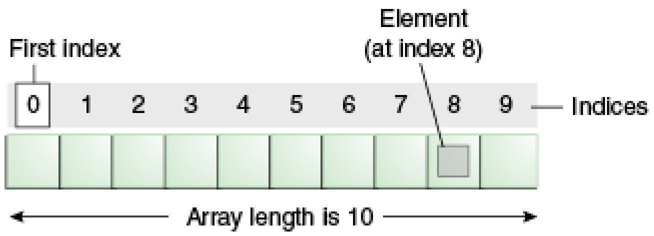
FLOW CONTROL STATEMENTS

Instructor: DieuNT1

Table of contents

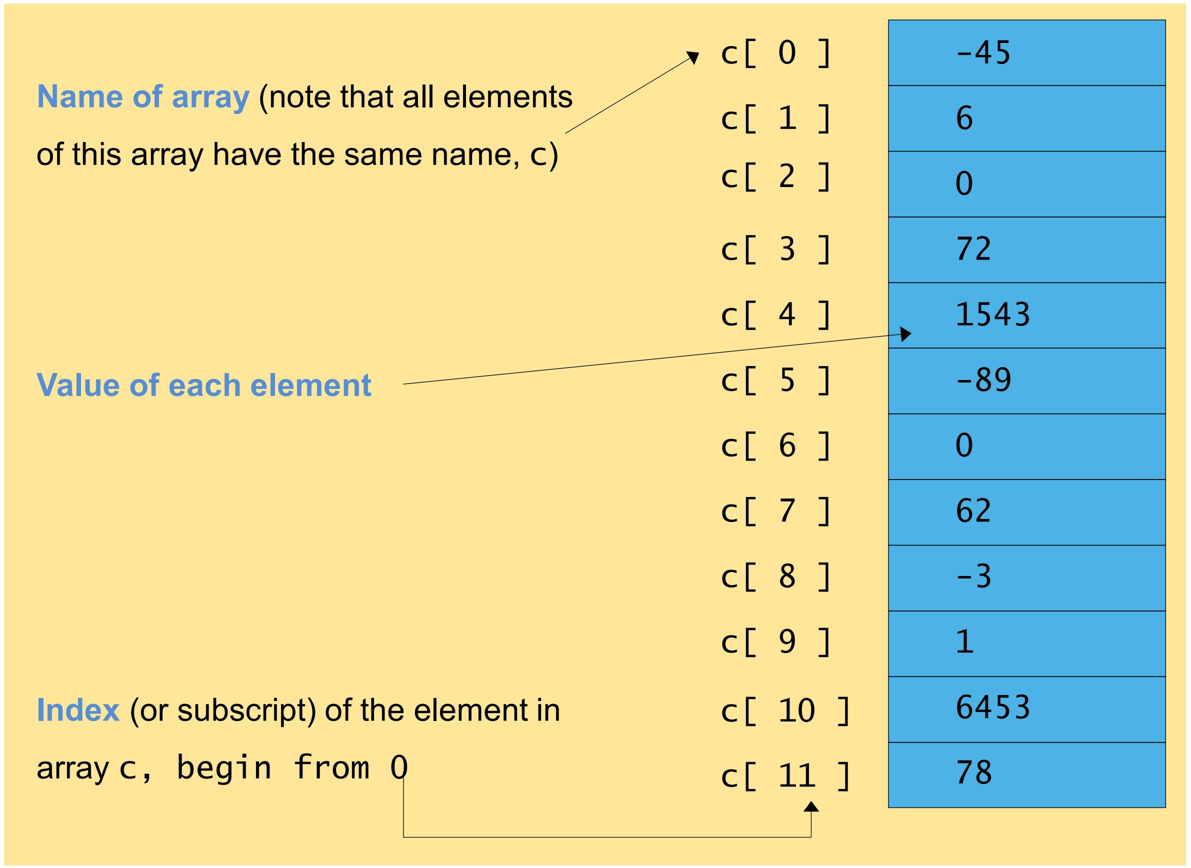
Java Arrays

- **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 0th index, 2nd element is stored on 1st 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.
- **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**:



Name of array (note that all elements of this array have the same name, c)
Value of each element
Index (or subscript) of the element in array c, begin from 0
c[0] -45
c[1] 6
c[2] 0
c[3] 72
c[4] 1543
c[5] -89
c[6] 0
c[7] 62
c[8] -3
c[9] 1
c[10] 6453
c[11] 78

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:**

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

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)
```

Section 2

Flow Control Statements

Flow Control Statements

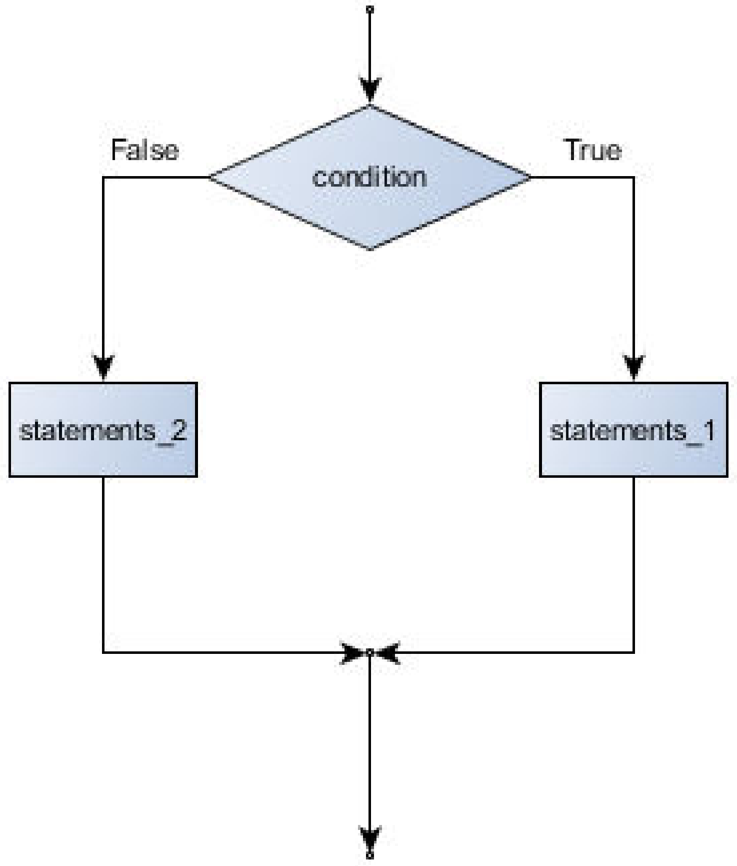
- **Decision-making**
  - ✓ if-else statement
  - ✓ switch-case statement
- **Loops**
  - ✓ while loop
  - ✓ do-while loop
  - ✓ for loop
- **Branching**
  - ✓ break
  - ✓ continue
  - ✓ return

if-else statement

- **Syntax:**

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

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



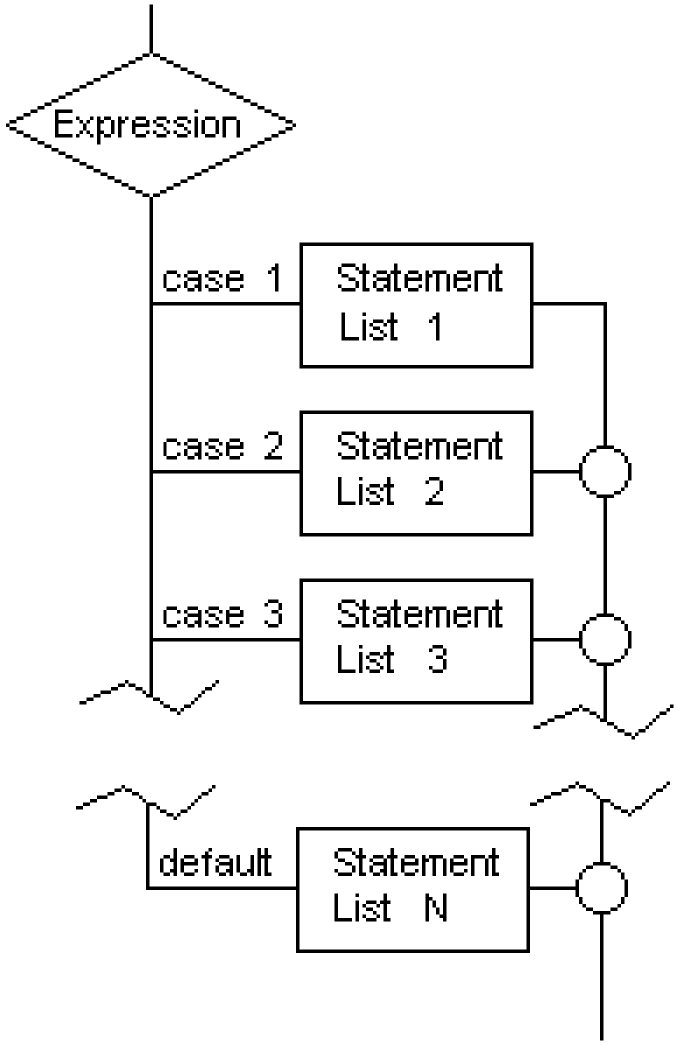
switch – case statement

- 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:**

```
switch(expression) {
    case value_1:
        statement_1; [ break;]
    case value_2:
        statement_2; [ break;]
    …
    case value_n:
        statement_n; [ break;]
    default:
        statement_n+1; [break;]
}
```



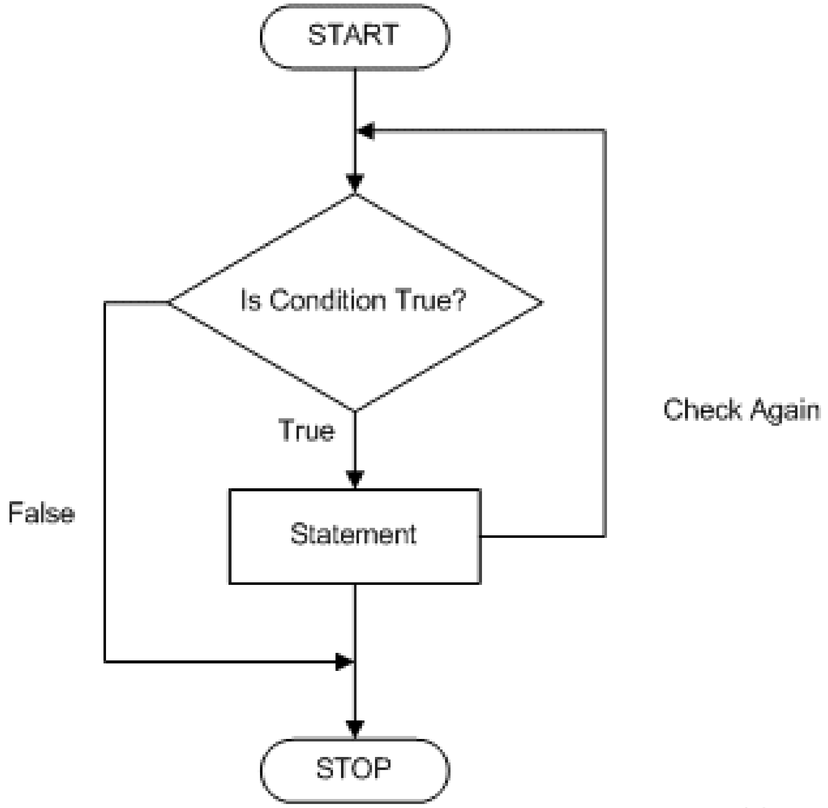
switch – case statement

```
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

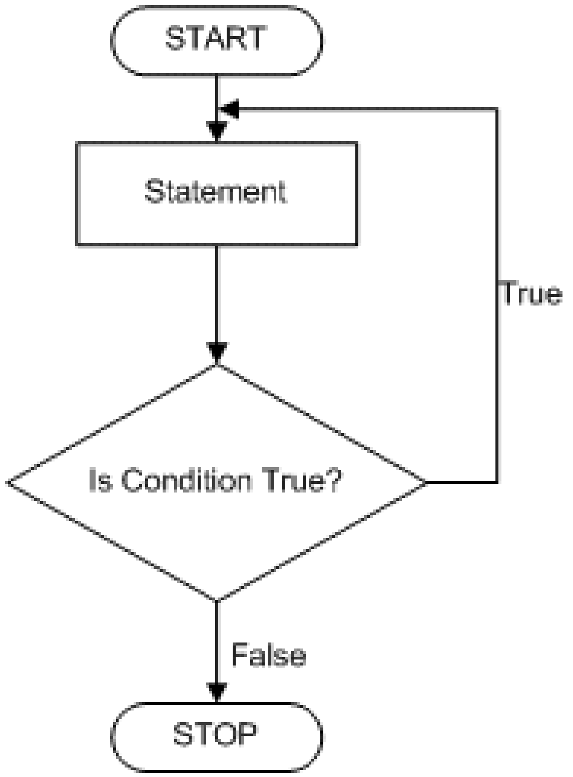
- **Example:**

```
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

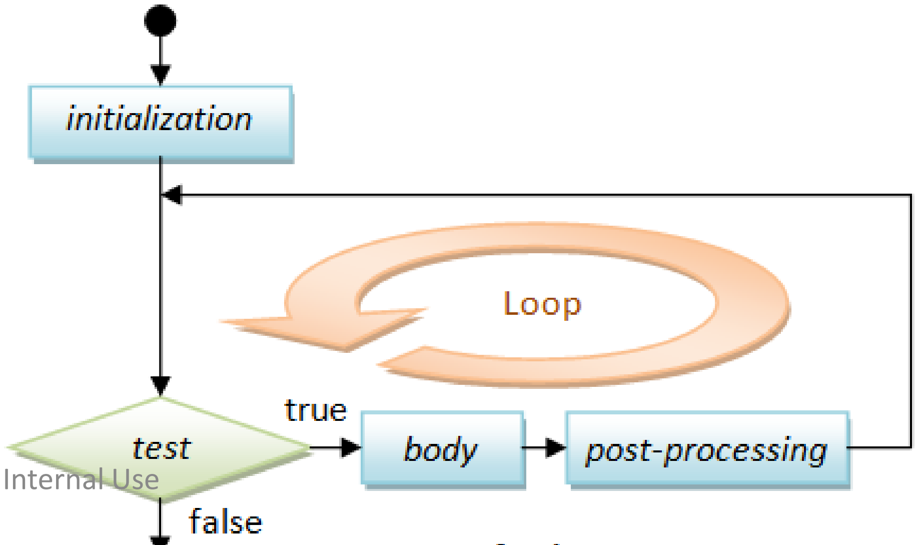
- **Example:**

```
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;
}
```



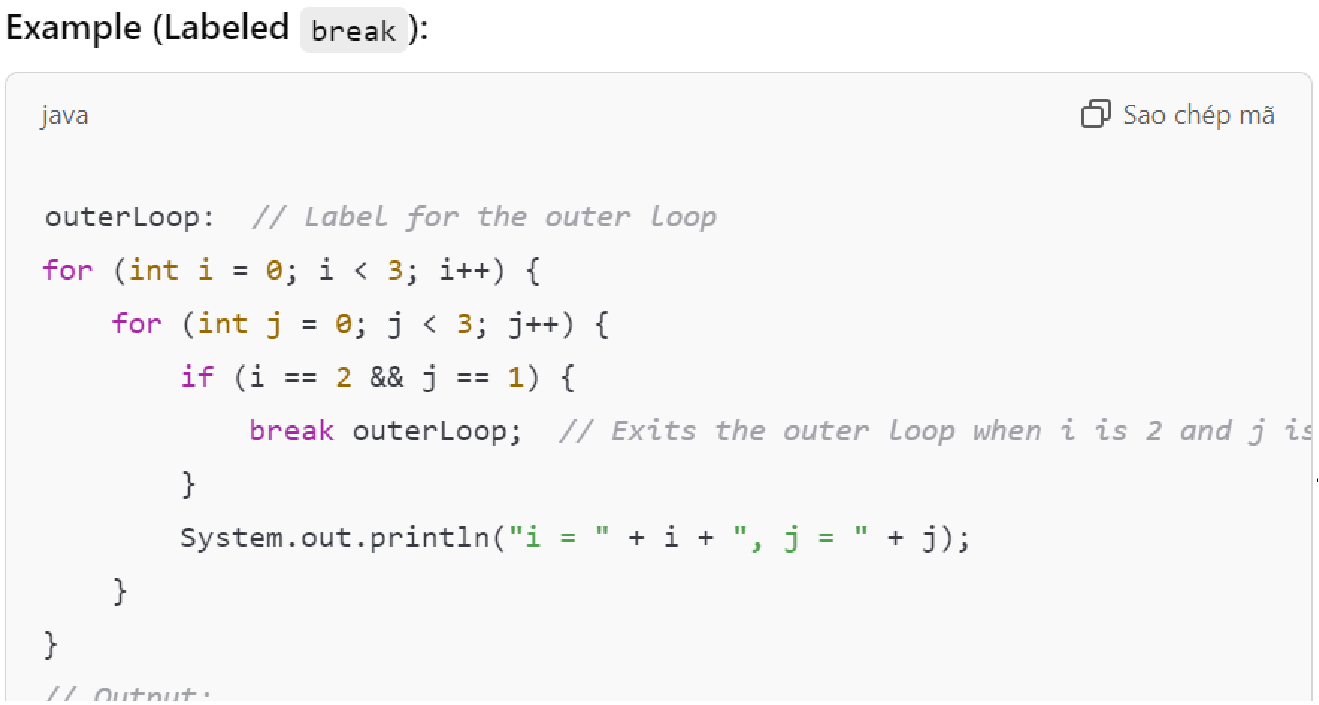
for Loop

- 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:



Example (Labeled `break`):

```java
outerLoop:  // Label for the outer loop
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        if (i == 2 && j == 1) {
            break outerLoop;  // Exits the outer loop when i is 2 and j is
        }
        System.out.println("i = " + i + ", j = " + j);
    }
}
// Output:
```

```
count);
}
```

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 cũng gồm labeled và unlabeled giống như break

Continue statement

- **Example:**

```java
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.

SUMMARY

- ◊ **Arrays**
  - ◊ Single Dimensional Array
  - ◊ Two Dimensional Array
- ◊ **Flow Control Statements**
  - ◊ if..else
  - ◊ switch-case
  - ◊ while
  - ◊ do..while
  - ◊ for
  - ◊ break,continue, return

Thank you