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.

Problem Statement

Write a program in java to delete all consonants from an input string and print the resultant string.

Algorithm

Algorithm RemoveConsonant

Input: A string "str" is taken as the input from the user.
Output: Resultant string "str1" without any consonants.

Step 1: Start
Step 2: Input ("Enter a string", str)
Step 3: For (i = 0 to (lengthOf(str) – 1)) do *//lengthOf( ) determines the length of the given string*
Step 3.1: If (str[i] = 'a' or str[i] = 'e' or str[i] = 'i' or str[i] = 'o' or str[i] = 'u' or str[i] = 'A' or str[i] = 'E' or str[i] = 'I' or str[i] = 'O' or str[i] = 'U') then
Step 3.1.1: str1 ← str1+str(i) *//here '+' operator acts as a concatenating operator string str1 was initially null*
Step 3.2: End If
Step 4: End For
Step 5: Print ("Input string without consonants:", str1)
Step 6: Stop

Source Code

```
import java.util.Scanner;
class SM                        // SM class to handle string manipulation
{
        String input;
        SM(String input)        // Constructor to initialize the input string
        {
                this.input = input;
        }

        String removeConso()    // Method to remove consonants from the input string
        {
                String result = "";
                for (int i = 0; i < input.length(); i++)
                {
                        char c = input.charAt(i);
                        if (c == '.' || c == ' ')       // Replace dots and spaces with spaces in the result
```

```
                {
                        result += ' ';
                }
                // Check if the character is a vowel and add it to the result
                else if (isVowel(c))
                {
                        result += c;
                }
        }
        return result;
    }

    // Method to check if a character is a vowel
    boolean isVowel(char c)
    {
        return "aeiouAEIOU".indexOf(c) != -1;
    }
}

// Main class to handle user input and interaction with SM class
class RemConso
{
    public static void main(String[] args)
    {
        Scanner inputScanner = new Scanner(System.in);      // Create a Scanner object for user input
        System.out.print("Enter a string: ");                // Input: Read the string from user
        String userInput = inputScanner.nextLine();

        // Check input string is empty or not
        if (userInput.isEmpty())
        {
            System.out.println("Input string is empty. Please provide a valid string.");
        }
        else
        {
            SM manipulator = new SM(userInput);
            String result = manipulator.removeConso();

            if (result.isEmpty())
            {
                System.out.println("Resultant string has no consonants.");
            }
            else
            {
                System.out.println("Resultant string after removing consonants: " + result + " ");
            }
        }
    }
}
```

Result

E:\JAVA\College>*javac A24.java*

E:\JAVA\College>*java RemConso*
Enter a string: which should resolve the compilation error you encountered.
Resultant string after removing consonants: i ou eoe e oiaio eo ou eouee

E:\JAVA\College>*java RemConso*
Enter a string: Ae
Resultant string after removing consonants: Ae

E:\JAVA\College>*java RemConso*
Enter a string: 85
Resultant string has no consonants.

E:\JAVA\College>*java RemConso*
Enter a string:
Input string is empty. Please provide a valid string.

E:\JAVA\College>*java RemConso*
Enter a string: I go to college everyday.
Resultant string after removing consonants: I o o oee eea

Discussion

The ***String*** class in Java is a final class that represents a sequence of characters. All string literals in Java programs are implemented as instances of this class. Strings are immutable, i.e., their values can't be modified once they are created. However, ***StringBuffer*** and ***StringBuilder*** classes support mutable strings.

- ***Object-Oriented Approach:*** The code showcases an object-oriented approach by separating concerns into two classes: *'SM'* for string manipulation and '*RemConso'* for user interaction.
- ***String Manipulation:*** The *SM class* performs consonant removal from the provided input string, employing a method to check for vowels and replacing dots/spaces with spaces.
- ***User Interaction:*** The '*RemConso*' class interacts with the user via the console, prompting for input and displaying the resultant string after removing consonants, handling empty inputs with appropriate messages.
- ***Readability and Maintainability:*** The code employs meaningful variable names, method abstraction, and comments, enhancing its readability and making it easier for future modifications or debugging.

Problem Statement

Write a program in java to check whether a string is palindrome or not.

Algorithm

Algorithm Palindrome

Input: A string "str" is taken as the input from the user.
Output: Checking whether the string is palindrome or not.

Step 1: Start
Step 2: Input ("Enter a string:", str)
Step 3: For (i = 0 to (lengthOf(str) – 1)) do *//lengthOf( ) determines the length of the given string*
Step 3.1: st1 ← str[i] + str1 *//here '+' operator acts as a concatenating operator string str1 was initially null*
Step 4: End For
Step 5: If (str1 = str) then
Step 5.1: Print ("Given string is palindrome")
Step 6: Else
Step 6.1: Print ("Given string is non-palindrome")
Step 7: End If
Step 8: Stop

Source Code

```
import java.util.Scanner;
class PalChecker                          // PalChecker class to handle palindrome checking
{
        String inputString;
        PalChecker(String inputString)    // Constructor to initialize input string
        {
                this.inputString = inputString;
        }

        boolean isPalindrome()            // Method to check if the input string is a palindrome
        {
                if (inputString.length() == 0)          // Handling empty input
                {
                        System.out.println("!! Enter a String !!");
                        return false;
                }
```

```
                // Process the input string for palindrome checking
                String ps = ps(inputString);
                int l = 0;
                int r = ps.length() - 1;

                // Check for palindrome property
                while (l < r)
                {
                        if (ps.charAt(l) != ps.charAt(r))
                        {
                                return false;
                        }
                        l++;
                        r--;
                }
                return true;
        }

        // Method to process the string (remove non-alphanumeric characters, convert to lowercase)
        String ps(String str)
        {
                return str.replaceAll("[^a-zA-Z0-9]", "").toLowerCase();
        }
}

class Palindrome
{
        public static void main(String[] args)
        {
                Scanner inputScanner = new Scanner(System.in);          // Create a Scanner object for user input
                System.out.print("Enter a string: ");                    // Input: Read the string from user
                String userInput = inputScanner.nextLine();

                // Create PalChecker object to check palindrome
                PalChecker checker = new PalChecker(userInput);
                if (checker.isPalindrome())
                {
                        System.out.println("The entered string is a palindrome.");
                }
                else
                {
                        System.out.println("The entered string is not a palindrome.");
                }
        }
}
```

Result

E:\JAVA\College>*javac A25.java*

E:\JAVA\College>*java Palindrome*
Enter a string: level
The entered string is a palindrome.

E:\JAVA\College>*java Palindrome*
Enter a string: Level
The entered string is a palindrome.

E:\JAVA\College>*java Palindrome*
Enter a string:
!! Enter a String !!

E:\JAVA\College>*java Palindrome*
Enter a string: 1221
The entered string is a palindrome.

E:\JAVA\College>*java Palindrome*
Enter a string: School
The entered string is not a palindrome.

Discussion

The ***String*** class in Java is a final class that represents a sequence of characters. All string literals in Java programs are implemented as instances of this class. Strings are immutable, i.e., their values can't be modified once they are created. However, ***StringBuffer*** and ***StringBuilder*** classes support mutable strings.

- ***Palindrome Checking Logic:*** The code effectively determines whether a given input string is a palindrome, ignoring non-alphanumeric characters and considering case-insensitivity.
- ***Modular Structure:*** It employs a class-based approach, separating concerns with the *PalChecker* class managing the palindrome checking logic and string processing.
- ***User Interaction:*** The main method within the Palindrome class interacts with the user via the console, prompting for input and displaying whether the entered string is a palindrome or not.
- ***Error Handling:*** The program accounts for an empty input, prompting the user to enter a string when encountering an empty input scenario.
- ***Readability and Maintainability:*** The use of well-named methods (*isPalindrome, ps*), meaningful variable names, and comments enhances code readability, aiding future modifications or debugging.

Problem Statement

Write a Java method to count all words in a string.
Test Data:
Input the string: *The quick brown fox jumps over the lazy dog.*
Expected Output:
Number of words in the string: *9*

Algorithm

Algorithm WC

Input: A string "str" is taken as the input from the user.
Output: Counts and prints the number of words present in the given string.

Step 1: Start
Step 2: Input ("Enter a string:", str)
Step 3: For (i = 0 to (lengthOf(str) - 1)) do *//lengthOf( ) determines the length of given string*
Step 3.1: If (str[i] = ' ') then
Step 3.1.1: counter ← counter + 1 *//counter variable was initialized to 1*
Step 3.2: End If
Step 4: End For
Step 5: Print ("Number of words present in the given string is:", counter)
Step 6: Stop

Source Code

```
import java.util.Scanner;
class WordCounter                    // WordCounter class to handle word counting functionality
{
        String inputString;

        // Constructor to initialize input string
        WordCounter(String inputString)
        {
                this.inputString = inputString;
        }

        // Method to count words in the input string
        int countWords()
        {
                if (inputString.length() == 0)          // Handling empty input
                {
                        System.out.println(" !! Enter a String !!");
                        return 0;
                }

                return CWC(inputString);
        }

        // Method to calculate word count from the input text
        int CWC(String text)
        {
                int count = 0;
                for (int i = 0; i < text.length(); i++)
                {
                        char c = text.charAt(i);
                        if (c == ' ')
                        {
                                count++;
                        }
                }
                return count + 1;          // Adding 1 for the last word which doesn't end with a space
        }
}

class WC
{
        public static void main(String[] args)
        {
                Scanner inputScanner = new Scanner(System.in);          // Create a Scanner object for user input
                System.out.print("Enter a string: ");                    // Prompt the user to enter the string
                String userInput = inputScanner.nextLine();

                // Create WordCounter object to count words
                WordCounter wordCounter = new WordCounter(userInput);
                int wordCount = wordCounter.countWords();
                System.out.println("Number of words in the string: " + wordCount);
        }
}
```

Result

E:\JAVA\College>*javac A26.java*

E:\JAVA\College>*java WC*
Enter a string: Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible
number of words in the string: 19

E:\JAVA\College>*java WC*
Enter a string:
!! Enter a String !!

E:\JAVA\College>*java WC*
Enter a string: 12
number of words in the string: 1

E:\JAVA\College>*java WC*
Enter a string: Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior
number of words in the string: 36

Discussion

The ***String*** class in Java is a final class that represents a sequence of characters. All string literals in Java programs are implemented as instances of this class. Strings are immutable, i.e., their values can't be modified once they are created. However, ***StringBuffer*** and ***StringBuilder*** classes support mutable strings.

- ***Modular Structure:*** The code is organized into a *WordCounter* class handling word counting logic and a *WC* class for user interaction and program execution.
- ***Word Counting Methodology:*** The *WordCounter* class effectively counts words in a given string by tallying the spaces and adding 1 for the last word, accommodating various string lengths and formats.
- ***Input Handling:*** The program gracefully manages empty inputs, prompting the user to enter a string when encountering an empty input scenario.
- ***Readability and Maintainability:*** Well-named methods (*countWords, CWC*), meaningful variable names, and comments enhance code readability and comprehension, aiding future modifications or debugging.
- ***User Interaction:*** The WC class interacts with the user via the console, prompting for input and displaying the count of words in the provided string, ensuring a user-friendly experience.

Problem Statement

Write a Java program to find whether a region in the current string matches a region in another string. (Both strings and both region of the strings will be given by the user.)
Sample Output:
Input:
str1= *University Exam*
str2= *Examination*
Output: -
str1[11 - 13] == str2[0 - 2]? *true*
str1[0 - 3] == str2[0 - 3]? *false*
str1[14 - 11] == str2[0 - 3]? *true*

Algorithm

Algorithm SRM

Input: A string "str" and a sub-string "sub" are taken as the input from the user, along the positions of the given strings to be compared between.
Output: Checking whether the given position of the sub-string matches with the given position of the given string.

Step 1: Start
Step 2: Input ("Enter a string and a sub-string:", str, sub)
Step 3: Input ("Mention the positions of the string:", x1, y1)
Step 4: Input ("Mention the positions of the sub-string:", x2, y2)
Step 5: If (str[x1, y1] = sub[x2, y2]) then
    Step 5.1: Print ("True: It's a match!")
Step 6: Else
    Step 6.1: Print ("False: It doesn't match")
Step 7: End If
Step 8: Stop

Source Code

```
import java.util.Scanner;
class SM                                    // SM class to handle string comparison logic
{
        String str1;
        String str2;

        // Constructor to initialize the two strings for comparison
        SM(String str1, String str2)
        {
                this.str1 = str1;
                this.str2 = str2;
        }

        // Method to compare regions of the two strings based on provided indices
        boolean matchStrings(int s1, int e1, int s2, int e2)
        {
                if (str1.length() == 0 || str2.length() == 0)          // Handling empty strings
                {
                        System.out.println(" !! Enter String !!");
                        return false;
                }

                // Validating provided indices and comparing string regions
                if (s1 >= 0 && e1 < str1.length() && s2 >= 0 && e2 < str2.length())
                {
                        int l1 = e1 - s1 + 1;
                        int l2 = e2 - s2 + 1;

                        if (l1 != l2)
                        {
                                return false;           // Regions have different lengths
                        }

                        for (int i = 0; i < l1; i++)
                        {
                        char c1 = str1.charAt(s1 + i);
                        char c2 = str2.charAt(s2 + i);

                                if (c1 != c2)
                                {
                                        return false;   // Characters at the same position are different
                                }
                        }
                        return true;        // Both regions are identical
                }
                else
                {
                        System.out.println("!! Invalid Index !!");
                        return false;       // Provided indices are out of bounds
                }
        }
}
```

```
class SRM
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);
		System.out.print("Enter the first string: ");
		String str1 = input.nextLine();

		System.out.print("Enter the second string: ");
		String str2 = input.nextLine();

		System.out.print("Enter the starting index of the first string: ");
		int s1 = input.nextInt();

		System.out.print("Enter the ending index of the first string: ");
		int e1 = input.nextInt();

		System.out.print("Enter the starting index of the second string: ");
		int s2 = input.nextInt();

		System.out.print("Enter the ending index of the second string: ");
		int e2 = input.nextInt();

		// Creating an instance of SM and performing string comparison
		SM stringMatcher = new SM(str1, str2);
		boolean isMatching = stringMatcher.matchStrings(s1, e1, s2, e2);

		// Displaying the comparison result
		if(isMatching)
		{
			System.out.println("Output: -");
			System.out.println("str1[" + s1 + " - " + e1 + "] == str2[" + s2 + " - " + e2 + "]? -> true");
		}
		else
		{
			System.out.println("Output: -");
			System.out.println("str1[" + s1 + " - " + e1 + "] == str2[" + s2 + " - " + e2 + "]? -> false");
		}
	}
}
```

Result

E:\JAVA\College>*java SRM*
Enter the first string: University Exam
Enter the second string: Examination
Enter the starting index of the first string: 11
Enter the ending index of the first string: 13
Enter the starting index of the second string: 0
Enter the ending index of the second string: 2

Output: -
str1[11 - 13] == str2[0 - 2]? -> true

E:\JAVA\College>*java SRM*
Enter the first string: University Exam
Enter the second string: Examination
Enter the starting index of the first string: 0
Enter the ending index of the first string: 3
Enter the starting index of the second string: 0
Enter the ending index of the second string: 3

Output: -
str1[0 - 3] == str2[0 - 3]? -> false

E:\JAVA\College>*java SRM*
Enter the first string: University Exam
Enter the second string: Examination
Enter the starting index of the first string: 14
Enter the ending index of the first string: 11
Enter the starting index of the second string: 0
Enter the ending index of the second string: 3

Output: -
str1[14 - 11] == str2[0 - 3]? -> false

E:\JAVA\College>*java SRM*
Enter the first string: University Exam
Enter the second string:
!! Enter String !!

E:\JAVA\College>*java SRM*
Enter the first string: Java
Enter the second string: Java For
Enter the starting index of the first string: 1
Enter the ending index of the first string: 4
Enter the starting index of the second string: 0
Enter the ending index of the second string: 3

!! Invalid Index !!

Discussion

The ***String*** class in Java is a final class that represents a sequence of characters. All string literals in Java programs are implemented as instances of this class. Strings are immutable, i.e., their values can't be modified once they are created. However, ***StringBuffer*** and ***StringBuilder*** classes support mutable strings.

- ***Class Organization:*** The code has been refactored into a structured object-oriented layout, with a '*SM*' class handling the logic for comparing specific regions of two strings.
- ***String Comparison Logic:*** The '*SM*' class effectively compares selected regions of the input strings, handling scenarios like different lengths and out-of-bound indices.
- ***User Interaction and Input Handling:*** The '*SRM*' class interacts with the user, prompting for strings and indices, ensuring a user-friendly experience. It performs validations on indices and empty string inputs.
- ***Output Presentation:*** After comparison, the program displays whether the selected regions in both strings match, providing clear and concise output.
- ***Readability and Maintainability:*** Meaningful variable names, method abstraction, and comments enhance code readability, aiding in future modifications or debugging while explaining the purpose of each segment.

Problem Statement

Write a program in java to create Box class with parameterised constructor with an object argument to initialise length, breadth and height also create a method volume which returns the volume of the box and print it in main method.

Source Code

```
import java.util.Scanner;
class Box
{
	double length;
	double breadth;
	double height;

	// Parameterized constructor
	Box(double length, double breadth, double height)
	{
		this.length = length;
		this.breadth = breadth;
		this.height = height;
	}

	// Method to calculate and return the volume of the box
	double volume()
	{
		return length * breadth * height;
	}
}

class BoxTest
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);

		// Get user input for the dimensions
		System.out.print("Enter the length of the box: ");
		double length = input.nextDouble();

		System.out.print("Enter the breadth of the box: ");
		double breadth = input.nextDouble();

		System.out.print("Enter the height of the box: ");
		double height = input.nextDouble();

		// Check if any input is negative or empty
		if (length <= 0 || breadth <= 0 || height <= 0)
		{
			System.out.println("Invalid input. Length, breadth, and height must be positive values.");
		}
		else
		{
			// Create a Box object with user-input dimensions
			Box myBox = new Box(length, breadth, height);
```

```
                    // Calculate and print the volume
                    double boxVolume = myBox.volume();
                    System.out.println("Volume of the box is: " + boxVolume);
                }

                // Close the scanner
                input.close();
        }
}
```

Result

E:\JAVA\College>*javac A28.java*

E:\JAVA\College>*java BoxTest*
Enter the length of the box: 12
Enter the breadth of the box: 4
Enter the height of the box: 6
Volume of the box is: 288.0

E:\JAVA\College>*java BoxTest*
Enter the length of the box: -4
Enter the breadth of the box: 5
Enter the height of the box: 2
Invalid input. Length, breadth, and height must be positive values.

E:\JAVA\College>*java BoxTest*
Enter the length of the box: 4
Enter the breadth of the box: 0
Enter the height of the box: 5
Invalid input. Length, breadth, and height must be positive values.

E:\JAVA\College>*java BoxTest*
Enter the length of the box: 5
Enter the breadth of the box: 8
Enter the height of the box: 10
Volume of the box is: 400.0

E:\JAVA\College>*java BoxTest*
Enter the length of the box: 3
Enter the breadth of the box: 3
Enter the height of the box: 3
Volume of the box is: 27.0

Discussion

- ***Object-Oriented Design:*** The program demonstrates OOP principles by defining a '*Box*' *class* with attributes and methods to calculate its volume, allowing for easy reuse and abstraction.
- ***Encapsulation:*** The '*Box*' *class* encapsulates data (length, breadth, height) and functionality (volume calculation) within a single entity, promoting clean code organization and maintenance.
- ***User Interaction:*** The '*BoxTest*' class interacts with the user through the console, prompting for box dimensions and handling negative or non-positive inputs gracefully with informative messages.
- ***Error Handling:*** It ensures that all dimensions are positive values, providing clear feedback when encountering invalid inputs, ensuring the calculation is performed accurately.
- ***Input Validation:*** The program appropriately validates user input, preventing negative values or zeros from being used for box dimensions, maintaining the integrity of the calculations.

Problem Statement

Create a class Student(instance variables: *roll_no*: integer, *name*: String) with following operations

a) *create parameterised constructor to initialise the objects.*

b) *create a method* ***isEqual()*** *to check whether the two objects are equal or not which* ***returns*** *the Boolean value and gets two objects.*

c) *print the result in* ***main method*** *if objects are equals or not.*

Source Code

```
import java.util.Scanner;
class Student
{
	int roll_no;
	String name;

	// Parameterized constructor to initialize the objects
	Student(int roll_no, String name)
	{
		this.roll_no = roll_no;
		this.name = name;
	}

	// Method to check if two objects are equal
	boolean isEqual(Student oStudent)
	{
		return this.roll_no == oStudent.roll_no && this.name.equals(oStudent.name);
	}
}

class SRN
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);

		// Get user input for the first student
		System.out.print("Enter the roll number of the first student: ");
		int roll1 = input.nextInt();

		// Consume the newline character
		input.nextLine();

		System.out.print("Enter the name of the first student: ");
		String name1 = input.nextLine();

		// Get user input for the second student
		System.out.print("Enter the roll number of the second student: ");
		int roll2 = input.nextInt();

		// Consume the newline character
		input.nextLine();

		System.out.print("Enter the name of the second student: ");
		String name2 = input.nextLine();
```

```
                // Create two Student objects with user-input values
                Student s1 = new Student(roll1, name1);
                Student s2 = new Student(roll2, name2);

                // Check if the two objects are equal and print the result
                boolean areEqual = s1.isEqual(s2);
                if (areEqual)
                {
                        System.out.println("The two student objects are equal.");
                }
                else
                {
                        System.out.println("The two student objects are not equal.");
                }

                // Close the scanner
                input.close();
        }
}
```

Result

E:\JAVA\College>*javac A29.java*

E:\JAVA\College>*java SRN*
Enter the roll number of the first student: 12
Enter the name of the first student: Mohan
Enter the roll number of the second student: 13
Enter the name of the second student: Ram
The two student objects are not equal.

E:\JAVA\College>*java SRN*
Enter the roll number of the first student: 1
Enter the name of the first student: Mohan
Enter the roll number of the second student: 1
Enter the name of the second student: Mohan
The two student objects are equal.

Discussion

- **Object Creation:** The code creates a *'Student'* class encapsulating student attributes and a method to check if two student objects have equal roll numbers and names.
- **User Interaction:** The *'SRN'* class interacts with the user through the console, prompting for student roll numbers and names, facilitating the creation of two *'Student'* objects based on user input.
- **Parameterized Constructor:** The *'Student'* class employs a parameterized constructor to initialize object attributes at the time of creation, ensuring immediate object setup.
- **Equality Check:** The *'isEqual'* method within the *'Student'* class compares two 'Student' objects for equality based on roll number and name, determining if they represent the same student.
- **Input Handling:** The program handles user input using a *'Scanner'*, ensuring proper data types are used for roll numbers while handling newline characters to read strings correctly, ensuring accurate object creation and comparison.

Problem Statement

A class called MyPoint, which models a 2D point with x and y coordinates. It contains: *two instance variables x (int) and y (int)***. A default (or "no-argument" or "no-arg") constructor that construct a point at the default location of (0, 0). A overloaded constructor that constructs a point with the given x and y coordinates. A method setXY() to set both x and y. A method getXY() which returns the x and y in a 2-element int array. A toString() method that returns a string description of the instance in the format "(x, y)". A method called distance(int x, int y) that returns the distance from this point to another point at the given (x, y) coordinates, Write the MyPoint class. Also write a test driver (called TestMyPoint) to test all the public methods defined in the class.**

Source Code

```
import java.util.Scanner;
import java.lang.String;
import java.lang.Math;
class MyPoint
{
	int x;
	int y;

	// Default constructor
	MyPoint()
	{
		this.x = 0;
		this.y = 0;
	}

	// Overloaded constructor
	MyPoint(int x, int y)
	{
		this.x = x;
		this.y = y;
	}

	// Method to set both x and y
	void setXY(int x, int y)
	{
		this.x = x;
		this.y = y;
	}

	// Method to get x and y in a 2-element int array
	int[] getXY()
	{
		int[] coordinates = {this.x, this.y};
		return coordinates;
	}

	// Method to return a string description of the instance in the format "(x, y)"
	public String toString()
	{
		return "(" + this.x + ", " + this.y + ")";
	}
```

```
        // Method to calculate the distance from this point to another point at (x, y)
        double distance(MyPoint p)
        {
                int xDiff = this.x - p.x;
                int yDiff = this.y - p.y;
                return Math.sqrt(Math.pow(xDiff, 2) + Math.pow(yDiff, 2));
        }
}

class TestMyPoint
{
        public static void main(String[] args)
        {
                Scanner input = new Scanner(System.in);

                // Get user input for the First point
                System.out.print("Enter x1: ");
                int x1 = input.nextInt();
                System.out.print("Enter y1: ");
                int y1 = input.nextInt();

                // Get user input for the second point
                System.out.print("Enter x2: ");
                int x2 = input.nextInt();
                System.out.print("Enter y2: ");
                int y2 = input.nextInt();

                // Create MyPoint objects
                MyPoint P1 = new MyPoint();
                MyPoint P2 = new MyPoint(x2, y2);

                P1.setXY(x1, y1);               //initialize P1 using parameterized constructor

                System.out.println("Point 1 coordinates: (" + P1.getXY()[0] + ", " + P1.getXY()[1] + ")");
                System.out.println("Point 2 coordinates: " + P2.toString());

                System.out.printf("Distance between Point 1 and Point 2: %.2f" ,P1.distance(P2));

                // Close the scanner
                input.close();
        }
}
```

Result

E:\JAVA\College>*javac A30.java*

E:\JAVA\College>*java TestMyPoint*
Enter x1: 3
Enter y1: 2
Enter x2: 7
Enter y2: 8
Point 1 coordinates: (3, 2)
Point 2 coordinates: (7, 8)
Distance between Point 1 and Point 2: 7.21

E:\JAVA\College>*java TestMyPoint*
Enter x1: 0
Enter y1: 0
Enter x2: 3
Enter y2: 4
Point 1 coordinates: (0, 0)
Point 2 coordinates: (3, 4)
Distance between Point 1 and Point 2: 5.0

E:\JAVA\College>*java TestMyPoint*
Enter x1: 5
Enter y1: 6
Enter x2: 8
Enter y2: 3
Point 1 coordinates: (5, 6)
Point 2 coordinates: (8, 3)
Distance between Point 1 and Point 2: 4.24

E:\JAVA\College>*java TestMyPoint*
Enter x1: 3
Enter y1: -2
Enter x2: -3
Enter y2: 4
Point 1 coordinates: (3, -2)
Point 2 coordinates: (-3, 4)
Distance between Point 1 and Point 2: 8.49

Discussion

- **Class Structure:** The code comprises a *'MyPoint'* class encapsulating coordinates, offering methods for setting coordinates, retrieving as an array, generating a string description, and computing distances between points.
- **Constructors and Methods:** Two constructors, one default and one parameterized, allow flexible instantiation of *'MyPoint'* objects, along with methods for setting coordinates, getting coordinates as an array, generating string representations, and calculating distances.
- **User Interaction:** The *'TestMyPoint'* class interacts with the user, prompting for coordinates of two points via the console, creating *'MyPoint'* objects accordingly.
- **Coordinate Handling:** User inputs are used to initialize and manipulate *'MyPoint'* objects, showcasing the class's ability to manage coordinates and compute distances between points accurately.
- **Mathematical Calculations:** The code utilizes mathematical functions from the *'Math'* class to calculate the Euclidean distance between two points, demonstrating a practical use-case of mathematical computations in programming.

Problem Statement

Write a program in java to find out the factorial of number 5 and 4 using variable length arguments.

Source Code

```
class Factorial
{
        // Method to calculate factorial of numbers passed as variable arguments
        static int factorial(int... nums)
        {
                int res = 1;                    // Variable to store the factorial result
                for (int n : nums)              // Loop through each number in the array
                {
                        for (int i = 1; i <= n; i++)            // Calculate factorial for each number
                        {
                                res *= i;                       // Multiply to calculate factorial
                        }
                        System.out.println("Factorial of " + n + " is: " + res);   // Print factorial of the current number
                        res = 1;                // Reset result for the next number
                }
                return res;                     // Return the final factorial (will always be 1 due to reset)
        }

        public static void main(String[] args)
        {
                Factorial.factorial(5, 4);              // Calling factorial method with numbers 5 and 4
        }
}
```

Result

E:\JAVA\College>*javac A31.java*

E:\JAVA\College>*java Factorial*
Factorial of 5 is: 120
Factorial of 4 is: 24

Discussion

- ***Factorial Calculation:*** The method *'factorial'* takes in variable arguments *'int... nums'*, where it iterates through each number in the array. For each number *'n'*, it computes its factorial by iteratively multiplying from 1 to *'n'*.
- ***Resetting Result:*** After computing the factorial for each number, it prints the result and resets *'res'* to 1 for the next number in the list.
- ***Main Method Invocation:*** In the *'main'* method, *'Factorial.factorial(5, 4);'* invokes the *'factorial'* method with arguments 5 and 4.
- ***Output:*** For each number in the input list (5 and 4), it calculates and displays their respective factorials. However, there might be an issue with the output as the variable *'res'* is reset after computing each factorial, so the returned *'res'* value at the end might not accurately represent the factorial of the last number processed.

Problem Statement

Create a class named 'Rectangle' with two data members 'length' and 'breadth' and two methods to print the area and perimeter of the rectangle respectively. Its constructor having parameters for length and breadth is used to initialise length and breadth of the rectangle. Let class 'Square' inherit the 'Rectangle' class. Print the area and perimeter of a rectangle and a square. (length and breadth or side must be taken from the user.) Also write a test driver class (called GeoDiagram) to test inherited as well as new property for child class.

Source Code

```
import java.util.Scanner;

class Rectangle
{
	double length;
	double breadth;

	// Constructor for Rectangle
	Rectangle(double length, double breadth)
	{
		this.length = length;
		this.breadth = breadth;
	}

	// Method to calculate area of Rectangle
	void calArea()
	{
		double area = length * breadth;
		System.out.println("Area: " + area);
	}

	// Method to calculate perimeter of Rectangle
	void calPerimeter()
	{
		double perimeter = 2 * (length + breadth);
		System.out.println("Perimeter: " + perimeter);
	}

	// Method to check if dimensions are negative
	boolean isNegative()
	{
		return length < 0 || breadth < 0;
	}
}

class Square extends Rectangle
{
	// Constructor for Square
	Square(double side)
	{
		super(side, side);
	}
}
```

```
class GeoDiagram
{
	public static void main(String[] args)
	{
		Scanner scanner = new Scanner(System.in);

		// Input for Rectangle
		System.out.print("Enter length of Rectangle: ");
		double lengthR= scanner.nextDouble();
		System.out.print("Enter breadth of Rectangle: ");
		double breadthR = scanner.nextDouble();

		// Creating Rectangle object
		Rectangle rectangle = new Rectangle(lengthR, breadthR);

		// Check if Rectangle dimensions are negative
		if (rectangle.isNegative())
		{
			System.out.println("Length or breadth cannot be negative.");
		}
		else
		{
			System.out.println(" || Rectangle ||");
			// Calculate and display area & perimeter of Rectangle
			rectangle.calArea();
			rectangle.calPerimeter();
		}

		// Input for Square
		System.out.print("\nEnter side of Square: ");
		double sideS = scanner.nextDouble();

		// Creating Square object
		Square square = new Square(sideS);

		// Check if Square side is negative
		if (square.isNegative())
		{
			System.out.println("Side of square cannot be negative.");
		}
		else
		{
			System.out.println(" || Square ||");
			// Calculate and display area & perimeter of Square
			square.calArea();
			square.calPerimeter();
		}
		scanner.close();
	}
}
```

Result

E:\JAVA\College\Test>*javac T3.java*

E:\JAVA\College\Test>*java GeoDiagram*
Enter length of Rectangle: 5
Enter breadth of Rectangle: 10
|| Rectangle ||
Area: 50.0
Perimeter: 30.0

Enter side of Square: 6
|| Square ||
Area: 36.0
Perimeter: 24.0

E:\JAVA\College\Test>*java GeoDiagram*
Enter length of Rectangle: 5
Enter breadth of Rectangle: 10
|| Rectangle ||
Area: 50.0
Perimeter: 30.0

Enter side of Square: -4
Side of square cannot be negative.

E:\JAVA\College\Test>*java GeoDiagram*
Enter length of Rectangle: -3
Enter breadth of Rectangle: 7
Length or breadth cannot be negative.

Enter side of Square: 6
|| Square ||
Area: 36.0
Perimeter: 24.0

E:\JAVA\College\Test>*java GeoDiagram*
Enter length of Rectangle: -3
Enter breadth of Rectangle: 7
Length or breadth cannot be negative.

Enter side of Square: -3
Side of square cannot be negative.

Discussion

- ***Inheritance:*** The *'Rectangle'* class serves as the base class, while the *'Square'* class extends *'Rectangle'*, inheriting its properties and methods.
- ***Constructor Usage:*** Both *'Rectangle'* and *'Square'* classes have constructors to initialize their respective dimensions.
- ***Polymorphism:*** The *'calArea'* and *'calPerimeter'* methods are overridden in the *'Square'* class to accommodate the specific formulas for squares, showcasing polymorphism.
- ***Input Handling:*** The *'GeoDiagram'* class utilizes a 'Scanner' to get user input for rectangle and square dimensions.
- ***Validation:*** It includes a check for negative dimensions using the *'isNegative'* method within each shape class to ensure valid inputs.
- ***Calculation and Display:*** After creating instances of the *'Rectangle'* and *'Square'* classes, the program computes and displays their areas and perimeters based on the user-provided dimensions.

Problem Statement

Create a class named 'Calculator' with two data members int: x and int: y and four methods to print the result of addition, subtraction, multiplication, and division of the x and y respectively. Let class 'MyCalculator' inherit the 'Calculator' class, and it has extra property to find square of a given number. (all instance variables must be taken from the user.) Also write a test driver class (called MyCalculation) to test inherited as well as new property for child class.

Source Code

```
import java.util.Scanner;

class Calculator
{
	int x;
	int y;

	Calculator(int x, int y)
	{
		this.x = x;
		this.y = y;
	}

	//Method to perform addition
	void addition()
	{
		System.out.println("Addition result: " + (x + y));
	}

	//Method to perform subtraction
	void subtraction()
	{
		System.out.println("Subtraction result: " + (x - y));
	}

	//Method to perform multiplication
	void multiplication()
	{
		System.out.println("Multiplication result: " + (x * y));
	}

	//Method to perform division
	void division()
	{
		if (y != 0)
		{
			System.out.println("Division result: " + ((double) x / y));
		}
		else
		{
			System.out.println("Cannot divide by zero.");
		}
	}
}
```

```
class MyCalculator extends Calculator
{
	MyCalculator(int x, int y)
	{
		super(x, y);
	}

	//Method to find squares
	void square()
	{
		System.out.println("Square of " + x + ": " + (x * x));
		System.out.println("Square of " + y + ": " + (y * y));
	}
}

class MyCalculation
{
	public static void main(String[] args)
	{
		Scanner scanner = new Scanner(System.in);

		//Input for x and y values
		System.out.print("Enter value for x: ");
		int x = scanner.nextInt();
		System.out.print("Enter value for y: ");
		int y = scanner.nextInt();

		// Create an instance of MyCalculator
		MyCalculator myCalculator = new MyCalculator(x, y);

		// Perform operations using MyCalculator methods
		myCalculator.addition();
		myCalculator.subtraction();
		myCalculator.multiplication();
		myCalculator.division();
		myCalculator.square();

		scanner.close();
	}
}
```

Result

E:\JAVA\College>*javac A33.java*

E:\JAVA\College>*java MyCalculation*
Enter value for x: 54
Enter value for y: 26
Addition result: 80
Subtraction result: 28
Multiplication result: 1404
Division result: 2.076923076923077
Square of 54: 2916
Square of 26: 676

E:\JAVA\College>*java MyCalculation*
Enter value for x: 86
Enter value for y: -57
Addition result: 29
Subtraction result: 143
Multiplication result: -4902
Division result: -1.5087719298245614
Square of 86: 7396
Square of -57: 3249

E:\JAVA\College>*java MyCalculation*
Enter value for x: -14
Enter value for y: -52
Addition result: -66
Subtraction result: 38
Multiplication result: 728
Division result: 0.2692307692307692
Square of -14: 196
Square of -52: 2704

E:\JAVA\College>*java MyCalculation*
Enter value for x: -8
Enter value for y: 0
Addition result: -8
Subtraction result: -8
Multiplication result: 0
Cannot divide by zero.
Square of -8: 64
Square of 0: 0

E:\JAVA\College>*java MyCalculation*
Enter value for x: 0
Enter value for y: 0
Addition result: 0
Subtraction result: 0
Multiplication result: 0
Cannot divide by zero.
Square of 0: 0
Square of 0: 0

Discussion

- ***Calculator Class:*** Contains methods for addition, subtraction, multiplication, and division, performing operations on two integer values *'x'* and *'y'*.
- ***Inheritance:*** The *'MyCalculator'* class extends *'Calculator'*, inheriting its methods and constructor.
- ***Additional Method:*** *'MyCalculator'* introduces a new method, *'square()'*, which calculates and displays the squares of the input values *'x'* and *'y'*.
- ***Input Handling:*** Utilizes a *'Scanner'* to prompt the user for integer inputs for *'x'* and *'y'* in the *'MyCalculation'* class.
- ***Result Display:*** After creating an instance of *'MyCalculator'*, the program computes and displays the results of addition, subtraction, multiplication, division (with a check to avoid division by zero), and the squares of the entered values.

Problem Statement

Create a superclass 'Person' and two subclasses 'Student' and 'Staff'. The following are the instance variables and methods: For 'Person' instance variables- name:String, address:String. Initiate variable through constructor, incorporate one method setPerson() that updates Person variables , another method tostring() that shows Person details.

For 'Student' sub class instance variables: program:String, year:String, fees:double. Initiate both 'Student' and 'Person' variables through constructor, incorporate one method setStudent() that updates both student and 'Person' data, another method tostring() that shows 'Person-Student' details as name=, address=, program=, year=, fees=. [input:- program = B.A./B.Sc/B.Com and year = first/second/third]

For 'Staff' subclass instance variables: college:String, pay:double. Initiate both 'Staff' and 'Person' variables through constructor, incorporate one method setStaff() that updates both 'staff' and 'Person' data, another method tostring() that shows 'Person-Staff' details as name=, address=, college=, pays=. Write the classes and a test driver main class to test all functions mentioned above.

Source Code

```
import java.util.Scanner;

// Class defining a Person with name and address
class Person
{
	String name;
	String address;

	Person(String name, String address)
	{
		this.name = name;
		this.address = address;
	}

	void setPerson(String name, String address)
	{
		this.name = name;
		this.address = address;
	}

	public String toString()
	{
		return " Name: " + name + "\n Address: " + address;
	}
}

// Class defining a Student, extending the Person class
class Student extends Person
{
	String program;
	String year;
	double fees;
```

```
	Student(String name, String address, String program, String year, double fees)
	{
		super(name, address);
		this.program = program;
		this.year = year;
		this.fees = fees;
	}

	void setStudent(String name, String address, String program, String year, double fees)
	{
		super.setPerson(name, address);
		this.program = program;
		this.year = year;
		this.fees = fees;
	}

	public String toString()
	{
		return super.toString() + "\n Program: " + program + "\n Year: " + year + "\n Fees: " + fees;
	}
}

// Class defining a Staff, extending the Person class
class Staff extends Person
{
	String college;
	double pay;

	public Staff(String name, String address, String college, double pay)
	{
		super(name, address);
		this.college = college;
		this.pay = pay;
	}

	void setStaff(String name, String address, String college, double pay)
	{
		super.setPerson(name, address);
		this.college = college;
		this.pay = pay;
	}

	public String toString()
	{
		return super.toString() + "\n College: " + college + "\n Pay: " + pay;
	}
}

// Main class College to handle user input and object creation
class College
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);
```

```
                while (true)                    // Displaying menu options
                {
                        System.out.print("\n");
                        System.out.println(" 1. Input student details");
                        System.out.println(" 2. Input staff details");
                        System.out.println(" 0. Exit");
                        System.out.print("Enter your choice: ");
                        int choice = sc.nextInt();                          // Getting user choice
                        sc.nextLine();
                        switch (choice)                 // Processing user choice
                        {
                        case 1:                         // Inputting student details
                                System.out.print("Enter student's name: ");
                                String studentName = sc.nextLine();
                                System.out.print("Enter address: ");
                                String studentAddress = sc.nextLine();
                                System.out.print("Enter current program: ");
                                String studentProgram = sc.nextLine();
                                System.out.print("Enter year: ");
                                String studentYear = sc.nextLine();
                                System.out.print("Enter fees: ");
                                double studentFees = sc.nextDouble();
                                System.out.print("--- Person-Student details ---\n");
                                Student student = new Student(studentName, studentAddress, studentProgram,
studentYear, studentFees);
                                System.out.println(student.toString());
                                break;

                        case 2:                         // Inputting staff details
                                System.out.print("Enter staff's name: ");
                                String name = sc.nextLine();
                                System.out.print("Enter address: ");
                                String address = sc.nextLine();
                                System.out.print("Enter college name: ");
                                String college = sc.nextLine();
                                System.out.print("Enter salary: ");
                                double pay = sc.nextDouble();
                                System.out.print("--- Person-Staff details ---\n");
                                Staff staff=new Staff(name,address,college,pay);
                                System.out.println(staff.toString());
                                break;
                        case 0:                         // Exiting the program
                                System.out.println("Exiting program");
                                return;
                        default:                // Handling invalid input
                                System.out.println("Invalid choice. Please enter a valid option.");
                        }
                }
        }
}
```

Result

E:\JAVA\College>*javac A34.java*

E:\JAVA\College>*java College*

1. Input student details
2. Input staff details
0. Exit
Enter your choice: 1
Enter student's name: Ramesh
Enter address: Kolkata
Enter current program: CMSA
Enter year: Third
Enter fees: 5000
--- Person-Student details ---
Name: Ramesh
Address: Kolkata
Program: B.Sc.
Year: Third
Fees: 5000.0

1. Input student details
2. Input staff details
0. Exit
Enter your choice: 0
Exiting program

E:\JAVA\College>*java College*

1. Input student details
2. Input staff details
0. Exit
Enter your choice: 2
Enter staff's name: Sujoy
Enter address: Garia
Enter college name: ABC College
Enter salay: 50000
--- Person-Staff details ---
Name: Sujoy
Address: Garia
College: ABC College
Pay: 50000.0

1. Input student details
2. Input staff details
0. Exit
Enter your choice: 0
Exiting program

E:\JAVA\College>*java College*

1. Input student details
2. Input staff details
0. Exit
Enter your choice: 3
Invalid choice. Please enter a valid option.

1. Input student details
2. Input staff details
0. Exit
Enter your choice: 0
Exiting program

Discussion

- **Person Class**
  - Defines a person with *name* and *address* attributes. Provides a method to set these attributes and a *toString()* method to display the person's details.

- **Student Class**
  - Extends the *Person* class. Adds attributes specific to a student such as *program*, *year*, and *fees*. Overrides the *toString()* method to display student-specific details along with general person details.

- **Staff Class**
  - Also extends the *Person* class. Contains attributes like *college* and *pay*. Overrides the *toString()* method to display staff-specific details along with general person details.

- **College Class (Driver Class)**
  - Uses a menu-driven system to interact with the user. Allows input of either *student* or *staff* details based on user choice. Creates objects of Student or Staff classes depending on user input and displays their details using the overridden *toString()* methods.

Problem Statement

Create a base class 'Cube' having instance variable side:double. Initiate variable using constructor, a method 'getVolume() : double' that calculates volume and print it. Create a derived class 'Cylinder' having instance variable height:double. Initiate variables of both classes through constructor, override method 'getVolume() : double' to calculate volume of cylinder taking 'side' variable of base class as 'radius' and print it.

Source Code

```
import java.util.Scanner;
class Cube
{
	double side;

	// Constructor for Cube class
	Cube(double side)
	{
		this.side = side;
	}

	// Method to calculate and print volume of the base Cube
	double getVolume()
	{
		return side * side * side;
	}

	// Method to check if the side length is zero or negative
	boolean isZOrN()
	{
		return side <= 0;
	}
}

class Cylinder extends Cube
{
	double height;

	// Constructor for Cylinder class
	Cylinder(double side, double height)
	{
		super(side);
		this.height = height;
	}

	// Overriding method to calculate and print volume of the Cylinder
	@Override
	double getVolume()
	{
		// Using the base class 'side' variable as the radius for the Cylinder
		double radius = super.side;
		return Math.PI * radius * radius * height;
	}
```

```
        // Method to check if the height is zero or negative
        boolean isZOrN()
        {
                return height <= 0;
        }
}

class Sqcy
{
        public static void main(String[] args)
        {
                Scanner sc = new Scanner(System.in);
                // Input for Cube
                double side = getValidInput(sc, "Enter side length of the cube: ");
                if (side > 0)
                {
                        Cube cube = new Cube(side);
                        System.out.println("Volume of Cube: " + cube.getVolume());
                }
                else
                {
                        System.out.println("Invalid input for side length of the cube.");
                }

                // Input for Cylinder
                double cylinderSide = getValidInput(sc, "\nEnter side length of the cylinder: ");
                double height = getValidInput(sc, "Enter height of the cylinder: ");
                if (cylinderSide > 0 && height > 0)
                {
                        Cylinder cylinder = new Cylinder(cylinderSide, height);
                        System.out.println("Volume of Cylinder: " + cylinder.getVolume());
                }
                else
                {
                        System.out.println("Invalid input for side length or height of the cylinder.");
                }

                sc.close();
        }

        // Method to get valid positive input from the user
        private static double getValidInput(Scanner sc, String message)
        {
                double input;
                do
                {
                        System.out.print(message);
                        input = sc.nextDouble();
                        if (input <= 0)
                        {
                                System.out.println("Input must be a positive value.");
                        }
                }
                while (input <= 0);
                return input;
        }
}
```

Result

E:\JAVA\College>*javac A35.java*

E:\JAVA\College>*java Sqcy*
Enter side length of the cube: 12.39
Volume of Cube: 1902.0149190000002

Enter side length of the cylinder: 10
Enter height of the cylinder: 15.7
Volume of Cylinder: 4932.3004661359755

E:\JAVA\College>*java Sqcy*
Enter side length of the cube: -4
Input must be a positive value.
Enter side length of the cube: 0
Input must be a positive value.
Enter side length of the cube: 9
Volume of Cube: 729.0

Enter side length of the cylinder: 10
Enter height of the cylinder: 12
Volume of Cylinder: 3769.9111843077517

E:\JAVA\College>*java Sqcy*
Enter side length of the cube: 6
Volume of Cube: 216.0

Enter side length of the cylinder: -10
Input must be a positive value.

Enter side length of the cylinder: 10
Enter height of the cylinder: 0
Input must be a positive value.
Enter height of the cylinder: 12
Volume of Cylinder: 3769.9111843077517

Discussion

- **Cube Class**
  - Has a constructor that initializes the side length of the square. Includes a method *getVolume()* to calculate the volume of the cube (assuming it's a cube) by cubing the side length. Provides a method *isZOrN()* to check if the side length is zero or negative.

- **Cylinder Class**
  - Extends the *Cube* class, inheriting the side length attribute. Introduces a new attribute *height* for the cylinder. Overrides the *getVolume()* method to calculate the volume of the cylinder using its specific formula (π * radius^2 * height). Includes a method *isZOrN()* to check if the height is zero or negative.

- **Sqcy Class (Driver Class)**
  - Takes user input for the side length of a cube and calculates its volume if the side length is valid. Asks for the side length and height of a cylinder, then computes its volume if both dimensions are valid. Employs a *getValidInput()* method to ensure that user inputs are positive values.

Problem Statement

Consider you are designing vehicles engine with 'speed:int, gear:int'. You can define your engine functionalities 'speedUp(value)' and 'changeGear(value)' in an interface. The class which is implementing the interface should implement all the methods in the interface.

Source Code

```
import java.util.Scanner;
interface Vehicle
{
	void speedUp(int value);
	void gearUp(int value);
}

class VehicleEngine implements Vehicle
{
	int speed, gear;

	// initializes instance variables of VehicleEngine class
	VehicleEngine(int speed, int gear)
	{
		this.speed = speed;
		this.gear = gear;
	}

	// speeds up the speed of the vehicle
	public void speedUp(int value)
	{
		if (value >= 0)
		{
			value -= speed;
			speed += value;
			System.out.println("The speed of your vehicle is increased by " + value + " Km/h and current
speed is: " + speed + " Km/h\n");
		}
		else
		{
			System.out.println(value + " Km/h can't be reached!");
		}
	}

	// gears up the gear of the vehicle
	public void gearUp(int value)
	{
		value -= gear;
		gear += value;
		System.out.println("The gear of your vehicle is increased by " + value + " and current gear is: " + gear +
"\n");
	}
}

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

```
	{
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter the initial speed and gear of vehicle: ");
		int speed = sc.nextInt();		// takes initial speed as the input from the user
		int gear = sc.nextInt();		// takes initial gear as the input from the user
		VehicleEngine ve = new VehicleEngine(speed, gear);		// creates an instance of VehicleEngine class
with user-defined values
		System.out.println("Speed of your vehicle: " + ve.speed + " Km/h");
		System.out.println("Gear of your vehicle: " + ve.gear);
		boolean flag = true;
		if (speed < 0 || gear < 0)
		{
			System.out.println("Invalid input: initial speed or gear of the vehicle can't be negative");
			flag = false;
		}
		while (flag)
		{
			System.out.println("1. Change the speed of Vehicle\n2. Change the gear of Vehicle\n0.
Continue with current speed and gear");
			System.out.print("Enter your choice: ");
			int choice = sc.nextInt();
			switch (choice)
			{
			case 1:
				System.out.print("Enter the speed you want to reach: ");
				speed = sc.nextInt();
				if (speed >= 0)
				ve.speedUp(speed);
				else
				System.out.println(speed + " Km/h can't be reached!");
				break;
			case 2:
				System.out.print("Enter the gear you want to reach between 1 to 5: ");
				gear = sc.nextInt();
				if (gear < 6 && gear > 0)
				ve.gearUp(gear);
				else
				System.out.println("Gear " + gear + " can't be reached!");
				break;
			case 0:
				flag = false;
				break;
			default:
				System.out.println("Invalid input: Try again!");
			}
		}
		sc.close();
	}
}
```

Result

E:\JAVA\College>*javac A36.java*

E:\JAVA\College>*java Car*
Enter the initial speed and gear of vehicle: 30 2
Speed of your vehicle: 30 Km/h
Gear of your vehicle: 2
1. Change the speed of Vehicle
2. Change the gear of Vehicle
0. Continue with current speed and gear
Enter your choice: 1
Enter the speed you want to reach: 20
The speed of your vehicle is increased by -10 Km/h and current speed is: 20 Km/h

1. Change the speed of Vehicle
2. Change the gear of Vehicle
0. Continue with current speed and gear
Enter your choice: 0

E:\JAVA\College>*java Car*
Enter the initial speed and gear of vehicle: 60 3
Speed of your vehicle: 60 Km/h
Gear of your vehicle: 3
1. Change the speed of Vehicle
2. Change the gear of Vehicle
0. Continue with current speed and gear
Enter your choice: 2
Enter the gear you want to reach between 1 to 5: 4
The gear of your vehicle is increased by 1 and current gear is: 4

1. Change the speed of Vehicle
2. Change the gear of Vehicle
0. Continue with current speed and gear
Enter your choice: 0

E:\JAVA\College>*java Car*
Enter the initial speed and gear of vehicle: -40 0
Speed of your vehicle: -40 Km/h
Gear of your vehicle: 0
Invalid input: initial speed or gear of the vehicle can't be negative

E:\JAVA\College>*java Car*
Enter the initial speed and gear of vehicle: 0 0
Speed of your vehicle: 0 Km/h
Gear of your vehicle: 0
1. Change the speed of Vehicle
2. Change the gear of Vehicle
0. Continue with current speed and gear
Enter your choice: 1
Enter the speed you want to reach: 60
The speed of your vehicle is increased by 60 Km/h and current speed is: 60 Km/h

1. Change the speed of Vehicle
2. Change the gear of Vehicle
0. Continue with current speed and gear
Enter your choice: 2

Enter the gear you want to reach between 1 to 5: 4
The gear of your vehicle is increased by 4 and current gear is: 4

1. Change the speed of Vehicle
2. Change the gear of Vehicle
0. Continue with current speed and gear
Enter your choice: 0

E:\JAVA\College>*java Car*
Enter the initial speed and gear of vehicle: 60 4
Speed of your vehicle: 60 Km/h
Gear of your vehicle: 4
1. Change the speed of Vehicle
2. Change the gear of Vehicle
0. Continue with current speed and gear
Enter your choice: 2
1. Change the speed of Vehicle
2. Change the gear of Vehicle
0. Continue with current speed and gear
Enter your choice: 1
Enter the speed you want to reach: -40
-40 Km/h can't be reached!
Enter the gear you want to reach between 1 to 5: 6
Gear 6 can't be reached!
1. Change the speed of Vehicle
2. Change the gear of Vehicle
0. Continue with current speed and gear
Enter your choice: 3
Invalid input: Try again!
1. Change the speed of Vehicle
2. Change the gear of Vehicle
0. Continue with current speed and gear
Enter your choice: 0

Discussion

- **Vehicle Interface**
  - Defines the expected behavior for a vehicle using the *speedUp* and *gearUp* methods. Provides a blueprint for any class representing a vehicle to implement these methods.

- **VehicleEngine Class**
  - Implements the *Vehicle* interface, specifying the functionality for altering speed and gear. Manages the current speed and gear of the vehicle as instance variables. Offers methods to modify these attributes based on user input.

- **Car Class (Driver Class)**
  - Acts as the main driver class, interacting with the user through the console. Takes initial speed and gear inputs, ensuring they are valid (non-negative). Provides a menu-driven interface to change the vehicle's speed and gear based on user choices. Implements a loop to continuously prompt the user until they decide to exit.

Problem Statement

Create an abstract class employee (instance variables: emp_no:integer,name:String) , having its properties and abstract function for calculating net salary and displaying the information.

Derive manager(instance variables: emp_no:integer, name:String, salary:Double) method net salary which calculates the pf (7.5% of basic) and allowances [DA (80% of basic), HRA (5% of basic), Medical (3% of basic)] on the salary and clerk class(instance variables: emp_no:integer, name:String, salary:Double) method net salary which calculates the pf (7.5% of basic) and allowances [DA (50% of basic), HRA (5% of basic), Medical (3% of basic)] on the salary.

From this abstract class and implement the abstract method net salary and override the display method.

Source Code

```
import java.util.Scanner;

// Abstract class Employee
abstract class Employee
{
	int emp_no;
	String name;
	Employee(int emp_no, String name)
	{
		this.emp_no = emp_no;
		this.name = name;
	}

	// Abstract method to calculate net salary (to be implemented by subclasses)
	abstract double calNetSalary();

	// Method to validate salary input
	static double checkSalary(Scanner sc)
	{
		double salary;
		do
		{
			System.out.print("Salary: ");
			salary = sc.nextDouble();
			if (salary <= 0)
			{
				System.out.println("Salary cannot be negative or zero. Please enter a valid salary.");
			}
		}
		while (salary <= 0);
			return salary;
	}

	// Method to display employee information
	void displayInfo(double salary, double pf, double da, double hra, double medical)
	{
		double netSalary = salary + pf + da + hra + medical;
		System.out.println("Employee ID: " + emp_no);
```

```
                System.out.println("Employee Name: " + name);
                System.out.println("Basic Salary: " + salary);
                System.out.println("PF: " + pf);
                System.out.println("DA: " + da);
                System.out.println("HRA: " + hra);
                System.out.println("Medical Allowance: " + medical);
                System.out.println("Net Salary: " + netSalary);
        }
}

// Manager class inheriting from Employee
class Manager extends Employee
{
        double salary;
        Manager(int emp_no, String name, double salary)
        {
                super(emp_no, name);
                this.salary = salary;
        }

        // Override method to calculate net salary for Manager
        @Override
        double calNetSalary()
        {
                double pf = 0.075 * salary;
                double da = 0.8 * salary;
                double hra = 0.05 * salary;
                double medical = 0.03 * salary;
                displayInfo(salary, pf, da, hra, medical);
                return salary + pf + da + hra + medical;
        }
}

// Clerk class inheriting from Employee
class Clerk extends Employee
{
        double salary;
        Clerk(int emp_no, String name, double salary)
        {
                super(emp_no, name);
                this.salary = salary;
        }

        // Override method to calculate net salary for Clerk
        @Override
        double calNetSalary()
        {
                double pf = 0.075 * salary;
                double da = 0.5 * salary;
                double hra = 0.05 * salary;
                double medical = 0.03 * salary;
                displayInfo(salary, pf, da, hra, medical);
                return salary + pf + da + hra + medical;
        }
}
```

```
// Main class
class Office
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);

		// Input for Manager details
		System.out.println(" >> Enter details for Manager >>");
		System.out.print("Employee Number: ");
		int mgEmpNo = sc.nextInt();
		sc.nextLine(); // Consume newline
		System.out.print("Name: ");
		String mgName = sc.nextLine();
		double mgSalary = Employee.checkSalary(sc);
		Manager mg = new Manager(mgEmpNo, mgName, mgSalary);

		// Input for Clerk details
		System.out.println("\n >> Enter details for Clerk >>");
		System.out.print("Employee Number: ");
		int cEmpNo = sc.nextInt();
		sc.nextLine(); // Consume newline
		System.out.print("Name: ");
		String cName = sc.nextLine();
		double cSalary = Employee.checkSalary(sc);
		Clerk c = new Clerk(cEmpNo, cName, cSalary);

		// Display Manager information
		System.out.println("\n :: Manager Information ::");
		mg.calNetSalary();

		// Display Clerk information
		System.out.println("\n :: Clerk Information ::");
		c.calNetSalary();

		sc.close();
	}
}
```

Result

E:\JAVA\College>*javac A37.java*

E:\JAVA\College>*java Office*
>> Enter details for Manager >>
Employee Number: 45861
Name: S. Roy
Salary: 75000

>> Enter details for Clerk >>
Employee Number: 8456
Name: R. Sen
Salary: 35000

:: Manager Information ::
Employee ID: 45861
Employee Name: S. Roy
Basic Salary: 75000.0
PF: 5625.0
DA: 60000.0
HRA: 3750.0
Medical Allowance: 2250.0
Net Salary: 146625.0

:: Clerk Information ::
Employee ID: 8456
Employee Name: R. Sen
Basic Salary: 35000.0
PF: 2625.0
DA: 17500.0
HRA: 1750.0
Medical Allowance: 1050.0
Net Salary: 57925.0

E:\JAVA\College>*java Office*
>> Enter details for Manager >>
Employee Number: 545
Name: Dinesh
Salary: 0
Salary cannot be negative or zero. Please enter a valid salary.
Salary: 50000

>> Enter details for Clerk >>
Employee Number: 568
Name: Akash
Salary: -20000
Salary cannot be negative or zero. Please enter a valid salary.
Salary: 25000

:: Manager Information ::
Employee ID: 545
Employee Name: Dinesh
Basic Salary: 50000.0
PF: 3750.0
DA: 40000.0
HRA: 2500.0
Medical Allowance: 1500.0
Net Salary: 97750.0

:: Clerk Information ::
Employee ID: 568
Employee Name: Akash
Basic Salary: 25000.0
PF: 1875.0
DA: 12500.0
HRA: 1250.0
Medical Allowance: 750.0
Net Salary: 41375.0

```
E:\JAVA\College>java Office
 >> Enter details for Manager >>
Employee Number: 656
Name: Himesh
Salary: 750000

 >> Enter details for Clerk >>
Employee Number: 6854
Name: Harish
Salary: 150000

 :: Manager Information ::
Employee ID: 656
Employee Name: Himesh
Basic Salary: 750000.0
PF: 56250.0
DA: 600000.0
HRA: 37500.0
Medical Allowance: 22500.0
Net Salary: 1466250.0

 :: Clerk Information ::
Employee ID: 6854
Employee Name: Harish
Basic Salary: 150000.0
PF: 11250.0
DA: 75000.0
HRA: 7500.0
Medical Allowance: 4500.0
Net Salary: 248250.0
```

Discussion

- **Employee Class**
  - *Employee* serves as an abstract class that defines common attributes and methods for all employees within the organization. Holds *emp_no* (Employee Number) and name (Name) as instance variables. Initializes *emp_no* and *name* for an employee. Contains the abstract method *calNetSalary()* to calculate the net salary (to be implemented by subclasses). Provides *checkSalary()* to validate and ensure that the entered salary is valid. *displayInfo()* displays employee details such as ID, name, basic salary, allowances, and net salary.

- **Manager Class**
  - *Manager* extends the *Employee* class and specializes in managing roles. Has an additional attribute *salary* to store the manager's salary. Initializes the manager object with an employee number, name, and salary. Implements the *calNetSalary()* method specific to managers, calculating PF, DA, HRA, Medical allowances, and the net salary.

- **Clerk Class**
  - *Clerk* extends the *Employee* class and represents clerical staff members. Similar to the Manager class, it has an added attribute *salary* for the clerk's salary. Initializes the clerk object with an employee number, name, and salary. Implements the *calNetSalary()* method specific to clerks, calculating PF, DA, HRA, Medical allowances, and the net salary.

- **Office Class**
  - *Office* serves as the main entry point for the program. Initializes a *Scanner* object to receive user input. Gathers details for a manager and a clerk - employee number, name, and salary. Creates instances of *Manager* and *Clerk* classes using the provided details. Prints out the calculated net salary information for both the manager and clerk.

Problem Statement

Write a program in java that handles both 'ArrayIndexOutOfBoundsException' and 'ArithmeticException'.

Source Code

```
import java.util.Scanner;

class ExceptionHandling
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);
		try
		{
			// Handling ArrayIndexOutOfBoundsException
			System.out.print("Enter the size of the array: ");
			int size = sc.nextInt();
			int[] arr = new int[size];

			System.out.println("Enter elements for the array:");
			for (int i = 0; i < size; i++)
			{
				// Taking user input for array elements
				System.out.print("Enter element " + i + ": ");
				arr[i] = sc.nextInt();
			}

			System.out.print("Enter the index of the dividend in the array: ");
			int dividendIndex = sc.nextInt();
			int dividend = arr[dividendIndex];

			System.out.print("Enter the index of the divisor in the array: ");
			int divisorIndex = sc.nextInt();
			int divisor = arr[divisorIndex];

			int result = dividend / divisor;
			System.out.println("Result of division: " + result);
		}
		catch (ArithmeticException e)
		{
			// Handling ArithmeticException
			System.out.println(">>>> Message: " + e.getMessage());
		}
		catch (ArrayIndexOutOfBoundsException e)
		{
			// Handling ArrayIndexOutOfBoundsException
			System.out.println(">>>> Message: " + e.getMessage());
		}
	}
}
```

Result

E:\JAVA\College>*javac A38-1.java*

E:\JAVA\College>*java ExceptionHandling*
Enter the size of the array: 5
Enter elements for the array:
Enter element 0: 15
Enter element 1: 0
Enter element 2: 75
Enter element 3: 36
Enter element 4: 40
Enter the index of the dividend in the array: 2
Enter the index of the divisor in the array: 0
Result of division: 5

E:\JAVA\College>*java ExceptionHandling*
Enter the size of the array: 5
Enter elements for the array:
Enter element 0: 15
Enter element 1: 0
Enter element 2: 75
Enter element 3: 36
Enter element 4: 40
Enter the index of the dividend in the array: 5
>>>> Message: Index 5 out of bounds for length 5

E:\JAVA\College>*java ExceptionHandling*
Enter the size of the array: 5
Enter elements for the array:
Enter element 0: 15
Enter element 1: 0
Enter element 2: 75
Enter element 3: 36
Enter element 4: 40
Enter the index of the dividend in the array: 4
Enter the index of the divisor in the array: 1
>>>> Message: / by zero

E:\JAVA\College>*java ExceptionHandling*
Enter the size of the array: 5
Enter elements for the array:
Enter element 0: 15
Enter element 1: 0
Enter element 2: 75
Enter element 3: 36
Enter element 4: 40
Enter the index of the dividend in the array: 0
Enter the index of the divisor in the array: -1
>>>> Message: Index -1 out of bounds for length 5

E:\JAVA\College>*java ExceptionHandling*
Enter the size of the array: 5
Enter elements for the array:
Enter element 0: -15
Enter element 1: 0
Enter element 2: 75
Enter element 3: 36
Enter element 4: 40
Enter the index of the dividend in the array: 2
Enter the index of the divisor in the array: 0
Result of division: -5

Discussion

- ***Array Creation and Input***
  - It prompts the user to input the size of an array and then populate the array with user-defined elements. User-provided values populate the array, setting the stage for potential exceptions due to accessing indices outside the array bounds.

- ***Division from Array Elements***
  - It prompts the user to input indices for the dividend and divisor within the array. Using the array elements at the specified indices, it performs division and attempts to calculate the result.

- ***Exception Handling***
  - The program employs a try-catch block to manage exceptions that might arise during runtime. Specifically, it catches *ArithmeticException* and *ArrayIndexOutOfBoundsException*.

- ***ArithmeticException Handling***
  - If a division by zero occurs during the calculation (*divisor* is 0), it catches and handles the *ArithmeticException*, displaying an error message.

- ***ArrayIndexOutOfBoundsException Handling***
  - It catches the *ArrayIndexOutOfBoundsException* that might arise if the user specifies indices outside the bounds of the array. This is crucial to prevent crashes due to invalid array access.

- ***Error Messaging***
  - For both exceptions, the program outputs informative messages (*e.getMessage()*) to indicate the specific type of error encountered.

Problem Statement

Write a program to create your own exception as 'NegativeValueException' whenever negative values are put in an array.

Source Code

```
import java.util.Scanner;

// Custom exception class for negative values
class NegativeValueException extends Exception
{
	NegativeValueException(String message)
	{
		super(message);
	}
}

class CustomException
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);
		try
		{
			// Getting the size of the array from the user
			int size = checkArraySize(sc);
			int[] arr = new int[size];

			// Getting user input for array elements
			System.out.println("Enter elements for the array (negative values will throw an exception):");
			for (int i = 0; i < size; i++)
			{
				System.out.print("Enter element " + i + ": ");
				int value = sc.nextInt();

				// Checking if the input value is negative, throwing custom exception if negative
				if (value < 0)
				{
					throw new NegativeValueException(" >>>> Negative value not allowed in the
array");
				}

				arr[i] = value;
			}

			// Displaying array elements if all inputs are valid
			System.out.println("Array elements:");
			for (int element : arr)
			{
				System.out.print(element + " ");
			}
```

```
                }
                catch (NegativeValueException e)
                {
                        // Catching the custom exception and displaying a message
                        System.out.println(e.getMessage());
                }
        }

        // Method to check if array size is greater than zero
        private static int checkArraySize(Scanner sc)
        {
                int size;
                do
                {
                        System.out.print("Enter the size of the array: ");
                        size = sc.nextInt();
                        if (size <= 0)
                        {
                                System.out.println(" >>>> Array size should be greater than zero.");
                        }
                }
                while (size <= 0);
                return size;
        }
}
```

Result

E:\JAVA\College>*javac A39.java*

E:\JAVA\College>*java CustomException*
Enter the size of the array: 0
>>>> Array size should be greater than zero.
Enter the size of the array: 4
Enter elements for the array (negative values will throw an exception):
Enter element 0: 5
Enter element 1: 6
Enter element 2: 4
Enter element 3: 1
Array elements:
5 6 4 1

E:\JAVA\College>*java CustomException*
Enter the size of the array: -4
>>>> Array size should be greater than zero.
Enter the size of the array: 4
Enter elements for the array (negative values will throw an exception):
Enter element 0: 8
Enter element 1: 6
Enter element 2: -1
>>>> Negative value not allowed in the array

E:\JAVA\College>*java CustomException*
Enter the size of the array: 5
Enter elements for the array (negative values will throw an exception):
Enter element 0: 8
Enter element 1: 9
Enter element 2: 4
Enter element 3: 6
Enter element 4: 2
Array elements:
8 9 4 6 2

E:\JAVA\College>*java CustomException*
Enter the size of the array: 4
Enter elements for the array (negative values will throw an exception):
Enter element 0: 8
Enter element 1: 6
Enter element 2: 1
Enter element 3: -7
>>>> Negative value not allowed in the array

E:\JAVA\College>*java CustomException*
Enter the size of the array: 4
Enter elements for the array (negative values will throw an exception):
Enter element 0: 9
Enter element 1: -8
>>>> Negative value not allowed in the array

Discussion

- ***Array Input and Exception Handling***
  - The program prompts the user to enter the size of the array. It then creates an array of integers based on the entered size. A loop is used to populate the array with user-inputted integers.
  - If the user inputs a negative value, a custom *'NegativeValueException'* is thrown.

- ***Try-Catch Block***
  - The array creation and element population are enclosed within a try block. Inside the try block, user input for array size and elements is attempted. If any part of the try block throws a *'NegativeValueException'*, the catch block catches it.
  - The catch block handles the exception by displaying a message to the user.

- **Custom Exception Handling**
  - The *'NegativeValueException'* is a custom exception class. It extends the base *'Exception'* class, allowing for the creation of specific exception types. When a negative value is encountered during array element input, this exception is thrown.

- ***Array Size Validation***
  - The *'checkArraySize'* method ensures the entered array size is valid. It uses a do-while loop to repeatedly prompt the user for a size greater than zero until a valid input is given.

- ***Flow of Execution***
  - The program starts by asking for the size of the array. If a valid size is provided, it proceeds to take input for array elements. Each element input is checked for negativity.
  - If a negative value is encountered, it throws a custom exception, and the catch block handles it by displaying an error message.

❖ **Control Flow**

- The try block executes the code that might throw exceptions. If any exception occurs within the try block, the catch block catches it, preventing the program from terminating abruptly. In this case, the catch block catches the custom *'NegativeValueException'* and displays a message to notify the user about the prohibited input.

Problem Statement

Write a program in java to display multiplication-tables for given two numbers simultaneously.

Source Code

```
import java.util.Scanner;

// Define a class to generate multiplication tables for a given number
class Multi extends Thread
{
	int num;

	// Constructor to initialize the number
	Multi(int num)
	{
		this.num = num;
	}

	// Run method to display the multiplication table
	public void run()
	{
		System.out.println("Multiplication table for " + num + ":");
		for (int i = 1; i <= 10; i++)
		{
			System.out.println(num + " x " + i + " = " + (num * i));
			try
			{
				Thread.sleep(500);		// Adding a small delay for readability
			}
			catch (InterruptedException e)
			{
				e.printStackTrace();
			}
		}
	}
}

// Main Class
class Th
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);

		try
		{
			// Ask user for the first number
			System.out.print("Enter first number: ");
			int n1 = sc.nextInt();

			// Ask user for the second number
			System.out.print("Enter second number: ");
			int n2 = sc.nextInt();

			sc.close();			// Close the scanner
```

```
                        if (n1 <= 0 || n2 <= 0)
                        {
                                System.out.println("Please enter positive numbers greater than zero.");
                        }
                        else
                        {
                                // Create instances of Multiplication Tables for each number
                                Multi t1 = new Multi(n1);
                                Multi t2 = new Multi(n2);

                                // Start threads to display multiplication tables
                                t1.start();
                                t2.start();
                        }
                }
                catch (Exception e)
                {
                        System.out.println("Please enter valid numbers.");
                }
        }
}
```

Result

E:\JAVA\College>*javac A40.java*

E:\JAVA\College>*java Th*
Enter first number: 5
Enter second number: 3
Multiplication table for 3:
Multiplication table for 5:
5 x 1 = 5
3 x 1 = 3
5 x 2 = 10
3 x 2 = 6
5 x 3 = 15
3 x 3 = 9
5 x 4 = 20
3 x 4 = 12
5 x 5 = 25
3 x 5 = 15
5 x 6 = 30
3 x 6 = 18
5 x 7 = 35
3 x 7 = 21
5 x 8 = 40
3 x 8 = 24
5 x 9 = 45
3 x 9 = 27
3 x 10 = 30
5 x 10 = 50

E:\JAVA\College>*java Th*
Enter first number: 2
Enter second number: 4
Multiplication table for 4:
Multiplication table for 2:
4 x 1 = 4
2 x 1 = 2
2 x 2 = 4
4 x 2 = 8
2 x 3 = 6
4 x 3 = 12
4 x 4 = 16
2 x 4 = 8
2 x 5 = 10
4 x 5 = 20
2 x 6 = 12
4 x 6 = 24
2 x 7 = 14
4 x 7 = 28
2 x 8 = 16
4 x 8 = 32
2 x 9 = 18
4 x 9 = 36
2 x 10 = 20
4 x 10 = 40

E:\JAVA\College>*java Th*
Enter first number: -2
Enter second number: 4
Please enter positive numbers greater than zero.

E:\JAVA\College>*java Th*
Enter first number: A
Please enter valid numbers.

Discussion

- ***Multi Class***
  - This class extends the *Thread* class, indicating that its instances can be executed as threads. It contains a *run()* method that defines the logic to display the multiplication table for a given number. Inside *run(),* a loop runs from 1 to 10, displaying the multiplication table for the given number.

- ***Th Class (Main)***
  - The *Th* class is where the program starts its execution. It contains the *main()* method, the entry point of the program. Uses a *Scanner* to take user input for two numbers.
  - ***Performs input validation***
    - Checks if the entered numbers are positive integers greater than zero. If not, it prompts the user to enter valid numbers.
  - Creates instances of *Multi* for the two input numbers. Starts both threads to display the multiplication tables simultaneously.

- ***Try-Catch Blocks***
  - sc.nextInt() method could throw exceptions if the input is not an integer. The try-catch block captures this exception, allowing the program to handle invalid inputs gracefully. If the entered numbers are not positive integers greater than zero, the program displays a message prompting the user to enter valid numbers. The catch block for *Exception* is a general catch-all for any unexpected errors that might occur during the execution of the program.

- ***Multithreading***
  - The program demonstrates the use of multithreading by creating two instances of *Multi* class and starting both threads simultaneously. Each thread (representing each number) executes independently, displaying the multiplication tables concurrently.

- ***Functionality***
  - It takes user input for two numbers. Validates the input to ensure it meets the criteria (positive integers greater than zero). Displays the multiplication tables for both numbers concurrently using separate threads.

Problem Statement

Write a program in java to create a list of numbers and then sort in ascending order as well as in descending order simultaneously.

Source Code

```
import java.util.Scanner;

class CustomSort
{
	// Method to perform ascending sort
	void ascendingSort(int array[], int n)
	{
		for (int i = 1; i < n; i++)
		{
			int j = i - 1, lock = array[i];
			while (j >= 0 && array[j] > lock)
			{
				array[j + 1] = array[j];
				j--;
			}
			array[j + 1] = lock;
		}
	}

	// Method to perform descending sort
	void descendingSort(int array[], int n)
	{
		for (int i = 1; i < n; i++)
		{
			int j = i - 1, lock = array[i];
			while (j >= 0 && array[j] < lock)
			{
				array[j + 1] = array[j];
				j--;
			}
			array[j + 1] = lock;
		}
	}
}

class MyThread1 extends Thread
{
	CustomSort customSort;
	int array[];

	MyThread1(CustomSort customSort, int array[])
	{
		this.customSort = customSort;
		this.array = array;
	}
	public void run()
	{
		customSort.ascendingSort(array, array.length);
```

```
                System.out.print("Your input array in ascending order:");
                for (int i : array)
                        System.out.print("\t" + i);
                System.out.println();
        }
}

class MyThread2 extends Thread
{
        CustomSort customSort;
        int array[];

        MyThread2(CustomSort customSort, int array[])
        {
                this.customSort = customSort;
                this.array = array;
        }

        public void run()
        {
                customSort.descendingSort(array, array.length);
                System.out.print("Your input array in descending order:");
                for (int i : array)
                        System.out.print("\t" + i);
                System.out.println();
        }
}

class Bubble
{
        public static void main(String[] args)
        {
                Scanner input = new Scanner(System.in);
                System.out.print("Kindly enter the length of your array: ");
                int length = input.nextInt();
                if (length > 0)
                {
                        int array[] = new int[length];
                        // Taking input elements for the array
                        for (int i = 0; i < length; i++)
                        {
                                System.out.print("Enter element no." + (i + 1) + ": ");
                                array[i] = input.nextInt();
                        }
                        // Displaying the input array
                        System.out.print("Your input array:");
                        for (int i : array)
                                System.out.print("\t" + i);
                        System.out.println();

                        // Sorting threads initialization
                        CustomSort sort = new CustomSort();
                        MyThread1 thread1 = new MyThread1(sort, array);
                        MyThread2 thread2 = new MyThread2(sort, array);

                        // Starting ascending sort thread
                        thread1.start();
```

```
                        try
                        {
                                thread1.join();
                        }
                        catch (InterruptedException ie)
                        {
                                System.out.print(ie);
                        }

                        // Starting descending sort thread
                        thread2.start();
                }
                else
                {
                        System.out.println("Invalid input: Array length can't be less than or equal to zero!");
                }
                input.close();
        }
}
```

Result

E:\JAVA\College>*javac A41.java*

E:\JAVA\College>*java Bubble*
Kindly enter the length of your array: 5
Enter element no.1: 5
Enter element no.2: 2
Enter element no.3: 4
Enter element no.4: 6
Enter element no.5: 3
Your input array: 5 2 4 6 3
Your input array in ascending order: 2 3 4 5 6
Your input array in descending order: 6 5 4 3 2

E:\JAVA\College>*java Bubble*
Kindly enter the length of your array: -5
Invalid input: Array length can't be less than or equal to zero!

E:\JAVA\College>*java Bubble*
Kindly enter the length of your array: 5
Enter element no.1: 0
Enter element no.2: -4
Enter element no.3: 5
Enter element no.4: -7
Enter element no.5: 9
Your input array: 0 -4 5 -7 9
Your input array in ascending order: -7 -4 0 5 9
Your input array in descending order: 9 5 0 -4 -7

E:\JAVA\College>*java Bubble*
Kindly enter the length of your array: 0
Invalid input: Array length can't be less than or equal to zero!

Discussion

- ***CustomSort Class***
  - Contains methods *ascendingSort* and *descendingSort*. *ascendingSort* sorts an array in ascending order using an insertion sort algorithm. *descendingSort* sorts an array in descending order using a similar insertion sort algorithm but in reverse order.

- ***MyThread1 Class***
  - Extends the *Thread* class and implements sorting in ascending order. Holds a reference to an instance of *CustomSort* and an array. The run method executes the *ascendingSort* method from *CustomSort*.

- ***MyThread2 Class***
  - Extends the *Thread* class and implements sorting in descending order. Similar to *MyThread1* but uses *descendingSort* method from *CustomSort*.

- ***DriverSort Class***
  - Contains the main method where the program execution begins. Prompts the user to input the length of the array and the elements. Creates an instance of *CustomSort* and two threads (*MyThread1* and *MyThread2*) to perform sorting. Displays the original array and the sorted arrays (ascending and descending).

- ***Thread Synchronization***
  - *MyThread1* uses *join()* to ensure that *MyThread2* starts execution only after *MyThread1* completes its sorting operation. This is for the sake of clarity in output.

Problem Statement

Write a program in java to connect Java and MySQL using JDBC, and display the database-table content in the result.

Java Database Connectivity with 7 Steps

There are 7 steps to connect any java application with the database using JDBC. These steps are as follows:

Step 1: Import the package
Step 2: Load and Register the Driver class
Step 3: Create/Establish connection
Step 4: Create statement
Step 5: Execute queries
Step 6: Process the result
Step 7: Close connection

Source Code

```
import java.sql.*;

public class MysqlCon
{
	public static void main(String[] args)
	{
		try {
			// Load the MySQL JDBC driver
			Class.forName("com.mysql.cj.jdbc.Driver");

			// Establish a connection to the database
			Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/college","root","");

			// Create a statement
			Statement stmt = con.createStatement();

			// Execute a query to retrieve data from the 'student' table
			ResultSet rs = stmt.executeQuery("select * from student");

			// Iterate through the result set and print data
			while(rs.next())
				System.out.println(rs.getInt(1)+"\t"+rs.getString(2));

			// Close the connection
			con.close();
		} catch(Exception e) {
			// Handle any exceptions that occur during the process
			e.printStackTrace();
		}
	}
}
```

Result

ROLL	SNAME
1	RAM MITRA
2	SHYAM HALDER
3	JADHU SARDAR
4	MADHU NASKAR
6	BARUN TIKADAR

Figure 1 : In The Database

Figure 2 : JDBC Connection and display the database in the command prompt

Discussion

- The ***forName()*** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.
- The ***getConnection()*** method of ***DriverManager*** class is used to establish connection with the database.
- The ***createStatement()*** method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.
- The ***executeQuery()*** method of Statement interface is used to execute queries to the database.
- This method returns the object of ***ResultSet*** that can be used to get all the records of a table.
- JDBC By closing connection object statement and ***ResultSet*** will be closed automatically. The ***close()*** method of Connection interface is used to close the connection.

Problem Statement

Write a program in java to insert a new row of a table within a database (connection establishment between Java and MySQL using JDBC), and display the database-table content in the result.

Java Database Connectivity with 7 Steps

There are 7 steps to connect any java application with the database using JDBC. These steps are as follows:

Step 1: Import the package
Step 2: Load and Register the Driver class
Step 3: Create/Establish connection
Step 4: Create statement
Step 5: Execute queries
Step 6: Process the result
Step 7: Close connection

Source Code

```
import java.sql.*;

class MysqlInsert
{
	public static void main(String[] args) throws Exception
	{
		Connection conn = null;

		// Database connection parameters
		String driver = "com.mysql.cj.jdbc.Driver";
		String db = "college";
		String url = "jdbc:mysql://localhost/" + db;
		String user = "root";
		String pass = "";

		// SQL query to insert data into the 'student' table
		String query = "insert into student values (7,'xyz')";

		try
		{
			// Load the MySQL JDBC driver
			Class.forName(driver);

			// Establish a connection to the database
			conn = DriverManager.getConnection(url, user, pass);

			// Create a statement
			Statement st = conn.createStatement();

			// Execute the SQL insert query
			int count = st.executeUpdate(query);
			System.out.println("----" + count + " row(s) affected----\n");

			// Retrieve and print all data from the 'student' table
			ResultSet rs = st.executeQuery("select * from student");
			while (rs.next())
```

```
                {
                        System.out.println(rs.getInt(1) + "\t" + rs.getString(2) + " ");
                }

                // Close the result set, statement, and connection
                rs.close();
                st.close();
                conn.close();
        }
        catch (SQLException e)
        {
                // Handle SQL exceptions
                System.out.println(e.getMessage());
        }
    }
}
```

Result

ROLL	SNAME
1	RAM MITRA
2	SHYAM HALDER
3	JADHU SARDAR
4	MADHU NASKAR
6	BARUN TIKADAR

Figure 1 : Before data Insertion in the Database

ROLL	SNAME
1	RAM MITRA
2	SHYAM HALDER
3	JADHU SARDAR
4	MADHU NASKAR
6	BARUN TIKADAR
7	xyz

Figure 3 : After data Insertion in the database

Figure 2 : Data Insertion and display the database in the command prompt

Discussion

- The ***forName()*** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.
- The ***getConnection()*** method of ***DriverManager*** class is used to establish connection with the database.
- The ***createStatement()*** method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.
- ***executeUpdate(query):*** Alters the database by executing SQL statements like INSERT, UPDATE, or DELETE, returning the count of affected rows.
- ***getString():*** Retrieves a specific column's value as a String from the current row in the ***ResultSet***, useful for fetching string data during result iteration.
- The ***executeQuery()*** method of Statement interface is used to execute queries to the database.
- This method returns the object of ***ResultSet*** that can be used to get all the records of a table.
- JDBC By closing connection object statement and ***ResultSet*** will be closed automatically. The ***close()*** method of Connection interface is used to close the connection.

Problem Statement

Write a program in Java using JDBC to update some information of a table into a database.

Java Database Connectivity with 7 Steps

There are 7 steps to connect any java application with the database using JDBC. These steps are as follows:

Step 1: Import the package
Step 2: Load and Register the Driver class
Step 3: Create/Establish connection
Step 4: Create statement
Step 5: Execute queries
Step 6: Process the result
Step 7: Close connection

Source Code

```
import java.sql.*;

class MysqlUpdate
{
	public static void main(String[] args) throws Exception
	{
		Connection conn = null;

		// Database connection parameters
		String driver = "com.mysql.cj.jdbc.Driver";
		String db = "college";
		String url = "jdbc:mysql://localhost/" + db;
		String user = "root";
		String pass = "";

		// SQL query to update data in the 'STUDENT' table
		String query = "UPDATE STUDENT SET SNAME='PQR' WHERE ROLL=7;";

		try {
			// Load the MySQL JDBC driver
			Class.forName(driver);

			// Establish a connection to the database
			conn = DriverManager.getConnection(url, user, pass);

			// Create a statement
			Statement st = conn.createStatement();

			// Execute the SQL update query
			int count = st.executeUpdate(query);
			System.out.println("----" + count + " row(s) affected----\n");

			// Retrieve and print all data from the 'student' table
			ResultSet rs = st.executeQuery("select * from student");
			while (rs.next())
			{
				System.out.println(rs.getInt(1) + "\t" + rs.getString(2) + " ");
			}
```

```
                    // Close the result set, statement, and connection
                    rs.close();
                    st.close();
                    conn.close();
                }
                catch (SQLException e)
                {
                    // Handle SQL exceptions
                    System.out.println(e.getMessage());
                }
        }
}
```

Result

ROLL	SNAME
1	RAM MITRA
2	SHYAM HALDER
3	JADHU SARDAR
4	MADHU NASKAR
6	BARUN TIKADAR
7	xyz

Figure 1 : Before Updating data in the Database

ROLL	SNAME
1	RAM MITRA
2	SHYAM HALDER
3	JADHU SARDAR
4	MADHU NASKAR
6	BARUN TIKADAR
7	PQR

Figure 3 : After Updating data in the Database

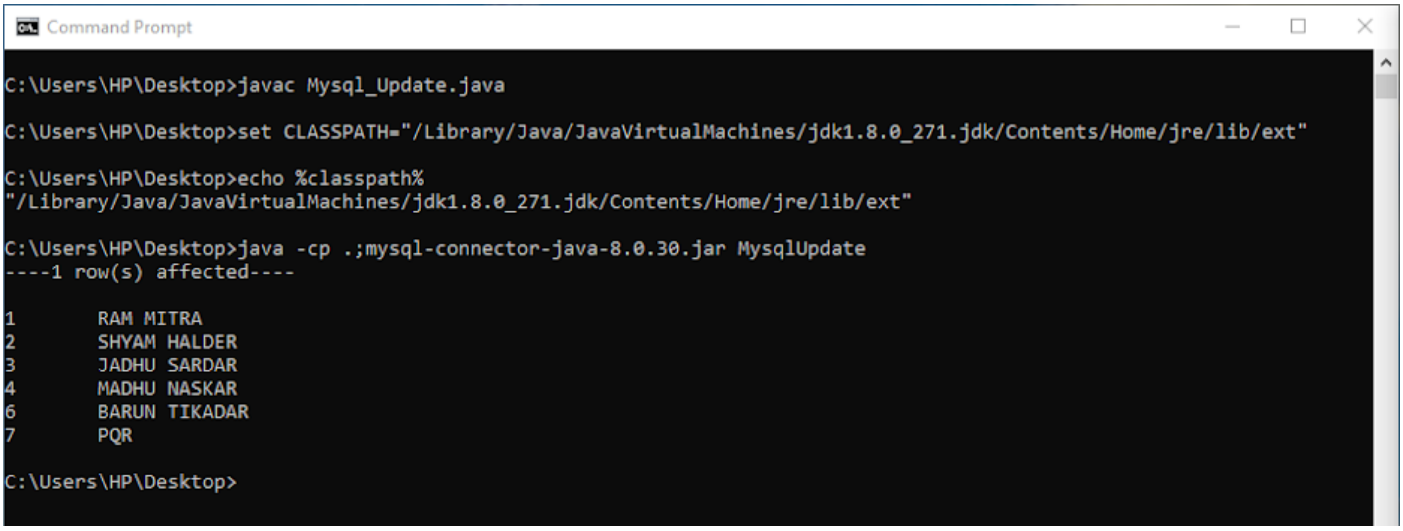

Figure 2 : Data Updating and display the database in the command prompt

Discussion

- The ***forName()*** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.
- The ***getConnection()*** method of ***DriverManager*** class is used to establish connection with the database.
- The ***createStatement()*** method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.
- ***executeUpdate(query):*** Alters the database by executing SQL statements like INSERT, UPDATE, or DELETE, returning the count of affected rows.
- ***getString():*** Retrieves a specific column's value as a String from the current row in the ***ResultSet***, useful for fetching string data during result iteration.
- The ***executeQuery()*** method of Statement interface is used to execute queries to the database.
- This method returns the object of ***ResultSet*** that can be used to get all the records of a table.
- JDBC By closing connection object statement and ***ResultSet*** will be closed automatically. The ***close()*** method of Connection interface is used to close the connection.

Problem Statement

Write a program in Java using JDBC to delete an entry from a table within a database.

Java Database Connectivity with 7 Steps

There are 7 steps to connect any java application with the database using JDBC. These steps are as follows:

Step 1: Import the package
Step 2: Load and Register the Driver class
Step 3: Create/Establish connection
Step 4: Create statement
Step 5: Execute queries
Step 6: Process the result
Step 7: Close connection

Source Code

```
import java.sql.*;

class MysqlUpdate
{
	public static void main(String[] args) throws Exception
	{
		Connection conn = null;

		// Database connection parameters
		String driver = "com.mysql.cj.jdbc.Driver";
		String db = "college";
		String url = "jdbc:mysql://localhost/" + db;
		String user = "root";
		String pass = "";

		// SQL query to delete data in the 'STUDENT' table
		String query = “DELETE FROM STUDENT WHERE ROLL=7;";

		try {
			// Load the MySQL JDBC driver
			Class.forName(driver);

			// Establish a connection to the database
			conn = DriverManager.getConnection(url, user, pass);

			// Create a statement
			Statement st = conn.createStatement();

			// Execute the SQL update query
			int count = st.executeUpdate(query);
			System.out.println("----" + count + " row(s) affected----\n");

			// Retrieve and print all data from the 'student' table
			ResultSet rs = st.executeQuery("select * from student");
			while (rs.next())
			{
				System.out.println(rs.getInt(1) + "\t" + rs.getString(2) + " ");
			}
```

```
                    // Close the result set, statement, and connection
                    rs.close();
                    st.close();
                    conn.close();
                }
                catch (SQLException e)
                {
                    // Handle SQL exceptions
                    System.out.println(e.getMessage());
                }
        }
}
```

Result

ROLL	SNAME
1	RAM MITRA
2	SHYAM HALDER
3	JADHU SARDAR
4	MADHU NASKAR
6	BARUN TIKADAR
7	PQR

Figure 1 : Before Deleting data in the Database

ROLL	SNAME
1	RAM MITRA
2	SHYAM HALDER
3	JADHU SARDAR
4	MADHU NASKAR
6	BARUN TIKADAR

Figure 3 : After Deleting data in the Database

Figure 2 : Data Deleting and display the database in the command prompt

Discussion

- The ***forName()*** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.
- The ***getConnection()*** method of ***DriverManager*** class is used to establish connection with the database.
- The ***createStatement()*** method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.
- ***executeUpdate(query):*** Alters the database by executing SQL statements like INSERT, UPDATE, or DELETE, returning the count of affected rows.
- ***getString():*** Retrieves a specific column's value as a String from the current row in the ***ResultSet***, useful for fetching string data during result iteration.
- The ***executeQuery()*** method of Statement interface is used to execute queries to the database.
- This method returns the object of ***ResultSet*** that can be used to get all the records of a table.
- JDBC By closing connection object statement and ***ResultSet*** will be closed automatically. The ***close()*** method of Connection interface is used to close the connection.