Lecture 02

Object-Oriented Programming

Control Statements

Prepared by **Hok Tin**

Content

- Interactive program
- Type Casting
- Java Expressions
- Decision Making statements
  - if statements
  - switch statement
- Arrays
- Loop statements
  - do while loop
  - while loop
  - for loop
  - for-each loop
- Jump statements
  - break statement
  - continue statement

Interactive program

❑ Reading Input

- Learn to write a program that ***reads data written by a user***
- Learn to use **Scanner** class which allows the user to ***take input from the console***

```
1  import java.util.Scanner;
2
3  public class Test {
4      public static void main(String[] args) {
5          // create Scanner object
6          Scanner input = new Scanner(System.in);
7
8          System.out.print("Enter your name: ");
9          String name = input.nextLine();
10
11         System.out.println("HI " + name);
12
13         // closes the scanner
14         input.close();
15     }
16 }
```

1. Import Scanner class before using it
2. Create Scanner object
3. Waiting for user input
4. Close if no more is used

Output:

```
Enter your name: Sok Sao
HI Sok Sao
```

Interactive program

❑ Java Scanner Methods

- The **Scanner** class provides ***various methods*** that allow us to read inputs of different types.

Method	Description
`nextInt()`	reads an `int` value from the user
`nextFloat()`	reads a `float` value form the user
`nextBoolean()`	reads a `boolean` value from the user
`nextLine()`	reads a line of text from the user
`next()`	reads a word from the user
`nextByte()`	reads a `byte` value from the user `(-128 >= byte <= 127)`
`nextDouble()`	reads a `doubl` e value from the user
`nextShort()`	reads a `short` value from the user
`nextLong()`	reads a `long` value from the user

- Scan user input in Java syntax

```
int age = input.nextInt();

float money = input.nextFloat();

boolean isBig = input.nextBoolean();

String longStr = input.nextLine();

String word = input.next();

byte num = input.nextByte();

double balance = input.nextDouble();

short s = input.nextShort();

long l = input.nextLong();
```

Type Casting

❑ Widening Type Casting

- Java automatically converts one data type to another data type

- Converting **int** to **double**

```
int num = 10;
System.out.println("The integer value: " + num);

double data = num;
System.out.println("The double value: " + data);
```

Output:

```
The integer value: 10

The double value: 10.0
```

- Converting **double** to **int**

```
double pi = 3.14;
System.out.println("The double value: " + pi);

int num = pi;
System.out.println("The integer value: " + num);
```

Output:

```
error: incompatible types: possible
lossy conversion from double to int

int num = pi;
                    ^

1 error
```

Type Casting

❑ Narrowing Type Casting

- We manually convert one data type into another using the ***parenthesis ()***

○ Converting **double** into **int**

```
int num = 10;
System.out.println("The integer value: " + num);

double data = (double)num;
System.out.println("The double value: " + data);
```

Output:

```
The integer value: 10

The double value: 10.0
```

○ Converting **double** to **int**

```
double pi = 3.14;
System.out.println("The double value: " + pi);

int data = (int)pi;
System.out.println("The integer value: " + data);
```

Output:

```
The double value: 3.14

The integer value: 3
```

Type Casting

❑ Converting between String and Int

- Converting **Int** into **String**

```
int num = 10;
System.out.println("The integer value is: " + num);

String data = String.valueOf(num);
System.out.println("The string value is: " + data);
```

Output:

```
The integer value is: 10
The string value is: 10
```

- Converting **String** into **Int**

```
String data = "10";
System.out.println("The string value is: " + data);

int num = Integer.parseInt(data);
System.out.println("The integer value is: " + num);
```

Output:

```
The string value is: 10
The integer value is: 10
```

Java Expressions

❑ Arithmetic Operators

- Used to perform arithmetic operations on variables and data

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operation (Remainder after division)

```
int a = 12, b = 5;

System.out.println("a + b = " + (a + b));
System.out.println("a - b = " + (a - b));
System.out.println("a * b = " + (a * b));
System.out.println("a / b = " + (a / b));
System.out.println("a % b = " + (a % b));
```

Output:

```
a + b = 17
a - b = 7
a * b = 60
a / b = 2
a % b = 2
```

Java Expressions

❑ Assignment operators

- Operators are symbols that perform operations on variables and values
- It assigns the value on its right to the variable on its left

Operator	Example	Equivalent to
=	a = b;	a = b;
+=	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

```java
int tmp, a = 4;

tmp = a;
System.out.println("variable using =: " + tmp);

tmp += a;
System.out.println("variable using +=: " + tmp);

tmp *= a;
System.out.println("variable using *=: " + tmp);
```

Output:

```
variable using =: 4

variable using +=: 8

variable using *=: 32
```

Java Expressions

❑ Comparison/Relational Operators

- All of these expressions return a boolean value

Operator	Meaning	Example
==	Equal	x == 3
!=	Not equal	x != 3
<	Less than	x < 3
>	Greater than	x > 3
≤	Less than or equal to	x ≤ 3
≥	Greater than or equal to	x ≥ 3

```
// create variables
int a = 7, b = 11;

System.out.println(a == b);  // false
System.out.println(a != b);  // true
System.out.println(a > b);  // false
System.out.println(a < b);  // true
System.out.println(a >= b);  // false
System.out.println(a <= b);  // true
```

❑ Logical Operators

- Check whether an expression is true or false

Operator	Example	Meaning
`&&` (Logical AND)	expression1 **&&** expression2	`true` only if both `expression1` and `expression2` are `true`
`||` (Logical OR)	expression1 **||** expression2	`true` if either `expression1` or `expression2` is `true`
`!` (Logical NOT)	**!**expression	`true` if `expression` is `false` and vice versa

```
System.out.println((5 > 3) && (8 > 5));  // true

System.out.println((5 < 3) || (8 > 5));  // true

System.out.println(!(5 > 3));  // false
```

Decision Making statements

❑ Block Statements

- A block statement is a group of other statements surrounded by **curly bracket {}**
- New block creates a new local scope for the statements inside it

```
1  public class Test {
2      public static void main(String[] args) {
3
4          {
5              int num = 20;
6              num++;
7          }
8
9          Syetem.out.println(num);
10     }
11 }
```

1. Start of block statement

2. End of block statement

Output:

```
Exception in thread "main"
java.lang.Error: Unresolved compilation
problem:

        num cannot be resolved to a
variable
```

Decision Making statements

❑ Decision Making statements

- Decide which block statement to be executed and when
- Evaluate the Boolean expression and control the program flow
- There are ***two types*** of decision-making statements in Java, i.e., **If** statement and **switch** statement
- Execute the bock statement if the ***condition is true***:

- Using ***condition*** to control the ***block statement***

```
Condition {
    int num = 20;
    num++;
    System.out.println(num);
}
```

Ex. Using ***If condition*** to control the ***block statement***

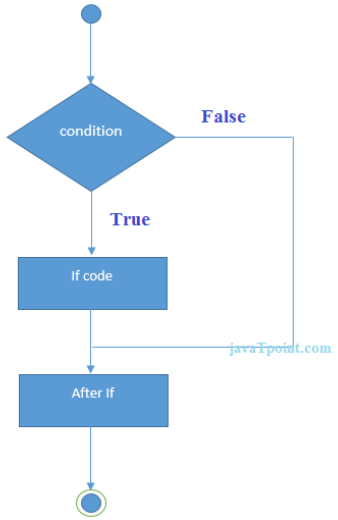
```
if (1 < 2) {
    int num = 20;
    num++;
    System.out.println(num);
}
```

Decision Making statements

IF/ELSE Statements

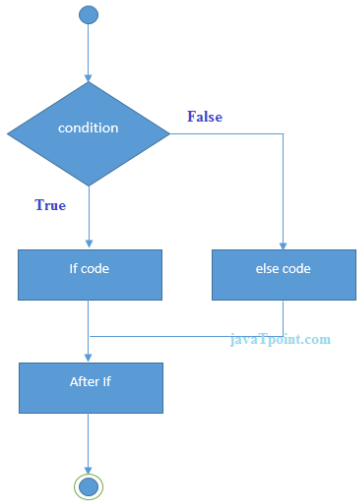
- The Java if statement tests the condition; It executes the if block if condition is true

- **IF**




```
1 public class Test {
2
3     public static void main(String[] args) {
4
5         int x = 5, y = 10;
6         if (x < y) {
7             System.out.println("x is smaller than y");
8         }
9
10    }
11 }
```

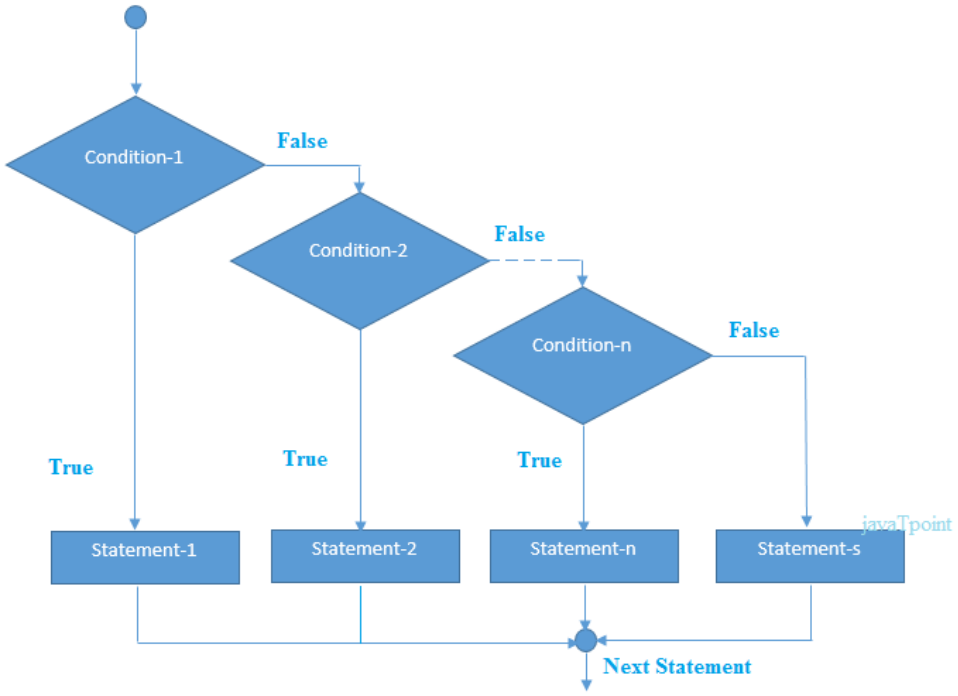
- **IF/Else**




```
1 public class Test {
2
3     public static void main(String[] args) {
4
5         int x = 15, y = 0;
6
7         if (x < y) {
8             System.out.println("x is smaller than y");
9         }
10        else {
11            System.out.println("x is bigger.");
12        }
13
14    }
15 }
```

Decision Making statements

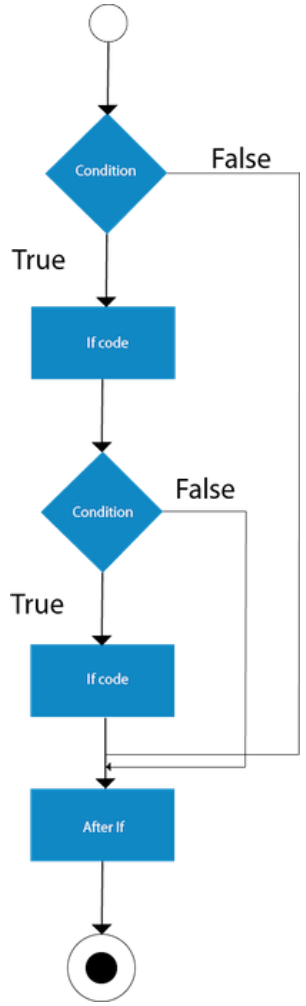
- **ELSE IF Statements**
  - When you have more than two conditions




```
1  public class Test {
2      public static void main(String[] args) {
3          int marks=65;
4
5          if(marks<50){
6              System.out.println("fail");
7          }
8          else if(marks>=50 && marks<60){
9              System.out.println("D grade");
10         }
11         else if(marks>=60 && marks<70){
12             System.out.println("C grade");
13         }
14         else if(marks>=70 && marks<80){
15             System.out.println("B grade");
16         }
17         else if(marks>=80 && marks<90){
18             System.out.println("A grade");
19         }else if(marks>=90 && marks<100){
20             System.out.println("A+ grade");
21         }else{
22             System.out.println("Invalid!");
23         }
24     }
25 }
```

Decision Making statements

- **Nested IF statement**
  - When you have a condition inside another condition




```java
public class Test {
    public static void main(String[] args) {

        //Creating two variables for age and weight
        int age=20;
        int weight=80;

        //applying condition on age and weight
        if(age>=18){
            if(weight>50){
                System.out.println("You are eligible to donate blood");
            }
        }
    }
}
```

Decision Making statements

❑ SWITCH Statements

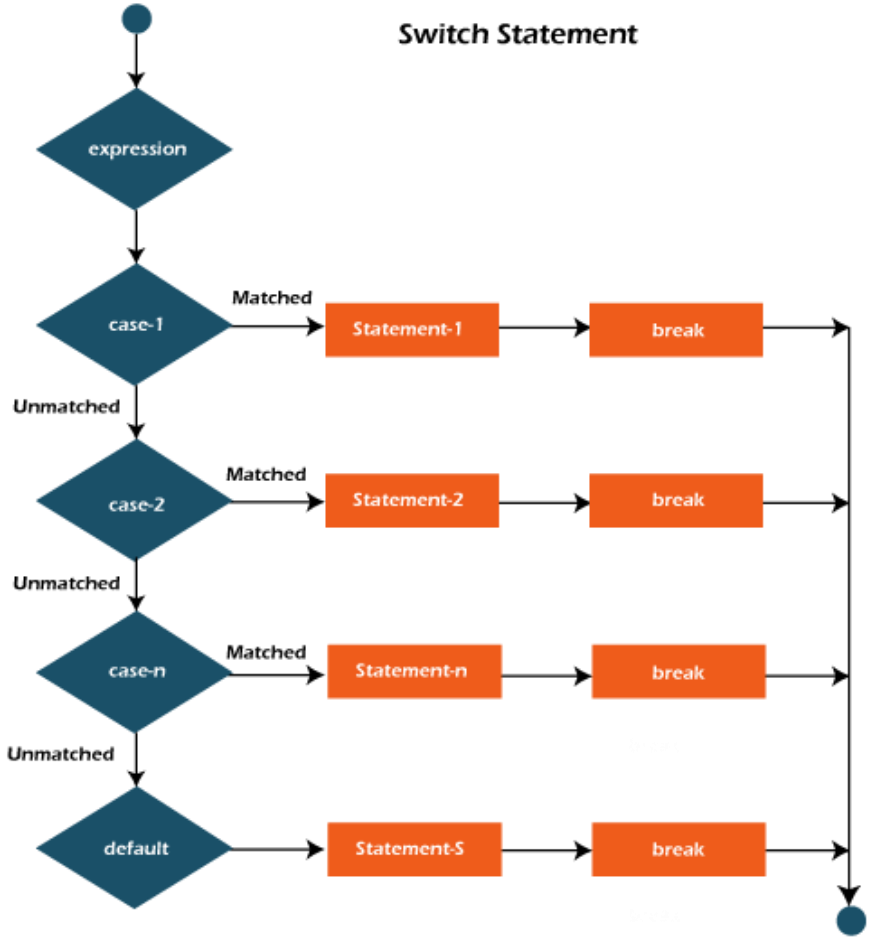
- The Java *switch statement* executes one statement from ***multiple conditions***
- It is like **if-else-if** statement
- The switch statement works with **byte**, **short**, **int**, **long**, **enum** types
- Since Java 7, you can also use **String** in the switch statement

```
switch(expression) {
case value1:
    //code to be executed;
    break;
case value2:
    //code to be executed;
    break;

    ......

default:
    // code to be executed if all cases are not matched;
}
```

- ***N number of case values*** for a switch expression
- The case value must be ***literal*** or ***constant***
- It doesn't allow variables
- The case values must be ***unique***
- In case of ***duplicate value***, it renders compile-time error
- Each case statement can have a break statement which is ***optional***
- The case value can have a ***default label*** which is ***optional***

Decision Making statements

❑ SWITCH Statement with Number case




```java
public class Test {
    public static void main(String[] args) {

        //Declaring a variable for switch expression
        int number=20;

        //Switch expression
        switch(number){
        //Case statements
        case 10:
            System.out.println("10");
            break;
        case 20:
            System.out.println("20");
            break;
        case 30:
            System.out.println("30");
            break;
        //Default case statement
        default:
            System.out.println("Not in 10, 20 or 30");
        }
    }
}
```

Decision Making statements

❑ Fall-through

```
1  public class Test {
2      public static void main(String[] args) {
3
4          int number=20;
5
6          //switch expression with int value
7          switch(number){
8
9          //switch cases without break statements
10         case 10: System.out.println("10");
11         case 20: System.out.println("20");
12         case 30: System.out.println("30");
13         default:System.out.println("Not in 10, 20 or 30");
14
15         }
16     }
17 }
```

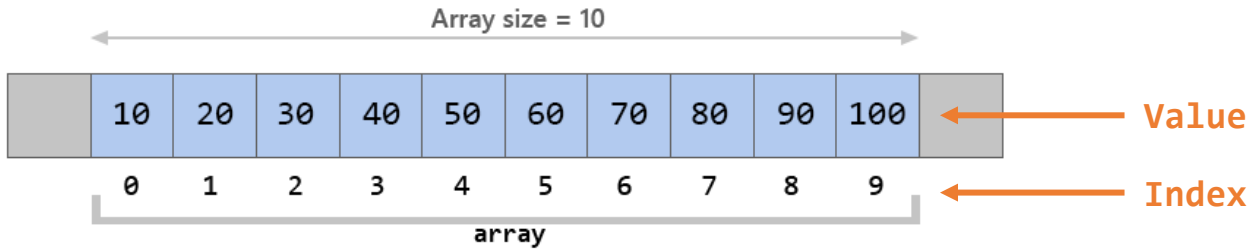
Output:

```
30

30

Not in 10, 20, or 30
```

Arrays

❑ What is arrays in Java?

- Enable you to ***collect objects into*** an easy-to-manage ***list***
- It contains any type of value (base **types** or **objects**)
- But you ***can't store different types*** in a single array



- Following examples are declaration of array:

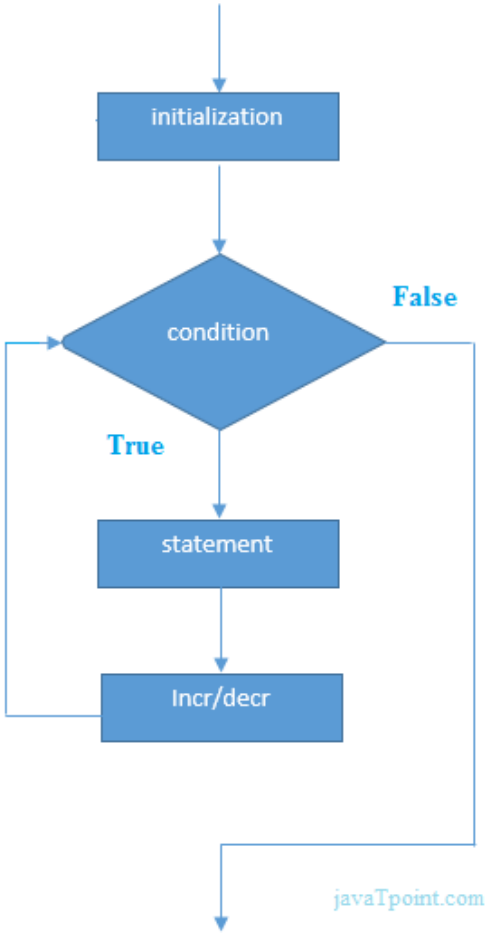
```
String str_arr[] = { "FFFF", "GGGGG"};  // [FFFF, GGGGG]

String[] names = {"John", "Mary", "Bob"}; // [John, Mary, Bob]

String[] cars = new String[] {"Tesla", "Toyota", "Hyundai"}; // [Tesla, Toyota, Hyundai]

int ages[] = {32, 18, 4}; // [32, 18, 4]

int[] nums = new int[10]; // [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Loop statements

❑ for Loop

- **Initialization**: It is the initial condition which is executed once when the loop starts
- **Condition**: It is the second condition which is executed each time to test the condition of the loop
- **Increment/Decrement**: It increments or decrements the variable value
- **Statement**: The statement of the loop is executed each time until the second condition is false

```
for (initialization; condition; increment / decrement) {
  //statement or code to be executed
}
```



Loop statements

☞ **Example #1:**

```
1  public class ForExample {
2    public static void main(String[] args) {
3      //Code of Java for loop
4      for (int i = 1; i <= 10; i++) {
5        System.out.println(i);
6      }
7    }
8  }
```

☞ **Example #2:**

```
1  public class NestedForExample {
2    public static void main(String[] args) {
3      //loop of i
4      for (int i = 1; i <= 3; i++) {
5        //loop of j
6        for (int j = 1; j <= 3; j++) {
7          System.out.println(i + " " + j);
8        } //end of i
9      } //end of j
10   }
11 }
```

Loop statements

❑ for-each Loop

- It is easier to use than simple for loop because we don't need to increment value and use subscript notation
- It works on the basis of elements and not the index
- It returns element one by one in the defined variable

```
for (data_type variable: array_name) {
  //code to be executed
}
```

Loop statements

☞ **Example #1:**

```
1  public class ForEachExample {
2    public static void main(String[] args) {
3      //Declaring an array
4      int arr[] = {12, 23, 44, 56, 78 };
5      //Printing array using for-each loop
6      for (int i: arr) {
7        System.out.println(i);
8      }
9    }
10 }
```

☞ **Example #2:**

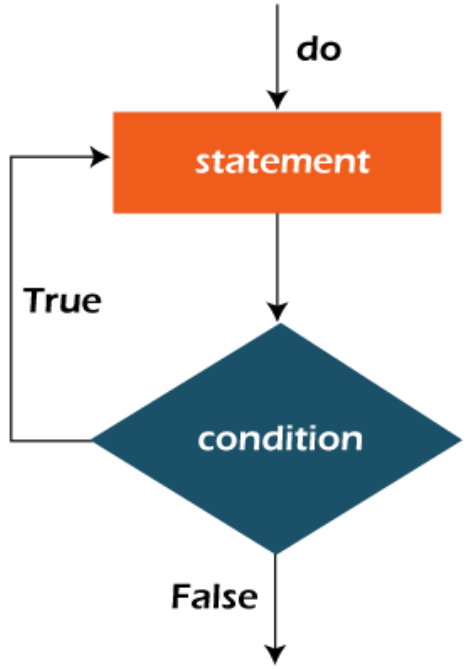
```
1  public class ForEachExample {
2    public static void main(String[] args) {
3      //Declaring an array
4      String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
5      //Printing array using for-each loop
6      for (String i: cars) {
7        System.out.println(i);
8      }
9    }
10 }
```

Loop statements

❑ do-while Loop

- Used to ***iterate a block statement repeatedly***, unless the specified condition is true.
- ***Executed at least once*** because condition is checked after loop body
- do-while loop is called an **exit control loop**

```
do{
 //code to be executed / loop body
 //update statement
}while (condition);
```



Loop statements

☞ **Example #1:**

```
1 public class DoWhileExample {
2   public static void main(String[] args) {
3     int i = 1;
4     do {
5       System.out.println(i);
6       i++;
7     } while (i <= 10);
8   }
9 }
```

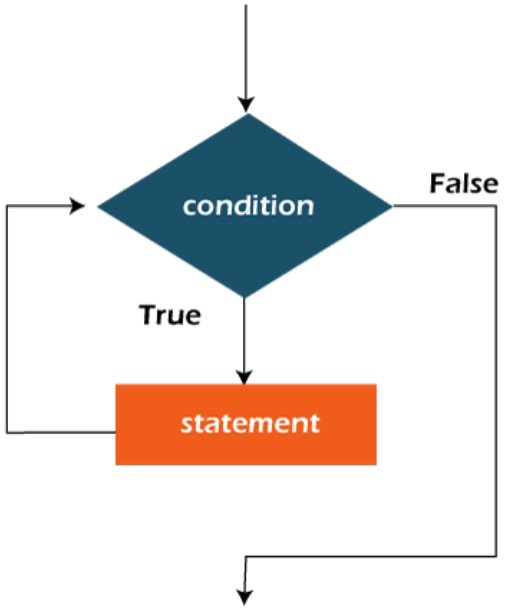
☞ **Example #2:**

```
1 public class DoWhileExample2 {
2   public static void main(String[] args) {
3     do {
4       System.out.println("HIII~");
5     } while (true);
6   }
7 }
```

Loop statements

❑ while Loop

- Used to ***iterate a block statement repeatedly*** unless the specified Boolean condition is true
- As soon as the Boolean condition becomes false, the loop automatically stops
- It is considered as a repeating if statement




```
while (condition) {
  //code block
  //to be executed
}
```

Loop statements

☞ **Example #1:**

```
1 public class WhileExample {
2   public static void main(String[] args) {
3     int i = 1;
4     while (i <= 10) {
5       System.out.println(i);
6       i++;
7     }
8   }
9 }
```

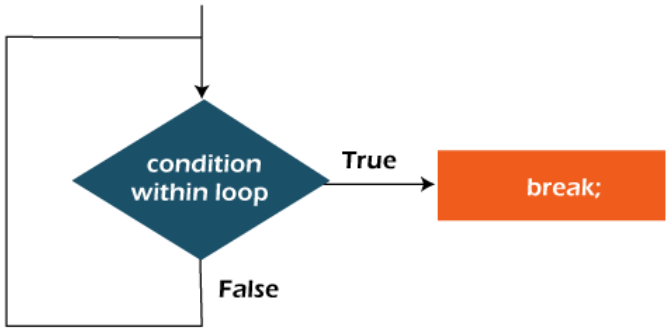
☞ **Example #2:**

```
1 public class WhileExample2 {
2   public static void main(String[] args) {
3     while (true) {
4       System.out.println("HIII~~");
5     }
6   }
7 }
```

Jump statements

❑ Break Statement

- Used to ***break loop*** *(any For and While loop)* or *switch* 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



Flowchart of break statement

```
while (testExpression) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

```
do {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

Loop statements

Example: Break Statement in for loop

```java
public class BreakExample {
  public static void main(String[] args) {
    //using for loop
    for (int i = 1; i <= 10; i++) {
      if (i == 5) {
        //breaking the loop
        break;
      }
      System.out.println(i);
    }
  }
}
```

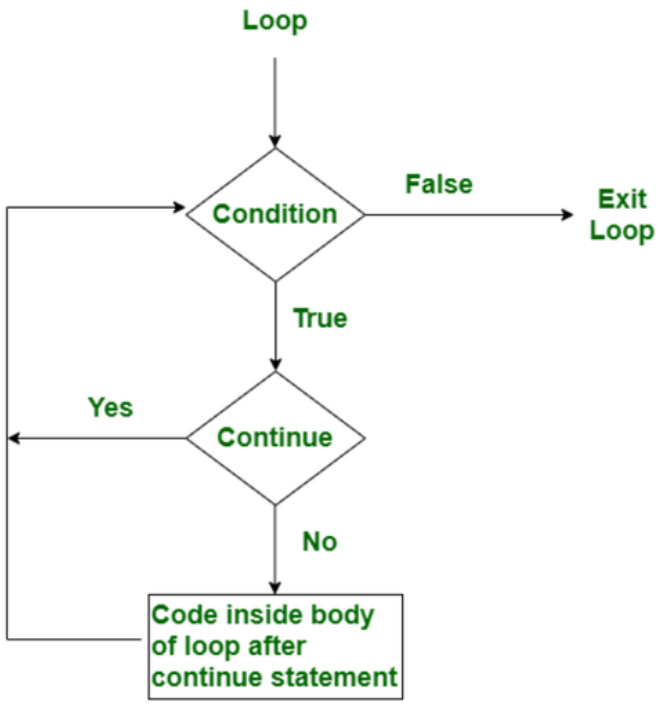
Example: Break Statement in while loop

```java
public class BreakExample {
    public static void main(String[] args) {
        int i = 0;
        while (i < 10) {
            System.out.println(i);
            i++;
            if (i == 4) {
                break;
            }
        }
    }
}
```

Jump statements

❑ Continue Statement

- In loop, when you need to ***jump to the next iteration*** of the loop immediately
- It can be used with ***for loop*** or ***while loop***
- It continues the current flow of the program and ***skips the remaining code*** at the specified condition




```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
} while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

Loop statements

☞ **Example: Continue Statement in for loop**

```java
public class ContinueExample {
  public static void main(String[] args) {
    //for loop
    for (int i = 1; i <= 10; i++) {
      if (i == 5) {
        //using continue statement
        continue; //it will skip the rest statement
      }
      System.out.println(i);
    }
  }
}
```

☞ **Example: Continue Statement in while loop**

```java
public class ContinueWhileExample {
  public static void main(String[] args) {
    //while loop
    int i = 1;
    while (i <= 10) {
      if (i == 5) {
        //using continue statement
        i++;
        continue; //it will skip the rest statement
      }
      System.out.println(i);
      i++;
    }
  }
}
```

Good luck 👌

Refs:

- Teach Yourself JAVA in 21 Days, Laura Lemay and Charles L. Perkins
- https://www.programiz.com/java-programming
- https://www.javatpoint.com/java-tutorial
- https://java-programming.mooc.fi/