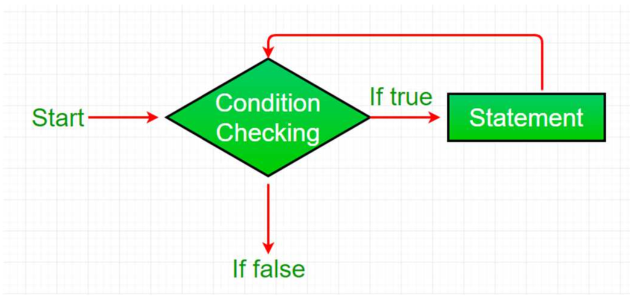
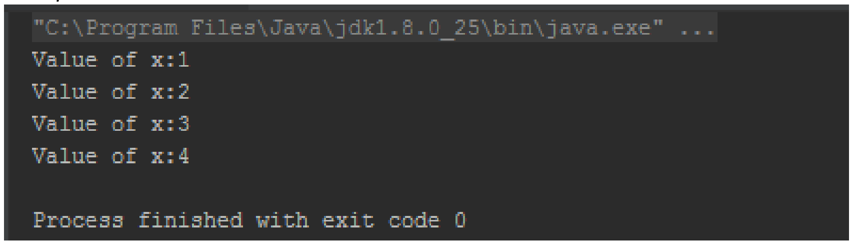


Once the condition is evaluated to true, the statements in the loop body are executed.
When the condition becomes false, the loop terminates which marks the end of its life cycle.

JAVA PROGRAM TO ILLUSTRATE WHILE LOOP

```
package sample;

public class WhileLoopDemo {
    public static void main(String args[])
    {
        int x = 1;

        // Exit when x becomes greater than 4
        while (x <= 4)
        {
            System.out.println("Value of x:" + x);

            // Increment the value of x for
            // next iteration
            x++;
        }
    }
}
```

Output:

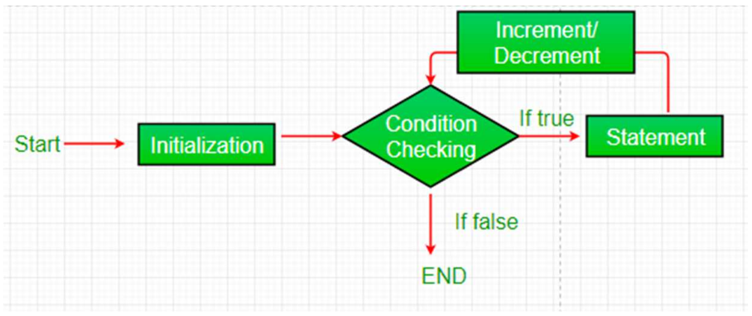


```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
Value of x:1
Value of x:2
Value of x:3
Value of x:4

Process finished with exit code 0
```

for loop: a for statement consumes the initialization, condition and increment/decrement in one line.

Syntax:

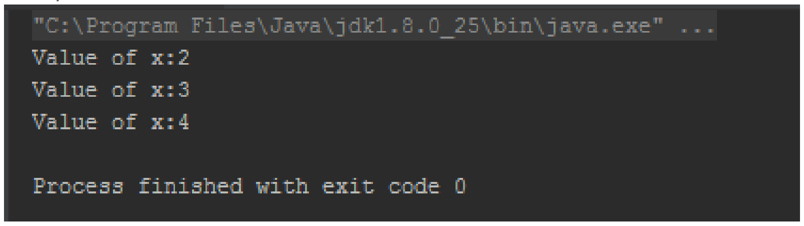
```
for (initialization condition; testing condition;
increment/decrement)
{
    statement(s)
}
```



1. **Initialization condition:** Here, we initialize the variable in use. It marks the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.
2. **Testing Condition:** It is used for testing the exit condition for a loop. It must return a Boolean value.
3. **Statement execution:** Once the condition is evaluated to true, the statements in the loop body are executed.
4. **Increment/ Decrement:** It is used for updating the variable for next iteration.
5. **Loop termination:** When the condition becomes false, the loop terminates marking the end of its life cycle.

```
package sample;
// Java program to illustrate for loop.
public class ForLoopDemo {
    public static void main(String args[])
    {
        // for loop begins when x=2
        // and runs till x <=4
        for (int x = 2; x <= 4; x++)
            System.out.println("Value of x:" + x);
    }
}
```

Output

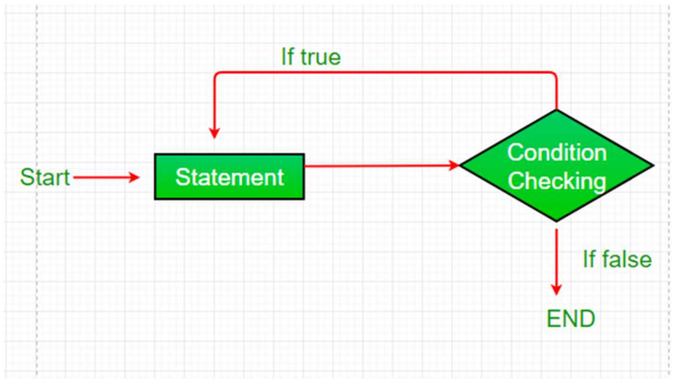


```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
Value of x:2
Value of x:3
Value of x:4

Process finished with exit code 0
```

do while: do while loop is similar to while loop with only difference that it checks for condition after executing the statements

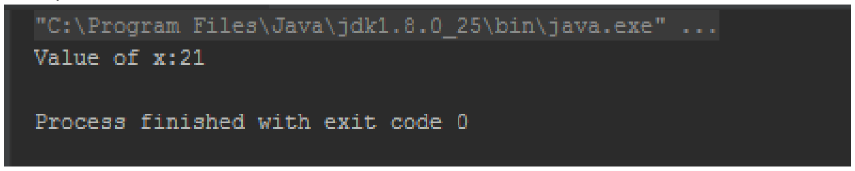
Syntax:

```
do
    {
        statements..
    }
while (condition);
```



```
package sample;
// Java program to illustrate do-while loop
public class DowhileloopDemo {
    public static void main(String args[])
    {
        int x = 21;
        do
        {
            // The line will be printed even
            // if the condition is false
            System.out.println("Value of x:" + x);
            x++;
        }
        while (x < 20);
    }
}
```

Output



```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
Value of x:21

Process finished with exit code 0
```

DECISION MAKING - IF | IF-ELS+E | NESTED-IF | IF-ELSE-IF | SWITCH-CASE |JUMP

Decision making in programming is similar to decision making in real life. In programming also we face some situations where we want a certain block of code to be executed when some condition is fulfilled.

if: if statement is the most simple decision making statement. It is used if a certain condition is true then a block of statement is executed otherwise not.

Syntax:

```
if(condition)
        {
        // Statements to execute if
        // condition is true
        }
```

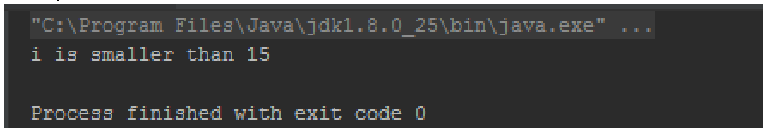
if-else: The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. We can use the else statement with if statement to execute a block of code when the condition is false.

Syntax:

```
if (condition)
        {
        // Executes this block if
        // condition is true
        }
        else
        {
        // Executes this block if
        // condition is false
        }
```

```
package sample;
// Java program to illustrate if-else statement
public class IfElseDemo {
    public static void main(String args[])
    {
        int i = 10;

        if (i < 15)
            System.out.println("i is smaller than 15");
        else
            System.out.println("i is greater than 15");
    }
}
```

Output



```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
i is smaller than 15

Process finished with exit code 0
```

nested-if:

A nested if is an if statement that is the target of another if or else. Nested if statements means an if statement inside an if statement.

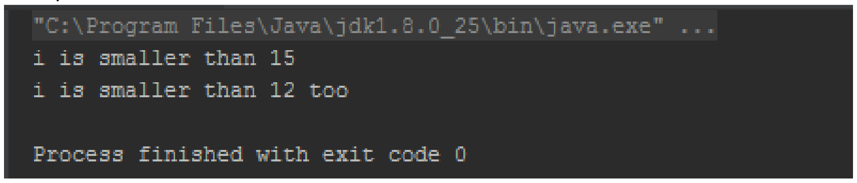
Nested if statements means an if statement inside an if statement.
i.e, we can place an if statement inside another if statement.

Syntax:

```
if (condition1)
        {
        // Executes when condition1 is true
        if (condition2)
        {
        // Executes when condition2 is true
        }
        }
```

```
package sample;
// Java program to illustrate nested-if statement
public class NestedIfDemo {
    public static void main(String args[])
    {
        int i = 10;

        if (i == 10)
        {
            // First if statement
            if (i < 15)
                System.out.println("i is smaller than 15");

            // Nested - if statement
            // Will only be executed if statement above
            // it is true
            if (i < 12)
                System.out.println("i is smaller than 12 too");
            else
                System.out.println("i is greater than 15");
        }
    }
}
```

Output



```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
i is smaller than 15
i is smaller than 12 too

Process finished with exit code 0
```

if-else-if ladder:

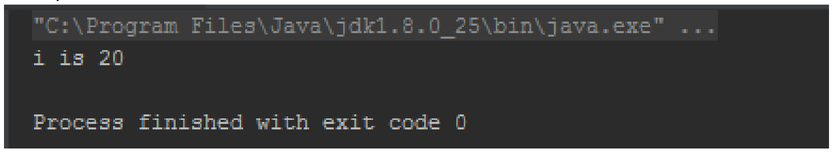
The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

Syntax:

```
if (condition)
        statement;
        else if (condition)
        statement;
        .
        .
        else
        statement;
```

```
package sample;
// Java program to illustrate if-else-if ladder
public class IfelseifDemo {
    public static void main(String args[])
    {
        int i = 20;

        if (i == 10)
            System.out.println("i is 10");
        else if (i == 15)
            System.out.println("i is 15");
        else if (i == 20)
            System.out.println("i is 20");
        else
            System.out.println("i is not present");
    }
}
```

Output



```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
i is 20

Process finished with exit code 0
```

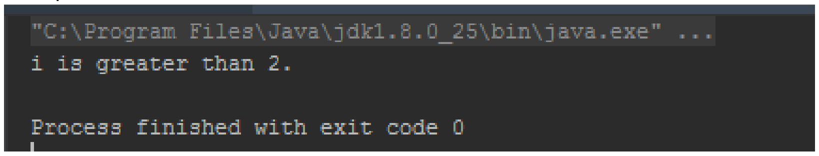
switch-case The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.

Syntax:

```
switch (expression)
        {
        case value1:
        statement1;
        break;
        case value2:
        statement2;
        break;
        .
        .
        case valueN:
        statementN;
        break;
default:
        statementDefault;
        }
```

```
package sample;
// Java program to illustrate switch-case
public class SwitchCaseDemo {
    public static void main(String args[])
    {
        int i = 9;
        switch (i)
        {
            case 0:
                System.out.println("i is zero.");
                break;
            case 1:
                System.out.println("i is one.");
                break;
            case 2:
                System.out.println("i is two.");
                break;
            default:
                System.out.println("i is greater than 2.");
        }
    }
}
```

Output:



```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
i is greater than 2.

Process finished with exit code 0
```

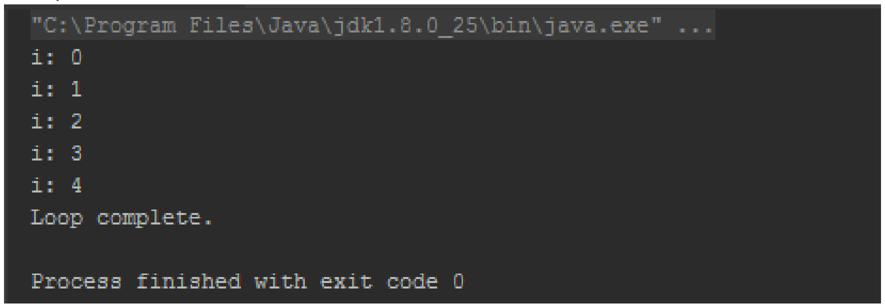
jump: Java supports three jump statement: **break, continue** and **return**. These three statements transfer control to other part of the program.

BREAK: In Java, break is majorly used for:

- Terminate a sequence in a switch statement (discussed above).
- To exit a loop.

```java
package sample;
// Java program to illustrate using
// break to exit a loop
public class BreakLoopDemo {
    public static void main(String args[])
    {
        // Initially loop is set to run from 0-9
        for (int i = 0; i < 10; i++)
        {
            // terminate loop when i is 5.
            if (i == 5)
                break;

            System.out.println("i: " + i);
        }
        System.out.println("Loop complete.");
    }
}
```

Output



```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
i: 0
i: 1
i: 2
i: 3
i: 4
Loop complete.

Process finished with exit code 0
```

Using break as a Form of Goto

Java uses label. A Label is use to identifies a block of code.

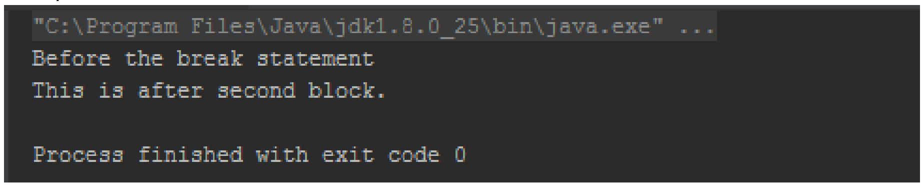
Syntax:

```
label:
        {
        statement1;
        statement2;
        statement3;
        .
        .
        }
```

Now, break statement can be used to jump out of target block.

```java
package sample;
// Java program to illustrate using break with goto
public class BreakLabelDemo {
    public static void main(String args[])
    {
        boolean t = true;

        // label first
        first:
        {
            // Illegal statement here as label second is not
            // introduced yet break second;
            second:
            {
                third:
                {
                    // Before break
                    System.out.println("Before the break statement");

                    // break will take the control out of
                    // second label
                    if (t)
                        break second;
                    System.out.println("This won't execute.");
                }
                System.out.println("This won't execute.");
            }

            // Third block
            System.out.println("This is after second block.");
        }
    }
}
```

Output

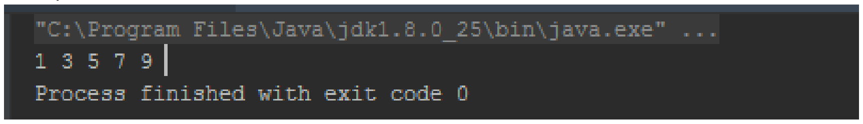


```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
Before the break statement
This is after second block.

Process finished with exit code 0
```

Continue: Sometimes it is useful to force an early iteration of a loop. That is, you might want to continue running the loop but stop processing the remainder of the code in its body for this particular iteration.

```java
package sample;
// Java program to illustrate using
// continue in an if statement

public class ContinueDemo {
    public static void main(String args[])
    {
        for (int i = 0; i < 10; i++)
        {
            // If the number is even
            // skip and continue
            if (i%2 == 0)
                continue;

            // If number is odd, print it
            System.out.print(i + " ");
        }
    }
}
```

Output

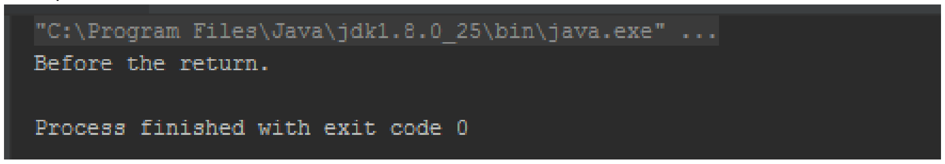


```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
1 3 5 7 9
Process finished with exit code 0
```

Return: The return statement is used to explicitly return from a method.

```
package sample;
// Java program to illustrate using return
public class Return {
    public static void main(String args[])
    {
        boolean t = true;
        System.out.println("Before the return.");

        if (t)
            return;

        // Compiler will bypass every statement
        // after return
        System.out.println("This won't execute.");
    }
}
```

Output



```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
Before the return.

Process finished with exit code 0
```

COMMENTS

Proper use of comments makes maintenance easier and finding bugs easily. Comments are ignored by the compiler while compiling a code.
In Java there are three types of comments:

1. *Single – line comments.*
2. *Multi – line comments.*
3. *Documentation comments.*

Single-line Comments

Syntax:

```
//Comments here( Text in this line only is considered as comment )
```

```
package sample;
//Java program to show single line comments
public class Comment {
    public static void main(String args[])
    {
        // Single line comment here
        System.out.println("Single line comment above");
    }
}
```

Multi-line Comments

Syntax:

```
/*Comment starts
continues
continues
.
.
.
Comment ends*/
```

```
package sample;
//Java program to show multi line comments
public class MultiLineComments {
    public static void main(String args[])
    {
        System.out.println("Multi line comments below");
      /*Comment line 1
      Comment line 2
      Comment line 3*/
    }
}
```

Documentation Comments

It helps to generate a documentation page for reference

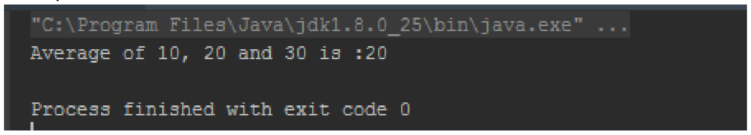
Syntax:

```
/**Comment start
 *
 *tags are used in order to specify a parameter
 *or method or heading
 *HTML tags can also be used
 *such as <h1>
 *
 *comment ends*/
```

Available tags to use: @author {@code} {@docRoot} @deprecated @exception {@link} @param @return @throws {@value} @version

```
package sample;
//Java program to illustrate frequently used
// Comment tags

/**
 * <h1>Find average of three numbers!</h1>
 * The FindAvg program implements an application that
 * simply calculates average of three integers and Prints
 * the output on the screen.
 *
 * @author Pratik Agarwal
 * @version 1.0
 * @since 2017-02-18
 */

public class FindAvg {
    /**
     * This method is used to find average of three integers.
     * @param numA This is the first parameter to findAvg method
     * @param numB This is the second parameter to findAvg method
     * @param numC This is the second parameter to findAvg method
     * @return int This returns average of numA, numB and numC.
     */
    public int findAvg(int numA, int numB, int numC)
    {
        return (numA + numB + numC)/3;
    }

    /**
     * This is the main method which makes use of findAvg method.
     * @param args Unused.
     * @return Nothing.
     */

    public static void main(String args[])
    {
        FindAvg obj = new FindAvg();
        int avg = obj.findAvg(10, 20, 30);

        System.out.println("Average of 10, 20 and 30 is :" + avg);
    }
}
```

Output



```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
Average of 10, 20 and 30 is :20

Process finished with exit code 0
```