Java SE & Spring

Module 1: Java SE

04.Flow Control Statements

Flow Control Statements

- The statements inside your source files are generally executed from top to bottom, in the order that they appear. Control flow statements, however, break up the flow of execution by employing decision making, looping, and branching, enabling your program to conditionally execute particular blocks of code.
- All control flow statements are associated with a business condition – when `true`, the code block executes; when `false` it is skipped.

Conditions and if-else statements

You can use conditions to perform different actions for different decisions. Java has the following conditional statements:

- Use **`if`** to specify a block of code to be executed, if a specified condition is true
- Use **`else`** to specify a block of code to be executed, if the same condition is false
- Use **`else if`** to specify a new condition to test, if the first condition is false
- Use **`switch`** to specify many alternative blocks of code to be executed

---

Further reading: https://www.w3schools.com/java/java_conditions.asp

The `if` statement

The **`if`** statement is the most basic of all the control flow statements. It tells your program to execute a certain section of code only if a particular test evaluates to **`true`**.

Syntax:

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

Examples:

```
boolean isCarMovingFast = true;
int speed = 130;

if (isCarMovingFast){
   speed -= 40;
}
```

```
int yourAge = 21;

if(yourAge <= 12){
    System.out.println("You're a
child!");
}
```

The `if-else` statement

The **`if-else`** statement provides a secondary path of execution when an "**`if`**" clause evaluates to false.

Syntax:

```
1  if (condition) {
2    // block of code to be executed if the condition is true
3  } else {
4    // block of code to be executed if the condition is false
5  }
```

Example:

```
1  if (isCarMovingFast){
2      System.out.println("Please drive slowly!");
3  }
4
5  else {
6      System.out.println("Thank you! Keep driving");
7  }
```

The `if-else if` statement

Use the else **`if`** statement to specify a new condition **`if`** the first condition is false.

Syntax:

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
// block of code to be executed if the condition1 is false and condition2 is
true
} else {
// block of code to be executed if the condition1 is false and condition2 is
false
}
```

Example:

```
if(age <= 12){
   System.out.println("You're a child!");
}
else if (age >=13 && age <= 19){
   System.out.println("You're a teenager!");
}
else{
   System.out.println("You're an adult.");
}
```

The `switch` statement

Unlike **`if`** and **`if-else`** statements, the **`switch`** statement can have a number of possible execution paths. A **`switch`** works with the **`byte`**, **`short`**, **`char`**, and **`int`** primitive data types. It also works with enumerated types (discussed in Enum Types), the String class, and a few special classes that wrap certain primitive types: Character, Byte, Short, and Integer(We will discuss wrapper classes later.)

The execution logic of the **`switch`** statement:

- The **`switch`** expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- The **`break`** and **`default`** keywords are optional.

The `switch` statement

Syntax:

```
switch (switch-expression)
{
        case label-1:
                statements;
        break;

        case label-2:
                statements;
        break;

        case label-3:
                statements;
        break;

        default:
                statements;
}
```

The switch statement

Example:

```
1  int day = 4;
2  switch (day) {
3    case 1:
4    case 2:
5    case 3:
6    case 4:
7    case 5:
8      System.out.println("Today is a weekday! Study hard ;)");
9      break;
10   case 6:
11     System.out.println("Today is Saturday");
12     break;
13   case 7:
14     System.out.println("Today is Sunday");
15     break;
16   default:
17     System.out.println("Sorry, there are only seven days in a week :)");
18 }
```

Loops

Loops can execute a block of code as long as a specified condition is reached, in other words, loops are used to repeat a block of code. For example, if you want to show a message 100 times, then you can use a loop. Loops are handy because they save time, reduce errors, and they make code more readable. There are four types of loops in Java:

1. for loop
2. for-each loop (enhanced for loop)
3. while loop
4. do-while loop

Further reading: https://www.w3schools.com/java/java_conditions.asp

for loop

When we know exactly how many times we want to loop through a block of code, we can use the **for** loop

Syntax:

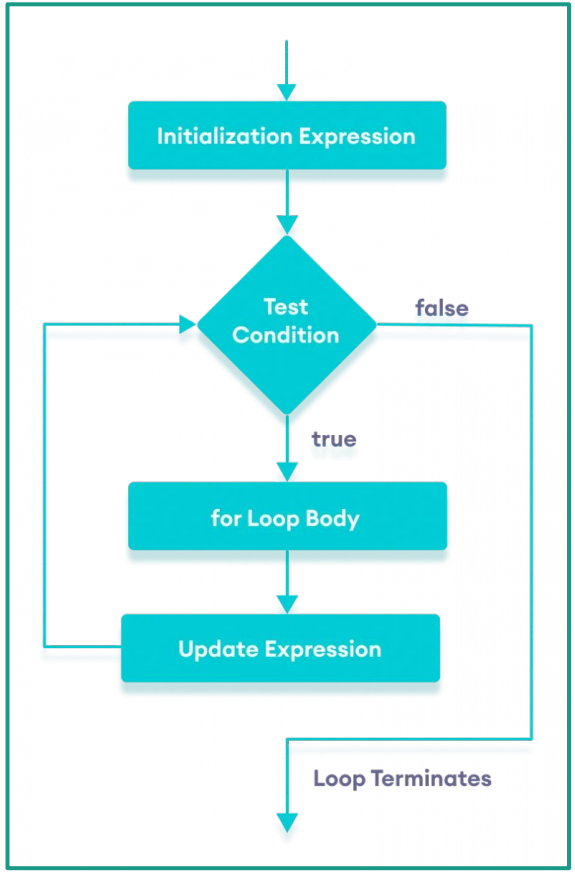
```
1  for (initialExpression; testExpression; updateExpression) {
2      // body of the loop
3  }
```

- The **initialExpression** initializes and/or declares variables and **is executed only once**.
- The **testExpression** is evaluated. If the test condition is true, the body of the **for** loop is executed.
- The **updateExpression** updates the value of **initialExpression**.
- The **testExpression** is evaluated again. The process continues until the condition is false.

Further reading: https://www.programiz.com/java-programming/for-loop

for loop

Flowchart:



for loop

Example:

```
1  public static void main(String[] args) {
2
3      int sum = 0;
4      int n = 100;
5
6      // for loop
7      for (int i = 0; i <= n; i++) {
8        // body inside for loop
9        sum += i;      // sum = sum + i
10     }
11
12     System.out.println("Sum = " + sum);
13   }
14 }
15
16 // Output: Sum = 5050
```

for-each loop

In Java, the for-each loop is used to iterate through elements of arrays and collections (like ArrayList). It is also known as the enhanced for loop.

Syntax:

```
1  for(dataType item : array[]) {
2      ...
3  }
4  //or
5  for(dataType item : Collection<>) {
6      ...
7  }
```

- array/collection - an array or a collection
- item - each item of array/collection is assigned to this variable
- dataType - the data type of the array/collection

Further reading: https://www.programiz.com/java-programming/for-loop

for-each loop

Examples:

```
1  // create an array
2  int[] numbers = {12, 7, 0, 6, -5};
3
4  // for each loop
5  for (int number: numbers) {
6     System.out.println(number);
7  }
```

```
1  String[] programmingLanguages = {"Java", "C++", "Javascript", "Python"};
2  for (String language : programmingLanguages) {
3    System.out.println(language);
4  }
```

`while` loop

Java **`while`** loop is used to run a specific code until a certain condition is met.

Syntax:

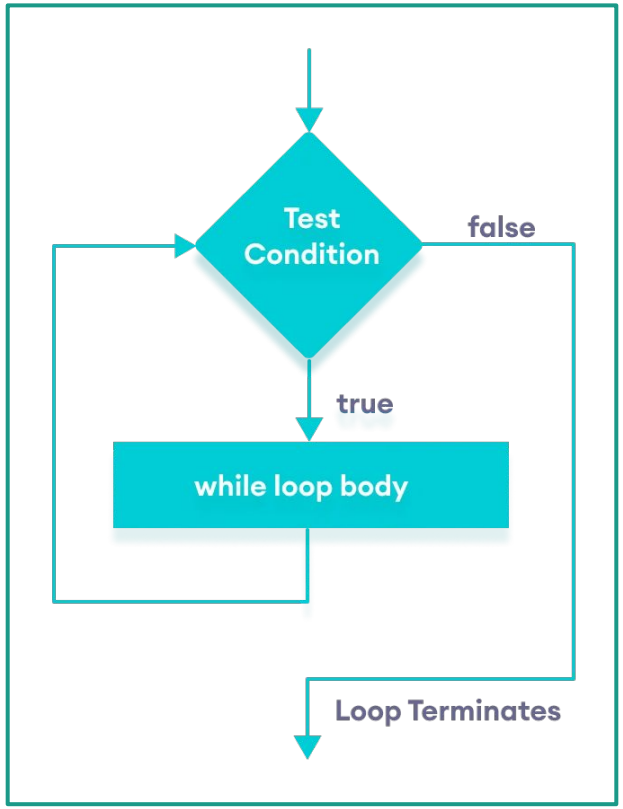
```
while (testExpression) {
    // body of loop
}
```

1. A **`while`** loop evaluates the textExpression inside the parenthesis ().
2. If the textExpression evaluates to true, the code inside the while loop is executed.
3. The textExpression is evaluated again.
4. This process continues until the textExpression is false.
5. When the textExpression evaluates to false, the loop stops.

Further reading: https://www.programiz.com/java-programming/for-loop

while loop

Flowchart:



while loop

Example 1:

```
1  // declare variables
2  int i = 1, n = 5;
3
4  // while loop from 1 to 5
5  while(i <= n) {
6    System.out.println(i);
7    i++;
8  }
```

while loop

Example 2:

```
1  int sum = 0;
2
3      // create an object of Scanner class
4      Scanner input =  new Scanner(System.in);
5
6      // take integer input from the user
7      System.out.println("Enter a number");
8      int number = input.nextInt();
9
10     // while loop continues
11     // until entered number is positive
12     while (number >= 0) {
13       // add only positive numbers
14       sum += number;
15
16       System.out.println("Enter a number");
17       number = input.nextInt();
18     }
19
20     System.out.println("Sum = " + sum);
21     input.close();
```

do-while loop

The **do...while** loop is similar to while loop. However, the body of **do...while** loop is executed once before the test expression is checked.

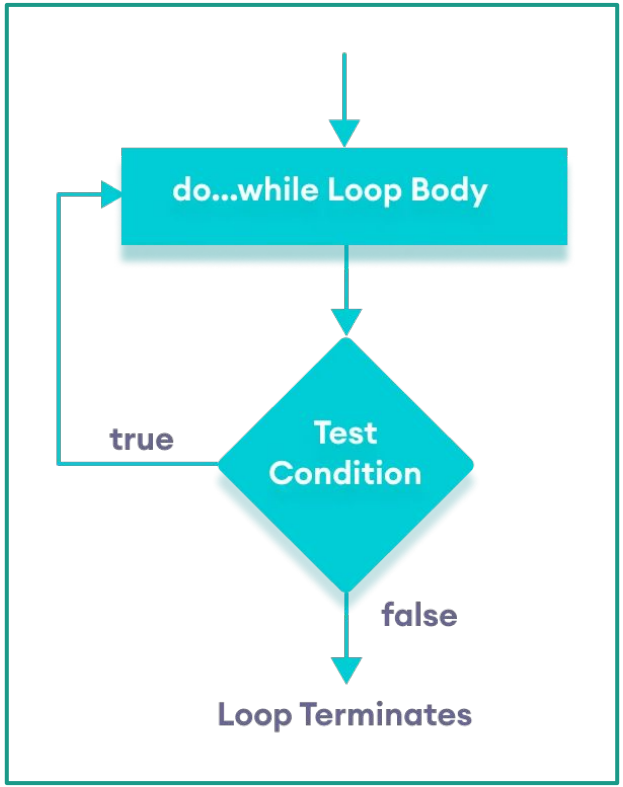
Syntax:

```
do {
    // body of loop
} while(textExpression)
```

1. The body of the loop is executed at first. Then the **textExpression** is evaluated.
2. If the **textExpression** evaluates to true, the body of the loop inside the do statement is executed again.
3. The **textExpression** is evaluated once again.
4. If the **textExpression** evaluates to true, the body of the loop inside the do statement is executed again.
5. This process continues until the **textExpression** evaluates to false. Then the loop stops.

do-while loop

Flowchart:



do-while loop

Example 1:

```
1  // declare variables
2  int i = 1, n = 5;
3
4  // while loop from 1 to 5
5  while(i <= n) {
6    System.out.println(i);
7    i++;
8  }
```

do-while loop

Example 2:

```
1  int sum = 0;
2  int number = 0;
3
4  // create an object of Scanner class
5  Scanner input = new Scanner(System.in);
6
7  // do-while loop continues
8  // until entered number is positive
9  do {
10  // add only positive numbers
11    sum += number;
12    System.out.println("Enter a number");
13    number = input.nextInt();
14    } while(number >= 0);
15
16 System.out.println("Sum = " + sum);
17 input.close();
```

`break` Statement

- While working with loops, it is sometimes desired to skip some statements inside the loop or terminate the loop immediately without checking the test expression. In such cases, **`break`** and **`continue`** statements are used.
- The **`break`** statement in Java terminates the loop immediately, and the control of the program moves to the next statement following the loop.
- It is almost always used with decision-making statements (Java **`if-else`** Statement).
- We have already seen the break statement used in an earlier slide of this presentation(**`switch`**).

Further reading: https://www.programiz.com/java-programming/for-loop

break Statement

Example:

```
1  for (int i = 1; i <= 10; ++i) {
2
3  // if the value of i is 5 the loop terminates
4    if (i == 5) {
5       break;
6    }
7    System.out.println(i);
8  }
```

Further reading: https://www.programiz.com/java-programming/for-loop

`continue` Statement

- The **`continue`** statement skips the current iteration of a loop (for, while, do...while, etc).
- After the continue statement, the program moves to the end of the loop. And, test expression is evaluated.
- Breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

Further reading: https://www.w3schools.com/java/java_break.asp

continue Statement

Example:

```
1  for (int i = 1; i <= 10; ++i) {
2
3  // if value of i is between 4 and 9 continue is executed
4    if (i > 4 && i < 9) {
5      continue;
6    }
7    System.out.println(i);
8  }
```

Further reading: https://www.programiz.com/java-programming/for-loop

Practice Time: Let’s Code Together!

Let’s code together!🚀

📌**Exercise 1:**

1. Define a variable that holds the current balance of the bank account.
2. Assume we withdraw some amount of money, define the amount of withdraw.
3. Check and notify the result for whether the balance is enough for withdraw process.

📌**Exercise 2:**

1. Calculate the total amount between two positive numbers. You can define first and last numbers, also increment amount. Hint: use for-loop 😉
2. Do the calculation above with while and do-while loops.

Questions ?

Next: Arrays