
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 for traversing Arrays Basics of lambda expressions (advanced Java)

Lets understand below requirement

- For a given set of 10 random positive numbers (integers), do the following:
  - Print only the odd numbers
  - Print squares of only those even numbers that are greater than 10
  - Count the occurrences of 2 consecutive odd numbers
  - Identify duplicate numbers in the list
  - Calculate the sum of all the numbers
  - Calculate the sum of remainders when each of the numbers are divided by 5

Lets break this down into concepts

- For a given set of 10 random positive numbers (integers), do the following:
  1. Print only the odd numbers
  2. Print squares of only those even numbers that are greater than 10
  3. Count the occurrences of 2 consecutive odd numbers
  4. Identify duplicate numbers in the list
  5. Calculate the sum of all the numbers
  6. Calculate the sum of remainders when each of the numbers are divided by 5
  7. Find out if a number is present in the list

operators conditionals logical looping Array or ArrayList

Conditions and Operators

- **Conditionals** – IF ELSE or SWITCH statements

- 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

Loops

Loops using for

Loops using while

Loop using enhanced-for

```
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

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

- 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