**TYPES OF STATEMENTS.**

**SEQUENTIAL STATEMENTS**

- ➔ Linear Execution of a program or line by line execution of program.
- ➔ The flow of the program will be interrupted when there's an error or exception.

**CONTROL STATEMENTS.**

- ➔ Flow of Execution is decided based on **control expression** value
- ➔ Control expression value will always be a boolean type either **true or false.**
- ➔ If TRUE , some block will be executed . If False some other block will be executed.

- ❖ **TYPES**
  - ➢ If
  - ➢ If else
  - ➢ If else if else if …. Else
  - ➢ If if if if … else
  - ➢ nested if
  - ➢ nested if else
  - ➢ nested if else if
- ❖ Note that if is in lowercase letters. Uppercase letters (If or IF) will generate an error.
- ❖ Scope of block/code is represented using {} curly brackets

- ❖ For **if , else , else if** no need of {} if the body has only one statement.

- ❖ Formulating Control Expression we can use any operator which gives boolean value.

- ❖ Use **multiple if statements** whenever you want to execute all if block's irrespective of whether the previous if block is true or false.

- ❖ Every and every condition is checked , if true will execute the body , else check the next condition.

- ❖ **SWITCH**
  - ➢ The switch statement selects one of many code blocks to be executed.
  - ➢ switch expression must be of byte, short, int, long (with its Wrapper type), enums and string
  - ➢ case values must be unique & must be of switch expression type only.
  - ➢ The case value must be literal or constant. It doesn't allow variables & duplicates.
  - ➢ Not needed to write a **break** statement for **default,** when written as the last statement.

- ➢ **Break** saves a lot of execution time , because it "ignores" the execution of all the rest of the case in the switch block.
- ➢ If we skip the break statement , all cases will be printed after the true condition.

**LOOPING STATEMENTS**

- ➔ The Flow of the program is a process of repetitive statements or iterations to do a particular task.
- ➔ **TYPES**
- ➔ **FOR LOOP.**
  - ◆ for loop is used to iterate a part of the program several times.
  - ◆ If the number of iterations is fixed, it is recommended to use a for loop.
    - Simple for loop
    - for each
    - Labelled for loop

```
for(initialization; condition; increment/decrement){
        //statement or code to be executed
}
```

- ➔ for(;;) → will run into infinite loop
- ➔ Condition must be boolean type

- ➔ **WHILE LOOP**
  - ◆ While loop is used to iterate a part of the program several items as long as the condition is true.
  - ◆ Choose WHILE LOOP , when we don't know no. of the iterations.
  - ◆ Condition must be boolean type

```
while (boolean condition)
{
        loop statements...
        update_expression;
}
```

  - ◆ Not Incrementing/Decrementing of index variables will raise an infinite loop.

  - ◆ {} curly brackets for block of scope is not mandatory when we have only one statement/instruction inside the body of for/while/dowhile loops.

  - ◆ In Nested LOOPS , for one each iteration of OUTER LOOP , the INNER LOOP has to complete all its iterations.

➔ **DO WHILE LOOP**.

- do-while loop is an Exit control loop. Therefore, unlike for or while loop, a do-while check for the condition after executing the statements of the loop body.
- Even if the condition is false , the loop will execute at least one time.
- If you pass true in the do-while loop, it will be an infinitive do-while loop.

➔ **LABELLED LOOP.**

- It is a good practice to label a loop when using a nested loop.
- We can also use labels with continue and break statements.
- Using labelled break & continue we can directly break/continue outer loop.

➔ **LOOPING STATEMENTS**

- **break**
  - When a break is encountered inside a loop, the loop is immediately terminated , the program control resumes at the next statement outside the loop.
  - break is used to break loop or switch statement.
  - It breaks the current flow of the program at specified condition. In the case of the inner loop, it breaks only the **inner loop**.
  - We can use **break** all types of loops

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

➔ **ARRAYS**

- Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- Arrays are stored in contagious memory [consecutive memory locations].
- implements the interfaces Cloneable and java.io.Serializable.
- a direct superclass of an array type is Object.
- The size of an array must be specified by **int or short value** and not long.
- An array can contain **primitives (int, char, etc.)** and **object (or non-primitive)**

➔ **CREATION & INITIALIZING & ACCESSING**.
- ◆ array declaration has two components: the type and the name

Ex: type var-name[];
type[] var-name;

➔ When an array is declared, only a reference of an array is created.
➔ To create or give memory to the array, you create an array like this:

- ◆ var-name = new type [size];

➔ 2 Ways to create an array.
- ◆ Using new operator
- ◆ Using literals ({data1,data2,data3..})

➔ Each element in the array is accessed via its index.
➔ The index begins with 0 and ends at (total array size)-1
➔ Invalid index : AIOB Exception.
➔ When an array created with size , it will initialise with default values.
➔ **Properties**
- ◆ Declaration and initialization of array is different
- ◆ We can pass array as a parameter
- ◆ we can declare Anonymous array
- ◆ we can return an array from a function.
- ◆ Array base address: The address of the first array element is called the array base address

➔ 2D Array | Multidimensional

- ◆ data is stored in row and column based index (also known as matrix form).

**syntax**:

```
dataType[][] arrayRefVar;
dataType [][]arrayRefVar; dataType arrayRefVar[][];
dataType []arrayRefVar[];

int[][] arr=new int[3][3]
```

➔ **Jagged Array :** creating odd number of columns in a 2D array
- ◆ It is an array of arrays with different no. of columns

➔ We can copy an array to another by the arraycopy() method of System class.

➔ We can create a clone of the Java array.
- ◆ Cloning a single-dimensional array, creates the deep copy. It means, it will copy the actual value.

- Cloning  of a multidimensional array, it creates a **shallow copy** of the Java array which means it copies the references.