BATCH NAME: UNIQUE

Sowjanya Morisetty(245132)

Yashwanth Ravula(245098)

Meghana Bareddy(245111)

Rayona Mathew(245072)

Loops in Java

The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is **fixed**, it is recommended to use for loop.

There are three types of for loops in Java.

1.Java for Loop

2.Java while Loop

3.Java do while Loop

Java Simple for Loop

In for loop , We can initialize the variable, check condition and increment/decrement value. It consists of four parts:

1. **Initialization**: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.
2. **Condition**: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.
3. **Increment/Decrement**: It increments or decrements the variable value. It is an optional condition.
4. **Statement**: The statement of the loop is executed each time until the second condition is false.

Syntax:

```
for(initialization; condition; increment/decrement){
//statement or code to be executed
}
```

Example:

```
public class ForExample {
public static void main(String[] args) {
    //Code of Java for loop
    for(int i=1;i<=10;i++){
        System.out.println(i);
    }  } }
```

OUTPUT:

```
1
2
3
4
5
6
7
```

```
8
9
10
```

Time and Space Complexity of for loop

Assuming that the input data has a size of n, the time complexity of a for loop is generally O(n). This means that the loop will execute a number of times proportional to the size of the input data.
The space complexity of a for loop is usually constant, meaning that it does not depend on the size of the input data. This is because the loop only requires a constant amount of memory to keep track of the loop index.
However, if the loop performs operations that require additional memory, such as creating new variables or data structures, then the space complexity of the loop may be higher.
It is important to note that the time and space complexity of a for loop can vary depending on the specific implementation and language used. It is always a good practice to analyze the time and space complexity of your code to ensure optimal performance.

Java Nested for Loop

If we have a for loop inside the another loop, it is known as nested for loop. The inner loop executes completely whenever outer loop executes.

Example:

```
public class NestedForExample {
public static void main(String[] args) {
for(int i=1;i<=3;i++){
for(int j=1;j<=3;j++){
        System.out.println(i+" "+j);
}
}
}
}
```

Output:

```
1 2

1 2

1 3

2 1

2 2

2 3

3 1

3 2

3 3
```

The time and space complexity of nested for loops depend on the number of loops and the size of the input data.there are two nested for loops, and the input data size is n, the time complexity will be O(n^2). This is because the outer loop will execute n times, and the inner loop will execute n times for each iteration of the outer loop, resulting in a total of n*n = n^2 iterations.The space complexity of nested for loops can also depend on the number of loops and the operations performed inside them. If the operations require additional memory, such as creating new variables or data structures, then the space complexity may increase. However, in general, the space complexity of nested for loops is also proportional to the number of loops.

For example, if there are two nested for loops, each of which creates a constant amount of memory for each iteration, the space complexity would be O(1). However, if the inner loop creates a new array or data structure for each iteration, then the space complexity could be O(n^2).It is important to note that the time and space complexity of nested for loops can quickly become computationally expensive as the number of loops and input data size increase. It is always a good practice to analyze the time and space complexity of your code and consider alternative approaches if necessary to ensure optimal performance.

Java for-each Loop

The for-each loop is used to traverse array or collection in Java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

It works on the basis of elements and not the index. It returns element one by one in the defined variable.

Syntax:

```
for(data_type variable : array_name){
//code to be executed
}
```

Example:

```
public class ForEachExample {
public static void main(String[] args) {
    //Declaring an array
    int arr[]={12,23,44,56,78};
    //Printing array using for-each loop
    for(int i:arr){
        System.out.println(i);
    }
}
10.       }
```

Test it Now

Output:

```
12

23

44

56

78
```

The time and space complexity of a for-each loop, also known as a "for-in" loop, depends on the size of the input data.

Assuming the input data has a size of n, the time complexity of a for-each loop is generally O(n). This is because the loop will execute once for each element in the input data.

The space complexity of a for-each loop is usually constant, meaning that it does not depend on the size of the input data. This is because the loop only requires a constant amount of memory to keep track of the loop index.

However, if the loop performs operations that require additional memory, such as creating new variables or data structures, then the space complexity of the loop may be higher.

It is important to note that the time and space complexity of a for-each loop can vary depending on the specific implementation and language used. It is always a good practice to analyze the time and space complexity of your code to ensure optimal performance.

Java While Loop

The Java *while loop* is used to iterate a part of the program repeatedly until the specified Boolean condition is true. As soon as the Boolean condition becomes false, the loop automatically stops.

The while loop is considered as a repeating if statement. If the number of iteration is not fixed, it is recommended to use the while loop.

Syntax:

1. **while** (condition){
2. //code to be executed
3. I ncrement / decrement statement
4. }

The different parts of do-while loop:

1. Condition: It is an expression which is tested. If the condition is true, the loop body is executed and control goes to update expression. When the condition becomes false, we exit the while loop.

Example:

i <=100

2. Update expression: Every time the loop body is executed, this expression increments or decrements loop variable.

Example:

i++;

Example:

public class WhileExample {
1. **public static void** main(String[] args) {

```
    int i=1;
    while(i<=10){
        System.out.println(i);
    i++;
    }
}
}
```

The time and space complexity of a while loop depends on the condition and the number of iterations it takes to exit the loop.
Assuming the while loop condition is based on the size of the input data and the input data has a size of n, the time complexity of the while loop is generally O(n) if the loop executes n times or less. If the loop executes more than n times, the time complexity is O(m), where m is the number of loop iterations.
The space complexity of a while loop can also depend on the condition and the operations performed inside the loop. If the loop requires additional memory, such as creating new variables or data structures, then the space complexity may increase. However, in general, the space complexity of a while loop is also proportional to the number of iterations.
For example, if the loop creates a new array or data structure for each iteration, then the space complexity could be O(m). However, if the loop only creates a constant amount of memory for each iteration, the space complexity would be O(1).
It is important to note that the time and space complexity of a while loop can quickly become computationally expensive if the loop condition is not properly defined or if the loop executes for a large number of iterations. It is always a good practice to analyze the time and space complexity of your code and consider alternative approaches if necessary to ensure optimal performance.

Java do-while Loop

The Java *do-while loop* is used to iterate a part of the program repeatedly, until the specified condition is true. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use a do-while loop.

Java do-while loop is called an **exit control loop**.. The Java *do-while loop* is executed at least once because condition is checked after loop body.

Syntax:

```
do{
//code to be executed / loop body
//update statement
}while (condition);
```

The different parts of do-while loop:

1. Condition: It is an expression which is tested. If the condition is true, the loop body is executed and control goes to update expression. As soon as the condition becomes false, loop breaks automatically.

Example:

i <=100

2. Update expression: Every time the loop body is executed, the this expression increments or decrements loop variable.

Example:

i++;

```
public class DoWhileExample {
public static void main(String[] args) {
    int i=1;
    do{
        System.out.println(i);
    i++;
    }while(i<=10);
}
}
```

```
OUTPUT :

1
2
3
4
5
6
7
8
9
10
```

The time and space complexity of a do-while loop is similar to that of a while loop, and also depends on the condition and the number of iterations it takes to exit the loop.
Assuming the do-while loop condition is based on the size of the input data and the input data has a size of n, the time complexity of the do-while loop is generally O(n) if the loop executes n times or less. If the loop executes more than n times, the time complexity is O(m), where m is the number of loop iterations.
The space complexity of a do-while loop can also depend on the condition and the operations performed inside the loop. If the loop requires additional memory, such as creating new variables or data structures, then the space complexity may increase. However, in general, the space complexity of a do-while loop is also proportional to the number of iterations.

For example, if the loop creates a new array or data structure for each iteration, then the space complexity could be O(m). However, if the loop only creates a constant amount of memory for each iteration, the space complexity would be O(1).
It is important to note that the main difference between a do-while loop and a while loop is that a do-while loop will always execute at least once, regardless of the loop condition. Therefore, the time and space complexity of a do-while loop may be slightly higher than that of a while loop, depending on the specific implementation and conditions of the loop.
It is always a good practice to analyze the time and space complexity of your code and consider alternative approaches if necessary to ensure optimal performance.