# JAVA @17

Statements

# Objective

After completing this session, you will be able to understand,

- Statement
- Selection statement
- Iteration statement
- Transfer statement

# Java Statements and Blocks

What is a Statement?

A statement is a complete instruction terminated by a semi-colon.

Example: Assignment Statement

```
String name="Greeting";
```

What is a Block?

A block is group of statements enclosed in curly brackets.

Example:

```
{
    name="Ramesh";
    age=12;
}
```

**Java executes one statement after the other in the order they are written**

# Java Control Statements

What are Control Statements?

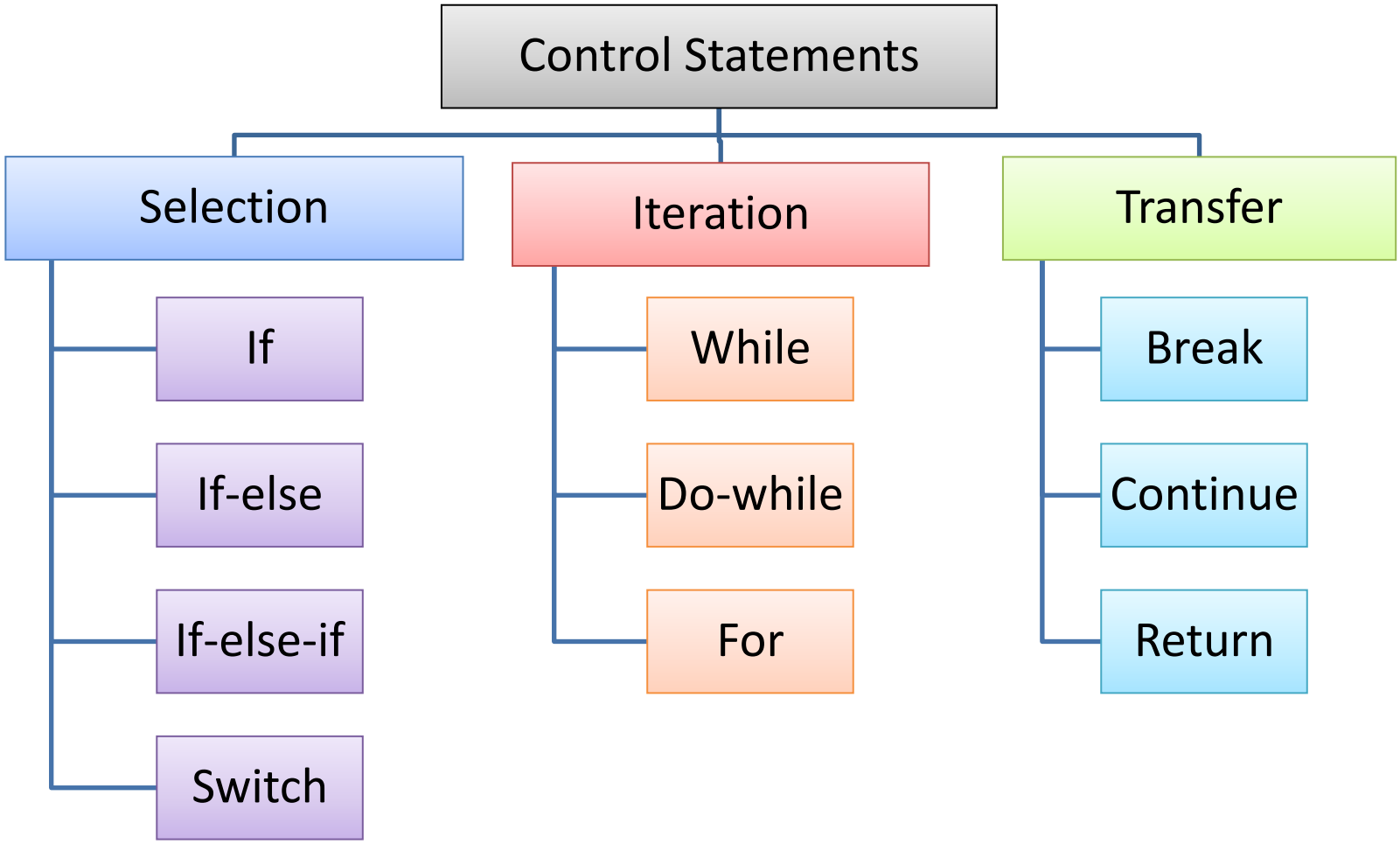
The control statements allows developers to control the flow of program's execution based upon some conditions during run time.

The control statements allows developers to,

*Repetitive execution of statements – Executing a statement 'N' number of times*

*Conditional execution of statements – Execute statements based on some condition.*

# Control Statements - Categories



# Selection Statements

**What are Selection statements?**
Selection statements allow the program to choose different paths of execution based upon the outcome of an conditional expression or the state of a variable.

Java supports the following selection statements

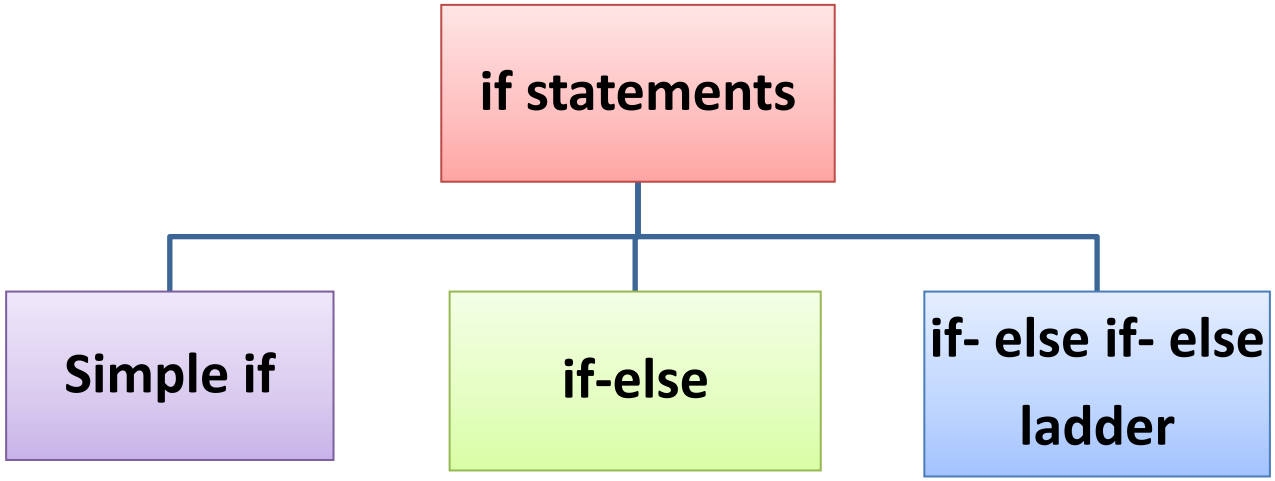
- **if else** statement
- **switch** statement

# Selection Statement - IF

**If Statement:**

The if statements can check **conditions** starting from very simple to quite complex and execute statements.

Conditions are nothing but a single relational expression or the combination of more than one relational expressions with logical operators. If statement comes in **three** different constructs



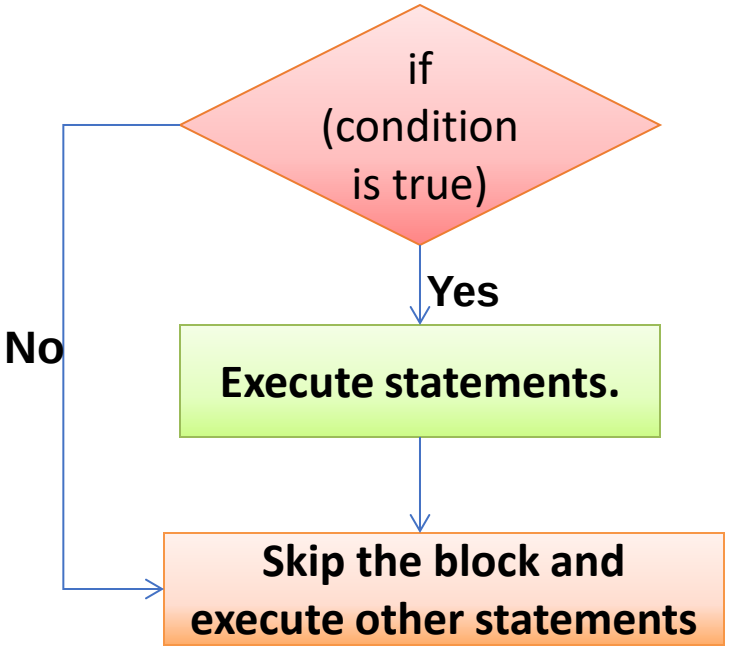
# A simple If Statement

**Features of Simple If statement:**

The simple if statement allows the execution of a **single** statement or a **block** of statements enclosed within curly braces.

The if statement handles a very **simple** situation and executes only when the condition is true otherwise the whole statement body is skipped .

**Illustration:**



# Simple if Statement

**Syntax of Simple If statement:**

**Option I:**

```
if(condition1 ) {
        //statement body;
}
```

**Option II:**

```
if(condition1 && condition2){
        //statement body;
}
```

The condition can be a **single** relational expression or a **combination** of more than one relational expression separated using a logical operators

# if Statement Example

**Examples of a simple If statement**:

**Example1: Without curly braces**

```
int a = 10;
int b = 20;
If(a>b)
  System.out.println("Value of a is greater than b");
```

**Example2: With curly braces**

```
int empId=1201;
int retirementAge=58;
int empAge=36;
if((empAge<retirementAge)  && (empId==1201)){
    System.out.println("Calculate Salary ");
}
```

**If both the condition age and id are satisfied the message will be printed.**

It is a good programming practice to use the curly braces regardless of whether there is one or more statements.

# if-else Statement

**What is an if-else statement?**

| Simple If statement | Else statement for alternative flow | If-else Statement |
|---|---|---|

If-else can handle **two** blocks of code, and only one of those blocks will be executed based on the condition outcome.

**Syntax**

```
if( <condition1> ) {
      statements
}
else{
      statements
}
```

If **condition1** is **satisfied** these statements are executed

If **condition1** is **not satisfied** these statements are executed

# if-else Statement Example

**Example of an if-else statement:**

```
int empId=1201;
int retirementAge=58;
int empAge=36;
if( (empAge<retirementAge)  && (empId==1201) ){
    System.out.println("Calculate Salary ");
    }
else {
    System.out.println("Calculate Pension");
}
```

If both the condition **age** and **id** are satisfied the message will be printed.

If the conditions in if block is not satisfied, this block will be fired.

# if-else if Statement

**What is an if-else if statement?**

**Multiple If-else statements** → **If-else If or ladder if-else Statement**

***If-else if*** can handle **more than two blocks** of code, and only one of those blocks will be executed.

**Syntax**

```
if( <condition1> ) {
        statements
}
else if(<condition2>){
        statements
}
else{
        statements
}
```

If **condition1** is **satisfied** these statements are executed.

If **condition1** is **not satisfied** and **condition2 is satisfied** these statements are executed.

If **both conditions are not satisfied** these statements are executed.

# if-else if Statement Example

```
package com.statements.demo;

 class IfElseIfExample {

   public static void main(String args[]) {
   int month = 4;
   String season;
   if(month == 12 || month == 1 || month == 2)
      season = "Winter";
   else if(month == 3 || month == 4 || month == 5)
      season = "Spring";
   else if(month == 6 || month == 7 || month == 8)
      season = "Summer";
   else if(month == 9 || month == 10 || month == 11)
      season = "Autumn";
   else
      season = "Bogus Month";
   System.out.println("April is in the " + season + ".");
   }
 }
```

If statement

Else If statement

Else statement

# Nested if Statements

**What is a nested if statement?**

The if statement in java can be nested, in other words, an if statement can be present inside another if statement

**Example**:

The discount % of T.V is calculated based on the below criteria

If T.V is LED, check the screen size

If screen size is 32, discount % = 10

If screen size is 46, discount % = 15

If T.V is LCD, discount % = 5

**Nested if statement**
**If the type of TV is LED, then size check is done**

```
public void checkDiscount(){

    String typeOfTV = "LED";
    int sizeofTV = 32;
    int discount;

    if("LED".equals(typeOfTV)){
        if(sizeofTV==32){
            discount = 10;
        }else if (sizeofTV == 46){
            discount = 15;
        }
    }else if("LCD".equals(typeOfTV)){
        discount = 5;
    }
}
```

# Lend a Hand – if else if

1. Create a java class "***NumberCheck***" add a method ***displayBigNumber*** three int parameters "***num1***" ,"***num2***" and "***num3***".
2. The method ***displayBigNumber*** will check and print the biggest of the three numbers in the following format

   **"<result> + is the Biggest Number"**

3. Create a java class "MainProgram" add a main method which will
   - Create a object instance of the NumberCheck.
   - Trigger the method ***displayBigNumber*** by passing values of num1,num2, and num3 as 11,18 and 7.
4. The message needs to be displayed in the console.

# Solution

```
public class Numbercheck {

    void displayBigNumber(int num1,int num2,int num3){

        int biggestNumber;

        if(num1>num2){
            if(num1>num3){
                biggestNumber = num1;
            }else if(num3 > num2){
                biggestNumber = num3;
            }else{
                biggestNumber = num2;
            }
        }else if(num2>num3){
            biggestNumber = num2;
        }else{
            biggestNumber = num3;
        }
        System.out.println(biggestNumber+ " is the biggest number");
    }
}
public class MainProgram {

    public static void main(String[] args) {
         Numbercheck check = new Numbercheck();
         check.displayBigNumber(11, 18, 7);
    }
}
```

# Switches in Real life

In the below illustration the respective switches are used to control the working of the respective electrical appliances.

**Example:** Switch on Fan use the fan switch and so on….

# Switch Statement

Similarly when developing software applications to control the flow of execution in executing a particular block of statements we use the ***switch*** statement.

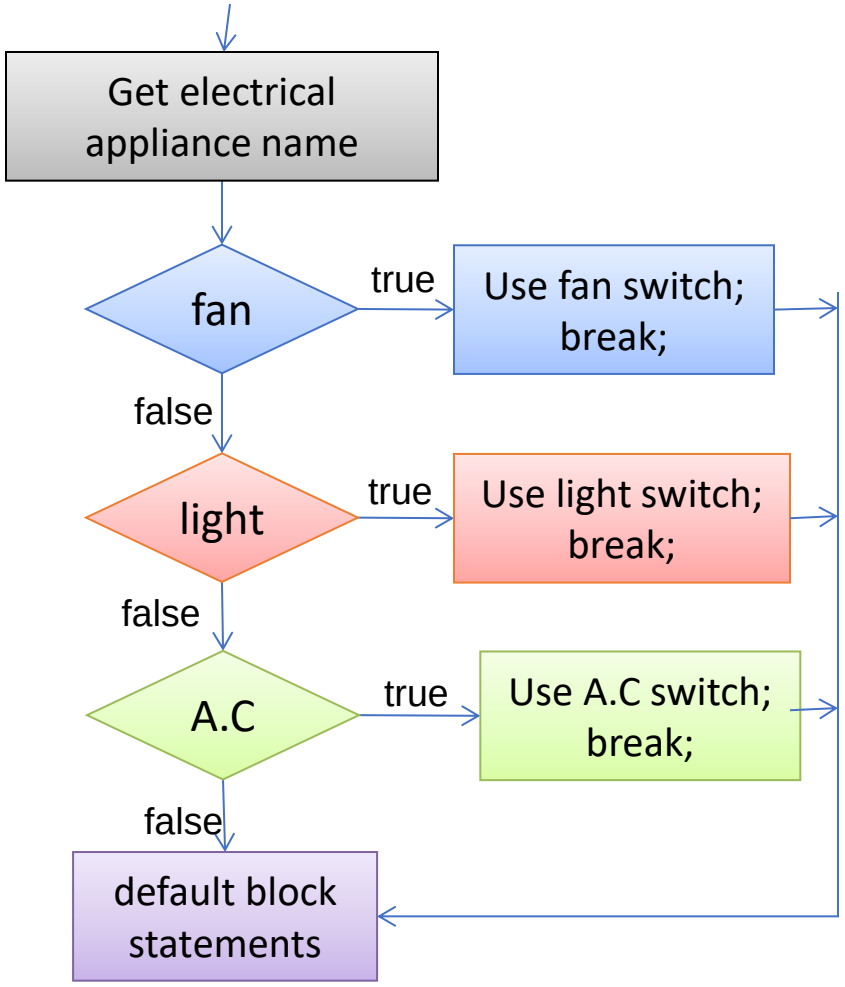
The switch statement allows  to **choose** a **block** of statements to run from a number of option available.

This can also be implemented using nested if-else. So what is the difference. We will see the difference soon.

# How to write Switch Statement

Syntax:

```
switch (expression) {
case value1:
  // statement sequence
  break;
case value2:
  // statement sequence
  break;
case value N:
  // statement sequence
  break;
default:
  // default statements
  break;
}
```



# Switch Statement Example

**Example:**

```
int x=6%2;
switch (x) {
    case 0:
      System.out.println("The value of x is 0." );
      break;
    case 1:
      System.out.println("The value of x is 1." );
      break;
   default:
      System.out.println("The value of x is default.");
      break;
}
```

The argument of switch() must be one of the types byte, short, char, int

There should be no duplicate case labels i.e., the same value cannot be used twice.

# How Switch works?

**Lets see how Switch works without break statements,**

Assume the value of X is 10

```
switch (x){
case 12:
   //Statements
   break;
case 10:
   //Statements
   break;
case 15:
   //Statements
   break;
default:
   //Statements
   break;
}
```

Switch Statement Executed

Case 10 condition passes

Case 10 statements executed

Break statement executed and control goes outside the switch block.

Break statement breaks the execution control flow and control passed outside the switch block.

# How Switch works without break?

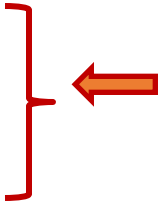
**Lets see how Switch works without break statements,**

```
Assume the value of X is 10
switch (x){
case 12:
   //Statements
case 10:
   //Statements
case 15:
   //Statements
default:
   //Statements
}
```

Switch Statement Executed

Case 10 condition passes

Case 10 statements executed



Since case 10 block does not have break the case 15 and the default block statements will be executed .

After all the cases statements are executed the control goes outside the switch blocks.

# Switch Statement

**Some facts about switch statement:**

- Java first evaluates the *switch expression* and jumps to the case which matches the value of the expression
- Once the correct match is found, all statements from that point are executed till a ***break*** statement is encountered
- Once break statement is encountered, the flow jumps to the statements after the switch structure
- If none of the cases are satisfied, default block is executed. The default block does not have to be at the end of the switch.

# switch Vs if

| If-else | switch |
| --- | --- |
| **This can test expressions based on ranges of values or conditions.**<br><br>**Example: if(a==10 && b=21)** | **This tests expressions based only on a single integer, enumerated value, or String object.**<br><br>**Example: switch(i)**<br>**// where i is an int.** |

Based on the condition to be evaluated developers can either go for **switch or if-else.**

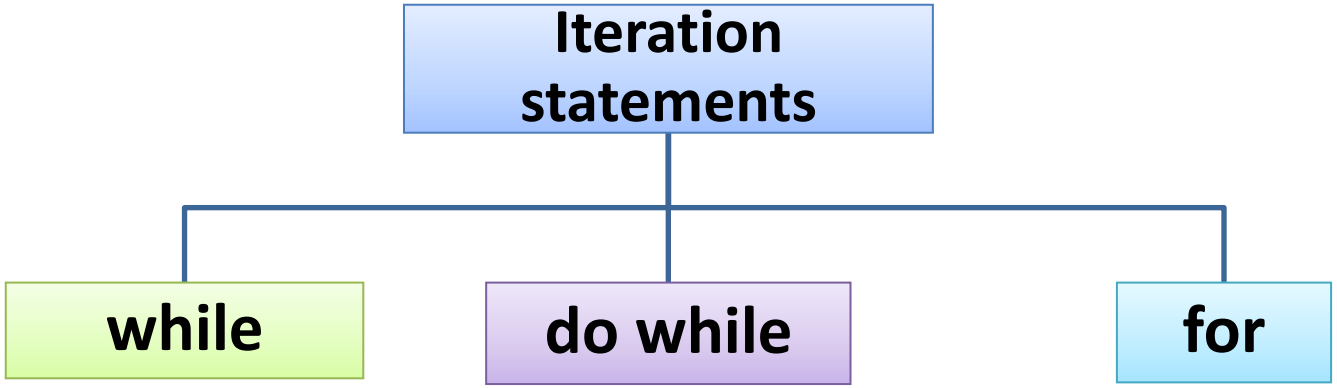
# Iteration Statement

**What are Iteration statements?**

***Iteration Statements*** are used execute a block of statements **repeatedly** as long as a certain **condition** is **true**.

A single relational expression or the combination of more than one relational expression with logical operators are given as conditions.

Java offers three iteration constructs



# While Statement

The while loop is Java's most fundamental iteration statement.

**Simple Problem statement to understand the usage of while statement:**

**John** has to develop a small java program which needs to **print** a welcome **message** as long as the number of guests is greater than zero.

The above problem statement can be easily done using while loop

**Example:**

```
while(countOfGuests>0){
        System.out.println("Welcome to my party");
        countOfGuests--;
}
```

# While Statement

**Facts about while statement:**

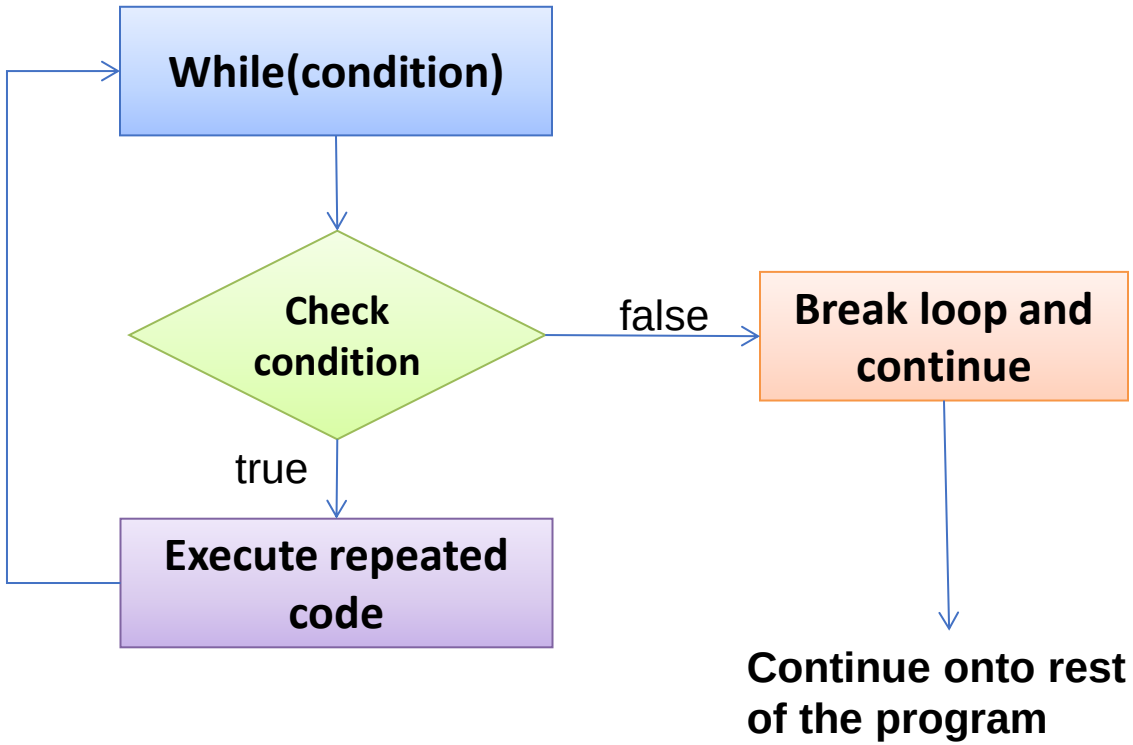
The while loop is a **statement** or **block** of statements that is **repeated** as long as some **condition** is **satisfied**

**Syntax**

```
while (boolean_expression) {
statement1;
    statement 2;
    …..
}
```

The statements in *while* loop are executed as long as the *boolean expression* is true

# Illustration of a while statement



# Lend a Hand – while

1. Create a java class "WelcomeMessage" and add a method named **printMessage** which would display "Welcome All".
2. Create a java class "TestProgram" add a main method which will
   - Create an instance of the **WelcomeMessage** and trigger the method **printMessage** five times.
   - The message "Welcome All" should be displayed 5 times.
3. The message needs to be displayed in the console.

**Use while Statement**

# while Statement Example

**Develop the code as illustrated below.**

```
class WelcomeMessage{
    void printMessage(){
        System.out.println("Welcome All");
    }
}
class TestProgram {

   public static void main(String[] args) {
        int count =5;
        WelcomeMessage message=new WelcomeMessage();
        while(count>0){
            message.printMessage();
            count--; }
    }
}
```

# do-while Statement

**What is a Do while Loop:**

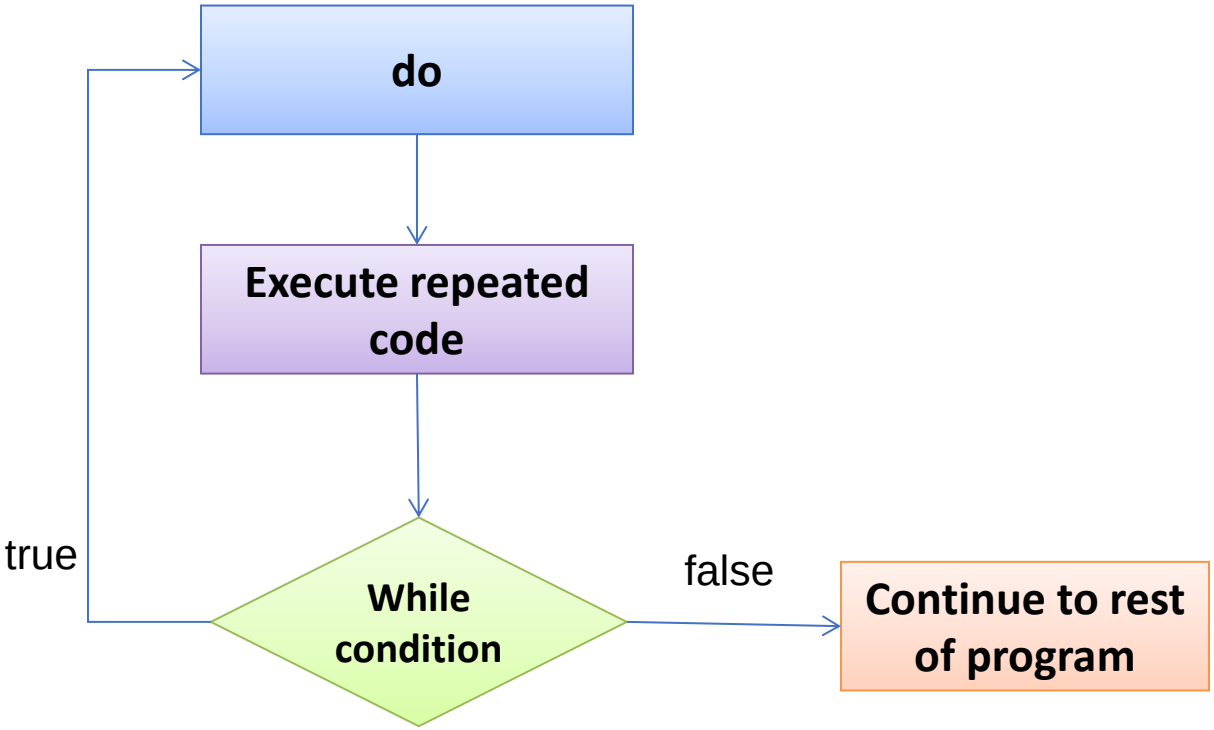
It is **similar** to **while** loop except that the do-while **execute** the block **once**, and then **checks** the **while** condition.

The do-while loop always executes its body at least once, because its conditional expression is at the end of the loop.

**Syntax**

```
do {
  statement1;
  statement2;
} while(boolean_expression);
```

# Illustration of a do while statement



# Example of do-while Statement

**Example of a do while statement:**

```
public class DoWhileExample {

    public static void main(String[] args) {
        int i = 6;
        do {
            System.out.println("i is : " + i);
            i++;
        } while (i < 5);
    }
}
```

The value of **i** is printed for the first time, even though it does not match the condition **i < 5**

**Output:**

```
i is : 6
```

# Lend a Hand – do while

1. Create a java class "WelcomeMessage" and add a method named **printMessage** which would display "Welcome All".
2. Create a java class "TestProgram" add a main method which will
   - Create an instance of the **WelcomeMessage** and trigger the method **printMessage** five times.
   - The message "Welcome All" should be displayed 5 times.
3. The message needs to be displayed in the console.

**Use do while Statement**

# do-while Statement Example

**Develop the code as illustrated below.**

```
class WelcomeMessage{
    void printMessage(){
        System.out.println("Welcome All");
    }
 }
class TestProgram {

   public static void main(String[] args) {
        int count =5;
        WelcomeMessage message=new WelcomeMessage();
        do{
            message.printMessage();
            count--; }
        while(count>0);
    }
  }
```

# for Statement

**What is a for loop?**

***For statement*** is similar to while loop is used to repeat the execution of the code till a condition is met.

**Syntax:**

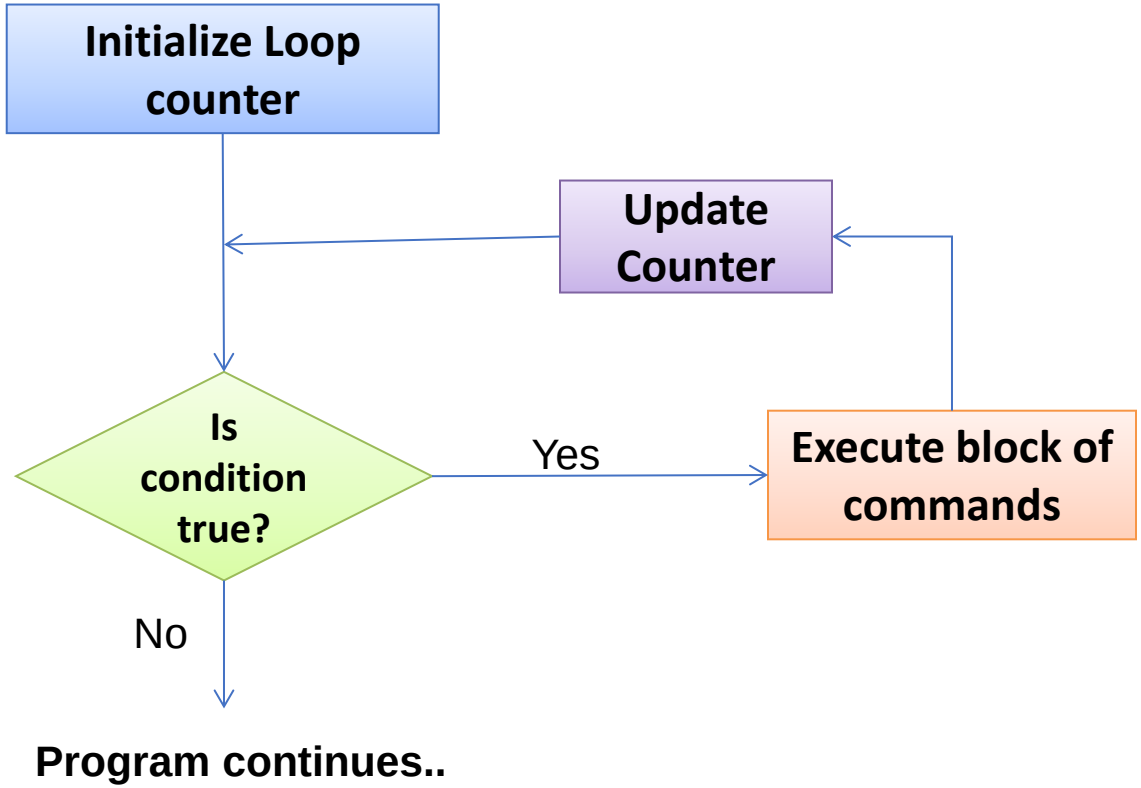
```
for(initialization; loopCondition; iteration) {
    statements;
}
```

The initialization allows to declare and/or initialize loop variables, and is executed only once.

The loopCondition compares the loop variable to some limit value. If the loop condition is not met it is broken.

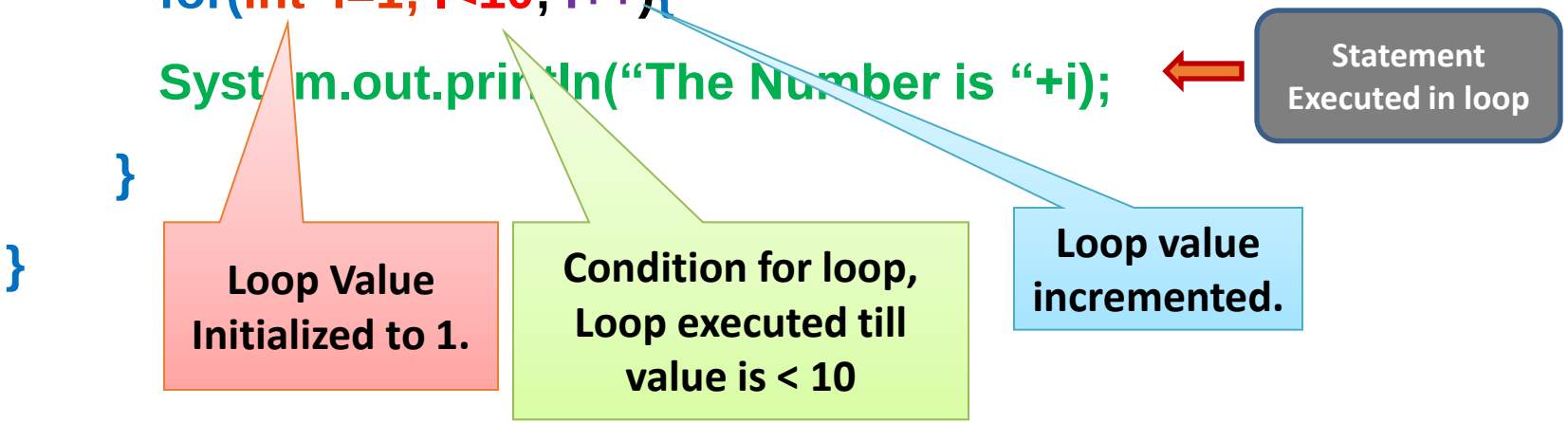
The iteration usually increments or decrements the values of the loop variables

# Illustration of a for statement



# for Statement Example

```
class Example{
    public static void main(String []args){
      for(int  i=1; I<10; i++){
      Syst  m.out.pri  ln("The Nu.mber is "+i);
    }
}
```



# Lend a Hand – for Loop statement

1. Create a java class "WelcomeMessage" and add a method named **printMessage** which would display "Welcome All".
2. Create a java class "TestProgram" add a main method which will
   - Create an instance of the **WelcomeMessage**. and trigger the method **printMessage** five times.
   - The message "Welcome All" should be displayed 5 times in the console.
3. The message needs to be displayed in the console.

**Use For Loop Statement**

# for Statement Example

**Develop the code as illustrated below.**

```
package com.statements.demo;

class DisplayMessage{
    void printMessage(){
         System.out.println("Welcome All");
         }
    }
 class TestProgram {
    public static void main(String[] args) {
    DisplayMessage message=new DisplayMessage();
    int num=5;
    for(int i=1;i<=num;i++){
         message.printMessage();}

       }

 }
```

# Transfer Statement

**What are transfer statements?**
The **transfer** Statements in Java alter the normal control flow of the statements. They allow you to redirect the flow of program execution.

Transfer Statements are used to quit either the current iteration of a loop or the entire loop.

Java offers two transfer statements

1. break.
2. continue.
3. return.

# break Statement

***Break*** statement used for,

1. Used to terminates a statement sequence in a switch statement.
2. Used to exit loops in  Iteration Statement.

**Problem statement:**
This program iterates through the 100 employees and calculate salary . If one employee is minor age, i.e. age  < 18 it should break the loop and stop the execution.

```
while (employeecount<=100) {
    if(employeeAge <18)
    {
      break;
    }
    calculateSalary();
}
```

# continue Statement

***Continue*** Statements stops the processing the remaining code in the body of the particular iteration Statement and continue with the next loop.

**Problem statement:**
This program iterates through the 100 employees and calculate salary . If one employee is minor age, i.e. age < 18 it should SKIP the salary calculation logic for the employee and proceed with other employees.

```
while (employeecount<100) {
    if(employeeAge <18)
    {
       continue;
    }
    calculateSalary();
}
```

# continue Statement

- Continue Statement can be used within selection statements which are inside iteration statements .

- In while and do-while statements, a continue statement causes control to be transferred directly to the conditional expression that controls the loop.

- In a for statement, a continue statement skips the current iteration and causes the control to got to the first statement of the loop to proceed with the next iteration.

# return Statement

**Return statement:**

The return statement **exits** from the **current method**, and the control flow returns to where the method was invoked.

The return statement has two forms:

One that returns a value

One that doesn't.

**Option 1:** To return a value, simply put the value or expression that needs to be returned after the return keyword.

**return <value/expression>;**

**Option 2:** When the return type of method is void, use the form of return that does not return a value

**return;**

In this case, the execution of the method is stopped.

# Lend a Hand - return statement

```
public class TestProgram {

    public static void main(String[] args) {
        int count = 5;
        int i;
        WelcomeMessage welcome = new WelcomeMessage();
        for (i=1;i<=count;i++){
            welcome.printMessage();
            if(i==3){
                return;
            }
            System.out.println("After if loop "+i);
        }
        System.out.println("Final returned value of i is "+i);
    }
}
```

Let us use the same WelcomeMessage class that we developed for the previous example

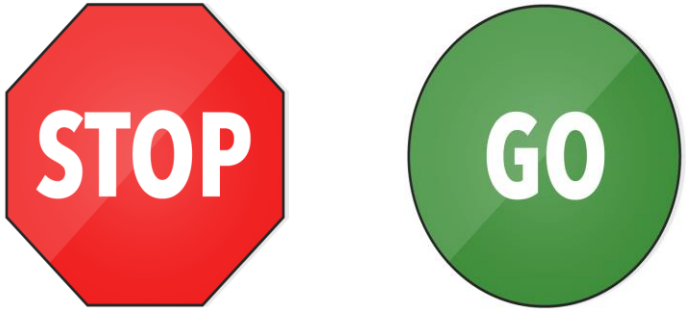
**When i is equal to 3, the return statement is executed and the execution of the method is stopped**

**Output**

```
Welcome all
After if loop 1
Welcome all
After if loop 2
Welcome all
```

**Try the same example with break and continue statement and see how the program behaves.**

# Time To Reflect




Trainees to reflect the following topics before proceeding.

1. What statements will you use to execute a block of code repetitively.
2. How to stop a execution of a loop?
3. Difference between do while and while statement?
4. What is the difference between switch and If statements?

Thank you

*You have successfully completed*

# Statements