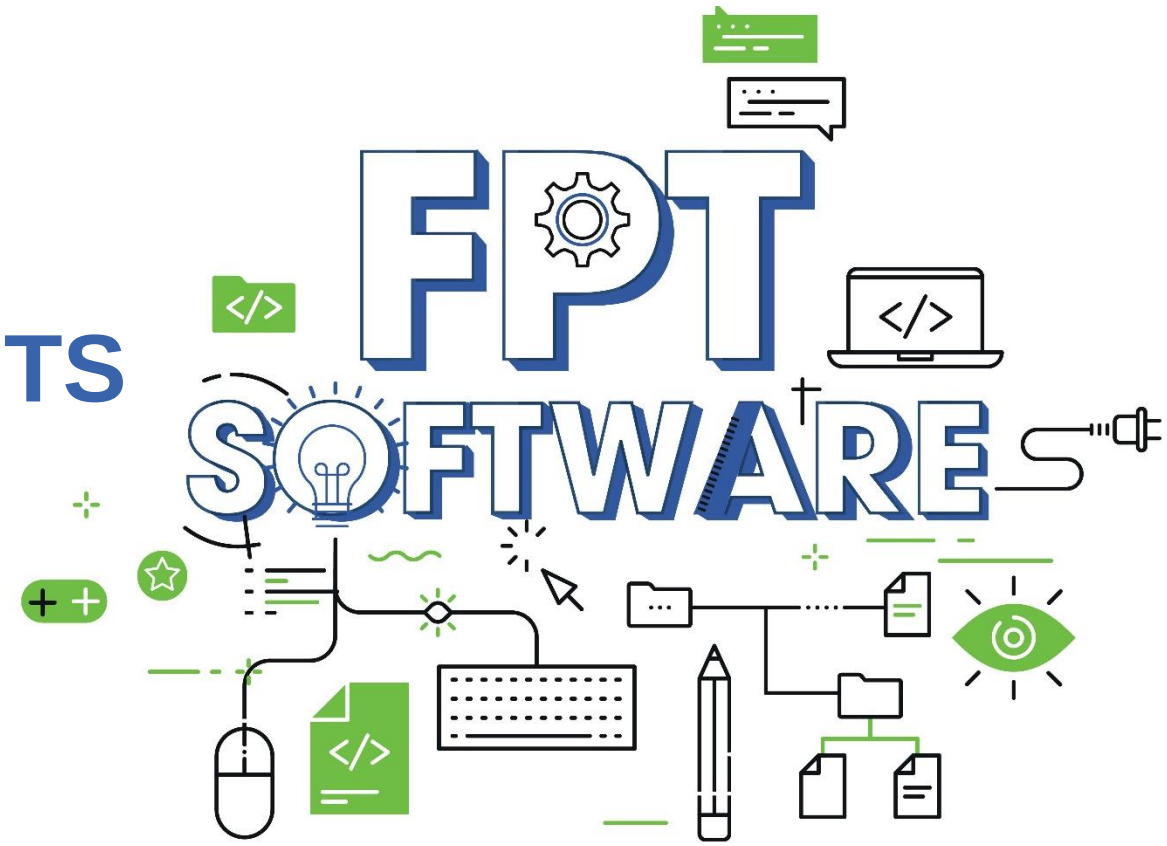
FLOW CONTROL STATEMENTS



Instructor: DieuNT1

Agenda

01. Arrays

- Single Dimensional Array
- Two Dimensional Array

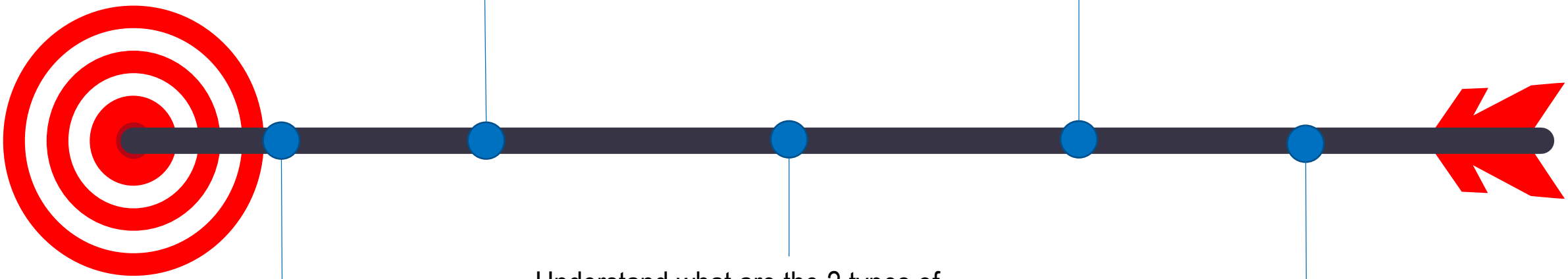
02. Flow Control Statements

- if..else
- switch-case
- While
- do..while
- for
- break,continue, return

Lesson Objectives

Understand the **difference between Single Dimensional Array and Two Dimensional Array**

Knowing what are the 3 types of **loops in Java**?



Understand what are the 2 types of **decision making statements in Java**?

Knowing what is **dimensional array in Java?**

Knowing what are the 3 **branching statements in Java**?

Section 1

Arrays

Java Arrays

Java array is an object which contains elements of a similar data type.

The elements of an array are stored in a contiguous memory location.

For example, you can create an array that can hold 100 values of int type.

It is a data structure where we store similar elements:

- We can store only a fixed set of elements in a Java array.
- Array in Java is index-based, the first element of the array is stored at the 0^{th} index, 2^{nd} element is stored on 1^{st} index and so on.
- We can store primitive values or objects in an array in Java
- Like C/C++, we can also create single dimentional or multidimentional arrays in Java.

Java Arrays

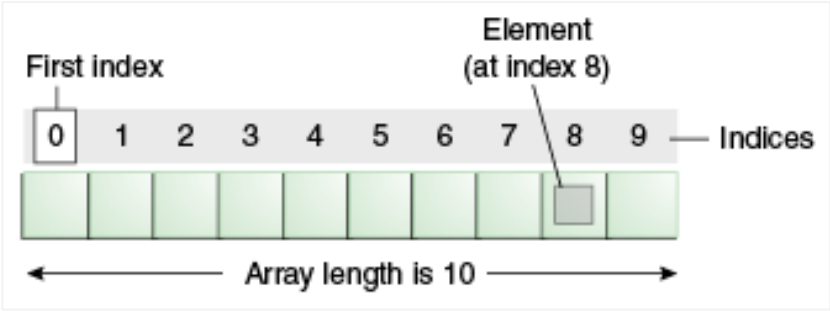
- **Arrays:**

Data structures

Related data items of **same type**

Remain same size once created

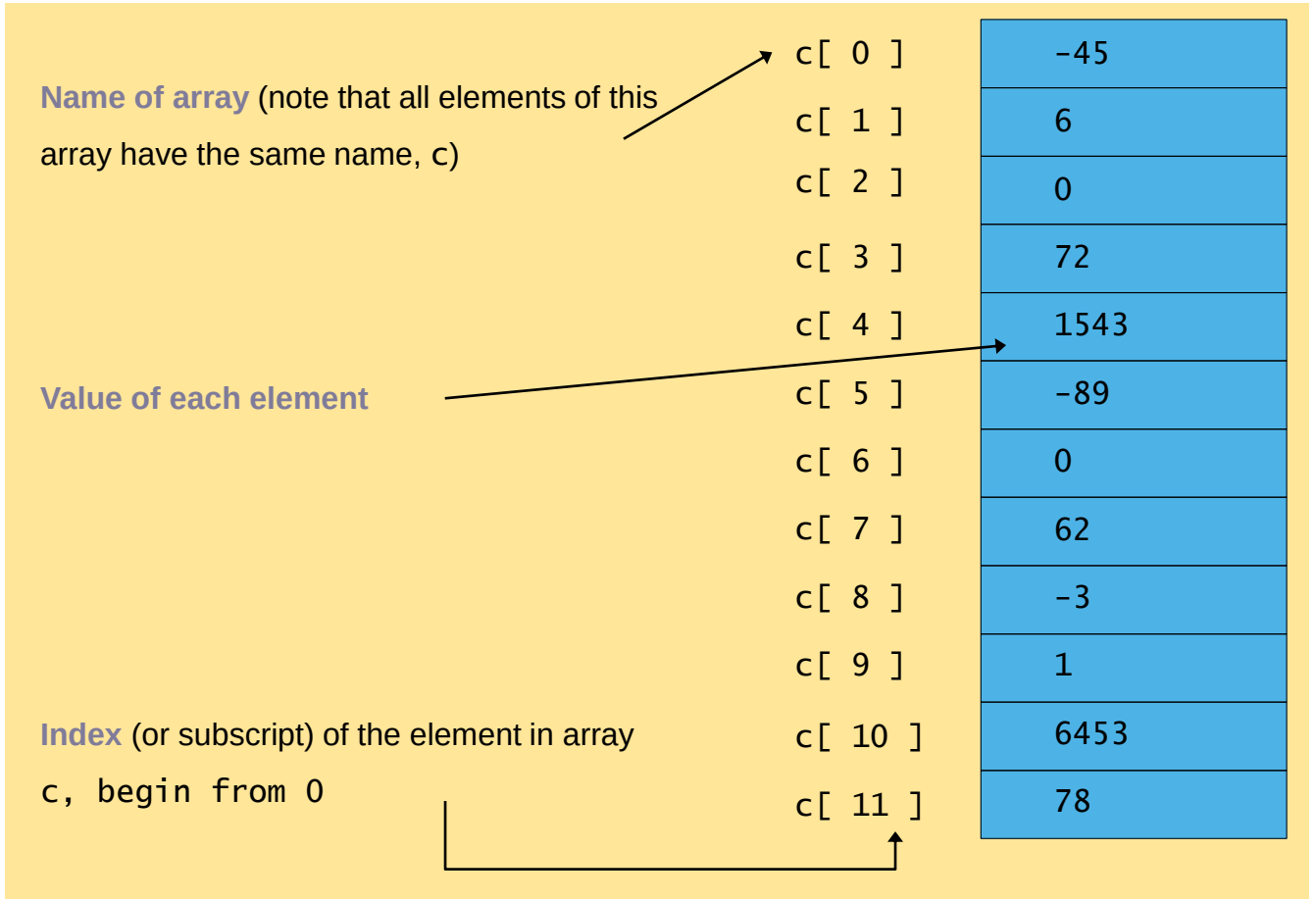
- *Fixed-length* entries



Types of Array in Java

- **There are two types of array:**
  - ✓ Single Dimensional Array
  - ✓ Multidimensional Array

- **Single Dimensional Array Structure**:



Single Dimensional Array in Java

- **Syntax:** Three ways to declare an array are

```
datatype[] identifier;
datatype[] identifier = new datatype[size];
datatype[] identifier = {value1,value2,…valueN};
```

- You can also place the square brackets after the array's name:

```
datatype identifier[];//this form is discouraged
```

- **Example:**

```
byte[] bArray;
       float[] fArray = new float[20];
       int[] iArray = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
```

Array Declarations

- Examine array **bArray, fArray, iArray:**
  - ✓**bArray, fArray, iArray** is the array *name*
  - ✓**fArray.**length accesses array c's *length*
  - ✓**iArray** has 10 *elements*:

    iArray[0], **iArray** [1], … , **iArray** [9]

    - The *value* of **iArray** [0] is 32

Array Index

- Also called subscript
- Position number in square brackets
- Always begin from zero
  - ✓ Must >= 0 and < array's length
- **Example:**

```
fArray[0] = 12.5f;
        for (int counter = 0; counter <
iArray.length; counter++){
                output += counter + "\t" +
iArray[counter] + "\n";
        }
```

Passing Array to a Method in Java

- We can pass the Java array to method so that we can reuse the same logic on any array.

```
public class TestArray {

public static void main(String[] args) {
    int[] intArray = { 5, 22, 16, 8, 89, 6 };

    System.out.println("Max of value:" + findMax(intArray));

  }

  static int findMax(int[] intArray) {
    int max = intArray[0];

    for (int i = 1; i < intArray.length; i++) {
      intArray[i] *= 2;
      if (intArray[i] > max) {
        max = intArray[i];
      }
    }

    return max;
  }
}
```

ArrayIndexOutOfBoundsException

- The Java Virtual Machine (JVM) throws an **ArrayIndexOutOfBoundsException** if length of the array in negative, equal to the array size or greater than the array size while traversing the array.

```
public class TestArrayException {

  public static void main(String[] args) {
    int arr[] = { 50, 60, 70, 80 };
    for (int i = 0; i <= arr.length; i++) {
      System.out.println(arr[i]);
    }
  }
}
```

- **Output**:

```
50
60
70
80
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
at fa.training.jpe.TestArrayException.main(TestArrayException.java:15)
```

Multidimensional Arrays

- In such case, data is stored in **row** and **column** based index (also known as matrix form).
- **Syntax:**

```
1.dataType[][] arrayRefVar; (or)
2.dataType [][]arrayRefVar; (or)
3.dataType arrayRefVar[][]; (or)
4.dataType []arrayRefVar[];
```

- **Example:**

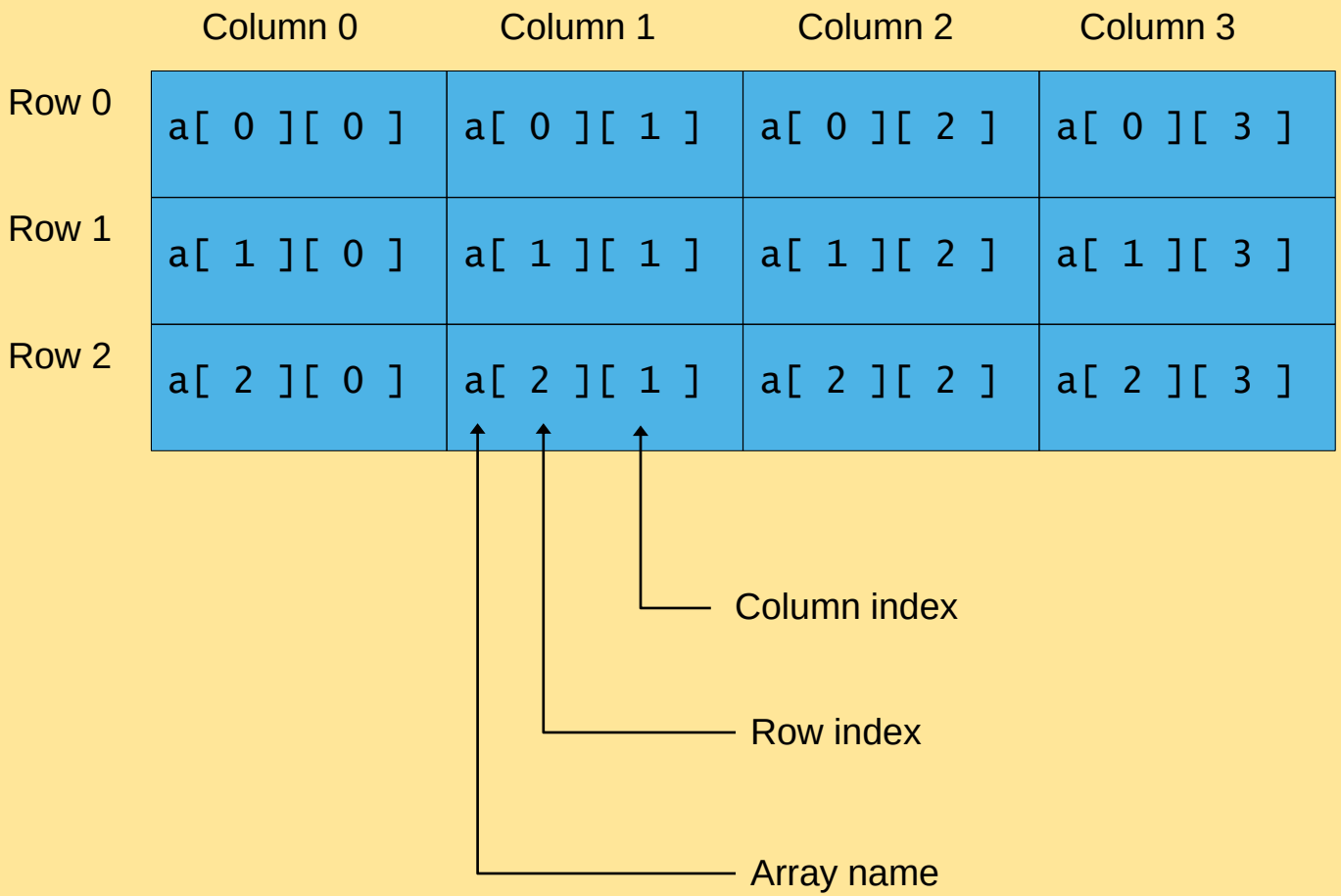
```
int[][] arr=new int[3][3];//3 row and 3 column
```

- **Initialize Multidimensional Array in Java**

```
1.arr[0][0]=1;
2.arr[0][1]=2;
3.arr[0][2]=3;
4.arr[1][0]=4;
5.arr[1][1]=5;
6.arr[1][2]=6;
7.arr[2][0]=7;
8.arr[2][1]=8;
9.arr[2][2]=9;
```

Multidimensional Arrays

- **Two-dimensional array structure**



Multidimensional Arrays

- **Example**: Let's see the simple example to *declare*, *instantiate*, *initialize* and *print* the 2Dimensional array.

```
public class Test2Dimensional {
      public static void main(String[] args) {
          // Declaring and initializing 2D array
          int arr[][] = { { 1, 2, 3 }, { 2, 4, 5 }, { 4, 4, 5 } };

          // Printing 2D array
          for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
              System.out.print(arr[i][j] + "\t");
            }
            System.out.println();
          }
        }
}
```

- **Output**

```
1	2	3
2	4	5
4	4	5
```

Jagged Array in Java

Jagged Array is an array of arrays with different number of columns.

Example:

```
public class TestJaggedArray {
      public static void main(String[] args) {
          // Declaring a jagged array
          int[][] jagArray = new int[3][];

          jagArray[0] = new int[3];
          jagArray[1] = new int[5];
          jagArray[2] = new int[2];

          Random random = new Random(2);

          // Initializing a jagged array
          for (int i = 0; i < jagArray.length; i++) {
            for (int j = 0; j < jagArray[i].length; j++) {
              jagArray[i][j] = random.nextInt(100);
            }
          }

          // Printing the data of a jagged array
          for (int i = 0; i < jagArray.length; i++) {
            for (int j = 0; j < jagArray[i].length; j++) {
              System.out.print(jagArray[i][j] + "\t");
            }
            System.out.println();
          }
        }
}
```

- **Output**

```
8       72      40
67      89      50      6       19
47      68
```

Copying a Java Array

- We can copy an array to another by the **arraycopy()** method of **System** class.
- **Syntax:**

```
public static void arraycopy(Object src,  int srcPos,Object dest,

                      int destPos, int length)
```

- **Example:**

```
public class TestArrayCopyDemo {
      public static void main(String[] args) {
          // Declaring a source array
          char[] copyFrom = { 'F', 'P', 'T', 'S', 'o', 'f', 't', 'w', 'a', 'r', 'e',
              'A', 'c', 'a', 'd', 'e', 'm', 'y' };

          // Declaring a destination array
          char[] copyTo = new char[10];

          // Copying array using System.arraycopy() method
          System.arraycopy(copyFrom, 3, copyTo, 0, 8);

          // Printing the destination array
          System.out.println(String.valueOf(copyTo));
        }
}
```

- **Output:**

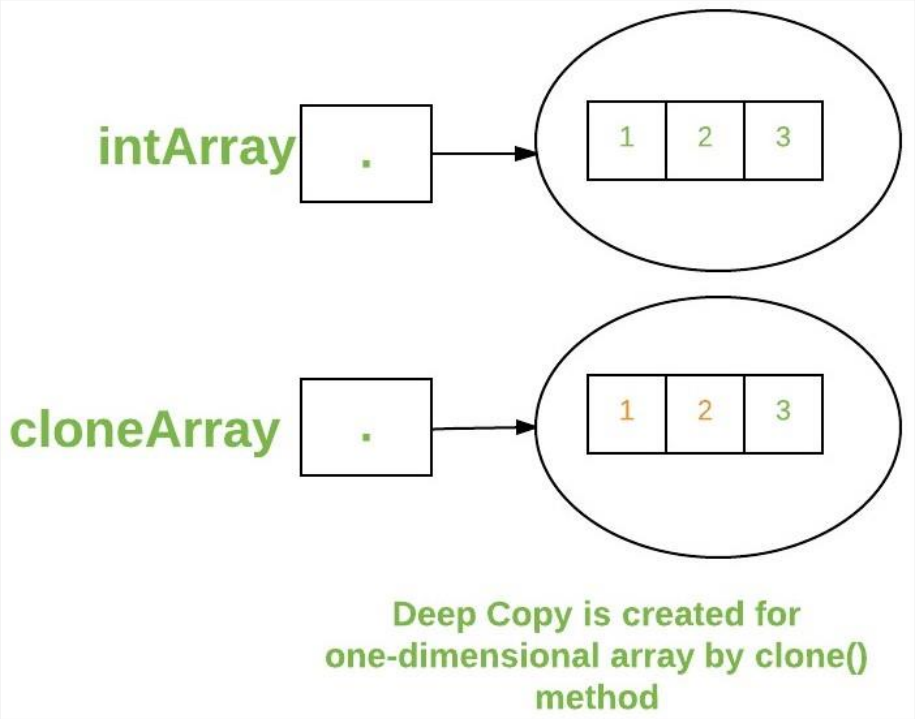
```
Software
```

Cloning an Array in Java

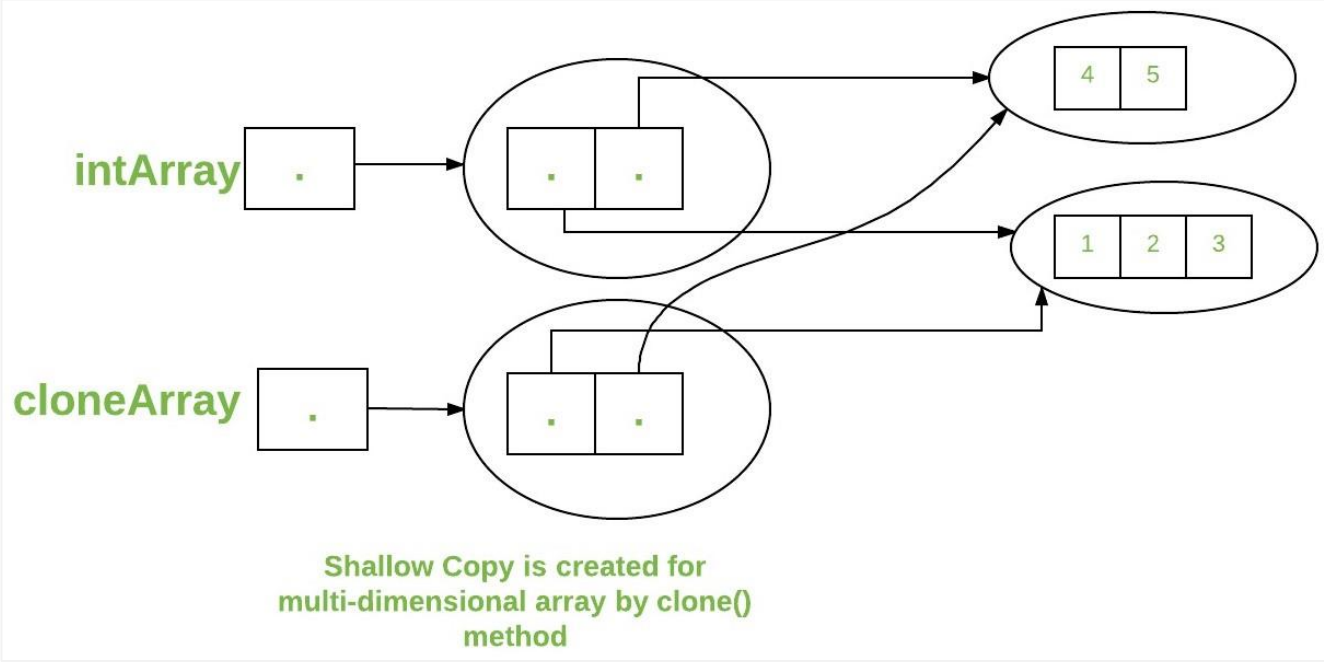
- We can clone an array by the **clone**() method.
- **Example**:

```
public class TestCloneArray {
     public static void main(String[] args) {
         int arr[] = { 12, 5, 18, 8, 6 };
         System.out.println("Printing original array:");
         for (int value : arr)
           System.out.println(value);

         System.out.println("Printing clone of the array:");
         int carr[] = arr.clone();
         for (int value : carr)
           System.out.println(value);

         System.out.println("Are both equal?");
         System.out.println(arr == carr);
       }
}
```

Cloning an Array in Java



Deep Copy is created for one-dimensional array by clone() method



Shallow Copy is created for multi-dimensional array by clone() method

Section 2

Flow Control Statements

Flow Control Statements

Decision-making

- if-else statement
- switch-case statement

Loops

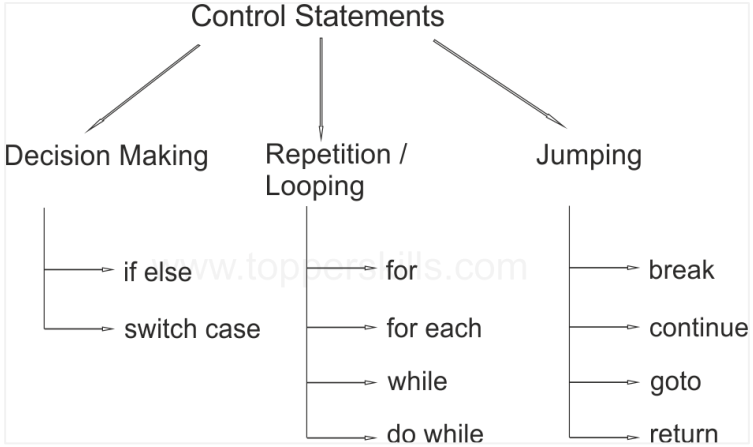
- while loop
- do-while loop
- for loop

Branching

- break
- continue
- return

Flow Control Statements

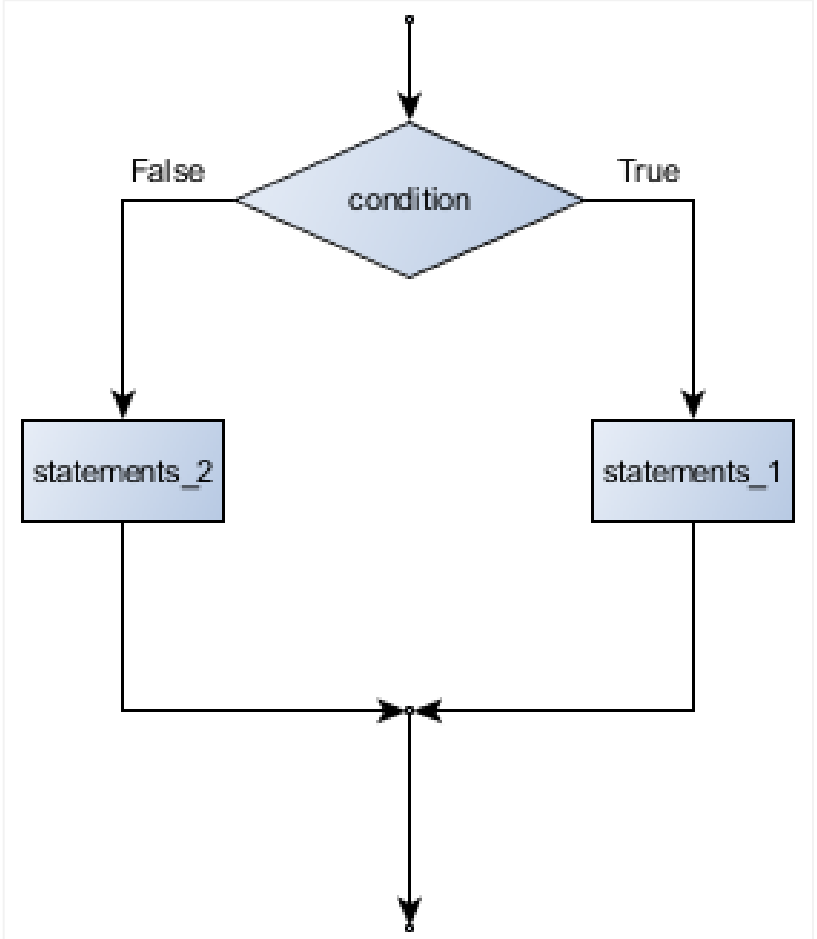
- All application development environments provide a decision making process called flow control statements that direct the application execution.
- Flow control enables a developer to create an application that can examine[kiểm tra] the existing conditions, and decide a suitable course of action.
- Loops or iteration are an important programming construct that can be used to repeatedly execute a set of actions.
- Jump statements allow the program to execute in a non-linear fashion.



if-else statement

- **Syntax:**

```
if (condition){
        action1;
} else {
        action2;
}
```



- **Note:**
  - ✓"**else**" is optional
  - ✓Alternative way to if-else is conditional operator (?:)

if-else statement

- **Example:**

```
public class CheckNum {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int num = 10;
        if (num % 2 == 0) {
        System.out.println(num + " is an even
        number");
        } else {
        System.out.println(num + " is an odd
        number");
        }
    }
}
```

switch – case statement

Unlike if-then and if-then-else statements, the switch statement can have a number of possible execution paths.

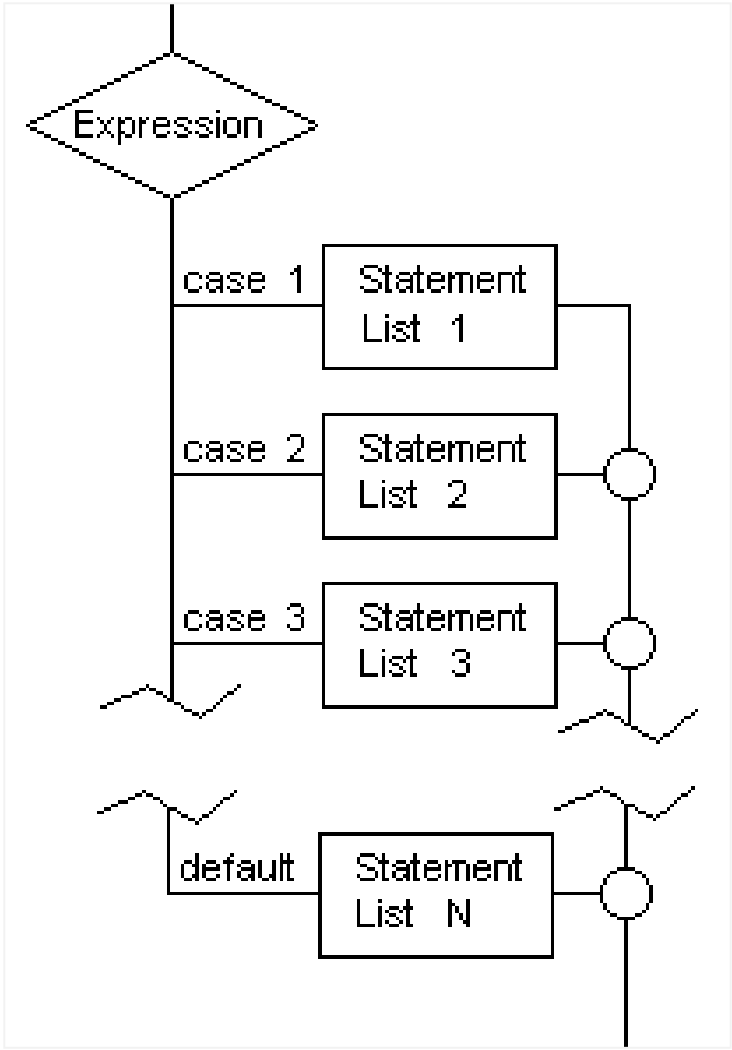
A switch works with the byte, short, char, and int primitive data types.

It also works with enumerated types, the String class, and a few special classes that wrap certain primitive types: Character, Byte, Short, and Integer (discussed in Numbers and Strings).

switch – case statement

- Syntax:

```
    case value_1:
       statement_1; [ break;]
    case value_2:
       statement_2; [ break;]
    …
    case value_n:
       statement_n; [ break;]
    default:
       statement_n+1; [break;]
}
```



switch – case statement

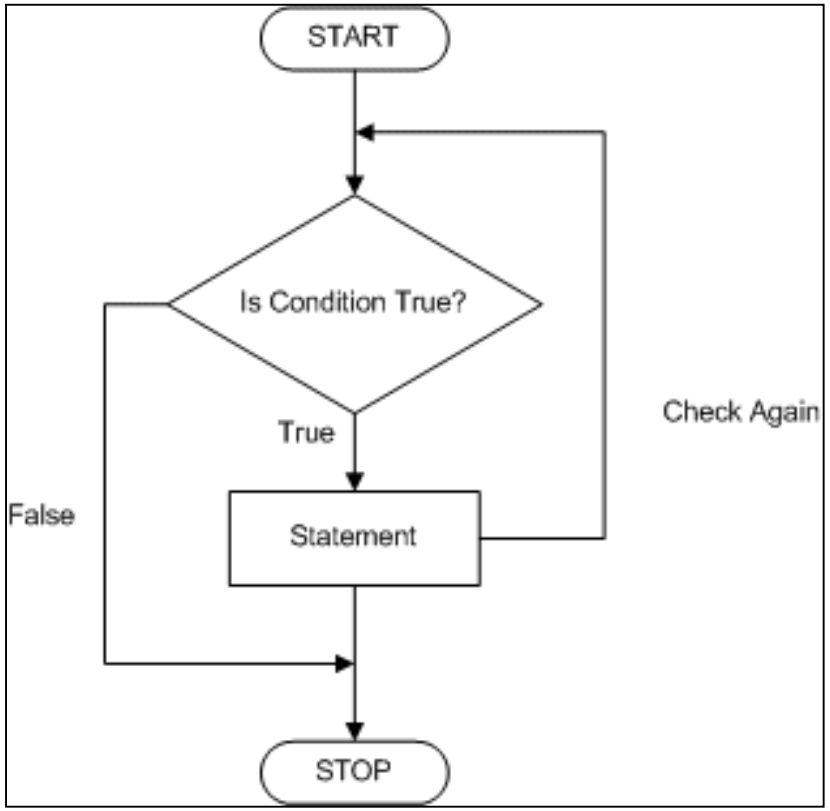
```
public class SwitchDemo2 {
    public static void main(String[] args) {

        int month = 2;
        int year = 2000;
        int numDays = 0;

        switch (month) {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                numDays = 31;
                break;
```

```
            case 4:
            case 6:
            case 9:
            case 11:
                numDays = 30;
                break;
            case 2:
                if ( ((year % 4 == 0) && !(year % 100 == 0))
                     || (year % 400 == 0) )
                    numDays = 29;
                else
                    numDays = 28;
                break;
        }
        System.out.println("Number of Days = " + numDays);
    }
}
```

while Loop

- `while` loops are used for situations when a loop has to be executed as long as certain condition is True.
- The number of times a loop is to be executed is not pre-determined, but depends on the condition.
- **The syntax is:**

```
while (condition) {
    action statements;
}
```



while Loop

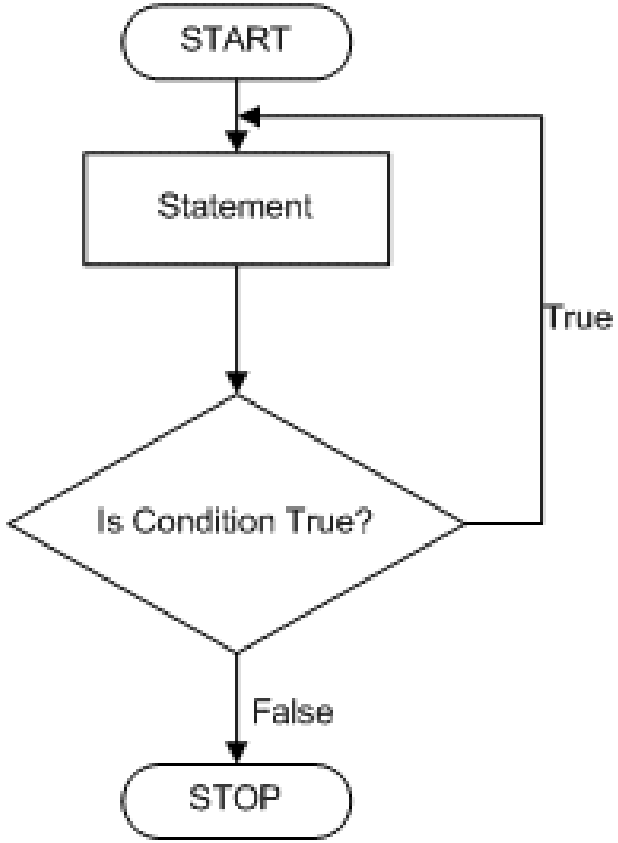
- **Example:**

```
public class FactDemo {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int num = 5, fact = 1;
        while (num >= 1) {
        fact *= num;// fact = fact * num;
        num--;
        }
        System.out.println("The factorial of 5 is : " +
                                        fact);
    }
}
```

do – while Loop

- The `do-while` loop executes certain statements till the specified condition is True.
- These loops are similar to the `while` loops, except that a `do-while` loop executes at least once, even if the specified condition is False.
- **The syntax is:**

```
do {
    action statements;
} while (condition);
```



do – while Loop

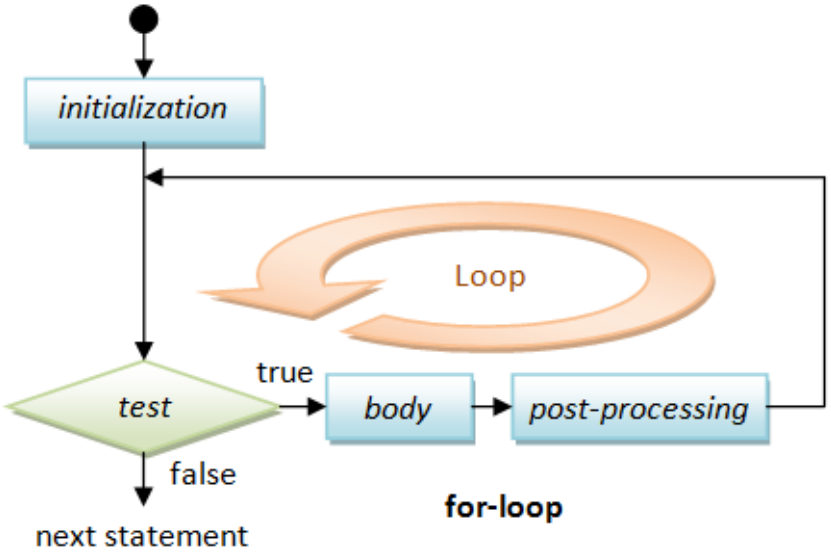
- **Example:**

```
public class DoWhileDemo {
    public static void main(String[] args) {
        int count = 1, sum = 0;
        do {
        sum += count;
        count++;
        } while (count <= 100);
        System.out.println("The sum of first 100 numbers is
                                : " + sum);
    }
}
```

for Loop

- All loops have some common features: a counter variable that is initialized before the loop begins, a condition that tests the counter variable and a statement that modifies the value of the counter variable.
- The `for` loop provides a compact format for incorporating these features.
- Syntax:

```
for (initialization;loopContinuationCondition; increment)
{
statement;
}
```



for Loop

- Example:

```
public class ForDemo {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int count = 1, sum = 0;
        for (count = 1; count <= 10; count += 2) {
        sum += count;
        }
        System.out.println("The sum of first 5 odd numbers is : " + sum);
    }
}
```

Break Statements

- The break statement has two forms: labeled and unlabeled.
- Use unlabeled break to terminate a switch, for, while, or do-while loop
- Use labeled break to terminates an outer statement
- Example:

```
public class BreakDemo {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        for (int count = 1; count <= 100; count++) {
        if (count == 10) {
        break;
        }
        System.out.println("The value of num is : " + count);
        }
        System.out.println("The loop is over");
    }
}
```

Continue statement

- The continue statement skips the current iteration of a for, while , or do-while loop.
- The unlabeled form skips to the end of the innermost loop's body and evaluates the boolean expression that controls the loop.
- The labeled continue statement skips the current iteration of an outer loop marked with the given label.

Continue statement

- **Example:**

```
public class ContinueDemo {
    public static void main(String[] args) {
        String searchMe = "peter piper picked a peck of pickled peppers";
        int max = searchMe.length();
        int numPs = 0;
        for (int i = 0; i < max; i++) {
        // interested only in p's
            if (searchMe.charAt(i) != 'p') {
                    continue;
            }
            numPs++;
        }
        System.out.println("Found " + numPs + " p's in the string.");
    }
}
```

Return statement

- The return statement exits from the current method, and control flow returns to where the method was invoked.
- The return statement has **two forms**:
  - ✓Returns a value: `return ++count;`
  - ✓Doesn't returns a value (void): `return;`
- The data type of the returned value must match the type of the method's declared return value.

Practice time

- **Exercise 1**: Program to find the frequency of each element in the array.
- Excersie 2: Program to left rotate the elements of an array.

(Nếu có đủ thời gian thì cho thực hành/demo tại lớp, nếu không thì học viên có thể được hướng dẫn để làm thêm ở nhà).

SUMMARY

01. Arrays

- Single Dimensional Array
- Two Dimensional Array

02. Flow Control Statements

- if..else
- switch-case
- While
- do..while
- for
- break,continue, return

Questions

THANK YOU!