NON-PRIMITIVE DATA TYPES (ARRAYS, STRINGS AND LIST)

Arrays: Objects that store a fixed-size sequential collection of elements of the same type. Arrays are created using the array initializer syntax **[]** or using the **new** keyword with a specified size.

Arrays in Java are declared using square brackets **[]**. There are two ways to initialize an array:

1. **Using Array Initializer:**

Declaration and initialization in a single line **int[] numbers = {1, 2, 3, 4, 5};**

2. **Using the new Keyword:**

Declaration and initialization separately **int[] numbers = new int[5];** // Creates an array of size 5

numbers[0] = 1; numbers[1] = 2; // Assign values to other elements...

Iterating Through an Array:

Arrays can be traversed using loops such as **for** or **foreach**.

```
int[] numbers = {1, 2, 3, 4, 5};
for (int i = 0; i < numbers.length; i++) {
   System.out.println(numbers[i]); // Prints each element of the array
}

// Enhanced for loop (foreach loop)
for (int x : numbers) {
   System.out.println(x); // Prints each element of the array
}
```

Multidimensional Arrays

Multidimensional arrays are declared by specifying multiple sets of square brackets **[]**. You can initialize a multidimensional array in several ways:

1. **Using Array Initializer:**

```
// Declaration and initialization in a single line int[][] matrix = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
```

2. **Using the new Keyword:**

Declaration and initialization separately **int[][] matrix = new int[3][3];**// Creates a 3x3 matrix

matrix[0][0] = 1;

matrix[0][1] = 2; // Assign values to other elements...

Accessing Elements:

Elements of a multidimensional array are accessed using multiple indices corresponding to each dimension.

int[][] matrix = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };

int element = matrix[1][2]; // Accessing the element at row 1, column 2 (6)

Length of Each Dimension:

You can find the length of each dimension of a multidimensional array using the **length** property.

int[][] matrix = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };

int rows = matrix.length; // Returns the number of rows (3)

int columns = matrix[0].length; // Returns the number of columns (3)

Iterating Through a Multidimensional Array:

You can traverse a multidimensional array using nested loops.

```
int[][] matrix = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
for (int i = 0; i < matrix.length; i++) {
   for (int j = 0; j < matrix[i].length; j++) {
      System.out.print(matrix[i][j] + " ");
   }
   System.out.println(); // Move to the next line after each row
}
```

Jagged Arrays

In Java, a jagged array is a multidimensional array in which the rows can have different lengths. Unlike regular multidimensional arrays, where each row has the same number of columns, jagged arrays allow for flexibility in representing irregular data structures. Here's an overview of jagged arrays in Java:

Declaration and Initialization:

Jagged arrays are declared and initialized similarly to regular multidimensional arrays, but each row can have a different length.

```
int[][] jaggedArray = {
   {1, 2},
   {3, 4, 5},
   {6, 7, 8, 9}
};
```

Accessing Elements:

Elements of a jagged array are accessed using multiple indices corresponding to each dimension.

```
int[][] jaggedArray = {
   {1, 2},
   {3, 4, 5},
   {6, 7, 8, 9}
};
int element = jaggedArray[1][2]; // Accessing the element at row 1, column 2 (5)
```

Length of Each Row:

You can find the length of each row of a jagged array using the **length** property of each row.

```
int[][] jaggedArray = {
   {1, 2},
   {3, 4, 5},
   {6, 7, 8, 9}
};
int rowLength1 = jaggedArray[0].length; // Returns the length of row 0 (2)
int rowLength2 = jaggedArray[1].length; // Returns the length of row 1 (3)
```

Iterating Through a Jagged Array:

You can traverse a jagged array using nested loops.

```
int[][] jaggedArray = {
   {1, 2},
   {3, 4, 5},
   {6, 7, 8, 9}
};
for (int i = 0; i < jaggedArray.length; i++) {
   for (int j = 0; j < jaggedArray[i].length; j++) {
      System.out.print(jaggedArray[i][j] + " ");
   }
   System.out.println(); // Move to the next line after each row
}
```

STRINGS

String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

There are two ways to create String object:

1. By string literal
2. By new keyword

1) Java String literal is created by using double quotes. For Example:

1. String s="welcome";

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

1. String s1="Welcome";
2. String s2="Welcome";//It doesn't create a new instance

2) By new keyword

1. String s=**new** String("Welcome");

String s1="java";//creating string by Java string literal

char ch[]={'s','t','r','i','n','g','s'};

String s2=new String(ch);//converting char array to string

String s3=new String("example");//creating Java string by new keyword

System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

The above code, converts a ***char*** array into a **String** object. And displays the String objects ***s1, s2***, and ***s3*** on console using ***println()*** method.

Java String charAt()

The **Java String class charAt()** method returns *a char value at the given index number*.

```
String name="Sukkur IBA";
char ch=name.charAt(4);  //returns the char value at the 5th index
System.out.println(ch);
```

Java String contains()

The **Java String class contains()** method searches the sequence of characters in this string. It returns *true* if the sequence of char values is found in this string otherwise returns *false*. The contains() method searches case-sensitive char sequence.

```
String name2="what do you know about me";
System.out.println(name2.contains("do you know"));  // true
System.out.println(name2.contains("about"));  // true
System.out.println(name2.contains("hello"));  // false
```

Java String length()

The **Java String class length()** method finds the length of a string.

```
String s1="Sukkur";
String s2="IBA";
System.out.println("string length is: "+s1.length()); // 6 is length
System.out.println("string length is: "+s2.length()); // 3 is length
```

Java String isEmpty()

The **Java String class isEmpty()** method checks if the input string is empty or not. Note that here empty means the number of characters contained in a string is zero.

```
String ie1="";
String ie2="SIBAU";
System.out.println(ie1.isEmpty());  //true
System.out.println(ie2.isEmpty()); //false
```

Java String equals()

The **Java String class equals()** method compares the two given strings based on the content of the string. If any character is not matched, it returns false. If all characters are matched, it returns true.

```
String s1="SukkurIBA";
String s2="SukkurIBA";
String s3="SUKKURIBA";
String s4="University";
System.out.println(s1.equals(s2)); //true because content and case are same
System.out.println(s1.equals(s3)); //false because case is not same
System.out.println(s1.equals(s4)); //false because content is not same
```

Java String concat

The **Java String class concat()** method *combines specified string at the end of this string*. It returns a combined string. It is like appending another string.

```
String str1 = "Sukkur ";
String str2 = "IBA ";
String str3 = "University";

// Concatenating one string
String str4 = str1.concat(str2);
System.out.println(str4);

// Concatenating multiple strings
String str5 = str1.concat(str2).concat(str3);
System.out.println(str5);
```

Java String replace()

The **Java String class replace()** method returns a string replacing all the old char or CharSequence to new char or CharSequence.

```
String rep1="I am studying in Sukkur. I am in Sukkur IBA University";
String replaceString=rep1.replace("am","was"); //replaces all occurrences of "am" to "was"
System.out.println(replaceString);
```

Java List

List in Java provides the facility to maintain the *ordered collection*. It contains the index-based methods to insert, update, delete and search the elements. It can have the duplicate elements also. We can also store the null elements in the list.

The List interface is found in the java.util package and inherits the Collection interface. It is a factory of ListIterator interface. Through the ListIterator, we can iterate the list in forward and backward directions. The implementation classes of List interface are ArrayList, LinkedList, Stack and Vector. The ArrayList and LinkedList are widely used in Java programming. The Vector class is deprecated since Java 5.

How to create List

```
//Creating a List of type String using ArrayList
List<String> list=new ArrayList<String>();

//Creating a List of type Integer using ArrayList
List<Integer> list=new ArrayList<Integer>();

//Creating a List of type Book using ArrayList
List<Book> list=new ArrayList<Book>();
```

//Creating a List

```
List<String> list=new ArrayList<String>();
//Adding elements in the List
list.add("Mango");
list.add("Apple");
list.add("Banana");
list.add("Grapes");
//Iterating the List element using for-each loop
for(String fruit:list)
 System.out.println(fruit);
```

How to convert Array to List

We can convert the Array to List by traversing the array and adding the element in list one by one using list.add() method. Let's see a simple example to convert array elements into List.

```
//Creating Array
String[] array={"Java","Python","PHP","C++"};
System.out.println("Printing Array: "+Arrays.toString(array));

//Converting Array to List
List<String> list=new ArrayList<String>();

for(String lang:array){
list.add(lang);
}
System.out.println("Printing List: "+list);
```

OUTPUT

```
Printing Array: [Java, Python, PHP, C++]
Printing List: [Java, Python, PHP, C++]
```

How to convert List to Array

We can convert the List to Array by calling the list.toArray() method. Let's see a simple example to convert list elements into array.

```
List<String> fruitList = new ArrayList<>();
 fruitList.add("Mango");
 fruitList.add("Banana");
 fruitList.add("Apple");
```

```
fruitList.add("Strawberry");
//Converting ArrayList to Array
String[] array = fruitList.toArray(new String[fruitList.size()]);
System.out.println("Printing Array: "+Arrays.toString(array));
System.out.println("Printing List: "+fruitList);
```

OUTPUT

Printing Array: [Mango, Banana, Apple, Strawberry]

Printing List: [Mango, Banana, Apple, Strawberry]

Get and Set Element in List

The get() method returns the element at the given index, whereas the set() method changes or replaces the element.

```
//Creating a List
List<String> list=new ArrayList<String>();
//Adding elements in the List
list.add("Mango");
list.add("Apple");
list.add("Banana");
list.add("Grapes");
//accessing the element
System.out.println("Returning element: "+list.get(1));//it will return the 2nd element, because index starts from 0
//changing the element
list.set(1,"Dates");
//Iterating the List element using for-each loop
for(String fruit:list){
 System.out.println(fruit);  }
```

How to Sort List

There are various ways to sort the List, here we are going to use Collections.sort() method to sort the list element. The java.util package provides a utility class Collections which has the static method sort(). Using the Collections.sort() method, we can easily sort any List.

```
List<String> list1=new ArrayList<String>();
 list1.add("Mango");
 list1.add("Apple");
 list1.add("Banana");
 list1.add("Grapes");
 //Sorting the list
 Collections.sort(list1);
  //Traversing list through the for-each loop
 for(String fruit:list1)
   System.out.println(fruit);

System.out.println("Sorting numbers...");
 //Creating a list of numbers
 List<Integer> list2=new ArrayList<Integer>();
 list2.add(21);
 list2.add(11);
 list2.add(51);
 list2.add(1);
 //Sorting the list
 Collections.sort(list2);
  //Traversing list through the for-each loop
 for(Integer number:list2)
   System.out.println(number);
```

OUTPUT

Apple

Banana

Grapes

Mango

Sorting numbers...

1

11

21

51

Example of List: Book

Let's see an example of List where we are adding the Books.

```
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {
   this.id = id;
   this.name = name;
   this.author = author;
   this.publisher = publisher;
   this.quantity = quantity;
}
}
class ListExample5 {
public static void main(String[] args) {
```

```
   //Creating list of Books
   List<Book> list=new ArrayList<Book>();

   //Creating Books
Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
Bookb2=new Book(102,"Data Communications and Networking","Forouzan","Mc Graw Hill",4);
Book b3=new Book(103,"Operating System","Galvin","Wiley",6);

   //Adding Books to list
   list.add(b1);
   list.add(b2);
   list.add(b3);

   //Traversing list
   for(Book b:list){
   System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
   }
}
}
```

OUTPUT

```
101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications and Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6
```

PRACTICE TASK

1. Write a Java program that calculates the sum of elements in an array of integers.
    - Declare and initialize an array of integers with some values.
    - Iterate through the array and calculate the sum of all elements.
    - Print the sum to the console.

2. Write a Java program that initializes and prints a 2D array of integers.
    - Declare and initialize a 2D array of integers with some values.
    - Iterate through the 2D array and print each element to the console.

3. Write a Java program that initializes and prints a jagged array of integers.
    - Declare and initialize a jagged array of integers with different lengths for each row.
    - Iterate through the jagged array and print each element to the console.

4. Write a Java program that calculates and prints the length of a given string.
    - Declare a string with some text.
    - Use the `length()` method of the String class to find the length of the string.
    - Print the length to the console.

5. Write a Java program that converts a given string to uppercase.
    - Declare a string with some text.
    - Use the `toUpperCase()` method of the String class to convert to uppercase.
    - Print the uppercase string to the console.

6. Write a Java program that checks if a given string contains a specific substring.
    - Declare a string with some text.
    - Declare a substring to search for.
    - Use the `contains()` method of String class to check if the string contains the substring.
    - Print the result (true or false) to the console.

7. Write a Java program that replaces occurrences of a specific character in a given string with another character.

- Declare a string with some text.
- Specify the character to be replaced and the character to replace it with.
- Use the `replace()` method of String class to replace occurrences of specified character.
- Print the modified string to the console.

8. Write a Java program that creates a list of strings, adds some strings to the list, and then prints each string in the list.

- Declare a list of strings using the ArrayList class.
- Add some strings to the list using the add() method.
- Iterate through the list and print each string to the console.