String (concat, indexOf, split, length,toLowerCase, toUpperCase, replace, trim)

String

- A string is a sequence of characters, typically used to represent text. In many programming languages, including Java, JavaScript, Python, and C++, strings are immutable, meaning they cannot be changed once created.
- However, you can perform various operations on strings to manipulate them.

Concatenation

Concatenation is the process of combining two or more strings into a single string. It is often performed using the concatenation operator (+) or the concat() method.

```
let str1 = "Hello";

let str2 = "World";

let result = str1 + " " + str2; // Using the concatenation operator

console.log(result); // Output: "Hello World"


let result2 = str1.concat(" ", str2); // Using the concat() method

console.log(result2); // Output: "Hello World"
```

Index of()

indexOf(): The indexOf() method returns the index of the first occurrence of a specified value within a string. If the value is not found, it returns -1.

- String str = "Hello, World!";
- int index = str.indexOf("World");
- System.out.println(index); // Output: 7

Split of()

split(): The split() method splits a string into an array of substrings based on a specified delimiter and returns the array.

```
String str = "Hello,World,Java";

String[] parts = str.split(",");

for (String part : parts) {

    System.out.println(part);}
```

Output:

Hello

World

Java

Length of()

The length property or method returns the number of characters in a string.

- let str = "Hello";
- let length = str.length;
- console.log(length); // Output: 5

toLowerCase() and toUpperCase():

The toLowerCase() method converts a string to lowercase, while the toUpperCase() method converts it to uppercase.

- String str = "Hello, World!";
- String lowercase = str.toLowerCase();
- String uppercase = str.toUpperCase();
- System.out.println(lowercase); // Output: "hello, world!"
- System.out.println(uppercase); // Output: "HELLO, WORLD!"

replace()

replace():

The replace() method replaces all occurrences of a specified value with another value in a string.

```
String str = "Hello, World!";

String newStr = str.replace("Hello", "Hi");

System.out.println(newStr);
```

Output

```
Hi, World!
```

trim()

trim():

The trim() method removes whitespace from both ends of a string.

```
let str = "   Hello, World!   ";

let trimmed = str.trim();

console.log(trimmed); // Output: "Hello, World!"
```

Examples

```
String str = "Hello, World!";
int index = str.indexOf("World");
System.out.println(index);
```

Output??

Examples

```
String str = "Hello, World!";
String[] parts = str.split(",");
for (String part : parts) {
    System.out.println(part);
}
```

Output??

Examples

```
String str = "Hello";
int length = str.length();
System.out.println(length);
```

Output??

Examples

```
String str = "Hello, World!";
String lowercase = str.toLowerCase();
String uppercase = str.toUpperCase();
System.out.println(lowercase);
System.out.println(uppercase);
```

Output??

Examples

```
String str = "Hello, World!";
String newStr = str.replace("World", "Universe");
System.out.println(newStr);
```

Output??

Examples

```
String str = "Hello, World!";
String newStr = str.replace("World", "Universe");
System.out.println(newStr);
```

Output??

Examples

```
String str = "   Hello, World!   ";
String trimmed = str.trim();
System.out.println(trimmed);
```

Output??

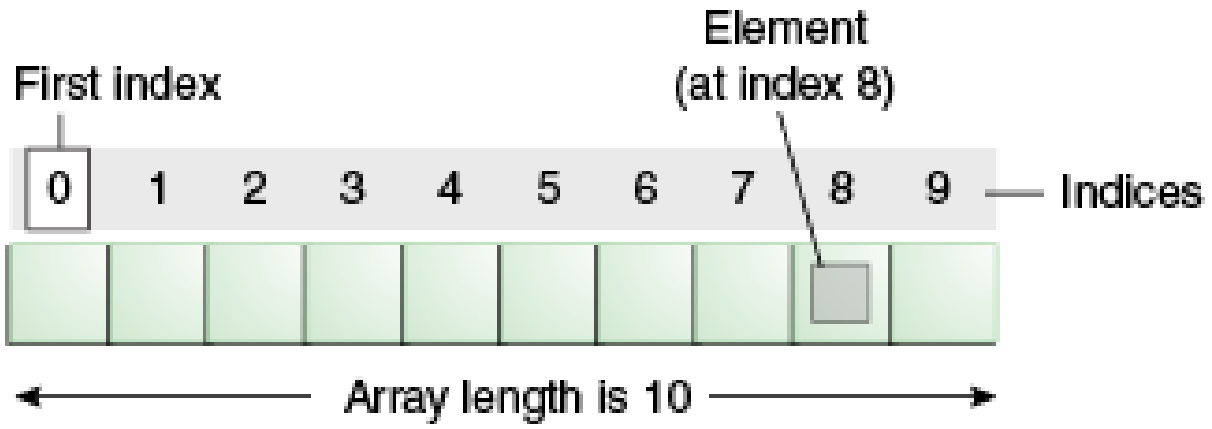
Arrays

What is Array in Java

- Array in java is a group of like-typed variables referred to by a common name.
- An array is a collection of similar type of elements which has contiguous memory location.
- Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location.

Continued

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location.



Pros and Cons

Advantages

- Code Optimization: It makes the code optimized, we can retrieve or sort the data efficiently.
- Random access: We can get any data located at an index position.

Disadvantages

- Size Limit: We can store only the fixed size of elements in the array. It doesn't grow its size at runtime.
- To solve this problem, collection framework is used in Java which grows automatically.

Types

There are two types of array.

- **Single Dimensional Array**
- **Multidimensional Array**

Single Dimensional Array

Syntax to Declare an Array in Java

dataType[] arr; (or)

dataType []arr; (or)

dataType arr[];

Operations in Array

- Declaration

e.g int a[]=new int[2]

- Initialization

e.g. a[0]=10

a[1]=12

- Traversing array

e.g. for(int i=0;i<a.length;i++)//length is the property of array

System.out.println(a[i]);

Example 1

```
class Testarray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//traversing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```

Point to be Noted

We can declare and initialize the java array together by:

For e.g.

int a[]={33,3,4,5};//declaration and initialization

Example-2

```
1. class Testarray1{
2. public static void main(String args[]){
3. int a[]={33,3,4,5};//declaration and initialization
4. //printing array
5. for(int i=0;i<a.length;i++)//length is the property of array
6. System.out.println(a[i]);
7. }}
```

For-each Loop for Java Array

- We can also print the Java array using for-each loop.
- The Java for-each loop prints the array elements one by one.
- It holds an array element in a variable, then executes the body of the loop.

Syntax

The syntax of the for-each loop is given below:

```
for(data_type variable:array){
//body of the loop
}
```

Example

```
1. class Testarray1{
2. public static void main(String args[]){
3. int arr[]={33,3,4,5};
4. //printing array using for-each loop
5. for(int i:arr)
6. System.out.println(i);
7. }}
```

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.

Example

```
1. class Testarray2{
2. //creating a method which receives an array as a parameter
3. static void min(int arr[]){
4. int min=arr[0];
5. for(int i=1;i<arr.length;i++)
6.  if(min>arr[i])
7.   min=arr[i];
8. System.out.println(min);
9. }
10.
11. public static void main(String args[]){
12. int a[]={33,3,4,5};//declaring and initializing an array
13. min(a);//passing array to method
14. }}
```

Anonymous Array in Java

Java supports the feature of an anonymous array, so you don't need to declare the array while passing an array to the method.

Anonymous Array in Java

```
1. public class TestAnonymousArray{
2. //creating a method which receives an array as a parameter
3. static void printArray(int arr[]){
4. for(int i=0;i<arr.length;i++)
5. System.out.println(arr[i]);
6. }
7.
8. public static void main(String args[]){
9. printArray(new int[]{10,22,44,66});//passing anonymous array to method
10.}}
```

Array IndexOut Of Bounds Exception

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.

Example

```
1. //Java Program to demonstrate the case of
2. //ArrayIndexOutOfBoundsException in a Java Array.
3. public class TestArrayException{
4. public static void main(String args[]){
5. int arr[]={50,60,70,80};
6. for(int i=0;i<=arr.length;i++){
7. System.out.println(arr[i]);
8. }
9. }}
```

Multidimensional Array in Java

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

Syntax to Declare Multidimensional Array in Java

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

Multidimensional Array in Java

Example to instantiate Multidimensional Array in Java

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

Example

```
1. //Java Program to illustrate the use of multidimensional array
2. class Testarray3{
3. public static void main(String args[]){
4. //declaring and initializing 2D array
5. int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
6. //printing 2D array
7. for(int i=0;i<3;i++){
8.  for(int j=0;j<3;j++){
9.    System.out.print(arr[i][j]+" ");
10. }
11. System.out.println();
12.}
13.}}
```

Jagged Array

- If we are creating odd number of columns in a 2D array, it is known as a jagged array.
- In other words, it is an array of arrays with different number of columns.

Jagged Array

```
1.  class TestJaggedArray{
2.    public static void main(String[] args){
3.      //declaring a 2D array with odd columns
4.      int arr[][] = new int[3][];
5.      arr[0] = new int[3];
6.      arr[1] = new int[4];
7.      arr[2] = new int[2];
8.      //initializing a jagged array
9.      int count = 0;
10.     for (int i=0; i<arr.length; i++)
11.       for(int j=0; j<arr[i].length; j++)
12.         arr[i][j] = count++;
13.
14.     //printing the data of a jagged array
15.     for (int i=0; i<arr.length; i++){
16.       for (int j=0; j<arr[i].length; j++){
17.         System.out.print(arr[i][j]+" ");
18.       }
19.       System.out.println();//new line
20.     }
21.   }
```

Examples

```
1.  public class OneDimensionalArrayExample {
2.      public static void main(String[] args) {
3.          // Declare an array of integers
4.          int[] numbers = new int[5]; // Creates an array of size 5
5.
6.          // Initialize array elements
7.          numbers[0] = 10;
8.          numbers[1] = 20;
9.          numbers[2] = 30;
10.         numbers[3] = 40;
11.         numbers[4] = 50;
12.         // Access array elements
13.         System.out.println("Element at index 2: " + numbers[2]); // Output: 30
14.
15.         // Loop through array elements
16.         System.out.println("Array elements:");
17.         for (int i = 0; i < numbers.length; i++) {
18.             System.out.println(numbers[i]);
```

```
public class BasicArrayExample {
    public static void main(String[] args) {
        // Declare and initialize an array of strings
        String[] fruits = {"Apple", "Banana", "Orange", "Grapes", "Mango"};

        // Access and print array elements using a loop
        System.out.println("Fruits in the array:");
        for (int i = 0; i < fruits.length; i++) {
            System.out.println(fruits[i]);
        }

        // Modify an element
        fruits[2] = "Pineapple";

        // Print the modified array
        System.out.println("\nUpdated fruits in the array:");
        for (int i = 0; i < fruits.length; i++) {
            System.out.println(fruits[i]);
        }
    }
}
```

```
public class ArrayCalculationExample {
    public static void main(String[] args) {
        // Declare and initialize an array of integers
        int[] numbers = {5, 10, 15, 20, 25};

        // Calculate the sum of array elements
        int sum = 0;
        for (int i = 0; i < numbers.length; i++) {
            sum += numbers[i];
        }
        // Calculate the average
        double average = (double) sum / numbers.length;

        // Print the sum and average
        System.out.println("Array elements:");
        for (int i = 0; i < numbers.length; i++) {
            System.out.println(numbers[i]);
        }
        System.out.println("\nSum: " + sum);
        System.out.println("Average: " + average);
    }
}
```

Methods in Java

What are Methods and Arguments

- In Java, methods can take zero or more parameters, also known as arguments.
- These parameters are placeholders for values that you want to pass into the method for processing.

Syntax Example:

```
public returnType methodName(parameterType1 paramName1, parameterType2 paramName2, ...)
{
 // Method body
}
```

Methods with Arguments and Return Values example

```
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }
    public int subtract(int a, int b) {
        return a - b;
    }
    public static void main(String[] args) {
        Calculator calc = new Calculator();

        int result1 = calc.add(5, 3);
        int result2 = calc.subtract(10, 4);

        System.out.println("Addition: " + result1);    // Output: Addition: 8
        System.out.println("Subtraction: " + result2); // Output: Subtraction: 6
    }
}
```

Methods with Arguments and Return Values example 2

```
public class RectangleCalculator {

    public static double calculateRectangleArea(double width, double height) {
        return width * height;
    }

    public static void main(String[] args) {
        double width = 5.0;
        double height = 3.0;

        double area = calculateRectangleArea(width, height);
        System.out.println("Rectangle Area: " + area);
    }
}
```

Methods with Arguments and Return Values example 3

```
public class MaxNumberFinder {

    public static int findMax(int num1, int num2, int num3) {
        int max = num1;
        if (num2 > max) {
            max = num2;
        }
        if (num3 > max) {
            max = num3;
        }
        return max;
    }
    public static void main(String[] args) {
        int num1 = 12;
        int num2 = 25;
        int num3 = 9;

        int maxNumber = findMax(num1, num2, num3);
        System.out.println("Maximum Number: " + maxNumber);
    }
}
```

Methods with Return Values:

- In Java, methods can also return values of a specific type.
- The return value is indicated by the return statement within the method body.
- The return type is defined when the method is declared.

Methods with Return Values Syntax

- public returnType methodName(parameterType1 paramName1, parameterType2 paramName2, ...) {
- // Method body
- return returnValue;
- }

Methods with Return Values Example

```
public double calculateAverage(int[] numbers) {
    int sum = 0;
    for (int num : numbers) {
        sum += num;
    }
    return (double) sum / numbers.length;
}
```

Methods with Return Values Example 2

```
public class EvenNumberChecker {

    public static boolean isEven(int number) {
        return number % 2 == 0;
    }

    public static void main(String[] args) {
        int num1 = 10;
        int num2 = 7;

        boolean even1 = isEven(num1);
        boolean even2 = isEven(num2);

        System.out.println(num1 + " is even? " + even1);  // Output: 10 is even? true
        System.out.println(num2 + " is even? " + even2);  // Output: 7 is even? false
    }
}
```

Static Variables and Static Method

- In Java, a static variable (also known as a class variable) belongs to the class rather than to any specific instance of the class.
- This means that all instances of the class share the same static variable.
- Static variables are initialized when the class is loaded and exist throughout the entire lifetime of the program.

Static Variables and Static Method

- Key points about static variables:
- They are declared using the static keyword.
- They are shared among all instances of the class.
- They are accessed using the class name, followed by the dot (.) operator.
- They are often used to store data that should be consistent across all instances of the class.

Static Variables and Static Method Example

```
public class Student {
    static int totalStudents = 0;

    Student() {
        totalStudents++;
    }
}
.
```

Static Method

- A static method in Java belongs to the class rather than to any instance of the class.
- It can be called using the class name, without creating an instance of the class.
- Static methods are often used for utility functions that are not tied to a specific instance of the class.

Static Method

- They are declared using the static keyword.
- They can only directly access other static members (variables or methods) of the class.
- They can be called using the class name, followed by the dot (.) operator.
- They are useful for operations that don't require access to instance-specific data.

Static Method Example

- public class MathUtils {
- static int add(int a, int b) {
- return a + b;
- }
- }
- In this Example add method is a static method of the MathUtils class that can be called using MathUtils.add(5, 3)

Static Method Use Cases

- Static variables and methods are commonly used for:
- Constants that are shared among all instances of a class (e.g., mathematical constants).
- Utility functions that don't require instance-specific data.
- Counting instances or maintaining shared data (as in the totalStudents example above).
- Creating factory methods to construct instances of a class.

Static Variables and Static Methods

```
public class Counter {
    static int count = 0;  // Static variable

    public Counter() {
        count++;
    }

    public static void displayCount() {
        System.out.println("Total instances: " + count);
    }

    public static void main(String[] args) {
        Counter c1 = new Counter();
        Counter c2 = new Counter();

        Counter.displayCount();  // Output: Total instances: 2
    }
}
```

Constructors and Destructors

What are Constructors

- In object oriented programming, both constructor and destructor are the member functions of a class having the same name as the class.
- A constructor helps in initialization of an object, i.e., it allocates memory to an object.
- On the other hand, a destructor deletes the created constructor when it is of no use which means it deallocates the memory of an object.

Contd.

- A constructor is a member function of a class that initializes the object and allocates the memory.
- A constructor has the same name as that of its class, thus it can be easily identified.
- It is always declared and defined in the public section of a class.
- A constructor does not have any return type. Therefore, it does not return anything, but it is not even void.

Syntax

```
class_name (arguments if any){

   ...

   ...

};
```

A single class may have multiple constructors that are differentiated based on the number and type of arguments passed. There are three main types of constructors, namely, **default constructor (no arg), parameterized constructor and Copy Constructor.**

Rules for Creating Java Constructor

- Constructor name must be the same as its class name
- A Constructor must have no explicit return type
- A Java constructor cannot be abstract, static, final, and synchronized

Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

<class_name>(){}

Default Constructor Example

```
//Java Program to create and call a default constructor
class Bike1{
//creating a default constructor
Bike1()
{
System.out.println("Bike is created");
}
//main method
public static void main(String args[]){
//calling a default constructor
Bike1 b=new Bike1();
```

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Example

```
1.  //Let us see another example of default constructor
2.  //which displays the default values
3.  class Student3{
4.  int id;
5.  String name;
6.  //method to display the value of id and name
7.  void display(){System.out.println(id+" "+name);}
8.
9.  public static void main(String args[]){
10. //creating objects
11. Student3 s1=new Student3();
12. Student3 s2=new Student3();
13. //displaying values of the object
14. s1.display();
15. s2.display();
16. }
17. }
```

Parameterized Method

- A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

- The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example

```
1.  //Java Program to demonstrate the use of the parameterized constructor.
2.  class Student4{
3.     int id;
4.     String name;
5.     //creating a parameterized constructor
6.     Student4(int i,String n){
7.     id = i;
8.     name = n;
9.     }
10.    //method to display the values
11.    void display(){System.out.println(id+" "+name);}
12.
13.    public static void main(String args[]){
14.    //creating objects and passing values
15.    Student4 s1 = new Student4(111,"Karan");
16.    Student4 s2 = new Student4(222,"Aryan");
17.    //calling method to display the values of object
18.    s1.display();
19.    s2.display();
20.   }
21. }
```

Constructor Overloading

- In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.
- Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Example

```
1.  //Java program to overload constructors
2.  class Student5{
3.     int id;
4.     String name;
5.     int age;
6.     //creating two arg constructor
7.     Student5(int i,String n){
8.     id = i;
9.     name = n;
10.    }
11.    //creating three arg constructor
12.    Student5(int i,String n,int a){
13.    id = i;
14.    name = n;
15.    age=a;
16.    }
17.    void display(){System.out.println(id+" "+name+" "+age);}
18.
19.    public static void main(String args[]){
20.    Student5 s1 = new Student5(111,"Karan");
21.    Student5 s2 = new Student5(222,"Aryan",25);
22.    s1.display();
23.    s2.display();
24.   }
25. }
```

Difference

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

Copy Constructor

Java Copy Constructor

1.There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in Java. They are:

By constructor

1.By assigning the values of one object into another

2.By clone() method of Object class

Example

```
1.  //Java program to initialize the values from one object to another object.
2.  class Student6{
3.      int id;
4.      String name;
5.      //constructor to initialize integer and string
6.      Student6(int i,String n){
7.      id = i;
8.      name = n;
9.      }
10.     //constructor to initialize another object
11.     Student6(Student6 s){
12.     id = s.id;
13.     name =s.name;
14.     }
15.     void display(){System.out.println(id+" "+name);}
16.
17.     public static void main(String args[]){
18.     Student6 s1 = new Student6(111,"Karan");
19.     Student6 s2 = new Student6(s1);
20.     s1.display();
21.     s2.display();
22.    }
```

Destructor

- A destructor is a member function of a class that deallocates the memory allocated to an object. A destructor is also declared and defined with the same name as that of the class. A destructor is preceded by a tilde (~) symbol. A single class has only a single destructor.

- ~ class_name (no arguments){
- ...
- ...
- };

Contd.

- A destructor does not have any argument and is always called in the reverse order of the constructor.

- Destructor are required for destroying the objects to release the memory allocated to them.

Difference

Constructor	Destructor
Constructors help allocate memory to an object.	Destructors deallocate the memory of an object.
Constructors can take arguments.	Destructors don't take any arguments.
Constructors are called automatically when an object is created.	Destructors are called automatically when the block is exited or when the program terminates.
Constructors allow an object to initialize a value before it is used.	They allow objects to execute code when it is being destroyed.
They are called in the successive order of their creation.	They are called in the reverse order of their creation.
There can be multiple constructors in a single class.	There is a single destructor in a class.
Constructors can be overloaded.	Destructors cannot be overloaded.
The concept of copy constructor allows an object to get initialized from another object.	There is no such concept in the case of destructors.

Access modifiers (public, protected, private, default)

- The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class.
- We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

Access modifiers (public, protected, private, default)

- There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

Access modifiers (public, protected, private, default)

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Access modifiers (public, protected, private, default)

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc.

Acc to syllabus, we have to learn the access modifiers only.