CHAPTER 8: REPETITIONS

8.1 Introducing Loops: Understanding When to Use Repetition

Suppose, you are told to print the first 50 positive even numbers. It would not be smart to write the same print statement 50 times, right? Since we are doing the same task over and over, we can simply put it into a loop. Using loops will significantly make tasks less tedious, less time consuming and smaller for us. Here we reduced nearly 50 lines of code to just 4 lines using repetition which is also knows as loops.

```
System.out.println(0);
System.out.println(2);
System.out.println(4);
…..
System.out.println(98);
```
	```
	int num= 0;
	for (int i=1; i<=50; i++) {
	    System.out.println(num);
	    num= num+2;
	}
	```

A loop structure has the following elements:

- **Initialization Expression(s)**: One or more variable is responsible for the loop to keep running. Hence, initialization expressions are required.
- **Test Expression (Condition)**: Every loop runs if a certain condition is fulfilled. As long as the condition satisfies, the loop must keep on running. The test expression always results in either **true** or **false**.
- **Body of the Loop**: Here the tasks that need repetitive occurrence are present.
- **Update Expression(s)**: The variable responsible for the loop to run must be updated in some way to make sure the loop runs a fixed number of times and not forever. Failure to update leads to an infinite loop. Which means, once we enter the loop, we can never get out, unless the program is closed forcefully.

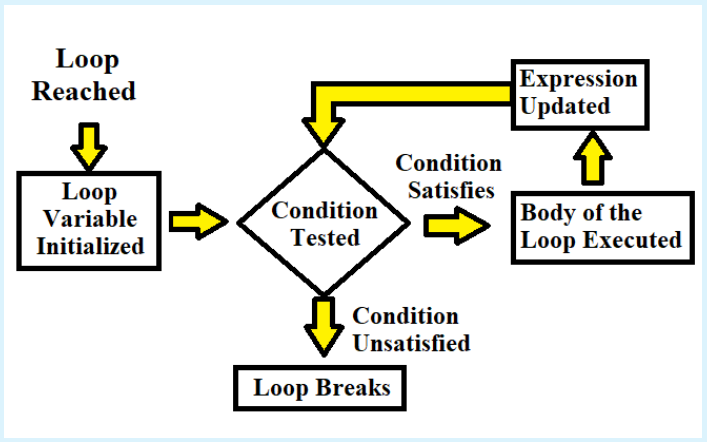
There are mainly three types of loops in Java:

- While
- For
- Do-While

Among these three, while and for loops are entry-controlled loops and do-while is exit-controlled. We shall understand more about these right away.

8.2 While Loop and For Loop

While loop and for loop operate in a very similar way. Execution process and flowchart of both while loop and for loop is shown below:

1. Control reaches the while loop. 2. The Condition is tested. 3. If Condition is true, the flow enters the Body of the Loop. Goes to step 5. 4. If Condition is false, the loop breaks and the flow goes to the immediate next line after the loop. 5. The statements inside the body of the loop get executed. 6. Expression is updated.	 Loop Reached Loop Variable Initialized Condition Tested Condition Satisfies Body of the Loop Executed Expression Updated Condition Unsatisfied Loop Breaks

7. Control flows back to Step 2. Thus, the loop keeps on running until the condition in step 2 is unsatisfied.	

While Loop Structure			

initialization_expression;
while (*test_expression*){
  ***body_of_the_loop*;**
  ***update_expression*;**
}
		int count= 1;
		while (count<4) {
		  System.out.println(count);
		  count= count+1;
		}
			Output:
			1
			2
			3

For Loop Structure			

for (*initialization_expression*;
***test_expression*;**
***update_expression*){**
  body_of_the_loop; }
	initialization_expression
	for (; *test_expression ;*){
	  body_of_the_loop;
	  update_expression; }
		//int i = 1;
		for (int i=1; i<4; i++) {
		  System.out.println(i);
		//i++; }
			Output:
			1
			2
			3

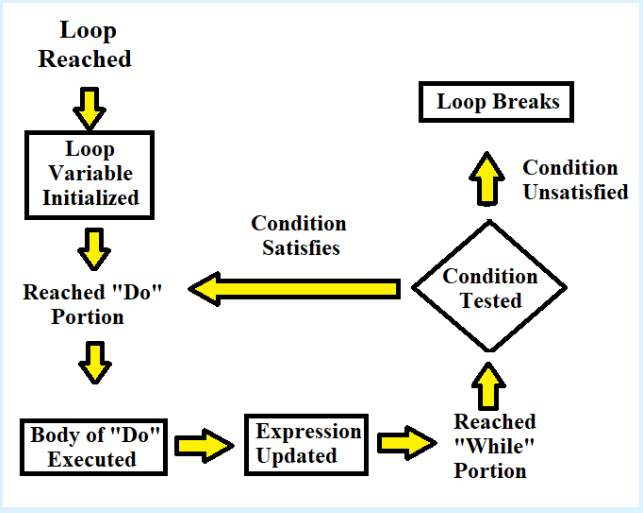
It is not necessary for the expression initialization to be inside the for loop. It can be done before the loop starts. Similarly, the expression update can also be done inside the body of the for loop.

While and for loops are entry control loops because before entering the loop the text expression always checked. Therefore, cases might also occur where we cannot enter the loop at all.

8.3 Do-While Loop

Do-while loop is an exit control loop because it is guaranteed for us to enter the loop. The loop will definitely run once, may or may not run more times. Therefore, the exit is controlled in do-while instead of the entry. The "Do" portion of do-while loop ensures the execution of the loop at least once. The "While" portion checks whether the loop should run further or not. If the condition in the "While" portion satisfies, the body of the "Do" portion is executed again.

Execution process and flowchart of do-while loop is shown below:

1. Control reaches the "Do" portion. 2. The Body of the Loop (Do) is executed. 3. Expression is updated. 4. Control reaches the "While" portion and the test condition is tested. 5. If Condition is true, the flow enters the Body of the Loop. Goes to step 2. 6. If Condition is false, the loop breaks and the flow goes to the immediate next line after the loop.	 Loop Reached Loop Variable Initialized Reached "Do" Portion Body of "Do" Executed Expression Updated Reached "While" Portion Condition Tested Condition Satisfies Condition Unsatisfied Loop Breaks

Do-While Loop Structure

initialization_expression;
do {
    body_of_the_loop;
    update_expression; }
while (*test_expression*);
	int x = 1;
	do {
	        System.out.println(x);
	        x++;}
	while (x < 5);
		Output:
		1
		2
		3
		4

	int x = 1;
	do {
	        System.out.println(x);
	        x++;}
	while (x < 1);
		Output:
		1
		
		Here the loop runs once even though the test condition was always unsatisfactory.

8.4 Infinite Loop

An infinite loop runs forever because the test expression is always satisfied. Infinite loops are a common mistake made by the students when they forget to update the loop controlling variable and therefore, the loop never breaks. Some ways of creating infinite loops are shown below.

Loop Type	**Examples**		

While	while(true){
	    System.out.println(5); }
		int x=1;
		while(x>0){
		    System.out.println(5);
		    x+=1; }
			int x=1;
			while(x>0){
			    System.out.println(5); }

For	for (int i =1; i>0; i++){
	    System.out.println(5); }
		for (int i =1; i>0;){
		    System.out.println(5);
		}
			for (; ;){
			    System.out.println(5);
			}

Infinite loops can be used to solve problems that require indefinite number of iterations. You will learn more on these in Chapter 8.7.

8.5 "Break" and "Continue" Keywords

When we need to stop the loop before finishing all the iterations, we use the **break** keyword. If we want to skip a certain iteration, we use the **continue** keyword.

int count=1;
while(count<5){
    System.out.println(count);
    count++;
    if (count==3){
        break;
    }
}
	for (int i=1; i<5; i++){
	    if (i==3){
	        continue;
	    }
	    System.out.println(i);
	}
		for (int i=1; i<10; i++){
		    if (i==3){
		        continue;
		    }
		    if (i>3){
		        break;
		    }
		    System.out.println(i);
		}

Output: 1 2	Output: 1 2 4	Output: 1 2
Here, after the second iteration, the loop breaks.	Here, the third iteration is skipped.	Here, the third iteration is skipped and the loop breaks in the fourth iteration.

Remember that in case of nested loops (a loop inside another loop), the break or continue keywords will only apply on the loop these keywords are directly in, not on the outer loops.

8.6 Nested Loops

Let us understand the concept of understanding nested loops using a code example.

Print the number of divisors of all the numbers ranging from 1 to 10.

```
Output:
1 has 1 divisor(s)
2 has 2 divisor(s)
3 has 2 divisor(s)
4 has 3 divisor(s)
5 has 2 divisor(s)
6 has 4 divisor(s)
7 has 2 divisor(s)
8 has 4 divisor(s)
9 has 3 divisor(s)
10 has 4 divisor(s)
```

```
class DivisorCount {
    public static void main(String[] args) {
        for (int num=1; num<=10; num++){
            int div=0;
            for (int i=1; i<=num; i++){
                if (num%i==0){
                    div++;}
            }
            System.out.println(num+ " has "+div+" divisor(s)");
        }
    }
}
```

Explanation:
If we had to find the number of divisors of one number, we could accomplish this with just one loop. However, in this case we are told to do find the number of divisors of numbers from 1 to 10.
Therefore, we have two repetitive tasks here that are dependent on each other.

- The inner loop is used to count the number of divisors
- The outer loop is used to repeat the inner loop from numbers ranging from 1-10.

Now, let us understand how **break** and **continue** keywords work in a nested loop.

```
class A {
    public static void main(String[] args) {
        for (int x=1; x<=5; x++){
            for (int i=1; i<=x; i++){
                if (x==1){
                    continue;
                }
                if (x==4){
                    break;
                }
                System.out.println(i);
            }
        }
    }
}
```

Output:

```
1
2
1
2
3
1
2
3
4
5
```

Here, if you notice closely, having the break or continue keywords in the inner loop affected the inner loop and not the outer loop.

8.7 Definite vs Indefinite Repetition

When we know exactly how many times the loop will run (Definite), we can solve the problem running the loop a fixed number of times. If the number of iterations is unknown or may vary (Indefinite), then we may use an infinite loop with break statement.

For example, suppose you have to keep on taking inputs from the user until they enter 0. Now, there is no guarantee when the user will enter 0. The user could enter 0 in the very first iteration, or even in the 100th iteration. So, you have to run your loop indefinitely and break it when the condition is fulfilled. Here are a few examples to help you understand when to use definite and indefinite loops.

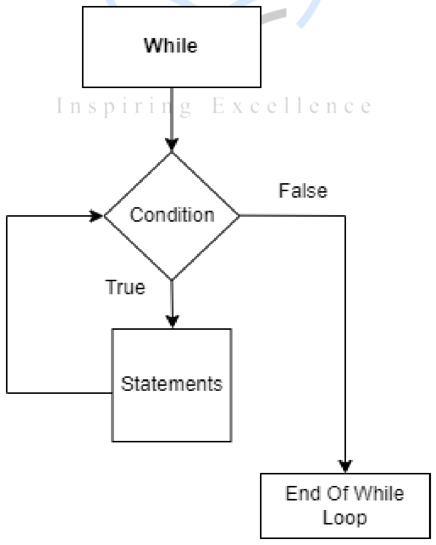
Definite	**Indefinite**		
Print all the even numbers within 1 to 10.	Print the first five numbers divisible by 3 within 1 to 25.	Keep taking user inputs until the user enters an even positive number. Then print that number.	

for (int i=1; i<=10; i++){
    if (i%2==0){
        System.out.println(i);
        }
}
	int num=0;
	for (int i=1; i<=25; i++){
	    if (i%3==0){
	        System.out.println(i);
	        num++; }
	    if (num==5){
	        break;}
	}
		Scanner sc= new Scanner(System.in);
		while(true){
		        int num= sc.nextInt();
		        if (num%2==0 && num>0){
		            System.out.println(num);
		            break;
		        }
		}	

Output: 2 4 6 8 10	Output: 3 6 9 12 15	Input: 5 0 -4 6	Output: 6

8.8 FLOWCHARTS

The following flowchart is what a flowchart of a loop looks like. After encountering a while loop, the condition is checked. If the condition satisfies, we enter the loop. We keep on iterating and executing the statements inside the loop as long as the condition satisfies. We break out of the loop as soon as the condition is dissatisfied.



Now we shall look at a few problem examples and their corresponding flowcharts.

8.9 WORKSHEET

Flowchart Problems

A. Draw the flowchart of a program that calculates the sum of all the numbers from 1 to 100 using a while loop.

B. Draw the flowchart of a program that prompts the user to enter a password. Keep asking for the password until the user enters the correct password "abc123". Once the correct password is entered, display a message "Access granted!".

C. Draw the flowchart of a program that prompts the user to enter a positive integer and then prints all the even numbers from 2 up to and including that number.

D. Draw the flowchart of a program that prompts the user to enter a series of numbers. Continue reading numbers until the user enters a negative number. Then, calculate and display the average of all the positive numbers entered.

Coding problems

A. Write a program that asks the user to enter a series of numbers and calculates their average. Display the average to the user.

B. Write a program that prompts the user to enter a positive integer and prints the factorial of that number.

C. Write a program that prints the multiplication table for a given number from 1 to 10.

D. Write a program that prints the ASCII values and characters for all uppercase letters (A-Z) using a loop.

E. Write a program that prompts the user to enter a positive integer and checks if it is a prime number. Display an appropriate message indicating whether the number is prime or not.

F. Write a program that prompts the user to enter a positive integer and checks if it is a perfect number. Display an appropriate message indicating whether the number is perfect or not.

G. Write a program that prompts the user to enter a positive integer and checks if it is a palindrome number. A number is a palindrome if it reads the same forwards and backwards. Display an appropriate message indicating whether the number is a palindrome or not.