Java Fundamentals Part 2

1. Loops

Loops can execute a block of code as long as a specified condition is reached.

Loops are handy because they save time, reduce errors, and they make code more readable.

1.1. Java While Loop

The `while` loop loops through a block of code as long as a specified condition is `true`:

1.1.1. Syntax

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

In the example below, the code in the loop will run, over and over again, as long as a variable (i) is less than 5:

1.1.2. Example

```
int i = 0;
while (i < 5) {
  System.out.println(i);
  i++;
}
```

> **Note:** Do not forget to increase the variable used in the condition, otherwise the loop will never end!

1.2. The Do/While Loop

The `do/while` loop is a variant of the `while` loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

1.2.1. Syntax

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

The example below uses a `do/while` loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

1.2.2. Example

```
int i = 0;do {
  System.out.println(i);
  i++;
}
while (i < 5);
```

> Do not forget to increase the variable used in the condition, otherwise the loop will never end!

1.3. Java For Loop

When you know exactly how many times you want to loop through a block of code, use the `for` loop instead of a `while` loop:

1.3.1. Syntax

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

- **Statement 1** is executed (one time) before the execution of the code block.
- **Statement 2** defines the condition for executing the code block.
- **Statement 3** is executed (every time) after the code block has been executed.

The example below will print the numbers 0 to 4:

1.3.2. Example

```
for (int i = 0; i < 5; i++) {
  System.out.println(i);
}
```

1.3.3. Example explained

- Statement 1 sets a variable before the loop starts (int i = 0).
- Statement 2 defines the condition for the loop to run (i must be less than 5). If the condition is true, the loop will start

over again, if it is false, the loop will end.

- Statement 3 increases a value (i++) each time the code block in the loop has been executed.

1.3.4. Another Example

```
for (int i = 0; i <= 10; i = i + 2) {
  System.out.println(i);
}
```

1.4. For-Each Loop

There is also a "**for-each**" loop, which is used exclusively to loop through elements in an **array**:

1.4.1. Syntax

```
for (type variableName : arrayName) {
  // code block to be executed
}
```

The following example outputs all elements in the **cars** array, using a "**for-each**" loop:

1.4.2. Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
  System.out.println(i);
}
```

2. Java Break and Continue

2.1. Java Break

You have already seen the `break` statement used in an earlier chapter of this tutorial. It was used to "jump out" of a `switch` statement.

The `break` statement can also be used to jump out of a **loop**.

This example jumps out of the loop when i is equal to 4:

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  System.out.println(i);
}
```

2.2. Java Continue

The `continue` statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 4:

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  System.out.println(i);
}
```

2.3. Break and Continue in While Loop

You can also use `break` and `continue` in while loops:

2.3.1. Break Example

```
int i = 0;
while (i < 10) {
  System.out.println(i);
  i++;
  if (i == 4) {
    break;
  }
}
```

2.3.2. Continue Example

```
int i = 0;
while (i < 10) {
  if (i == 4) {
    i++;
    continue;
  }
  System.out.println(i);
  i++;
}
```

3. Strings

Strings are used for storing text.

A `String` variable contains a collection of characters surrounded by double quotes:

Example

Create a variable of type `String` and assign it a value:

```
String greeting = "Hello";
```

3.1. String Length

A String in Java is actually an object, which contain methods that can perform certain operations on strings. For example, the length of a string can be found with the `length()` method:

Example

```
String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
System.out.println("The length of the txt string is: " + txt.length());
```

3.2. More String Methods

There are many string methods available, for example `toUpperCase()` and `toLowerCase()`:

Example

```
String txt = "Hello World";
System.out.println(txt.toUpperCase());   // Outputs "HELLO WORLD"
System.out.println(txt.toLowerCase());   // Outputs "hello world"
```

3.3. Finding a Character in a String

The `indexOf()` method returns the **index** (the position) of the first occurrence of a specified text in a string (including whitespace):

Example

```
String txt = "Please locate where 'locate' occurs!";
System.out.println(txt.indexOf("locate")); // Outputs 7
```

> Java counts positions from zero. 0 is the first position in a string, 1 is the second, 2 is the third …

4. Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with **square brackets**:

```
String[] cars;
```

We have now declared a variable that holds an array of strings. To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

To create an array of integers, you could write:

```
int[] myNum = {10, 20, 30, 40};
```

4.1. Access the Elements of an Array

You access an array element by referring to the index number.

This statement accesses the value of the first element in cars:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars[0]);
// Outputs Volvo
```

4.2. Change an Array Element

To change the value of a specific element, refer to the index number:

```
cars[0] = "Opel";
```

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
cars[0] = "Opel";
System.out.println(cars[0]);
// Now outputs Opel instead of Volvo
```

4.3. Array Length

To find out how many elements an array has, use the `length` property:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars.length);
// Outputs 4
```

4.4. Loop Through an Array with For-Each

There is also a "**for-each**" loop, which is used exclusively to loop through elements in arrays:

```
for (type variable : arrayname) {
  ...
}
```

The following example outputs all elements in the **cars** array, using a "**for-each**" loop:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
  System.out.println(i);
}
```

The example above can be read like this: **for each** `String` element (called **i** - as in **i**ndex) in **cars**, print out the value of **i**.

If you compare the `for` loop and **for-each** loop, you will see that the **for-each** method is easier to write, it does not require a counter (using the length property), and it is more readable.

4.5. Multidimensional Arrays

A multidimensional array is an array containing one or more arrays.

To create a two-dimensional array, add each array within its own set of **curly braces**:

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
```

myNumbers is now an array with two arrays as its elements.

To access the elements of the **myNumbers** array, specify two indexes: one for the array, and one for the element inside that array. This example accesses the third element (2) in the second array (1) of myNumbers:

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
int x = myNumbers[1][2];
System.out.println(x); // Outputs 7
```

We can also use a `for loop` inside another `for loop` to get the elements of a two-dimensional array (we still have to point to the two indexes):

Arrays


```
public class MyClass {
  public static void main(String[] args) {
    int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
    for (int i = 0; i < myNumbers.length; ++i) {
      for(int j = 0; j < myNumbers[i].length; ++j) {
        System.out.println(myNumbers[i][j]);
      }
    }
  }
}
```

Thank you

全国统一咨询热线：400-690-6115

北京|上海|广州|深圳|天津|成都|重庆|武汉|济南|青岛|杭州|西安