Day5: String, Array, CLA, Scanner

String class:

In Java, a string is an object that represents a sequence of characters. there is a predefined class in java called **String**, which belongs to **java.lang** package, this String class is used to create a string object.

We use **double quotes** to represent a string object in Java, for example

```
String message = "Welcome to Masai";
```

Here the **message** is a variable of String type which is initialized with the string object

“Welcome to Masai”. this message variable is not a primitive type variable like int, char, byte, etc.

Instead, this message variable is a reference variable of type String class.

There are two ways to create a String object in Java:

1. By string literal

   String s = “Welcome”;
2. By new keyword

   String s = new String(”Welcome”);

   .

1. By string literal:

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

String s="Welcome";

In Java, the JVM maintains a **string pool** to store all of its strings inside the memory. The string pool helps in reusing the strings.

Here, we are directly providing the value “Welcome”. Hence, the compiler first checks the string pool to see if this string already exists.

- **If this string already exists**, the new string is not created. Instead, the new reference points to the already existing string “Welcome”.
- **If the string doesn't exist**, the new string “Welcome” is created inside this string pool area.

Example:

```
String s1 = "Welcome";
String s2 = "Welcome";

if(s1 == s2)
  System.out.println("same");
else
  System.out.println("not same");


output:
  same
```

1. By new keyword:

```
String s1 = new String("Welcome");
```

Here the value of the string is not directly provided, hence a new “Welcome” string is created inside the heap memory even though the “Welcome” string object is already present inside the string pool area.

Example:

```
String s1 = new String("Welcome");
String s2 = new String("Welcome");


if(s1 == s2)
  System.out.println("same");
else
  System.out.println("not same");
```

```
output:
  not same
```

String class provides various constructors to create a new string object:

some of them are:

1. **String(byte[] bytes)** – Construct a new String by decoding the *byte array*

   example :

   .

   ```
   byte[] bytes = {100,101,102};
   String str =new String(bytes);


   output:
   def
   ```

2. **String(char[] chars)** – Allocates a new String from the given *Character array*

   example:

   ```
   char chars[] = {'M', 'a', 's', 'a', 'i'};
   String s = new String(chars);

   output:
   Masai
   ```

3. **String(StringBuffer sbuffer)** – Allocates a new string from the string in StringBuffer

   example:

   ```
   StringBuffer sbuffer = new StringBuffer("Masai");
   String s = new String(sbuffer); //Masai
   ```

4. **String(StringBuilder sbuilder)** – Allocates a new string from the string in StringBuilder

   example:

```
StringBuilder sbuilder = new StringBuilder("Masai");
String s = new String(sbuilder); //Masai
```

Java String Operations:

The Java language provides special support for the string concatenation operator (+), and for the conversion of other objects to strings.

If one operand of this (+) operator is a string then the entire expression with result a string object

Example1:

```
    String name = "Java ";
    System.out.println("Student Name is : " + name);
```

Example:

```
System.out.println("Hello" + 10 + 20 + "Welcome");

output: Hello1020Welcome
```

Java String class provides various methods to perform different operations on strings.

some of them are:

1. **int length():** Returns the number of characters in the String
2. **Char charAt(int i):** Returns the character at ith index.
3. **String substring (int i):** Return the substring from the ith index character to end
4. **String substring (int i, int j):** Returns the substring from i to j-1 index

5. **String concat(String str):** Concatenates specified string to the end of this string.
6. **int indexOf (String s):** Returns the index within the string of the first occurrence of the specified string.
7. **int indexOf (String s, int i):** Returns the index within the string of the first occurrence of the specified string, starting at the specified index.
8. **Int lastIndexOf(String s):** Returns the index within the string of the last occurrence of the specified string.
9. **boolean equals(Object otherObj):** Compares this string to the specified object.
10. **boolean equalsIgnoreCase(String anotherString):** Compares string to another string, ignoring case considerations.
11. **int compareToIgnoreCase(String anotherString):** Compares two string lexicographically, ignoring case considerations.
12. **String toUpperCase():** Converts all the characters in the String to upper case.
13. **String trim():** Returns the copy of the String, by removing whitespaces at both ends. It does not affect whitespaces in the middle.
14. **String replace (char oldChar, char newChar):** Returns new string by replacing all occurrences of *oldChar* with *newChar.*
15. char[] toCharArray(): return the char array from an string object.

Java Strings are Immutable

In Java, strings are **immutable**. This means, once we create a string, we cannot change that string.

example1:

```
String message = "Welcome";
message.concat(" user");
System.out.println(message);

output:
Welcome
```

Reason: Here, we are using the concat() method to add another string to the previous string. since string objects are immutable, if we call any method to do any modification in the string object, that method will return a new string object with modified content.

example2

```
String message = "Welcome";
String newMessage = message.concat(" user");
System.out.println(message);
System.out.println(newMessage);

output:
Welcome
Welcome user
```

Why String objects are immutable in Java?

As Java uses the concept of String literal. Suppose there are 5 reference variables, all refer to one object "Masai". If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why String objects are immutable in Java.

Note: If we want to create a mutable (modifiable) String object, we should use either StringBuffer or StringBuilder classes. both classes belong to java.lang package also.

Difference between StringBuffer and StringBuilder class

1. Most of the methods of StringBuffer is *synchronized* i.e. thread-safe. It means two threads can't call the methods of StringBuffer simultaneously. whereas most of the methods of StringBuilder is *non-synchronized* i.e. not thread-safe. It means two threads can call the methods of StringBuilder simultaneously.

   String buffers are safe for use by multiple threads. The methods are synchronized where necessary so that all the operations on any particular instance behave as if they occur in some serial order that is consistent with the order of the method calls made by each of the individual threads involved.

   Thread-safe code is code that will work even if many Threads are executing it simultaneously.

1. StringBuffer is *less efficient* than StringBuilder. whereas StringBuilder is *more efficient* than StringBuffer.

These StringBuffer and StringBuilder class provides many methods to perform modification in the same string object. example

1. public **StringBuilder append**(String anotherString): It is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
2. public StringBuilder **reverse**(): is used to reverse the string.
3. public StringBuilder replace(int startIndex, int endIndex, String str): It is used to replace the string from specified startIndex and endIndex.

Example:

```
StringBuilder message = new StringBuilder("Welcome");
message.append(" to Masai");

System.out.println(message);

output:
Welcome to Masai;
```

Java String compare:

There are three ways to compare String in Java:

1. By Using equals() Method
2. By Using == Operator
3. By compareTo() Method

1.**By Using equals() Method**

The equals() method compares the original content of the string. It compares values of string for equality. String class provides the following two methods:

- **public boolean equals(Object another)** compares this string to the specified object.
- **public boolean equalsIgnoreCase(String another)** compares this string to another string, ignoring the case.

2. **By Using == operator**

The == operator compares references not values.

example

```
String s1="Hello";
String s2="Hello";
String s3=new String(Hello");
System.out.println(s1==s2);//true (because both refer to same instance)
System.out.println(s1==s3);//false(because s3 refers to instance created in nonpool)
```

3. **String compare by compareTo() method**

   The String class compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

   Suppose s1 and s2 are two String objects. If:

   - **s1 == s2** : The method returns 0.
   - **s1 > s2** : The method returns a positive value.
   - **s1 < s2** : The method returns a negative value.

   example:

```
String s1="Sachin";
String s2="Sachin";
String s3="Ratan";
System.out.println(s1.compareTo(s2));//0
System.out.println(s1.compareTo(s3));//1(because s1>s3)
System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )
```

Array In Java:

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.**

```
int[] marks;
```

Example:

If we want to store the names of 100 people then we can create an array of the string type that can store 100 names.

`String[] names = new String[100];`

Note: In Java array is a special type of object whose class is non-existence. whenever JVM encounters [], it will create an array object in the memory.

example:

```
int[] marks = new int[10];
System.out.println(marks);// it will print the address of array object
```

• The variables in the array are ordered, and each has an index beginning from 0.

• The **size** of an array must be specified by int or short value and not long.

In the above example marks is an array of int variables where the first variable is marks[0].

Note: at the time of creating an array object, all the variables inside the array will be initialized with their default value.

```
int[] marks = new int[10];

System.out.println(marks[0]); //0
System.out.println(marks[1]);//0
System.out.println(marks[15]);//Exception: ArrayIndexOutOfBoundException
```

If we try to access an element from an array beyond its size, then it will raise a runtime exception called **ArrayOutOfBoundException.**

Note: Inside every array object, there is a non-static variable called “**length**” is there, which will represent the size of an array.

```
System.out.println(marks.lenght);//10
```

In Java there is two way to create an array :

1. using new operator

2. using curly bracket

1. **Using new operator:**

   Using this approach we declare an array first and then we initialize each element of that array

   example:

```
//declare an array
int[] marks = new int[5];


//initializing array
marks[0]=100;
marks[1]=120;
marks[2]=150;
--
```

2. Using curly bracket:

   Using this approach, we declare and initialize an array in a single statement.

   example

```
int[] marks = {100,120,150,180};
```

   Accessing Array Elements:

I Problem

Creating an integer array with some value and printing each elements from that array

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

   // create an array
   int[] marks = {40, 40, 55, 25, 55};

   // access each array elements
   System.out.println("Accessing Elements of Array:");
   System.out.println("First Element: " + marks[0]);
   System.out.println("Second Element: " + marks[1]);
   System.out.println("Third Element: " + marks[2]);
```

```
   System.out.println("Fourth Element: " + marks[3]);
   System.out.println("Fifth Element: " + marks[4]);
 }
}
```

We can also loop through each element of the array. For example,

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

   // create an array
   int[] marks = {58, 45, 52};

   // loop through the array
   // using for loop
   System.out.println("Using for Loop:");
   for(int i = 0; i < marks.length; i++) {
     System.out.println(marks[i]);
   }
 }
}
```

We can also use the for-each loop to iterate through the elements of an array. For example,

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

   // create an array
   int[] marks = {45, 42, 55};

   // loop through the array
   // using for loop
   System.out.println("Using for-each Loop:");
   for(int m : marks) {
     System.out.println(m);
   }
 }
}
```

We Problem

Compute Sum and Average of Array Elements

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

        int[] numbers = {2, -9, 0, 5, 12, -25, 22, 9, 8, 12};
```

```
        int sum = 0;
        double average;

        // access all elements using for each loop
        // add each element in sum
        for (int number: numbers) {
            sum += number;
        }

        // get the total number of elements
        int arrayLength = numbers.length;

        // calculate the average
        // convert the average from int to double
        average =  (double)sum / arrayLength;

        System.out.println("Sum = " + sum);
        System.out.println("Average = " + average);
    }
}
```

You Problem:

Write a non-static method inside class Main, which will take an initialized integer array and return the largest number from the supplied array. call this method from the main method of main class by supplying an initialized integer array and print the returned result.

> Note: As we can create an array of primitive types like (byte, short, int, float, etc.), we can also create an array of reference types also like an array of Student, Employee, Product also.

example

```
class Student
{
    public int roll_no;
    public String name;
    Student(int roll_no, String name)
    {
        this.roll_no = roll_no;
        this.name = name;
    }

    public void printDetails(){

      System.out.println("Roll is :"+roll_no+" Name is :"+name);

    }
```

```
}
```

```
class Main{

public static void main(String[] args){

Student[] students = new Student[3];

students[0] = new Student(10,"Ram");
students[1] = new Student(20,"Ramesh");
students[2] = new Student(40,"Amit");


for(Student student:students){
  student.printDetails();
}

}
}
```

Note: in the above example total 4 objects are created in the memory one for the student array object and three for the student object.

The above student array we can create by using curly bracket also.

ex:

```
Student[] students = {new Student(10,"Ram"),new Student(20,"Ramesh"),new Student(40,"Amit")};
```

Since in Java, arrays are treated as an object, the array variable will be treated as a reference variable, and the default value of any reference variable is null.

example:

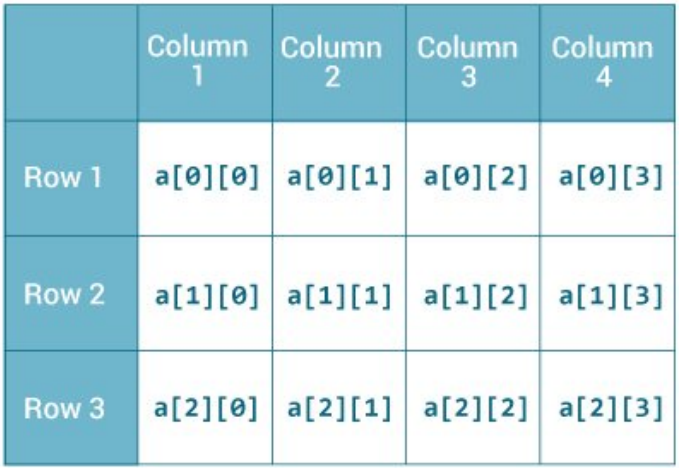
```
int[] numbers = null;
```

Multi-Dimensional array in java

In Java, we can declare a multi-dimensional array also.

A multidimensional array is an array of arrays. That is, each element of a multidimensional array is an array itself. For example,

```
int [][] matrix = new int[3][4];//3 array type variable where each variable itself is an array of 4 variable
```



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

we can initialize the 2-D array as follows :

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

	Column 1	Column 2	Column 3	Column 4
Row 1	1 a[0][0]	2 a[0][1]	3 a[0][2]	
Row 2	4 a[1][0]	5 a[1][1]	6 a[1][2]	9 a[1][3]
Row 3	7 a[2][0]			

Print all elements of 2d array Using Loop

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

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

        for (int i = 0; i < a.length; ++i) {
            for(int j = 0; j < a[i].length; ++j) {
                System.out.println(a[i][j]);
            }
        }
    }
}
```

You Problem

Print the above 2d array using for..each loop.

Command Line Argument:

The command-line arguments in Java allow the programmers to pass the arguments during the execution of a program. The users can pass the arguments during the execution by passing the command-line arguments inside the main() method.

We can use these command-line arguments as input in our Java program.

We need to pass the arguments as space-separated values. We can pass both strings and primitive data types(int, double, float, char, etc.) as command-line arguments. These arguments convert into a string array and are provided to the main() function as a string array argument.

When command-line arguments are supplied to JVM, JVM wraps these and supplies them to args[]. It can be confirmed that they are actually wrapped up in an args array by checking the length of args using args.length.

We Problem :

Get 2 numbers from the CLA and print the sum of both numbers.

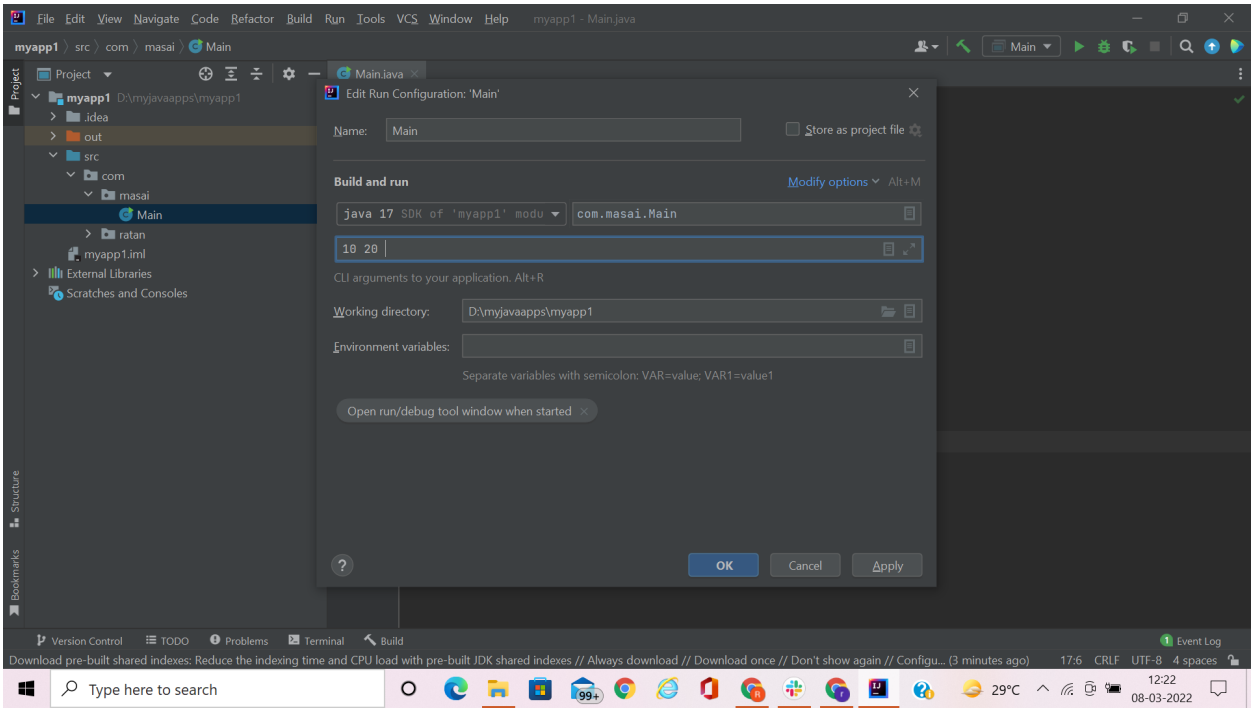
if supplied number count is less than or more than 2 number then it should print the proper message like (”Please supply only 2 numbers”)

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

        if(args.length == 2){

            int num1 = Integer.parseInt(args[0]);
            int num2 = Integer.parseInt(args[1]);

            System.out.println("The Result is "+ (num1+num2));

        }else
            System.out.println("Please supply only 2 numbers");


    }
}
```

Note: Here we have used the parseInt method of the Integer class, which is a static method, used to convert the string representation of a number to the primitive number.

To supply the command line argument with IntelliJ IDE :

Step 1: right click on the Main class → select “modify run Configuration...”



Step 2: Run this Main class once again.

Scanner class in Java:

The Scanner class is used to read the input data from different sources like input streams, users, files, etc. This Scanner class belongs to **java.util** package.

Example:

```
Scanner input = new Scanner(System.in);
```

The `System.in` parameter is used to take input from the standard input. It works just like taking inputs from the keyboard.

To use the Scanner class inside our application, we need to import this class.

```
import java.util.Scanner;
```

Java Scanner Methods to Take Input

The `Scanner` class provides various methods that allow us to read inputs of different types.

`nextInt()` reads an `int` value from the user.

`nextFloat()` reads a `float` value form the user.

`nextBoolean()` reads a `boolean` value from the user

`nextLine()` reads a line of text from the user

`next()` reads a word from the user

`nextByte()` reads a `byte` value from the user

`nextDouble()` reads a `doubl`e value from the user

`nextShort()` reads a `short` value from the user.

`nextLong()` reads a `long` value from the user

Example 1: Read a Line of Text Using Scanner

```
import java.util.Scanner;

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

    // creates an object of Scanner
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter your name: ");

    // takes input from the keyboard
    String name = scanner.nextLine();

    // prints the name
    System.out.println("My name is " + name);


  }
}
```

Reading an integer from the user input:

```
import java.util.Scanner;

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

```
    // creates a Scanner object
    Scanner scanner = new Scanner(System.in);

    System.out.println("Enter number 1: ")
    int num1 = scanner.nextInt();

    System.out.println("Enter number 2: ")
    int num2 = scanner.nextInt();

    System.out.println("Result is : " + (num1+num2));

  }
}
```

Difference between next and nextLine method:

example next() method:

```
import java.util.Scanner;

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

    // creates an object of Scanner
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter your name: ");

    // reads the entire word
    String value = scanner.next();
    System.out.println("Using next(): " + value);

    scanner.close();
  }
}

Output:
Enter your name: Ram Kumar
Using next(): Ram
```

In the above example, we have used the `next()` method to read a string from the user.

Here, we have provided the full name. However, the `next()` method only reads the first name.

This is because the `next()` method reads input up to the **whitespace** character. Once the **whitespace** is encountered, it returns the string (excluding the whitespace).

example nextLine() method:

```
import java.util.Scanner;

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

    // creates an object of Scanner
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter your name: ");

    // reads the entire word
    String value = scanner.nextLine();
    System.out.println("Using next(): " + value);

    scanner.close();
  }
}

Output:
Enter your name: Ram Kumar
Using nextLine(): Ram Kumar
```

Unlike `next()` the `nextLine()` method reads the entire line of input including spaces. The method is terminated when it encounters a next line character, `\n`

.

References:

https://www.javatpoint.com/

https://www.geeksforgeeks.org/

https://www.programiz.com/java-programming/