Iteration statements are statements which appear in the source code **only once**, but it **execute many times**.

Such kind of statements are called **loops**.

Almost all the programming languages support looping instructions.

```java
a = 15;  
b = a / 10;  
c = a + b;

if ( a>=c5 )
    a = a + 4;
else
    a = a - 7;

while ( a>=5 )
{
    b = 9 + a;  
    a --;
}  // while
```
Java has three kinds of iteration statements:

- **WHILE** loop
- **FOR** loop
- **DO . . . WHILE** loop

Iteration (repetition) statements cause Java to execute one or more statements as long as a condition exists. Each repetition statement requests:

1. a **control variable**/loop counter
2. the **initial value** of the control variable
3. the **increment** (**decrement**) by which the control variable is modified each time through the loop
4. the **loop continuation condition** that determines if looping should continue.
Basic elements of iterations

/* example with the while statement
   prints numbers from 1 through 10 */
public class While_Test1
{
  public static void main(String[] args)
  {
      int counter = 1;
      while (counter <= 10)
      {
          System.out.print(counter);
          counter++;
      }
      System.out.println();
  }
} // class
Java Loop structures

Do-while structure

While structure
while (loop-continuation-condition) {
    // loop-body;
    Statement(s);
}

int count = 0;
while (count < 100) {
    System.out.println("Welcome to Java!");
    count++;
} // while

(A) Loop Continuation Condition?
   true
     Statement(s) (loop body)
   false

(B) (count < 100)?
   true
     System.out.println("Welcome to Java!");
     count++;
   false
We designate zero to be a **sentinel** (ץקייב) value that indicates the end of the input.

A **sentinel** must always be outside the normal range of values entered.

```java
int sum=0,num,count=0; // sum of series, input variable and loop counter
double avg; // average of series
System.out.println("enter an integer (0 to quit) :");
num = reader.nextInt();
while (num != 0)
{
    count++;
    sum + = num;
    System.out.println("enter an integer (0 to quit) :");
    num = reader.nextInt();
}
// while
avg = (double)sum/count;
System.out.println("The average is : "+avg);
```

Casting Sentinel value of 0 to terminate loop. Sentinel value is not counted.
While loop example 1

• Input:
  – Two integers – num1 and num2

• Output:
  – How many times num1 contains num2
    This is the result of the integer division num1/num2

• Note:
  – Do not use the division operator ( / )!
public class Ex1While
{
    static Scanner reader = new Scanner(System.in);
    public static void main(String[ ] args)
    {
        int res=0; // help variable
        System.out.println("enter two integers : ");
        int num1=reader.nextInt();
        int num2=reader.nextInt();
        while ( (res+1) * num2 <= num1)
        {
            res ++;
            System.out.println("num1 contains num2 : " +res+" times");
        }
    }
} // class
public class Ex2While
{
    static Scanner reader = new Scanner(System.in);
    public static void main(String[] args)
    {
        int res=0; // ?
        System.out.println("enter a positive number: ");
        int num = reader.nextInt();
        while ( num>0) {
            res+= num % 10;
            num/= 10;
        } // while
        System.out.println("res= "+res);
    } // main
} //class Ex2While
Infinite loops

• It is the programmer’s responsibility to ensure that the condition of a loop will eventually become false. If it doesn’t, the loop body will execute forever.
• This situation, called an infinite loop (לולאה אין סופית).

Example 1:

```c
int count=1;
while (count != 50)
  count += 2;
```

This loop will never terminate because `count` will never equal 50.

Example 2:

```c
double num=2.0;
while (num != 0.0)
  num = num - 0.1;
```

This loop will never terminate because `num` will never have a value exactly equal to 0.0.
FOR loop statement

- Equivalent to while... Any for loop can be converted to while loop and vice versa.

- If we want to perform something for a predefined number of times, **better use for**.

- If we just wait for something to happen (not after a certain number or iterations), **better use while**.

- The for loop has **three expressions** that are contained within parentheses and **separated with a semicolon**.
FOR loop - structure

program statements before the for loop...

for (Initialization ; Conditional ; Iteration )
{
    for loop header
    for loop body
}

program statements after for loop...
FOR loop - order of execution

Three kinds of for loop header expressions

● Before the loop begins the first part of the header, called initialization, is executed.
● The second part of the header is the boolean condition, which is evaluated before the loop body. If true, the body is executed.
● The iteration part is executed after each iteration of the loop.

Initialization is executed only once

For loop body

iteration

conditional

false

true

initialization
This program section reads an integer and computes its factorial (n!).

```java
int fact = 1; // factorial
System.out.println("enter an integer : ");
int n = reader.nextInt(); // input variable
for (int i = 1; i <= n; i++)
    fact *= i;
System.out.println("factorial is :" + fact);
```

A **trace table** is a technique used to test programs.

**Trace table** (טבלת מעקב)

<table>
<thead>
<tr>
<th>n</th>
<th>i</th>
<th>i&lt;=n</th>
<th>fact</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>T</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>T</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>F</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

**Before loop execution**

input: 3
public static void main(String[ ] args)
{
    int res = 0; // sum of digits
    System.out.println("enter a positive number:");
    int num = reader.nextInt(); //num=123

    while ( num > 0 ) {
        res += num % 10;
        num /= 10;
    } // while

    System.out.println("res= "+res);
} // main
This program section checks if the input integer is the prime number

```java
int prime = 1;  // help variable
System.out.println(" enter an integer ");
int x = reader.nextInt();  // input variable
if (x>3)
{
    for (int i = 2; i < x && prime == 1; i++)
        if (x%i == 0)
            prime = 0;
    switch (prime)
    {
        case 1:
            System.out.println(x+" is prime number ");
            break;
        case 0:
            System.out.println(x+" is not prime number ");
            break;
    } // switch
} // if (x>3)
else
    System.out.println(x+" is prime number ");
```
Do ...while loop statement

do
{
    statement(s)
}
while (expression) ;

• Similar to while loops
  – Except the condition is evaluated after the loop body. The condition is written at the end of the loop to indicate that it is not evaluated until the loop body is executed.
  – The loop body is always executed at least once, even if the expression is never true.
Do ...while loop execution

Loop Continuation Condition?

true

Statement(s)
(loop body)

false
Do ...while loop example 1

Waiting for legal input

// program statement before the do...while loop

System.out.print("Please, enter a positive number: ");
do {
    int num = reader.nextInt();
    if (num <= 0)
        System.out.println("Input error! Try again ");
} while (num <= 0);

// program statements after the do...while loop
This program section reads an integer and reverses its digit mathematically.

```java
int reversNum = 0; // reversed number
System.out.println("enter an integer : ");
int num = reader.nextInt(); // input variable
do
{
    int lastDigit = num % 10;
    reversNum = (reversNum*10 )+ lastDigit;
    num = num / 10;
}
while (num > 0);
System.out.println(" That number reversed is "+reversNum);
```
This program section reads an integer and …?

```java
int x = 1; // ?
System.out.println("enter an integer : ");
int num = reader.nextInt(); // input variable
do
{
   x *= num;
} while ( -- num >= 1);
System.out.println(" x= “ + x );
```
The Fibonacci sequence is named after Leonardo of Pisa, who was known as Fibonacci.

In mathematics, the Fibonacci numbers or Fibonacci series or Fibonacci sequence are the numbers in the following integer sequence:

\[
\begin{align*}
n: & \quad 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ \ldots \\
F(n): & \quad 0 \ 1 \ 1 \ 2 \ 3 \ 5 \ 8 \ \ldots
\end{align*}
\]

\[
F_n = \begin{cases} 
0 & \text{if } n = 0 \\
1 & \text{if } n = 1 \\
F_{n-1} + F_{n-2} & \text{if } n > 1
\end{cases}
\]
public static void main(String[] args) {
    int n; // number of element in the series
    do {
        System.out.print(" Enter an positive integer => ");
        n = reader.nextInt();
    } while (n < 0);
    int Fn2 = 0;
    int Fn1 = 1;
    int Fn = 0;
    if (n == 1) 
        Fn = 1;
    for (int ind=2; ind <= n; ind++)
    {
        Fn = Fn1 + Fn2;
        Fn2 = Fn1;
        Fn1 = Fn;
    } // for
    System.out.println("Fib(" + n + ")= " + Fn);
} // main
Nested loops

• The body of loop can contain another loop. This situation is called a nested loop (לולאות מקוננות).

• Note, that for each iteration of outer loop, the inner loop executes completely.

The next program section reads the integer number \( n \), real number \( x \) and calculates the sum of the mathematical series:

\[
1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!}
\]
double sum=1.0; // sum of series
int f; // n factorial
double h; // x degree
int i= 1; //loop counter
System.out.print( "enter an integer : " );
int n = reader.nextInt();
System.out.print( "enter real number : " );
double x = reader.nextDouble();

while (i <= n)
{
    f = 1;
    h = 1.0;
    for (int j = 1; j <= i ;j++)
    {
        f*=j; // calculates factorial
        h*=x; // calculates x degree
    }
    sum+=h/f;
    i++;
}
System.out.print( "sum = : " +sum );
A jump statement transfers control to another part of the program. There are two kinds of jump statements: `break` and `continue`.

- When `break` is encountered, the loop is exited regardless of whether the condition is still true.
  
The `break` statement tells Java to exit a code block (loop body) defined by opening and closing braces and used in a loop.
  
The program then continues to run from the first line after the while/for/do…while body’s loop.
  
  If called within a nested loop, `break` breaks out of the inner loop only.

- When `continue` is encountered, the rest of the loop is ignored.
  
The program then continues to run from the beginning of the loop.
Using break statement

program statement before the loop ...

\[
\text{loop}(\text{expression})
\]
\[
\{\ \\
\text{.} \\
\text{.} \\
\text{break;} \\
\text{.} \\
\} // end loop
\]

program statement after the loop ...

program statement before the nested loops ...

\[
\text{loop1}(\text{expression1}) \{ \ \\
\text{.} \\
\text{loop2}(\text{expression2}) \{ \ \\
\text{.} \\
\text{break;} \\
\text{.} \\
\} // end loop2
\]

// continue with loop1
}\ // end loop1

program statement after the nested loops ...


Using continue statement

program statement before the loop…

```c
loop ( expression )
{
    continue;
}
```

Ignore executing rest of the loop’s body in this iteration

program statements after the loop…
// continue with the program
break and continue examples

Brake out of loop at  \text{i}=5

```java
for (int i = 1; i <= 10; i++)
{
    if (i == 5)
        break;
    System.out.print (i + " ");
} // for
System.out.println ( ) ;
```

Output : 1 2 3 4

Skip printing the value 5

```java
for (int i = 1; i <= 10; i++)
{
    if (i == 5)
        continue;
    System.out.print (i + " ");
} // for
System.out.println ( ) ;
```

Output : 1 2 3 4 6 7 8 9 10
Caution!

- Similarly, the following **while loop** is wrong:

```java
int i = 0;
while (i < 10);
{
    System.out.println("i is " + i);
    i++;
}
```

- In the case of the **do…while loop**, the following semicolon is needed to end the loop.

```java
int i = 0;
do {
    System.out.println("i is " + i);
    i++;
} while (i < 10);
```