PROJECT FILE

ISC 2025

Subject:

Computer Science

Name	:	Ankit Sharma
Class & Sec	:	XII Science A
Roll Number	:	10
Session	:	2024 – 2025

Certificate of Completion

This is to certify that ____**Ankit**____ ____**Sharma**____ is a bona fide student of XII Science A. He has successfully completed the computer science project in JAVA language during the session 2024 - 25 for the fulfillment of ISC exam by the council.

Visiting Examiner	**Internal Examiner**
Date	**Date**

Index

Sl. No.	Category / Topic	Pg. No.
1	**Number Programs** (Program **1** – **10**)	**03 - 62**
2	**Array and Matrix Programs** (Program **11** – **20**)	**63 - 127**
3	**String Programs** (Program **21** – **30**)	**128 - 191**

Number Programs

Number 1:

A **Prime-Adam** integer is a positive integer (without leading zeros) which is a prime as well as an Adam number.
Prime number: A number which has only two factors, i.e. 1 and the number itself. Example: 2, 3, 5, 7 ... etc.
Adam number: The square of a number and the square of its reverse are reverse to each other.

Example: If n = 13 and reverse of 'n' = 31, then,
$(13)^2 = 169$
$(31)^2 = 961$ which is reverse of 169
thus 13, is an Adam number.

Accept two positive integers m and n, where m is less than n as user input. Display all Prime-Adam integers that are in the range between m and n (both inclusive) and output them along with the frequency, in the format given below:
Test your program with the following data and some random data:

Example 1
INPUT:
m = 5
n = 100

OUTPUT:
THE PRIME-ADAM INTEGERS ARE:
11 13 31
FREQUENCY OF PRIME-ADAM INTEGERS IS: 3

Example 2
INPUT:
m = 100
n = 200

OUTPUT:
THE PRIME-ADAM INTEGERS ARE:
101 103 113
FREQUENCY OF PRIME-ADAM INTEGERS IS: 3

Example 3
INPUT:
m = 50
n = 70

OUTPUT:
THE PRIME-ADAM INTEGERS ARE:
NIL
FREQUENCY OF PRIME-ADAM INTEGERS IS: 0

Example 4
INPUT:
m = 700
n = 450

OUTPUT:
INVALID INPUT

Algorithm:

1. **Input Validation**:
   - Accept the inputs m and n. If m is greater than n, display "INVALID INPUT" and exit the program.

2. **Prime Check**:
   - For every number i in the range from m to n (both inclusive), check if i is a prime number.
   - A number is prime if it has only two factors: 1 and itself. You can check divisibility from 2 to the square root of i.

3. **Reverse and Adam Check**:
   - If i is prime, reverse the number i.
   - Square both i and its reverse.
   - If the square of the reverse of i is the reverse of the square of i, then the number is an Adam number.

4. **Print Prime-Adam Integers**:
   - Print the found Prime-Adam numbers as the loop progresses.
   - Count the frequency of such numbers when one is found.

5. **Print NIL case and frequency**:
   - If there are no Prime-Adam integers (count = 0), output "NIL" and frequency 0.
   - Else output the frequency (count).

Solution:

```
import java.util.Scanner;


public class PrimeAdam {


  // Function to reverse a number
  public static int reverse(int num) {
    int rev = 0;

    while (num > 0) {
      rev = rev * 10 + num % 10;
      num /= 10;
    }
    return rev;
  }


  // Function to check if a number is prime
  public static boolean isPrime(int num) {
    if (num < 2) return false;

    for (int i = 2; i <= Math.sqrt(num); i++) {
      if (num % i == 0) return false;
    }
    return true;
  }


  // Function to check if a number is an Adam number
  public static boolean isAdam(int num) {

    int rev = reverse(num);
    return num * num == reverse(rev * rev);
  }
```

```
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    // Input m and n
    System.out.print("Enter m: ");
    int m = sc.nextInt();
    System.out.print("Enter n: ");
    int n = sc.nextInt();
    sc.close();

    // Check for invalid input
    if (m > n) {
      System.out.println("INVALID INPUT");
      return;
    }

    System.out.println("THE PRIME-ADAM INTEGERS ARE:");

    int count = 0;
    // Loop through the range [m, n]
    for (int i = m; i <= n; i++) {
      // Check if number is prime and Adam and print it
      if (isPrime(i) && isAdam(i)) {
        count++;
        System.out.print(i + " ");
      }
    }

    // Output the results
    if (count == 0) {
      System.out.println("NIL");
    }
    System.out.println("\nFREQUENCY OF PRIME-ADAM INTEGERS IS:
" + count);
  }
}
```

Output:

```
Enter m: 5
Enter n: 100
THE PRIME-ADAM INTEGERS ARE:
11 13 31
FREQUENCY OF PRIME-ADAM INTEGERS IS: 3
```

```
Enter m: 50
Enter n: 500
THE PRIME-ADAM INTEGERS ARE:
101 103 113 211 311
FREQUENCY OF PRIME-ADAM INTEGERS IS: 5
```

```
Enter m: 32
Enter n: 96
THE PRIME-ADAM INTEGERS ARE:
NIL

FREQUENCY OF PRIME-ADAM INTEGERS IS: 0
```

```
Enter m: 130
Enter n: 70
INVALID INPUT
```

Variable Description Table:

Scope	Name	Datatype	Description
main	`m`	`int`	Starting range value input by the user
main	`n`	`int`	Ending range value input by the user
main	`i`	`int`	Loop variable to iterate from m to n
reverse	`rev`	`int`	Stores the reversed value of a number
main	`count`	`int`	Counts the number of Prime-Adam integers found

Number 2:

A **Goldbach** number is a positive even integer that can be expressed as the sum of two odd primes.

Note: All even integer numbers greater than 4 are Goldbach numbers.

Example:
6 = 3 + 3
10 = 3 + 7
10 = 5 + 5
Hence, 6 has one odd prime pair 3 and 3. Similarly, 10 has two odd prime pairs, i.e. 3 and 7, 5 and 5.

Write a program to accept an even integer 'N' where N > 9 and N < 50. Find all the odd prime pairs whose sum is equal to the number 'N'.

Test your program with the following data and some random data:

Example 1

INPUT:
N = 14

OUTPUT:
PRIME PAIRS ARE:
3, 11
7, 7

Example 2

INPUT:
N = 30

OUTPUT:
PRIME PAIRS ARE:
7, 23
11, 19
13, 17

Example 3

INPUT:
N = 17

OUTPUT:
INVALID INPUT. NUMBER IS ODD.

Example 4

INPUT:
N = 126

OUTPUT:
INVALID INPUT. NUMBER OUT OF RANGE.

Algorithm:

1. **Input Validation**:
   - Accept an integer input N.
   - Check if N is within the valid range (i.e., greater than 9 and less than 50).
   - If N is outside the range, print "INVALID INPUT. NUMBER OUT OF RANGE" and terminate.
   - If N is odd, print "INVALID INPUT. NUMBER IS ODD" and terminate.
2. **Prime Pair Search**:
   - Start with the smallest odd prime (a = 3).
   - For each odd prime a less than or equal to N/2, compute b = N - a.
   - Check if both a and b are prime.
   - If they are, print the pair a, b.
3. **Prime Checking**:
   - Implement a function isPrime() to check if a given number is prime. A number is prime if it has exactly two divisors: 1 and itself. i.e. not divisible by any other number.
4. **Looping for Odd Primes**:
   - Increment a by 2 in each iteration to check only odd numbers.
   - Continue the loop until a exceeds N/2.
5. **Output**:
   - Display all valid prime pairs whose sum equals N.

Solution:

```
import java.util.Scanner;


public class GoldbachNumber {

  // Function to check if a number is prime
  public static boolean isPrime(int num) {

    // Prime numbers are greater than 1
    if (num < 2) {
      return false;
    }

    for (int i = 2; i <= num / 2; i++) {
      // If divisible by any num other than 1 and itself
      if (num % i == 0) {
        return false;
      }
    }
    // Prime if not divisible by any num other than 1 and itself
    return true;
  }


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

    // Input the value of N
    System.out.print("ENTER THE VALUE OF N: ");
    int n = in.nextInt();

    in.close();
```

```
    // Validate if N is within the range and even

    if (n <= 9 || n >= 50) {
      System.out.println("INVALID INPUT. NUMBER OUT OF RANGE.");
      return;
    }

    if (n % 2 != 0) {
      System.out.println("INVALID INPUT. NUMBER IS ODD.");
      return;
    }


    // Output prime pairs
    System.out.println("PRIME PAIRS ARE:");

    // Start checking odd primes starting from 3
    for (int a = 3; a <= n / 2; a += 2) {
      // Calculate b as the complement of a
      int b = n - a;

      if (isPrime(a) && isPrime(b)) {
        // Print the prime pair
        System.out.println(a + ", " + b);
      }
    }

  }
}
```

Output:

```
ENTER THE VALUE OF N: 10
PRIME PAIRS ARE:
3, 7
5, 5
```

```
ENTER THE VALUE OF N: 8
INVALID INPUT. NUMBER OUT OF RANGE.
```

```
ENTER THE VALUE OF N: 50
INVALID INPUT. NUMBER OUT OF RANGE.
```

```
ENTER THE VALUE OF N: 25
INVALID INPUT. NUMBER IS ODD.
```

Variable Description Table:

Name	Datatype	Description
`n`	`int`	The input number entered by the user.
`a`	`int`	First odd prime candidate in the pair. Starts at 3.
`b`	`int`	Second odd prime in the pair, calculated as b = n - a.
`c`	`int`	Counter used in isPrime function to count divisors of a number.

Number 3:

A company manufactures packing cartons in four sizes, i.e. cartons to accommodate 6 boxes, 12 boxes, 24 boxes and 48 boxes. Design a program to accept the number of boxes to be packed (N) by the user (maximum up to 1000 boxes) and display the break-up of the cartons used in descending order of capacity (i.e. preference should be given to the highest capacity available, and if boxes left are less than 6, an extra carton of capacity 6 should be used.)

Test your program with the following data and some random data:

Example 1

INPUT:
N = 726

OUTPUT:
48 * 15 = 720
6 * 1 = 6
Remaining boxes = 0
Total number of boxes = 726
Total number of cartons = 16

Example 2

INPUT:
N = 140

OUTPUT:
48 * 2 = 96
24 * 1 = 24
12 * 1 = 12
6 * 1 = 6
Remaining boxes = 2 * 1 = 2
Total number of boxes = 140
Total number of cartons = 6

Example 3

INPUT:
N = 4296

OUTPUT:
INVALID INPUT

Algorithm:

1. **Input**:
   - Read the number of boxes N to be packed.
2. **Validation**:
   - If N is less than 1 or greater than 1000, print "INVALID INPUT" and terminate the program.
3. **Carton Sizes**:
   - Create an array with the available carton sizes [48, 24, 12, 6].
4. **Initialize Variables**:
   - Initialize a variable total to store the total number of cartons used.
   - Initialize t to store the number of boxes left to be packed.
5. **Determine Carton Count**:
   - For each carton size in descending order, calculate how many cartons of that size are needed using integer division (t / carton_size).
   - Subtract the packed boxes from the total boxes using the modulo operator (t = t % carton_size).
   - Add the number of cartons used to the total.
6. **Remaining Boxes**:
   - If there are any boxes left (t != 0), print the remaining boxes, assign one additional carton of size 6, and increment total by 1.
7. **Output**:
   - Print the total number of boxes.
   - Print the total number of cartons used.

Solution:

```
import java.util.Scanner;

public class CartonBoxes {

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

    System.out.print("Enter number of boxes (N): ");
    int n = in.nextInt();

    in.close();

    if (n < 1 || n > 1000) {
      System.out.println("INVALID INPUT");
      return;
    }

    int cartonSizes[] = { 48, 24, 12, 6 };

    int total = 0;
    int t = n;

    // loop through each carton size
    for (int i = 0; i < cartonSizes.length; i++) {

      // find number of cartons needed for this size
      int cartonCount = t / cartonSizes[i];
      t = t % cartonSizes[i];
      total += cartonCount;
```

```
      if (cartonCount != 0) {
        System.out.println(cartonSizes[i] + " * " +
         cartonCount + " = " + (cartonSizes[i] * cartonCount));
      }
    }

    // 1 carton of capacity 6 needed if boxes left < 6
    if (t != 0) {
      System.out.println("Remaining boxes = " + t +
                         " * 1 = " + t);
      total++;
    } else {
      System.out.println("Remaining boxes = 0");
    }

    System.out.println("Total number of boxes = " + n);
    System.out.println("Total number of cartons = " + total);
  }
}
```

Output:

```
Enter number of boxes (N): 100
48 * 2 = 96
Remaining boxes = 4 * 1 = 4
Total number of boxes = 100
Total number of cartons = 3
```

```
Enter number of boxes (N): 90
48 * 1 = 48
24 * 1 = 24
12 * 1 = 12
6 * 1 = 6
Remaining boxes = 0
Total number of boxes = 90
Total number of cartons = 4
```

```
Enter number of boxes (N): 1200
INVALID INPUT
```

Variable Description Table:

Name	Datatype	Description
`n`	`int`	Stores the number of boxes to be packed (input).
`cartonSizes`	`int[]`	Array containing the sizes of cartons [48, 24, 12, 6].
`total`	`int`	Stores the total number of cartons used.
`t`	`int`	Tracks the remaining boxes to be packed.
`cartonCount`	`int`	Temporary variable to store how many cartons of each size are needed.

Number 4:

A Special number is a number in which the sum of the factorial of its digits is equal to the number.

Example: 145 (1! + 4! + 5! = 145). Thus, 145 is a special number.

Design a class Special to check if the given number is a Special number or not. Some of the members of the class are given below:

Class name: Special

Data members /instance variables:

`n` : integer to store the number

Member functions:

`Special()`	: default constructor.
`void read()`	: to accept the number.
`int factorial(int x)`	: return the factorial of a number using recursion technique.
`boolean isSpecial()`	: checks for the special number by invoking the function factorial( ) and returns true if Special, otherwise returns false.
`void display()`	: to show the result with an appropriate message.

Algorithm:

1. **Initialize the Class**:
   - Create a default constructor to initialize variables.
2. **Accept Input**:
   - The `read()` method takes input from the user and stores it in n.
3. **Factorial Calculation**:
   - The `factorial(int x)` method calculates the factorial of a number using recursion.
4. **Check Special Number**:
   - Extract each digit of the number.
   - For each digit, calculate its factorial.
   - Sum the factorials of all the digits.
   - Compare the sum with the original number.
   - If the sum matches the number, it's a Special number.
5. **Display Result**:
   - Based on the outcome of the `isSpecial()` method, print an appropriate message.

Solution:

```
import java.util.Scanner;

public class Special {

  // Data member to store the number
  int n;

  // Default constructor to initialize n
  Special() {
    n = 0;
  }


  // Method to accept the number from the user
  void read() {
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter the number: ");
    n = sc.nextInt();

    sc.close();
  }


  // Recursive method to calculate factorial of a number
  int factorial(int x) {
    if (x == 0 || x == 1) {
      return 1;
    } else {
      return x * factorial(x - 1);
    }
  }
```

```
  // Method to check if the number is a Special number
  boolean isSpecial() {

    // Copy of the number to extract digits
    int temp = n;
    int sum = 0;

    // Loop to calculate sum of factorials of digits
    while (temp > 0) {
      int digit = temp % 10;
      // Add the factorial of the digit to sum
      sum += factorial(digit);
      temp /= 10;
    }

    // If sum equals the original number, it's Special
    return sum == n;
  }

  // Method to display the result
  void display() {

    System.out.println(n + " is " +
      (isSpecial() ? "" : "not ") + "a Special number.");
  }

  // Main method to execute the program
  public static void main(String[] args) {

    // Create an object of the Special class
    Special obj = new Special();

    obj.read();

    obj.display();
  }
}
```

Output:

```
Enter the number: 145
145 is a Special number.
```

```
Enter the number: 123
123 is not a Special number.
```

```
Enter the number: 1
1 is a Special number.
```

```
Enter the number: 0
0 is a Special number.
```

Variable Description Table:

Name	Datatype	Description
`n`	`int`	Stores the number entered by the user to check if it's special.
`digit`	`int`	Stores individual digits of the number n.
`sum`	`int`	Accumulates the sum of the factorial of the digits of n.
`temp`	`int`	Temporary copy of n used to extract digits for processing.
`x`	`int`	Parameter of the factorial() method representing the number whose factorial is being calculated.
`sc`	`Scanner`	Used to take input from the user.

Number 5:

A Composite Magic number is a positive integer which is composite as well as a magic number.

Composite number: A composite number is a number which has more than two factors.

For example: Factors of 10 are: 1, 2, 5, 10

Magic number: A Magic number is a number in which the eventual sum of the digit is equal to 1.

For example: $28 = 2 + 8 = 10 = 1 + 0 = 1$

Accept two positive integers 'm' and 'n', where m is less than n. Display the number of composite magic integers that are in the range between m and n (both inclusive) and output them along with frequency, in the format specified below:

Sample Input:

m=10, n=100

Output: The composite magic numbers are 10, 28, 46, 55, 64, 82, 91, 100

Frequency of composite magic numbers: 8

Sample Input:

m=120, n=90

Output: Invalid input

Algorithm:

1. **Composite check**
   - Input: A positive integer n.
   - Check for factors:
     - Initialize a counter to count the number of factors.
     - Iterate overall numbers i from 1 to n.
     - If `n % i == 0`, increment the factor counter.
     - If the number of factors is greater than 2, then it is composite; otherwise, it is not.

2. M**agic Number check**
   - Calculate the sum of digits of the number:
   - Convert the number to a string or extract digits using modulus (`%`) and division (`/`).
   - Sum the digits.
   - Reduce the sum recursively
   - If the sum of the digits is greater than 9, repeat the process until the sum is a single digit.
   - Check if the sum is equal to 1:
   - If the final single digit is 1, then the number is a magic number.

3. **Combine both checks**
   - If the number is composite and a magic number, then print that it is a composite magic number.

Solution:

```
import java.util.Scanner;

public class MagicComposite {

  // Function to check if a number is composite
  public static boolean isComposite(int num) {

    int num_sqrt = (int) Math.sqrt(num);
    for (int i = 2; i <= num_sqrt; i++) {
      if (num % i == 0) {
        return true;
      }
    }
    return false;
  }

  // Function to check if a number is magic
  public static boolean isMagic(int num) {

    // Reduce to the sum of digits
    while (num > 9) {
      num = sumOfDigits(num);
    }

    return num == 1;
  }

  // Helper function to calculate
  // the sum of digits of a number
  public static int sumOfDigits(int num) {
    int sum = 0;

    while (num > 0) {
      sum += num % 10;
      num /= 10;
    }
    return sum;
  }
```

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

    Scanner sc = new Scanner(System.in);

    // Input the range (m and n)
    System.out.print("Enter the lower bound (m): ");
    int m = sc.nextInt();
    System.out.print("Enter the upper bound (n): ");
    int n = sc.nextInt();

    sc.close();

    System.out.println("Magic Composite numbers between " +
                        m + " and " + n + " are:");

    // Iterate through all numbers in the range [m, n]
    for (int i = m; i <= n; i++) {
      if (isComposite(i) && isMagic(i)) {

        // Print if the number is both composite and magic
        System.out.println(i);
      }
    }
  }
}
```

Output:

```
Enter the lower bound (m): 1
Enter the upper bound (n): 30
Magic Composite numbers between 1 and 30 are:
10
28
```

```
Enter the lower bound (m): 50
Enter the upper bound (n): 120
Magic Composite numbers between 50 and 120 are:
55
64
82
91
100
118
```

Variable Description Table:

Name	Datatype	Description
`m`	`int`	The lower bound of the input range provided by the user.
`n`	`int`	The upper bound of the input range provided by the user.
`i`	`int`	Loop variable that iterates through all numbers between m and n to check for composite magic numbers.
`counter`	`int`	Counter used to count the number of divisors of a number to determine if it is composite.
`sum`	`int`	Stores the sum of digits of a number during the magic number check process.
`scanner`	`Scanner`	Object of Scanner class used to take input for m and n from the user.

Number 6:

A Circular Prime is a prime number that remains prime under cyclic shifts of its digits. When the leftmost digit is removed and replaced at the end of the remaining string of digits, the generated number is still prime. The process is repeated until the original number is reached again.

A number is said to be prime if it has only two factors 1 and itself.

Accept a positive number N and check whether it is a circular prime or not. The new numbers formed after the shifting of the digits should also be displayed.

Test your program with the following data and some random data:

Example 1

INPUT:
N = 197
OUTPUT:
197
971
719
197 IS A CIRCULAR PRIME

Example 2

INPUT:
N = 29
OUTPUT:
29
92
29 IS NOT A CIRCULAR PRIME.

Algorithm:

1. Input the Number:

- Accept a positive integer n from the user.
- If n is less than or equal to zero, display "INVALID INPUT" and terminate.

2. Check if the Number is Prime:

- Call the `isPrime(num)` function to check whether the number n is a prime number.
- If n is not prime, print that n is not a circular prime and terminate.

3. Calculate the Number of Digits:

- Use the `getDigitCount(num)` function to count the number of digits in n and store it in `digitCount`.

4. Display the Original Number:

- Print the original number n.

5. Shift the Digits and Check Primality:

- Set n2 to the original number n.
- For each shift:
  - Extract the leftmost digit of n2 (using t1 = n2 / divisor).
  - Remove the leftmost digit and shift the remaining digits (using t2 = n2 % divisor).
  - Form the new shifted number by appending t1 to the end of t2 (n2 = t2 * 10 + t1).

6. Check Circular Prime:

- If all shifted numbers are prime, print that n is a circular prime.
- Otherwise, print that n is not a circular prime.

Solution:

```
import java.util.Scanner;

public class CircularPrime {
  // Method to check if number is prime
  public static boolean isPrime(int num) {
    if (num <= 1) return false;
    int numSqrt = (int) Math.sqrt(num);

    for (int i = 2; i <= numSqrt; i++) {
      if (num % i == 0) {
        return false;
      }
    }
    return true;
  }

  // Method to check the number of digits
  public static int getDigitCount(int num) {
    int digitCount = 0;
    while (num != 0) {
      digitCount++;
      num /= 10;
    }
    return digitCount;
  }

  // Main method of the class circular prime
  public static void main(String args[]) {
    Scanner in = new Scanner(System.in);
    System.out.print("N = ");
    int n = in.nextInt();
    in.close();
```

```
    if (n <= 0) {
      System.out.println("INVALID INPUT.");
      return;
    }

    // Check if the number is a circular prime
    boolean isCircularPrime = true;
    int digitCount = getDigitCount(n);
    int divisor = (int) (Math.pow(10, digitCount - 1));
    int n2 = n;

    for (int i = 0; i < digitCount; i++) {
      System.out.println(n2);

      if (!isPrime(n2)) {
        isCircularPrime = false;
        break;
      }

      // Rotate the number to its right
      int t1 = n2 / divisor;
      int t2 = n2 % divisor;
      n2 = t2 * 10 + t1;
    }

    System.out.println(n + " IS " +
      (isCircularPrime ? "" : "NOT ") + "A CIRCULAR PRIME");
  }
}
```

Output:

```
N = 197
197
971
719
197 IS A CIRCULAR PRIME
```

```
N = 1973
1973
9731
1973 IS NOT A CIRCULAR PRIME
```

```
N = -197
INVALID INPUT.
```

```
N = 0
INVALID INPUT.
```

Variable Description Table:

Name	Datatype	Description
N	int	Stores the input number to be checked for circular prime.
numStr	String	Stores the string representation of the number N.
shiftedNum	int	Stores the integer value of the cyclically shifted number.
isCircularPrime	boolean	Flag to check whether the number is a circular prime.
len	int	Stores the length of the number N in digits.
i	int	Loop control variable used for rotating and checking each shifted number.
tempStr	String	Temporary string used to create cyclic shifts of the number.
isPrime	boolean	Flag used to check whether a number is prime.

Number 7:

A class Palin has been defined to check whether a positive number is a palindrome. Additionally, the class checks if the sum of the digits of the palindrome number is an even number. The number N is a palindrome if the original number and its reverse are the same.

Some of the members of the class are given below:

Class name: Palin

Data members (instance variables):

`num` : An integer to store the original number.

`Revnum` : An integer to store the reverse of the number.

`sumDigits` : An integer to store the sum of the digits of the original number.

Methods/Member functions:

1. `Palin()` : Constructor to initialize num, revnum, and sumDigits with legal initial values.
2. `void accept()` : To accept the number using user input.
3. `int reverse(int y)` : Recursively reverses the parameterized argument y and stores it in revnum.
4. `int sumOfDigits(int x)`: Recursively calculates the sum of digits of the number x and stores it in sumDigits.
5. `void check()` : Checks whether the number is a palindrome by invoking reverse() and then checks if the sum of its digits is even by invoking sumOfDigits(). Displays the appropriate messages.

Algorithm:

1. **Class Initialization (Palin Constructor):**
   - Initialize `num`, `revnum`, and `sumDigits` to 0.

2. **Accept Input (`accept()` Method):**
   - Accept a positive number from the user using a Scanner.
   - Store this number in num.

3. **Reverse the Number (`reverse(int y)` Method):**
   - If y is 0, return 0.
   - Otherwise, use recursion to reverse the digits of y.
   - Calculate the reversed number by combining the last digit of y and recursively processing the rest of the number.
   - Update `revnum` with the reversed number.

4. **Sum of Digits (`sumOfDigits(int x)` Method):**
   - If x is 0, return 0.
   - Otherwise, recursively sum the digits of x.
   - Add the last digit of x to the sum obtained by recursively processing the rest of the number.
   - Store the total sum in `sumDigits`.

5. **Check Palindrome and Sum of Digits (`check()` Method):**
   - Invoke `reverse()` using num as the argument and store the result in `revnum`.
   - Compare num and `revnum` to check if they are the same.
   - If they are equal, invoke `sumOfDigits()` and store the result in `sumDigits`.
   - Check if `sumDigits` is even.
   - Display messages indicating whether the number is a palindrome and whether the sum of its digits is even.

Solution:

```
import java.util.Scanner;

public class Palin {

  // Data Members
  int num;
  int revnum;
  int sumDigits;

  // Constructor to initialize the instance variables
  public Palin() {
    num = 0;
    revnum = 0;
    sumDigits = 0;
  }

  // Method to accept a number from the user
  public void accept() {
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter a positive number: ");
    num = sc.nextInt();

    sc.close();
  }

  // Recursive method to reverse the number
  public int reverse(int y) {
    if (y == 0) {
      return 0;
    } else {
      int lastDigit = y % 10;
      revnum = revnum * 10 + lastDigit;
      reverse(y / 10);
      return revnum;
    }
  }
```

```
  // Recursive method to sum the digits of the number
  public int sumOfDigits(int x) {
    if (x == 0) {
      return 0;
    } else {
      int lastDigit = x % 10;
      sumDigits += lastDigit;
      sumOfDigits(x / 10);
      return sumDigits;
    }
  }

  // Method to check if the number is a palindrome and sum of
digits is even
  public void check() {
    revnum = 0;
    sumDigits = 0;

    int reversed = reverse(num);
    int sum = sumOfDigits(num);

    if (num == reversed) {
      System.out.println(num + " is a palindrome.");
      System.out.println("The sum of its digits (" + sum +
        ") is " + (sum % 2 == 0 ? "even" : "odd") + ".");
    } else {
      System.out.println(num + " is not a palindrome.");
    }
  }

  public static void main(String[] args) {
    Palin palinObj = new Palin();

    palinObj.accept();

    palinObj.check();
  }
}
```

Output:

```
Enter a positive number: 1221
1221 is a palindrome.
The sum of its digits (6) is even.
```

```
Enter a positive number: 1234
1234 is not a palindrome.
```

```
Enter a positive number: 7
7 is a palindrome.
The sum of its digits (7) is odd.
```

```
Enter a positive number: 010
10 is not a palindrome.
```

Variable Description Table:

Name	Datatype	Description
`num`	`int`	Stores the original input number.
`revnum`	`int`	Stores the reverse of the number.
`sumDigits`	`int`	Stores the sum of the digits of the number.
`y`	`int`	Parameter in reverse() method representing the number to reverse.
`x`	`int`	Parameter in sumOfDigits() method representing the number for digit summation.
`lastDigit`	`int`	Stores the last digit of a number during reverse and sum operations.

Number 8:

Design a class Perfect to check if a given number is a perfect number or not. [A number is said to be perfect if sum of the factors of the number excluding itself is equal to the original number]

Example : $6 = 1 + 2 + 3$
(where 1, 2 and 3 are factors of 6, excluding itself)

Some of the members of the class are given below:

Class name : Perfect

Data members/instance variables:

`num` : to store the number

Methods/Member functions:

`Perfect (int nn)` : parameterized constructor to initialize the datamember `num=nn.`

`int sum_of_factors(int i)` : returns the sum of the factors of the `number(num)`, excluding itself, using recursive technique.

`void check( )` : checks whether the given number is perfect by invoking the function `sum_of_factors()` and displays the result with an appropriate message.

Algorithm:

1. **Define Class:** Create a class named Perfect.
2. **Declare Data Member:** Create an instance variable num of type int to store the number.
3. **Constructor:** Create a parameterized constructor that takes an integer `nn` and initializes num with `nn`.
4. **Sum of Factors Method:**
   - **Recursive Method:** Create a method named/called as `sum_of_factors(int i)` that recursively calculates the sum of all factors of num (excluding itself).
   - **Base Case:** If i is 1, return 0 (no more factors to check).
   - **Recursive Step:** If num is divisible by i, add i to the sum. Call `sum_of_factors(i - 1)` to check the next smaller number.
5. **Check Method:** Create a method check() that:
   - Calls `sum_of_factors(num - 1)` to get the sum of factors of num excluding itself.
   - Checks if the sum equals num. If yes, print that num is a perfect number. Otherwise, print that it is not a perfect number.

Solution:

```
import java.util.Scanner;

public class Perfect {

  int num;

  Perfect(int nn) { num = nn; }

  // Recursive method for sum of factors, excluding itself
  int sum_of_factors(int i) {
    if (i == 0) {
      return 0; // Base case: no more factors to check
    }
    // Add factor to sum if it is a perfect number
    return (num % i == 0 ? i : 0) + sum_of_factors(i - 1);
  }

  // Method to check if the number is perfect
  void check() {
    int sum = sum_of_factors(num - 1); // not number itself
    System.out.println(num + " is " +
      (sum == num ? "" : "not ") + "a perfect number.");
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a number: ");
    int number = sc.nextInt();
    sc.close();

    Perfect obj = new Perfect(number);

    obj.check();
  }
}
```

Output:

```
Enter a number to check if it is perfect: 28
28 is a perfect number.
```

```
Enter a number to check if it is perfect: 8128
8128 is a perfect number.
```

```
Enter a number to check if it is perfect: 256
256 is not a perfect number.
```

```
Enter a number to check if it is perfect: 1
1 is not a perfect number.
```

Variable Description Table:

Name	Datatype	Description
`num`	`int`	Stores the number to check for perfection.
`nn`	`int`	Parameter used to initialize num in the constructor.
`i`	`int`	The current divisor in the sum_of_factors() method.
`sum`	`int`	Holds the sum of factors of num, excluding itself, in the check() method.
`number`	`int`	Holds the user input number to be checked for perfection in the main method.
`sc`	`Scanner`	Object to take user input.
`obj`	`Perfect`	Object of class Perfect to call the check() method for verification.

Number 9:

Design a class ArmNum to check if a given number is an Armstrong number or not.

[A number is said to be Armstrong if sum of its digits raised to the power of length of the number is equal to the number]

Example: $371 = 3^3 + 7^3 + 1^3$

$1634 = 1^4 + 6^4 + 3^4 + 4^4$

Thus 371, 1634 are all examples of Armstrong numbers.

Some of the members of the class are given below:

Class name: ArmNum

Data members/instance variables:

`n` : to store the number
`l` : to store the length of the number

Methods/Member functions:

`ArmNum (int nn)` : parameterized constructor to initialize the data member n=nn.

`int sum_pow(int i)` : returns the sum of each digit raised to the power of the length of the number using recursive technique.

`void isArmstrong( )` : checks whether the given number is an Armstrong number by invoking the function `sum_pow( )` and displays the result with an appropriate message.

Algorithm:

1. **Define Class:** Create a class named ArmNum.
2. **Declare Data Members:**
   - n (int) to store the given number.
   - l (int) to store the number of digits (length) of n.
3. **Constructor:** Create a parameterized constructor that:
   - Takes an integer nn.
   - Initializes n with nn.
   - Converts n to a string to find its length (l).
4. **Sum of Powers Method:**
   - Create a method sum_pow(int i) that:
     - Recursively calculates the sum of each digit of n raised to the power of l.
     - **Base Case:** If i is 0, return 0.
     - **Recursive Step:** Get the last digit of i, compute its power l, add to the sum, and call sum_pow(i/10) for the remaining digits.
5. **Check Armstrong Method:**
   - Create a method isArmstrong() that:
     - Calls sum_pow(n) to compute the sum of digits raised to the power of l.
     - Compares this sum to n.
     - Displays an appropriate message indicating whether n is an Armstrong number.
6. **Main Method:**
   - Accepts user input for the number.
   - Creates an object of ArmNum to check if the number is an Armstrong number.

Solution:

```
import java.util.Scanner;

public class ArmNum {

  // Instance variables
  int n;
  int l;


  ArmNum(int nn) {
    n = nn;

    // Convert number to string to find its length
    l = String.valueOf(n).length();
  }


  // Recursive method to calculate the sum of digits
  // raised to the power of the length
  int sum_pow(int i) {

    if (i == 0) {
      return 0; // Base case: no more digits to process
    }

    int digit = i % 10;

    // Add digit^l to sum and recurse for remaining digits
    return (int) Math.pow(digit, l) + sum_pow(i / 10);
  }
```

```
  // Method to check if the number is an Armstrong number
  void isArmstrong() {

    // get the sum of digits raised to the power of length
    int sum = sum_pow(n);

    System.out.println(n + " is " +
      (sum == n ? "" : "not ") + "an Armstrong number.");
  }


  public static void main(String[] args) {

    Scanner sc = new Scanner(System.in);

    System.out.print("Enter a number to check if it is an
Armstrong number: ");
    int number = sc.nextInt();

    sc.close();

    // Create an object of ArmNum class
    ArmNum obj = new ArmNum(number);

    // Check if the number is an Armstrong number
    obj.isArmstrong();

  }
}
```

Output:

```
Enter a number to check if it is an
Armstrong number: 153
153 is an Armstrong number.
```

```
Enter a number to check if it is an
Armstrong number: 123
123 is not an Armstrong number.
```

```
Enter a number to check if it is an
Armstrong number: 5
5 is an Armstrong number.
```

```
Enter a number to check if it is an
Armstrong number: 9474
9474 is an Armstrong number.
```

Variable Description Table:

Name	Datatype	Description
`n`	`int`	Stores the number to check for Armstrong status.
`l`	`int`	Stores the length (number of digits) of n.
`i`	`int`	Used as a parameter in the sum_pow() method to represent the current number being processed.
`digit`	`int`	Stores the last digit of i in the sum_pow() method.
`sum`	`int`	Holds the sum of digits raised to the power of l in the isArmstrong() method.
`nn`	`int`	Parameter to initialize n in the constructor.
`sc`	`Scanner`	Object to take user input in the main method.
`obj`	`ArmNum`	Object of the class ArmNum to check for Armstrong number.

Number 10:

Design a class NumDude to check if a given number is a Dudeney number or not.

(A Dudeney number is a positive integer that is a perfect cube, such that the sum of its digits is equal to the cube root of the number.)

Example: $5832 = (5 + 8 + 3 + 2)3 = (18)3 = 5832$

Some of the members of the class are given below:

Class name: NumDude

Data member/instance/variable:

`num` : to store a positive integer number

Methods/Member functions:

`NumDude()` : default constructor to initialise the data member with legal initial value.

`void input()` : to accept a positive integer number.

`int sumDigits(int x)` : returns the sum of the digits of number 'x' using recursive technique.

`void is Dude()` : checks whether the given number is a Dudeney number by invoking the function sumDigits() and displays the result with an appropriate message.

Algorithm:

1. **Create the Class**:
   - Set up a class named NumDude to hold the number.
2. **Initialize the Number**:
   - Use a constructor to set the number to 0.
3. **Get Input**:
   - Create a method to ask the user for a positive number. If invalid, prompt again.
4. **Calculate Sum of Digits**:
   - Create a method to add all digits of the number using a recursive approach i.e. add last digit to sum of digits of the number without the last digit and so on.
5. **Check for Dudeney Number**:
   - Find the sum of digits.
   - Cube this sum and compare it to the original number.
   - Display whether the number is a Dudeney number.
6. **Run the Program**:
   - Create an instance of NumDude.
   - Call methods to get input and check the number.

Solution:

```
import java.util.Scanner;

public class NumDude {
  // Instance variable
  private int num;

  public NumDude() { num = 0; }

  public void input() {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a positive integer: ");
    num = sc.nextInt();
    if (num <= 0) {
      System.out.println("Invalid input! Enter positive integer.");
      input(); // Recur to get a valid input
    } else sc.close();
  }

  // Recursive method to calculate the sum of digits
  public int sumDigits(int x) {
    if (x == 0) return 0;
    return (x % 10) + sumDigits(x / 10);
  }

  public void isDude() {
    int sum = sumDigits(num);
    int sumCubed = sum * sum * sum;

    System.out.println(num + " is " +
      (sumCubed == num ? "" : "NOT ") + "a Dudeney number.");
  }

  public static void main(String[] args) {
    NumDude numDude = new NumDude();
    numDude.input();
    numDude.isDude();
  }
}
```

Output:

```
Enter a positive integer: 512
512 is a Dudeney number.
```

```
Enter a positive integer: 123
123 is a Dudeney number.
```

```
Enter a positive integer: -10
Invalid input! Please enter a positive integer.
Enter a positive integer: 30
30 is NOT a Dudeney number.
```

Variable Description Table:

Name	Datatype	Description
`num`	`int`	Stores the positive integer input by the user.
`x`	`int`	Used in the recursive sumDigits method to hold the current number during calculations.
`sum`	`int`	Stores the sum of the digits of num. Calculated using the sumDigits method.
`sumCubed`	`int`	Stores the cube of the sum of digits. This is used to check if the number is a Dudeney number.
`sc`	`Scanner`	Used to take input from the user in the input method.

Array & Matrix Programs

Matrix 1:

Write a program to declare a matrix A[ ][ ] of order (M x N) where 'M' is the number of rows and 'N' is the number of columns such that the value of 'M' must be greater than 0 and less than 10 and the value of 'N' must be greater than 2 and less than 6. Allow the user to input digits (0 - 7) only at each location, such that each row represents an octal number.

Example:

2	3	1	(decimal equivalent of 1st row = 153 i.e. $2x8^2 + 3x8^1 + 1x8^0$)
4	0	5	(decimal equivalent of 2nd row = 261 i.e. $4x8^2 + 0x8^1 + 5x8^0$)
1	5	6	(decimal equivalent of 3rd row = 110 i.e. $1x8^2 + 5x8^1 + 6x8^0$)

Perform the following tasks on the matrix:

1. Display the original matrix.
2. Calculate the decimal equivalent for each row and display as per the format given below.

Test your program for the following data and some random data:

Example 1:
INPUT:
M = 1
N = 3
ENTER ELEMENTS FOR ROW 1: 1 4 4

OUTPUT:

FILLED MATRIX	**DECIMAL EQUIVALENT**
1 4 4	100

Example 2:
INPUT:
M = 3
N = 4
ENTER ELEMENTS FOR ROW 1: 1 1 3 7
ENTER ELEMENTS FOR ROW 2: 2 1 0 6
ENTER ELEMENTS FOR ROW 3: 0 2 4 5

OUTPUT:

FILLED MATRIX	DECIMAL EQUIVALENT
1 1 3 7	607
2 1 0 6	1094
0 2 4 5	165

Example 3:
INPUT:
M = 3
N = 3
ENTER ELEMENTS FOR ROW 1: 2 4 8

OUTPUT:
INVALID INPUT

Example 4:
INPUT:
M = 4
N = 6

OUTPUT:
OUT OF RANGE

Algorithm:

1. **Input Validation**:
   - Accept the inputs M and N. Ensure that M is between 1 and 9, and N is between 3 and 5. If the input is out of range, display "OUT OF RANGE" and terminate the program.

2. **Matrix Declaration and Input**:
   - Declare a matrix A[M][N] of size M by N.
   - For each element in the matrix, prompt the user to input a digit between 0 and 7. If any element is outside this range, display "INVALID INPUT" and terminate the program.

3. **Display the Original Matrix**:
   - Display the matrix in a tabular form as provided by the user.

4. **Convert Each Row to Decimal**:
   - For each row in the matrix, convert it from an octal number to its decimal equivalent. This can be done using the formula:

$$\text{decimal_value} = A[i][0] \times 8^{(N-1)} + A[i][1] \times 8^{(N-2)} + \cdots A[i][N-1] \times 8^{0}$$

   - Calculate the decimal equivalent for each row.

5. **Display the Decimal Equivalents**:
   - Print the original matrix along with its decimal equivalent for each row.

Solution:

```
import java.util.Scanner;


public class OctalMatrix {


  public static void main(String[] args) {

    Scanner sc = new Scanner(System.in);

    // Input M and N values
    System.out.print("Enter number of rows M: ");
    int M = sc.nextInt();
    System.out.print("Enter number of columns N: ");
    int N = sc.nextInt();

    // Check for valid range of M and N
    if (M <= 0 || M >= 10 || N <= 2 || N >= 6) {
      System.out.println("OUT OF RANGE");
      sc.close();
      return;
    }


    // Declare the matrix
    int[][] A = new int[M][N];


    // Input elements for the matrix and validate each element
    for (int i = 0; i < M; i++) {

      System.out.println("Enter elements for row"
                         + (i + 1) + ": ");
```

```
      for (int j = 0; j < N; j++) {
        A[i][j] = sc.nextInt();

        // Check if input is a valid octal digit (0-7)
        if (A[i][j] < 0 || A[i][j] > 7) {
          System.out.println("INVALID INPUT");
          sc.close();
          return;
        }
      }
    }

    sc.close();


    // Display the filled matrix and the decimal equivalent
    System.out.println("FILLED MATRIX\tDECIMAL EQUIVALENT");

    for (int i = 0; i < M; i++) {
      int decimal_value = 0;

      // Display the row and calculate its decimal equivalent
      for (int j = 0; j < N; j++) {
        System.out.print(A[i][j] + "\t");
        decimal_value += A[i][j] * Math.pow(8, N - j - 1);
      }

      // Print the decimal equivalent for the row
      System.out.println(decimal_value);
    }
  }
}
```

Output:

```
Enter number of rows M: 3
Enter number of columns N: 4
Enter elements for row 1:
1 2 3 4
Enter elements for row 2:
5 6 7 0
Enter elements for row 3:
3 1 2 7
FILLED MATRIX   DECIMAL EQUIVALENT
1   2   3   4   668
5   6   7   0   3000
3   1   2   7   1623
```

```
Enter number of rows M: 10
Enter number of columns N: 6
OUT OF RANGE
```

```
Enter number of rows M: 2
Enter number of columns N: 3
Enter elements for row 1:
1 2 8
INVALID INPUT
```

Variable Description Table:

Name	Datatype	Description
`M`	`int`	Number of rows, input by the user
`N`	`int`	Number of columns, input by the user
`A`	`int[][]`	Matrix to store the digits
`i`	`int`	Loop variable for iterating through rows
`j`	`int`	Loop variable for iterating through columns
`decimal_value`	`int`	Stores the decimal equivalent of a row
`octal_digit`	`int`	Octal digit input by the user at each matrix location

Matrix 2:

Write a program to declare a single-dimensional array a[ ] and a square matrix b[ ][ ] of size N, where N > 2 and N < 10. Allow the user to input positive integers into the single dimensional array.

Perform the following tasks on the matrix:

1. Sort the elements of the single-dimensional array in ascending order using any standard sorting technique and display the sorted elements.
2. Fill the square matrix b[ ][ ] in the following format:
   If the array a[ ] = {5, 2, 8, 1}
   then, after sorting a[ ] = {1, 2, 5, 8}
   Then, the matrix b[ ][ ] would fill as below:

1	2	5	8
1	2	5	1
1	2	1	2
1	1	2	5

3. Display the filled matrix in the above format.

Test your program for the following data and some random data:

Example 1

INPUT:
N = 3
ENTER ELEMENTS OF SINGLE DIMENSIONAL ARRAY: 3 1 7

OUTPUT:
SORTED ARRAY: 1 3 7
FILLED MATRIX

1	**3**	**7**
1	**3**	**1**
1	**1**	**3**

Example 2

INPUT:
N = 13

OUTPUT:
MATRIX SIZE OUT OF RANGE

Example 3

INPUT:
N = 5
ENTER ELEMENTS OF SINGLE DIMENSIONAL ARRAY: 10 2 5 23 6

OUTPUT:
SORTED ARRAY: 2 5 6 10 23
FILLED MATRIX

2	5	6	10	23
2	5	6	10	2
2	5	6	2	5
2	5	2	5	6
2	2	5	6	10

Algorithm:

1. **Input and Validation:**
   - Accept the size N for the matrix and array. Ensure N is between 3 and 9 (both inclusive). If out of range, display an error message and exit.
2. **Array Input:**
   - Accept N positive integers into the single-dimensional array `a[]`.
3. **Sorting:**
   - Sort the array `a[]` in ascending order using the bubble sort algorithm.
4. **Matrix Filling:**
   - Fill the matrix `b[][]` such that:
     - First row `b[0]` holds the sorted array `a[]`.
     - Each subsequent row has the same elements as the previous row but shifted cyclically leftward.
5. **Display:**
   - Display the sorted array.
   - Display the matrix.

Solution:

```
import java.util.Scanner;

public class Array {

  // Sorting array using bubble sort
  public static void sortArray(int arr[]) {
    int n = arr.length;
    for (int i = 0; i < n - 1; i++) {
      for (int j = 0; j < n - i - 1; j++) {
        if (arr[j] > arr[j + 1]) {
          int t = arr[j];
          arr[j] = arr[j + 1];
          arr[j + 1] = t;
        }
      }
    }
  }

  // Function to fill matrix b[][] using cyclic shifts
  public static void fillMatrix(int a[], int b[][]) {
    int n = a.length;
    // First row is the sorted array itself
    for (int j = 0; j < n; j++) {
      b[0][j] = a[j];
    }

    // For each row cyclically shift prev row left by 1
position
    for (int i = 1; i < n; i++) {
      for (int j = 0; j < n; j++) {
        // Shift elements cyclically
        b[i][j] = b[i - 1][(j + 1) % n];
      }
    }
  }
```

```
  public static void main(String args[]) {
    Scanner in = new Scanner(System.in);
    System.out.print("ENTER VALUE OF N: ");
    int n = in.nextInt();

    if (n <= 2 || n >= 10) {
      System.out.println("MATRIX SIZE OUT OF RANGE");
      in.close();
      return;
    }

    int a[] = new int[n], b[][] = new int[n][n];

    System.out.println("ENTER ELEMENTS OF" +
                       "SINGLE DIMENSIONAL ARRAY:");
    for (int i = 0; i < n; i++) {
      a[i] = in.nextInt();
    }

    in.close();

    sortArray(a);
    System.out.println("SORTED ARRAY:");
    for (int i = 0; i < n; i++) {
      System.out.print(a[i] + " ");
    }

    fillMatrix(a, b);
    System.out.println("\nFILLED MATRIX:");
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        System.out.print(b[i][j] + " ");
      }
      System.out.println();
    }
  }
}
```

Output:

```
ENTER VALUE OF N: 3
ENTER ELEMENTS OF SINGLE DIMENSIONAL
ARRAY:
5 1 4
SORTED ARRAY:
1 4 5
FILLED MATRIX:
1 4 5
4 5 1
5 1 4
```

```
ENTER VALUE OF N: 2
MATRIX SIZE OUT OF RANGE
```

```
ENTER VALUE OF N: 10
MATRIX SIZE OUT OF RANGE
```

```
ENTER VALUE OF N: 9
ENTER ELEMENTS OF SINGLE DIMENSIONAL
ARRAY:
1 2 3 4 5 6 7 8 9
SORTED ARRAY:
1 2 3 4 5 6 7 8 9
FILLED MATRIX:
1 2 3 4 5 6 7 8 9
2 3 4 5 6 7 8 9 1
3 4 5 6 7 8 9 1 2
4 5 6 7 8 9 1 2 3
5 6 7 8 9 1 2 3 4
6 7 8 9 1 2 3 4 5
7 8 9 1 2 3 4 5 6
8 9 1 2 3 4 5 6 7
9 1 2 3 4 5 6 7 8
```

Variable Description Table:

Name	Datatype	Description
a[]	int[]	Single-dimensional array to hold N positive integers.
b[][]	int[][]	2D square matrix of size N x N filled with sorted elements.
n	int	Size of the matrix (between 3 and 9).
i, j	int	Loop variables used for bubble sorting.
r, c	int	Loop variables for filling and displaying the matrix.
temp	int	Temporary variable used for shifting rows cyclically.

Matrix 3:

Write a program to declare a matrix a[ ][ ] of order (m × n) where 'm' is the number of rows and 'n' is the number of columns such that the values of both 'm' and 'n' must be greater than 2 and less than 10. Allow the user to input integers into this matrix.

Perform the following tasks on the matrix:

1. Display the original matrix.
2. Sort each row of the matrix in ascending order using any standard sorting technique.
3. Display the changed matrix after sorting each row.

Test your program for the following data and some random data:

Example 1

INPUT:
M = 4
N = 3
ENTER ELEMENTS OF MATRIX:

11	-2	3
5	16	7
9	0	4
3	1	8

OUTPUT:
ORIGINAL MATRIX

11	-2	3
5	16	7
9	0	4
3	1	8

MATRIX AFTER SORTING ROWS

-2	3	11
5	7	16
0	4	9
1	3	8

Example 2

INPUT:
M = 3
N = 3
ENTER ELEMENTS OF MATRIX

22	5	19
7	36	12
9	13	6

OUTPUT:
ORIGINAL MATRIX

22	5	19
7	36	12
9	13	6

MATRIX AFTER SORTING ROWS

5	19	22
7	12	36
6	9	13

Example 3

INPUT:
M = 11
N = 5

OUTPUT:
MATRIX SIZE OUT OF RANGE.

Algorithm:

1. **Input Matrix Dimensions:**
   - Read values for m (number of rows) and n (number of columns).
   - Validate that m and n are both greater than 2 and less than 10. If not, display "MATRIX SIZE OUT OF RANGE" and terminate.
2. **Input Matrix Elements:**
   - Create a 2D array a[m][n] to store the matrix elements.
   - Use a nested loop to allow the user to input values for each element in the matrix.
3. **Display Original Matrix:**
   - Traverse through the matrix and print its original form, row by row.
4. **Sort Each Row:**
   - Implement bubble sort on each row of the matrix. For each row, compare adjacent elements and swap them if they are out of order. Repeat this process until all elements in the row are sorted.
5. **Display Sorted Matrix:**
   - After sorting, traverse the matrix again and print each row.

Solution:

```
import java.util.Scanner;

public class ArraySort {

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

    // Input number of rows (m) and columns (n)
    System.out.print("ENTER THE VALUE OF M: ");
    int m = in.nextInt();
    System.out.print("ENTER THE VALUE OF N: ");
    int n = in.nextInt();

    // Check if matrix size is valid
    if (m <= 2 || m >= 10 || n <= 2 || n >= 10) {
      System.out.println("MATRIX SIZE OUT OF RANGE.");
      in.close();
      return;
    }

    // Initialize matrix
    int[][] a = new int[m][n];

    // Input matrix elements
    System.out.println("ENTER ELEMENTS OF MATRIX:");

    for (int i = 0; i < m; i++) {
      System.out.println("ENTER ELEMENTS OF ROW " +(i+1)+ ":");

      for (int j = 0; j < n; j++) {
        a[i][j] = in.nextInt();
      }
    }

    in.close();
```

```
    // Display the original matrix
    System.out.println("ORIGINAL MATRIX");
    for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
        System.out.print(a[i][j] + " ");
      }
      System.out.println();
    }

    // Sorting each row using bubble sort
    for (int i = 0; i < m; i++) {
      for (int j = 0; j < n - 1; j++) {
        for (int k = 0; k < n - j - 1; k++) {
          if (a[i][k] > a[i][k + 1]) {
            // Swap a[i][k] and a[i][k + 1]
            int t = a[i][k];
            a[i][k] = a[i][k + 1];
            a[i][k + 1] = t;
          }
        }
      }
    }

    // Display matrix after sorting rows
    System.out.println("MATRIX AFTER SORTING ROWS");
    for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
        System.out.print(a[i][j] + " ");
      }
      System.out.println();
    }
  }
}
```

Output:

```
ENTER THE VALUE OF M: 3
ENTER THE VALUE OF N: 4
ENTER ELEMENTS OF MATRIX:
ENTER ELEMENTS OF ROW 1:
1 9 5 6
ENTER ELEMENTS OF ROW 2:
2 10 3 8
ENTER ELEMENTS OF ROW 3:
4 7 13 11
ORIGINAL MATRIX
1 9 5 6
2 10 3 8
4 7 13 11
MATRIX AFTER SORTING ROWS
1 5 6 9
2 3 8 10
4 7 11 13
```

```
ENTER THE VALUE OF M: 2
ENTER THE VALUE OF N: 3
MATRIX SIZE OUT OF RANGE.
```

```
ENTER THE VALUE OF M: 11
ENTER THE VALUE OF N: 8
MATRIX SIZE OUT OF RANGE.
```

Variable Description Table:

Name	Datatype	Description
`m`	`int`	Stores the number of rows in the matrix, input by the user.
`n`	`int`	Stores the number of columns in the matrix, input by the user.
`a`	`int[][]`	2D array to hold matrix elements.
`t`	`int`	Temporary variable used for swapping elements during sorting.
`i`	`int`	Loop counter for rows.
`j`	`int`	Loop counter for columns.
`k`	`int`	Inner loop counter for bubble sort iteration within each row.

Matrix 4:

The names of the teams participating in a competition should be displayed on a banner vertically, to accommodate as many teams as possible in a single banner. Design a program to accept the names of N teams, where $2 < N < 9$ and display them in vertical order, side by side with a horizontal tab (i.e. eight spaces).

Test your program for the following data and some random data:

Example 1

INPUT:

N = 3
Team 1: Emus
Team 2: Road Rols
Team 3: Coyote

OUTPUT:

```
E       R       C
m       o       o
u       a       y
s       d       o
        R       t
        o       e
        l
        s
```

Example 2

INPUT:

N = 4
Team 1: Royal
Team 2: Mars
Team 3: De Rose
Team 4: Kings

OUTPUT:

```
R       M       D       K
o       a       e       i
y       r       R       n
a       s       o       g
l               s       s
                e
```

Example 3

INPUT:

N = 10

OUTPUT:

INVALID INPUT

Algorithm:

1. **Input Validation:**
   - Accept the number N of teams participating in the competition.
   - Validate that N is greater than 2 and less than 9. If not, print "INVALID INPUT" and terminate the program.
2. **Input Team Names:**
   - Create a string array teams[ ] to store the names of the teams.
   - Use a loop to allow the user to input the names of all N teams.
   - Track the length of the longest team name (`highLen`).
3. **Display Teams Vertically:**
   - For each index i from 0 to `highLen - 1`:
     - For each team, print the i-th character. If the current team name length is less than i, print a space followed by a tab.
     - After printing the characters of all teams at the current index i, move to the next line.

Solution:

```
import java.util.Scanner;

public class Banner {

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

    // Input number of teams (n)
    System.out.print("ENTER THE VALUE OF N: ");
    int n = in.nextInt();
    in.nextLine(); // Clear the buffer

    // Check if the number of teams is within the valid range
    if (n <= 2 || n >= 9) {
      System.out.println("INVALID INPUT");
      in.close(); return;
    }

    // Initialize array to store team names
    String[] teams = new String[n];
    int highLen = 0;

    // Input team names
    for (int i = 0; i < n; i++) {
      System.out.print("Team " + (i + 1) + ": ");
      teams[i] = in.nextLine();

      // Track the length of the longest team name
      if (teams[i].length() > highLen) {
        highLen = teams[i].length();
      }
    }
```

```
    in.close();

    // Print teams vertically side by side with tab separation
    for (int i = 0; i < highLen; i++) {
      for (int j = 0; j < n; j++) {
        int len = teams[j].length();
        if (i >= len) {
          // If the current character index exceeds the team
          // name length, print space with tab
          System.out.print(" \t");

        } else {

          // Print the i-th character with tab separation
          System.out.print(teams[j].charAt(i) + "\t");
        }
      }

      System.out.println();
    }

  }
}
```

Output:

```
ENTER THE VALUE OF N: 4
Team 1: Eagles
Team 2: Lions
Team 3: Panthers
Team 4: Bears
E       L       P       B
a       i       a       e
g       o       n       a
l       n       t       r
e       s       h       s
s               e
                r
                s
```

```
ENTER THE VALUE OF N: 3
Team 1: The TriForce
Team 2: Terrific Trio
Team 3: Triple Sizzle
T       T       T
h       e       r
e       r       i
        r       p
T       i       l
r       f       e
i       i
F       c       S
o               i
r       T       z
c       r       z
e       i       l
        o       e
```

```
ENTER THE VALUE OF N: 2
INVALID INPUT
```

```
ENTER THE VALUE OF N: 10
INVALID INPUT
```

Variable Description Table:

Name	Datatype	Description
`n`	`int`	Stores the number of teams participating.
`teams[]`	`String[]`	Array to store the names of the teams.
`highLen`	`int`	Stores the length of the longest team name.
`i`	`int`	Loop counter to traverse through the characters vertically.
`j`	`int`	Loop counter to traverse through the team names.
`len`	`int`	Stores the length of the current team name being processed.

Matrix 5:

The result of a quiz competition is to be prepared as follows:

The quiz has five questions with four multiple choices (A, B, C, D), with each question carrying 1 mark for the correct answer. Design a program to accept the number of participants N such that N must be greater than 3 and less than 11. Create a double-dimensional array of size (Nx5) to store the answers of each participant row-wise. Calculate the marks for each participant by matching the correct answer stored in a single-dimensional array of size 5. Display the scores for each participant and also the participant(s) having the highest score.

Example: If the value of N = 4, then the array would be:

	Q1	**Q2**	**Q3**	**Q4**	**Q5**
Participant 1	A	B	B	C	A
Participant 2	D	A	D	C	B
Participant 3	A	A	B	A	C
Participant 4	D	C	C	A	B

Key to the question: D C C B A

Note: Array entries are line fed (i.e. one entry per line)

Test your program for the following data and some random data.

Example 1

INPUT:

N = 5
Participant 1: D A B C C
Participant 2: A A D C B
Participant 3: B A C D B
Participant 4: D A D C B
Participant 5: B C A D D
Key: B C D A A

OUTPUT:

Scores:
Participant 1 = 0
Participant 2 = 1
Participant 3 = 1
Participant 4 = 1
Participant 5 = 2
Highest Score:
Participant 5

Example 2

INPUT:

N = 4
Participant 1: A C C B D
Participant 2: B C A A C
Participant 3: B C B A A
Participant 4: C C D D B
Key: A C D B B

OUTPUT:

Scores:
Participant 1 = 3
Participant 2 = 1
Participant 3 = 1
Participant 4 = 3
Highest Score:
Participant 1
Participant 4

Example 3

INPUT:

N = 12

OUTPUT:

INPUT SIZE OUT OF RANGE.

Algorithm:

1. **Input N**:
   - Accept the number of participants N (must be between 4 and 10).
2. **Validation**:
   - If N is less than 4 or greater than 10, print "INPUT SIZE OUT OF RANGE" and terminate the program.
3. **Initialize Arrays**:
   - Create a 2D array answers of size N x 5 to store the participants' answers.
   - Create a 1D array key of size 5 to store the correct answers.
4. **Accept Answers**:
   - For each participant, accept their answers to the 5 questions and store them in the corresponding row of the answers array.
5. **Accept Answer Key**:
   - Read the correct answers into the key array.
6. **Score Calculation**:
   - Initialize an array score to store the marks for each participant.
   - Compare each participant's answers with the answer key. For each correct answer, increment the participant's score.
   - Track the highest score using a variable hScore.
7. **Output Scores**:
   - Print the score of each participant after calculating it.
8. **Determine and Display Highest Score**:
   - After scoring all participants, identify and print the participants who have the highest score.

Solution:

```
import java.util.Scanner;

public class QuizCompetition {

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

    System.out.print("N = ");
    int n = in.nextInt();

    // Input validation: N should be between 4 and 10
    if (n <= 3 || n >= 11) {
      System.out.println("INPUT SIZE OUT OF RANGE.");
      in.close();
      return;
    }

    char[][] answers = new char[n][5];
    char[] key = new char[5];

    // Accept answers for each participant
    for (int i = 0; i < n; i++) {
      System.out.print("Participant " + (i + 1) + ": ");
      for (int j = 0; j < 5; j++) {
        answers[i][j] = in.next().charAt(0);
      }
    }

    // Accept the correct answers (Answer Key)
    System.out.print("Key: ");
    for (int i = 0; i < 5; i++) {
      key[i] = in.next().charAt(0);
    }

    in.close();
```

```
    int[] scores = new int[n];
    int highestScore = 0;

    // Calculate the score for each participant
    System.out.println("Scores:");
    for (int i = 0; i < n; i++) {
      // Initialize the score for each participant
      scores[i] = 0;
      for (int j = 0; j < 5; j++) {
        if (answers[i][j] == key[j]) {
          scores[i]++;
        }
      }

      if (scores[i] > highestScore) {
        highestScore = scores[i];
      }

      System.out.println("Participant " + (i + 1) +
                         " = " + scores[i]);
    }

    // Find and display participants with the highest score
    System.out.println("Highest Score:");
    for (int i = 0; i < n; i++) {
      if (scores[i] == highestScore) {
        System.out.println("Participant " + (i + 1));
      }
    }
  }
}
```

Output:

```
N = 5
Participant 1: A B C D E
Participant 2: A B C D E
Participant 3: A C C D E
Participant 4: A B C D A
Participant 5: A B B D C
Key: A B C D E
Scores:
Participant 1 = 5
Participant 2 = 5
Participant 3 = 4
Participant 4 = 4
Participant 5 = 3
Highest Score:
Participant 1
Participant 2
```

Variable Description Table:

Name	Datatype	Description
`n`	`int`	Stores the number of participants (input).
`answers`	`char[][]`	2D array storing participants' answers for each question.
`key`	`char[]`	1D array storing the correct answers for the quiz.
`score`	`int[]`	Array storing the scores of each participant.
`hScore`	`int`	Stores the highest score among participants.
`i, j`	`int`	Loop variables for iterating over participants and answers.

Matrix 6:

A Circular queue is a linear data structure which works on the principle of FIFO, enables the user to enter data from the rear end and remove data from the front end with the rear end connected to the front end to form a circular pattern.

Define a class CirQueue with the following details:

Class name: CirQueue

Data members / instance variables:

cq[ ]	:	array to store the integers
cap	:	stores the maximum capacity of the array
front	:	to point the index of the front end
rear	:	to point the index of the rear end

Member functions:

`CirQueue(int max)`: constructor to initialize the data member cap=max, front=0 and rear=0.

`void push(int n)` : to add integer in the queue from the rear end if possible, otherwise display the message "QUEUE IS FULL".

`int pop()` : removes and returns the integer from the front end of the queue if any, else returns -9999.

`void show()` : displays the queue elements.

Algorithm:

1. **Initialize the Circular Queue:**
   - Input the capacity cap of the queue.
   - Set `front` and `rear` to 0, and create an array `cq[]` of size cap.
2. **Push (Enqueue) Operation:**
   - Check if the queue is full by verifying if `(rear + 1) % cap == front`.
     - If full, display "QUEUE IS FULL".
     - If not full, add the integer n to the position rear in the array.
     - Increment rear circularly: `rear = (rear + 1) % cap`.
3. **Pop (Dequeue) Operation:**
   - Check if the queue is empty by verifying if front == rear.
     - If empty, return `-9999`.
     - If not empty, retrieve the integer from position front.
     - Increment front circularly: `front = (front + 1) % cap`.
     - Return the dequeued value.
4. **Show Queue Elements:**
   - If `front == rear`, the queue is empty; display "QUEUE IS EMPTY".
   - Else, display elements from front to rear in circular order.
5. **Main Program Logic:**
   - Repeat the following steps until the user chooses to exit:
     - Input the user's choice (push, pop, show, exit).
     - Perform the corresponding action based on the choice.
     - Exit the program if the choice is 4.

Solution:

```
import java.util.Scanner;

public class CirQueue {

  // Instance variables
  int[] cq;
  int cap;
  int front;
  int rear;

  // Constructor to initialize queue with given capacity
  CirQueue(int max) {

    cap = max;
    cq = new int[cap];

    front = 0;
    rear = 0;
  }

  // Method to add an integer to the queue (enqueue)
  void push(int n) {

    if ((rear + 1) % cap == front) {
      System.out.println("QUEUE IS FULL");

    } else {
      // Add to rear position
      cq[rear] = n;

      // Increment rear circularly
      rear = (rear + 1) % cap;
    }
  }
```

```
// Method to remove and return integer from queue (dequeue)
int pop() {

  if (front == rear) {
    // Queue is empty
    return -9999;

  } else {
    // Retrieve front element
    int val = cq[front];

    // Increment front circularly
    front = (front + 1) % cap;
    return val;
  }
}


// Method to display the elements of the queue
void show() {

  if (front == rear) {
    System.out.println("QUEUE IS EMPTY");

  } else {
    System.out.print("Queue elements: ");
    int i = front;

    while (i != rear) {

      System.out.print(cq[i] + " ");
      // Circular increment
      i = (i + 1) % cap;
    }

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

```
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter the capacity of the circular queue: ");
    int max = sc.nextInt();
    CirQueue queue = new CirQueue(max);

    while (true) {
      System.out.println("\n1. Push\n2. Pop\n3. Show\n4. Exit");
      System.out.print("Enter your choice: ");
      int choice = sc.nextInt();
      System.out.println();

      switch (choice) {
        case 1:
          System.out.print("Enter an integer to push: ");
          int value = sc.nextInt();
          queue.push(value);
          break;

        case 2:
          int removedValue = queue.pop();
          if (removedValue == -9999) {
            System.out.println("QUEUE IS EMPTY");
          } else {
            System.out.println("Popped value: " + removedValue);
          }
          break;

        case 3:
          queue.show();
          break;

        case 4:
          sc.close();
          System.exit(0);

        default:
          System.out.println("Invalid choice, try again.");
      }
    }
  }
}
```

Output:

```
Enter the capacity of the circular queue: 3

1. Push
2. Pop
3. Show
4. Exit
Enter your choice: 1
Enter an integer to push: 5

1. Push
2. Pop
3. Show
4. Exit
Enter your choice: 1
Enter an integer to push: 10

1. Push
2. Pop
3. Show
4. Exit
Enter your choice: 1
Enter an integer to push: 15
QUEUE IS FULL

1. Push
2. Pop
3. Show
4. Exit
Enter your choice: 2
Popped value: 5

1. Push
2. Pop
3. Show
4. Exit
Enter your choice: 1
Enter an integer to push: 20

1. Push
2. Pop
3. Show
4. Exit
Enter your choice: 3
Queue elements: 10 20
```

Variable Description Table:

Name	Datatype	Description
cq[]	int[]	Array to store the integers in the circular queue.
cap	int	Stores the maximum capacity of the circular queue array.
front	int	Index that points to the front end of the queue (element to be dequeued next).
rear	int	Index that points to the rear end of the queue (position to enqueue the next element).
n	int	Input integer to be added to the queue in the push() method.
max	int	Input value for the maximum capacity of the circular queue, passed to the constructor.
val	int	Temporary variable used to store the dequeued integer in the pop() method.
choice	int	Variable used in the main() method to store the user's menu choice.
i	int	Loop counter used in the show() method to iterate through the queue elements for display.
removedValue	int	Stores the value returned by the pop() method to check if the queue was empty.

Matrix 7:

A dequeue enables the user to add and remove integers from both the ends i.e. front and rear.

Define a class DeQueue with the following details:

Class name : DeQueue

Data Members:

`ele[]`	:	array to hold the integer elements.
`cap`	:	stores the maximum capacity of the array.
`front`	:	to point the index of the front.
`rear`	:	to point the index of the rear.

Member functions:

`DeQueue(int max)` : constructor to initialize the data member `cap = max, front = rear = 0` and create the integer array.

`void pushfront(int v)` : to add integers from the front index if possible else display the message("full from front").

`int popfront()` : to remove the return elements from front. If array is empty then return `-999`.

`void pushrear(int v)` : to add integers from the front index if possible else display the message("full from rear").

`int poprear()` : to remove and return elements from rear. If the array is empty then return `-999`.

Algorithm:

1. **Initialization**:
   - Create a constructor `DeQueue(int max)` to initialize the front, rear to 0, and cap to max. Create an array `ele[]` of size cap.
2. **Push Front**:
   - Check if the array is full from the front. If yes, display "Full from front".
   - Otherwise, decrement front (circularly) and insert the value at the front position.
3. **Pop Front**:
   - If the array is empty, return `-999`.
   - Otherwise, return the element at the front and increment front (circularly).
4. **Push Rear**:
   - Check if the array is full from the rear. If yes, display "Full from rear".
   - Otherwise, insert the value at the rear and increment rear (circularly).
5. **Pop Rear**:
   - If the array is empty, return `-999`.
   - Otherwise, decrement rear (circularly) and return the value at the rear.

Solution:

```
public class DeQueue {

  // Data members
  int ele[];
  int cap;
  int front;
  int rear;

  // Constructor to initialize the DeQueue
  // with a given maximum capacity
  DeQueue(int max) {
    cap = max;
    ele = new int[cap];
    front = rear = -1;
  }

  // Method to add an element at the front
  void pushfront(int v) {

    // Check if full from the front
    if ((front == 0 && rear == cap-1) || (front == rear+1)) {
      System.out.println("Full from front");

    } else {
      if (front == -1) { // Initially empty DeQueue
        front = rear = 0;
      } else if (front == 0) {
        front = cap - 1; // Wrap around to end
      } else {
        front--; // Move front backward
      }

      ele[front] = v; // Insert element at front
    }
  }
```

```
// Method to remove and return the element from the front
int popfront() {

  // Check if DeQueue is empty
  if (front == -1) {
    return -999; // Return -999 if empty
  }

  int temp = ele[front]; // Store the element to be returned
  if (front == rear) { // Only one element was present
    front = rear = -1;
  } else if (front == cap - 1) {
    front = 0; // Wrap around to start
  } else {
    front++; // Move front forward
  }

  return temp;
}

// Method to add an element at the rear
void pushrear(int v) {

  // Check if full from the rear
  if ((front == 0 && rear == cap-1) || (front == rear+1)) {
    System.out.println("Full from rear");

  } else {
    if (rear == -1) { // Initially empty DeQueue
      front = rear = 0;
    } else if (rear == cap - 1) {
      rear = 0; // Wrap around to start
    } else {
      rear++; // Move rear forward
    }

    ele[rear] = v; // Insert element at rear
  }
}
```

```
  // Method to remove and return the element from the rear
  int poprear() {

    // Check if DeQueue is empty
    if (front == -1) {
      return -999; // Return -999 if empty
    }
    int temp = ele[rear]; // Store the element to be returned
    if (front == rear) { // Only one element was present
      front = rear = -1;
    } else if (rear == 0) {
      rear = cap - 1; // Wrap around to end
    } else {
      rear--; // Move rear backward
    }
    return temp;
  }

  // Main method to test the DeQueue
  public static void main(String[] args) {
    DeQueue dq = new DeQueue(5);

    dq.pushfront(10);
    dq.pushrear(20);
    dq.pushfront(30);
    dq.pushrear(40);
    dq.pushfront(50);

    System.out.println(dq.popfront());
    System.out.println(dq.poprear());
    System.out.println(dq.popfront());
    System.out.println(dq.poprear());
    System.out.println(dq.popfront());
  }
}
```

Output:

```
50
40
30
20
10
```

Variable Description Table:

Name	Datatype	Description
`ele[]`	`int[]`	Array to store the integers in the DeQueue.
`cap`	`int`	Stores the maximum capacity of the array.
`front`	`int`	Index to point to the front of the DeQueue.
`rear`	`int`	Index to point to the rear of the DeQueue.
`v`	`int`	Stores the integer value being inserted into the queue.
`max`	`int`	Maximum capacity passed as argument to constructor.

Matrix 8:

Write a program to declare a square matrix `A[][]` of order (M × M) where 'M' must be greater than 3 and less than 10. Allow the user to input positive integers into this matrix. Perform the following tasks on the matrix:

1. Sort the non-boundary elements in ascending order using any standard sorting technique and rearrange them in the matrix.
2. Calculate the sum of both the diagonals.
3. Display the original matrix, rearranged matrix and only the diagonal elements of the rearranged matrix with their sum.

Test your program for the following data and some random data:

Example 1

INPUT:

M = 4

9	2	1	5
8	13	8	4
15	6	3	11
7	12	23	8

OUTPUT:

ORIGINAL MATRIX

9	2	1	5
8	13	8	4
15	6	3	11
7	12	23	8

REARRANGED MATRIX

9	2	1	5
8	3	6	4
15	8	13	11
7	12	23	8

DIAGONAL ELEMENTS

9			5
	3	6	
	8	13	
7			8

SUM OF THE DIAGONAL ELEMENTS = 59

Algorithm:

1. Input the Matrix Size:

- Prompt the user to enter the matrix size m.
- If m is less than 3 or greater than or equal to 10, print an "OUT OF RANGE" message and exit.

2. Input the Matrix Elements:

- Prompt the user to input elements for a square matrix `a[m][m]`.
- Ensure that only positive integers are accepted. If any negative value is input, display "INVALID INPUT" and exit.

3. Display the Original Matrix:

- Use `printMatrix(a, m)` to print the matrix as entered by the user.

4. Sort the Non-Boundary Elements:

- Extract non-boundary elements from the matrix into an array `b[]`.
- Sort the array `b[]` using the bubble sort technique.
- Place the sorted values of `b[]` back into the non-boundary cells of the matrix `a[][]`.

5. Display the Rearranged Matrix:

- Print the rearranged matrix after sorting non-boundary elements using `printMatrix(a, m)`.

6. Compute and Print the Sum of Diagonal Elements:

- Use the `computePrintDiagonalSum(a, m)` method to:
  - Identify and print diagonal elements (both primary and secondary diagonals).
  - Compute the sum of the diagonal elements and display it.

Solution:

```
import java.util.Scanner;

public class MatrixSort {

  // Computes and prints the sum of the diagonal
  // elements in the matrix
  public static void computePrintDiagonalSum(int a[][], int m) {

    int sum = 0;
    System.out.println("DIAGONAL ELEMENTS");
    for (int i = 0; i < m; i++) {
      for (int j = 0; j < m; j++) {

        if (i == j || i + j == m - 1) {
          sum += a[i][j];
          System.out.print(a[i][j] + "\t");
        } else {
          System.out.print("\t");
        }
      }
      System.out.println();
    }

    System.out.println("SUM OF THE DIAGONAL ELEMENTS = " + sum);
  }

  // Method to print Matrix
  public static void printMatrix(int a[][], int m) {

    for (int i = 0; i < m; i++) {
      for (int j = 0; j < m; j++) {
        System.out.print(a[i][j] + "\t");
      }
      System.out.println();
    }
  }
```

```
// Sorts the non-boundary elements of
// the matrix in ascending order
public static void sortNonBoundaryMatrix(int a[][], int m) {

  int b[] = new int[(m - 2) * (m - 2)];
  int k = 0;
  for (int i = 1; i < m - 1; i++) {
    for (int j = 1; j < m - 1; j++) {
      b[k++] = a[i][j];
    }
  }

  for (int i = 0; i < k - 1; i++) {
    for (int j = 0; j < k - i - 1; j++) {
      if (b[j] > b[j + 1]) {
        int t = b[j];
        b[j] = b[j + 1];
        b[j + 1] = t;
      }
    }
  }

  k = 0;
  for (int i = 1; i < m - 1; i++) {
    for (int j = 1; j < m - 1; j++) {
      a[i][j] = b[k++];
    }
  }
}

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

  System.out.print("ENTER MATRIX SIZE (M): ");
  int m = in.nextInt();

  if (m <= 3 || m >= 10) {
    System.out.println("THE MATRIX SIZE IS OUT OF RANGE.");
    in.close();
    return;
  }
```

```
    int a[][] = new int[m][m];

    System.out.println("ENTER ELEMENTS OF MATRIX");
    for (int i = 0; i < m; i++) {

      for (int j = 0; j < m; j++) {

        a[i][j] = in.nextInt();

        // Only positive integers should be enetered
        if (a[i][j] < 0) {

          System.out.println("INVALID INPUT");
          in.close();
          return;
        }
      }
    }

    in.close();

    System.out.println("ORIGINAL MATRIX");
    printMatrix(a, m);

    sortNonBoundaryMatrix(a, m);
    System.out.println("REARRANGED MATRIX");
    printMatrix(a, m);

    computePrintDiagonalSum(a, m);
  }
}
```

Output:

```
ENTER MATRIX SIZE (M): 4
ENTER ELEMENTS OF MATRIX
1 9 2 8
3 7 4 6
10 5 11 12
23 13 8 15
ORIGINAL MATRIX
1       9       2       8
3       7       4       6
10      5       11      12
23      13      8       15
REARRANGED MATRIX
1       9       2       8
3       4       5       6
10      7       11      12
23      13      8       15
DIAGONAL ELEMENTS
1                       8
        4       5
        7       11
23                      15
SUM OF THE DIAGONAL ELEMENTS = 74
```

```
ENTER MATRIX SIZE (M): 3
THE MATRIX SIZE IS OUT OF RANGE.
```

```
ENTER MATRIX SIZE (M): 10
THE MATRIX SIZE IS OUT OF RANGE.
```

```
ENTER MATRIX SIZE (M): 5
ENTER ELEMENTS OF MATRIX
1 2 3 4 5
6 -7 8 9 -10
INVALID INPUT
```

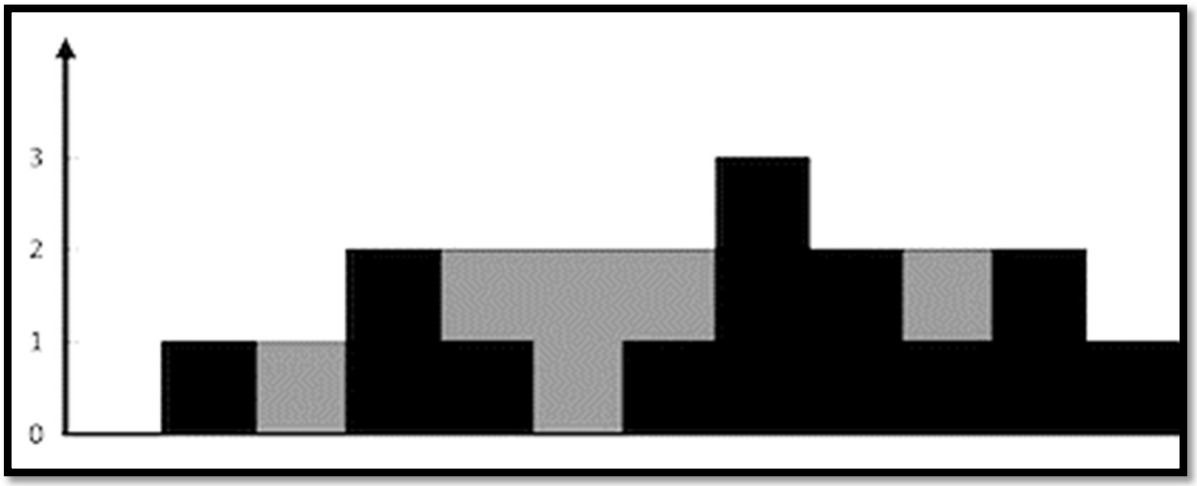
Variable Description Table:

Name	Datatype	Description
a	int[][]	2D array storing the matrix input by the user.
m	int	Size of the matrix (number of rows and columns).
b	int[]	1D array storing non-boundary elements of the matrix for sorting.
k	int	Counter used to index into the b[] array during extraction and sorting.
sum	int	Stores the sum of diagonal elements of the matrix.
i, j	int	Loop variables used to traverse the matrix rows and columns.
t	int	Temporary variable used for swapping elements in the bubble sort.
divisor	int	Used to shift digits while generating cyclic permutations (in the circular prime example).

Matrix 9:

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Example 1:



Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6

Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (grey section) are being trapped.

Example 2:

Input: height = [4,2,0,3,2,5]
Output: 9

Algorithm:

1. **Initialize**:
   - Set `n` as the length of the array.
   - Create two arrays `leftMax` and `rightMax` of size n.
   - Create a variable `waterTrapped` and initialize it to 0.
2. **Calculate Left Maximums**:
   - Set `leftMax[0]` to `height[0]`.
   - Iterate from the second element to the end of the array:
     - `leftMax[i]` = `max(leftMax[i-1], height[i])`.
3. **Calculate Right Maximums**:
   - Set `rightMax[n-1]` to `height[n-1]`.
   - Iterate from the second-to-last element to the beginning of the array:
     - `rightMax[i]` = `max(rightMax[i+1], height[i])`.
4. **Calculate Trapped Water**:
   - Iterate through the height array:
     - At each index `i`, calculate the trapped water as `min(leftMax[i], rightMax[i]) - height[i]`.
     - Add this value to `waterTrapped`.
5. **Return the Result**:
   - Output `waterTrapped`.

Solution:

```
public class RainWaterTrapping {

  public static int trap(int[] height) {
    int n = height.length;
    if (n == 0)
      return 0;

    int[] leftMax = new int[n];
    int[] rightMax = new int[n];
    int waterTrapped = 0;

    // Fill leftMax array
    leftMax[0] = height[0];
    for (int i = 1; i < n; i++) {
      leftMax[i] = Math.max(leftMax[i - 1], height[i]);
    }

    // Fill rightMax array
    rightMax[n - 1] = height[n - 1];
    for (int i = n - 2; i >= 0; i--) {
      rightMax[i] = Math.max(rightMax[i + 1], height[i]);
    }

    // Calculate trapped water
    for (int i = 0; i < n; i++) {
      waterTrapped += Math.min(leftMax[i], rightMax[i]) - height[i];
    }

    return waterTrapped;
  }

  public static void main(String[] args) {
    int[] height = { 0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1 };

    System.out.println("Trapped water: " + trap(height));
  }
}
```

Output:

```
int[] height = {1, 3, 2, 4, 1, 0, 1, 3};

Trapped water: 8
```

```
int[] height = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1};

Trapped water: 6
```

```
int[] height = {4, 4, 4, 4};

Trapped water: 0
```

Variable Description Table:

Name	Datatype	Description
`height`	`int[]`	Array representing the height of bars.
`n`	`int`	Length of the height array.
`leftMax`	`int[]`	Array storing the max height to the left of each bar.
`rightMax`	`int[]`	Array storing the max height to the right of each bar.
`waterTrapped`	`int`	Stores the total water trapped.
`i`	`int`	Loop counter used for iterating through arrays.

Matrix 10:

Design a class MatRev to reverse each element of a matrix. Example:

Some of the members of the class are given below:

72	371	5
12	6	426
5	123	94

Becomes

27	173	5
21	6	624
5	321	49

Class name: MatRev

Data members/instance variables:

`arr[][]` : to store integer elements
`m` : to store the number of rows
`n` : to store the number of columns

Member functions/methods:

`MatRev(int mm, int nn)` : parameterised constructor to initialise the data members `m = mm` and `n = nn`

`void fillarray()` : to enter elements in the array.

`int reverse(int x)` : returns the reverse of the number x.

`void revMat(MatRev P)` : reverses each element of the array of the parameterized object and stores it in the array of the current object.

`void show()` : displays the array elements in matrix form.

Algorithm:

1. **Initialize the Class**:
   - In the parameterized constructor, initialize m and n with the given values.
   - Declare and initialize a 2D array `arr` of size m x n.

2. **Fill the Array**:
   - In the `fillarray()` method, prompt the user to input elements for the m x n matrix.
   - Store the input values in the array arr.

3. **Reverse a Number**:
   - Define the `reverse(int x)` method to reverse the digits of the integer x.
   - Use a loop to extract digits and build the reversed number.

4. **Reverse the Matrix Elements**:
   - In the `revMat(MatRev P)` method, loop through the elements of the parameter matrix P.
   - For each element, call the `reverse()` method to get its reversed value.
   - Store the reversed value in the corresponding position of the current matrix.

5. **Display the Matrix**:
   - In the `show()` method, print the matrix in a formatted form, row by row.

Solution:

```
import java.util.Scanner;

public class MatRev {

  // Instance variables
  private int[][] arr;
  private int m, n;

  MatRev(int mm, int nn) {
    m = mm; n = nn;
    arr = new int[m][n];
  }

  // Method to fill the array with user inputs
  public void fillarray() {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the elements of the matrix (" +
                        m + "x" + n + "):");
    for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
        arr[i][j] = sc.nextInt();
      }
    }
  }

  // Method to reverse a number
  public int reverse(int x) {
    int rev = 0;
    while (x != 0) {
      rev = rev * 10 + x % 10;
      x /= 10;
    }
    return rev;
  }
```

```
  // Method to reverse each element of the matrix
  public void revMat(MatRev P) {
    for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
        arr[i][j] = reverse(P.arr[i][j]);
      }
    }
  }

  // Method to display the matrix in a formatted way
  public void show() {
    for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
        System.out.print(arr[i][j] + "\t");
      }
      System.out.println();
    }
  }

  // Main method to demonstrate the functionality
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    // Create the first matrix object
    System.out.print("Enter the number of rows: ");
    int rows = sc.nextInt();
    System.out.print("Enter the number of columns: ");
    int cols = sc.nextInt();

    MatRev original = new MatRev(rows, cols);
    original.fillarray();

    MatRev reversed = new MatRev(rows, cols);
    reversed.revMat(original);

    System.out.println("Original Matrix:");
    original.show();
    System.out.println("Reversed Matrix:");
    reversed.show();
  }
}
```

Output:

```
Enter the number of rows: 2
Enter the number of columns: 2
Enter the elements of the matrix (2x2):
23 64
84 57
Original Matrix:
23      64
84      57
Reversed Matrix:
32      46
48      75
```

```
Enter the number of rows: 2
Enter the number of columns: 3
Enter the elements of the matrix (2x3):
12 34 56
78 90 11
Original Matrix:
12      34      56
78      90      11
Reversed Matrix:
21      43      65
87      9       11
```

```
Enter the number of rows: 1
Enter the number of columns: 1
Enter the elements of the matrix (1x1):
123
Original Matrix:
123
Reversed Matrix:
321
```

Variable Description Table:

Name	Datatype	Description
`arr`	`int[][]`	Stores the elements of the matrix in a 2D array.
`m`	`int`	Stores the number of rows in the matrix.
`n`	`int`	Stores the number of columns in the matrix.
`x`	`int`	Stores the current number to be reversed in the reverse() method.
`rev`	`int`	Holds the reversed value of a number.
`i, j`	`int`	Loop control variables for iterating through the matrix elements.
`P`	`MatRev`	Represents the object from which elements are reversed.

String Programs

String 1:

Write a program to accept a sentence which may be terminated by either '.', '?' or '!' only. The words are to be separated by a single blank space and are in UPPER CASE.

Perform the following tasks:

1. Check for the validity of the accepted sentence only for the terminating character.
2. Arrange the words in ascending order of their length. If two or more words have the same length, then sort them alphabetically.
3. Display the original sentence along with the converted sentence.

Test your program for the following data and some random data:

Example 1:
INPUT:
AS YOU SOW SO SHALL YOU REAP.

OUTPUT:
AS YOU SOW SO SHALL YOU REAP.
AS SO SOW YOU YOU REAP SHALL

Example 2:
INPUT:
SELF HELP IS THE BEST HELP.

OUTPUT:
SELF HELP IS THE BEST HELP.
IS THE BEST HELP HELP SELF

Example 3:
INPUT:
BE KIND TO OTHERS.

OUTPUT:
BE KIND TO OTHERS.
BE TO KIND OTHERS

Example 4:
INPUT:
NOTHING IS IMPOSSIBLE#

OUTPUT:
INVALID INPUT

Algorithm:

1. **Input the Sentence**:
   - Read a sentence from the user using the Scanner class.
   - The sentence must be in uppercase and can only end with '.', '?', or '!' to be considered valid.
2. **Validate the Sentence**:
   - Use the `isValidString()` function to check the following conditions:
     - The sentence should not be null or empty.
     - The last character must be either '.', '?', or '!'.
   - If the sentence does not meet these conditions, print "INVALID INPUT" and terminate the program.
3. **Print the Original Sentence**:
   - If the sentence is valid, print the original sentence.
4. **Remove the Terminating Character**:
   - Remove the last character ('.', '?', or '!') from the input sentence for further processing.
5. **Split the Sentence into Words**:
   - Split the sentence into words using space (" ") as the delimiter.
6. **Sort the Words**:
   - Sort the words by their length in ascending order.
   - If two words have the same length, sort them alphabetically.
   - The `sortString()` function handles the sorting using a modified bubble sort approach and the swap() function to swap words.
7. **Concatenate the Sorted Words**:
   - After sorting, concatenate the words back into a single sentence with a space between each word.
8. **Print the Sorted Sentence**:
   - Display the sentence with words arranged based on the sorting criteria.

Solution:

```
import java.util.Scanner;

public class StringCheck {

  // Method to check if the sentence ends with '.', '?', or '!'
  public static boolean isValidString(String str) {
    int len = str.length();
    char last_ch = str.charAt(len - 1);
    if (str == null || len == 0 ||
      (last_ch != '.' && last_ch != '?' && last_ch != '!')) {
      return false;
    }
    return true;
  }

  // Method to swap two elements in an array
  public static void swap(String[] words, int i, int j) {
    String temp = words[i];
    words[i] = words[j];
    words[j] = temp;
  }

  // Method to sort words by length, then alphabetically
  public static String sortString(String ipStr) {

    String words[] = ipStr.split(" ");
    int wordCount = words.length;

    for (int i = 0; i < wordCount - 1; i++) {
      for (int j = i + 1; j < wordCount; j++) {
        if (words[i].length() > words[j].length() ||
           (words[i].length() == words[j].length() &&
            words[i].compareTo(words[j]) > 0)) {

          swap(words, i, j);
        }
      }
    }
```

```
    String sorted = ""; // string to be returned
    for (int i = 0; i < wordCount - 1; i++) {
      sorted += words[i] + " ";
    }
    // Append last word without space
    sorted += words[wordCount - 1];

    return sorted;
  }

  public static void main(String args[]) {
    Scanner in = new Scanner(System.in);
    System.out.println("Enter a sentence:");
    String str = in.nextLine();
    int len = str.length();
    System.out.println();
    in.close();

    if (!isValidString(str)) {
      System.out.println("INVALID INPUT");
      return;
    }

    System.out.println(str);

    // Remove the terminating character ('.', '?', '!')
    str = str.substring(0, len - 1);

    String sortedStr = sortString(str);
    System.out.println(sortedStr);
  }
}
```

Output:

```
Enter a sentence:
The quick brown fox jumps over the lazy dog.

The quick brown fox jumps over the lazy dog.
The dog fox the lazy over brown jumps quick
```

```
Enter a sentence:
This is a valid test sentence!

This is a valid test sentence!
a is This test valid sentence
```

```
Enter a sentence:
This is not a valid test sentence

INVALID INPUT
```

Variable Description Table:

Name	Datatype	Description
`str`	`String`	Stores the sentence input by the user.
`words`	`String[]`	Stores the array of words split from the sentence.
`wordCount`	`int`	Holds the number of words in the sentence.
`last_ch`	`char`	Stores the last character of the input sentence for validation.
`len`	`int`	Holds the length of the input sentence.
`sorted`	`String`	Stores the concatenated string of sorted words.
`temp`	`String`	Temporary variable for swapping words during sorting.
`i, j`	`int`	Loop variables used for sorting and word processing.

String 2:

Write a program to accept a sentence which may be terminated by either '.', '?' or '!' only. The words are to be separated by a single blank space and are in uppercase.

Perform the following tasks:

(a) Check for the validity of the accepted sentence.

(b) Convert the non-palindrome words of the sentence into palindrome words by concatenating the word by its reverse (excluding the last character).

Example:

The reverse of the word HELP would be LEH (omitting the last alphabet) and by concatenating both, the new palindrome word is HELPLEH. Thus, the word HELP becomes HELPLEH.

Note: The words which end with repeated alphabets, for example ABB would become ABBA and not ABBBA and XAZZZ becomes XAZZZAX.

Palindrome word: Spells same from either side. Example: DAD, MADAM etc.

(c) Display the original sentence along with the converted sentence.

Test your program for the following data and some random data:

Example 1

INPUT:
THE BIRD IS FLYING.

OUTPUT:
THE BIRD IS FLYING.
THEHT BIRDRIB ISI FLYINGNIYLF

Example 2

INPUT:
IS THE WATER LEVEL RISING?

OUTPUT:
IS THE WATER LEVEL RISING?
ISI THEHT WATERETAW LEVEL RISINGNISIR

Example 3

INPUT:
THIS MOBILE APP LOOKS FINE.

OUTPUT:
THIS MOBILE APP LOOKS FINE.
THISIHT MOBILELIBOM APPA LOOKSKOOL FINENIF

Example 3

INPUT:
YOU MUST BE CRAZY#

OUTPUT:
INVALID INPUT

Algorithm:

1. **Input Validation**:
    - Accept a sentence from the user.
    - Convert the sentence to uppercase.
    - Check if the sentence ends with a valid punctuation mark (i.e., ., ?, or !). If not, display "INVALID INPUT" and exit.
2. **Process Each Word**:
    - Remove the terminating punctuation.
    - Split the sentence into individual words using space as the delimiter.
3. **Palindrome Check**:
    - For each word, check if it's a palindrome:
        - If a word reads the same forwards and backwards, it's a palindrome.
    - If the word is not a palindrome, convert it into one by reversing it (excluding the last character) and concatenating it to the original word. Handle words that end with repeated characters specially (i.e., avoid repeating the same character multiple times).
4. **Concatenate the Converted Words**:
    - Construct a new sentence by joining the converted palindrome words.
5. **Display**:
    - Display the original sentence and the transformed sentence.

Solution:

```
import java.util.Scanner;

public class Palindrome {

  // Function to check if a word is a palindrome
  public static boolean isPalindrome(String word) {
    int len = word.length();

    for (int i = 0; i < len / 2; i++) {
      if (word.charAt(i) != word.charAt(len - 1 - i)) {
        return false;
      }
    }
    return true;
  }

  // Function to make a word palindrome character
  public static String makePalindrome(String word) {

    int len = word.length(), i = len - 1;;
    char lastChar = word.charAt(len - 1);

    // Find the first character from the end
    // that is not equal to the last character
    while (i > 0 && word.charAt(i) == lastChar) {i--;}

    // Appending the reverse (up to character i)
    String result = word;
    for (int j = i; j >= 0; j--) {
      result += word.charAt(j);
    }

    return result;
  }
```

```
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.println("ENTER THE SENTENCE:");
    String ipStr = in.nextLine().trim().toUpperCase();
    int len = ipStr.length();
    in.close();

    // Check if the input sentence ends with '.', '?', or '!'
    char lastChar = ipStr.charAt(len - 1);
    if (lastChar != '.' && lastChar != '?' && lastChar != '!')
{
      System.out.println("INVALID INPUT");
      return;
    }

    // Remove the punctuation at the end
    String str = ipStr.substring(0, len - 1);

    // Split the sentence into words based on spaces
    String[] words = str.split(" ");
    String result = "";

    for (int i = 0; i < words.length; i++) {
      String word = words[i];

      // If the word is already a palindrome, keep it as is
      result += (isPalindrome(word) ? word :
makePalindrome(word)) + " ";
    }

    // Output the original sentence with punctuation then
result
    System.out.println("\n" + ipStr);
    System.out.println(result);
  }
}
```

Output:

```
ENTER THE SENTENCE:
Hello world.

HELLO WORLD.
HELLOLLEH WORLDLROW
```

```
ENTER THE SENTENCE:
Racecar is awesome!

RACECAR IS AWESOME!
RACECAR ISI AWESOMEMOSEWA
```

```
ENTER THE SENTENCE:
This is a test
INVALID INPUT
```

```
ENTER THE SENTENCE:
WOW MOM RACECAR!

WOW MOM RACECAR!
WOW MOM RACECAR
```

Variable Description Table:

Name	Datatype	Description
`ipStr`	`String`	Input sentence entered by the user, trimmed and converted to uppercase.
`len`	`int`	Length of the input sentence.
`lastChar`	`char`	Last character of the input sentence ('.', '!', or '?').
`str`	`String`	Sentence excluding the last character (punctuation).
`words[]`	`String[]`	Array of words obtained by splitting str based on spaces.
`isPalinWord`	`boolean`	Indicates if the current word is a palindrome.
`palinWord`	`String`	Palindrome version of the current word.
`i`	`int`	Loop index for iterating over the words and characters.
`result`	`String`	Sentence after converting non-palindrome words into palindrome words.

String 3:

Caesar Cipher is an encryption technique which is implemented as ROT13 ('rotate by 13 places'). It is a simple letter substitution cipher that replaces a letter with the letter 13 places after it in the alphabets, with the other characters remaining unchanged.

ROT13

A a	B b	C c	D d	E e	F f	G g	H h	I i	J j	K k	L l	M m
↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
N n	O o	P p	Q q	R r	S s	T t	U u	V v	W w	X x	Y y	Z z

Write a program to accept a plain text of length L, where L must be greater than 3 and less than 100.

Encrypt the text if valid as per the Caesar Cipher.

Test your program with the sample data and some random data:

Example 1

INPUT:

Hello! How are you?

OUTPUT:

The cipher text is:
Uryyb! Ubj ner lbh?

Example 2

INPUT:

Encryption helps to secure data.

OUTPUT:

The cipher text is:
Rapelcgvba urycf gb frpher qngn.

Example 3

INPUT:

You

OUTPUT:

INVALID LENGTH

Algorithm:

1. **Input plain text**:
   - Read a string (plain text) from the user.
2. **Check text length**:
   - Ensure the length of the text is greater than 3 and less than 100. If not, display "INVALID LENGTH" and terminate.
3. **Initialize result string**:
   - Prepare an empty string to store the encrypted text.
4. **Loop through each character in the plain text**:
   - If the character is an uppercase letter between 'A' and 'M' or lowercase between 'a' and 'm', add 13 to its ASCII value.
   - If the character is an uppercase letter between 'N' and 'Z' or lowercase between 'n' and 'z', subtract 13 from its ASCII value.
   - If the character is not a letter (non-alphabetic), leave it unchanged.
5. **Store encrypted characters**:
   - Append each processed character to the result string.
6. **Output the cipher text**:
   - Print the final encrypted string.

Solution:

```
import java.util.Scanner;

public class CaesarCipher {
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.println("Enter plain text:");
    String str = in.nextLine();
    in.close();

    int len = str.length();
    if (len <= 3 || len >= 100) {  // Validate length
      System.out.println("INVALID LENGTH");
      return;
    }

    // Initialize an empty string to store cipher text
    String cipher = "";

    // Loop through each character in the plain text
    for (int i = 0; i < len; i++) {
      char ch = str.charAt(i);

      if  ((ch >= 'A' && ch <= 'M') ||
           (ch >= 'a' && ch <= 'm'))   {
        // Rotate by +13 for A-M or a-m
        cipher += (char) (ch + 13);
      } else if ((ch >= 'N' && ch <= 'Z') ||
                 (ch >= 'n' && ch <= 'z'))   {
        // Rotate by -13 for N-Z or n-z
        cipher += (char) (ch - 13);
      } else {
        // Leave non-alphabetic characters unchanged
        cipher += ch;
      }
    }

    System.out.println("The cipher text is:\n" + cipher);
  }
}
```

Output:

```
Enter plain text:
HelloWorld!
The cipher text is:
UryybJbeyq!
```

```
Enter plain text:
Hi
INVALID LENGTH
```

```
Enter plain text:
This Is A Very Long String That Exceeds The Character Limit Of
One Hundred So It Should Be Invalid In This Case.
INVALID LENGTH
```

```
Enter plain text:
But this does not exceed the limit!!
The cipher text is:
Ohg guvf qbrf abg rkprrq gur yvzvg!!
```

Variable Description Table:

Name	Datatype	Description
`str`	`String`	Stores the user input plain text
`len`	`int`	Length of the input plain text
`ch`	`char`	Holds each character of the string while processing
`i`	`int`	Loop counter to iterate through each character of the string
`cipher`	`String`	The final encrypted cipher text
`sb`	`String`	The encrypted result as it's being constructed

String 4:

Design a class DateConvert to find the date and the month from a given day number for a particular year.

Example: If day number is 64 and the year is 2020, then the corresponding date would be:

March 4, 2020 i.e. (31 + 29 + 4 = 64)

Some of the members of the class are given below:

Class name: DateConvert

Data members/instance variables:

n	:	integer to store the day number
d	:	integer to store the day of the month (date)
m	:	integer to store the month
y	:	integer to store the year

Methods/Member functions:

`DateConvert()` : constructor to initialize the data members with legal initial values.

`void accept()` : to accept the day number and the year.

`void day_to_date()`: converts the day number to its corresponding date for a particular year and stores the date in 'd' and the month in 'm'.

`void display()` : displays the month name, date and year.

Algorithm:

1. **Initialize Variables:**
   - Create a class `DateConvert` with instance variables: `n` (day number), `d` (day of the month), `m` (month), and `y` (year).
2. **Input:**
   - In the `accept()` method, take user input for the day number (`n`) and the year (`y`), also validating them.
3. **Determine Leap Year:**
   - In the `day_to_date()` method, check if the given year is a leap year. A leap year occurs if:

     The year is divisible by 4, but not divisible by 100, unless divisible by 400.
4. **Set Month Boundaries:**
   - Define arrays for the number of days in each month for leap and non-leap years.
   - Use a loop to find the corresponding month by subtracting the number of days in each month from `n` until `n` becomes less than the number of days in that month.
5. **Determine Date:**
   - The remaining value of `n` will be the date, and the month will be the index of the loop.
6. **Display Result:**
   - In the `display()` method, map the month number to its name and print the date, month name, and year.

Solution:

```
import java.util.Scanner;

public class DateConvert {

  // Instance variables
  int n, d, m, y;

  // Constructor to initialize data members
  DateConvert() {
    n = 0; d = 0; m = 0; y = 0;
  }

  // Method to accept day number and year
  void accept() {
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter the day number: ");
    n = sc.nextInt();
    System.out.print("Enter the year: ");
    y = sc.nextInt();

    int maxDays = isLeapYear(y) ? 366 : 365;

    if (n < 1 || n > maxDays) {
      System.out.println("Invalid input! Day must be in 1 - "
                         + maxDays);
      accept();
    }
    sc.close();
  }

  // Method to check if the year is a leap year
  boolean isLeapYear(int year) {
    return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
  }
```

```
  // Method to convert day number to month and date
  void day_to_date() {
    // Days in months for non-leap years and leap years
    int[] daysInMonths = { 31, 28, 31, 30, 31, 30,
                           31, 31, 30, 31, 30, 31 };
    // Update February days if leaf year
    daysInMonths[1] = isLeapYear(y) ? 1 : 0;

    // Determine the month and date
    int remainingDays = n;
    for (int i = 0; i < 12; i++) {
      if (remainingDays <= daysInMonths[i]) {
        m = i + 1; // Month number
        d = remainingDays; // Day of the month
        break;
      } else {
        remainingDays -= daysInMonths[i];
      }
    }
  }

  // Method to display the date in month name format
  void display() {
    // Month names array
    String[] monthNames = {
        "January", "February", "March", "April",
        "May", "June", "July", "August",
        "September", "October", "November", "December"
    };
    // Display the result
    System.out.println(monthNames[m-1] + " " + d + ", " + y);
  }

  // Main method to run the program
  public static void main(String[] args) {
    DateConvert obj = new DateConvert();

    obj.accept();
    obj.day_to_date();
    obj.display();
  }
}
```

Output:

```
Enter the day number: 61
Enter the year: 2023
March 2, 2023
```

```
Enter the day number: 60
Enter the year: 2024
February 29, 2024
```

```
Enter the day number: 366
Enter the year: 2024
December 31, 2024
```

```
Enter the day number: 366
Enter the year: 2023
Invalid input! Please enter a day number between 1 - 365.
Enter the day number: 365
Enter the year: 2023
December 31, 2023
```

Variable Description Table:

Name	Datatype	Description
`n`	`int`	Stores the day number provided by the user.
`d`	`int`	Stores the day of the month (calculated).
`m`	`int`	Stores the month number (calculated).
`y`	`int`	Stores the year provided by the user.
`maxdays`	`int`	Maximum days for the given year type.
`daysInMonths`	`int[]`	Array to hold the number of days in each month for a given year type (leap or non-leap).
`monthNames`	`String[]`	Array to hold month names for display.

String 5:

A class Mix has been defined to mix two words, character by character, in the following manner:

The first character of the first word is followed by the first character of the second word and so on. If the words are of different length, the remaining characters of the longer word are put at the end.

Example: If the First word is "JUMP" and the second word is "STROLL", then the required word will be "JSUTMRPOLL"

Some of the members of the class are given below:

Class name: Mix

Data member/instance variable:

`wrd` : to store a word

`len` : to store the length of the word

Member functions/methods:

`Mix()` : default constructor to initialize the data members with legal initial values.

`void feedword()` : to accept the word in UPPER case.

`void mix_word(Mix P, Mix Q)`: mixes the words of objects P and Q as stated above and stores the resultant word in the current object.

`void display()` : displays the word.

Algorithm:

1. **Initialize Variables:**
   - Define the class `Mix` with two instance variables: `wrd` (to store the word) and `len` (to store the length of the word).
   - Create a constructor to initialize `wrd` and `len` with default values.
2. **Feed Word:**
   - In the `feedword()` method, accept a word in uppercase and assign it to `wrd`.
   - Calculate and store the length of the word in `len`.
3. **Mix Word:**
   - Define the `mix_word()` method which takes two Mix objects (P and Q) as parameters.
   - Loop through the characters of both words:
     - Add one character from `P.wrd` followed by one character from `Q.wrd` to the resultant word.
   - If the words are of different lengths, append the remaining characters of the longer word.
   - Store the resultant word in the current object.
4. **Display:**
   - Define the `display()` method to print the mixed word.

Solution:

```
import java.util.Scanner;

public class Mix {

  // Instance variables
  String wrd;
  int len;

  // Constructor to initialize default values
  Mix() {
    wrd = "";
    len = 0;
  }

  // Method to accept a word in UPPER case
  void feedword() {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter the word in UPPER CASE: ");
    // Accept word
    wrd = sc.nextLine();
    // Calculate and store length
    len = wrd.length();
  }

  // Method to mix words of two objects P and Q
  void mix_word(Mix P, Mix Q) {
    // Minimum length of the two words
    int minLen = Math.min(P.len, Q.len);
    String result = ""; // To store the mixed word

    // Mix the characters from both words alternately
    for (int i = 0; i < minLen; i++) {
      result += P.wrd.charAt(i); // Add character from P
      result += Q.wrd.charAt(i); // Add character from Q
    }
```

```
    // Append remaining characters from the longer word
    if (P.len > Q.len) {
      result += P.wrd.substring(minLen);
    } else if (Q.len > P.len) {
      result += Q.wrd.substring(minLen);
    }

    // Store the result in the current object's wrd variable
    this.wrd = result;
    this.len = result.length();
  }

  // Method to display the mixed word
  void display() {
    System.out.println("The mixed word is: " + wrd);
  }

  // Main method
  public static void main(String[] args) {
    // Create objects for Mix class
    Mix word1 = new Mix();
    Mix word2 = new Mix();
    Mix mixedWord = new Mix();

    // Accept words for word1 and word2
    word1.feedword();
    word2.feedword();

    // Mix the two words
    mixedWord.mix_word(word1, word2);

    // Display the mixed word
    mixedWord.display();

  }
}
```

Output:

```
Enter the word in UPPER CASE: JAVA
Enter the word in UPPER CASE: PYTHON
The mixed word is: JPAYVTAHON
```

```
Enter the word in UPPER CASE: HELLO
Enter the word in UPPER CASE: WORLD
The mixed word is: HWEOLRLLOD
```

Variable Description Table:

Name	Datatype	Description
wrd	String	To store the word in uppercase letters.
len	int	Stores the length of the word.
P	Mix	First object of class Mix containing a word.
Q	Mix	Second object of class Mix containing a word.
result	String	To store the mixed result of words from objects P and Q.
minLen	int	The minimum length between the two words.
i	int	Loop counter to iterate through characters in the words.

String 6:

Design a class StringSort which enables multiple words to be sorted in alphabetical order, both within each word and among the words as well. It also allows for case-insensitive sorting. Additionally, the class should handle a sentence, where each word is first sorted alphabetically, and then the entire sentence is rearranged based on alphabetical order of the words. The class should also provide functionality to handle punctuation and spaces efficiently.

Class Details:

Class Name: StringSort

Data members / instance variables:

`Sentence` : stores the input sentence as a string.

`words[]` : array of words from the sentence.

`wordLen[]` : an array storing the length of each word.

`sortedSentence` : stores the rearranged sentence with words and characters sorted alphabetically.

Member Functions:

`StringSort()` : A default constructor to initialize data members with default values.

`void readSentence()` : Accepts the input sentence from the user and splits it into words.

`void sortWord(String word)` : A utility function to sort individual words alphabetically using a standard sorting technique.

`void arrangeWords()` : Sorts each word and then rearranges the words of the sentence in alphabetical order.

`void display()` : Displays the original sentence along with the fully sorted sentence (characters and words).

Algorithm:

1. **Initialization**:
   - Initialize the sentence and other data members in the constructor.
2. **Read Sentence**:
   - Accept a sentence input from the user.
   - Split the sentence into words and store them in an array `words[]`.
3. **Sort Individual Words**:
   - For each word in the array, sort the characters using a sorting technique like Bubble Sort or Selection Sort.
4. **Sort Entire Sentence**:
   - Once the individual words are sorted, sort the `words[]` array alphabetically (case-insensitive) to rearrange the words.
5. **Display**:
   - Print the original sentence and the sorted sentence, displaying each word and the entire sentence in alphabetical order.

Solution:

```
import java.util.Scanner;

public class StringSort {
  String sentence;
  String words[];
  String sortedSentence;

  StringSort() {
    sentence = "";
    sortedSentence = "";
  }

  void readSentence() {
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter a sentence: ");
    sentence = sc.nextLine();
    sentence = sentence.subString(0, sentence.length() - 1);
    sc.close();

    words = sentence.split(" ");
  }

  String sortWord(String word) {
    char[] temp = word.toCharArray();
    int n = temp.length;

    for (int i = 0; i < n - 1; i++) {
      for (int j = 0; j < n - i - 1; j++) {
        if (temp[j] > temp[j + 1]) {
          char swap = temp[j];
          temp[j] = temp[j + 1];
          temp[j + 1] = swap;
        }
      }
    }
    return new String(temp);
  }
```

```
  void arrangeWords() {
    for (int i = 0; i < words.length; i++) {
      words[i] = sortWord(words[i].toLowerCase());
    }

    for (int i = 0; i < words.length - 1; i++) {
      for (int j = 0; j < words.length - i - 1; j++) {
        if (words[j].compareTo(words[j + 1]) > 0) {
          String swap = words[j];
          words[j] = words[j + 1];
          words[j + 1] = swap;
        }
      }
    }

    sortedSentence = String.join(" ", words);
  }

  void display() {
    System.out.println("Original Sentence:\n" + sentence);
    System.out.println("Sorted Sentence (Words and Characters):\n" +
                        sortedSentence);
  }

  public static void main(String[] args) {
    StringSort sorter = new StringSort();

    sorter.readSentence();

    sorter.arrangeWords();

    sorter.display();
  }
}
```

Output:

```
Enter a sentence: The quick brown fox jumps.
Original Sentence:
The quick brown fox jumps
Sorted Sentence (Words and Characters):
bnorw cikqu eht fox jmpsu
```

```
Enter a sentence: Hello World Amazing JAVA
Original Sentence:
Hello World Amazing JAV
Sorted Sentence (Words and Characters):
aagimnz ajv dlorw ehllo
```

```
Enter a sentence: Hello, world! This is a test.
Original Sentence:
Hello, world! This is a test
Sorted Sentence (Words and Characters):
!dlorw ,ehllo a estt hist is
```

Variable Description Table:

Name	Datatype	Description
`sentence`	`String`	Stores the original sentence input.
`words[]`	`String[]`	Array to store the individual words of the sentence.
`sortedSentence`	`String`	Stores the sentence with alphabetically sorted words and characters.
`wordLen[]`	`int[]`	Array to store the length of each word.
`word`	`String`	Stores the individual word to be sorted.
`temp[]`	`char[]`	Temporary array to store characters of a word for sorting.

String 7:

Write a program to accept a sentence which may be terminated by either '.', '?' or '!' only. The words may be separated by more than one blank space and are in UPPER CASE.
Perform the following tasks:

1. Find the number of words beginning and ending with a vowel.
2. Place the words which begin and end with a vowel at the beginning, followed by the remaining words as they occur in the sentence.

Test your program with the sample data and some random data:

Example 1

INPUT:
ANAMIKA AND SUSAN ARE NEVER GOING TO QUARREL ANYMORE.

OUTPUT:
NUMBER OF WORDS BEGINNING AND ENDING WITH A VOWEL = 3
ANAMIKA ARE ANYMORE AND SUSAN NEVER GOING TO QUARREL

Example 2

INPUT:
YOU MUST AIM TO BE A BETTER PERSON TOMORROW THAN YOU ARE TODAY.

OUTPUT:
NUMBER OF WORDS BEGINNING AND ENDING WITH A VOWEL = 2
A ARE YOU MUST AIM TO BE BETTER PERSON TOMORROW THAN YOU TODAY

INPUT:
HOW ARE YOU@

OUTPUT:
INVALID INPUT

Algorithm:

1. **Input the Sentence:**
   - Read the sentence from the user in uppercase.
   - Remove any extra spaces at the beginning or end of the sentence.
2. **Check Sentence Termination:**
   - Ensure the sentence ends with either a period (.), question mark (?), or exclamation mark (!).
   - If it doesn't end with one of these, print "INVALID INPUT" and stop the program.
3. **Remove the Punctuation:**
   - Remove the last character (punctuation mark) from the sentence.
4. **Initialize Variables:**
   - Initialize a counter (count) to keep track of the number of words that start and end with a vowel.
5. **Process Each Word:**
   - Loop through the sentence and split it manually into words based on spaces.
   - For each word:
     - Check if the first and last characters are vowels.
6. **Combine Results:**
   - Combine the words starting and ending with a vowel (stored in `sbVowel`) with the other words (stored in `sbOther`).
7. **Display Results:**
   - Print the number of words that start and end with a vowel.
   - Print the sentence with vowel words at the beginning, followed by the remaining words.

Solution:

```
import java.util.Scanner;

public class VowelWord {

  public static boolean isVowel(char ch) {
    ch = Character.toUpperCase(ch);
    return (ch == 'A' || ch == 'E' ||
      ch == 'I' || ch == 'O' || ch == 'U');
  }

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

    System.out.println("ENTER THE SENTENCE:");
    String ipStr = in.nextLine().trim().toUpperCase();
    int len = ipStr.length();

    in.close();

    // Check for valid sentence termination
    char lastChar = ipStr.charAt(len - 1);
    if (lastChar != '.' &&
        lastChar != '?' &&
        lastChar != '!') {

      System.out.println("INVALID INPUT");
      return;
    }

    // Remove the punctuation mark from
    // the end of the sentence
    String str = ipStr.substring(0, len - 1).trim();

    // Initialize variables for processing
    String vowelsWords = "";
    String otherWords = "";
    int count = 0;
```

```
    // Split the sentence into words manually
    int start = 0;
    for (int i = 0; i < str.length(); i++) {

      if (str.charAt(i) == ' ' || i == str.length() - 1) {
        String word;

        if (i == str.length() - 1) {
          word = str.substring(start);
        } else {
          word = str.substring(start, i);
        }

        if (!word.isEmpty()) {
          // Check if the word starts and ends with a vowel
          int wordLen = word.length();
          if (isVowel(word.charAt(0)) &&
              isVowel(word.charAt(wordLen - 1))) {
            count++;
            vowelsWords += word + " ";
          } else {
            otherWords += word + " ";
          }
        }
        start = i + 1; // Move to the next word
      }
    }

    // Construct the new string
    String newStr = (vowelsWords + otherWords).trim();

    // Output results
    System.out.println("NUMBER OF WORDS BEGINNING AND ENDING
WITH A VOWEL = " + count);
    System.out.println(newStr);
  }
}
```

Output:

```
ENTER THE SENTENCE:
Apples are amazing.
NUMBER OF WORDS BEGINNING AND ENDING WITH A VOWEL = 1
ARE APPLES AMAZING
```

```
ENTER THE SENTENCE:
Java is an amazing language!
NUMBER OF WORDS BEGINNING AND ENDING WITH A VOWEL = 0
JAVA IS AN AMAZING LANGUAGE
```

```
ENTER THE SENTENCE:
  An orange  apple  is delicious!
NUMBER OF WORDS BEGINNING AND ENDING WITH A VOWEL = 2
ORANGE APPLE AN IS DELICIOUS
```

Variable Description Table:

Name	Datatype	Description
ipStr	String	Input sentence converted to uppercase and trimmed.
len	int	Length of the input string ipStr.
lastChar	char	Last character of the trimmed input string.
str	String	Input string without the final punctuation mark.
vowelsWords	String	Concatenated string of words that start and end with a vowel.
otherWords	String	Concatenated string of words that do not start and end with a vowel.
count	int	Number of words that start and end with a vowel.
start	int	Index where the current word starts in the input string.
i	int	Index used for iterating through the input string.
word	String	Current word extracted from the input string.
wordLen	int	Length of the current word.
ch	char	Character used for checking if it is a vowel.

String 8:

A class Adder has been defined to add any two accepted times. The time is entered in the 24-hour format (HH) as a string and then converted to an integer array representing hours and minutes. Additionally, the class checks if the total hours exceed 24 hours after addition, in which case it resets the hours to fit within the 24-hour format.

Example:

- **Time A** - "06:35" (6 hours 35 minutes)
- **Time B** - "19:50" (19 hours 50 minutes)
- Their sum is - "02:25" (since 26 hours 25 minutes wraps around to 24-hour format)

The details of the class members are given below:

Class name: Adder

Data members (instance variables):

`a[]`: An integer array of size 2 to hold the hours and minutes.

Member functions/methods:

`Adder() :` Constructor to assign 0 to the array elements.
`void readtime(String input) :` Takes a time input in the format "HH" as a string, splits it, and stores hours and minutes in the array.

`void addtime(Adder X, Adder Y) :` Adds the time of the two parameterized objects X and Y, storing the result in the current calling object. If the sum of hours exceeds 24, it wraps around the 24-hour format.

`void disptime() :` Displays the array elements with an appropriate message (hours = and minutes =).

Algorithm:

1. **Class Initialization (Adder Constructor):**
   - Initialize an integer array a with size 2.
   - Set a[0] (hours) and a[1] (minutes) to 0.

2. **Reading Time (`readtime` Method):**
   - Accept a time in the format "HH" as a string parameter.
   - Split the string using ":" to extract hours and minutes.
   - Convert these substrings to integers and store them in the array a.

3. **Adding Time (`addtime` Method):**
   - Sum the hours (`X.a[0] + Y.a[0]`) and minutes (`X.a[1] + Y.a[1]`) from the objects X and Y.
   - If the minutes sum is 60 or more, convert the excess into hours by:
     - Incrementing the hour sum by minutes / 60.
     - Setting the new minutes to minutes % 60.
   - If the total hours exceed 24, set the hours to hours % 24 to wrap around to the 24-hour format.
   - Store the calculated hours and minutes in the current object (`this.a[0] and this.a[1]`).

4. **Displaying Time (`disptime` Method):**
   - Print the current object's hours and minutes with the message: "hours = X, minutes = Y".

Solution:

```
import java.util.Scanner;

public class Adder {

   // Array to store hours (a[0]) and minutes (a[1])
  int[] time = new int[2];

  // Constructor to initialize hours and minutes to 0
  public Adder() {
    time[0] = 0; // hours
    time[1] = 0; // minutes
  }

  // Method to read and split time input (HH:MM)
  // into hours and minutes
  public void readtime(String input) {

    String[] timeParts = input.split(":");
    time[0] = Integer.parseInt(timeParts[0]); // hours
    time[1] = Integer.parseInt(timeParts[1]); // minutes

    if (time[0] < 0 || time[0] > 23 ||
        time[1] < 0 || time[1] > 59) {

      System.out.println("Invalid time input");
      System.exit(0);
    }
  }

  // Method to add time from two Adder objects
  public void addtime(Adder X, Adder Y) {
    int hoursSum = X.time[0] + Y.time[0];
    int minutesSum = X.time[1] + Y.time[1];

    if (minutesSum >= 60) {
      hoursSum += minutesSum / 60;
      minutesSum = minutesSum % 60;
    }
```

```
    if (hoursSum >= 24) {
      hoursSum = hoursSum % 24;
    }

    this.time[0] = hoursSum;
    this.time[1] = minutesSum;
  }

  // Method to display the time in hours and minutes
  public void disptime() {
    System.out.println("hours = " + time[0] +
                       ", minutes = " + time[1]);
  }

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

    Adder time1 = new Adder();
    Adder time2 = new Adder();
    Adder sumTime = new Adder();

    System.out.print("Enter the first time (HH:MM): ");
    time1.readtime(sc.nextLine());
    System.out.print("Enter the second time (HH:MM): ");
    time2.readtime(sc.nextLine());

    sumTime.addtime(time1, time2);

    System.out.println("The sum of the two times is:");
    sumTime.disptime();

    sc.close();
  }
}
```

Output:

```
Enter the first time (HH:MM): 10:30
Enter the second time (HH:MM): 12:15
The sum of the two times is:
hours = 22, minutes = 45
```

```
Enter the first time (HH:MM): 11:45
Enter the second time (HH:MM): 02:30
The sum of the two times is:
hours = 14, minutes = 15
```

```
Enter the first time (HH:MM): 20:50
Enter the second time (HH:MM): 5:30
The sum of the two times is:
hours = 2, minutes = 20
```

```
Enter the first time (HH:MM): 25:30
Invalid time input
```

Variable Description Table:

Name	Datatype	Description
a	int[]	Array of size 2 to store hours (a[0]) and minutes (a[1]).
input	String	String input representing the time in "HH" format.
X	Adder	First Adder object parameter used in addtime().
Y	Adder	Second Adder object parameter used in addtime().
hours	int	Temporary variable to store the sum of hours.
minutes	int	Temporary variable to store the sum of minutes.
minutesSum	int	Sum of minutes from X and Y in addtime() method.
hoursSum	int	Sum of hours from X and Y in addtime() method.

String 9:

Design a program to accept a day number (between 1 and 366), year (in 4 digits) from the user to generate and display the corresponding date. Also, accept 'N' (1 <= N <= 100) from the user to compute and display the future date corresponding to 'N' days after the generated date.

Display an error message if the value of the day number, year and N are not within the limit or not according to the condition specified.

Test your program with the following data and some random data:

Example 1

INPUT:
DAY NUMBER: 255
YEAR: 2018
DATE AFTER (N DAYS): 22

OUTPUT:
DATE: 12TH SEPTEMBER, 2018
DATE AFTER 22 DAYS: 4TH OCTOBER, 2018

Example 2

INPUT:
DAY NUMBER: 360
YEAR: 2018
DATE AFTER (N DAYS): 45

OUTPUT:
DATE: 26TH DECEMBER, 2018
DATE AFTER 45 DAYS: 9TH FEBRUARY, 2019

Example 3

INPUT:
DAY NUMBER: 500
YEAR: 2018
DATE AFTER (N DAYS): 33

OUTPUT:
DAY NUMBER OUT OF RANGE

Example 4

INPUT:
DAY NUMBER: 150
YEAR: 2018
DATE AFTER (N DAYS): 330

OUTPUT:
DATE AFTER (N DAYS) OUT OF RANGE

Algorithm:

1. **Input Validation**:
   - Accept `dayNumber`, `year`, and `N` from the user.
   - Check if `dayNumber` is between 1 and 366 for a leap year, and 1 and 365 for a non-leap year.
   - Check if `N` is between 1 and 100.
   - If the input is invalid, display an appropriate error message and terminate.
2. **Leap Year Calculation**:
   - A year is a leap year if it is divisible by 4 but not divisible by 100 unless divisible by 400.
3. **Date Calculation**:
   - Create an array of days in each month for leap years or non-leap years by adding a day to February if leap year.
   - Using the `dayNumber`, calculate the corresponding month and day in that month.
4. **Future Date Calculation**:
   - Add `N` days to the initial date.
   - Adjust the month and year if the number of days exceeds the number of days in the current month.
   - If the month exceeds 12, increment the year accordingly.
5. **Edge Cases**:
   - Ensure that if the date after `N` days exceeds the given year's day limit or `dayNumber` is out of range, appropriate error messages are displayed.

Solution:

```
import java.util.Scanner;

public class DateCalculator {

  // Checks if a year is a leap year.
  public static boolean isLeapYear(int y) {
    return (y % 4 == 0 && y % 100 != 0) || (y % 400 == 0);
  }

  // Computes the date given a day number and year.
  public static String computeDate(int day, int year) {
    int[] monthDays = { 31, 28, 31, 30, 31,
                        30, 31, 31, 30, 31, 30, 31 };
    String[] monthNames = {
      "JANUARY", "FEBRUARY", "MARCH", "APRIL",
      "MAY", "JUNE", "JULY", "AUGUST",
      "SEPTEMBER", "OCTOBER", "NOVEMBER", "DECEMBER"
    };

    // Adjust for February in leap year
    monthDays[1] += isLeapYear(year) ? 1 : 0;

    int i = 0, daySum = 0;
    for (i = 0; i < monthDays.length; i++) {
      daySum += monthDays[i];
      if (daySum >= day) break;
    }
    int date = day + monthDays[i] - daySum;

    return date + getDaySuffix(date) + " " +
           monthNames[i] + ", " + year;
  }
```

```
// Gets the suffix for a day number (ST, ND, RD, TH).
public static String getDaySuffix(int day) {
  if (day >= 11 && day <= 13) {
    return "TH";
  }

  switch (day % 10) {
    case 1:
      return "ST";
    case 2:
      return "ND";
    case 3:
      return "RD";
    default:
      return "TH";
  }
}


public static void main(String[] args) {

  Scanner in = new Scanner(System.in);

  System.out.print("DAY NUMBER: ");
  int dayNum = in.nextInt();

  System.out.print("YEAR: ");
  int year = in.nextInt();

  System.out.print("DATE AFTER (N DAYS): ");
  int n = in.nextInt();

  in.close();

  if (dayNum < 1 || dayNum > 366) {
    System.out.println("DAY NUMBER OUT OF RANGE");
    return;
  }
```

```
    if (n < 1 || n > 100) {
      System.out.println("DATE AFTER (N DAYS) OUT OF RANGE");
      return;
    }

    // Original date calculation
    String dateStr = computeDate(dayNum, year);

    // Future date calculation
    int nDays = dayNum + n;
    int nYear = year;

    boolean leap = isLeapYear(year);


    if (leap && nDays > 366) {
      nYear = nYear + 1;
      nDays = nDays - 366;
    } else if (!leap && nDays > 365) {
      nYear = nYear + 1;
      nDays = nDays - 365;
    }

    String nDateStr = computeDate(nDays, nYear);


    // Output results
    System.out.println();
    System.out.println("DATE: " + dateStr);

    System.out.println("DATE AFTER " + n + " DAYS: " +
                        nDateStr);
  }
}
```

Output:

```
DAY NUMBER: 100
YEAR: 2023
DATE AFTER (N DAYS): 30

DATE: 10TH APRIL, 2023
DATE AFTER 30 DAYS: 10TH MAY, 2023
```

```
DAY NUMBER: 60
YEAR: 2024
DATE AFTER (N DAYS): 5

DATE: 29TH FEBRUARY, 2024
DATE AFTER 5 DAYS: 5TH MARCH, 2024
```

```
DAY NUMBER: 400
YEAR: 2025
DATE AFTER (N DAYS): 10
DAY NUMBER OUT OF RANGE
```

```
DAY NUMBER: 150
YEAR: 2000
DATE AFTER (N DAYS): -5
DATE AFTER (N DAYS) OUT OF RANGE
```

Variable Description Table:

Name	Datatype	Description
dayNum	int	The input day number (1-366 for leap years, 1-365 for others)
year	int	The input year (4 digits)
N	int	Number of days to add to the original date
monthDays	int[]	Array holding the days in each month
monthNames	String[]	Array holding the names of each month
leap	boolean	Indicates if the input year is a leap year
daySum	int	Running sum of days to determine the month
date	int	The final day of the month after calculating
i	int	Loop variable to iterate through the monthDays array
nDays	int	The day number after adding N days
nYear	int	The year for the future date
dateStr	String	Formatted date for the initial day number
nDateStr	String	Formatted date after N days

String 10:

A class Rearrange has been defined to modify a word by bringing all the vowels in the word at the beginning followed by the consonants.
Example: ORIGINAL becomes OIIARGNL

Some of the members of the class are given below:

Class name: Rearrange

Data member/instance variable:

wrd	:	to store a word
newwrd	:	to store the rearranged word

Member functions/methods:

Rearrange( )	: default constructor
void readword( )	: to accept the word in UPPER case
void freq_vow_con( )	: finds the frequency of vowels and consonants in the word and displays them with an appropriate message
void arrange( )	: rearranges the word by bringing the vowels at the beginning followed by consonants
void display( )	: displays the original word along with the rearranged word

Algorithm:

1. **Initialize the Class:**
   - In the constructor, initialize the `wrd` and `newwrd` as empty strings.
2. **Accept Input:**
   - In the `readword()` method, accept a word from the user and convert it to uppercase.
3. **Find Frequency of Vowels and Consonants:**
   - In the `freq_vow_con()` method, loop through the characters of the word.
   - Count how many vowels (A, E, I, O, U) and consonants are present.
4. **Rearrange the Word:**
   - In the `arrange()` method, loop through the word twice:
   - First, collect all the vowels and append them to `newwrd`.
   - Then, collect all the consonants and append them to `newwrd` after the vowels.
5. **Display the Results:**
   - In the `display()` method, print the original word and the rearranged word.

Solution:

```
import java.util.Scanner;

public class Rearrange {

  String wrd;
  String newwrd;

  Rearrange() {
    wrd = "";
    newwrd = "";
  }

  // Method to accept the word in uppercase
  public void readword() {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter a word in UPPER case:");
    wrd = sc.next().toUpperCase();
    sc.close();
  }

  // Method to find and show the freq of vowels and consonants
  public void freq_vow_con() {
    int vowels = 0, consonants = 0;
    // Loop through the characters of the word
    for (int i = 0; i < wrd.length(); i++) {
      char ch = wrd.charAt(i);
      // Check if the character is a vowel
      if (ch == 'A' || ch == 'E' ||
          ch == 'I' || ch == 'O' || ch == 'U') {
        vowels++;
      } else if (ch >= 'A' && ch <= 'Z') consonants++;
    }

    System.out.println("Vowels: " + vowels);
    System.out.println("Consonants: " + consonants);
  }
```

```
  // Method to rearrange word with vowels first, then consonants
  public void arrange() {
    newwrd = ""; // Reset newwrd
    // Add vowels to newwrd first
    for (int i = 0; i < wrd.length(); i++) {
      char ch = wrd.charAt(i);
      if (ch == 'A' || ch == 'E' ||
          ch == 'I' || ch == 'O' || ch == 'U') {
        newwrd += ch;
      }
    }
    // Then add consonants to newwrd
    for (int i = 0; i < wrd.length(); i++) {
      char ch = wrd.charAt(i);
      if (!(ch == 'A' || ch == 'E' ||
          ch == 'I' || ch == 'O' || ch == 'U')) {
        newwrd += ch;
      }
    }
  }

  // Method to display the original word and the rearranged word
  public void display() {
    System.out.println("Original word: " + wrd);
    System.out.println("Rearranged word: " + newwrd);
  }

  public static void main(String[] args) {
    Rearrange obj = new Rearrange();
    obj.readword();
    obj.freq_vow_con();
    obj.arrange();
    obj.display();
  }
}
```

Output:

```
Enter a word in UPPER case:
HELLO
Vowels: 2
Consonants: 3
Original word: HELLO
Rearranged word: EOHLL
```

```
Enter a word in UPPER case:
AEIOU
Vowels: 5
Consonants: 0
Original word: AEIOU
Rearranged word: AEIOU
```

```
Enter a word in UPPER case:
BCDFG
Vowels: 0
Consonants: 5
Original word: BCDFG
Rearranged word: BCDFG
```

Variable Description Table:

Name	Datatype	Description
`wrd`	`String`	Stores the original input word in uppercase.
`newwrd`	`String`	Stores the rearranged word (vowels first, then consonants).
`sc`	`Scanner`	Used to accept input from the user in the readword() method.
`vowels`	`int`	Stores the count of vowels in the word.
`consonants`	`int`	Stores the count of consonants in the word.
`i`	`int`	Loop variable used to iterate through the characters of wrd.
`ch`	`char`	Stores the current character being checked in freq_vow_con().