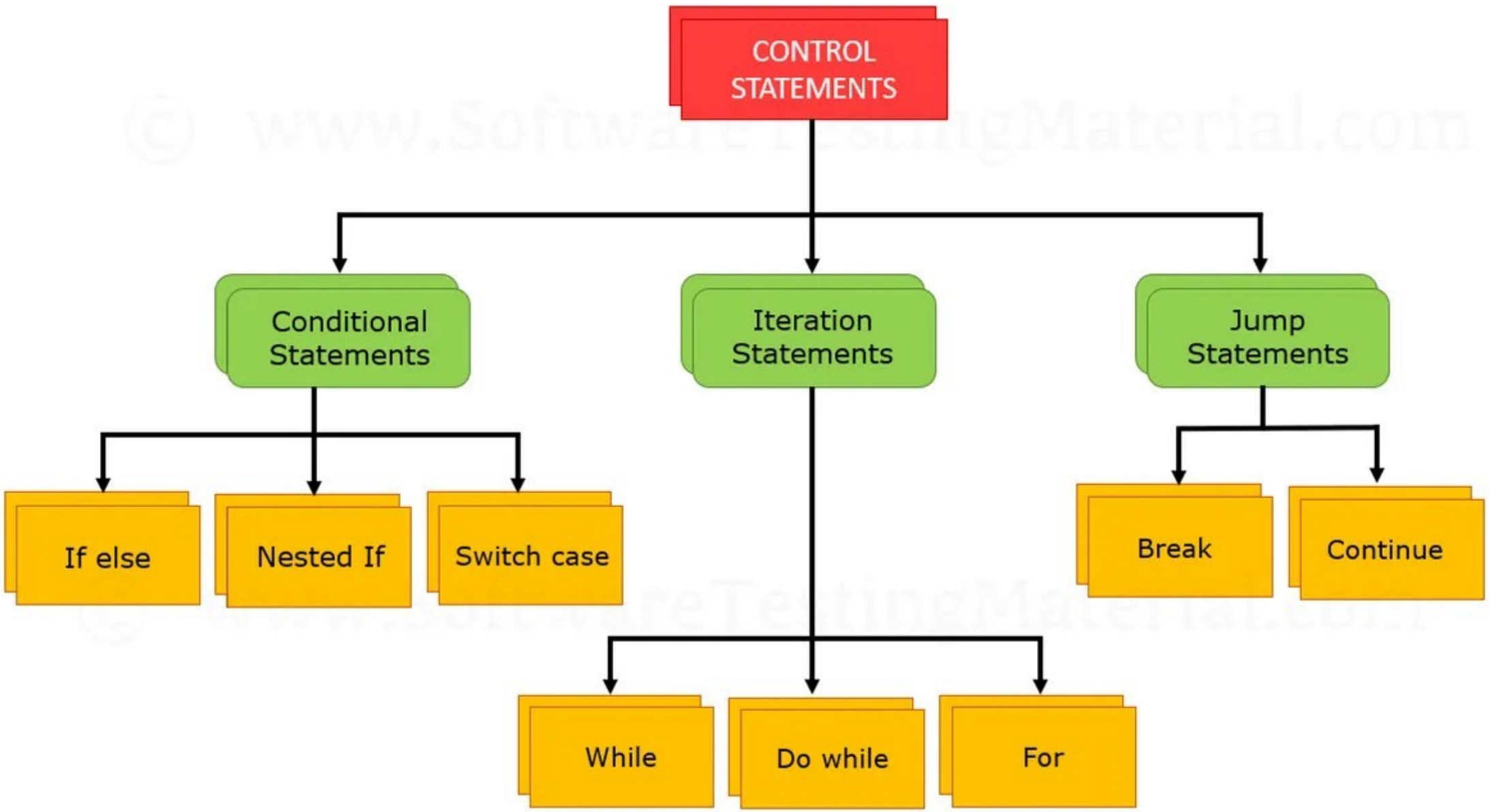
Object Oriented Programming with Java

Zheng Chen

Basic Programming

1. Conditional Statements
2. Iteration Statements
3. Jump Statements
4. Array

Java Control Statements



Basic Programming

1. **Conditional Statements**
2. Iteration Statements
3. Jump Statements
4. Array

Conditional Statements

- allow you to control the flow of the program during run time on the basis of the outcome of an expression or state of a variable.

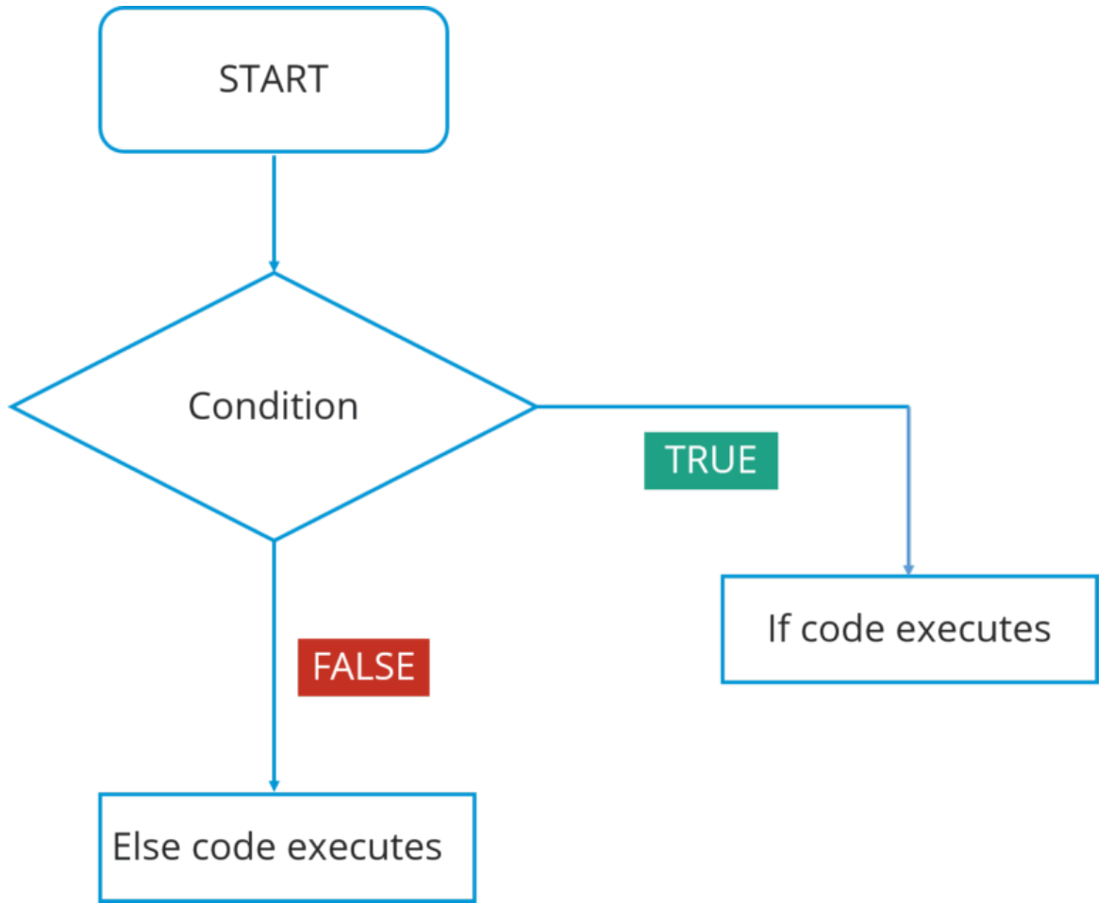
- can be further classified into the following:
  - **If-else Statements**
  - **Switch Statements**

If-else Statements

- **if statement:**
  - tells program to execute a certain section of code only if a particular test evaluates to true.

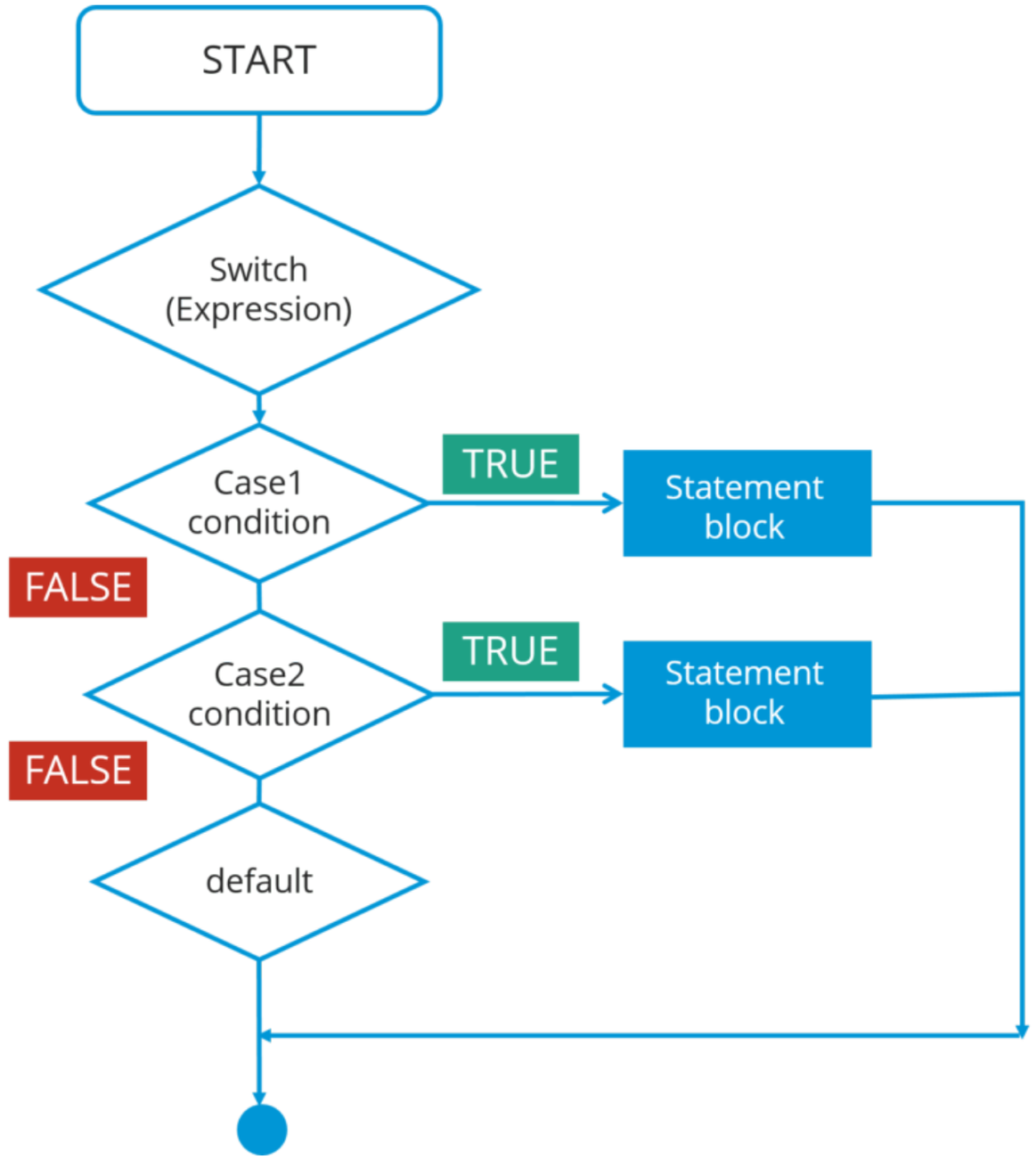
- **Nested if statement:**
  - An if statement inside another the statement.

- **if-else statement:**
  - If a condition is true then the section of code under if would execute else the section of code under else would execute.



Switch Case Statement

- The switch statement in Java is a multi branch statement.
- Switch works with the byte, short, char, and int primitive data types. It also works with enumerated types, the String class, and a few special classes that wrap certain primitive types such as Character, Byte, Short, and Integer.



Basic Programming

1. Conditional Statements
2. **Iteration Statements**
3. Jump Statements
4. Array

Iteration Statements

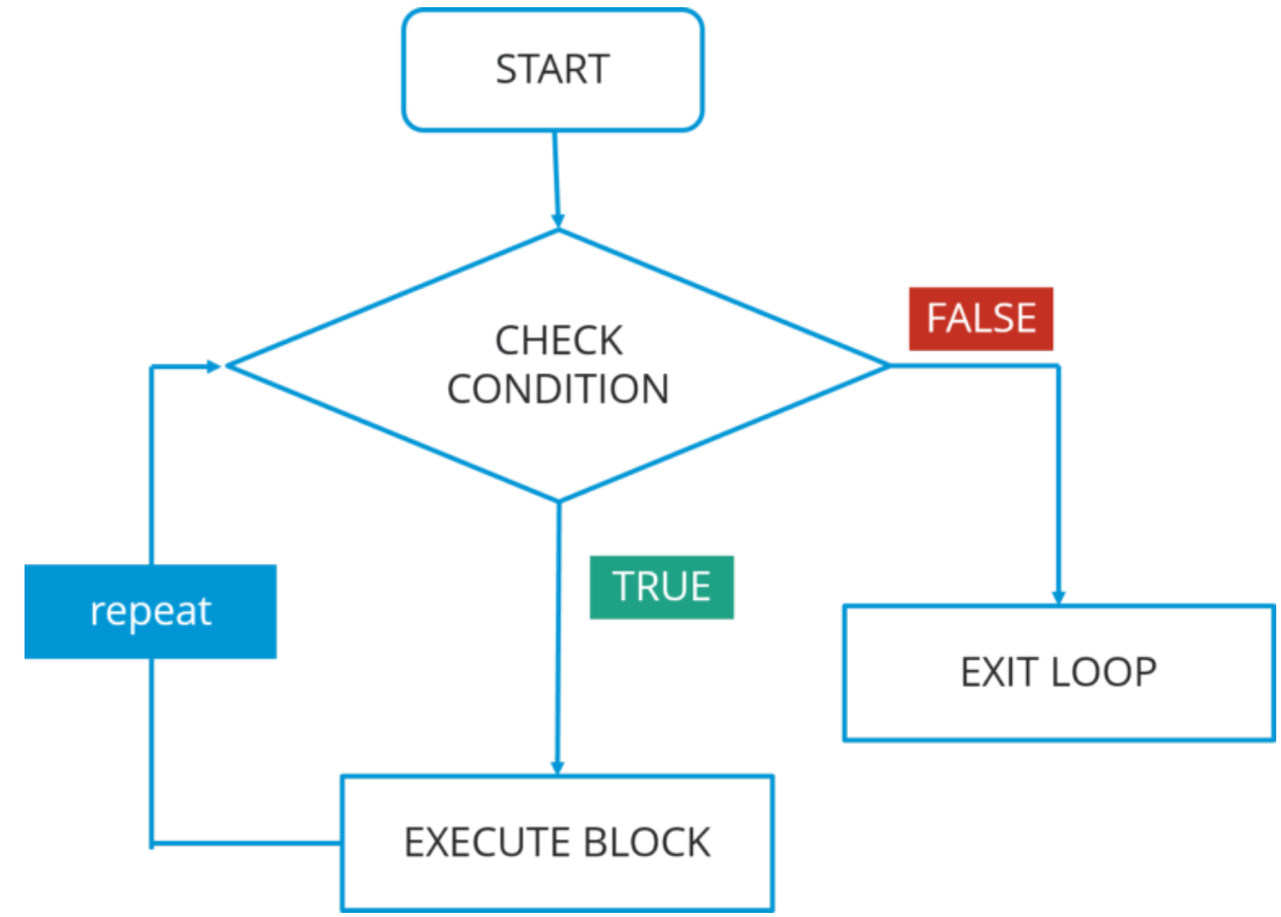
- Java offers three constructs for Iteration:
  - **for**
  - **while**
  - **do-while**

- All iterations have three operations: initialize a variable, test it to see whether we're done, and update the variable to be tested again.

While Statement

- Repeat a group of statements while a given condition is true. It tests the condition before executing the loop body.

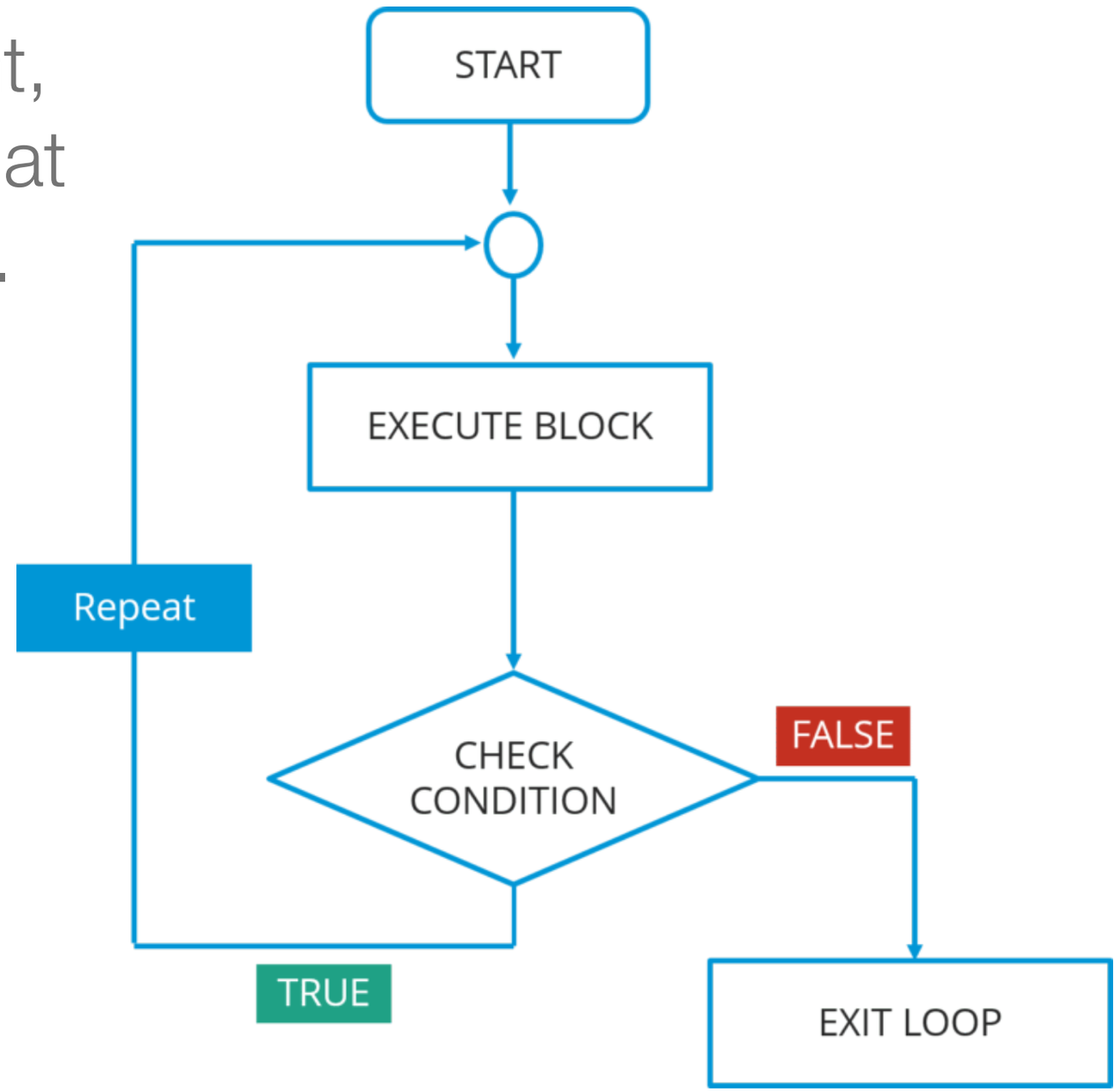
```
while (expression) {
    // statement(s)
}
```



Do-while Statement

- It is like a while statement, but it tests the condition at the end of the loop body. Also, it will executes the program **at least once**.

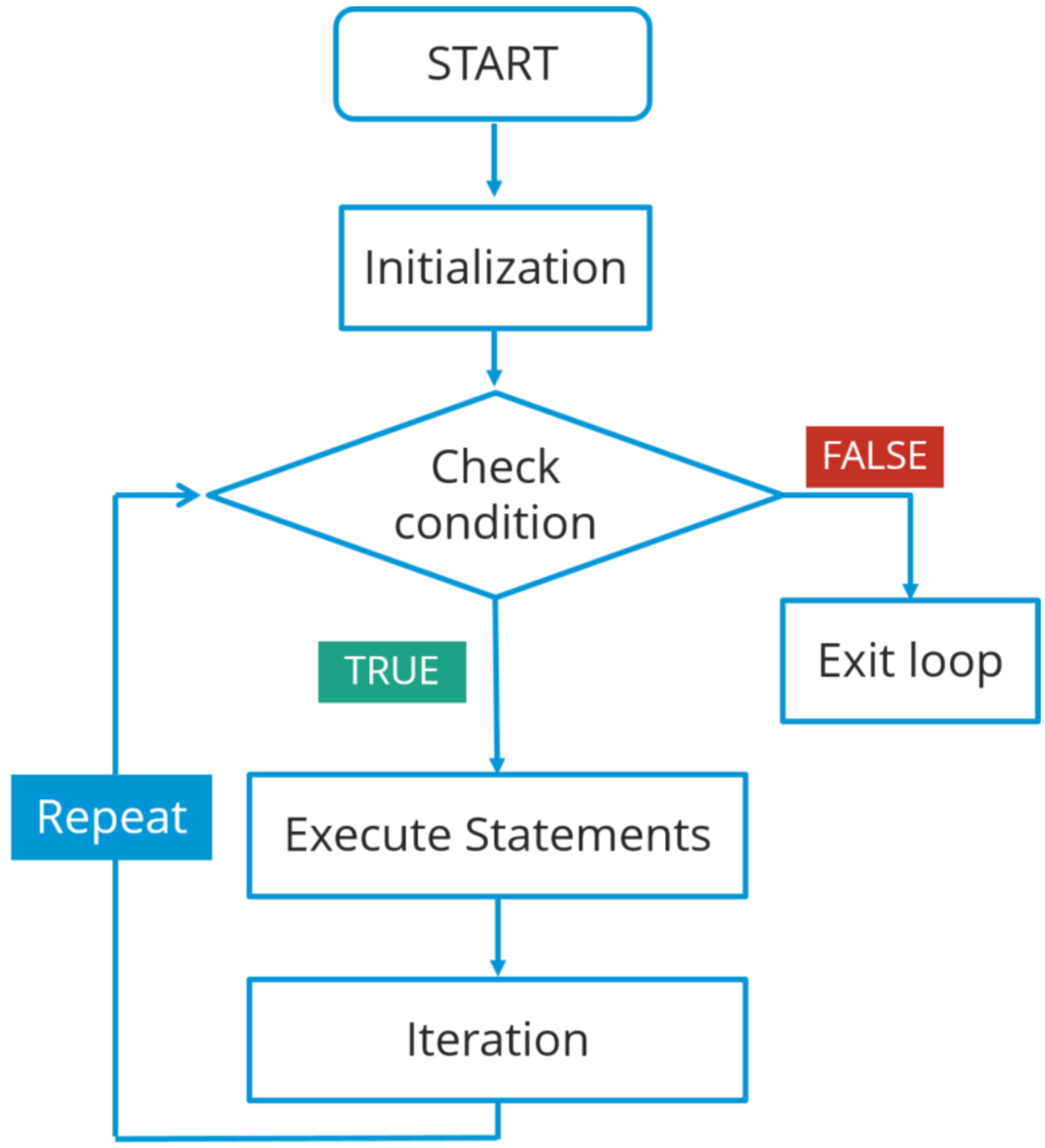
```
do
{
   //statement(s);
} while(condition);
```



For statement

- For statement execute a sequence of statements multiple time where you can manage the loop variable.

```
for (initialization;
termination; increment)
{
    //statement(s)
}
```



Basic Programming

1. Conditional Statements
2. Iteration Statements
3. **Jump Statements**
4. Array

Jump Statements

- Jump statement are used to transfer the control to another part of your program.

- There are two jump statements in Java:

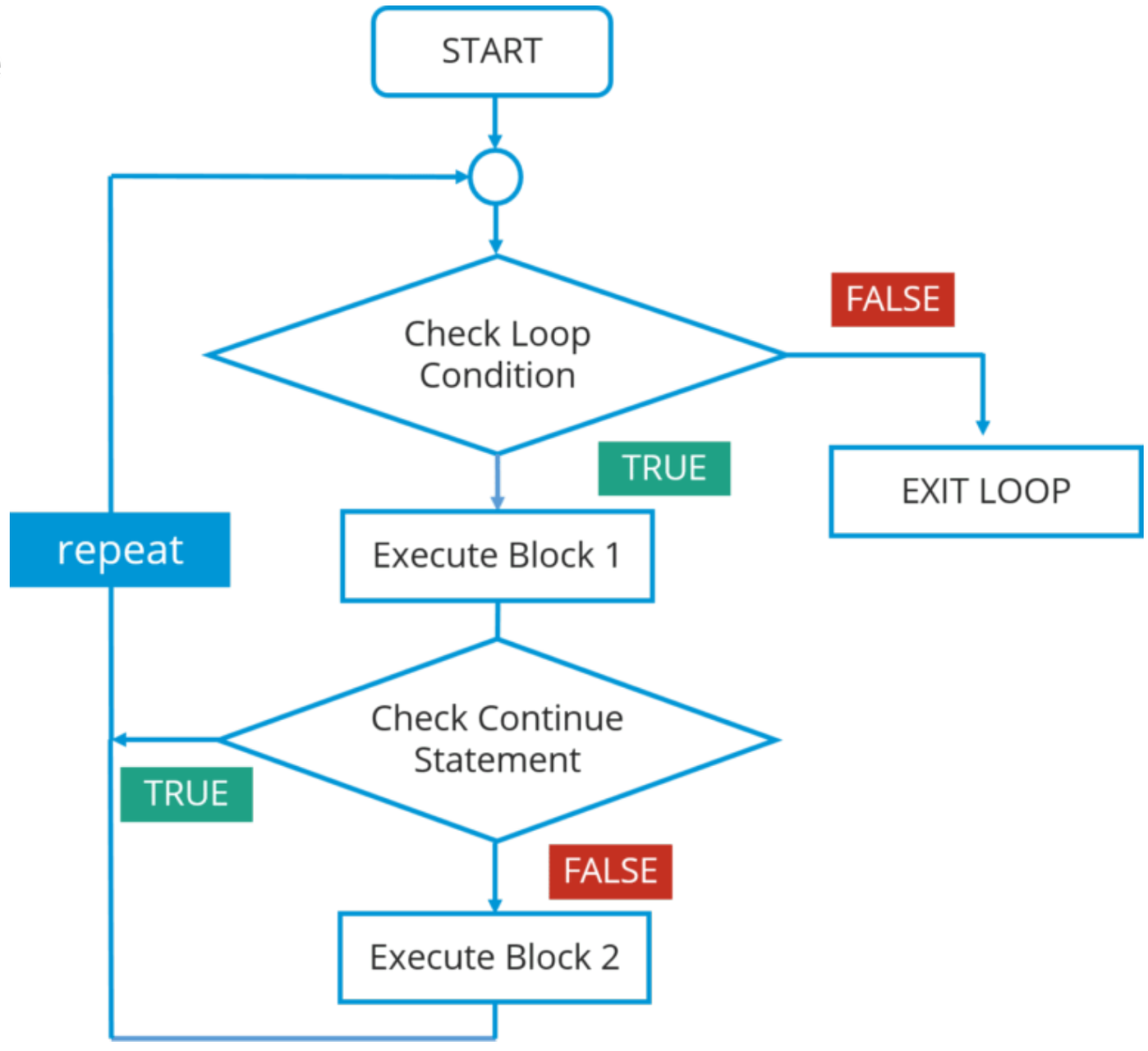
continue

break

~~Goto~~

Continue statement

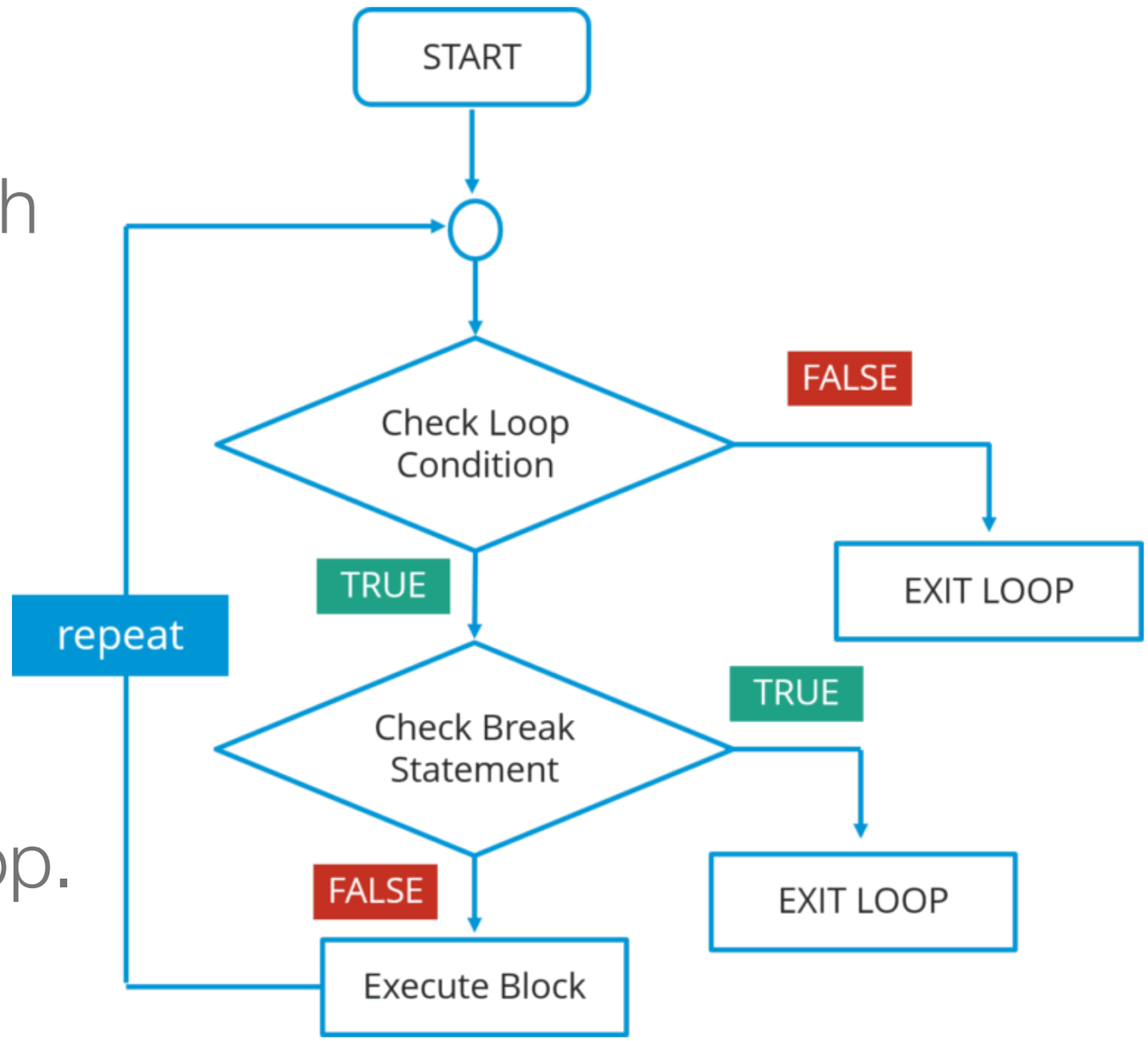
- Whenever the continue statement is encountered inside a loop, control immediately **jumps to the beginning of the loop** for next iteration by skipping the execution of statements inside the body of loop for the current iteration.



Break statement

- The Break statement in Java is used to break a loop statement or switch statement. The Break statement **breaks the current flow** at a specified condition.

- In case of inner loop, it breaks just the inner loop.



Nested Loops and Labels

- Looping structures can be nested.
  - By default, a break or continue statement affects the **innermost loop** in which it is located.

- You can apply an optional **label** to a looping structure.
  - The label is an identifier followed by a : placed before the looping structure.

- Both the break and the continue statement accept an optional label argument.
  - In that case, the break or continue statement affects the looping structure with that label.

Label Example

```
OUTER:
for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= 10; j++) {
        if (i == j) continue OUTER;
    }
}
```

```
public class SearchForNumber2D {
    public static void main (String[] args) {
        int[][] nums = { {1, 3, 7, 5}, {5, 8, 4, 6}, {7, 4, 2, 9} };
        int search = 4;
        FOUND:
        for (int i = 0; i < nums.length; i++) {
            for (int j = 0; j < nums[i].length; j++) {
                if (nums[i][j] == search) {
                System.out.println(
                    "Found " + search + " at position " + i + "," + j);
                break FOUND;
                }
            }
        }
    }
}
```

Label Example 2

Basic Programming

1. Conditional Statements
2. Iteration Statements
3. Jump Statements
4. **Array**

Array

- An array is a group of like-typed variables that are referred to by a common name.

- Array can contains primitives data types as well as objects of a class depending on the definition of array.
  - In case of primitives data types, the actual values are stored in contiguous memory locations.
  - In case of objects of a class, the actual objects are stored in heap segment.

Arrays in Java

- In Java all arrays are **dynamically allocated**.
- Arrays are objects in Java, we can find their length using member **length**.
- A Java array variable can also be declared like other variables with [] after the data type.
- The variables in the array are ordered and each have an index **beginning from 0**.
- The size of an array must be specified by an **int value** and not long or short.
- Every array type implements the interfaces Cloneable and java.io.Serializable.

Declaring Array

- Syntax

  dataType[] arrayRefVar;   // preferred way.

  or

  dataType arrayRefVar[]; // works but not preferred way.

- Example

  double[] myList;   // preferred way.

  or

  double myList[];   // works but not preferred way.

Creating Arrays

- Syntax

  arrayRefVar = new dataType[arraySize];

- Declaring, creating, and assigning combined in one statement

  dataType[] arrayRefVar = new dataType[arraySize];

- Array Literal

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

Accessing Array Elements using for Loop

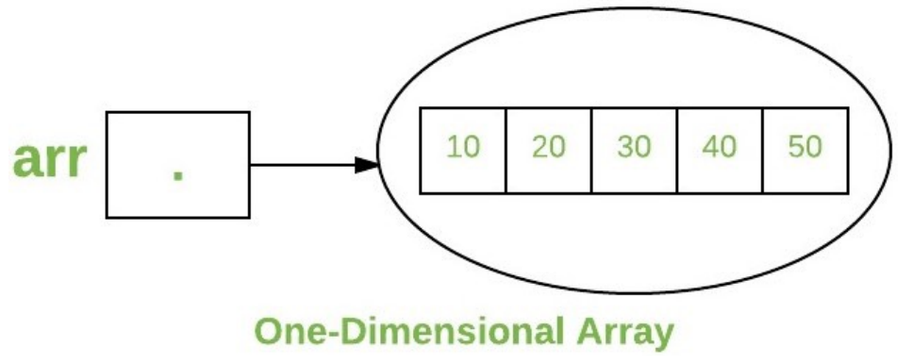
```
for (int i = 0; i < arr.length; i++)
  System.out.println("Element at index " + i + " : "+ arr[i]);
```

——————or——————

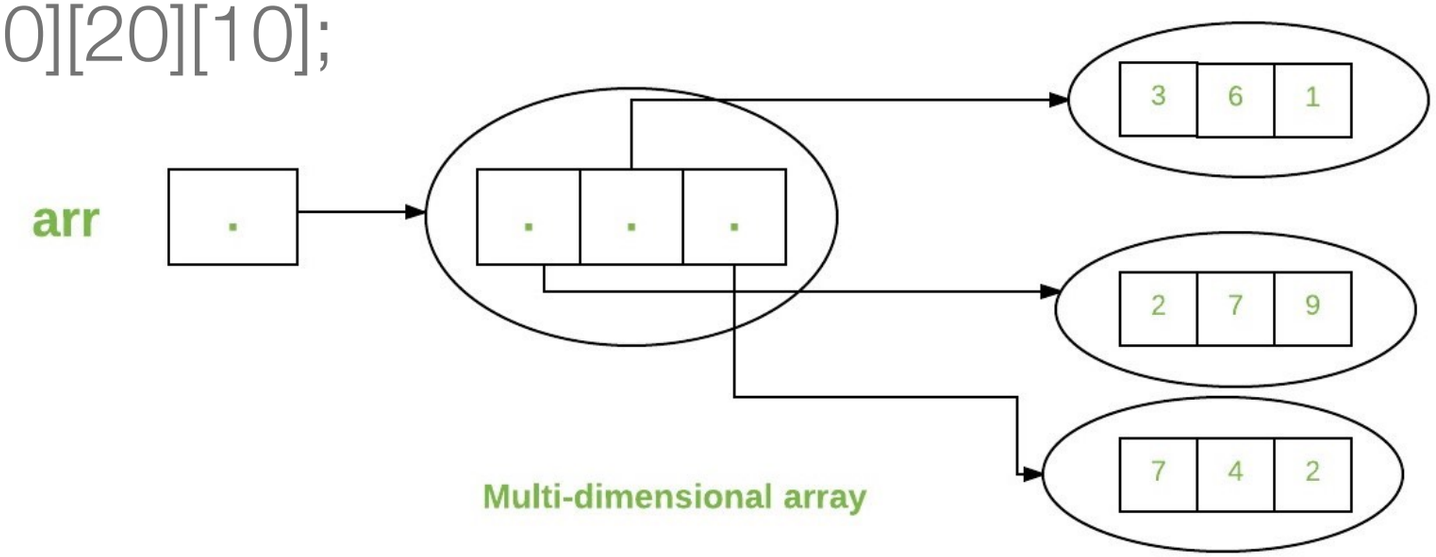
```
for (int i: arr)
   System.out.println("Element in arr " + " : " + i);
```

Multidimensional Arrays

- Multidimensional arrays are arrays of arrays with each element of the array holding the reference of other array.



- int[][] intArray = new int[10][20]; //a 2D array or matrix
- int[][][] intArray = new int[10][20][10]; //a 3D array



The Arrays Class

- The java.util.Arrays class contains various static methods for sorting and searching arrays, comparing arrays, and filling array elements.

```
public static int binarySearch(Object[] a, Object key)

public static boolean equals(Object[] a, Object[] a2)

public static void fill(int[] a, int val)

public static void sort(Object[] a)
```

String

- String is a type that has some of the characteristics of both a primitive and an object.

- String is an object, java.lang.String class defines it. A String object is a sequence of characters.

- Java offers special support for the String class that lets String objects act like primitives.

String Operations

```
String myString = "my string";
String yourString = "your string";
String ourString = myString + " " + yourString;
System.out.println(myString);
System.out.println(yourString);
System.out.println(ourString);
```

———————output———————

- my string
- your string
- my string your string

The String Class

No.	Method	Description
1	char charAt(int index)	returns char value for the particular index
2	int length()	returns string length
5	String substring(int beginIndex)	returns substring for given begin index.
7	boolean contains(CharSequence s)	returns true or false after matching the sequence of char value.
13	String replace(char old, char new)	replaces all occurrences of the specified char value.
16	String[] split(String regex)	returns a split string matching regex.
21	int indexOf(String substring)	returns the specified substring index.
22	int indexOf(String substring, int fromIndex)	returns the specified substring index starting with given index.
27	String trim()	removes beginning and ending spaces of this string.

The StringBuffer Class

- .....

Homework

Write a method to sort an int array, and then compare the performance with the Arrays.sort() method.