Build COMPETENCY
across your TEAM

SYNERGETICS GET IT RIGHT

Microsoft Partner
Gold Cloud Platform
Silver Learning

Language Fundamentals

Google Cloud Platform

Windows Azure

amazon web services   APACHE kafka A distributed streaming platform   Apache SOFTWARE FOUNDATION   APACHE Spark   Java   hadoop

Language Fundamentals

Keywords

Primitive Data Types

Operators and Assignments

Variables and Literals

Flow Control: Java’s Control Statements

Java Keywords

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

*	not used
**	added in 1.2
***	added in 1.4
****	added in 5.0

Primitive data types

The eight primitive data types in Java are:

- boolean, the type whose values are either true or false
- char, the character type whose values are 16-bit Unicode characters

The arithmetic types:

the integral types:

- byte
- short
- int
- long

The floating-point types:

- float
- double

Primitive data types in Java

Type	Description	Default	Size	Example Literals
boolean	true or false	false	1 bit	true, false
byte	twos complement integer	0	8 bits	(none)
char	Unicode character	\u0000	16 bits	'a', '\u0041', '\101', '\\', '\'', '\n', 'ß'
short	twos complement integer	0	16 bits	(none)
int	twos complement integer	0	32 bits	-2, -1, 0, 1, 2
long	twos complement integer	0	64 bits	-2L, -1L, 0L, 1L, 2L
float	IEEE 754 floating point	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d

The Type Promotion Rules

Widening conversions do not lose information about the magnitude of a value.

For example, an int value is assigned to a double variable.

This conversion is legal because doubles are wider than ints.

Java's widening conversions are

- From a byte to a short, an int, a long, a float, or a double
- From a short to an int, a long, a float, or a double
- From a char to an int, a long, a float, or a double
- From an int to a long, a float, or a double
- From a long to a float or a double
- From a float to a double

The Type Promotion Rules

- Widening conversions:
- char->int
- byte->short->int->long->float->double

Here are the Type Promotion Rules:

All byte and short values are promoted to int.
If one operand is a long, the whole expression is promoted to long.
If one operand is a float, the entire expression is promoted to float.
If any of the operands is double, the result is double.

The Type Promotion Rules

- In the following code, f * b, b is promoted to a float and the result of the subexpression is float.

```
public class Main {

  public static void main(String args[]) {
    byte b = 4;
    float f = 5.5f;
    float result = (f * b);
    System.out.println("f * b = " + result);
  }
}
```

```
Output: f * b = 22.0
```

Type Demotion in Java

- Demotion is not implicitly possible in java.
- Demotion may lead to loss of precision or undesired output.
- Demotion can be done using something called as Data Type Casting.
- Demotion is to be done explicitly

```
float f = (float)22.4;            *decimal by default are considered double in java


int radius = 22;
int area = (int)(Math.PI * Math.pow(radius, 2));   *Math.PI is a double constant value
```

Java - Basic Operators

Java provides a rich set of operators to manipulate variables.

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

The Arithmetic Operators

- Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra.

The following table lists the arithmetic operators

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

The Relational Operators

- There are following relational operators supported by Java language.

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

The Bitwise Operators

- Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

- Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60 and b = 13; now in binary format they will be as follows −

```
a = 0011 1100
b = 0000 1101
-----------------
a&b = 0000 1100
a|b = 0011 1101
a^b = 0011 0001
~a  = 1100 0011
```

The Bitwise Operators

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

The Logical Operators

- The following table lists the logical operators –

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

The Assignment Operators

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Miscellaneous Operators

- There are few other operators supported by Java Language

- Conditional Operator (? :)
- Conditional operator is also known as the **ternary operator**. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide, which value should be assigned to the variable.

```
variable x = (expression) ? value if true : value if false
```

- instanceof Operator

```
Vehicle a = new Car();
boolean result =  a instanceof Car;
System.out.println( result );
```

Precedence of Java Operators

- Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated.
- Certain operators have higher precedence than others
- Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom.

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left toright
Unary	++ - - ! ~	Right to left
Multiplicative	* /	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	>> >= < <=	Left to right
Equality	>== !=	Left to right
Bitwise AND	>&	Left to right
Bitwise XOR	>^	Left to right
Bitwise OR	>|	Left to right
Logical AND	>&&	Left to right
Logical OR	>||	Left to right
Conditional	?:	Right to left
Assignment	>= += -= *= /= %= >>= <<= &= ^= |=	Right to left

Variables and Literals

- A variable provides us with named storage that our programs can manipulate.
- Each variable in Java has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.
- You must declare all variables before they can be used.

data type variable [= value][, variable [= value] ...] ;

Here *data type* is one of Java's datatypes and *variable* is the name of the variable. To declare more than one variable of the specified type, you can use a comma-separated list.

- Following are valid examples of variable declaration and initialization

```
int a, b, c;          // Declares three ints, a, b, and c.
int a = 10, b = 10;  // Example of initialization
byte B = 22;          // initializes a byte type variable B.
double pi = 3.14159; // declares and assigns a value of PI.
char a = 'a';         // the char variable a iis initialized with value 'a'
```

Types of Variables In Java

There are three kinds of variables in Java

- Local variables
- Instance variables
- Class/Static variables

Local Variables

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor, or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

```
public class Test {
  public void pupAge() {
    int age = 0;
    age = age + 7;
    System.out.println("Puppy age is : " + age);
  }

  public static void main(String args[]) {
    Test test = new Test();
    test.pupAge();
  }
}
```

Instance Variables

- Instance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. *ObjectReference.VariableName*.

Instance Variables

```
import java.io.*;
public class Employee {

   // this instance variable is visible for any child class.
   public String name;

   // salary  variable is visible in Employee class only.
   private double salary;

   // The name variable is assigned in the constructor.
   public Employee (String empName) {
      name = empName;
   }

   // The salary variable is assigned a value.
   public void setSalary(double empSal) {
      salary = empSal;
   }

   // This method prints the employee details.
   public void printEmp() {
      System.out.println("name  : " + name );
      System.out.println("salary :" + salary);
   }

   public static void main(String args[]) {
      Employee empOne = new Employee("Ransika");
      empOne.setSalary(1000);
      empOne.printEmp();
   }
}
```

Class/Static Variables

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.
- Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks.
- Static variables can be accessed by calling with the class name *ClassName.VariableName*.
- When declaring class variables as public static final, then variable names (constants) are all in upper case. If the static variables are not public and final, the naming syntax is the same as instance and local variables.

Java Control Flow Statements

- The control flow statements in Java allow you to run or skip blocks of code when special conditions are met.

The “if” Statement

- The “if” Statement

The “if” statement in Java works exactly like in most programming languages. With the help of “if” you can choose to execute a specific block of code when a predefined condition is met. The structure of the “if” statement in Java looks like this:

```
if (condition) {
        // execute this code
}
```

```
public class FlowControlExample {
        public static void main(String[] args) {
                int age = 2;
                System.out.println("Peter is " + age + " years old");
                if (age < 4) {
                        System.out.println("Peter is a baby");
                }
        }
}
```

The “if else” Statement

- Whit this statement you can control what to do if the condition is met and what to do otherwise.

```
public class FlowControlExample {
	public static void main(String[] args) {
		int age = 10;
		System.out.println("Peter is " + age + " years old");
		if (age < 4) {
			System.out.println("Peter is a baby");
		} else {
			System.out.println("Peter is not a baby anymore");
		}
	}
}
```

The switch Statement

```
public class SwitchExample {
        public static void main(String[] args) {
                int numOfAngles = 3;

                switch (numOfAngles) {
                case 3:
                        System.out.println("triangle");
                        break;
                case 4:
                        System.out.println("rectangle");
                        break;
                case 5:
                        System.out.println("pentagon");
                        break;
                default:
                        System.out.println("Unknown shape");
                }
        }
}
```

In some cases you can avoid using multiple if-s in your code and make your code look better.

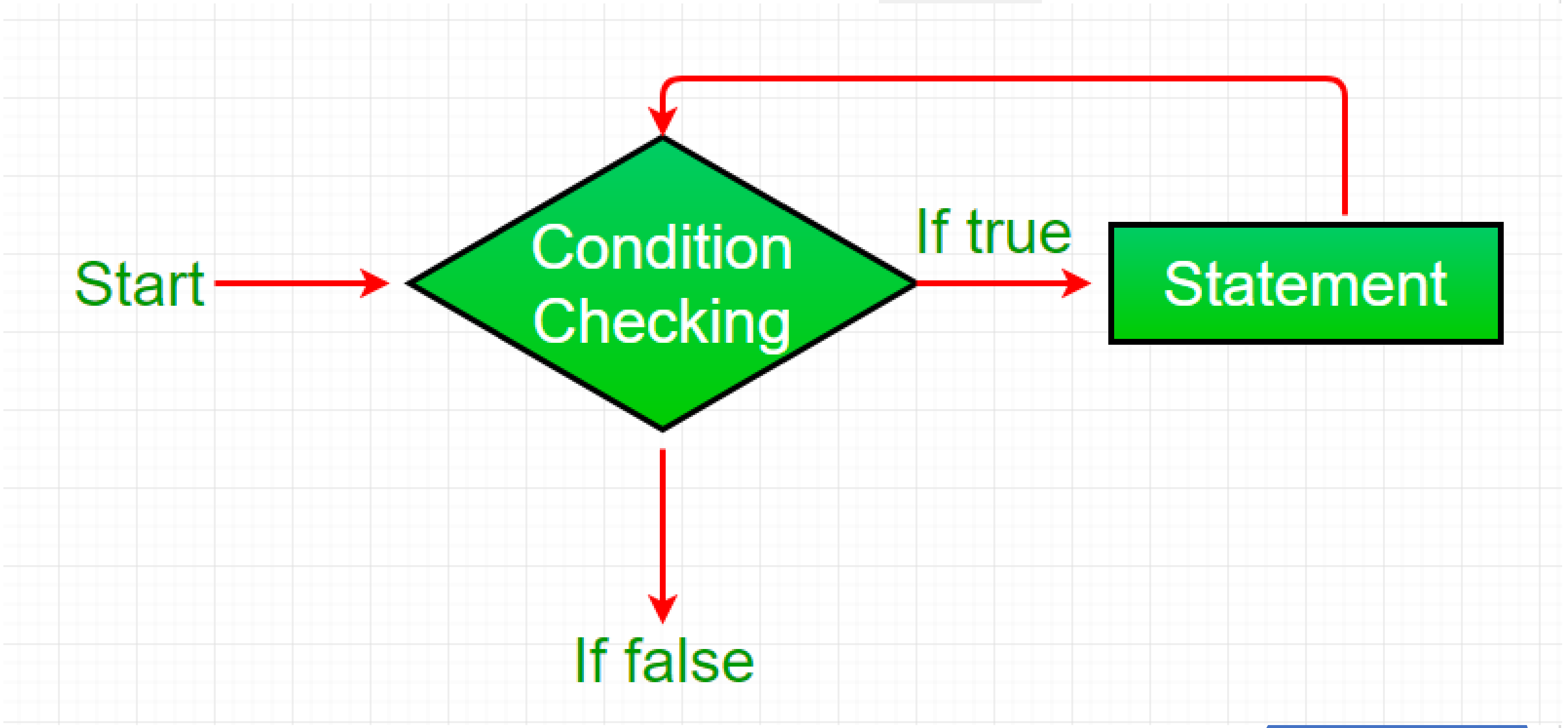
Loops in Java

- Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.
- Java provides three ways for executing the loops.
- While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

The While 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. while (boolean condition)

```
{
        loop statements...
}
```



The 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.

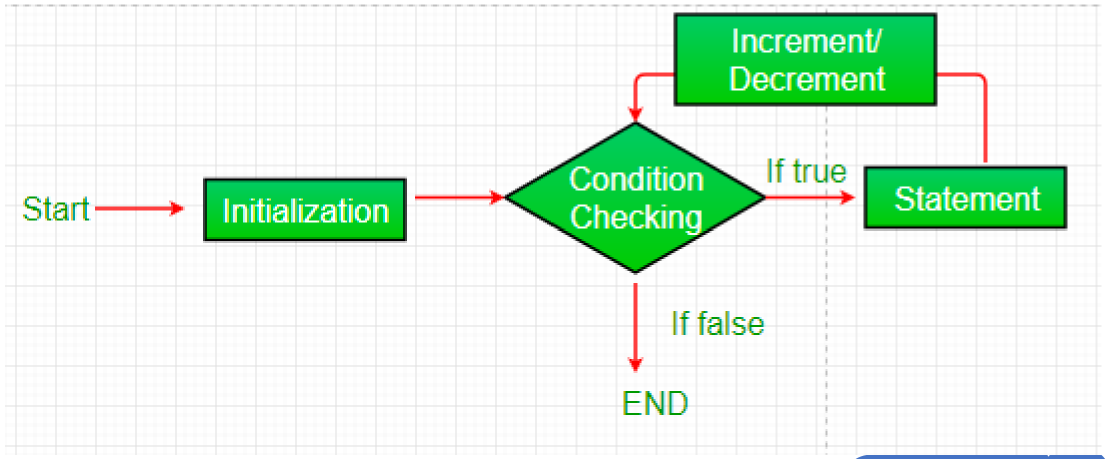
The While Loop

```
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++;
    }
```

The 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.

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



The 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.

The For Loop

```
// 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);
```

The Enhanced For loop

- Java also includes another version of for loop introduced in Java 5.
- Enhanced for loop provides a simpler way to iterate through the elements of a collection or array.
- It is inflexible and should be used only when there is a need to iterate through the elements in sequential manner without knowing the index of currently processed element.

```
for (T element:Collection obj/array)
{
   statement(s)
}
```

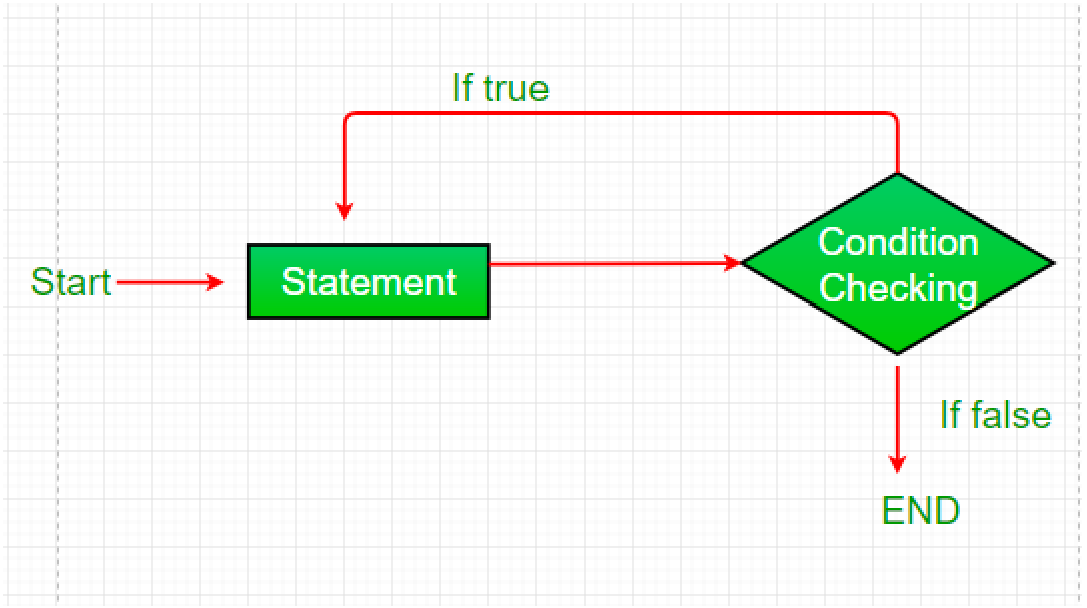
The Enhanced For loop

```
String array[] = {"Ron", "Harry", "Hermoine"};

    //enhanced for loop
    for (String x:array)
    {
        System.out.println(x);
    }
```

The 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.**

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



The 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.

The Do-While Loop

```
int x =1;
       do
       {
           // The line will be printed even
           // if the condition is false
           System.out.println("Value of x:" + x);
           x++;
       }
       while (x < 20);
```

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

Q & A

Contact: amitmahadik@synergetics-india.com

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

Thank You