Anu Arora, Assistant Professor, GNA University

STATEMENTS IN JAVA

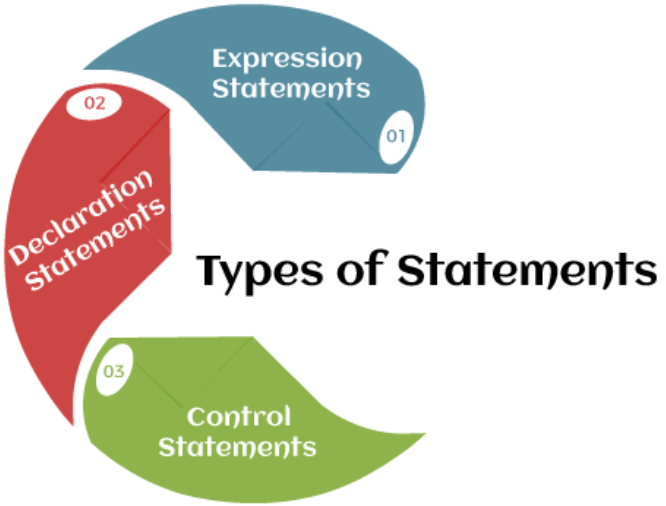
Statements in Java

- In Java, a **Statement** is an executable instruction that tells the compiler what to perform.
- It forms a complete command to be executed and can include one or more expressions.
- A sentence forms a complete idea that can include one or more clauses.
- All Java statements must end in a semicolon (;). This tells Java that it should attempt to process all information up to that semicolon

Types of Statements

Java statements can be broadly classified into the following categories:

- Expression Statements
- Declaration Statements
- Control Statements



Expression Statements

- Expression is an essential building block of any Java program
- Generally, it is used to generate a new value. Sometimes, we can also assign a value to a variable
- In Java, expression is the combination of values, variables, operators, and method calls.

Types of Expressions in Java

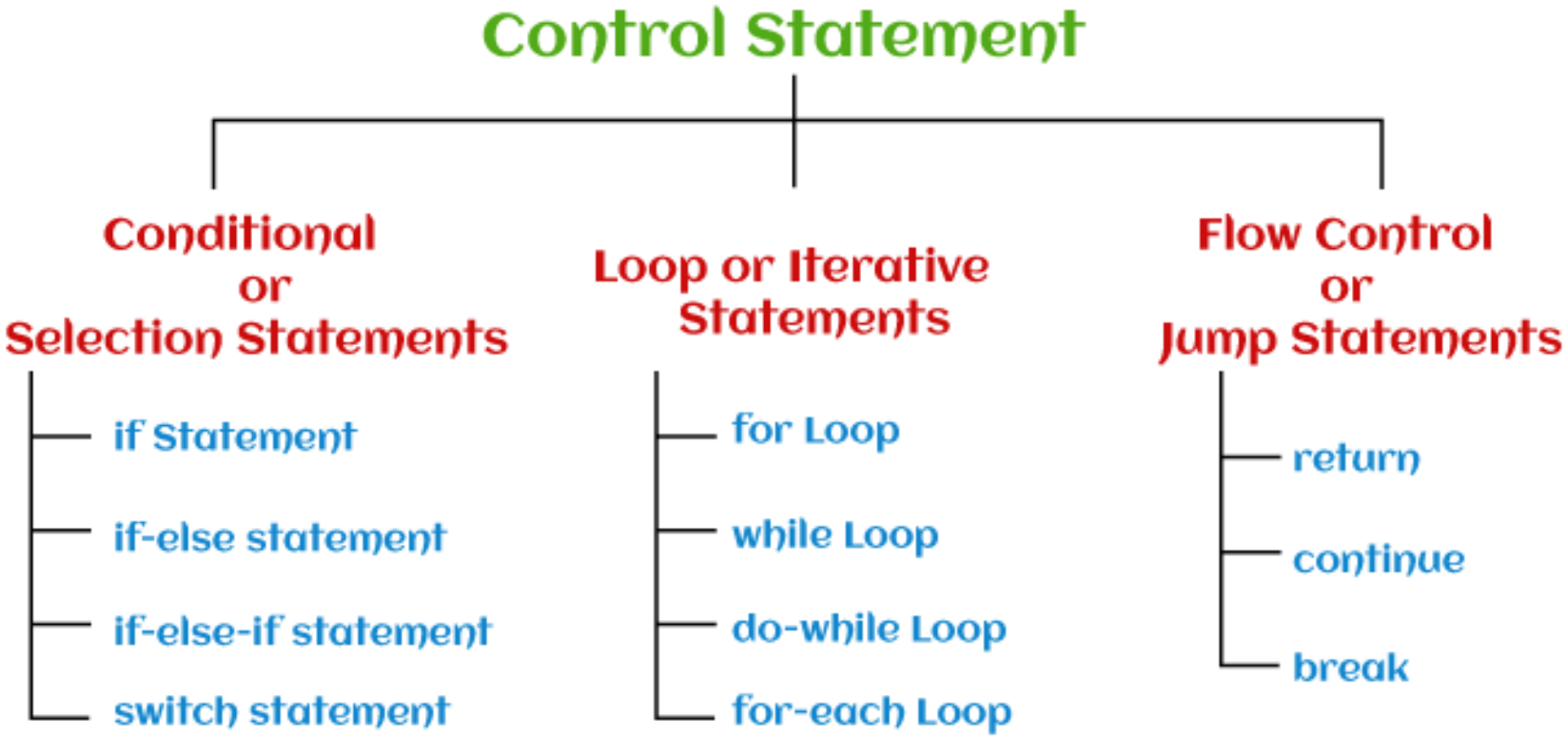
- Expressions that **produce** a value.
- Expressions that **assign** a value.
- Expression that **neither produces any result nor assigns a value.**

Declaration Statements

- In declaration statements, we declare variables and constants by specifying their data type and name.
- A variable holds a value that is going to use in the Java program.
- Example: int quantity;

Control Statement

- Control statements decide the flow (order or sequence of execution of statements) of a Java program.
- In Java, statements are parsed from top to bottom. Therefore, using the control flow statements can interrupt a particular section of a program based on a certain condition.


Control Statement
Conditional or Selection Statements
if Statement
if-else statement
if-else-if statement
switch statement
Loop or Iterative Statements
for Loop
while Loop
do-while Loop
for-each Loop
Flow Control or Jump Statements
return
continue
break

Example

```
//declaration statement
int number;
//expression statement
number = 4;
//control flow statement
if (number < 10 ) {
  //expression statement
 System.out.println(number + " is less than ten");
 }
```

Assignment

- Write a Java Program which enters a value in CMs and converts that to Millimeters.
- Write a Java Program which enters a temperature in Celsius and converts that to Fahrenheit.
- Write a Java Program which enters a value in Rs and converts that to USD Dollars and AUS dollar.

Decision-Making statements:

- As the name suggests, decision-making statements decide which statement to execute and when.
- Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided.
- There are two types of decision-making statements in Java, i.e., If statement and switch statement.

if statement

- In Java, the "if" statement is used to evaluate a condition.
- The control of the program is diverted depending upon the specific condition.
- The condition of the If statement gives a Boolean value, either true or false.

Types of if-statements

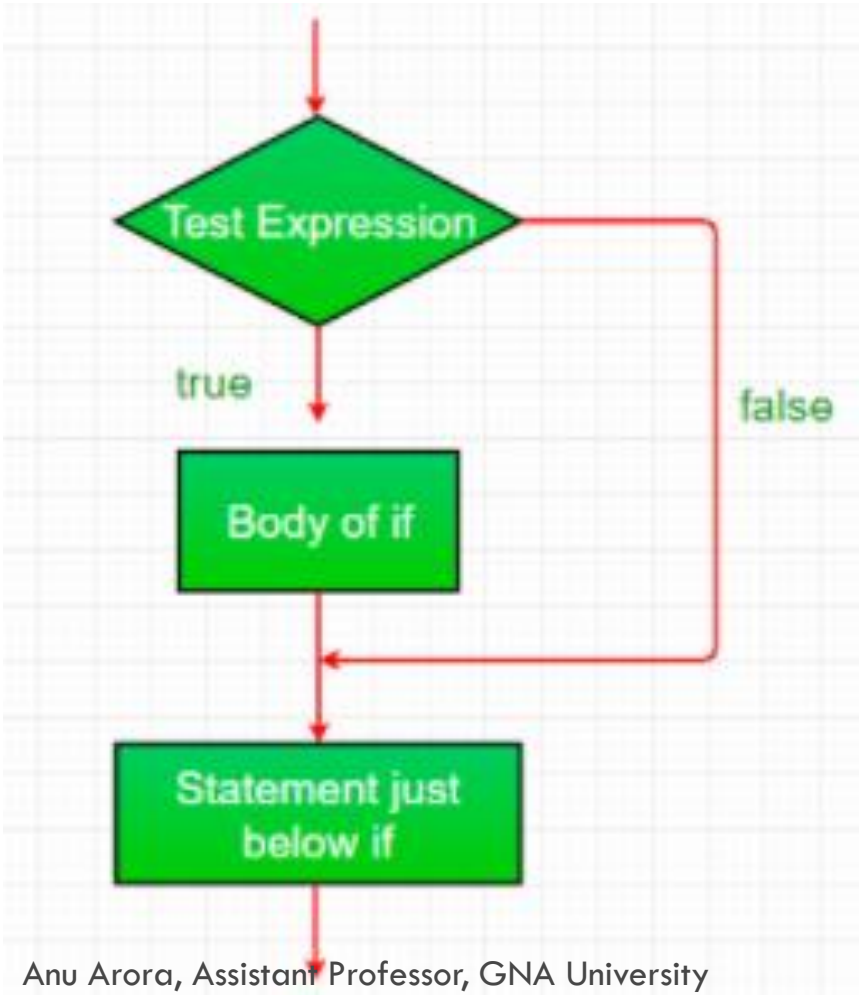
In Java, there are four types of if-statements given below:

- Simple if statement
- if-else statement
- if-else-if ladder
- Nested if-statement

Simple if statement:

- It is the most basic statement among all control flow statements in Java.
- It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.
- Syntax of if statement is given below:

```
if(condition) {
statement 1; //executes when condition is true
}
```

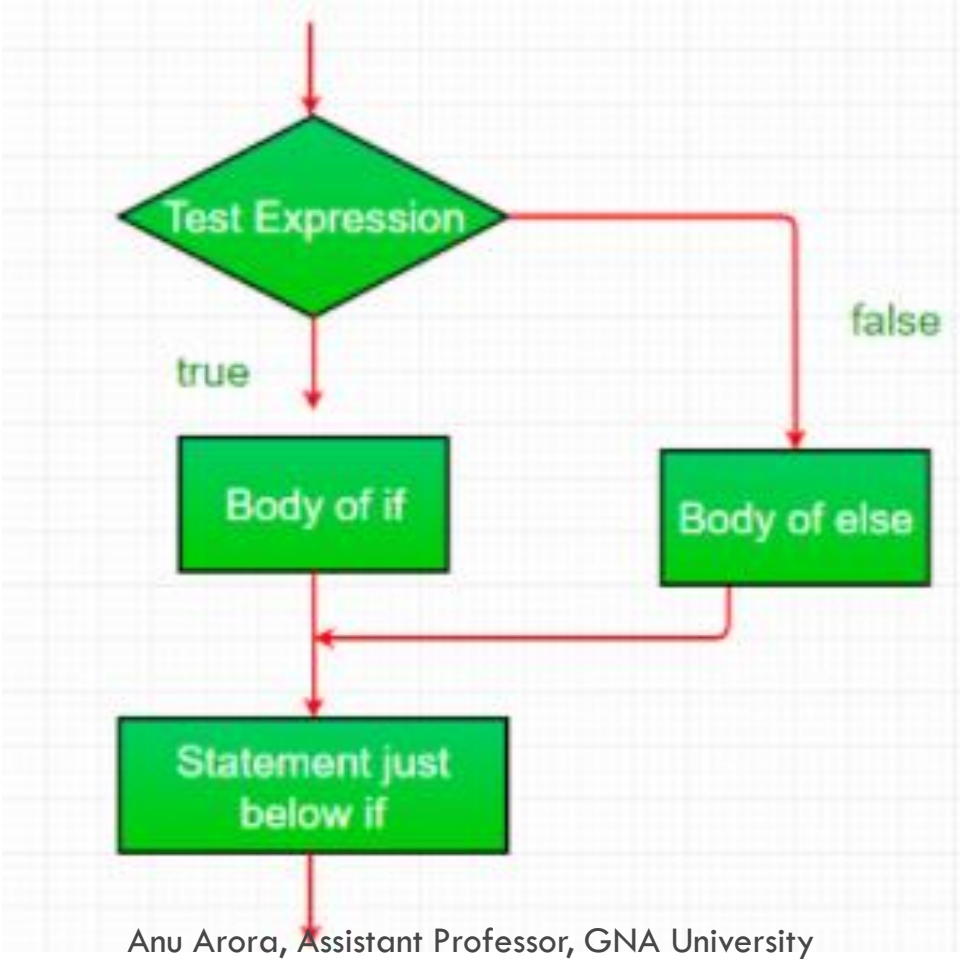

Test Expression
true
false
Body of if
Statement just
below if

if-else statement

- The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block.
- The else block is executed if the condition of the if-block is evaluated as false.

- Syntax:

```
if(condition) {
statement 1; //executes when condit
    //ion is true
}
else{
statement 2; //executes when condit
    //ion is false
}
```

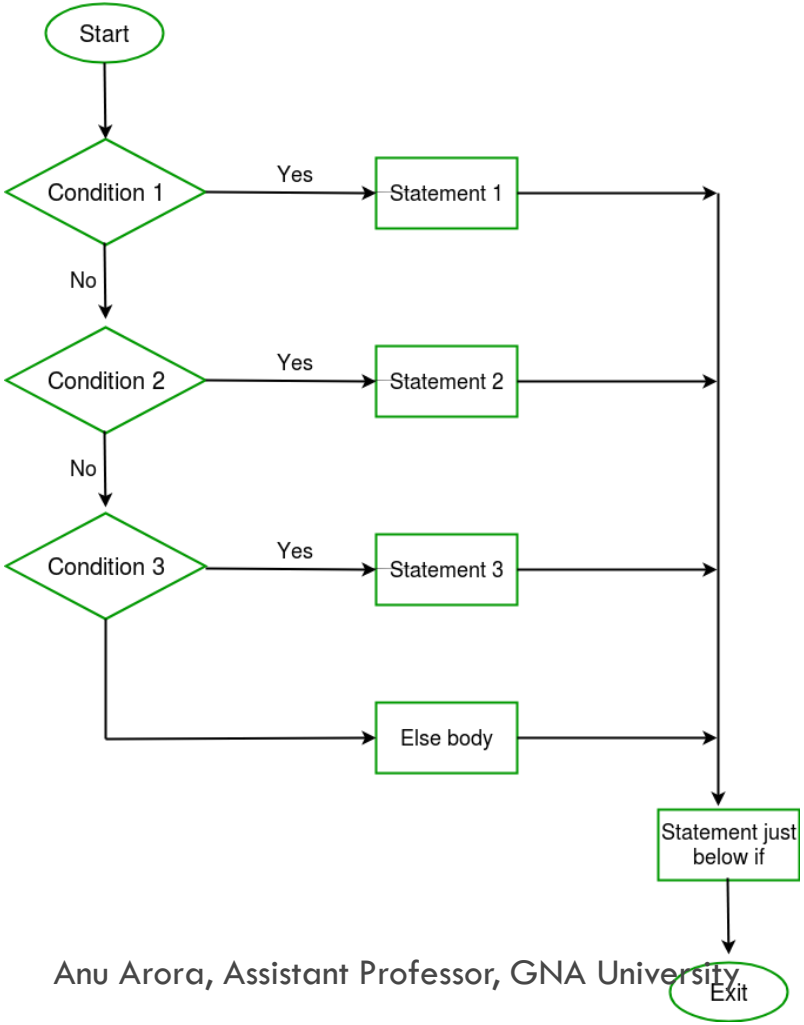

Test Expression
true
false
Body of if
Body of else
Statement just below if

if-else-if ladder:

- The if-else-if statement contains the if-statement followed by multiple else-if statements.
- In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true.
- We can also define an else statement at the end of the chain.

Syntax:

```
if(condition 1) {
statement 1; //executes when condition
      //1 is true  }
else if(condition 2) {
statement 2; //executes when condition
      //2 is true }
else {
statement 3; //executes when all the
//conditions are false }
```

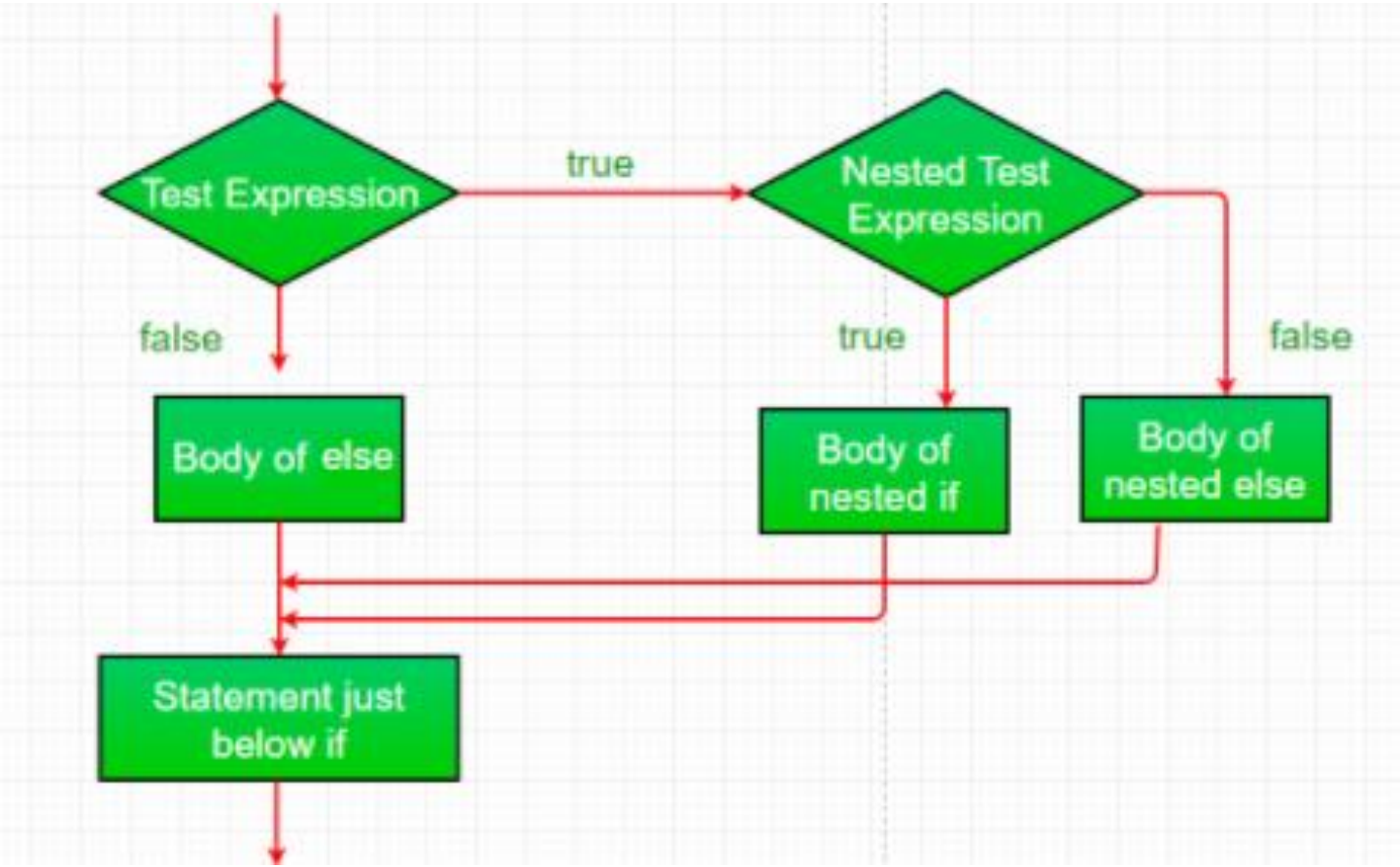

Start
Condition 1
Yes
Statement 1
No
Condition 2
Yes
Statement 2
No
Condition 3
Yes
Statement 3
Else body
Statement just below if
Exit

Nested if-statement

- In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.

Syntax:

```
if(condition 1) {
statement 1; //executes when condit
    ion 1 is true
if(condition 2) {
statement 2; //executes when condit
    ion 2 is true  }
else{
statement 2; //executes when condit
    ion 2 is false  }  }
```


Test Expression
true
Nested Test Expression
false
true
false
Body of else
Body of nested if
Body of nested else
Statement just below if

Switch Statement:

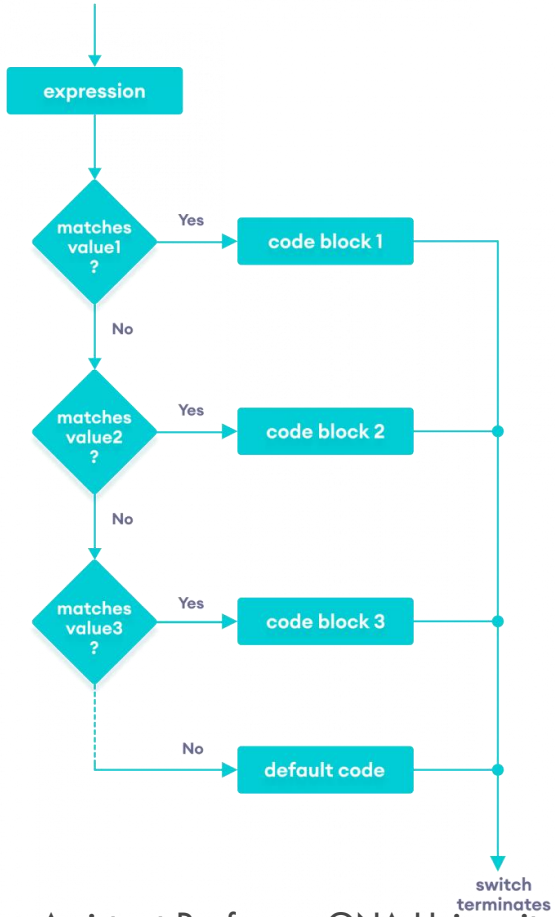
- In Java, Switch statements are similar to if-else-if statements.
- The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched.
- The switch statement is easier to use instead of if-else-if statements.
- It also enhances the readability of the program.

Points to be noted about switch statement:

- The case variables can be int, short, byte, char, or enumeration. String type is also supported since version 7 of Java
- Cases cannot be duplicate
- Default statement is executed when any of the case doesn't match the value of expression. It is optional.
- Break statement terminates the switch block when the condition is satisfied. It is optional, if not used, next case is executed.
- While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value.

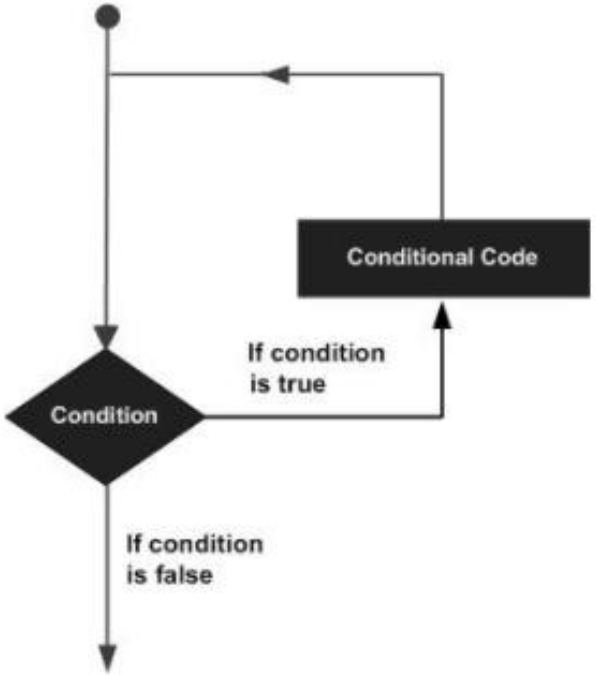
Syntax

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


expression
matches value1 ?
Yes
code block 1
No
matches value2 ?
Yes
code block 2
No
matches value3 ?
Yes
code block 3
No
default code
switch terminates

Loops/Iterative Statements

- A **loop** statement allows us to execute a statement or group of statements multiple times



Types of loops

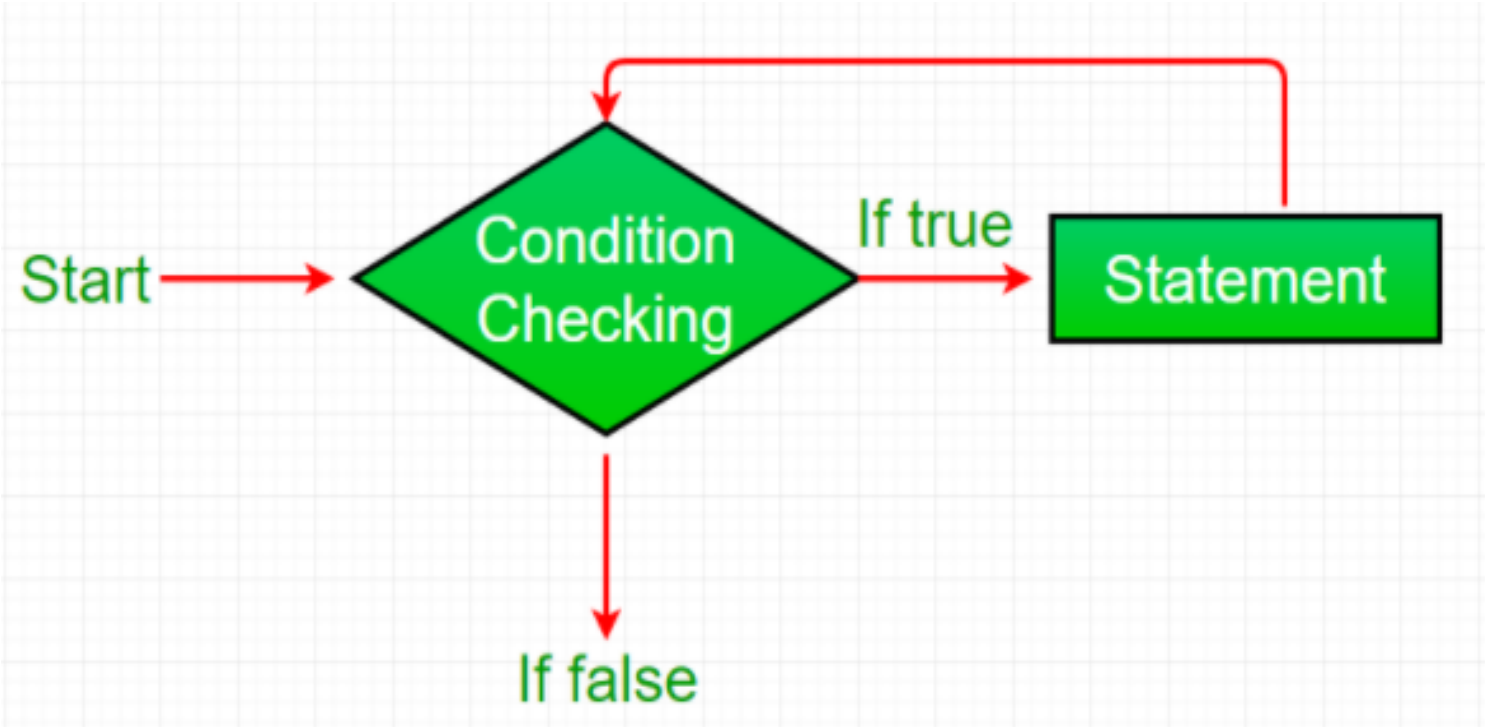
In Java, there are three types of loops.

- while loop
- do...while loop
- for loop

while loop

- A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition.
- The while loop can be thought of as a repeating if statement.

Syntax :while (boolean condition) { loop statements... }


Start
Condition Checking
If true
Statement
If false

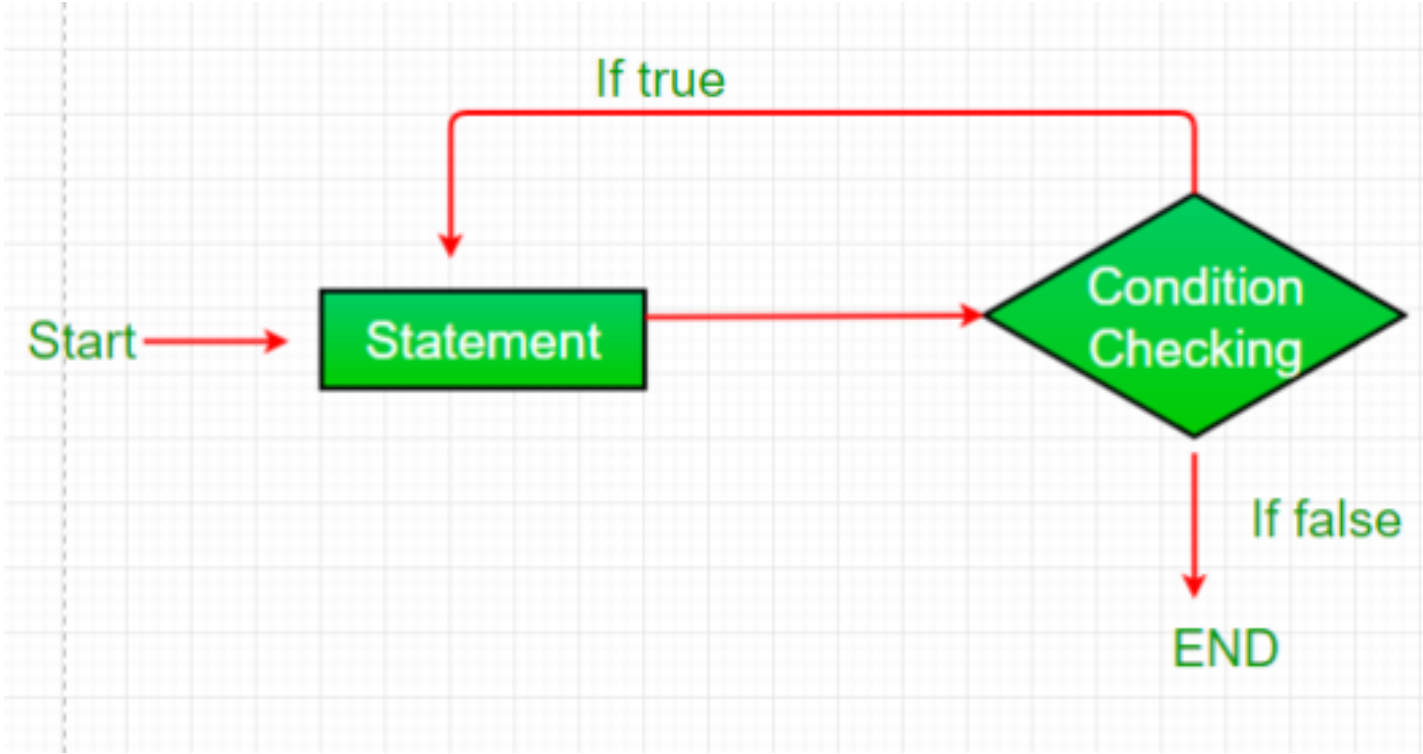
while loop

- While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called **Entry control loop**
- Once the condition is evaluated to true, the statements in the loop body are executed. Normally the statements contain an update value for the variable being processed for the next iteration.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.

do while loop

- do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of **Exit Control Loop.**

Syntax: do { statements.. } while (condition);


If true
Start
Statement
Condition
Checking
If false
END

do while loop

- do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
- After the execution of the statements, and update of the variable value, the condition is checked for true or false value. If it is evaluated to true, next iteration of loop starts.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.
- It is important to note that the do-while loop will execute its statements atleast once before any condition is checked, and therefore is an example of exit control loop.

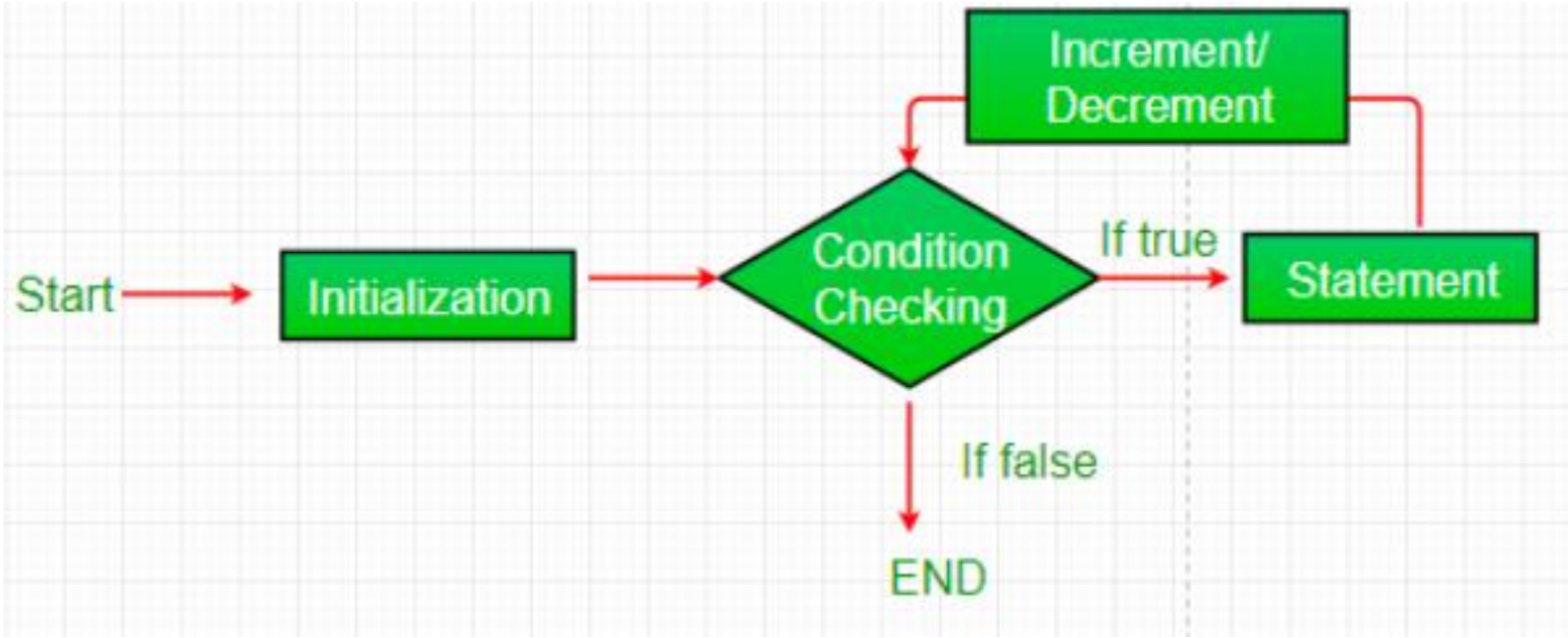
for loop

- for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

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

for loop

- **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.
- **Testing Condition:** It is used for testing the exit condition for a loop. It must return a boolean value. It is also an **Entry Control Loop** as the condition is checked prior to the execution of the loop statements.
- **Statement execution:** Once the condition is evaluated to true, the statements in the loop body are executed.
- **Increment/ Decrement:** It is used for updating the variable for next iteration.
- **Loop termination:**When the condition becomes false, the loop terminates marking the end of its life cycle.


Increment/
Decrement
Start
Initialization
Condition
Checking
If true
Statement
If false
END

Jump Statements

Java supports three jump statements:

- **Break**
- **continue**
- **Return**

These three statements transfer control to another part of the program.

break statement

- When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- The Java *break* statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.
- We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

continue statement

- The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop.
- The Java *continue statement* is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.
- We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.

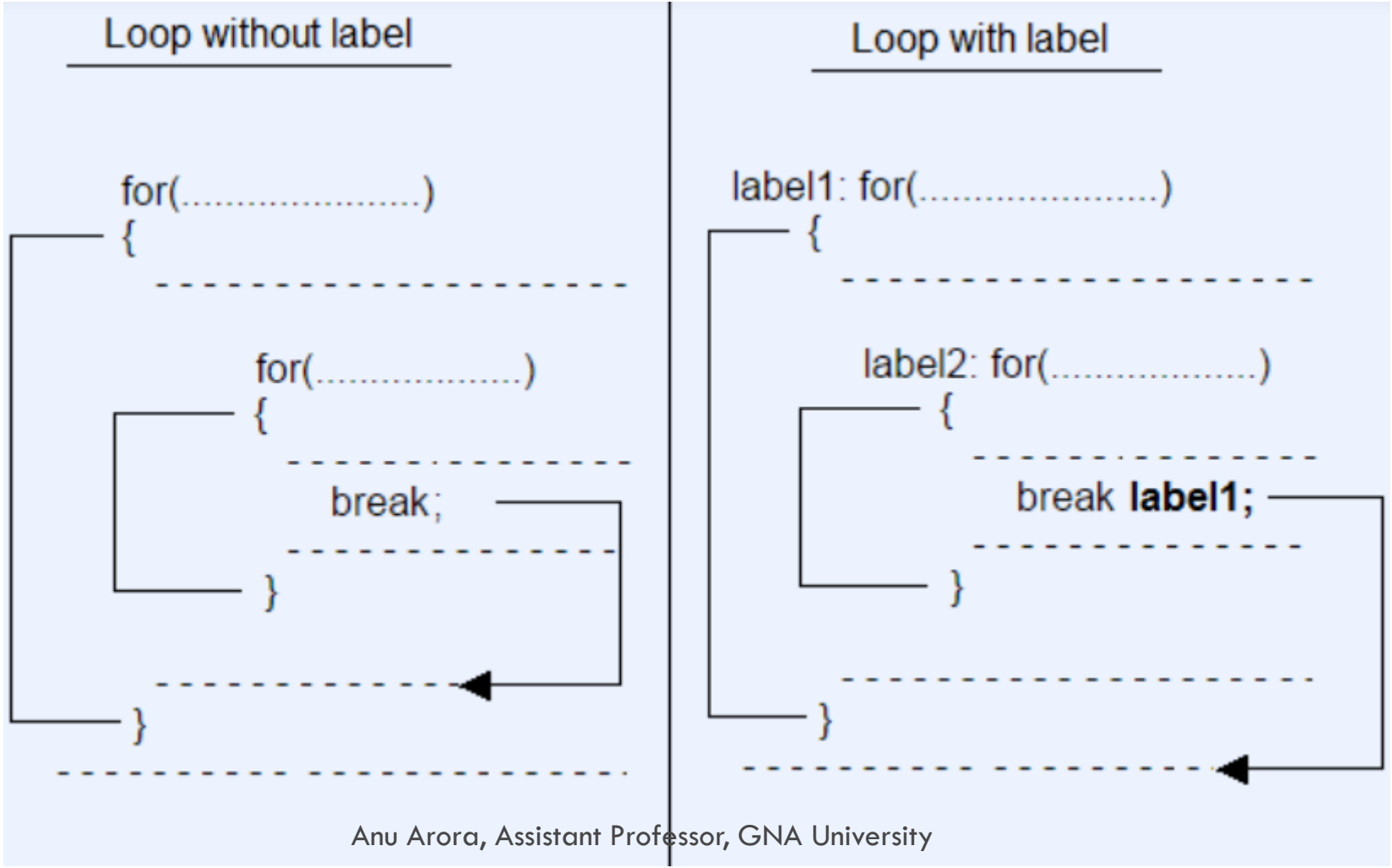
return statement

- The return statement is used to explicitly return from a method. That is, it causes program control to transfer back to the caller of the method.

Key	Break	Continue
Functionality	Break statement mainly used to terminate the enclosing loop such as while, do-while, for or switch statement wherever break is declared.	Continue statement mainly skip the rest of loop wherever continue is declared and execute the next iteration.
Executional flow	Break statement resumes the control of the program to the end of loop and made executional flow outside that loop.	Continue statement resumes the control of the program to the next iteration of that loop enclosing 'continue' and made executional flow inside the loop again.
Usage	As mentioned break is used for the termination of enclosing loop.	On other hand continue causes early execution of the next iteration of the enclosing loop.
Compatibility	Break statement can be used and compatible with 'switch', 'label'.	We can't use continue statement with 'switch', 'lablel' as it is not compatible with them.

Labeled Loop in Java

- A **label** is a valid variable name that denotes the name of the loop to where the control of execution should jump.
- To label a loop, place the label before the loop with a colon at the end.
- Therefore, a loop with the label is called a **labeled loop**.


Loop without label
for(.....)
{
for(.....)
{
break;
}
}
Loop with label
label1: for(.....)
{
label2: for(.....)
{
break label1;
}
}

Java Labeled for Loop

- Labeling a for loop is useful when we want to break or continue a specific for loop according to requirement.
- If we put a break statement inside an inner for loop, the compiler will jump out from the inner loop and continue with the outer loop again.

Syntax:

```
labelname:
for(initialization; condition; incr/decr)
{
//functionality of the loop
}
```

Java Labeled while Loop: Syntax

```
labelName:
while ( ... )
{
//statements to execute
}
```

References

- E.Balaguruswamy, Programming with JAVA, A primer, 3e, TATA McGraw-Hill Company.
- https://www.javatpoint.com/operators-in-java
- https://www.geeksforgeeks.org/operators-in-java/
- https://www.tutorialspoint.com/