
LETS LEARN JAVA
CODE CONSTRUCTION

Training Plan - Basics

Topics to be covered
Programming constructs – conditionals Mixing and Enhancing Conditions with operators Handling repetitive tasks through looping Looping Constructs Intro to arrays Looping through arrays Using enhanced for loop and forEach for traversing Arrays How to debug code

Topic – Method Definitions in Java

Log in to https://kahoot.it/

Adding Conditions in Code

Normal IF ELSE

Nested IF ELSE

IF ELSE LADDER

SWITCH Statement

Normal IF-ELSE

Normal IF ELSE

Nested IF ELSE

IF ELSE LADDER

SWITCH Statement

Lets program below scenarios

- If the age of a person is greater than 18 then he is eligible to vote
- If age is greater than 60, then status is senior citizen, and if age is less than 22 then status is student, else status is professional
- Accept what a student is studying (10^{th}, 12^{th}, Grad, PostGrad). Depending on that, print message as to what could be his next career option. Use Ladder IF-ELSE and then SWITCH statement
- Find out the first letter of a word and state if it starts with a vowel or consonant. Use IF ELSE and then SWITCH statement

Enhancing Conditions using Operators

- Multiple Operator Types to support different types of calculations and expressions.
- Non exhaustive list of operators are as below. Ones in red are most commonly used:
  - Unary Operators [expr++, expr--, ++expr, --expr, !, ~]
  - Arithmetic Operators [+, -, *, /, %]
  - Assignment Operators [=, +=, -=]
  - Relational Operators [>, <, >=, <=, ==, !=, instanceof] // How to compare two objects?
  - Logical Operators
    - Logical standard Operators [&&, | |]
    - Logical Bitwise Operators [&, |, ^]
  - Ternary Operator [? :]

Let's do hands on to understand these important operators

Doing same thing again and again… Loops

Loops using for

Loops using while

Loops using do.. while

Loop using enhanced-for

Using Streams and Lambda Expressions for looping (Advanced Java)

Basic structure of a loop

```
keep executing this block {
        if (condition for execution met) {
                // statements
        } else {
                // exit from loop
        }
}
```

Let us see this in code

```java
public class GenericLoop {
    public static void main(String[] args) {
        System.out.println("Enter the table name which you want to print : ");
        Scanner scanner = new Scanner(System.in);
        int table = scanner.nextInt();
        scanner.close();

        // Start my counter
        int tableCounter = 1;

        // Infinite Loop and handle everything manually
        while (true) {
            // condition under which the repeatable task needs to be executed
            if (tableCounter <= 10) {
                // what you want to do repeatedly
                System.out.println(String.format("%d x %d = %d", table, tableCounter, (table * tableCounter)));

                tableCounter++; // increment of the counter
            } else {
                break; // exit from loop once the counter has reached 11. If you do not break, then this loop will stay
                       // infinite
            }
        }
    }
}
```

Simplify same loop using conditions with “while”

```java
int tableCounter = 1;

while (true) {
    if (tableCounter <= 10) {
        System.out.println(String.format(
                "%d x %d = %d",
                table, tableCounter,
                (table * tableCounter)));
        tableCounter++;
    } else {
        break;
    }
}
```

```java
int tableCounter = 1;

while (tableCounter <= 10) {
    System.out.println(String.format(
            "%d x %d = %d",
            table, tableCounter,
            (table * tableCounter)));
    tableCounter++;
}
```

Cleaner Code
Implicit handling of exit/break conditions

`while (true)`

This is an infinite loop. Remember to keep a clear exit condition. Better to avoid, and be safe than sorry.

while vs. do-while

```java
int tableCounter = 1;

while (tableCounter <= 10) {
    System.out.println(String.format(
            "%d x %d = %d",
            table, tableCounter,
            (table * tableCounter)));
    tableCounter++;
}
```

First evaluate and then execute

```java
int tableCounter = 1;

do {
    System.out.println(String.format(
            "%d x %d = %d",
            table, tableCounter,
            (table * tableCounter)));
    tableCounter++;
} while (tableCounter <= 10);
```

Execute once, then check condition

for loop – super simplified looping

```
int tableCounter = 1;

while (tableCounter <= 10) {
    System.out.println(String.format(
            "%d x %d = %d",
            table, tableCounter,
            (table * tableCounter)));
    tableCounter++;
}
```

```
for (int tableCounter = 1; tableCounter <= 10; tableCounter++) {
    System.out.println(String.format(
            "%d x %d = %d",
            table, tableCounter, (table * tableCounter)));
}
```

Initialize Counter/Index variable
Evaluate
Increment counter/index

} All in one line

When to use which looping algorithm

- ✓ Looping x number of times → use for loops
- ✓ Looping till some condition is met/violated → use while loops

Lets program below scenarios

- Printing a table using a for loop
- Same scenario using while and do-while techniques
- Accept a set of 10 numbers, and for each do the following:
  - Print if number is even or odd
  - Print squares of only those even numbers that are greater than 10
  - Find sum of cubes of odd numbers less than 10

Arrays – holding many similar things

Think of below examples

- ✓ List of scores of a student for 8 semesters
- ✓ The list of hobbies that a person has
- ✓ List of names of people in a study group/class

In each of the cases, we are storing multiple values of same type in one variable instead of declaring multiple variables

```
float[] semesterGPA = new float[2];
semesterGPA[0] = 8.3f;
semesterGPA[1] = 8.7f;

int[] firstTenNumbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

Arrays – key points to note

```
float[] semesterGPA = new float[2];
semesterGPA[0] = 8.3f;
semesterGPA[1] = 8.7f;

int[] firstTenNumbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

String[] names = new String[] { "Mohit", "Muskaan", "Michael" };
```

- **Homogeneous composition**. All elements of Array can hold only same type of data.
- **Size of array has to be defined upfront**. Once defined/initialized, it **cannot be changed**. Hence Arrays are also called static Arrays.
- Array elements are accessed through position index that starts with 0
- Arrays can be nested. Can you think of a 2-dimensional array.

Arrays – printing elements

```java
String[] names = new String[] { "Mohit", "Muskaan", "Michael" };

System.out.println("\nPrint all names like an array");
System.out.println(Arrays.toString(names));

System.out.println("\nPrint names using normal for loop");
for (int i = 0; i <= names.length - 1; i++) {
    System.out.println(names[i]);
}

System.out.println("\nPrint names using normal for loop");
for (String name : names) {
    System.out.println(name);
}
```

Enhanced for loop is nowadays used most commonly for arrays and collections to avoid buds/issues related to position index

`Arrays.toString(names)`

This is also a common approach, but this is usually used for printing into logs.

Lets program below scenarios

- Accept the semester marks of a Student in an array, and display them with Semester number
- Store a set of 20 random integers in an array, and
  - Calculate the sum of those numbers
  - Find the max value within the array
  - Find the sum of all the even numbers in the array
  - Find the sum of squares of those numbers that are divisible by 6
  - Sort the array and print all consecutive pairs where sum is an even number
- Create an array of Strings (may be names of people) and
  - Find the name with biggest length
  - Find how many names contain the letter ‘o’ in them

When it gets complex, lets use debugger

- See how debugger is used
- Breakpoints
- “Debug as”
- Debug perspective..
- Step in and step out features (F5, F6, F8)
- Inspect element to see the value

That’s all for now. Please practice below

Write a program to accept dimensions of a matrix (n x m), then scan all values and then print the matrix. Use debugger as needed.