

INTRODUCTION TO JAVA

Java 1.0



ARRAYS

Lesson # 06



ARRAYS

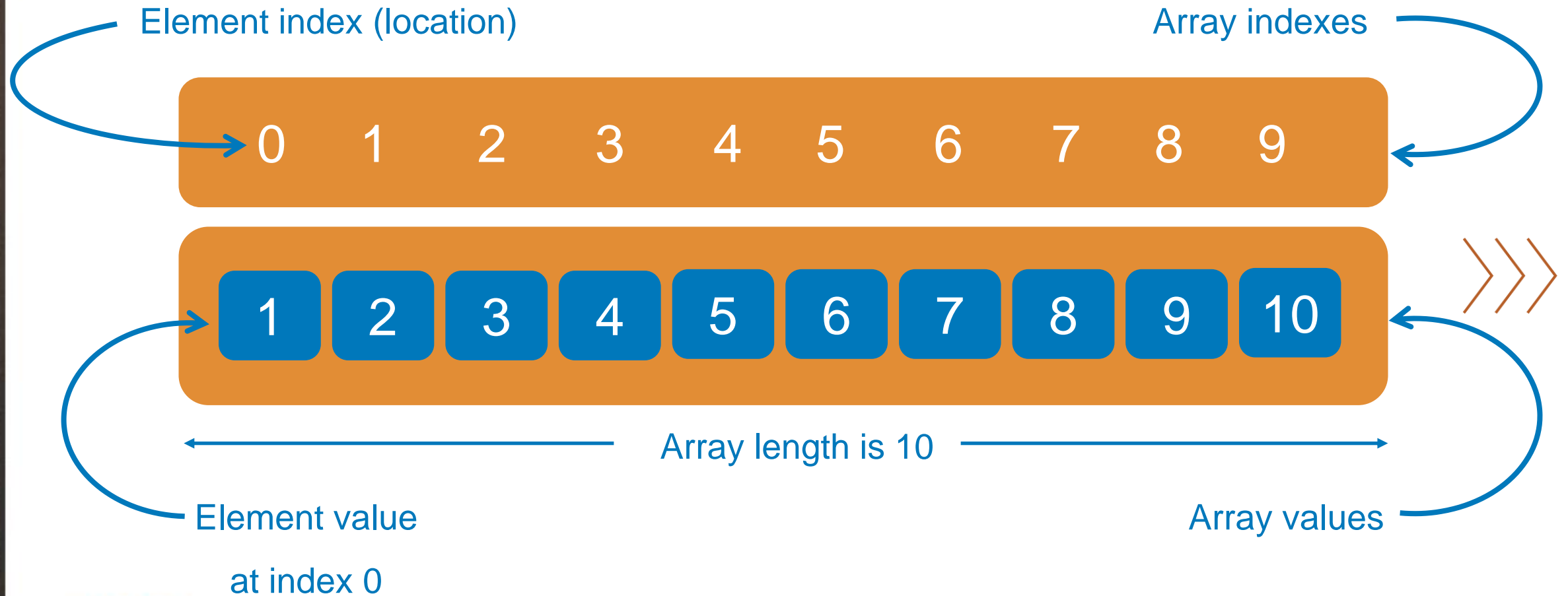


ARRAY DEFINITION

- An array is a **container** object that holds a **fixed** number of values of a **single type**
- The **length** of an array is established when the array is **created**
- **After** creation, its **length is fixed**



ARRAY VISUALISATION



ARRAY SYNTAX

- Array declaration **without** instantiation
- Array declaration **with** instantiation
- Array declaration **with** inline initialization

```
type[] name;
```

```
type[] name = new type[size];
```

```
type[] name = {var1, ..., varN};
```


ARRAY DECLARATION EXAMPLE

Code

```
int[] leapYears = new int[3];  
leapYears[0] = 2020;  
leapYears[1] = 2016;  
leapYears[2] = 2012;  
System.out.println("Leap years = " + Arrays.toString(leapYears));
```

Console Output

```
Leap years = [2020, 2016, 2012]
```

```
Process finished with exit code 0
```

ARRAY DECLARATION EXAMPLE

Code

```
int[] leapYears = {2020, 2016, 2012};  
System.out.println("Leap years = " + Arrays.toString(leapYears));
```

Console Output

```
Leap years = [2020, 2016, 2012]
```

```
Process finished with exit code 0
```


PROCESSING ARRAYS



WORKING WITH ARRAYS

- When working with arrays, **loops** are often used because of array **iterative** nature
- Array contains elements of the **single type** and **size** is **fixed** and known in advance



PRINTING ARRAY CONTENT

```
public class PrintingArrayDemo {  
  
    public static void main(String[] args) {  
        String[] alphabet = new String[5];  
  
        alphabet[0] = "A";  
        alphabet[1] = "B";  
        alphabet[2] = "C";  
        alphabet[3] = "D";  
        alphabet[4] = "E";  
  
        for (int i = 0; i < alphabet.length; i++) {  
            System.out.println "[" + i + "]: " + alphabet[i]);  
        }  
    }  
}
```



SUM OF ARRAY ELEMENTS

```
public class SumOfArrayElementsDemo {  
  
    public static void main(String[] args) {  
        int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9};  
  
        int sum = 0;  
        for (int i = 0; i < numbers.length; i++) {  
            sum += numbers[i];  
        }  
  
        System.out.println("Sum = " + sum);  
    }  
}
```



FOR EACH LOOPS



FOR EACH LOOP

- For each loop, also known as enhanced loop, is **another way** to traverse the array
- There is **no use** of the **index** or rather the **counter variable**
- Data type declared in the foreach **must match** the data type of the array that you are iterating
- Can access only **current** element
- **Significantly** reduces amount of code



FOR EACH LOOP SYNTAX

For each loop
declaration

```
type[] name = {var1, ..., varN};
```

```
for (type item : name) {  
    //Code goes in here  
}
```



Iterator
specification

Statement that executed inside of the loop body

FOR EACH LOOP DEMO

```
public static void main(String[] args) {  
    String[] dogBreeds = {  
        "Beagle",  
        "Golden Retriever",  
        "Pug",  
        "Shiba Inu"  
    };  
  
    for (String breed : dogBreeds) {  
        System.out.println(breed);  
    }  
}
```



STATIC KEYWORD



STATIC KEYWORD

- The keyword static indicates that the particular member belongs to a **type itself**, rather than to an **instance** of that type
- Only **one instance** of that static member is created which is **shared** across all instances of class
- **Can be applied** to the following elements:
 - Fields (variables)
 - Methods
 - Inner methods
 - Static code block



STATIC FIELDS

- Exactly a **single copy** of static field is created and **shared** among instances of that class
- No matter how many times class is initialized.. **Always single copy** of static field



STATIC FIELD EXAMPLE

```
public class Message {  
  
    public static int instancesCreated = 0;  
    private String text;  
  
    public Message(String text) {  
        this.text = text;  
        System.out.println("Creating message = '" + text + "'");  
        instancesCreated++;  
    }  
  
}
```



```
}
```


STATIC FIELD EXAMPLE

Code

```
System.out.println("Created = " + Message.instancesCreated);  
Message greeting = new Message("Hi!");  
Message question = new Message("How are you?");  
Message farewell = new Message("Goodbye!");  
System.out.println("Created = " + Message.instancesCreated);
```



Console Output

```
Created = 0  
Creating message = 'Hi!'  
Creating message = 'How are you?'  
Creating message = 'Goodbye!'  
Created = 3
```

WHEN TO USE STATIC FIELDS?

- When the value of variable is **independent** of objects
- When the value is supposed to be **shared** across all objects



KEY POINTS TO REMEMBER

- Since static fields **belong** to a **class**, they can be **accessed directly** using **class name** and don't need any object reference
- Static variables **can only** be **declared** at the **class level**
- Static fields can be **accessed without** object **initialization**
- Although static field can be accessed through reference, **access via class name is preferred**



STATIC METHODS

- Also belong to a **class** instead of the object
- Can be called **without** creating the object of the class in which they reside
- Generally used to perform an operation that is not **dependent** upon instance creation
- Widely used to create utility classes so that they can be obtained **without creating** a new object of these classes



STATIC METHOD EXAMPLE

```
public class QuickMaths {  
  
    public static int min(int[] numbers) {  
        if (numbers.length == 0) {  
            return 0;  
        }  
  
        int min = numbers[0];  
  
        for (int number : numbers) {  
            if (number < min) {  
                min = number;  
            }  
        }  
        return min;  
    }  
}
```



STATIC METHOD EXAMPLE

Code

```
int[] values = {44, 65, 61, 16, 89};  
int result = QuickMaths.min(values);  
System.out.println("result = " + result);
```



Console Output

```
result = 16  
Process finished with exit code 0
```

WHEN TO USE STATIC METHODS?

- To **access** or manipulate static variables and other static members that don't depend upon objects
- Widely used in **stateless** utility classes



KEY POINTS TO REMEMBER

- Static methods cannot be **overridden**
- Instance methods can **directly access** both **instance methods** and **instance variables**
- Instance methods can **directly access** both **static variables** and **static methods**
- Static methods **can access** all **static variables** and other **static methods**
- Static methods **cannot** access instance variables and instance methods directly; only via **object reference**



REFERENCES

REFERENCES

- <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html>
- <https://www.javatpoint.com/array-in-java>
- <https://www.baeldung.com/java-arrays-guide>
- <https://www.baeldung.com/java-static>
- <https://www.geeksforgeeks.org/static-keyword-java/>



QUESTIONS?



THANK YOU!

