] args) {

int i = 1;

while (i <= 10) {

System.out.println(i);

i++;

}

}

}

**Example 02: Do-While Loop**

**Write a program that asks the user to enter a number and keeps asking until the user enters a positive number.**

Solution:

import java.util.Scanner;

public class DoWhileLoopExample {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int number;

do {

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

number = scanner.nextInt();

} while (number <= 0);

System.out.println("You entered a positive number: " + number);

}

}

**Example 03: For Loop**

**Write a program that calculates the sum of the first 10 natural numbers using a for loop.**

Solution:

public class ForLoopExample {

public static void main(String[] args) {

int sum = 0;

**for (int i = 1; i <= 10; i++) {**

sum += i;

}

System.out.println("Sum of the first 10 natural numbers: " + sum);

}

}

**Example 04: For-Each Loop**

**Write a program that iterates over an array of strings and prints each string in uppercase.**

Solution:

public class ForEachLoopExample {

public static void main(String[] args) {

String[] names = {"Alice", "Bob", "Charlie"};

**for (String name : names) {**

System.out.println(name.toUpperCase());

}

}

}

**Example 5: forEach Loop**

**Write a program that calculates the total cost of items in a shopping list. Use a List of doubles to represent the prices of items.**

**Solution:**

import java.util.ArrayList;

import java.util.List;

public class ShoppingCart {

public static void main(String[] args) {

// Define the list of prices

List<Double> prices = new ArrayList<>();

prices.add(10.99);

prices.add(5.49);

prices.add(3.99);

prices.add(8.75);

// Calculate the total cost using forEach method

double[] totalCost = {0}; // Using an array to store totalCost to modify it in lambda expression

**prices.forEach(price -> totalCost[0] += price);**

// Print the total cost

System.out.println("Total cost of items in the shopping list: $" + totalCost[0]);

}

}

**Example 6: While Loop**

**Write a program to find the factorial of a given number using a while loop.**

**Solution:**

public class FactorialWhileLoop {

public static void main(String[] args) {

int number = 5;

int factorial = 1;

int i = 1;

while (i <= number) {

factorial \*= i;

i++;

}

System.out.println("Factorial of " + number + " is: " + factorial);

}

}

**Exercise 7: Do-While Loop**

**Write a program to print all uppercase letters in reverse order using a do-while loop.**

**Solution:**

public class UppercaseLetters {

public static void main(String[] args) {

char letter = 'Z';

do {

System.out.print(letter + " ");

letter--;

} while (letter >= 'A');

}

}

**Exercise 8: For Loop**

**Write a program to print the Fibonacci series up to the nth term using a for loop.**

**Solution:**

public class FibonacciSeries {

public static void main(String[] args) {

int n = 10;

int first = 0, second = 1;

System.out.println("Fibonacci series up to " + n + " terms:");

for (int i = 0; i < n; i++) {

System.out.print(first + " ");

int next = first + second;

first = second;

second = next;

}

}

}

**Exercise 9: For-Each Loop**

**Write a program to find the largest element in an array using a for-each loop.**

**Solution:**

public class LargestElement {

public static void main(String[] args) {

int[] numbers = {5, 8, 2, 10, 3};

int largest = numbers[0];

for (int number : numbers) {

if (number > largest) {

largest = number;

}

}

System.out.println("Largest element in the array: " + largest);

}

}