Java Programs

Lecture 3

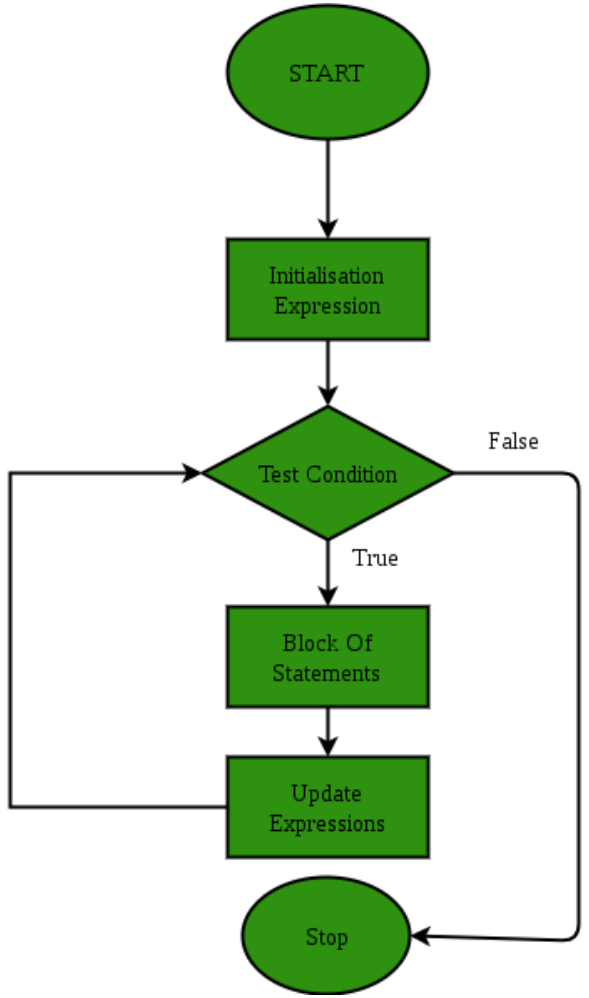
By
Dr. Radwa Rady & Dr. Ghada Fathy

Objectives

- Loops Statements:
  - ✓ While Loop
  - ✓ Do While Loop
  - ✓ For Loop
  - ✓ Nested Loops
- Break and Continue
- Arrays:
  - ✓ Types.
  - ✓ Access Elements.
  - ✓ Edit Elements.

Loops Statements

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement .

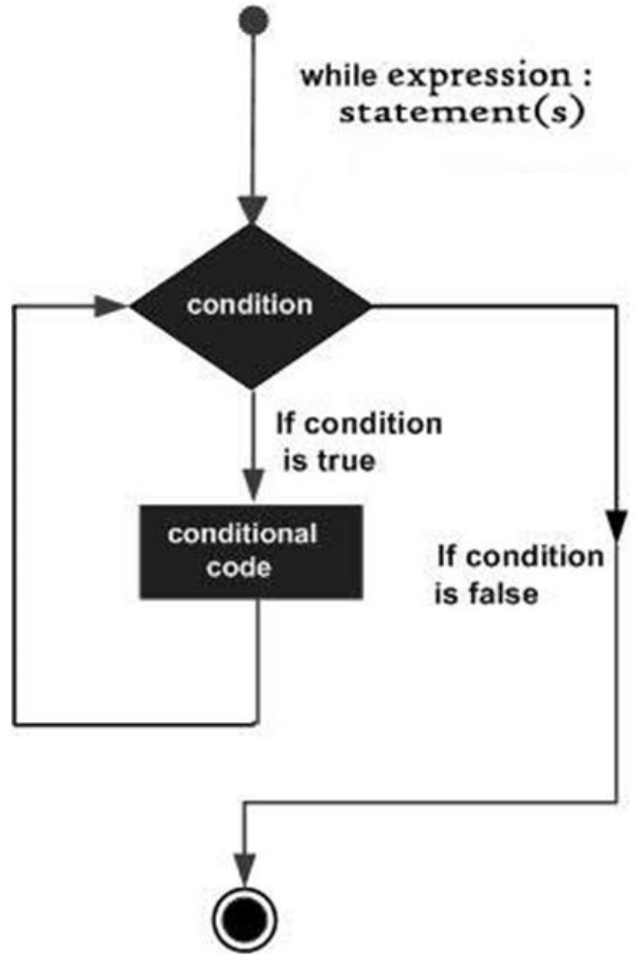


- Loops can execute a block of code as long as a specified condition is reached.
- Loops are handy because they save time, reduce errors, and they make code more readable.
- Example: Print numbers from 0 to 10.

While Loop

```
while (condition)
{
  // code block to be executed
}
```

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

        int i = 0;
        while (i < 5)
        {
            System.out.println(i);
            i++;
        }
    }
}
```



Do / While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

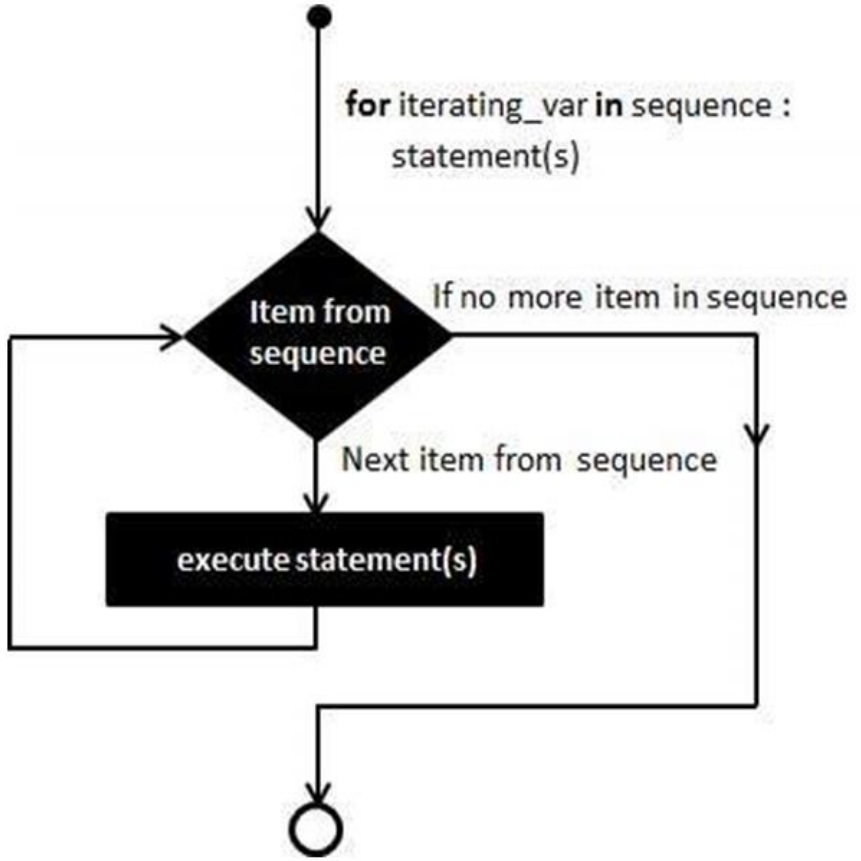
```
do
{
  // code block to be executed
}
while (condition);
```

```
public class Test {
    public static void main (String[] args) {
      int i = 0;
      do {
        System.out.println(i);
        i++;
      }
      while (i < 5);
    }
}
```

For Loop

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop.

```
for (statement 1; statement 2; statement 3)
{
  // code block to be executed
}
```



- **Statement 1** is executed (one time) before the execution of the code block.
- **Statement 2** defines the condition for executing the code block.
- **Statement 3** is executed (every time) after the code block has been execute

Example 1

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

        for (int i = 0; i < 5; i++)
        {
            System.out.println(i);
        }

    }
}
```

Comparison Between Loops

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

      for (int i = 0; i < 5; i++) {
        System.out.println(i);
      }

    }
}
```

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

      int i = 0;
      while (i < 5) {
        System.out.println(i);
        i++;
      }
    }
}
```

```java
public class Test {
    public static void main (String[] args) {
      int i = 0;
      do {
        System.out.println(i);
        i++;
      }
      while (i < 5);
    }
}
```

Example 2

This example will only print **even** values between **0** and **10**.

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

      for (int i = 0; i <= 10; i = i + 2) {
        System.out.println(i);
      }

    }
}
```

Nested Loops

It is also possible to place a loop inside another loop. This is called a nested loop. The "inner loop" will be executed one time for each iteration of the "outer loop".

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

      // Outer loop
      for (int i = 1; i <= 2; i++) {
        System.out.println("Outer: " + i); // Executes 2 times

        // Inner loop
        for (int j = 1; j <= 3; j++) {
          System.out.println(" Inner: " + j); // Executes 6 times (2 * 3)
        }
      }

    }
}
```

Outer: 1

Inner: 1

Inner: 2

Inner: 3

Outer: 2

Inner: 1

Inner: 2

Inner: 3

Break and Continue

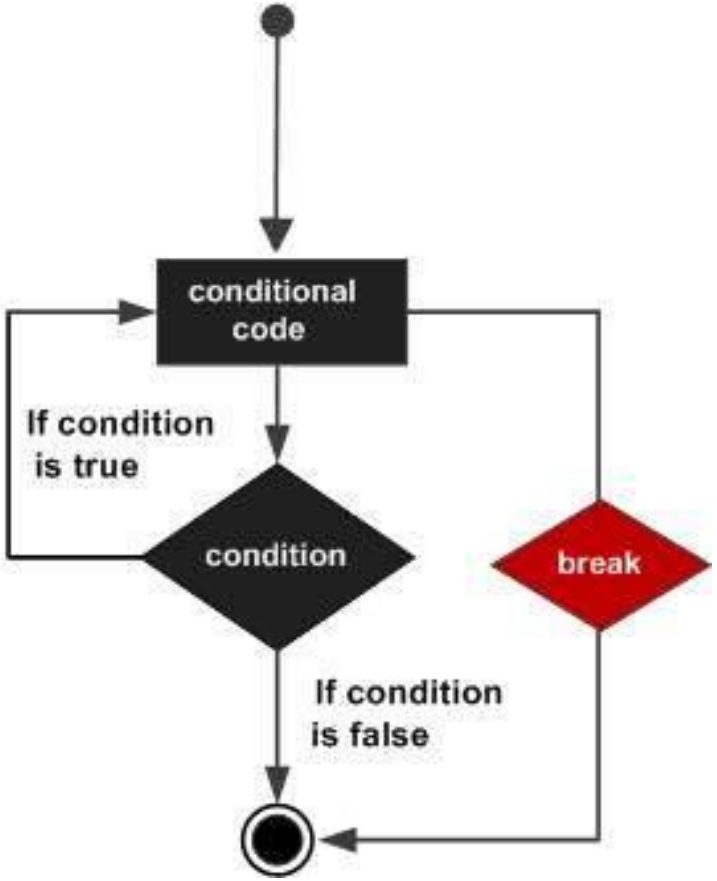
- The break statement can also be used to jump out of a loop.

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

- You can also use break and continue in while loops.

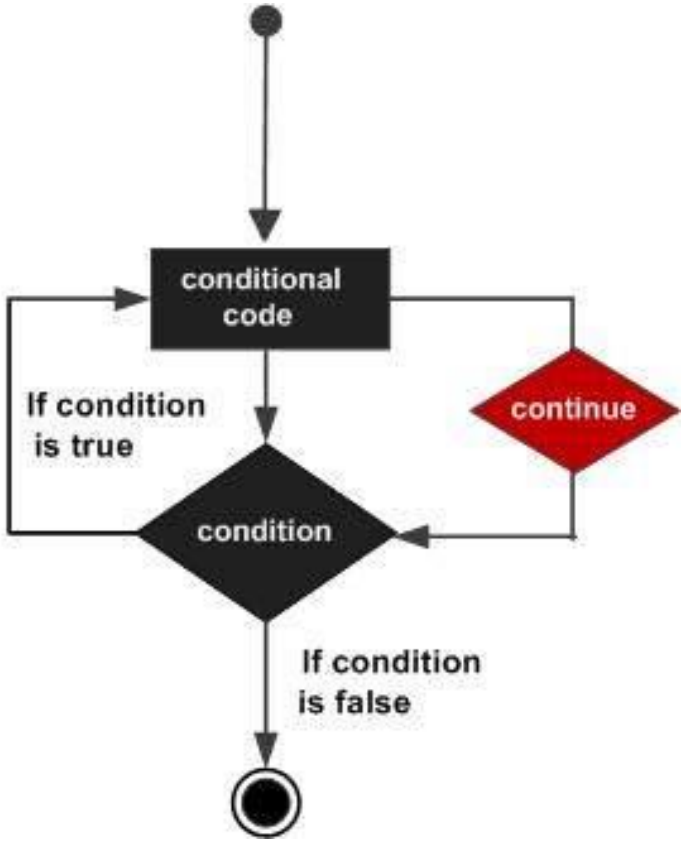
Break

- The break statement can also be used to jump out of a loop.



Continue

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



Example 1

This example stops the loop when i is equal to 4.

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

      for (int i = 0; i < 10; i++) {
        if (i == 4) {
          break;
        }
        System.out.println(i);
      }

    }
}
```

```
// Output
0
1
2
3
```

Example 2

This example skips the value of 4.

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

      for (int i = 0; i < 10; i++) {
        if (i == 4) {
          continue;
        }
        System.out.println(i);
      }

    }
}
```

```
// Output
0
1
2
3
5
6
7
8
9
```

Example 3

Break and Continue in While Loop.

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

      int i = 0;
      while (i < 10) {
        System.out.println(i);
        i++;
        if (i == 4) {
          break;
        }
      }

    }
}
```

```
// Output
0
1
2
3
```

Example 4

Break and Continue in While Loop.

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

      int i = 0;
      while (i < 10) {
        if (i == 4) {
          i++;
          continue;
        }
        System.out.println(i);
        i++;
      }

    }
}
```

```
// Output
1
2
3
5
6
7
8
9
```

Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with square brackets.

```
String[] cars;
```

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

```
int[] myNum = {10, 20, 30, 40};
```

Access Elements

You can access an array element by referring to the index number.

- **Note**: Array indexes start with **0**: [0] is the first element. [1] is the second element, etc.

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

      String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

      System.out.println(cars[0]);

    }
}
```

```
// Outputs
Volvo
```

Edit Elements

To change the value of a specific element, refer to the index number: `cars[0] = "Opel";`

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

      String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

      cars[0] = "Opel";
      System.out.println(cars[0]);

    }
}
```

```
// Outputs
Opel instead of Volvo
```

Array Length

To find out how many elements an array has, use the length property:

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

      String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
      System.out.println(cars.length);
      // Outputs 4

    }
}
```

```
// Outputs
4
```

Loops With Array

You can loop through the array elements with the for loop, and use the length property to specify how many times the loop should run.

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

      String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
      for (int i = 0; i < cars.length; i++) {
        System.out.println(cars[i]);
      }

    }
}
```

```
// Output
Volvo
BMW
Ford
Mazda
```

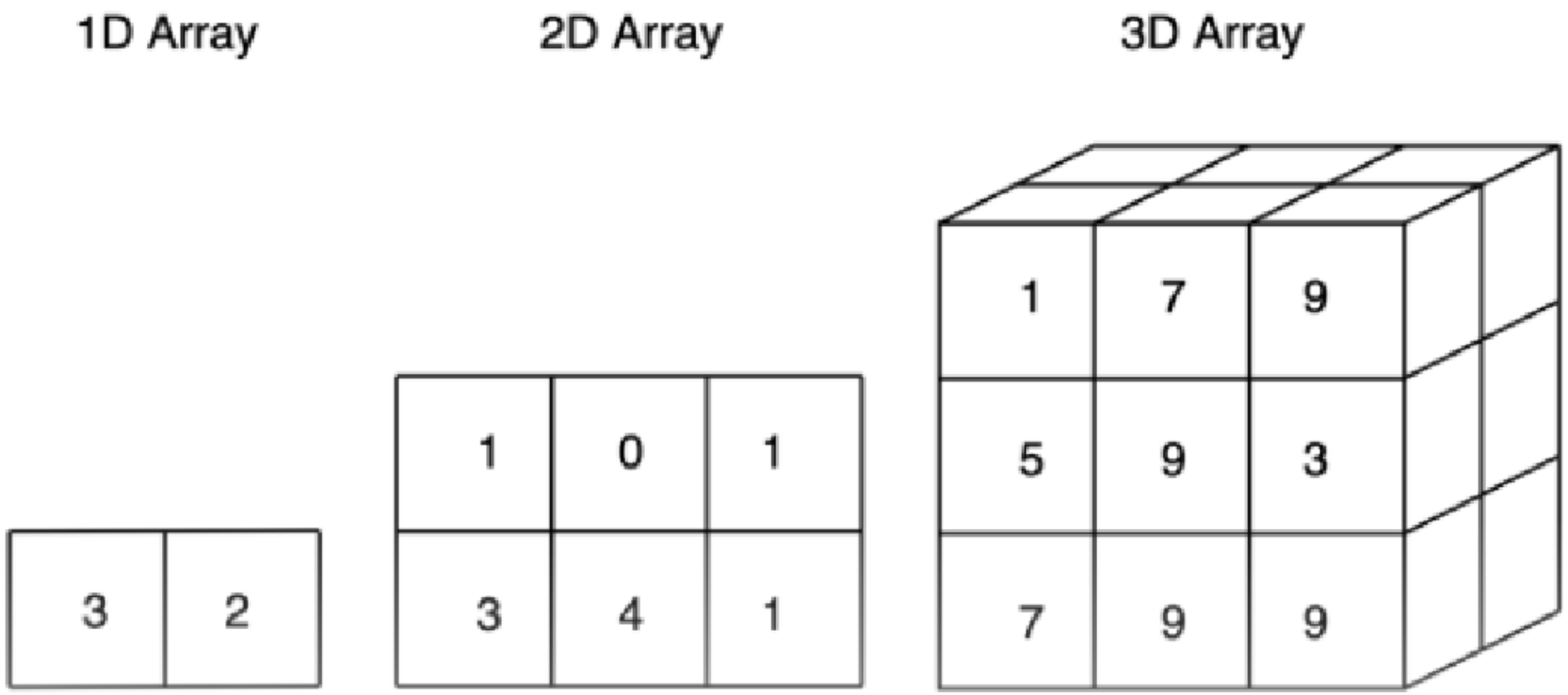
Loops With Array

There is also a "for-each" loop, which is used exclusively to loop through elements in arrays.

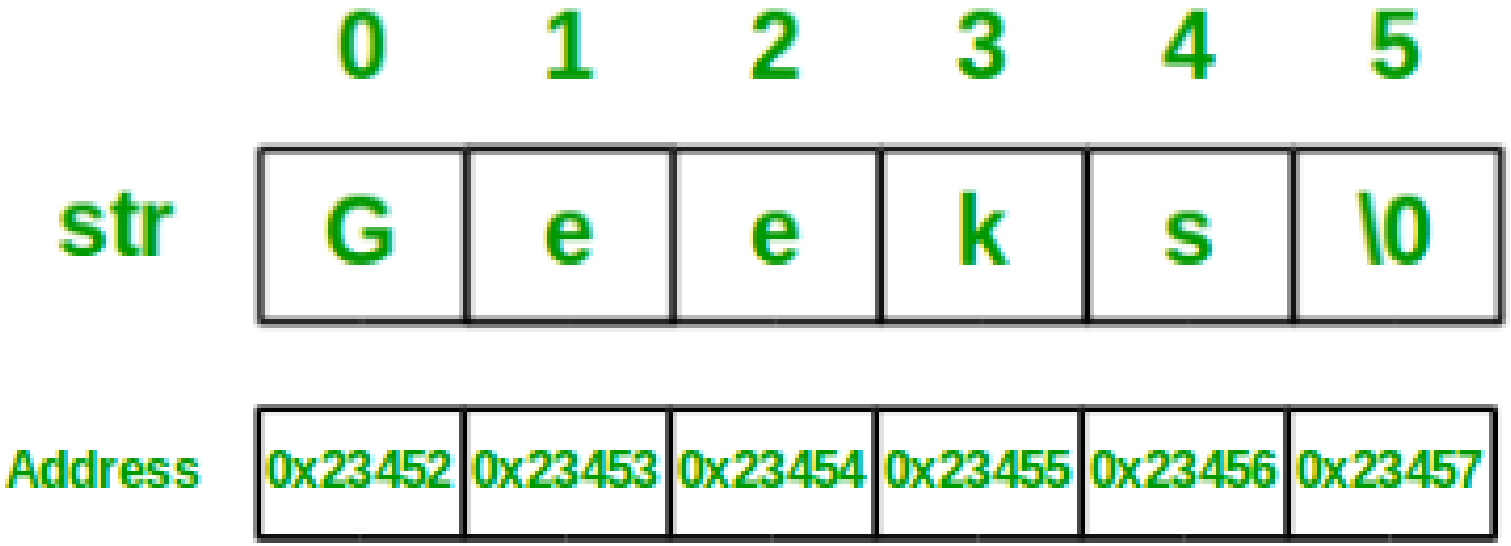
```
public class Test {
    public static void main (String[] args) {

      String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
      for (String i : cars) {
        System.out.println(i);
      }

    }
}
```

```
// Output
Volvo
BMW
Ford
Mazda
```

Arrays Dimensions



One-Dimensional Arrays



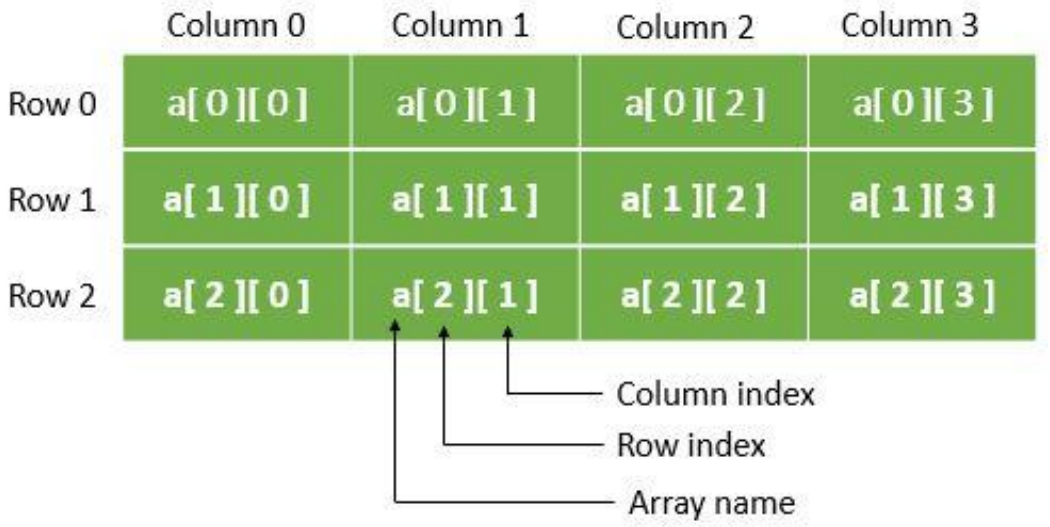
Two-Dimensional Arrays

Multidimensional arrays are useful when you want to store data as a tabular form, like a table with rows and columns.

To create a two-dimensional array, add each array within its own set of curly braces.

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7, 8} };
```

```
int[][] myNumbers = {
                      {1, 2, 3, 4},
                      {5, 6, 7, 8}
                    };
```

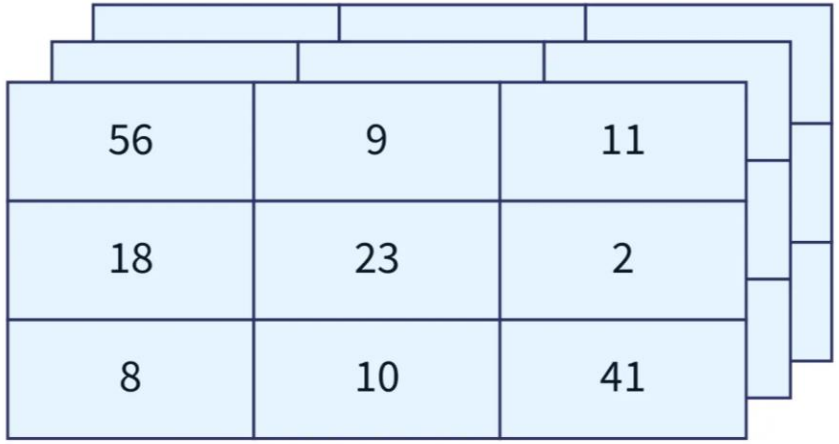


Multi-Dimensional Arrays

2-D Array

	Column 0	Column 1	Column 2
Row 0	a[0][0]	a[0][1]	a[0][2]
Row 1	a[1][0]	a[1][1]	a[1][2]
Row 2	a[2][0]	a[2][1]	a[2][2]
Row 3	a[3][0]	a[3][1]	a[3][2]
Row 4	a[4][0]	a[4][1]	a[4][2]

3-D Array



Access Elements

To access the elements of the myNumbers array, specify two indexes: one for the array, and one for the element inside that array. This example accesses the third element (2) in the second array (1) of myNumbers.

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

      int[][] myNumbers = { {1, 2, 3, 4},
                            {5, 6, 7, 8} };
      System.out.println(myNumbers[1][2]);
    }
}
```

```
// Output

7
```

Edit Elements

```
public class Test {

    public static void main (String[] args) {


      int[][] myNumbers = { {1, 2, 3, 4},
                            {5, 6, 7, 8} };

      myNumbers[1][2] = 9;

      System.out.println(myNumbers[1][2]);

    }                                                    // Outputs

}                                                        9 instead of 7
```

Example

We can also use a for loop inside another for loop to get the elements of a two-dimensional array (we still have to point to the two indexes).

```
public class Test
{
    public static void main (String[] args)
    {
        int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
        for (int i = 0; i < myNumbers.length; ++i)
        {
            for(int j = 0; j < myNumbers[i].length; ++j)
            {
              System.out.println(myNumbers[i][j]);
            }
        }
    }
}
```

	Column 0	Column 1	Column 2
Row 0	a[0][0]	a[0][1]	a[0][2]
Row 1	a[1][0]	a[1][1]	a[1][2]
Row 2	a[2][0]	a[2][1]	a[2][2]
Row 3	a[3][0]	a[3][1]	a[3][2]
Row 4	a[4][0]	a[4][1]	a[4][2]

```
// Output
1
2
3
4
5
6
7
```

Thanks

References: https://www.w3schools.com