Problem Statement

Write a JAVA program which prints the following information: -

NAME	email-id	EMP-CODE	PHONE
Anil	anil@cse	e10101	25764728
Bimal	bimal@cse	e10102	25764728

Each entry should be on a separate line.

Algorithm

Algorithm emp
Input: N.A.
Output: Display the Print.
Step 1: Start
Step 2: Print ()
Step 3: Stop

Source Code

```
class emp                       // Define a Java class named "emp"
{
        public static void main(String[] args)          // The main method is the entry point of the program
        {
                // Print column headers for employee information
                System.out.println("\n\tNAME\temail-id\tEMP-CODE\tPHONE\n");

                // Print the details of the first employee
                System.out.println("\tAnil\tanil@cse\te10101\t\t25764728");

                // Print the details of the second employee
                System.out.println("\tBimal\tbimal@cse\te10102\t\t25764728");
        }
}
```

Result

C:\JAVA\College>*javac A1.java*
C:\JAVA\College>*java emp*

NAME	*email-id*	*EMP-CODE*	*PHONE*
Anil	*anil@cse*	*e10101*	*25764728*
Bimal	*bimal@cse*	*e10102*	*25764728*

Discussion

Here, a ***class*** is a blueprint or a template for creating objects. It defines the structure and behaviour of objects.
The main method is a special method in Java, and it serves as the entry point for the execution of a Java program.
public: This modifier means that the main method is accessible from anywhere.
static: This keyword indicates that the method belongs to the class itself, rather than to an instance of the class. This is necessary because the program starts execution before any objects are created.
void: This specifies that the main method does not return any value.
String[] args: This parameter allows the program to accept command-line arguments as an array of strings. Command-line arguments can be used to provide input to the program when it starts.

Problem Statement

Write a JAVA program that prints the following line on the screen along with quotes.
"Can we print '\' with System.out.println() statement?"

Algorithm

Algorithm quote
Input: N.A.
Output: Display the Print.
Step 1: Start
Step 2: Print ()
Step 3: Stop

Source Code

```
class quote                    // Define a Java class named "emp"
{
        public static void main(String[] args)          // The main method is the entry point of the program
        {
                // Print a statement with double quotes and a backslash
                System.out.print("\"Can we print '\\' with System.out.println() statement?\"");
        }
}
```

Result

C:\JAVA\College>*javac A2.java*
C:\JAVA\College>*java quote*
"Can we print '\' with System.out.println() statement?"

Discussion

Here, a ***class*** is a blueprint or a template for creating objects. It defines the structure and behaviour of objects.
The main method is a special method in Java, and it serves as the entry point for the execution of a Java program.
public: This modifier means that the main method is accessible from anywhere.
static: This keyword indicates that the method belongs to the class itself, rather than to an instance of the class. This is necessary because the program starts execution before any objects are created.
void: This specifies that the main method does not return any value.
String[] args: This parameter allows the program to accept command-line arguments as an array of strings. Command-line arguments can be used to provide input to the program when it starts.

System is a class in the *java.lang* package that provides access to system-related functions and resources.
System.out is an instance of the *PrintStream* class and represents the standard output stream, typically the console or terminal.

Problem Statement

Write a program in java to check whether a given number (from user) is odd positive or not.

Num1=-2 Output:- "NO"
Num2=4 Output:- "NO"
Num3=0 Output:- "NO"
Num4=-11 Output:-"NO"
Num5=11 Output:- "ODD POSITIVE"

Algorithm

Algorithm OddPosCheck
Input: An integer number, whether positive or negative, is taken as the input from the user.
Output: Checks and prints whether the given number is odd positive or not.
Step 1: Start
Step 2: If (num > 0 and num%2 ≠ 0) then *//num is the number taken as the input from user*
Step 2.1: Print "ODD POSITIVE"
Step 3: Else
Step 3.1: Print "NO"
Step 4: End If
Step 5: Stop

Source Code

```
import java.util.Scanner;
class OddPosCheck
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);		// Create a Scanner object to read user input
		System.out.print("Enter a number: ");		// Prompt the user to enter a number
		int num = input.nextInt();			// Read an integer from the user's input
		if (num > 0 && num % 2 != 0)			// Check if the entered number is both positive and odd
		{
			System.out.println("ODD POSITIVE");		// If it is, print "ODD POSITIVE"
		}
		else
		{
			System.out.println("NO");			// If not, print "NO"
		}
	}
}
```

Result

C:\JAVA\College>*javac A3.java*
C:\JAVA\College>*java OddPosCheck*
Enter a number: -2
NO

C:\JAVA\College>*java OddPosCheck*
Enter a number: 4

NO

C:\JAVA\College>*java OddPosCheck*
Enter a number: 0
NO

C:\JAVA\College>*java OddPosCheck*
Enter a number: -11
NO

C:\JAVA\College>*java OddPosCheck*
Enter a number: 11
ODD POSITIVE

Discussion

Here,

- The ***java.util.Scanner*** class is a part of the Java Standard Library and is used for reading input from various sources, including the keyboard (System.in), files, or strings. It provides a convenient way to parse and process different types of data.
- ***System.in*** is an input stream connected to the standard input, which is typically the keyboard. It allows to read data entered by the user from the console.
- By creating a *Scanner* object with *System.in* as an argument, it initializes the Scanner to read input from the console.
- ***nextInt()*** is a method provided by the Scanner class, which reads the next token of input as an integer. It waits for the user to input an integer and press the Enter key.
- The entered integer is then stored in the variable '*num*' for further use in your program.

- This Java program, named "OddPosCheck," is designed to determine whether a user-entered integer is both positive and odd.
- It begins by importing the Scanner class to enable user input processing.
- The program prompts the user to input a number, which is then stored in the 'num' variable.
- It checks if the entered number is greater than zero (positive) and also has a remainder when divided by 2, which indicates it is an odd number. If both conditions are met, it prints "ODD POSITIVE."
- If the number is not positive and odd, it prints "NO." This program provides a simple way to identify positive odd numbers from user input.

Problem Statement

Write a program in java to print values of Fibonacci series up to 20.

Algorithm

Algorithm Fibonacci
Input: N.A.
Output: Prints the Fibonacci series up to 20.
Step 1: Start
Step 2: n ← 20, prev ← 0, current ← 1
Step 3: Print "Fibonacci series up to n terms:"
Step 4: For (i=1 to n)do
  Step 4.1: Print "prev "
  Step 4.2: next ← prev + current
  Step 4.3: prev ← current
  Step 4.4: current ← next
Step 5: End For
Step 6: Stop

Source Code

```
class Fibonacci
{
        public static void main(String[] args)
        {
                int n = 20, prev = 0, current = 1;      // Define & Initialize the maximum value for terms & the first two
                                                        // terms of the Fibonacci sequence
                System.out.println("Fibonacci Series up to " + n + ":");
                while (prev <= n)               // Continue looping until the current term is less than or equal to 'n'
                {
                        System.out.print(prev + " ");           // Print the current term
                        int next = prev + current;              // Calculate the next term in the sequence
                        prev = current;                         // Update 'prev' to the current value of 'current'
                        current = next;                         // Update 'current' to the calculated 'next' value
                }
        }
}
```

Result

C:\JAVA\College>*javac A4.java*
C:\JAVA\College>*java Fibonacci*
Fibonacci Series up to 20 terms:
0 1 1 2 3 5 8 13

Discussion

Here, *java.util.Scanner* class is used to read user input and it's found in the *java.util* package. The Scanner class has a number of methods for reading different types of inputs, including strings, integers, floats and characters. Also, the logic of the program is based on a "while loop" that iterates up to 20 and picks a number that resides in the Fibonacci series and prints the number.

Problem Statement

Write a program in java to print the following pattern (no. of rows is given by the user):-

1
121
12321
1234321

Algorithm

Algorithm NumberPattern
Input: Number of rows is taken as the input from the user.
Output: Prints the pattern in the given sequence.
Step 1: Start
Step 2: For (i = 1 to row) do *//row is the number of rows given by the user.*
Step 2.1: For (j = 1 to j = i) do
Step 2.1.1: Print(j)
Step 2.2: End For
Step 2.3: For (j = i-1 to j = 1) do
Step 2.3.1: Print(j)
Step 2.4: End For
Step 2.5: Print(New Line)
Step 3: End For
Step 4: Stop

Source Code

```
import java.util.Scanner;
class NumberPattern
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);		// Create a Scanner object to read user input
		System.out.print("Enter the number of rows: ");		// Prompt the user to enter the number of rows
		int nr = input.nextInt();
		if (nr <= 0) {
			System.out.println("Please enter a positive number greater than zero.");
		} else {
			for (int i = 1; i <= nr; i++)		// Loop to iterate through each row
			{
				for (int j = 1; j <= i; j++)		// Loop to print numbers from 1 to i in increasing order
				{
					System.out.print(j);
				}
				for (int j = i - 1; j >= 1; j--)	// Loop to print numbers from i-1 down to 1 in decreasing order
				{
					System.out.print(j);
				}
				System.out.println();		// Move to the next line to start a new row
			}
		}
	}
}
```

Result

C:\JAVA\College>*javac A5.java*
C:\JAVA\College>*java NumberPattern*
Enter the number of rows: 4
1
121
12321
1234321

C:\JAVA\College>*java NumberPattern*
Enter the number of rows: 8
1
121
12321
1234321
123454321
12345654321
1234567654321
123456787654321

C:\JAVA\College>*java NumberPattern*
Enter the number of rows: -1

Please enter a positive number greater than zero.

C:\JAVA\College>*java NumberPattern*
Enter the number of rows: 0

Please enter a positive number greater than zero.

Discussion

Here, *java.util.Scanner* class is used to read user input and it's found in the *java.util* package. The Scanner class has a number of methods for reading different types of inputs, including strings, integers, floats and characters.

- This Java program, named "*NumberPattern*," generates a numerical pattern based on user input for the number of rows.
- It utilizes the Scanner class to accept user input for the number of rows they want in the pattern.
- The program then employs *nested for loops* to create a pattern, with each row containing a series of numbers that increment from 1 to the row number and then decrease back to 1.
- The outer loop controls the number of rows in the pattern, and the inner loops handle the printing of numbers in both ascending and descending order.
- After each row is printed, a newline character is used to move to the next line, creating a visually appealing number pattern. This program provides a way to generate customizable numerical patterns based on user-defined row counts.

Problem Statement

Write a program in java to check if a given number (from user) is binary or not.

Num1=1001 Output: - "YES"

Num2=1002 Output: - "NO"

Algorithm

Algorithm BinaryCheck

Input: A number 'n' taken as the input from the user.

Output: Checks whether the number is a binary (only includes 0 and 1 as the digit) or not and prints output statement accordingly.

Step 1: Start
Step 2: flag ← 0
Step 3: While (num ≠ 0) do
  Step 3.1: rem ← (n%10)
  Step 3.2: If (rem ≠ 0 and rem ≠ 1) then
    Step 3.2.1: flag ← 1
    Step 3.2.2: Break
  Step 3.3: End If
  Step 3.4: num ← (num/10)
Step 4: End While
Step 5: If (flag = 0) then
  Step 5.1: Print "Yes: Binary"
Step 6: Else
  Step 6.1: Print "No: Not Binary"
Step 7: End If
Step 8: Stop

Source Code

```
import java.util.Scanner;
class BinaryCheck
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);		// Create a Scanner object to read user input
		System.out.print("Enter an integer: ");		// Prompt the user to enter an integer
		int num = sc.nextInt();
		boolean isBinary = true;		// Initialize a boolean variable to track if the number is binary
		while (num != 0)		// Loop to check each digit of the entered number
		{
			int a = num % 10;
			if (a != 0 && a != 1)		// Check if the digit is not 0 or 1, indicating a non-binary digit
			{
				isBinary = false;
				break;
			}
			num = num / 10;		// Remove the last digit to continue checking the remaining digits
		}
		if (isBinary)		// Check the result and print "YES" if it's binary, or "NO" if it's not
		{
			System.out.println("YES");
		}
```

```
                else
                {
                        System.out.println("NO");
                }
        }
}
```

Result

C:\JAVA\College>*javac A6.java*

C:\JAVA\College>*java BinaryCheck*
Enter an integer: 1001
YES

C:\JAVA\College>*java BinaryCheck*
Enter an integer: 1002
NO

C:\JAVA\College>*java BinaryCheck*
Enter an integer: 0001
YES

C:\JAVA\College>*java BinaryCheck*
Enter an integer: -100
NO

C:\JAVA\College>*java BinaryCheck*
Enter an integer: 2501
NO

C:\JAVA\College>*java BinaryCheck*
Enter an integer: 0000
YES

Discussion

Here, *java.util.Scanner* class is used to read user input and it's found in the *java.util* package. The Scanner class has a number of methods for reading different types of inputs, including strings, integers, floats and characters.

- The Java program "*BinaryCheck*" is designed to determine whether a user-input integer is a binary number.
- It starts by using the Scanner class to receive input from the user, specifically requesting an integer to be checked.
- The program then utilizes a while loop to examine each digit of the entered number, one at a time, to determine if they are either 0 or 1.
- If the program encounters a digit that is not 0 or 1, it sets a ***boolean variable*** '*isBinary*' to false and exits the loop, indicating that the number is not binary.
- Finally, it prints "YES" if '*isBinary*' is still true, meaning all digits were either 0 or 1, and "NO" if it's false, indicating the number contains non-binary digits. This program provides a straightforward way to validate if an input number is in binary form.

Problem Statement

Write a Java program to insert an element (specific position) into an array.

Algorithm

Algorithm ArrayInsert
Input: A number which is to be insert at a specific position in an array.
Output: After insertion the number display the full array. If user give a wrong position then print a warning –"Invalid Position".
Step 1: Start
Step 2: If (position < 0 and position > arraysize) then
Step 2.1: Print "Invalid Position"
Step 3: Else
Step 3.1: For (i=size to i > position) do
Step 3.1.1: array[i] ← array[i-1]
Step 3.2: End For
Step 3.3: array[position] ← Element to Insert
Step 3.4: size ← size + 1
Step 3.5: For (i=0 to i<size) do
Step 3.5.1: Print(array[i])
Step 3.6: End For
Step 4: Stop

Source Code

```
import java.util.Scanner;
class ArrayInsert
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);				// Create a Scanner object to read user input
		System.out.print("Enter the size of the array: ");		//Prompt the user to enter the size of array
		int size = sc.nextInt();					// Read an integer from the user's input
		int[] array = new int[size + 1];				// Increase array size by 1
		System.out.print("Enter the elements of the array: ");	// Input the elements of the array
		for (int i = 0; i < size; i++)
		{
			array[i] = sc.nextInt();
		}
		System.out.print("Enter the element to insert: ");		// Input the element to insert
		int elementToInsert = sc.nextInt();

	// Input the position to insert (0-based index)
		System.out.print("Enter the position to insert (0-based index): ");
		int position = sc.nextInt();
		if (position < 0 || position > size)
		{
			System.out.println("Invalid position. Position should be between 0 and " + size);
		}
		else
		{
			for (int i = size; i > position; i--)	// Shift elements to the right to make space for the new element
			{
```

```
                        array[i] = array[i - 1];
                    }
                    array[position] = elementToInsert;          // Insert the new element at the specified position
                    size++;                                     // Increase the size of the array
                    System.out.println("Array after insertion:");
                    for (int i = 0; i < size; i++)
                    {
                        System.out.print(array[i] + " ");
                    }
                }
            }
}
```

Result

C:\JAVA\College>*javac A7.java*

C:\JAVA\College>*java ArrayInsert*
Enter the size of the array: 5
Enter the elements of the array: 2 1 6 7 8
Enter the element to insert: -5
Enter the position to insert (0-based index): 3
Array after insertion:
2 1 6 -5 7 8

C:\JAVA\College>*java ArrayInsert*
Enter the size of the array: 6
Enter the elements of the array: -8 -6 8 0 -7 -3
Enter the element to insert: 0
Enter the position to insert (0-based index): 5
Array after insertion:
-8 -6 8 0 -7 0 -3

C:\JAVA\College>*java ArrayInsert*
Enter the size of the array: 5
Enter the elements of the array: 9 -14 0 -5 35
Enter the element to insert: 87
Enter the position to insert (0-based index): 2
Array after insertion:
9 -14 87 0 -5 35

C:\JAVA\College>*java ArrayInsert*
Enter the size of the array: 4
Enter the elements of the array: 6 0 8 3
Enter the element to insert: 5
Enter the position to insert (0-based index): 6
Invalid position. Position should be between 0 and 4

Discussion

Here, *java.util.Scanner* class is used to read user input and it's found in the *java.util* package. The Scanner class has a number of methods for reading different types of inputs, including strings, integers, floats and characters.

- **int[] array = new int[size + 1]:** This line of code declares an integer array named array. It allocates memory for the array based on the size provided by the user plus one. The addition of one extra element is for accommodating the element to be inserted at a specified position. This dynamic array resizing ensures that there is enough space to insert a new element without losing any existing data.

- **for (int i = 0; i < size; i++) {array[i] = sc.nextInt();}:** This loop is used to populate the elements of the array by reading integers from the user's input. It iterates from 0 to size - 1, where size represents the size of the array provided by the user. Each element is read using *sc.nextInt()* and assigned to the corresponding index in the array.

- The program allows the user to input the size of an array, followed by the array elements. It then prompts the user to enter an element and a position for insertion. If the position is valid (between 0 and the current size of the array), it inserts the element at the specified position and shifts the existing elements to accommodate the new one.

- *One important feature is the dynamic array resizing:* it increases the array size by one to accommodate the new element, allowing for flexible array expansion.

- The program includes error handling for invalid positions, ensuring that the user is prompted to enter a valid position within the array's bounds.

- After insertion, the program prints the updated array, which is useful for verifying the correctness of the insertion operation.

- Overall, this program provides a simple and functional way to insert an element into an array at a specified position while ensuring data integrity and handling potential errors.

Problem Statement

Write a Java program to remove a specific element from an array.

Algorithm

Algorithm RemoveElement

Input: A number which is to be remove from an array.

Output: After removing the number display the full array. If user give a wrong number then print a warning –"Element not found".

Step 1: Start
Step 2: Index ← -1
Step 3: For (i = 0 to i < size) do
  Step 3.1: if (array[i] = Element To Remove) then
    Step 3.1.1: Index ← i
    Step 3.1.2: Break
  Step 3.2: End If
Step 4: End For
Step 5: If (Index = -1) then
  Step 5.1: Print "Element not found in the array."
Step 6: Else
  Step 6.1: NewArray[] ← size - 1
  Step 6.2: For (i = 0 to i < Index) do
    Step 6.2.2: NewArray[i] ← array[i]
  Step 6.3: End For
  Step 6.4: For (i = Index + 1 to i < size) do
    Step 6.4.1: NewArray[i - 1] ← array[i]
  Step 6.5: End For
  Step 6.6: For (i = 0 to i < size - 1)
    Step 6.6.1: Print (NewArray[i])
  Step 6.7: End For
Step 7: Stop

Source Code

```
import java.util.Scanner;

class RemoveElement
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);
		System.out.print("Enter the size of the array: ");		// Input the size of the array
		int size = input.nextInt();
		int[] array = new int[size];				// Create an array of the specified size
		System.out.print("Enter the elements of the array: ");	// Input array elements
		for (int i = 0; i < size; i++)
		{
			array[i] = input.nextInt();
		}
		System.out.print("Enter the element to remove: ");	// Input the element to be removed
		int elementToRemove = input.nextInt();
		int indexToRemove = -1;
		for (int i = 0; i < size; i++)
		{
```

```
                if (array[i] == elementToRemove)
                {
                        indexToRemove = i;
                        break;
                }
        }

        if (indexToRemove == -1)
        {
                System.out.println("Element not found in the array.");
        }
        else
        {
                int[] newArray = new int[size - 1];             // Create a new array with one less element
                for (int i = 0; i < indexToRemove; i++)         // Copy elements before the index to be removed
                {
                        newArray[i] = array[i];
                }
                for (int i = indexToRemove + 1; i < size; i++)          // Copy elements after the index to be removed
                {
                        newArray[i - 1] = array[i];
                }

                System.out.println("Array after removing the element:");      // Display the modified array
                for (int i = 0; i < size - 1; i++)
                {
                        System.out.print(newArray[i] + " ");
                }
        }
    }
}
```

Result

C:\JAVA\College>*javac A8.java*

C:\JAVA\College>*java RemoveElement*
Enter the size of the array: 5
Enter the elements of the array: -5 6 -8 -2 0
Enter the element to remove: 0
Array after removing the element:
-5 6 -8 -2

C:\JAVA\College>*java RemoveElement*
Enter the size of the array: 8
Enter the elements of the array: 5 6 8 2 5 0 -8 4
Enter the element to remove: 5
Array after removing the element:
6 8 2 5 0 -8 4

C:\JAVA\College>*java RemoveElement*
Enter the size of the array: 5
Enter the elements of the array: 9 6 0 4 6
Enter the element to remove: -1
Element not found in the array.

C:\JAVA\College>*java RemoveElement*
Enter the size of the array: 5
Enter the elements of the array: 9 -4 5 0 7
Enter the element to remove: 5
Array after removing the element:
9 -4 0 7

Discussion

Here, *java.util.Scanner* class is used to read user input and it's found in the *java.util* package. The Scanner class has a number of methods for reading different types of inputs, including strings, integers, floats and characters.

- This Java program allows users to input the size of an array, the array elements, and an element they want to remove from the array.
- It first searches for the index of the element to remove in the input array using a linear search. If the element is not found, it prints a message indicating that the element is not in the array.
- If the element is found, it creates a new array, *newArray*, with one less element than the original array.
- The program then copies the elements before and after the index of the element to be removed into the *newArray*, effectively removing the specified element.
- Finally, it displays the modified array, showing the result after removing the specified element. This program provides a simple and functional way to remove an element from an array while preserving the order of the remaining elements.

Problem Statement

Write a Java program to find all pairs of elements in an array whose sum is equal to a specified number.

Algorithm

Algorithm PairSum

Input: Array size, Array elements and a number whose pair of sum to be find.

Output: Display all pair of elements in ana array whose sum is equal to specified number.

Step 1: Start
Step 2: flag ← 0, k ← 0, count←0
Step 3: For (i=0 to i<size) do
    Step 3.1: For (j=0 to j<size) do
        Step 3.1.1: If (i≠j) then
            Step 3.1.1.1: If (array[i] + array[j] = TargetNumber) then
                Step 3.1.1.1.1: ResultArray[k+1] ← array[i]
                Step 3.1.1.1.2: flag ← 1
                Step 3.1.1.1.3: For (p=0 to p<k) do
                    Step 3.1.1.1.3.1: if (ResultArray[p] = array[i]) then
                        Step 3.1.1.1.3.1.1: count ← count + 1
                    Step 3.1.1.1.3.2: End If
                Step 3.1.1.1.4: End For
                Step 3.1.1.1.5: If (count = 1) then
                    Step 3.1.1.1.5.1: Print (array[i], array[j])
                Step 3.1.1.1.6: End If
            Step 3.1.1.2: End If
        Step 3.1.2: End If
    Step 3.2: End For
Step 4: End For
Step 5: If (flag = 0) then
    Step 5.1: Print (TargetNumber)
Step 6: End If
Step 7: Stop

Source Code

```
import java.util.Scanner;
class PairSum
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);		// Input the number of elements in the array
		System.out.print("Enter the size of the array: ");
		int size = input.nextInt();
		int array[] = new int[size];
		int resultArray[] = new int[size * 2];
		int flag = 0, k = 0, count = 0;
		System.out.print("Enter the array elements: ");	// Input the array elements
		for (int i = 0; i < size; i++)
		{
			array[i] = input.nextInt();
		}						// Input the specific element to find pairs for
		System.out.print("Enter a number to find its summation pairs: ");
		int targetNumber = input.nextInt();
		System.out.println("Pairs with sum " + targetNumber+ ":");
		for (int i = 0; i < size; i++)			// Find and print pairs with the specified sum
		{
			for (int j = 0; j < size; j++)
			{
				if (i != j)
				{
					if (array[i] + array[j] == targetNumber)
					{
						resultArray[k++] = array[i];
						flag = 1;
						count = 0;

				// Count the occurrences of the same element in the resultArray
						for (int p = 0; p < k; p++)
						{
							if (resultArray[p] == array[i])
							{
								count++;
							}
						}
						if (count == 1)		// Print the pair if it's not a duplicate
						{
							System.out.print("(" + array[i] + "," + array[j] + ")  ");
						}
					}
				}
			}
		}
		System.out.println();
		if (flag == 0)		// Check if no pair was found
		{
			System.out.println("No summation pair is available for " + targetNumber);
		}
	}
}
```

Result

C:\JAVA\College>*javac A9.java*

C:\JAVA\College>*java PairSum*
Enter the size of the array: 5
Enter the elements of the array: 2 3 8 9 5
Enter a number to find its summation pairs: 5
Pairs with sum 5:
(2, 3) (3, 2)

C:\JAVA\College>*java PairSum*
Enter the size of the array: 5
Enter the elements of the array: -7 2 -5 0 6
Enter a number to find its summation pairs: -5
Pairs with sum -5:
(-7, 2) (2, -7) (-5, 0) (0, -5)

C:\JAVA\College>*java PairSum*
Enter the size of the array: 6
Enter the elements of the array: 9 4 8 6 7 2
Enter a number to find its summation pairs: 10
Pairs with sum 10:
(4, 6) (6, 4) (8, 2) (2, 8)

C:\JAVA\College>*java PairSum*
Enter the size of the array: 5
Enter the elements of the array: -5 0 6 7 -4
Enter a number to find its summation pairs: 50
Pairs with sum 50:
No summation pair is available for 50

C:\JAVA\College>*java PairSum*
Enter the size of the array: 10
Enter the array elements: 2 0 4 6 5 2 1 8 6 7
Enter a number to find its summation pairs: 4
Pairs with sum 4:
(2,2) (0,4) (4,0)

Discussion

Here, *java.util.Scanner* class is used to read user input and it's found in the *java.util* package. The Scanner class has a number of methods for reading different types of inputs, including strings, integers, floats and characters.

- This Java program takes user input for an array of integers and a specific target number.
- It then iterates through the array using nested loops to find pairs of elements that add up to the target number.
- The program avoids duplicate pairs and stores the result in a separate array.
- It uses flags and counters to track and print unique pairs, ensuring that no duplicates are displayed.
- If no pairs are found, it provides a user-friendly message indicating that there are no pairs in the array that meet the specified criteria.

Problem Statement

Write a Java program to remove the duplicate elements of a given array and return the new length of that array.

Algorithm

Algorithm RemoveDuplicates

Input: Insert array elements from user.

Output: After removing the duplicate numbers display the full array and the new size of the array. If no duplicates found in the array, then display – *"No duplicates found in the array."*

Step 1: Start
Step 2: NewSize ← 1
Step 3: For (i=1 to i<size) do
    Step 3.1: flag ← 0
    Step 3.2: For (j=0 to j<size) do
        Step 3.2.1: If (array[i] = array[j]) then
            Step 3.2.1.1: flag ← 1
            Step 3.2.1.2: Break
        Step 3.2.2: End If
    Step 3.3: End For
    Step 3.4: If (flag = 0) then
        Step 3.4.1: array[NewSize] ← array[i]
        Step 3.4.2: NewSize ← NewSize + 1
    Step 3.5: End If
Step 4: End For
Step 5: If (NewSize = size) then
    Step 5.1: Print "No duplicates found in the array"
Step 6: Else
    Step 6.1: Print (NewSize)
    Step 6.2: For (i=0 to i<NewSize) do
        Step 6.2.1: Print (array[i])
    Step 6.3: End For
Step 7: End If
Step 8: Stop

Source Code

```
import java.util.Scanner;

class RemoveDuplicates
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);
		System.out.print("Enter the size of the array: ");		// Input the size of the array
		int size = input.nextInt();
		int[] array = new int[size];		// Create an array of the specified size
		System.out.print("Enter the elements of the array: ");		// Input array elements
		for (int i = 0; i < size; i++)
		{
			array[i] = input.nextInt();
		}
		int newSize = 1; // Initialize the new size to 1 (at least one element is unique)
		for (int i = 1; i < size; i++)		// Remove duplicates and get the new length
		{
			boolean isDuplicate = false;
			for (int j = 0; j < newSize; j++)		// Check if the current element is a duplicate
			{
				if (array[i] == array[j])
				{
					isDuplicate = true;
					break;
				}
			}
			if (!isDuplicate)		// If it's not a duplicate, add it to the unique elements
			{
				array[newSize] = array[i];
				newSize++;
			}
		}
		if (newSize == size)
		{
			System.out.println("No duplicates found in the array.");
		}
		else
		{
			System.out.println("New length of the array: " + newSize);	// Display the new length and the
			System.out.println("Array after removing duplicates:");		modified array
			for (int i = 0; i < newSize; i++)
			{
				System.out.print(array[i] + " ");
			}
		}
	}
}
```

Result

C:\JAVA\College>*javac A10.java*

C:\JAVA\College>*java RemoveDuplicates*
Enter the size of the array: 5
Enter the elements of the array: -5 6 4 6 8
New length of the array: 4
Array after removing duplicates:
-5 6 4 8

C:\JAVA\College>*java RemoveDuplicates*
Enter the size of the array: 5
Enter the elements of the array: 8 6 4 8 9
New length of the array: 4
Array after removing duplicates:
8 6 4 9

C:\JAVA\College>*java RemoveDuplicates*
Enter the size of the array: 8
Enter the elements of the array: 4 6 4 -8 -2 -8 6 0
New length of the array: 5
Array after removing duplicates:
4 6 -8 -2 0

C:\JAVA\College>*java RemoveDuplicates*
Enter the size of the array: 6
Enter the elements of the array: 8 6 4 2 9 7
No duplicates found in the array.

Discussion

- The Java program is designed to remove duplicate elements from an array entered by the user.
- It begins by prompting the user to input the size of the array and the array elements.
- The program then iterates through the array, using a *nested loop* to check if the current element is a duplicate by comparing it to previously encountered elements.
- Non-duplicate elements are added to a modified version of the array, and the program keeps track of the new size.
- After processing the array, the program displays the new length of the array (which accounts for duplicates removed) and the modified array containing only unique elements. This program effectively demonstrates how to remove duplicates from an array while preserving the order of the remaining elements.

Problem Statement

Write a Java program to find the length of the longest consecutive elements sequence from a given unsorted array of integers.
Input: - Sample array: [49, 1, 3, 200, 2, 4, 70, 5]
The longest consecutive elements sequence is: [1, 2, 3, 4, 5], therefore the program will return its length
Output: - 5.

Algorithm

Algorithm LongestConsecutiveStreak
Input: Array size and elements from user.
Output: Display the length of the longest consecutive elements sequence from the given array.
Step 1: Start
Step 2: For (i=0 to i<size-1) do
Step 2.1: flag ← 0
Step 2.2: For (j=0 to j<size-i-1) do
Step 2.2.1: If (array[j] > array[j+1]) then
Step 2.2.1.1: Temp ← array[j]
Step 2.2.1.2: array[j] ← array[j+1]
Step 2.2.1.3: array[j+1] ← Temp
Step 2.2.1.4: flag ← flag + 1
Step 2.2.2: End If
Step 2.3: End For
Step 2.4: If (flag = 0) then
Step 2.4.1: Break
Step 2.5: End If
Step 3: End For
Step 4: CurrentStreak ← 1, LongestStreak ← 1
Step 5: For (i=0 to i<size-1) do
Step 5.1: If (array[i] +1 = array[i+1]) then
Step 5.1.1: CurrentStreak ← CurrentStreak + 1
Step 5.2: Else
Step 5.2.1: If (LongestStreak < CurrentStreak) then
Step 5.2.1.1: LongestStreak ← CurrentStreak
Step 5.2.1.2: CurrentStreak ← 1
Step 5.2.2: End If
Step 5.3: End If
Step 6: End For
Step 7: If (LongestStreak > CurrentStreak) then
Step 7.1: Print (LongestStreak)
Step 8: Else
Step 8.1: Print (CurrentStreak)
Step 9: End If
Step 10: Stop

Source Code

```
import java.util.Scanner;
class LongestConsecutiveStreak
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);			// create an instance of Scanner class
		System.out.print("Enter the required size of the array: ");
		int size = input.nextInt();				// take user input for array size
		if (size <= 0)
		{
			System.out.print("Invalid input! Array size can't be less than or equal to zero.");
		}
		else
		{
			int[] array = new int[size];
			System.out.print("Enter the elements of the array: ");
			for (int i = 0; i < size; i++)
			{
				array[i] = input.nextInt();
			}

		// Sort the array elements using the bubble sort technique
			for (int i = 0; i < size - 1; i++)
			{
				int flag = 0;
				for (int j = 0; j < size - i - 1; j++)
				{
					if (array[j] > array[j + 1])
					{
						int temp = array[j];
						array[j] = array[j + 1];
						array[j + 1] = temp;
						flag++;
					}
				}
				if (flag == 0)
				break;
			}
		int currentStreak = 1;
		int longestStreak = 1;
		for (int i = 0; i < size - 1; i++)
			{
			if (array[i] + 1 == array[i + 1])
				{
					currentStreak++;		// counts the longest consecutive streak
				}
				else
				{
					if (longestStreak < currentStreak)
					longestStreak = currentStreak;	// updates the previous longest consecutive streak
					currentStreak = 1;
				}
			}
			System.out.print("\nThe length of the longest consecutive streak present in the array is: ");
			if (longestStreak > currentStreak)
```

```
                    System.out.print(longestStreak);
                else
                    System.out.print(currentStreak);
            }
        }
}
```

Result

C:\JAVA\College>*javac A11.java*

C:\JAVA\College>*java LongestConsecutiveStreak*
Enter the required size of the array: 8
Enter the elements of the array: 49 1 3 200 2 4 70 5

The length of the longest consecutive streak present in the array is: 5

C:\JAVA\College>*java LongestConsecutiveStreak*
Enter the required size of the array: 10
Enter the elements of the array: 25 75 26 89 27 4 29 30 23 45

The length of the longest consecutive streak present in the array is: 3

C:\JAVA\College>*java LongestConsecutiveStreak*
Enter the required size of the array: 6
Enter the elements of the array: -5 -4 0 2 -3 7

The length of the longest consecutive streak present in the array is: 3

Discussion

- This Java program is designed to find the length of the longest consecutive streak of numbers in an array entered by the user.
- It starts by prompting the user to input the size of the array and the array elements. It includes input validation to ensure the array size is not less than or equal to zero.
- The program then sorts the array elements using *the bubble sort technique*, which helps arrange the elements in ascending order.
- After sorting, it iterates through the sorted array to calculate the longest consecutive streak of numbers. It keeps track of the *current streak* and updates the *longest streak* whenever a longer streak is encountered.
- Finally, the program displays the length of the longest consecutive streak found in the array. This program effectively demonstrates how to find the longest consecutive streak in an array after sorting it, providing a clear solution to the problem.

Problem Statement

Write a Java program to print the occurrence of an element from an array. Both the array and the element will be given by the user.

Example: -

Input: array = {0,2,4,8,2,9,2,0} element=2

Output: 3

Algorithm

Algorithm FrequencyOfElement

Input: Element whose frequency to be calculate. Array and the element will give by user.

Output: Display the occurrence of input element from array>

Step 1: Start

Step 2: count ← 0

Step 3: For (i=0 to i<size) do

  Step 3.1: If (array[i] = num) then

    Step 3.1.1: count ← count + 1

  Step 3.2: End If

Step 4: End For

Step 5: If (count = 0) then

  Step 5.1: Print (num is not present in the array)

Step 6: Else

  Step 6.1: Print (count)

Step 7: End If

Step 8: Stop

Source Code

```
import java.util.Scanner;

class FrequencyOfElement
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);			// create an instance of Scanner class
		System.out.print("Enter the required size of the array: ");
		int size = input.nextInt();				// take user input for array size
		if (size <= 0)
			System.out.print("Invalid input! Array size can't be less than or equal to zero.");
		else
		{
			int array[] = new int[size];
			System.out.print("Enter the elements of the array: ");		// Input the elements of the array
			for (int i = 0; i < size; i++)
			{
				array[i] = input.nextInt();
			}
			System.out.print("\nEnter an array element to find its frequency: ");
			int num = input.nextInt(); // take user input against the given problem
			int count = 0;
			for (int i = 0; i < size; i++)
			{
				if (array[i] == num)
					count++; // count the frequency of the given user input
			}
			if (count == 0)
				System.out.print(num + " is not present in the given array.");
			else
				System.out.print("Occurrence of " + num + " is: " + count);
		}
	}
}
```

Result

C:\JAVA\College>*javac A12.java*

C:\JAVA\College>*java FrequencyOfElement*
Enter the required size of the array: 8
Enter the elements of the array: 0 2 4 8 2 9 2 0

Enter an array element to find its frequency: 2
Occurrence of 2 is: 3

C:\JAVA\College>*java FrequencyOfElement*
Enter the required size of the array: 10
Enter the elements of the array: -7 -5 9 -5 -5 0 7 9 -5 8

Enter an array element to find its frequency: -5
Occurrence of -5 is: 4

C:\JAVA\College>*java FrequencyOfElement*
Enter the required size of the array: 10
Enter the elements of the array: 9 5 7 -5 9 0 4 9 -11 6

Enter an array element to find its frequency: 3
3 is not present in the given array.

Discussion

- This Java program is designed to find the frequency (number of occurrences) of a user-specified element in an array.
- It begins by creating an instance of the *Scanner class* to read user input and prompts the user to input the desired size of the array.
- The program includes input validation to ensure that the array size is not less than or equal to zero.
- Users are then prompted to input the elements of the array, and these elements are stored in an integer array.
- After inputting the array elements, users are asked to enter an element to find its frequency within the array. The program iterates through the array and counts how many times the specified element appears. Finally, it displays the frequency of the element, or a message indicating that the element is not present in the array if its frequency is zero. This program provides a straightforward solution for finding the frequency of an element in an array.

Problem Statement

Write a Java program to find and print the common elements from two single dimensional arrays. After that print the contents of both arrays except the common elements. Both the arrays will be given by the user.

Example: -

Input: array1 = {0,5,4,3,100} array2 = {18,3,100,6,78,2,15,18}

Output: 3, 100 array1 = {0,5,4} array2 = {18,6,78,2,15,18}

Algorithm

Algorithm CommonArrayElements

Input: A number which is to be remove from an array.

Output: After removing the number display the full array. If user give a wrong number then print a warning –"Element not found".

Step 1: Start
Step 2: i ← -1
Step 3: For (j=0 to j<size1) do
  Step 3.1: For (k=0 to k<size2) do
    Step 3.1.1: If (array1[j] = array2[k] and array1[j] ≠ -1 and array2[k] ≠ -1) then
      Step 3.1.1.1: aux[i+1] ← array1[j]
      Step 3.1.1.2: array1[j] ← -1
      Step 3.1.1.3: array2[k] ← -1
    Step 3.1.2: End If
  Step 3.2: End For
Step 4: End For
Step 5: If (i=-1) then
  Step 5.1: Print "No common element found"
Step 6: Else
  Step 6.1: For (j=0 to j<=i) do
    Step 6.1.1: Print (aux[j])
  Step 6.2: End For
Step 7: For (j from array1) do
  Step 7.1: If (j ≠ -1)
    Step 7.1.1: Print (j)
  Step 7.2: End If
Step 8: End For
Step 9: For (j from array1) do
  Step 9.1: If (j ≠ -1)
    Step 9.1.1: Print (j)
  Step 9.2: End If
Step 10: End For
Step 11: Stop

Source Code

```
import java.util.Scanner;

class CommonArrayElements
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);		// create an instance of Scanner class
		System.out.print("Enter the required size of the first array: ");
		int size1 = input.nextInt();			// take user input for the size of the first array
		System.out.print("Enter the required size of the second array: ");
		int size2 = input.nextInt();			// take user input for the size of the second array

		if (size1 <= 0 || size2 <= 0)
		{
			System.out.print("Invalid input! Array size can't be less than or equal to zero.");
		}
		else
		{
			// Initialize the arrays
			int array1[] = new int[size1];
			int array2[] = new int[size2];
			int aux[] = new int[size1];

			System.out.print("Enter the elements of the first array: ");	// Input the elements in first array
			for (int i = 0; i < size1; i++)
			{
				array1[i] = input.nextInt();
			}

			System.out.print("Enter the elements of the second array: ");	// Input the elements in second array
			for (int i = 0; i < size2; i++)
			{
				array2[i] = input.nextInt();
			}

			int i = -1;

			for (int j = 0; j < size1; j++)
			{
				for (int k = 0; k < size2; k++)
				{
					if (array1[j] == array2[k] && array1[j] != -1 && array2[k] != -1)
					{
						aux[++i] = array1[j]; // moves a copy of the common element into an
									auxiliary array
						array1[j] = -1;		// replaces the common element with -1
						array2[k] = -1;
					}
				}
			}
			if (i == -1)
			{
				System.out.println("\nNo common element found.");
			}
			else
```

```
                        {
                                System.out.print("\nThe common element(s) between the given arrays:");
                                for (int j = 0; j <= i; j++)
                                        System.out.print(" " + aux[j]);
                                System.out.print("\nFirst array elements without common array element(s):");
                                for (int j : array1)
                                {
                                        if (j != -1)
                                                System.out.print(" " + j);
                                }
                                System.out.print("\nSecond array elements without common array element(s): ");
                                for (int j : array2)
                                {
                                        if (j != -1)
                                                System.out.print(" " + j);
                                }
                        }
                }
        }
}
```

Result

C:\JAVA\College>*javac A13.java*

C:\JAVA\College>*java CommonArrayElements*
Enter the required size of the first array: 5
Enter the required size of the second array: 8
Enter the elements of the first array: 0 5 4 3 100
Enter the elements of the second array: 18 3 100 6 78 2 15 18

The common element(s) between the given arrays: 3 100
First array elements without common array element(s): 0 5 4
Second array elements without common array element(s): 18 6 78 2 15 18

C:\JAVA\College>*java CommonArrayElements*
Enter the required size of the first array: 8
Enter the required size of the second array: 5
Enter the elements of the first array: -6 5 7 -2 0 6 -4 -8
Enter the elements of the second array: -2 15 -8 0 25

The common element(s) between the given arrays: -2 0 -8
First array elements without common array element(s): -6 5 7 6 -4
Second array elements without common array element(s): 15 25

C:\JAVA\College> *java CommonArrayElements*
Enter the required size of the first array: 5
Enter the required size of the second array: 5
Enter the elements of the first array: 0 10 20 30 -40
Enter the elements of the second array: -5 5 15 25 -35

No common element found.

C:\JAVA\College>*java CommonArrayElements*
Enter the required size of the first array: 5

Enter the required size of the second array: 4
Enter the elements of the first array: -5 0 3 4 6
Enter the elements of the second array: -6 2 1 8

No common element found.

Discussion

- **for (int j : array1):** This loop iterates over each element in '*array1*'. In each iteration, the variable *'j'* takes on the value of the current element in '*array1*'. It allows you to work with the elements directly without needing an index variable.
- **for (int j : array2):** Similarly, this loop iterates over each element in '*array2*'. In each iteration, the variable *'j'* takes on the value of the current element in '*array2*'.

- This Java program is designed to find and display common elements between two arrays while also showing the elements unique to each array.
- It first creates an instance of the Scanner class to read user input and prompts the user to enter the sizes of two arrays.
- The program includes input validation to ensure that the array sizes are not less than or equal to zero.
- Users are then prompted to input the elements for both arrays, and these elements are stored in separate integer arrays.
- The program uses nested loops to compare each element of the first array with every element of the second array. When a common element is found, it is copied to an auxiliary array, and the corresponding elements in both arrays are marked as -1. The program then displays the common elements and the elements unique to each array. This program provides a clear solution for identifying common and unique elements between two arrays.

Problem Statement

Write a Java program to print the following pattern (where number of lines will be given by the user): -

***0**
00**
*****000**
0000****
[The above output is for 4 lines.]

Algorithm

Algorithm Pattern1
Input: No of rows given by user.
Output: Display pattern according to input row.
Step 1: Start
Step 2: num ← no. of row input from user
Step 3: If (num>0) then
    Step 3.1: For (i=1 to num) do
        Step 3.1.1: If (i%2 ≠ 0) then
            Step 3.1.1.1: For (j=0 to j<i) do
                Step 3.1.1.1.1: Print ("*")
            Step 3.1.1.2: End For
            Step 3.1.1.3: For (j=0 to j<i) do
                Step 3.1.1.3.1: Print ("0")
            Step 3.1.1.4: End For
            Step 3.1.1.5: Print (NewLine)
        Step 3.1.2: Else
            Step 3.1.2.1: For (j=0 to j<i) do
                Step 3.1.2.1.1: Print ("0")
            Step 3.1.2.2: End For
            Step 3.1.2.3: For (j=0 to j<i) do
                Step 3.1.2.3.1: Print ("*")
            Step 3.1.2.4: End For
            Step 3.1.2.5: Print (NewLine)
        Step 3.1.3: End If
    Step 3.2: End For
Step 4: Else
    Step 4.1: Print ("Enter Positive Number")
Step 5: End If
Step 6: Stop

Source Code

```
import java.util.Scanner;
class Pattern1
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);		// Create a Scanner object to read input from the user
		System.out.print("Enter the number of lines: ");	// Prompt the user to enter the number of lines
		int num = input.nextInt();			// Read the user's input as an integer
		if(num>0)					// Check if the input is positive
		{
			for (int i = 1; i <= num; i++)		// Loop through each line
			{
				if (i%2 != 0)			// Check if the current line number is odd
				{
					for (int j = 0; j < i; j++)	// Print '*' characters for the first half of the line
					{
						System.out.print("*");
					}
					for (int j = 0; j < i; j++)	// Print '0' characters for the second half of the line
					{
						System.out.print("0");
					}
					System.out.println();		// Move to the next line
				}
				else					// If the current line number is even, reverse the pattern
				{
					for (int j = 0; j < i; j++)	// Print '0' characters for the first half of the line
					{
						System.out.print("0");
					}
					for (int j = 0; j < i; j++)	// Print '*' characters for the second half of the line
					{
						System.out.print("*");
					}
					System.out.println();
				}
			}
		}
		else						// If the input is not positive, display an error message
		{
			System.out.println("!! Enter a positive number !!");
		}
	}
}
```

Result

C:\JAVA\College>*javac A14.java*

C:\JAVA\College>*java Pattern1*
Enter the number of lines: 4
**0*
*00***
****000*
*0000*****

C:\JAVA\College>*java Pattern1*
Enter the number of lines: 8
**0*
*00***
****000*
*0000*****
*****00000*
*000000*******
*******0000000*
*00000000*********

C:\JAVA\College>*java Pattern1*
Enter the number of lines: 0
!! Enter a positive number !!

C:\JAVA\College>*java Pattern1*
Enter the number of lines: -5
!! Enter a positive number !!

Discussion

- The program takes user input to determine the number of lines to be printed in a pattern.
- It first checks whether the input number is positive (greater than 0) to ensure that it's a valid input.
- If the input is positive, it enters a loop that iterates from 1 to the specified number of lines (inclusive).
- Inside the loop, it further checks if the current line number is odd (i%2 != 0). If it's odd, it prints '*' characters for the first half of the line and '0' characters for the second half, creating a pattern.
- If the current line number is even, it reverses the pattern by printing '0' characters for the first half and '*' characters for the second half.
- After completing each line, it moves to the next line by using System.out.println() to add a newline character.
- If the user enters a non-positive number, the program displays an error message.

Overall, the program generates a pattern of alternating *'*' and '0'* characters in a *triangular form*, with *odd-numbered lines starting with '**'* and *even-numbered lines starting with '0'*.

Problem Statement

Write a Java program to calculate matrix addition, subtraction and multiplication using switch case. The two matrices and the choice of operation between these two matrices will be given by the user.

Algorithm

Algorithm Calculator

Input: Rows and columns with matrix from user.

Output: Display resultant matrix or warning according to user input.

Step 1: Start

Step 2: r1, c1, r2, c2 ← input rows and columns for Matrix A and Matrix B

Step 3: If (r1 < 1 or c1 < 1 or r2 < 1 or c2 < 1) then

  Step 3.1: Print ("Numbers of rows and column will be positive")

  Step 3.2: Return

Step 4: End If

Step 5: Input Matrix A and Matrix B from user

Step 6: Chose option Choice for (Addition/Subtraction/Multiplication)

Step 7: If (Choice = 1) then *//addition*

  Step 7.1: If (r1 ≠ r2 or c1 ≠ c2) then

    Step 7.1.1: Print ("Matrix addition is not possible. Matrices A and B must have the same dimensions.");

    Step 7.1.2: Return

  Step 7.2: End If

  Step 7.3: For (i = 0 to i < r1) do

    Step 7.3.1: For (j = 0 to j < c2) do

      Step 7.3.1.1: Result[i][j] ← matrixA[i][j] + matrixB[i][j]

    Step 7.3.2: End For

  Step 7.4: End For

  Step 7.5: Break

Step 8: Else If (Choice = 2) then *//subtraction*

  Step 8.1: If (r1 ≠ r2 or c1 ≠ c2) then

    Step 8.1.1: Print ("Matrix subtraction is not possible. Matrices A and B must have the same dimensions.");

    Step 8.1.2: Return

  Step 8.2: End If

  Step 8.3: For (i = 0 to i < r1) do

    Step 8.3.1: For (j = 0 to j < c2) do

      Step 8.3.1.1: Result[i][j] ← matrixA[i][j] - matrixB[i][j]

    Step 8.3.2: End For

  Step 8.4: End For

  Step 8.5: Break

Step 9: Else if (Choice = 3) then *//multiplication*

  Step 9.1: If (c1 ≠ r2)

    Step 9.1.1: Print ("Matrix multiplication is not possible. Number of columns in A must be equal to the number of rows in B.");

    Step 9.1.2: Return

  Step 9.2: End If

  Step 9.3: For (i = 0 to i < r1) do

    Step 9.3.1: For (j = 0 to j < c2) do

      Step 9.3.1.1: Result[i][j] ← 0

      Step 9.3.1.2: For (k = 0 to k < c1) do

        Step 9.3.1.2.1: Result[i][j] += matrixA[i][k] * matrixB[k][j]

      Step 9.3.1.3: End For

    Step 9.3.2: End For

  Step 9.4: End For

  Step 9.5: Break

Step 10: Else *//default*

  Step 10.1: Print ("Invalid")

  Step 10.2: Return

Step 11: End If

Step 12: Stop

Source Code

```
import java.util.Scanner;
class Calculator
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);

		// Input dimensions of the matrices
		System.out.print("Enter the number of rows for matrix A: ");
		int m1 = input.nextInt();
		System.out.print("Enter the number of columns for matrix A: ");
		int n1 = input.nextInt();

		System.out.print("Enter the number of rows for matrix B: ");
		int m2 = input.nextInt();
		System.out.print("Enter the number of columns for matrix B: ");
		int n2 = input.nextInt();

		if (m1 < 1 || n1 < 1 || m2 < 1 || n2 < 1)
		{
			System.out.println("Please enter positive values for the number of rows and columns.");
			return;   // Exit the program
		}

		// Input matrices A and B
		int[][] matrixA = new int[m1][n1];
		int[][] matrixB = new int[m2][n2];

		System.out.println("Enter elements for matrix A:");
		for (int i = 0; i < m1; i++)
		{
			for (int j = 0; j < n1; j++)
			{
				matrixA[i][j] = input.nextInt();
			}
		}

		System.out.println("Enter elements for matrix B:");
		for (int i = 0; i < m2; i++)
		{
			for (int j = 0; j < n2; j++)
			{
				matrixB[i][j] = input.nextInt();
			}
		}

		// Choose the operation
		System.out.println("Choose operation:");
		System.out.println("1. Addition");
		System.out.println("2. Subtraction");
		System.out.println("3. Multiplication");
		System.out.print("Enter your choice (1/2/3): ");
		int choice = input.nextInt();

		int[][] result = new int[m1][n2];
```

```
// Perform the chosen operation
switch (choice)
{
        case 1:
                // Addition
                if (m1 != m2 || n1 != n2)
                {
                        System.out.println("Matrix addition is not possible. Matrices A and B must
                                                have the same dimensions.");
                        return;
                }
                for (int i = 0; i < m1; i++)
                {
                        for (int j = 0; j < n2; j++)
                        {
                                result[i][j] = matrixA[i][j] + matrixB[i][j];
                        }
                }
                break;
        case 2:
                // Subtraction
                if (m1 != m2 || n1 != n2)
                {
                        System.out.println("Matrix subtraction is not possible. Matrices A and B must
                                                have the same dimensions.");
                        return;
                }
                for (int i = 0; i < m1; i++)
                {
                        for (int j = 0; j < n2; j++)
                        {
                                result[i][j] = matrixA[i][j] - matrixB[i][j];
                        }
                }
                break;
        case 3:
                // Multiplication
                // Check if matrices can be operated on
                if (n1 != m2)
                {
                        System.out.println("Matrix multiplication is not possible. Number of columns
                                                in A must be equal to the number of rows in B.");
                        return;
                }

                for (int i = 0; i < m1; i++)
                {
                        for (int j = 0; j < n2; j++)
                        {
                                result[i][j] = 0;
                                for (int k = 0; k < n1; k++)
                                {
                                        result[i][j] += matrixA[i][k] * matrixB[k][j];
                                }
                        }
                }
                break;
```

```
                default:
                        System.out.println("Invalid choice");
                        return;
                }

                // Print the result matrix
                System.out.println("Result Matrix:");
                for (int i = 0; i < m1; i++)
                {
                        for (int j = 0; j < n2; j++)
                        {
                                System.out.print(result[i][j] + " ");
                        }
                        System.out.println();
                }
        }
}
```

Result

C:\JAVA\College>*javac A15.java*

C:\JAVA\College>*java Calculator*
Enter the number of rows for matrix A: 3
Enter the number of columns for matrix A: 3
Enter the number of rows for matrix B: 3
Enter the number of columns for matrix B: 3
Enter elements for matrix A:
2 2 1
1 5 0
0 0 1
Enter elements for matrix B:
5 7 1
0 3 0
1 0 8
Choose operation:
1. Addition
2. Subtraction
3. Multiplication
Enter your choice (1/2/3): 1
Result Matrix:
7 9 2
1 8 0
1 0 9

C:\JAVA\College>*java Calculator*
Enter the number of rows for matrix A: 3
Enter the number of columns for matrix A: 3
Enter the number of rows for matrix B: 3
Enter the number of columns for matrix B: 3
Enter elements for matrix A:
2 2 1
1 5 0
0 0 1

Enter elements for matrix B:
5 7 1
0 3 0
1 0 8
Choose operation:
1. Addition
2. Subtraction
3. Multiplication
Enter your choice (1/2/3): 2
Result Matrix:
-3 -5 0
1 2 0
-1 0 -7

C:\JAVA\College>*java Calculator*
Enter the number of rows for matrix A: 3
Enter the number of columns for matrix A: 4
Enter the number of rows for matrix B: 3
Enter the number of columns for matrix B: 3
Matrix multiplication is not possible. Number of columns in A must be equal to the number of rows in B.

C:\JAVA\College>*java Calculator*
Enter the number of rows for matrix A: 3
Enter the number of columns for matrix A: 3
Enter the number of rows for matrix B: 3
Enter the number of columns for matrix B: 3
Enter elements for matrix A:
1 2 3
3 4 -2
3 2 1
Enter elements for matrix B:
-1 1 1
3 4 -2
3 2 1
Choose operation:
1. Addition
2. Subtraction
3. Multiplication
Enter your choice (1/2/3): 3
Result Matrix:
14 15 0
3 15 -7
6 13 0

C:\JAVA\College>*java Calculator*
Enter the number of rows for matrix A: 3
Enter the number of columns for matrix A: -3
Enter the number of rows for matrix B: 3
Enter the number of columns for matrix B: 3
Please enter positive values for the number of rows and columns.

C:\JAVA\College>*java Calculator*
Enter the number of rows for matrix A: 3
Enter the number of columns for matrix A: 4
Enter the number of rows for matrix B: 4
Enter the number of columns for matrix B: 3
Enter elements for matrix A:
7 5 3 2
1 5 9 3
1 4 5 6
Enter elements for matrix B:
4 5 6
7 8 9
1 2 3
1 4 7
Choose operation:
1. Addition
2. Subtraction
3. Multiplication
Enter your choice (1/2/3): 1
Matrix addition is not possible. Matrices A and B must have the same dimensions.

C:\JAVA\College>*java Calculator*
Enter the number of rows for matrix A: 2
Enter the number of columns for matrix A: 2
Enter the number of rows for matrix B: 2
Enter the number of columns for matrix B: 2
Enter elements for matrix A:
1 2
3 4
Enter elements for matrix B:
7 8
6 5
Choose operation:
1. Addition
2. Subtraction
3. Multiplication
Enter your choice (1/2/3): 4
Invalid choice

C:\JAVA\College>*java Calculator*
Enter the number of rows for matrix A: 2
Enter the number of columns for matrix A: 2
Enter the number of rows for matrix B: 2
Enter the number of columns for matrix B: 2
Enter elements for matrix A:
1 2 5
3 4 4
Enter elements for matrix B:
7 8
6 5
Choose operation:
1. Addition
2. Subtraction
3. Multiplication
Enter your choice (1/2/3): 2
Matrix subtraction is not possible. Matrices A and B must have the same dimensions.

Discussion

- ***User Input for Matrix Dimensions:*** The program begins by prompting the user to input dimensions (number of rows and columns) for two matrices, A and B. It also performs checks to ensure that these dimensions are positive, displaying a warning message and exiting if they are not.

- ***Input of Matrices A and B:*** After obtaining the dimensions, the program creates two 2D arrays, *matrixA* and *matrixB*, to store the elements of the matrices. It then uses nested loops to populate these matrices with user-provided values.

- ***Operation Selection:*** The program allows the user to choose from three matrix operations: addition, subtraction, and multiplication. It displays options and prompts the user for their choice using a *switch* statement.

- ***Matrix Operations:*** Depending on the user's choice, the program performs the selected matrix operation. For addition and subtraction, it checks if the matrices A and B have the same dimensions before performing the operation. For multiplication, it checks if the number of columns in matrix A is equal to the number of rows in matrix B, as required for matrix multiplication. The results of the operations are stored in the *result* matrix.

- ***Result Display:*** Finally, the program prints the resulting matrix based on the chosen operation. It displays the elements of the result matrix in the standard matrix format.

- ***Error Handling:*** The program includes error handling to notify the user of invalid input or operations that cannot be performed due to incompatible matrix dimensions.

Problem Statement

Write a program in java that accepts a 2D matrix (m X n) and prints the matrix with row minimum and column maximum values in the following format: -

Example: - Input:

4	**3**	**5**
1	**0**	**7**
8	**4**	**6**

Output:

4	**3**	**5**	*3*
1	**0**	**7**	*0*
8	**4**	**6**	*4*
8	*4*	*7*	

Algorithm

Algorithm RMCM

Input: No. of row and column and the matrix.

Output: Display the resultant matrix with row minimum and column maximum.

Step 1: Start

Step 2: m ← no. of row input, n ← no. of column input

Step 3: If (m<1 or n<1) then

- **Step 3.1:** Print ("Please enter positive values for the number of rows and columns.")
- **Step 3.2:** Return

Step 4: End If

Step 5: For (i=0 to i<m) do

- **Step 5.1:** min ← matrix[i][0]
- **Step 5.2:** For (j=1 to j<n) do
  - **Step 5.2.1:** If (matrix[i][j] < min) then
    - **Step 5.2.1.1:** min ← matrix[i][j]
  - **Step 5.2.2:** End If
- **Step 5.3:** End For
- **Step 5.4:** rowMin[i] ← min

Step 6: End For

Step 7: For (i=0 to i<n) do

- **Step 7.1:** max ← matrix[0][i]
- **Step 7.2:** For (j=1 to j<m) do
  - **Step 7.2.1:** If (matrix[i][j] < max) then
    - **Step 7.2.1.1:** max ← matrix[i][j]
  - **Step 7.2.2:** End If
- **Step 7.3:** End For
- **Step 7.4:** colMax[j] ← max

Step 8: End For

Step 9: For (i=0 to i<m) do

- **Step 9.1:** For (j=0 to j<n) do
  - **Step 9.1.1:** Print (matrix[i][j])
- **Step 9.2:** End For
- **Step 9.3:** Print (rowMin[i])
- **Step 9.4:** Print (NewLine)

Step 10: End For

Step 11: For (j=0 to j<n) do

- **Step 11.1**: Print (colMax[j])

Step 12: End For

Step 13: Stop

Source Code

```
import java.util.Scanner;
class RMCM
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);

		// Input the dimensions of the matrix
		System.out.print("Enter the number of rows (m): ");
		int m = input.nextInt();
		System.out.print("Enter the number of columns (n): ");
		int n = input.nextInt();

		if (m < 1 || n < 1)
		{
			System.out.println("Please enter positive values for the number of rows and columns.");
			return;			// Exit the program
		}

		// Input the matrix
		int[][] matrix = new int[m][n];
		System.out.println("Enter the elements of the matrix:");
		for (int i = 0; i < m; i++)
		{
			for (int j = 0; j < n; j++)
			{
				matrix[i][j] = input.nextInt();
			}
		}

		// Find row minimums and column maximums
		int[] rowMin = new int[m];
		int[] colMax = new int[n];

		for (int i = 0; i < m; i++)
		{
			int min = matrix[i][0];
			for (int j = 1; j < n; j++)
			{
				if (matrix[i][j] < min)
				{
					min = matrix[i][j];
				}
			}
			rowMin[i] = min;
		}

		for (int j = 0; j < n; j++)
		{
			int max = matrix[0][j];
			for (int i = 1; i < m; i++)
			{
				if (matrix[i][j] > max)
				{
					max = matrix[i][j];
```

```
                                        }
                                }
                                colMax[j] = max;
                        }
                        System.out.print("\n");

                        // Print the modified matrix
                        System.out.println("Modified Matrix:");
                        for (int i = 0; i < m; i++)
                        {
                                for (int j = 0; j < n; j++)
                                {
                                        System.out.print(matrix[i][j] + "\t");
                                }
                                System.out.print(rowMin[i]);
                                System.out.println();
                        }

                        for (int j = 0; j < n; j++)
                        {
                                System.out.print(colMax[j] + "\t");
                        }
                }
}
```

Result

C:\JAVA\College>*javac A16.java*

C:\JAVA\College>*java RMCM*
Enter the number of rows (m): 3
Enter the number of columns (n): 3
Enter the elements of the matrix:
4 3 5
1 0 7
8 4 6

Modified Matrix:
4 3 5 3
1 0 7 0
8 4 6 4
8 4 7

C:\JAVA\College>*java RMCM*
Enter the number of rows (m): 3
Enter the number of columns (n): 4
Enter the elements of the matrix:
5 6 7 -8
6 0 -4 6
7 -10 50 8

Modified Matrix:
5 6 7 -8 -8
6 0 -4 6 -4
7 -10 50 8 -10
7 6 50 8

C:\JAVA\College>*java RMCM*
Enter the number of rows (m): 3
Enter the number of columns (n): 3
Enter the elements of the matrix:
-4 -6 -7
-8 -2 -4
0 -5 -1

Modified Matrix:
-4 -6 -7 -7
-8 -2 -4 -8
0 -5 -1 -5
0 -2 -1

C:\JAVA\College>*java RMCM*
Enter the number of rows (m): -3
Enter the number of columns (n): 3
Please enter positive values for the number of rows and columns.

Discussion

- ***User Input:*** The program starts by taking user input for the number of rows (m) and columns (n) of a matrix using the Scanner class. It also includes a check to ensure that the entered dimensions are positive, providing a warning message if not.
- ***Matrix Input:*** After obtaining the matrix dimensions, the program creates a 2D array called matrix to store the elements of the matrix. It then uses nested loops to populate the matrix with user-provided values.
- ***Row Minimums and Column Maximums:*** The program calculates the minimum value for each row (*rowMin*) and the maximum value for each column (*colMax*) in the matrix. It uses nested loops to iterate through the elements of the matrix and updates these arrays accordingly.
- ***Printing the Modified Matrix:*** The program prints the modified matrix, including the original matrix values and the row minimums. Each row is followed by its respective row minimum value. The column maximums are printed in a separate line.
- ***Program Structure:*** This program showcases a structured approach to matrix manipulation and demonstrates the use of loops and arrays to perform calculations on a matrix. It also provides user-friendly input prompts and checks for negative or zero matrix dimensions.
- ***Overall Purpose:*** The program's primary purpose is to take user input for a matrix, find the minimum value for each row and the maximum value for each column, and then display the modified matrix with these additional values. It's a practical example of working with matrices in Java and can be a useful tool for basic matrix analysis tasks.

Problem Statement

Write a program in java that accepts a 2D matrix (m X n) and prints the matrix with row minimum, column maximum and the total (sum) of all elements of the matrix [in the (m, n) position of resultant matrix] in the following format: -

Example: - Input:

4	**3**	**5**
1	**0**	**7**
8	**4**	**6**

Output:

4	**3**	**5**	*3*
1	**0**	**7**	*0*
8	**4**	**6**	*4*
8	*4*	*7*	*38*

Algorithm

Algorithm RMCM2

Input: No. of row and column and the matrix.

Output: Display the resultant matrix with row minimum and column maximum.

Step 1: Start

Step 2: m ← no. of row input, n ← no. of column input, sum ← 0

Step 3: If (m<1 or n<1) then

- **Step 3.1:** Print ("Please enter positive values for the number of rows and columns.")
- **Step 3.2:** Return

Step 4: End If

Step 5: For (i=0 to i<m) do

- **Step 5.1:** min ← matrix[i][0]
- **Step 5.2:** For (j=1 to j<n) do
  - **Step 5.2.1:** If (matrix[i][j] < min) then
    - **Step 5.2.1.1:** min ← matrix[i][j]
  - **Step 5.2.2:** End If
- **Step 5.3:** End For
- **Step 5.4:** rowMin[i] ← min

Step 6: End For

Step 7: For (i=0 to i<n) do

- **Step 7.1:** max ← matrix[0][i]
- **Step 7.2:** For (j=1 to j<m) do
  - **Step 7.2.1:** If (matrix[i][j] < max) then
    - **Step 7.2.1.1:** max ← matrix[i][j]
  - **Step 7.2.2:** End If
- **Step 7.3:** End For
- **Step 7.4:** colMax[j] ← max

Step 8: End For

Step 9: For (i=0 to i<m) do

- **Step 9.1:** For (j=0 to j<n) do
  - **Step 9.1.1:** sum += matrix[i][j]
- **Step 9.2:** End For

Step 10: End For

Step 11: For (i=0 to i<m) do

- **Step 11.1:** For (j=0 to j<n) do
  - **Step 11.1.1:** Print (matrix[i][j])
- **Step 11.2:** End For
- **Step 11.3:** Print (rowMin[i])
- **Step 11.4:** Print (NewLine)

Step 12: End For

Step 13: For (j=0 to j<n) do

- **Step 13.1**: Print (colMax[j])

Step 14: End For

Step 15: Print (sum)

Step 13: Stop

Source Code

```
import java.util.Scanner;
class RMCM2
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);

		// Input the dimensions of the matrix
		System.out.print("Enter the number of rows (m): ");
		int m = input.nextInt();
		System.out.print("Enter the number of columns (n): ");
		int n = input.nextInt();

		if (m < 1 || n < 1)
		{
			System.out.println("Please enter positive values for the number of rows and columns.");
			return;			// Exit the program
		}

		// Input the matrix
		int[][] matrix = new int[m][n];
		System.out.println("Enter the elements of the matrix:");
		for (int i = 0; i < m; i++)
		{
			for (int j = 0; j < n; j++)
			{
				matrix[i][j] = input.nextInt();
			}
		}

		// Find row minimums and column maximums
		int[] rowMin = new int[m];
		int[] colMax = new int[n];

		for (int i = 0; i < m; i++)
		{
			int min = matrix[i][0];
			for (int j = 1; j < n; j++)
			{
				if (matrix[i][j] < min)
				{
					min = matrix[i][j];
				}
			}
			rowMin[i] = min;
		}

		for (int j = 0; j < n; j++)
		{
			int max = matrix[0][j];
			for (int i = 1; i < m; i++)
			{
				if (matrix[i][j] > max)
				{
					max = matrix[i][j];
```

```
                                        }
                                }
                                colMax[j] = max;
                        }

                        // Calculate the total sum of all elements
                        int totalSum = 0;
                        for (int i = 0; i < m; i++)
                        {
                                for (int j = 0; j < n; j++)
                                {
                                        totalSum += matrix[i][j];
                                }
                        }

                        // Print the modified matrix with row minima, column maxima, and total sum
                        System.out.println("Modified Matrix:");
                        for (int i = 0; i < m; i++)
                        {
                                for (int j = 0; j < n; j++)
                                {
                                        System.out.print(matrix[i][j] + "\t");
                                }
                                System.out.print(rowMin[i] + "\t");
                                System.out.println();
                        }

                        for (int j = 0; j < n; j++)
                        {
                                System.out.print(colMax[j] + "\t");
                        }

                        // Print the total sum
                        System.out.println(totalSum);
                }
}
```

Result

C:\JAVA\College>*javac A17.java*

C:\JAVA\College>*java RMCM2*
Enter the number of rows (m): 3
Enter the number of columns (n): 3
Enter the elements of the matrix:
4 3 5
1 0 7
8 4 6
Modified Matrix:
4 3 5 3
1 0 7 0
8 4 6 4
8 4 7 38

C:\JAVA\College>*java RMCM2*
Enter the number of rows (m): 3

Enter the number of columns (n): 4
Enter the elements of the matrix:
5 6 7 -8
6 0 -4 6
7 -10 50 8
Modified Matrix:
5 6 7 -8 -8
6 0 -4 6 -4
7 -10 50 8 -10
7 6 50 8 73

C:\JAVA\College>*java RMCM2*
Enter the number of rows (m): 3
Enter the number of columns (n): 3
Enter the elements of the matrix:
-4 -6 -7
-8 -2 -4
0 -5 -1
Modified Matrix:
-4 -6 -7 -7
-8 -2 -4 -8
0 -5 -1 -5
0 -2 -1 -37

C:\JAVA\College>*java RMCM2*
Enter the number of rows (m): -4
Enter the number of columns (n): 4
Please enter positive values for the number of rows and columns.

Discussion

- ***User Input:*** The program starts by using the Scanner class to take user input for the number of rows (m) and columns (n) of a matrix. It checks if the values entered are positive, and if not, it displays an error message and exits the program.
- ***Matrix Input:*** After obtaining the dimensions of the matrix, the program creates a 2D array matrix to store the elements of the matrix. It then uses nested loops to fill in the matrix with user-provided values.
- ***Row Minimums and Column Maximums:*** Next, the program calculates the minimum value for each row (*rowMin*) and the maximum value for each column (*colMax*) in the matrix. It uses nested loops to iterate through the elements of the matrix and updates these arrays accordingly.
- ***Total Sum Calculation:*** The program also calculates the total sum of all elements in the matrix by iterating through it with nested loops and accumulating the values in the *totalSum* variable.
- ***Printing the Modified Matrix:*** The program then prints the modified matrix. It displays the original matrix along with the row minimums and column maximums. Each row is followed by its respective row minimum, and each column maximum is displayed in a separate row.
- ***Printing Total Sum:*** Finally, the program prints the total sum of all elements in the matrix.

Problem Statement

Write a program in java that sorts element in ascending order using insertion sort algorithm.

Algorithm

Algorithm InsertionSort
Input: An array of n number of elements from the user.
Output: Sorted array elements in ascending order of their values.
Step 1: Start
Step 2: For (i=1 to n-1) do
Step 2.1: k ← a[i]
Step 2.2: j← i-1
Step 2.3: While (j>=0) and (k < a[j])
Step 2.3.1: a[j+1] ← a[j]
Step 2.3.2: j ← j-1
Step 2.4: End While
Step 2.5: a[j+1] ← k
Step 3: End For
Step 4: For (i=0 to n-1) do
Step 4.1: Print (a[i])
Step 5: End For
Step 6: Stop

Source Code

```
import java.util.Scanner;
class InsertionSort
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);			// Create a Scanner object for user input
		System.out.print("Enter the number of elements: "); // Prompt the user to enter the number of elements
		int n = input.nextInt();
		if(n>0)					// Check if n is a positive integer
		{
			int[] arr = new int[n];				// Create an array to store the elements
			System.out.println("Enter the elements:");	// Prompt the user to enter the elements
			for (int i = 0; i < n; i++)
			{
				arr[i] = input.nextInt();
			}

			// Check if the array is already sorted
			boolean isSorted = true;
			for (int i = 1; i < n; i++)
			{
				if (arr[i] < arr[i - 1])
				{
					isSorted = false;
					break;
				}
			}
```

```
                        if (isSorted)
                        {
                                System.out.println("The input array is already sorted.");
                        }
                        else
                        {
                                // Perform Insertion Sort
                                for (int i = 1; i < n; i++)
                                {
                                        int k = arr[i];
                                        int j = i - 1;
                                        while (j >= 0 && arr[j] > k)
                                        {
                                                arr[j + 1] = arr[j];
                                                j--;
                                        }
                                        arr[j + 1] = k;
                                }

                                // Display the sorted array in ascending order
                                System.out.println("Sorted Array in Ascending Order:");
                                for (int i = 0; i < n; i++)
                                {
                                        System.out.print(arr[i] + " ");
                                }
                        }
                }
                else                    // Display an error message if n is not positive
                {
                        System.out.println(" !! Enter a positive number of elements !!");
                }
        }
}
```

Result

C:\JAVA\College>*javac A20.java*

C:\JAVA\College>*java InsertionSort*
Enter the number of elements: 8
Enter the elements:
5 6 -7 9 0 4 3 -8
Sorted Array in Ascending Order:
-8 -7 0 3 4 5 6 9

C:\JAVA\College>*java InsertionSort*
Enter the number of elements: 6
Enter the elements:
-9 -6 -4 -10 -8 0
Sorted Array in Ascending Order:
-10 -9 -8 -6 -4 0

C:\JAVA\College>*java InsertionSort*
Enter the number of elements: -5
!! Enter a positive number of elements !!

C:\JAVA\College>*java InsertionSort*
Enter the number of elements: 5
Enter the elements:
5 8 1 0 5
Sorted Array in Ascending Order:
0 1 5 5 8

C:\JAVA\College>*java InsertionSort*
Enter the number of elements: 6
Enter the elements:
-10 -9 -8 -6 -4 0
The input array is already sorted.

Discussion

- ***Input Handling:*** The program starts by taking input from the user, including the number of elements in the array and the actual elements themselves.
- ***Input Validation:*** It checks if the input array size is greater than 0 to ensure it's a valid input.
- ***Array Sorting Check:*** The program includes a check to determine if the input array is already sorted. It does so by comparing each element with its adjacent element. If the array is found to be sorted in ascending order, it prints a message indicating that it's already sorted.
- ***Insertion Sort:*** If the array is not sorted, the program proceeds to perform the Insertion Sort algorithm. It iterates through the array, placing each element in its correct position by comparing it with elements on its left side.
- ***Displaying Sorted Array:*** After sorting, the program displays the sorted array in ascending order.
- ***Input Validation for Non-Positive Size:*** It includes a condition to handle cases where the user enters a non-positive number for the array size, prompting the user to enter a positive number.

Time Complexity

- The best-case time complexity of insertion sort is ***O(n)*** when the array is already sorted.
- The average-case time complexity of insertion sort is also ***$O(n^2)$.***
- The worst-case scenario (when the array is not sorted), the time complexity of the program is ***$O(n^2)$.***

Problem Statement

Write a program in java that sorts element in ascending order using merge sort algorithm.

Algorithm

Algorithm MergeSort
Input: An array of n number of elements from the user.
Output: Sorted array elements in ascending order of their values.
Step 1: Start
Step 2: i ← start, j ← mid+1, k ← 0
Step 3: While (I ≤ mid AND j ≤ end)
    Step 3.1: If (Arr[i] ≤ Arr[j]) then
        Step 3.1.1: temp[k++] ← Arr[i++]
        Step 3.1.2: k =k+1
        Step 3.1.3: i = i+1
    Step 3.2: Else
        Step 3.2.1: temp[k++] ← Arr[j++]
        Step 3.2.2: k =k+1
        Step 3.2.3: j = j+1
    Step 3.3: End If
Step 4: End While
Step 5: While (i ≤ mid) do
    Step5.1: temp[k++] ← Arr[i++]
    Step5.2: k =k+1
    Step5.3: i = i+1
Step 6: End While
Step 7: While (j ≤ end) do
    Step7.1: temp[k++] ← Arr[j++]
    Step7.2: k =k+1
    Step7.3: j = j+1
Step 8: End While
Step 9: For (i = start to end) do
    Step 9.1: Arr[i] ← temp[i-start]
Step 10: End For
Step 11: End

Source Code

```
import java.util.Scanner;
class MergeSort
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);		// Create a Scanner object for user input

		// Input: Read the array size and elements from the user
		System.out.print("Enter the number of elements: ");
		int n = input.nextInt();
		if(n>0)
		{
			int[] arr = new int[n];
			System.out.println("Enter the elements:");
			for (int i = 0; i < n; i++)
			{
				arr[i] = input.nextInt();
			}

			// Check if the array is already sorted
			boolean isSorted = true;
			for (int i = 1; i < n; i++)
			{
				if (arr[i] < arr[i - 1])
				{
					isSorted = false;
					break;
				}
			}
			if (isSorted)
			{
				System.out.println("The input array is already sorted.");
			}
			else
			{
				// Perform merge sort
				mergeSort(arr, 0, n - 1);

				// Output: Display the sorted array
				System.out.println("Sorted array in ascending order:");
				for (int i = 0; i < n; i++)
				{
					System.out.print(arr[i] + " ");
				}
			}
		}
		else
		{
			System.out.println(" !! Enter a positive number of elements !!");
		}
	}

	// Merge Sort function
	static void mergeSort(int[] arr, int l, int r)
	{
```

```
        if (l < r)
        {
            int mid = (l + r) / 2;
            mergeSort(arr, l, mid);
            mergeSort(arr, mid + 1, r);
            merge(arr, l, mid, r);
        }
    }

    // Merge function to combine two sorted subarrays
    static void merge(int[] arr, int l, int mid, int r)
    {
        int n1 = mid - l + 1;
        int n2 = r - mid;

        int[] lArr = new int[n1];
        int[] rArr = new int[n2];

        for (int i = 0; i < n1; i++)
        {
            lArr[i] = arr[l + i];
        }
        for (int j = 0; j < n2; j++)
        {
            rArr[j] = arr[mid + 1 + j];
        }

        int i = 0, j = 0;
        int k = l;
        while (i < n1 && j < n2)
        {
            if (lArr[i] <= rArr[j])
            {
                arr[k] = lArr[i];
                i++;
            }
            else
            {
                arr[k] = rArr[j];
                j++;
            }
            k++;
        }
        while (i < n1)
        {
            arr[k] = lArr[i];
            i++;
            k++;
        }
        while (j < n2)
        {
            arr[k] = rArr[j];
            j++;
            k++;
        }
    }
}
```

Result

C:\JAVA\College>*javac A21.java*

C:\JAVA\College>*java MergeSort*
Enter the number of elements: 8
Enter the elements:
5 6 -7 9 0 4 3 -8
Sorted array in ascending order:
-8 -7 0 3 4 5 6 9

C:\JAVA\College>*java MergeSort*
Enter the number of elements: 7
Enter the elements:
-5 -7 -3 -8 0 -4 -2
Sorted array in ascending order:
-8 -7 -5 -4 -3 -2 0

C:\JAVA\College>*java MergeSort*
Enter the number of elements: -5
!! Enter a positive number of elements !!

C:\JAVA\College>*java MergeSort*
Enter the number of elements: 5
Enter the elements:
8 6 1 8 2
Sorted array in ascending order:
1 2 6 8 8

C:\JAVA\College>*java MergeSort*
Enter the number of elements: 6
Enter the elements:
-10 -9 -8 -6 -4 0
The input array is already sorted.

Discussion

- ***Input Handling:*** The program starts by taking input from the user, including the number of elements in the array and the actual elements themselves.
- ***Input Validation:*** It checks if the input array size is greater than 0 to ensure it's a valid input.
- ***Array Sorting Check:*** The program includes a check to determine if the input array is already sorted. It does so by comparing each element with its adjacent element. If the array is found to be sorted in ascending order, it prints a message indicating that it's already sorted.
- ***Merge Sort:*** If the array is not sorted, the program proceeds to perform the Merge Sort algorithm. It recursively divides the array into halves, sorts them separately, and then merges them back together to create a sorted array.
- ***Displaying Sorted Array:*** After sorting, the program displays the sorted array in ascending order.
- ***Input Validation for Non-Positive Size:*** It includes a condition to handle cases where the user enters a non-positive number for the array size, prompting the user to enter a positive number.

Time Complexity

- The Time Complexity of merge sort for Best case, average case and worst case is **O(n* logn).**

Problem Statement

Write a program in java that sorts element in ascending order using quick sort algorithm.

Algorithm

Algorithm partition(a, l, h)

Input: An unsorted array of elements of a fixed length, its lower and upper bounds are received as formal arguments from partition( ), called from quicksort( ).

Output: Initializes the "pivot" element, place it at its appropriate position and at the end returns the last index of the element from the array.

Step 1: Start
Step 2: pivot ← a[l], start ← l, end ← h
Step 3: While (start < end) do
  Step 3.1: While (a[start] <= pivot and start < h) do
    Step 3.1.1: start ← start + 1
  Step 3.2: End While
  Step 3.3: While (a[end] > pivot and end > l) do
    Step 3.3.1: end ← end -1
  Step 3.4: End While
  Step 3.5: If (start < end) then
    Step 3.5.1: swap(a[start], a[end]) *//swap( ) swaps the values of the given elements*
  Step 3.6: End If
Step 4: End While
Step 5: swap(a[l], a[end])
Step 6: Return end
Step 7: Stop

Algorithm quickSort(a[ ], l, h)

Input: An unsorted array of elements of a fixed length, its lower and upper bounds are taken as the input from the user.

Output: Recursively call the quickSort( ) and partition( ) time and again to disrupt the original array into smaller sub-arrays and returns the last location of the "pivot" element in the array at the end of partition( ).

Step 1: Start
Step 2: If (l < h) then
  Step 2.1: piv ← partition(a, l, h)
  Step 2.2: quicksort(a, l, piv-1)
  Step 2.3: quicksort(a, piv+1, h)
Step 3: End If
Step 4: Stop

Source Code

```
import java.util.Scanner;
class QuickSort
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);		// Create a Scanner object for user input

		// Input: Read the array size and elements from the user
		System.out.print("Enter the number of elements: ");
		int n = input.nextInt();

		if(n>0)
		{
			int[] arr = new int[n];
			System.out.println("Enter the elements:");
			for (int i = 0; i < n; i++)
			{
				arr[i] = input.nextInt();
			}

			// Check if the array is already sorted
			boolean isSorted = true;
			for (int i = 1; i < n; i++)
			{
				if (arr[i] < arr[i - 1])
				{
					isSorted = false;
					break;
				}
			}
			if (isSorted)
			{
				System.out.println("The input array is already sorted.");
			}
			else
			{
				// Perform quick sort
				quickSort(arr, 0, n - 1);

				// Output: Display the sorted array
				System.out.println("Sorted array in ascending order:");
				for (int i = 0; i < n; i++)
				{
					System.out.print(arr[i] + " ");
				}
			}
		}
		else
		{
			System.out.println(" !! Enter a positive number of elements !!");
		}
	}
```

```
        // Quick Sort function
        static void quickSort(int[] arr, int l, int h)
        {
                if (l < h)
                {
                        int piv = partition(arr, l, h);
                        quickSort(arr, l, piv - 1);
                        quickSort(arr, piv + 1, h);
                }
        }

        // Partition function to select a pivot and rearrange elements
        public static int partition(int[] arr, int l, int h)
        {
                int pivot = arr[h];
                int i = l - 1;
                for (int j = l; j < h; j++)
                {
                        if (arr[j] <= pivot)
                        {
                                i++;
                                int temp = arr[i];
                                arr[i] = arr[j];
                                arr[j] = temp;
                        }
                }
                int temp = arr[i + 1];
                arr[i + 1] = arr[h];
                arr[h] = temp;
                return i + 1;
        }
}
```

Result

C:\JAVA\College>*javac A22.java*

C:\JAVA\College>*java QuickSort*
Enter the number of elements: 8
Enter the elements:
5 6 -7 9 0 4 3 -8
Sorted array in ascending order:
-8 -7 0 3 4 5 6 9

C:\JAVA\College>*java QuickSort*
Enter the number of elements: 6
Enter the elements:
-8 -6 -1 -2 0 -4
Sorted array in ascending order:
-8 -6 -4 -2 -1 0

C:\JAVA\College>*java QuickSort*
Enter the number of elements: 7
Enter the elements:
9 12 3 48 62 0 7

Sorted array in ascending order:
0 3 7 9 12 48 62

C:\JAVA\College>*java QuickSort*
Enter the number of elements: -6
!! Enter a positive number of elements !!

C:\JAVA\College>*java QuickSort*
Enter the number of elements: 6
Enter the elements:
-10 -9 -8 -6 -4 0
The input array is already sorted.

Discussion

- ***Input Handling:*** The program starts by taking input from the user, including the number of elements in the array and the actual elements themselves.
- ***Input Validation:*** It checks if the input array size is greater than 0 to ensure it's a valid input.
- ***Array Sorting Check:*** The program includes a check to determine if the input array is already sorted. It does so by comparing each element with its adjacent element. If the array is found to be sorted in ascending order, it prints a message indicating that it's already sorted.
- ***Quick Sort:*** If the array is not sorted, the program proceeds to perform the Quick Sort algorithm. Quick Sort is known for its efficiency in sorting and works by selecting a pivot element and partitioning the array into two subarrays. The subarrays are then sorted recursively.
- ***Partition Function:*** The partition function selects a pivot and rearranges elements so that elements less than the pivot are on one side, and elements greater than the pivot are on the other side.
- ***Displaying Sorted Array:*** After sorting, the program displays the sorted array in ascending order.
- ***Input Validation for Non-Positive Size:*** It includes a condition to handle cases where the user enters a non-positive number for the array size, prompting the user to enter a positive number.

Time Complexity

- The best-case time complexity of quick sort is ***O(n*logn)*** occurs when the pivot element is the middle element or near to the middle element.
- The average-case time complexity of quick sort is ***O(n*logn)*** occurs when the array elements are in jumbled order that is not properly ascending and not properly descending.
- In quick sort, worst case occurs when the pivot element is either greatest or smallest element. Suppose, if the pivot element is always the last element of the array, the worst case would occur when the given array is sorted already in ascending or descending order. The worst-case time complexity of quicksort is **$O(n^2)$**.

Problem Statement

Write a program in java that sorts half of element in ascending and rest half of the elements in descending order.

Algorithm

Algorithm SortAseDes

Input: An unsorted array elements of a fixed length is taken as the input from the user

Output: Divide the array into two halves, such that the first half is sorted in ascending order of their values and the second half in descending order of their values.

Step 1: Start
Step 2: n ← size of array
Step 3: For (i=0 to i=n/2) do
    Step 3.1: For (j=i+1 to n-1) do
        Step 3.1.1: If (a[i] > a[j]) then
            Step 3.1.1.1: temp ← a[j]
            Step 3.1.1.2: a[i] ← a[j]
            Step 3.1.1.3: a[j] ← temp
        Step 3.1.2: End If
    Step 3.2: End For
Step 4: End For
Step 5: For (i=n/2 to i=n - 1) do
    Step 5.1: For (j=i+1 to n-1) do
        Step 5.1.1: If (a[i] < a[j]) then
            Step 5.1.1.1: temp ← a[i]
            Step 5.1.1.2: a[i] ← a[j]
            Step 5.1.1.3: a[j] ← temp
        Step 5.1.2: End If
    Step 5.2: End For
Step 5: End For
Step 6: Print ("Desired Array: ", a[i])
Step 7: Stop

Source Code

```
import java.util.Scanner;
class SortAseDes
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);		// Create a Scanner object for user input

		// Input: Read the number of elements
		System.out.print("Enter the number of elements: ");
		int n = input.nextInt();

		if(n>0)
		{
			int[] arr = new int[n];
			System.out.println("Enter the elements:");
			for (int i = 0; i < n; i++)
			{
				arr[i] = input.nextInt();
			}

			// Sort the first half in ascending order
			for (int i = 0; i < n / 2; i++)
			{
				for (int j = i + 1; j < n / 2; j++)
				{
					if (arr[i] > arr[j])
					{
						int temp = arr[i];
						arr[i] = arr[j];
						arr[j] = temp;
					}
				}
			}

			// Sort the second half in descending order
			for (int i = n / 2; i < n - 1; i++)
			{
				for (int j = i + 1; j < n; j++)
				{
					if (arr[i] < arr[j])
					{
						int temp = arr[i];
						arr[i] = arr[j];
						arr[j] = temp;
					}
				}
			}
			// Output: Display the sorted array
			System.out.println("Sorted array with the first half in ascending and the second half in
descending order:");
			for (int i = 0; i < n; i++)
			{
				System.out.print(arr[i] + " ");
			}
```

```
                }
                else
                {
                        System.out.println(" !! Enter a positive number of elements !!");
                }
        }
}
```

Result

C:\JAVA\College>*javac A23.java*

C:\JAVA\College>*java SortAseDes*
Enter the number of elements: 8
Enter the elements:
5 6 -7 9 0 4 3 -8
Sorted array with the first half in ascending and the second half in descending order:
-7 5 6 9 4 3 0 -8

C:\JAVA\College>*java SortAseDes*
Enter the number of elements: 7
Enter the elements:
-5 -7 -3 -8 0 -4 -2
Sorted array with the first half in ascending and the second half in descending order:
-7 -5 -3 0 -2 -4 -8

C:\JAVA\College>*java SortAseDes*
Enter the number of elements: -6
!! Enter a positive number of elements !!

C:\JAVA\College>*java SortAseDes*
Enter the number of elements: 5
Enter the elements:
8 6 1 8 2
Sorted array with the first half in ascending and the second half in descending order:
6 8 8 2 1

Discussion

- This Java program takes user input for the number of elements and then proceeds to take input for each of those elements, storing them in an integer array.
- The program sorts the first half of the array in ascending order and the second half in descending order using nested loops and a simple bubble sort algorithm. This means that the array is effectively split into two parts: the first half ascending and the second half descending.
- It provides an error message if the user enters a non-positive number of elements, reminding them to input a positive value.
- This program demonstrates basic array manipulation and sorting techniques. However, it uses a less efficient sorting algorithm (bubble sort), which may not be practical for larger datasets.
- Depending on the user's input, the program can create an interesting pattern in the sorted output, where the first half of the array is in ascending order, and the second half is in descending order.