The while Loop:

A while loop is a control structure that allows you to repeat a task a certain number of times.

**Syntax:**

The syntax of a while loop is:

|  |
| --- |
| while(Boolean\_expression) { //Statements }  |

When executing, if the *boolean\_expression* result is true then the actions inside the loop will be executed. This will continue as long as the expression result is true.

Here key point of the *while* loop is that the loop might not ever run. When the expression is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

In a while loop the loop variable has to be initialized before the while statement. Inside the loop the variable must be incremented or decremented so as to take the while loop to reach the condition given in the Boolean expression.

|  |  |
| --- | --- |
| 123456789101112121415 | public class loopsjava{ public static void main(String args[]) { int sum=0; int n=5; int i=1; while(i<=n) { sum=sum+i; System.out.println("the value of sum is = "+sum+ " when i is : "+i ); i++; }}} |

The output is given below:

|  |
| --- |
| the value of sum is = 1 when i is : 1the value of sum is = 3 when i is : 2the value of sum is = 6 when i is : 3the value of sum is = 10 when i is : 4the value of sum is = 15 when i is : 5 |

In the program line 7 initializes the loop variable i=1;

In line 8 the while loop gives the Boolean expression i<=n; That is, the loop will get executed till the value in i is less than n.(i.e till the condition is true) when the i becomes greater than n the loop condition becomes false and loop stops and the statement which follows the outside of the loop started to execute.

**The for Loop:**

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

A for loop is useful when you know how many times a task is to be repeated.

**Syntax:**

The syntax of a for loop is:

|  |
| --- |
| for(initialization; Boolean\_expression; update) { //Statements }  |

Example

|  |  |
| --- | --- |
| 1234567891011121314 | public class loopsjava{public static void main(String args[]){int sum=0;int n=5;for(int i=1;i<=n;i++) {sum=sum+i;System.out.println("the value of sum is = "+sum+ " when i is : "+i );}}} |

The for loop in line 7 the initialization is done by the statement ***int i=1;***

The Boolean expression is ***i<=n;*** this shows the condition on which the loop has to run. The loop update statement ***i++;*** This statement will take the loop step by step to reach the final condition given by Boolean-expression.

The lines 9 and 10 within for loop are executed till the Boolean-expression condition is satisfied.

The output of the above program is given below:

|  |
| --- |
| the value of sum is = 1 when i is : 1the value of sum is = 3 when i is : 2the value of sum is = 6 when i is : 3the value of sum is = 10 when i is : 4the value of sum is = 15 when i is : 5 |

**The do...while Loop:**

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

**Syntax:**

The syntax of a do...while loop is:

|  |
| --- |
| do { //Statements }while(Boolean\_expression);  |

|  |  |
| --- | --- |
| 123456789101112131415 | public class loopsjava{ public static void main(String args[]) { int sum=0; int n=5; int i=1; do { sum=sum+i; System.out.println("the value of sum is = "+sum+ " when i is : "+i ); i++; }while(i<=n);}} |

**The break Keyword:**

The *break* keyword is used to stop the entire loop. The break keyword must be used inside any loop or a switch statement.

The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

**The continue Keyword:**

The *continue* keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.

 In a for loop, the continue keyword causes flow of control to immediately jump to the update statement.

 In a while loop or do/while loop, flow of control immediately jumps to the Boolean expression.

**Syntax:**

The syntax of a continue is a single statement inside any loop:

|  |
| --- |
| continue;  |

**The if Statement:**

An if statement consists of a Boolean expression followed by one or more statements.

**Syntax:**

The syntax of an if statement is:

|  |
| --- |
| if(Boolean\_expression) { //Statements will execute if the Boolean expression is true }  |

**The if...else Statement:**

An if statement can be followed by an optional *else* statement, which executes when the Boolean expression is false.

**Syntax:**

The syntax of a if...else is:

|  |
| --- |
| if(Boolean\_expression){ //Executes when the Boolean expression is true }else{ //Executes when the Boolean expression is false }  |

**The if...else if...else Statement:**

An if statement can be followed by an optional *else if...else* statement, which is very usefull to test various conditions using single if...else if statement.

**Syntax:**

The syntax of a if...else is:

|  |
| --- |
| if(Boolean\_expression 1){ //Executes when the Boolean expression 1 is true }else if(Boolean\_expression 2){ //Executes when the Boolean expression 2 is true }else if(Boolean\_expression 3){ //Executes when the Boolean expression 3 is true }else { //Executes when the one of the above condition is true. }  |

**Nested if...else Statement:**

It is always legal to nest if-else statements. When using if , else if , else statements there are few points to keep in mind.

 An if can have zero or one else's and it must come after any else if's.

 An if can have zero to many else if's and they must come before the else.

 Once an else if succeeds, none of he remaining else if's or else's will be tested.

**Syntax:**

The syntax for a nested if...else is as follows:

|  |
| --- |
| if(Boolean\_expression 1){ //Executes when the Boolean expression 1 is true if(Boolean\_expression 2){ //Executes when the Boolean expression 2 is true } }  |

**The switch Statement:**

A *switch* statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

**Syntax:**

The syntax of switch is:

|  |
| --- |
| switch(expression){ case value : //Statements break; //optional case value : //Statements break; //optional //You can have any number of case statements. default : //Optional //Statements }  |

**Java Methods:**

A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.println method, for example, the system actually executes several statements in order to display a message on the console.

In general, a method has the following syntax:

|  |
| --- |
| modifier returnValueType methodName(list of parameters) { // Method body; }  |

A method definition consists of a method header and a method body. Here are all the parts of a method:

 **Modifiers:** The modifier, which is optional, tells the compiler how to call the method. This defines the access type of the method.

 **Return Type:** A method may return a value. The returnValueType is the data type of the value the method returns. Some methods perform the desired operations without returning a value. In this case, the returnValueType is the keyword **void**.

 **Method Name:** This is the actual name of the method. The method name and the parameter list together constitute the method signature.

 **Parameters:** A parameter is like a placeholder. When a method is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.

 **Method Body:** The method body contains a collection of statements that define what the method does.