RAJALAKSHMI ENGINEERING COLLEGE

[AUTONOMOUS]

RAJALAKSHMI NAGAR, THANDALAM – 602 105

RAJALAKSHMI
ENGINEERING COLLEGE

CS23333 OBJECT ORIENTED PROGRAMING USING JAVA

Laboratory Record Note Book

Name : .Sudharsan.D.....

Year / Branch / Section : . . II/IT/D .. . .

College Roll No. : . . . . . 2116231001220. . . . . .

Semester : . . . . . III. . . . . .

Academic Year : . . . . . 2024-2025 . . . . . .

RAJALAKSHMI ENGINEERING COLLEGE

[AUTONOMOUS]

RAJALAKSHMI NAGAR, THANDALAM – 602 105

BONAFIDE CERTIFICATE

Name : . .Sudharsan.D.. . . . . .

2116231001220

Academic Year : 2024-2025 Semester:. III Branch : IT-D

Register No.

Certified that this is the bonafide record of work done by the above student in the CS23333 –Object Oriented Programming using JAVA during the year 2024 - 2025.

Signature of Faculty in-charge

Submitted for the Practical Examination held on . . . . . .

Internal Examiner **External Examiner**

1.

Write a program to find whether the given input number is Odd.

If the given number is odd, the program should return 2 else it should return 1.

Note: The number passed to the program can either be negative. positive or zero. Zero should NOT be treated as Odd.

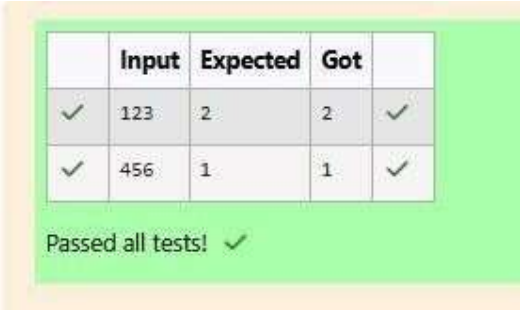
For example:

Input	Result
123	2
456	1

SOLUTION :

```
import java.util.Scanner; public
class oddorEven{
public static void
main(String[]args){ Scanner s=new
Scanner(System.in); int number =
s.nextInt(); if(number %2==0){
   System.out.println(1);
} else
{
   System.out.println(2);
}
}
}
```

OUTPUT :



	Input	Expected	Got	
✓	123	2	2	✓
✓	456	1	1	✓

Passed all tests! ✓

2.

Write a program that returns the last digit of the given number. Last digit is being referred to the least significant digit i.e. the digit in the ones (units) place in the given number.

The last digit should be returned as a positive number.

For example,

if the given number is 197, the last digit is 7

if the given number is -197, the last digit is 7

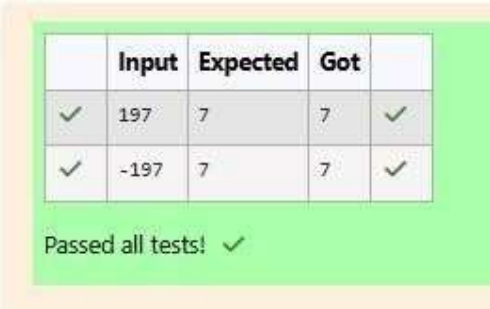
For example:

Input	Result
197	7
-197	7

SOLUTION :

```
import java.util.Scanner; import
java.lang.Math; public class LastDigit{
public static void main(String[]args){
Scanner s=new Scanner(System.in);
      int a = s.nextInt(); int
      lastDigit=Math.abs(a%10);
      System.out.println(lastDigit);
   }
   }
```

OUTPUT :



	Input	Expected	Got	
✓	197	7	7	✓
✓	-197	7	7	✓

Passed all tests! ✓

3.

Rohit wants to add the last digits of two given numbers.
For example,
If the given numbers are 267 and 154, the output should be 11.
Below is the explanation:
Last digit of the 267 is 7
Last digit of the 154 is 4
Sum of 7 and 4 = 11
Write a program to help Rohit achieve this for any given two numbers.
Note: Tile sign of the input numbers should be ignored.
i.e.
if the input numbers are 267 and 154, the sum of last two digits should be 11
if the input numbers are 267 and -154, the slim of last two digits should be 11
if the input numbers are -267 and 154, the sum of last two digits should be 11
if the input numbers are -267 and -154, the sum of last two digits should be 11

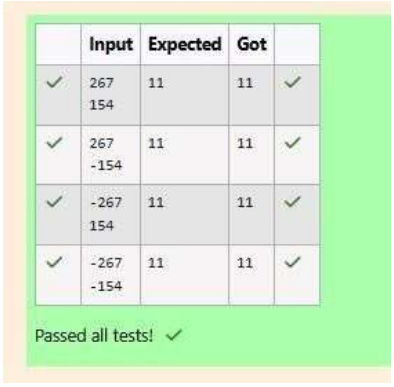
For example:

Input	Result
267 154	11
267 -154	11
-267 154	11
-267 -154	11

SOLUTION :

```
import java.util.Scanner; import
java.lang.Math;
 public class number{ public static void main(String[]args){
    Scanner s= new
    Scanner(System.in);
        int a = s.nextInt();
        int b = s.nextInt();
        System.out.println(Math.abs(a)%10+Math.abs(b)%10);
    }
 }
```

OUTPUT:



	Input	Expected	Got	
✓	267 154	11	11	✓
✓	267 -154	11	11	✓
✓	-267 154	11	11	✓
✓	-267 -154	11	11	✓

Passed all tests! ✓

Lab-02-Flow Control Statements

1.

Consider the following sequence:

1st term: 1

2nd term: 1 2 1

3rd term: 1 2 1 3 1 2 1

4th term: 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

And so on. Write a program that takes as parameter an integer n and prints the nth terms of this sequence.

Example Input:

1

Output:

1

Example Input:

4

Output:

1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

For example:

Input	Result
1	1
2	1 2 1
3	1 2 1 3 1 2 1
4	1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

SOLUTION :

```
import java.util.Scanner; public class
SequenceGenerator{ public static void
main(String[]args){ Scanner S = new
Scanner(System.in);
        int n =
        S.nextInt();
        String term = generateTerm(n);
        System.out.print(term);
    }
    private static String generateTerm(int n){ if
        (n==1){ return "1";
        }
        String prevTerm = generateTerm (n-1);
        StringBuilder currentTerm = new StringBuilder(prevTerm);
currentTerm.append("  "  +  n  +  "  ");
currentTerm.append(prevTerm);                return
currentTerm.toString();
}
```

```
}
```

OUTPUT :

	Input	Expected	Got	
✓	1	1	1	✓
✓	2	1 2 1	1 2 1	✓
✓	3	1 2 1 3 1 2 1	1 2 1 3 1 2 1	✓
✓	4	1 2 1 3 1 2 1 4 1 2 1 3 1 2 1	1 2 1 3 1 2 1 4 1 2 1 3 1 2 1	✓

Passed all tests! ✓

2.

Write a program that takes as parameter an integer n.
You have to print the number of zeros at the end of the factorial of n.
For example, 3! = 6. The number of zeros are 0. 5! = 120. The number of zeros at the end are 1.
Note: n! < 10^5
Example Input:
3
Output:
0
Example Input:
60
Output:
14
Example Input:
100
Output:
24
Example Input:
1024
Output:
253

For example:

Input	Result
3	0
60	14
100	24
1024	253

SOLUTION :

```
// Java program to count trailing 0s in n! import java.io.*; import java.util.Scanner; class
prog {
    // Function to return trailing
    // 0s in factorial of n
    static int findTrailingZeros(int n)
    { if (n < 0) // Negative Number Edge Case return -1;
```

```
        // Initialize result

        int count=0;
        // Keep dividing n by powers //
        of 5 and update count for (int i =
        5; n / i >= 1; i*=5          ){ count
        += n / i;
} return count;
    }

    // Driver Code
    public static void main(String[] args)
    {
        Scanner sc= new Scanner(System.in);
        int n=sc.nextInt();
        int
    res=findTrailingZeros(n);
    System.out.println(res); }
}
```

OUTPUT :

	Input	Expected	Got	
✓	3	0	0	✓
✓	60	14	14	✓
✓	100	24	24	✓
✓	1024	253	253	✓

Passed all tests! ✓

3.

Consider a sequence of the form 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149...

Write a method program which takes as parameter an integer n and prints the nth term of the above sequence. The nth term will fit in an integer value.

Example Input:

5

Output:

4

Example Input:

8

Output:

24

Example Input:

11

Output:

149

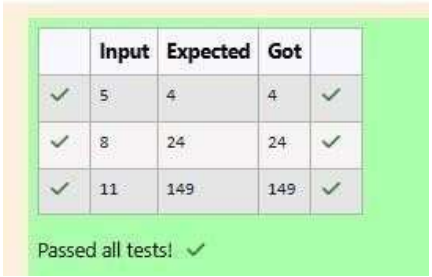
For example:

Input	Result
5	4
8	24
11	149

SOLUTION :

```
import java.util.Scanner;
class fibo3{ int a; int b;
int c; fibo3(int a,int b,int
c){ this.a = a; this.b = b;
this.c = c;
   }
   int nth(int x){
      if (x == 1){
      return 0;
      }
      else if(x == 2 && x == 3) return
         1;
      else{ int temp1,temp2,temp; int
         count = 4; while(x >=
         count){ temp =
         this.a+this.b+this.c; temp1 =
         this.c; this.c = temp; temp2 =
         this.b; this.b = temp1; this.a
         = temp2;
            count++;
         }
         return this.c;
      }
   }
}
public class Main{ public static void
   main(String[] args){ Scanner s = new
   Scanner(System.in); int t =
      s.nextInt(); fibo3 r =
      new fibo3(0,1,1);
      System.out.print(r.nth(t));
   }
}
```

OUTPUT :



	Input	Expected	Got	
✓	5	4	4	✓
✓	8	24	24	✓
✓	11	149	149	✓

Passed all tests! ✓

Lab-03-Arrays

1.

You are provided with a set of numbers (array of numbers).

You have to generate the sum of specific numbers based on its position in the array set provided to you.

This is explained below:

Example 1:

Let us assume the encoded set of numbers given to you is:

input1:5 and input2: {1, 51, 436, 7860, 41236}

Step 1:

Starting from the 0^{th} index of the array pick up digits as per below:

0^{th} index – pick up the units value of the number (in this case is 1).

1^{st} index - pick up the tens value of the number (in this case it is 5).

2^{nd} index - pick up the hundreds value of the number (in this case it is 4).

3^{rd} index - pick up the thousands value of the number (in this case it is 7).

4^{th} index - pick up the ten thousands value of the number (in this case it is 4).

(Continue this for all the elements of the input array).

The array generated from Step 1 will then be – {1, 5, 4, 7, 4}.

Step 2:

Square each number present in the array generated in Step 1.

{1, 25, 16, 49, 16}

Step 3:

Calculate the sum of all elements of the array generated in Step 2 to get the final result. The result will be = 107.

Note:

1) While picking up a number in Step1, if you observe that the number is smaller than the required position then use 0.

2) In the given function, input1[] is the array of numbers and input2 represents the number of elements in input1.

Example 2:

input1: 5 and input1: {1, 5, 423, 310, 61540}

Step 1:

Generating the new array based on position, we get the below array:

{1, 0, 4, 0, 6}

In this case, the value in input1 at index 1 and 3 is less than the value required to be picked up based on position, so we use a 0.

Step 2:

{1, 0, 16, 0, 36}

Step 3:

The final result = 53.

For example:

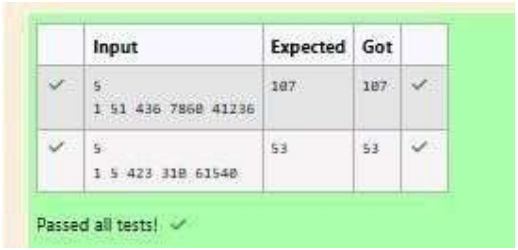
Input	Result
5 1 51 436 7860 41236	107
5 1 5 423 310 61540	53

SOLUTION :

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

```
    int size =scanner.nextInt();
    int[]inpar=new int[size]; for(int
    i=0;i<size;i++){
    inpar[i]=scanner.nextInt();
    } int[]dig=new int[size]; for(int
    i=0;i<size;i++){ int
    num=inpar[i]; if(i==0){
    dig[i]=num%10;
          }
          else if (i==1){ dig[i]=(num/10)%10;
          }
          else if(i==2){ dig[i]=(num/100)%10;
          }
          else if(i==3){
             dig[i]=(num/1000)%10;

          }
          else if(i==4){
             dig[i]=(num/10000)%10;
          } else{ dig[i]=0;
          } } int fin=0; for(int
    digi:dig){ fin+=digi*digi;
    }
    System.out.print(fin);
  }
}
```

OUTPUT :



	Input	Expected	Got	
✓	5 1 51 436 7860 41236	107	107	✓
✓	5 1 5 423 310 61540	53	53	✓

Passed all tests! ✓

2.

Given an array of numbers, you are expected to return the sum of the longest sequence of POSITIVE numbers in the array.

If there are NO positive numbers in the array, you are expected to return -1.

In this question's scope, the number 0 should be considered as positive.

Note: If there are more than one group of elements in the array having the longest sequence of POSITIVE numbers, you are expected to return the total sum of all those POSITIVE numbers (see example 3 below).

input1 represents the number of elements in the array.

input2 represents the array of integers.

Example 1:

input1 = 16

input2 = {-12, -16, 12, 18, 18, 14, -4, -12, -13, 32, 34, -5, 66, 78, 78, -79}

Expected output = 62

Explanation:

The input array contains four sequences of POSITIVE numbers, i.e. "12, 18, 18, 14", "12", "32, 34", and "66, 78, 78". The first sequence "12, 18, 18, 14" is the longest of the four as it contains 4 elements. Therefore, the expected output = sum of the longest sequence of POSITIVE numbers = 12 + 18 + 18 + 14 = 63.

Example 2:

input1 = 11

input2 = {-22, -24, 16, -1, -17, -19, -37, -25, -19, -93, -61}

Expected output = -1

Explanation:

There are NO positive numbers in the input array. Therefore, the expected output for such cases = -1.

Example 3:

input1 = 16

input2 = {-58, 32, 26, 92, -10, -4, 12, 0, 12, -2, 4, 32, -9, -7, 78, -79}

Expected output = 174

Explanation:

The input array contains four sequences of POSITIVE numbers, i.e. "32, 26, 92", "12, 0, 12", "4, 32", and "78". The first and second sequences "32, 26, 92" and "12, 0, 12" are the longest of the four as they contain 4 elements each. Therefore, the expected output = sum of the longest sequence of POSITIVE numbers = (32 + 26 + 92) + (12 + 0 + 12) = 174.

For example:

Input	Result
16 -12 -16 12 18 18 14 -4 -12 -13 32 34 -5 66 78 78 -79	62
11 -22 -24 -16 -1 -17 -19 -37 -25 -19 -93 -61	-1
16 -58 32 26 92 -10 -4 12 0 12 -2 4 32 -9 -7 78 -79	174

SOLUTION :

```
import java.util.Scanner; public class
longdig{ public static void
main(String[]args){ Scanner sc=new
Scanner(System.in);
        int n=sc.nextInt();
        int c = 1,v,seqtemp = 0,seq = 0,countmax = 0;
        int count = 0; while(c <= n){ v = sc.nextInt();
        if(v >= 0){ countmax= countmax + v;
            seqtemp++;
            }
            else{
                seqtemp = 0;
                countmax = 0;
            }
            if(seqtemp > seq ){
                seq = seqtemp;
                count = countmax;
            }
            else if (seq == seqtemp){
                count = count + countmax;
            }
        c++; }
        if (count == 0)
            System.out.print(-1);
        else
            System.out.print(count);
```

```
}
```

```
}
```

OUTPUT :

	Input	Expected	Got	
✓	16 -12 -16 12 18 18 14 -4 -12 -13 32 34 -5 66 78 78 -79	62	62	✓
✓	11 -22 -24 -16 -1 -17 -19 -37 -25 -19 -93 -61	-1	-1	✓
✓	16 -58 32 26 92 -10 -4 12 0 12 -2 4 32 -9 -7 78 -79	174	174	✓

Passed all tests! ✓

3.

Given an integer array as input, perform the following operations on the array, in the below specified sequence.

1. Find the maximum number in the array.
2. Subtract the maximum number from each element of the array.
3. Multiply the maximum number (found in step 1) to each element of the resultant array.

After the operations are done, return the resultant array.

Example 1:

input1 = 4 (represents the number of elements in the input1 array)

input2 = {1, 5, 6, 9}

Expected Output = {-72, -36, 27, 0}

Explanation:

Step 1: The maximum number in the given array is 9.

Step 2: Subtracting the maximum number 9 from each element of the array:

{(1 - 9), (5 - 9), (6 - 9), (9 - 9)} = {-8, -4, -3, 0}

Step 3: Multiplying the maximum number 9 to each of the resultant array:

{(-8 x 9), (-4 x 9), (3 x 9), (0 x 9)} = {-72, -36, -27, 0}

So, the expected output is the resultant array {-72, -36, -27, 0}.

Example 2:

input1 = 5 (represents the number of elements in the input1 array)

input2 = {10, 87, 63, 42, 2}

Expected Output = {-6699, 0, -2088, -3915, -7395}

Explanation:

Step 1: The maximum number in the given array is 87.

Step 2: Subtracting the maximum number 87 from each element of the array:

{(10 - 87), (87 - 87), (63 - 87), (42 - 87), (2 - 87)} = {-77, 0, -24, -45, -85}

Step 3: Multiplying the maximum number 87 to each of the resultant array:

{(-77 x 87), (0 x 87), (-24 x 87), (-45 x 87), (-85 x 87)} = {-6699, 0, -2088, -3915, -7395}

So, the expected output is the resultant array {-6699, 0, -2088, -3915, -7395}.

Example 3:

input1 = 2 (represents the number of elements in the input1 array)

input2 = {-9, 9}

Expected Output = {-162, 0}

Explanation:

Step 1: The maximum number in the given array is 9.

Step 2: Subtracting the maximum number 9 from each element of the array:

{(-9 - 9), (9 - 9)} = {-18, 0}

Step 3: Multiplying the maximum number 9 to each of the resultant array:

{(-18 x 9), (0 x 9)} = {-162, 0}

So, the expected output is the resultant array {-162, 0}.

Note: The input array will contain not more than 100 elements

For example:

Input	Result
4 1 5 6 9	-72 -36 -27 0
5 10 87 63 42 2	-6699 0 -2088 -3915 -7395

SOLUTION :

```
import java.util.Scanner; public
class res{ public static
int[]pa(int[]arr){
    int
    maxs=Integer.MIN_VALUE;
    for (int num:arr){
    if(num>maxs){
      maxs=num;
      }
    }
    for(int i=0;i<arr.length;i++){ arr[i]=(arr[i]-
      maxs)*maxs;
    }
    return arr;
  }
  public static void main(String[]args){
    Scanner scanner =new Scanner (System.in);
    int  n=scanner.nextInt();
    int[]arr=new int[n]; for(int
    i=0;i<n;i++){
    arr[i]=scanner.nextInt();
    } int[]res=pa(arr);
    for(int
    i=0;i<n;i++){
      System.out.print(res[i]+" ");
    }
    scanner.close();
  }
}
```

OUTPUT :

	Input	Expected	Got	
✓	4 1 5 6 9	-72 -36 -27 0	-72 -36 -27 0	✓
✓	5 10 87 63 42 2	-6699 0 -2088 -3915 -7395	-6699 0 -2088 -3915 -7395	✓
✓	2 -9 9	-162 0	-162 0	✓

Passed all tests! ✓

Lab-04-Classes and Objects

1.

Create a class called "Circle" with a radius attribute. You can access and modify this attribute using getter and setter methods. Calculate the area and circumference of the circle.

Area of Circle = πr^2

Circumference = $2\pi r$

Input:

2

Output:

Area = **12.57**
Circumference = **12.57**

For example:

Test	Input	Result
1	4	Area = 50.27 Circumference = 25.13

SOLUTION :

```
import java.io.*; import
java.util.Scanner; class
Circle
{ private double radius; public
   Circle(double radius){
      // set the instance variable radius
     this.radius =radius;
        } public void setRadius(double
   radius){
      // set the radius
     this.radius=radius;

   }
public double getRadius()       {
      // return the radius
     return radius;

   }
   public double calculateArea() { // complete the below statement
     return Math.PI*radius*radius;

   }
public double calculateCircumference()       {
      // complete the statement return
     2*Math.PI*radius;
   }
} class prog{ public static void
main(String[] args) { int r;
      Scanner sc= new Scanner(System.in);
      r=sc.nextInt();
      Circle c= new Circle(r);
      System.out.println("Area = "+String.format("%.2f",
      c.calculateArea()));
      // invoke the calculatecircumference method
      System.out.println("Circumference = "+String.format("%.2f" ,
c.calculateCircumference()));

      sc.close();
   }
}
```

OUTPUT :

	Test	Input	Expected	Got	
✓	1	4	Area = 50.27 Circumference = 25.13	Area = 50.27 Circumference = 25.13	✓
✓	2	6	Area = 113.10 Circumference = 37.70	Area = 113.10 Circumference = 37.70	✓
✓	3	2	Area = 12.57 Circumference = 12.57	Area = 12.57 Circumference = 12.57	✓

Passed all tests! ✓

2.

Create a Class Mobile with the attributes listed below,

private String manufacturer;
private String operating_system;
public String color;
private int cost;

Define a Parameterized constructor to initialize the above instance variables.

Define getter and setter methods for the attributes above.

for example : setter method for manufacturer is

void setManufacturer(String manufacturer){

this.manufacturer= manufacturer;

}

String getManufacturer(){

return manufacturer;}

Display the object details by overriding the toString() method.

For example:

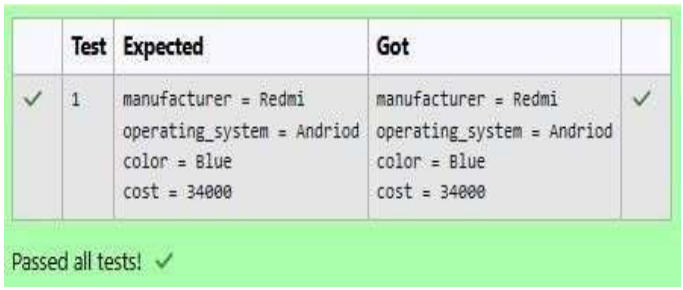
Test	Result
1	manufacturer = Redmi operating_system = Andriod color = Blue cost = 34000

SOLUTION :

```
public class mobile{
   private String man;
   private String os;
   public String clr;
   private int cost;
   public mobile(String man,String os,String clr,int cost){
      this.man=man; this.os=os; this.clr=clr; this.cost=cost;
      }
      public String toString(){ return "manufacturer = "+man+"\n"+"operating_system = "+os+"\n"+"color = "+ clr+"\n"+"cost = "+cost;
      }
      public static void main(String[]args){
```

```
        mobile mobile=new
      mobile("Redmi","Andriod","Blue",34000);
      System.out.println(mobile); }
}
```

OUTPUT :



	Test	Expected	Got	
✓	1	manufacturer = Redmi operating_system = Andriod color = Blue cost = 34000	manufacturer = Redmi operating_system = Andriod color = Blue cost = 34000	✓

Passed all tests! ✓

3.

Create a class Student with two private attributes, name and roll number. Create three objects by invoking different constructors available in the class Student.

Student()

Student(String name)

Student(String name, int rollno)

Input:

No input

Output:

No-arg constructor is invoked
1 arg constructor is invoked
2 arg constructor is invoked
Name =null , Roll no = 0
Name =Rajalakshmi , Roll no = 0
Name =Lakshmi , Roll no = 101

For example:

Test	Result
1	No-arg constructor is invoked 1 arg constructor is invoked 2 arg constructor is invoked Name =null , Roll no = 0 Name =Rajalakshmi , Roll no = 0 Name =Lakshmi , Roll no = 101

SOLUTION :

```
public class stud{ private String name; private int roll; public
  stud(){
     System.out.println("No-arg constructor is invoked"); name=null; roll=0;

  }
  public stud(String name){
     System.out.println("1 arg constructor is invoked"); this.name=name; roll=0;
```

```
}
public stud(String name,int roll){
   System.out.println("2 arg constructor is invoked"); this.name=name; this.roll=roll;

   }

public static void main (String[]args){ stud s1=new stud();
         stud s2=new stud("Rajalakshmi"); stud s3=new
         stud("Lakshmi",101);
         System.out.println("Name ="+s1.name+" , Roll no = "+s2.roll);
         System.out.println("Name ="+s2.name+" , Roll no = "+s2.roll);
         System.out.println("Name ="+s3.name+" , Roll no = "+s3.roll);
      }
   }
```

OUTPUT :

	Test	Expected	Got	
✓	1	No-arg constructor is invoked 1 arg constructor is invoked 2 arg constructor is invoked Name =null , Roll no = 0 Name =Rajalakshmi , Roll no = 0 Name =Lakshmi , Roll no = 101	No-arg constructor is invoked 1 arg constructor is invoked 2 arg constructor is invoked Name =null , Roll no = 0 Name =Rajalakshmi , Roll no = 0 Name =Lakshmi , Roll no = 101	✓

Passed all tests! ✓

Lab-05-Inheritance

1.

Create a class known as "BankAccount" with methods called deposit() and withdraw().

Create a subclass called SavingsAccount that overrides the withdraw() method to prevent withdrawals if the account balance falls below one hundred.

For example:

Result

```
Create a Bank Account object (A/c No. BA1234) with initial balance of $500:
Deposit $1000 into account BA1234:
New balance after depositing $1000: $1500.0
Withdraw $600 from account BA1234:
New balance after withdrawing $600: $900.0
Create a SavingsAccount object (A/c No. SA1000) with initial balance of $300:
Try to withdraw $250 from SA1000!
Minimum balance of $100 required!
Balance after trying to withdraw $250: $300.0
```

SOLUTION :

```
class BankAccount {
// Private field to store the account number private
String accountNumber;
```

```
// Private field to store the balance
```

```
    private double balance;

    // Constructor to initialize account number and balance public
    BankAccount(String accountNumber,double balance){
    this.accountNumber=accountNumber; this.balance=balance;
    }




    // Method to deposit an amount into the account public
    void deposit(double amount) {
        // Increase the balance by the deposit amount
     balance+=amount;
    }

    // Method to withdraw an amount from the account public
    void withdraw(double amount) {
        // Check if the balance is sufficient for the withdrawal if
        (balance >= amount) {
            // Decrease the balance by the withdrawal amount balance
            -= amount;
        } else {
            // Print a message if the balance is
        insufficient System.out.println("Insufficient
        balance"); }
    }

    // Method to get the current balance public
    double getBalance() { //
    Return the current balance return
        balance;
    }
    public String getAccountNumber(){ return
        accountNumber;
    }
}
class SavingsAccount extends BankAccount {
    // Constructor to initialize account number and balance
    public SavingsAccount(String accountNumber, double balance) {
        // Call the parent class constructor
        super(accountNumber,balance);
    }

    // Override the withdraw method from the parent class
    @Override
```

```
public void withdraw(double amount) {
    // Check if the withdrawal would cause the balance to drop below $100
```

```
      if (getBalance() - amount < 100) {
         // Print a message if the minimum balance requirement is not met
         System.out.println("Minimum balance of $100 required!");
      } else {
         // Call the parent class withdraw method
         super.withdraw(amount);
      }
   }
} public class Main {

   public static void main(String[] args) {
      // Print message to indicate creation of a BankAccount object
      System.out.println("Create a Bank Account object (A/c No. BA1234) with initial balance of $500:");
      // Create a BankAccount object (A/c No. "BA1234") with initial balance of $500
      BankAccount BA1234 = new BankAccount("BA1234", 500);
      // Print message to indicate deposit action
      System.out.println("Deposit $1000 into account BA1234:");
      // Deposit $1000 into account BA1234
     BA1234.deposit(1000);
      // Print the new balance after deposit
       System.out.println("New balance after depositing $1000: $"+BA1234.getBalance());

      // Print message to indicate withdrawal action
      System.out.println("Withdraw $600 from account BA1234:");
      // Withdraw $600 from account BA1234
    BA1234.withdraw(600);
      // Print the new balance after withdrawal
      System.out.println("New balance after withdrawing $600: $" +
BA1234.getBalance());

      // Print message to indicate creation of another SavingsAccount object
      System.out.println("Create a SavingsAccount object (A/c No. SA1000) with initial balance of $300:");
      // Create a SavingsAccount object (A/c No. "SA1000") with initial balance of $300
      SavingsAccount SA1000 = new SavingsAccount("SA1000", 300);

      // Print message to indicate withdrawal action
      System.out.println("Try to withdraw $250 from SA1000!");
      // Withdraw $250 from SA1000 (balance falls below $100)
      SA1000.withdraw(250);
      // Print the balance after attempting to withdraw $250
      System.out.println("Balance after trying to withdraw $250: $" +
SA1000.getBalance()); } }
```

OUTPUT :

	Expected	Got	
✓	Create a Bank Account object (A/c No. BA1234) with initial balance of $500: Deposit $1000 into account BA1234: New balance after depositing $1000: $1500.0 Withdraw $600 from account BA1234: New balance after withdrawing $600: $900.0 Create a SavingsAccount object (A/c No. SA1000) with initial balance of $300: Try to withdraw $250 from SA1000! Minimum balance of $100 required! Balance after trying to withdraw $250: $300.0	Create a Bank Account object (A/c No. BA1234) with initial balance of $500: Deposit $1000 into account BA1234: New balance after depositing $1000: $1500.0 Withdraw $600 from account BA1234: New balance after withdrawing $600: $900.0 Create a SavingsAccount object (A/c No. SA1000) with initial balance of $300: Try to withdraw $250 from SA1000! Minimum balance of $100 required! Balance after trying to withdraw $250: $300.0	✓

Passed all tests! ✓

2.

create a class called College with attribute String name, constructor to initialize the name attribute , a method called Admitted(). Create a subclass called CSE that extends Student class, with department attribute , Course() method to sub class. Print the details of the Student.

College:

String collegeName;

public College() { }

public admitted() { }

Student:

String studentName;

String department;

public Student(String collegeName, String studentName,String depart) { }

public toString()

Expected Output:

A student admitted in REC
CollegeName : REC
StudentName : Venkatesh
Department : CSE

For example:

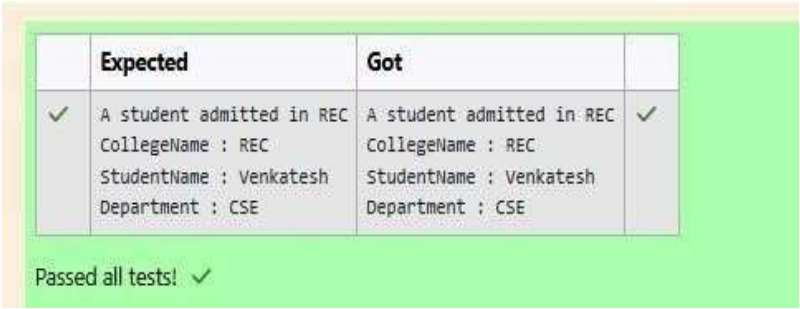
Result
A student admitted in REC CollegeName : REC StudentName : Venkatesh Department : CSE

SOLUTION :

```
class College
{ public  String collegeName;

public College(String collegeName) { // initialize
   the                instance                variables
   this.collegeName=collegeName; }

public void admitted() {
   System.out.println("A student admitted in "+collegeName);
} } class Student extends College{

String studentName;
String department;

public Student(String collegeName, String studentName,String department) {
  // initialize the instance variables super(collegeName);
  this.studentName=studentName; this.department=department;
```

```
}

public String toString(){
   // return the details of the student return "CollegeName :
"+collegeName+"\n"+"StudentName :
"+studentName+"\n"+"Department : "+department;
}
}
public class Main {
public static void main (String[] args) {
      Student s1 = new Student("REC","Venkatesh","CSE");
      s1.admitted();                        // invoke the admitted() method
      System.out.println(s1.toString());
}
}
```

OUTPUT :



	Expected	Got	
✓	A student admitted in REC CollegeName : REC StudentName : Venkatesh Department : CSE	A student admitted in REC CollegeName : REC StudentName : Venkatesh Department : CSE	✓

Passed all tests! ✓

3.

Create a class Mobile with constructor and a method basicMobile().

Create a subclass CameraMobile which extends Mobile class , with constructor and a method newFeature().

Create a subclass AndroidMobile which extends CameraMobile, with constructor and a method androidMobile().

display the details of the Android Mobile class by creating the instance. .

```
class Mobile{

}
class CameraMobile  extends Mobile {
}
class AndroidMobile extends CameraMobile {
}
```

expected output:

```
Basic Mobile is Manufactured
Camera Mobile is Manufactured
Android Mobile is Manufactured
Camera Mobile with 5MG px
Touch Screen Mobile is Manufactured
```

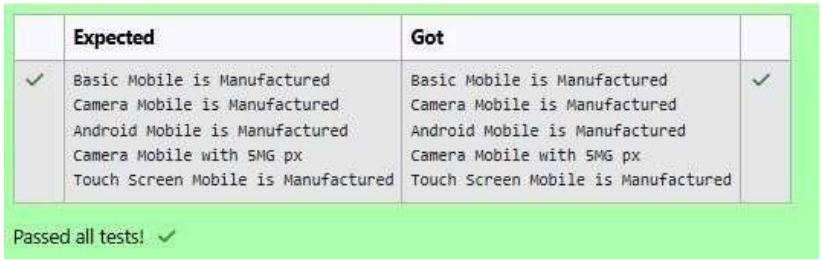
For example:

Result
Basic Mobile is Manufactured Camera Mobile is Manufactured Android Mobile is Manufactured Camera Mobile with 5MG px Touch Screen Mobile is Manufactured

SOLUTION :

```
class mob{
   mob(){
      System.out.println("Basic Mobile is Manufactured");
   }
   void basmob(){
      System.out.println("Basic Mobile is Manufactured");
   }
}
class cam extends mob{ cam(){
   super();
      System.out.println("Camera Mobile is Manufactured");
   }
   void newm(){
      System.out.println("Camera Mobile with 5MG px");

   }
}
class and extends cam{ and(){ super();
   System.out.println("Android Mobile is Manufactured");
   }
   void andmob(){
      System.out.println("Touch Screen Mobile is Manufactured");
   }
   } public class Main{ public static void main(String[]args){
and andmob=new and(); andmob.newm(); andmob.andmob();
   }

}
```

OUTPUT :



	Expected	Got	
✓	Basic Mobile is Manufactured Camera Mobile is Manufactured Android Mobile is Manufactured Camera Mobile with 5MG px Touch Screen Mobile is Manufactured	Basic Mobile is Manufactured Camera Mobile is Manufactured Android Mobile is Manufactured Camera Mobile with 5MG px Touch Screen Mobile is Manufactured	✓

Passed all tests! ✓

Lab-06-String, StringBuffer

1.

You are provided a string of words and a 2-digit number. The two digits of the number represent the two words that are to be processed.

For example:

If the string is "Today is a Nice Day" and the 2-digit number is 41, then you are expected to process the 4th word ("Nice") and the 1st word ("Today").

The processing of each word is to be done as follows:

Extract the Middle-to-Begin part: Starting from the middle of the word, extract the characters till the beginning of the word.

Extract the Middle-to-End part: Starting from the middle of the word, extract the characters till the end of the word.

If the word to be processed is "Nice":

Its Middle-to-Begin part will be "iN".

Its Middle-to-End part will be "ce".

So, merged together these two parts would form "iNce".

Similarly, if the word to be processed is "Today":

Its Middle-to-Begin part will be "doT".

Its Middle-to-End part will be "day".

So, merged together these two parts would form "doTday".

Note: Note that the middle letter 'd' is part of both the extracted parts. So, for words whose length is odd, the middle letter should be included in both the extracted parts.

Expected output:

The expected output is a string containing both the processed words separated by a space "iNce doTday"

Example 1:

input1 = "Today is a Nice Day"

input2 = 41

output = "iNce doTday"

Example 2:

input1 = "Fruits like Mango and Apple are common but Grapes are rare"

input2 = 39

output = "naMngo arGpes"

Note: The input string input1 will contain only alphabets and a single space character separating each word in the string.

Note: The input string input1 will NOT contain any other special characters.

Note: The input number input2 will always be a 2-digit number (>=11 and <=99). One of its digits will never be 0. Both the digits of the number will always point to a valid word in the input1 string.

For example:

Input	Result
Today is a Nice Day 41	iNce doTday
Fruits like Mango and Apple are common but Grapes are rare 39	naMngo arGpes

SOLUTION :

```
import java.util.*; public class mix{ public static void
main(String[] args){
      Scanner scan = new Scanner(System.in); String g =
      scan.nextLine(); int n = scan.nextInt(),ones,flag = 0;
      StringBuffer temp = new StringBuffer(); StringBuffer
      temp1 = new StringBuffer(); int space =
      0; while (n > 0){ ones = (n %10) - 1; for(int i = 0; i
          < g.length();i++){ if (g.charAt(i) == ' '){ space
            = space + 1;
            }
            else if(space == ones && flag == 0){
                temp.append(Character.toString(g.charAt(i)));
            }
            else if(space == ones && flag == 1){
            temp1.append(Character.toString(g.charAt(i))); }
```

```
            } space =
            0 ; flag =
            1; n = n
            /10;
        }
        rew m = new rew();
        System.out.println(m.r(temp1.toString()) + " " + m.r(temp.toString()));
    }
} class
rew{
    String r(String a){ int le
        = a.length(),n,q;
        StringBuffer temp3 = new StringBuffer();
        if(le % 2 == 1){ n = ((int)(le/2)); q =
        ((int)(le/2));
        } else{ n =
        ((int)(le/2)) - 1; q
            = ((int)(le/2));
        }
        for(int i = n;i >= 0;i--){
        temp3.append(Character.toString(a.charAt(i)));
            } for(int i = q;i < le;i++){
        temp3.append(Character.toString(a.charAt(i)));
        }
        return temp3.toString();
    }
}
```

OUTPUT :

	Input	Expected	Got	
✓	Today is a Nice Day 41	iNce doTday	iNce doTday	✓
✓	Fruits like Mango and Apple are common but Grapes are rare 39	naMngo arGpes	naMngo arGpes	✓

Passed all tests! ✓

2.

Given a String input1, which contains many number of words separated by : and each word contains exactly two lower case alphabets, generate an output based upon the below 2 cases.

Note:

1. All the characters in input 1 are lowercase alphabets.
2. input 1 will always contain more than one word separated by :
3. Output should be returned in uppercase.

Case 1:

Check whether the two alphabets are same.

If yes, then take one alphabet from it and add it to the output.

Example 1:

input1 = ww:ii:pp:rr:oo

output = WIPRO

Explanation:

word1 is ww, both are same hence take w

word2 is ii, both are same hence take i

word3 is pp, both are same hence take p

word4 is rr, both are same hence take r

word5 is oo, both are same hence take o

Hence the output is WIPRO

Case 2:

If the two alphabets are not same, then find the position value of them and find maximum value – minimum value.

Take the alphabet which comes at this (maximum value - minimum value) position in the alphabet series.

Example 2"

input1 = zx:za:ee

output = BYE

Explanation

word1 is zx, both are not same alphabets

position value of z is 26

position value of x is 24

max – min will be 26 – 24 = 2

Alphabet which comes in 2nd position is b

Word2 is za, both are not same alphabets

position value of z is 26

position value of a is 1

max – min will be 26 – 1 = 25

Alphabet which comes in 25th position is y

word3 is ee, both are same hence take e

Hence the output is BYE

For example:

Input	Result
ww:ii:pp:rr:oo	WIPRO
zx:za:ee	BYE

SOLUTION :

```
import java.util.*; class diff{ char different(char a, char b){ if ((int)a !=
(int)b) return
(char)((int)'a' + ((int)a-(int)b) - 1); return a;
        }
}
public class Main{ public static void main(String[] args){ Scanner scan =
    new
    Scanner(System.in); diff z = new diff();
        String q = scan.nextLine();
        StringBuffer ans = new StringBuffer();
        StringBuffer temp = new
        StringBuffer(); for(int i = 0;i <
        q.length();i++){ if(q.charAt(i) == ':'){
        temp.append(" ");
            } else{
            temp.append(Character.toString(q.charAt(i))); }
```

```
        }
        String h = temp.toString(); for(int i
        = 0;i < temp.length();i++){ if(i%3
        ==                                                                    0){
                ans.append(Character.toString(z.different(h.charAt(i),h.charAt(i+1))));
            }
        }
        System.out.print(ans.toString().toUpperCase());
    }
}
```

OUTPUT :

	Input	Expected	Got	
✓	ww:ii:pp:rr:oo	WIPRO	WIPRO	✓
✓	zx:za:oo	BYE	BYE	✓

Passed all tests! ✓

3.

Given 2 strings input1 & input2.

- Concatenate both the strings.
- Remove duplicate alphabets & white spaces.
- Arrange the alphabets in descending order.

Assumption 1:

There will either be alphabets, white spaces or null in both the inputs.

Assumption 2:

Both inputs will be in lower case.

Example 1:

Input 1: apple

Input 2: orange

Output: rponlgea

Example 2:

Input 1: fruits

Input 2: are good

Output: utsroigfeda

Example 3:

Input 1: ""

Input 2: ""

Output: null

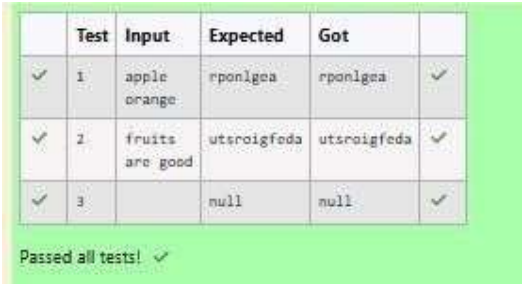
For example:

Test	Input	Result
1	apple orange	rponlgea
2	fruits are good	utsroigfeda

SOLUTION :

```
import java.util.*;


public class HelloWorld { public static void
   main(String[] args) {
      Scanner scan = new Scanner(System.in);
      String a = scan.nextLine();
      String b = scan.nextLine();
      StringBuffer ab = new StringBuffer();
      if(a.trim().isEmpty() && b.trim().isEmpty()){
      System.out.print("null");
      }
      else{
      for(int i = 0;i < a.length();i++){ if (a.charAt(i)
         != ' ') {
         ab.append(Character.toString(a.charAt(i))); }
      }
      for(int i = 0;i < b.length();i++){ if (b.charAt(i)
         != ' '){
         ab.append(Character.toString(b.charAt(i))); }
      }
      char[] d = ab.toString().toCharArray();
      Arrays.sort(d);
      for(int i = d.length - 1;i >= 1;i--){ if(d[i]
         != d[i-1])
         System.out.print(d[i]);
      }
      System.out.print(d[0]);
      }

   }
}
```

OUTPUT :



	Test	Input	Expected	Got	
✓	1	apple orange	rponlgea	rponlgea	✓
✓	2	fruits are good	utsroigfeda	utsroigfeda	✓
✓	3		null	null	✓

Passed all tests! ✓

Lab-07-Interfaces

1.

RBI issues all national banks to collect interest on all customer loans.

Create an RBI interface with a variable String parentBank="RBI" and abstract method rateOfinterest().

RBI interface has two more methods default and static method.

default void policyNote() {

System.out.println("RBI has a new Policy issued in 2023.");

}

static void regulations(){

System.out.println("RBI has updated new regulations on 2024.");

}

Create two subclasses SBI and Karur which implements the RBI interface.

Provide the necessary code for the abstract method in two sub-classes.

Sample Input/Output:

RBI has a new Policy issued in 2023
RBI has updated new regulations in 2024.
SBI rate of interest: 7.6 per annum.
Karur rate of interest: 7.4 per annum.

For example:

Test	Result
1	RBI has a new Policy issued in 2023 RBI has updated new regulations in 2024. SBI rate of interest: 7.6 per annum. Karur rate of interest: 7.4 per annum.

SOLUTION :

```
// Define the RBI interface interface RBI {
    // Variable declaration
    String parentBank = "RBI";

    // Abstract method double rateOfInterest();

    // Default method
    default void policyNote() {
        System.out.println("RBI has a new Policy issued in 2023"); }

    // Static method
    static void regulations() {
        System.out.println("RBI has updated new regulations in 2024.");
    }
}

// SBI class implementing RBI interface class SBI implements RBI
{
    // Implementing the abstract method public double
    rateOfInterest() {
```

```
        return 7.6;
    }
}

// Karur class implementing RBI
interface class Karur implements RBI { //
Implementing the abstract method public
double rateOfInterest() { return 7.4;
    }
}

// Main class to test the functionality
public class Main { public static void
main(String[] args) {
        // RBI policies and regulations
        RBI rbi = new SBI(); // Can be any class implementing RBI
        rbi.policyNote(); // Default method RBI.regulations(); //
        Static method

        // SBI bank details
        SBI sbi = new SBI();
        System.out.println("SBI rate of interest: " + sbi.rateOfInterest() + " per annum.");

        // Karur bank details
        Karur karur = new Karur();
        System.out.println("Karur rate of interest: " + karur.rateOfInterest() + " per annum.");
    }
}
```

OUTPUT :

	Test	Expected	Got	
✓	1	RBI has a new Policy issued in 2023 RBI has updated new regulations in 2024. SBI rate of interest: 7.6 per annum. Karur rate of interest: 7.4 per annum.	RBI has a new Policy issued in 2023 RBI has updated new regulations in 2024. SBI rate of interest: 7.6 per annum. Karur rate of interest: 7.4 per annum.	✓

Passed all tests! ✓

2.

Create interfaces shown below.

```
interface Sports {
public void setHomeTeam(String name);
public void setVisitingTeam(String name);
}
interface Football extends Sports {
public void homeTeamScored(int points);
public void visitingTeamScored(int points);}
```

create a class College that implements the Football interface and provides the necessary functionality to the abstract methods.

sample Input:

```
Rajalakshmi
Saveetha
22
21
```

Output:

```
Rajalakshmi 22 scored
Saveetha 21 scored
Rajalakshmi is the Winner!
```

For example:

Test	Input	Result
1	Rajalakshmi Saveetha 22 21	Rajalakshmi 22 scored Saveetha 21 scored Rajalakshmi is the winner!

SOLUTION :

```
import java.util.Scanner;

interface Sports { void setHomeTeam(String
    name); void
    setVisitingTeam(String name);
}

interface Football extends Sports { void
    homeTeamScored(int points); void
    visitingTeamScored(int points);
}

class College implements Football {
    private String homeTeam; private
    String visitingTeam; private int
    homeTeamPoints = 0; private int
    visitingTeamPoints = 0;

    public void setHomeTeam(String name) {
        this.homeTeam = name;
    }

    public void setVisitingTeam(String name) {
        this.visitingTeam = name;
```

```
} public void homeTeamScored(int points)
{
```

```
        homeTeamPoints += points;
        System.out.println(homeTeam + " " + points + " scored");
    }

    public void visitingTeamScored(int points) {
        visitingTeamPoints += points;
        System.out.println(visitingTeam + " " + points + " scored");
    }

    public void winningTeam() { if
        (homeTeamPoints > visitingTeamPoints) {
            System.out.println(homeTeam + " is the winner!");
        } else if (homeTeamPoints < visitingTeamPoints) {
            System.out.println(visitingTeam + " is the winner!");
        } else {
            System.out.println("It's a tie match.");
        }
    }
}

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

        // Get home team name
        String hname = sc.nextLine();

        // Get visiting team name
        String vteam = sc.nextLine();

        // Create College object College
        match = new College();
        match.setHomeTeam(hname);
        match.setVisitingTeam(vteam);

        // Get points scored by home team int
        htpoints = sc.nextInt();
        match.homeTeamScored(htpoints);

        // Get points scored by visiting team
        int vtpoints = sc.nextInt();
        match.visitingTeamScored(vtpoints);

        // Determine and print the winning team
        match.winningTeam();

        sc.close();
```

```
}
```

```
}
```

OUTPUT :

	Test	Input	Expected	Got	
✓	1	Rajalakshmi Saveetha 22 21	Rajalakshmi 22 scored Saveetha 21 scored Rajalakshmi is the winner!	Rajalakshmi 22 scored Saveetha 21 scored Rajalakshmi is the winner!	✓
✓	2	Anna Balaji 21 21	Anna 21 scored Balaji 21 scored It's a tie match.	Anna 21 scored Balaji 21 scored It's a tie match.	✓
✓	3	SRM VIT 20 21	SRM 20 scored VIT 21 scored VIT is the winner!	SRM 20 scored VIT 21 scored VIT is the winner!	✓

Passed all tests! ✓

3.

create an interface Playable with a method play() that takes no arguments and returns void. Create three classes Football, Volleyball, and Basketball that implement the Playable interface and override the play() method to play the respective sports.

```
interface Playable {
    void play();
}
class Football implements Playable {
    String name;
    public Football(String name){
        this.name=name;
    }
  public void play() {
    System.out.println(name+" is Playing football");
  }
}
```

Similarly, create Volleyball and Basketball classes.

Sample output:

```
Sadhvin is Playing football
Sanjay is Playing volleyball
Sruthi is Playing basketball
```

For example:

Test	Input	Result
1	Sadhvin Sanjay Sruthi	Sadhvin is Playing football Sanjay is Playing volleyball Sruthi is Playing basketball
2	Vijay Arun Balaji	Vijay is Playing football Arun is Playing volleyball Balaji is Playing basketball

SOLUTION :

```
import java.util.Scanner;

// Define the Playable interface interface
Playable {
    // Abstract method to play the respective sport
    void play();
}

// Football class implementing Playable interface
class Football implements Playable { String
name;

    // Constructor
    public Football(String name) { this.name
        = name;
    }

    // Override the play method
```

```
    public void play() {
        System.out.println(name + " is Playing football");
    }
}

// Volleyball class implementing Playable interface
class Volleyball implements Playable {
    String name;

    // Constructor
    public Volleyball(String name) { this.name
        = name;
    }

    // Override the play method
    public void play() {
        System.out.println(name + " is Playing volleyball");
    }
}

// Basketball class implementing Playable interface
class Basketball implements Playable {
    String name;

    // Constructor
    public Basketball(String name) { this.name
        = name;
    }

    // Override the play method
    public void play() {
        System.out.println(name + " is Playing basketball");
    }
}

// Main class to test the functionality
public class Main { public static void
main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input for Football player

        String footballPlayerName = scanner.nextLine();
        Football footballPlayer = new
        Football(footballPlayerName);

        // Input for Volleyball player
```

```
String volleyballPlayerName = scanner.nextLine();
Volleyball volleyballPlayer = new Volleyball(volleyballPlayerName);
```

```
        // Input for Basketball player

        String basketballPlayerName = scanner.nextLine();
        Basketball basketballPlayer = new Basketball(basketballPlayerName);

        // Call the play method for each player
        footballPlayer.play();
        volleyballPlayer.play();
        basketballPlayer.play();

        scanner.close();
    }
}
```

OUTPUT :

	Test	Input	Expected	Got	
✓	1	Sadhvin Sanjay Sruthi	Sadhvin is Playing football Sanjay is Playing volleyball Sruthi is Playing basketball	Sadhvin is Playing football Sanjay is Playing volleyball Sruthi is Playing basketball	✓
✓	2	Vijay Arun Balaji	Vijay is Playing football Arun is Playing volleyball Balaji is Playing basketball	Vijay is Playing football Arun is Playing volleyball Balaji is Playing basketball	✓

Passed all tests! ✓

Lab-08 - Polymorphism, Abstract Classes, final Keyword

1.

As a logic building learner you are given the task to extract the string which has vowel as the first and last characters from the given array of Strings.
Step1: Scan through the array of Strings, extract the Strings with first and last characters as vowels: these strings should be concatenated.
Step2: Convert the concatenated string to lowercase and return it.
If none of the strings in the array has first and last character as vowel, then return no matches found
input1: an integer representing the number of elements in the array.
input2: String array.
Example 1:
input1: 3
input2: {"oreo", "sirish", "apple"}
output: oreoapple
Example 2:
input1: 2
input2: {"Mango", "banana"}
output: no matches found
Explanation:
None of the strings has first and last character as vowel.

Hence the output is no matches found.
Example 3:
input1: 3
input2: {"Ate", "Ace", "Girl"}
output: ateace

For example:

Input	Result
3 oreo sirish apple	oreoapple
2 Mango banana	no matches found
3 Ate Ace Girl	ateace

SOLUTION :

```
import java.util.Scanner; public class

VowelStringExtractor {

   // Method to extract strings with vowels as first and last characters public
   static String extractVowelStrings(String[] stringArray) {
      StringBuilder result = new StringBuilder();
      String vowels = "aeiouAEIOU"; // String containing all vowels

      // Iterate through the array of strings for
      (String s : stringArray) {
         // Check if the string is not empty and if both the first and last characters are vowels
if (s.length() > 0 && vowels.indexOf(s.charAt(0)) != -1 &&
vowels.indexOf(s.charAt(s.length() - 1)) != -1) { result.append(s); // Append matching
string to the result }
      }

      // Return the concatenated string in lowercase or "no matches found"
      return result.length() > 0 ? result.toString().toLowerCase() : "no matches found"; }
```

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

        // Input for the number of strings

        int n = scanner.nextInt(); scanner.nextLine(); // Consume the newline character

        // Input for the strings in one line

        String input = scanner.nextLine();
        String[] strings = input.split(" "); // Split input into an array

        // Process and output the result
        String result = extractVowelStrings(strings); System.out.println(result);

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

OUTPUT :

	Input	Expected	Got	
✓	3 oreo sirish apple	oreoapple	oreoapple	✓
✓	2 Mango banana	no matches found	no matches found	✓
✓	3 Ate Ace Girl	ateace	ateace	✓

Passed all tests! ✓

2.

1. Final Variable:

- Once a variable is declared `final`, its value cannot be changed after it is initialized.
- It must be initialized when it is declared or in the constructor if it's not initialized at declaration.
- It can be used to define constants

```
final int MAX_SPEED = 120;  // Constant value, cannot be changed
```

2. Final Method:

- A method declared `final` cannot be overridden by subclasses.
- It is used to prevent modification of the method's behavior in derived classes.

```
public final void display() {
   System.out.println("This is a final method.");
}
```

3. Final Class:

- A class declared as `final` cannot be subclassed (i.e., no other class can inherit from it).
- It is used to prevent a class from being extended and modified.
- public final class Vehicle {
  // class code
  }

Given a Java Program that contains the bug in it, your task is to clear the bug to the output.

you should delete any piece of code.

For example:

Test	Result
1	The maximum speed is: 120 km/h This is a subclass of FinalExample.

SOLUTION :

```
// Final class definition final
class FinalExample {
    // Final variable
    final int MAX_SPEED = 120; // Constant value

    // Final method
    public final void display() {
        System.out.println("The maximum speed is: " + MAX_SPEED + " km/h");
    }
}

// Main class to test the final class public
class Test { public static void
main(String[] args) {
        // Create an instance of FinalExample
        FinalExample example = new FinalExample();
        example.display();

        // Uncommenting the following line will result in a compile-time error
        // because FinalExample is a final class and cannot be subclassed. //
        class SubclassExample extends FinalExample { }

        System.out.println("This is a subclass of FinalExample.");
    }
}
```

OUTPUT :

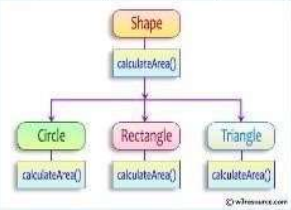
	Test	Expected	Got	
✓	1	The maximum speed is: 120 km/h This is a subclass of FinalExample.	The maximum speed is: 120 km/h This is a subclass of FinalExample.	✓

Passed all tests! ✓

3.

Create a base class Shape with a method called calculateArea(). Create three subclasses: Circle, Rectangle, and Triangle. Override the calculateArea() method in each subclass to calculate and return the shape's area.

In the given exercise, here is a simple diagram illustrating polymorphism implementation:




```
abstract class Shape {
   public abstract double calculateArea() ;
   }
}
```

System.out.printf("Area of a Triangle :%.2f%n",((0.5)*base*height)); // use this statement

sample Input :

4 // radius of the circle to calculate area PI*r*r

5 // length of the rectangle

6 // breadth of the rectangle to calculate the area of a rectangle

4 // base of the triangle

3 // height of the triangle

OUTPUT:

Area of a circle :50.27
Area of a Rectangle :30.00
Area of a Triangle :6.00

For example:

Test	Input	Result
1	4 5 6 4 3	Area of a circle: 50.27 Area of a Rectangle: 30.00 Area of a Triangle: 6.00
2	7 4.5 6.5 2.4 3.6	Area of a circle: 153.94 Area of a Rectangle: 29.25 Area of a Triangle: 4.32

SOLUTION :

```
import java.util.Scanner;

// Abstract class Shape abstract class Shape { public
abstract double calculateArea();
}

// Circle class class Circle extends Shape {
private double radius;

public Circle(double radius) {
this.radius = radius;
}

@Override
```

```
    public double calculateArea() { return Math.PI * radius * radius; // Area of
    circle: πr² }
}

// Rectangle class
class Rectangle extends Shape { private double
    length; private double breadth;

    public Rectangle(double length, double breadth) { this.length = length;
        this.breadth = breadth;
    }

    @Override
    public double calculateArea() { return length * breadth; // Area of rectangle: length
        * breadth
    }
}

// Triangle class
class Triangle extends Shape { private double
    base; private double height;

    public Triangle(double base, double height) { this.base = base; this.height
        = height;
    }

    @Override
    public double calculateArea() { return 0.5 * base * height; // Area of triangle: 0.5 *
        base * height
    }
}

// Main class to test the shapes public class ShapeTest {
public static void
main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input for Circle

        double radius = scanner.nextDouble();
        Circle circle = new Circle(radius);
        System.out.printf("Area of a circle: %.2f%n", circle.calculateArea());

        // Input for Rectangle
```

```
        double length = scanner.nextDouble();

        double breadth = scanner.nextDouble();
        Rectangle rectangle = new Rectangle(length, breadth);
        System.out.printf("Area of a Rectangle: %.2f%n", rectangle.calculateArea());

        // Input for Triangle double base =

        scanner.nextDouble();

        double height = scanner.nextDouble();
        Triangle triangle = new Triangle(base, height);
        System.out.printf("Area of a Triangle: %.2f%n", triangle.calculateArea());

        scanner.close();
    }
}
```

OUTPUT :

	Test	Input	Expected	Got	
✓	1	4 5 6 4 3	Area of a circle: 50.27 Area of a Rectangle: 30.00 Area of a Triangle: 6.00	Area of a circle: 50.27 Area of a Rectangle: 30.00 Area of a Triangle: 6.00	✓
✓	2	7 4.5 6.5 2.4 3.6	Area of a circle: 153.94 Area of a Rectangle: 29.25 Area of a Triangle: 4.32	Area of a circle: 153.94 Area of a Rectangle: 29.25 Area of a Triangle: 4.32	✓

Passed all tests! ✓

Lab-09-Exception Handling

1.

Write a Java program to create a method that takes an integer as a parameter and throws an exception if the number is odd.

Sample input and Output:

```
82 is even.
Error: 37 is odd.
```

Fill the preloaded answer to get the expected output.

For example:

Result
82 is even. Error: 37 is odd.

SOLUTION :

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

```
        int n = 82;
        trynumber(n);
        n = 37;
        trynumber(n); // Call the trynumber(n);
    }

    public static void trynumber(int n) { try { checkEvenNumber(n);
        // Call the checkEvenNumber()
        System.out.println(n + " is even.");
        } catch (Exception e) { // Catch the exception System.out.println("Error:
            " + e.getMessage());
        }
    }

    public static void checkEvenNumber(int number) { if (number % 2 != 0) { throw new
        RuntimeException(number + " is odd."); // Throw a RuntimeException }
    }
}
```

OUTPUT :

	Expected	Got	
✓	82 is even. Error: 37 is odd.	82 is even. Error: 37 is odd.	✓

Passed all tests! ✓

2.

In the following program, an array of integer data is to be initialized.
During the initialization, if a user enters a value other than an integer, it will throw an InputMismatchException exception.
On the occurrence of such an exception, your program should print "You entered bad data."
If there is no such exception it will print the total sum of the array.

/* Define try-catch block to save user input in the array "name"
If there is an exception then catch the exception otherwise print the total sum of the array. */

Sample Input:

3
5 2 1

Sample Output:

8

Sample Input:

2
1 g

Sample Output:

You entered bad data.

For example:

Input	Result
3 5 2 1	8
2 1 g	You entered bad data.

SOLUTION :

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

class prog { public static void
   main(String[] args) { Scanner sc = new
   Scanner(System.in); int length =
   sc.nextInt();
      // create an array to save user input int[]
      name = new int[length]; int sum = 0; // save
      the total sum of the array.

      /* Define try-catch block to save user input in the array "name" If
        there is an exception then catch the exception otherwise print
        the total sum of the array. */
      try { for (int i = 0; i < length; i++) { name[i] = sc.nextInt();
         // save user input in the array
         }

         // Calculate the total sum
         for (int num : name) {
            sum += num;
         }

         // Print the total sum
         System.out.println(sum);
      } catch (InputMismatchException e) {
         System.out.println("You entered bad data.");
      }

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

OUTPUT :

	Input	Expected	Got	
✓	3 5 2 1	8	8	✓
✓	2 1 g	You entered bad data.	You entered bad data.	✓

Passed all tests! ✓

3.

Write a Java program to handle ArithmeticException and ArrayIndexOutOfBoundsException.

Create an array, read the input from the user, and store it in the array.

Divide the 0th index element by the 1st index element and store it.

if the 1st element is zero, it will throw an exception.

if you try to access an element beyond the array limit throws an exception.

Input:

5

10 0 20 30 40

Output:

java.lang.ArithmeticException: / by zero
I am always executed

Input:

3
10 20 30

Output
java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3
I am always executed

For example:

Test	Input	Result
1	6 1 0 4 1 2 8	java.lang.ArithmeticException: / by zero I am always executed

SOLUTION :

```
import java.util.Scanner;

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

      // Read the size of the array int
      size = scanner.nextInt();

      // Initialize the array int[] numbers
      = new int[size];

      // Read the elements into the array
      for (int i = 0; i < size; i++) {
      numbers[i] = scanner.nextInt();
      }
try {
         // Attempt to perform division
         int result = numbers[0] / numbers[1]; // This may cause an ArithmeticException
      } catch (ArithmeticException e) {
         System.out.println(e); // Catch division by zero
      } catch (ArrayIndexOutOfBoundsException e) {
         System.out.println(e); // Catch accessing out of bounds
      } catch (Exception e) {
         System.out.println(e); // Catch any other exceptions
      } finally {
         // This block is always executed }
try {
         // Attempt to access an out-of-bounds index
         int outOfBoundsValue = numbers[3]; // This will trigger
ArrayIndexOutOfBoundsException if size < 4
      } catch (ArrayIndexOutOfBoundsException e) { System.out.println(e);
      } finally {
         // This block is always executed for the second try System.out.println("I am
         always executed");
      }

      scanner.close();
   }
}
```

OUTPUT :

	Test	Input	Expected	Got	
✓	1	6 1 0 4 1 2 8	java.lang.ArithmeticException: / by zero I am always executed	java.lang.ArithmeticException: / by zero I am always executed	✓
✓	2	3 10 20 30	java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3 I am always executed	java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3 I am always executed	✓

Passed all tests! ✓

Lab-10- Collection- List

1.

Given an ArrayList, the task is to get the first and last element of the ArrayList in Java.

```
Input: ArrayList = [1, 2, 3, 4]
Output: First = 1, Last = 4

Input: ArrayList = [12, 23, 34, 45, 57, 67, 89]
Output: First = 12, Last = 89
```

Approach:

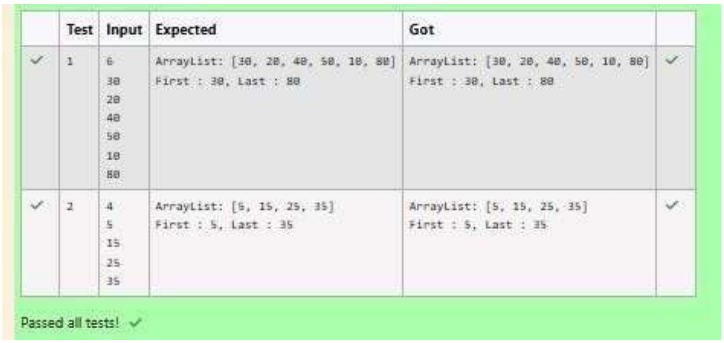
1. Get the ArrayList with elements.
2. Get the first element of ArrayList using the get(index) method by passing index = 0.
3. Get the last element of ArrayList using the get(index) method by passing index = size – 1.

SOLUTION :

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

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

      // Create an ArrayList
      ArrayList<Integer> numbers = new ArrayList<>();
```

```
        int numElements = scanner.nextInt();

        for (int i = 0; i < numElements; i++) { int
            number = scanner.nextInt();
            numbers.add(number);
        }
        System.out.println("ArrayList: " + numbers);

        // Get the first element int
        firstElement = numbers.get(0);

        // Get the last element
        int lastElement = numbers.get(numbers.size() - 1);

        // Print the results
        System.out.print("First : " + firstElement);
        System.out.println(", Last : " + lastElement);
    }
}
```

OUTPUT :



	Test	Input	Expected	Got	
✓	1	6 30 20 40 50 10 80	ArrayList: [30, 20, 40, 50, 10, 80] First : 30, Last : 80	ArrayList: [30, 20, 40, 50, 10, 80] First : 30, Last : 80	✓
✓	2	4 5 15 25 35	ArrayList: [5, 15, 25, 35] First : 5, Last : 35	ArrayList: [5, 15, 25, 35] First : 5, Last : 35	✓

Passed all tests! ✓

2.

The given Java program is based on the ArrayList methods and its usage. The Java program is partially filled. Your task is to fill in the incomplete statements to get the desired output.

list.set();

list.indexOf());

list.lastIndexOf())

list.contains()

list.size());

list.add();

list.remove();

The above methods are used for the below Java program.

SOLUTION :

```
import
java.util.ArrayList;
import   java.util.Scanner;
public class Prog {
```

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

```
 Scanner sc= new Scanner(System.in);
 int n = sc.nextInt();

 ArrayList<Integer> list = new ArrayList<Integer>();

 for(int i = 0; i<n;i++)
 list.add(sc.nextInt());

 // printing initial value ArrayList
 System.out.println("ArrayList: " + list);

//Replacing the element at index 1 with 100
list.set(1,100);

 //Getting the index of first occurrence of 100
System.out.println("Index of 100 = "+ list.indexOf(100)             );

//Getting the index of last occurrence of 100
 System.out.println("LastIndex of 100 = "+ list.lastIndexOf(100));
// Check whether 200 is in the list or not
System.out.println(list.contains(200)); //Output : false
 // Print ArrayList size
 System.out.println("Size Of ArrayList = "+list.size() );
//Inserting 500 at index 1
list.add(1,500);                              // code here
 //Removing an element from position 3
list.remove(3);                             // code here
System.out.print("ArrayList: " + list);
 }
}
```

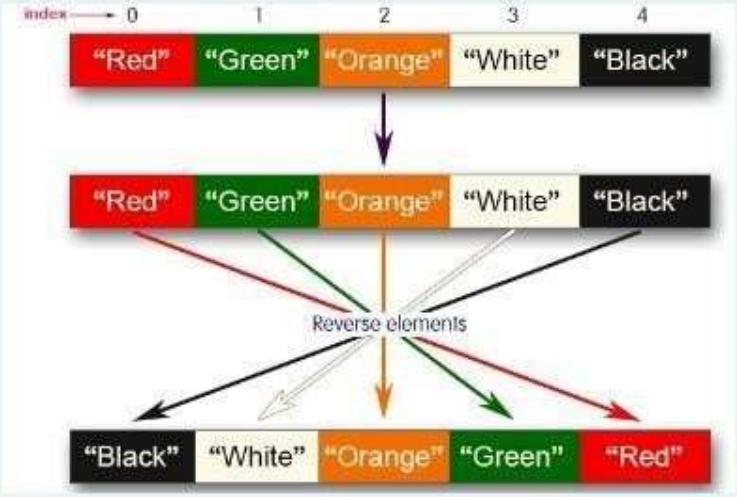
OUTPUT :

	Test	Input	Expected	Got	
✓	1	5 1 2 3 100 5	ArrayList: [1, 2, 3, 100, 5] Index of 100 = 1 LastIndex of 100 = 3 false Size Of ArrayList = 5 ArrayList: [1, 500, 100, 100, 5]	ArrayList: [1, 2, 3, 100, 5] Index of 100 = 1 LastIndex of 100 = 3 false Size Of ArrayList = 5 ArrayList: [1, 500, 100, 100, 5]	✓

Passed all tests! ✓

3.

Write a Java program to reverse elements in an array list.




```
Sample input and Output:
Red
Green
Orange
White
Black
Sample output
List before reversing :
[Red, Green, Orange, White, Black]
List after reversing :
[Black, White, Orange, Green, Red]
```

SOLUTION :

```
import   java.util.ArrayList;
import
java.util.Collections;
import java.util.Scanner;

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

      ArrayList<String> list = new ArrayList<>();
      int n = scanner.nextInt();

      for (int i = 0; i < n; i++) {
         String element = scanner.next();
         list.add(element);
      }

      System.out.println("List before reversing : ");
      System.out.println(list);

      Collections.reverse(list);

      System.out.println("List after reversing : ");
      System.out.println(list);
}
}
```

OUTPUT :

	Test	Input	Expected	Got	
✓	1	5 Red Green Orange White Black	List before reversing : [Red, Green, Orange, White, Black] List after reversing : [Black, White, Orange, Green, Red]	List before reversing : [Red, Green, Orange, White, Black] List after reversing : [Black, White, Orange, Green, Red]	✓
✓	2	4 CSE AIML AIDS CYBER	List before reversing : [CSE, AIML, AIDS, CYBER] List after reversing : [CYBER, AIDS, AIML, CSE]	List before reversing : [CSE, AIML, AIDS, CYBER] List after reversing : [CYBER, AIDS, AIML, CSE]	✓

Passed all tests! ✓

Lab-11-Set, Map

1.

Java HashSet class implements the Set interface, backed by a hash table which is actually a HashMap instance.

No guarantee is made as to the iteration order of the hash sets which means that the class does not guarantee the constant order of elements over time.

This class permits the null element.

The class also offers constant time performance for the basic operations like add, remove, contains, and size assuming the hash function disperses the elements properly among the buckets.

Java HashSet Features

A few important features of HashSet are mentioned below:

- Implements Set Interface.
- The underlying data structure for HashSet is Hashtable.
- As it implements the Set Interface, duplicate values are not allowed.
- Objects that you insert in HashSet are not guaranteed to be inserted in the same order. Objects are inserted based on their hash code.
- NULL elements are allowed in HashSet.
- HashSet also implements **Serializable** and **Cloneable** interfaces.
- `public class HashSet<E> extends AbstractSet<E> implements Set<E>, Cloneable, Serializable`

```
Sample Input and Output:
5
90
56
45
78
25
78
Sample Output:
78 was found in the set.
Sample Input and output:
3
2
7
9
5
Sample Input and output:
5 was not found in the set.
```

SOLUTION :

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

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

        // Read the number of elements int n = sc.nextInt();
```

```
        // Create a HashSet object to store numbers
        HashSet<Integer> numbers = new HashSet<>();

        // Add numbers to the HashSet for
        (int i = 0; i < n; i++) {
        numbers.add(sc.nextInt());
        }

        // Read the search key
        int skey = sc.nextInt();

        // Check if skey is present in the HashSet
        if (numbers.contains(skey)) {
            System.out.println(skey + " was found in the set.");
        } else {
            System.out.println(skey + " was not found in the set.");
        }

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

OUTPUT :

	Test	Input	Expected	Got	
✓	1	5 90 56 45 78 25 78	78 was found in the set.	78 was found in the set.	✓
✓	2	3 -1 2 4 5	5 was not found in the set.	5 was not found in the set.	✓

Passed all tests! ✓

2.

Write a Java program to compare two sets and retain elements that are the same.

Sample Input and Output:

5

Football

Hockey

Cricket

Volleyball

Basketball

7 // **HashSet 2:**

Golf

Cricket

Badminton

Football

Hockey

Volleyball

Handball

SAMPLE OUTPUT:

Football

Hockey

Cricket

Volleyball

Basketball

SOLUTION :

```
import java.util.HashSet; import
java.util.Scanner;
import java.util.Set;

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

      // Read the size of the first set
      int size1 = Integer.parseInt(scanner.nextLine());

      // Create a HashSet to store the first set of elements
      Set<String> set1 = new HashSet<>();

      // Read elements for the first set for (int
      i = 0; i < size1; i++) {
      set1.add(scanner.nextLine());
      }
      // Read the size of the second set
```

```
        int size2 = Integer.parseInt(scanner.nextLine());

        // Create a HashSet to store the second set of elements
        Set<String> set2 = new HashSet<>();

        // Read elements for the second set
        for (int i = 0; i < size2; i++) {
        set2.add(scanner.nextLine());
        }

        // Retain common elements using the retainAll() method
        set1.retainAll(set2);

        // Print the common elements for
        (String element : set1) {
            System.out.println(element);
        }

        scanner.close();
    }
}
```

OUTPUT :

	Test	Input	Expected	Got	
✓	1	5 Football Hockey Cricket Volleyball Basketball 7 Golf Cricket Badminton Football Hockey Volleyball Throwball	Cricket Hockey Volleyball Football	Cricket Hockey Volleyball Football	✓
✓	2	4 Toy Bus Car Auto 3 Car Bus Lorry	Bus Car	Bus Car	✓

Passed all tests! ✓

3.

Java HashMap Methods

containsKey() Indicate if an entry with the specified key exists in the map

containsValue() Indicate if an entry with the specified value exists in the map

putIfAbsent() Write an entry into the map but only if an entry with the same key does not already exist

remove() Remove an entry from the map

replace() Write to an entry in the map only if it exists

size() Return the number of entries in the map

Your task is to fill the incomplete code to get desired output

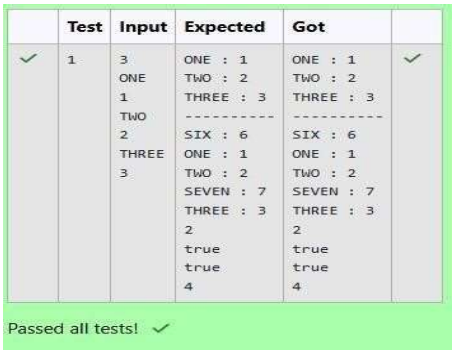
SOLUTION :

```
import
java.util.HashMap;
import
java.util.Map.Entry;
import   java.util.Scanner;
import java.util.Set; public
class Prog {

   public static void main(String[] args) {
      // Creating HashMap with default initial capacity and load factor
      HashMap<String, Integer> map = new HashMap<String, Integer>();

      String name; int
      num;
      Scanner sc = new Scanner(System.in); int
      n = sc.nextInt();

      for (int i = 0; i < n; i++) {
         name = sc.next(); num
         = sc.nextInt(); map.put(name,
         num);
      }
      // Printing key-value pairs
      Set<Entry<String, Integer>> entrySet = map.entrySet();

      for (Entry<String, Integer> entry : entrySet) {
         System.out.println(entry.getKey() + " : " + entry.getValue());
      }
      System.out.println(" ---------- ");

      // Creating another HashMap
      HashMap<String, Integer> anotherMap = new HashMap<String, Integer>();
```

```
// Inserting key-value pairs to anotherMap using put() method
anotherMap.put("SIX", 6);
```

```
        anotherMap.put("SEVEN", 7);

        // Inserting key-value pairs of map to anotherMap using putAll() method
        anotherMap.putAll(map); // This line fills in the missing code

        // Printing key-value pairs of anotherMap entrySet
        = anotherMap.entrySet();

        for (Entry<String, Integer> entry : entrySet) {
            System.out.println(entry.getKey() + " : " + entry.getValue());
        }

        // Adds key-value pair 'FIVE-5' only if it is not present in map
        map.putIfAbsent("FIVE", 5);

        // Retrieving a value associated with key 'TWO'
        int value = map.get("TWO");
        System.out.println(value); // Prints the value associated with key "TWO" (if it exists)

        // Checking whether key 'ONE' exists in map
        System.out.println(map.containsKey("ONE")); // Prints true if "ONE" is a key, false otherwise

        // Checking whether value '3' exists in map
        boolean valueExists = map.containsValue(3); // You can use a variable to store the result
        System.out.println(valueExists); // Prints true if value 3 exists in the map, false otherwise

        // Retrieving the number of key-value pairs present in map
        System.out.println(map.size()); // Prints the number of entries in the map
    }
}
```

OUTPUT :



	Test	Input	Expected	Got	
✓	1	3 ONE 1 TWO 2 THREE 3	ONE : 1 TWO : 2 THREE : 3 ---------- SIX : 6 ONE : 1 TWO : 2 SEVEN : 7 THREE : 3 2 true true 4	ONE : 1 TWO : 2 THREE : 3 ---------- SIX : 6 ONE : 1 TWO : 2 SEVEN : 7 THREE : 3 2 true true 4	✓

Passed all tests! ✓

Lab-12-Introduction to I/O, I/O Operations, Object Serialization

1.

You are provided with a string which has a sequence of 1's and 0's.

This sequence is the encoded version of a English word. You are supposed write a program to decode the provided string and find the original word.

Each alphabet is represented by a sequence of 0s.

This is as mentioned below:

Z : 0

Y : 00

X : 000

W : 0000

V : 00000

U : 000000

T : 0000000

and so on upto A having 26 0's (00000000000000000000000000).

The sequence of 0's in the encoded form are separated by a single 1 which helps to distinguish between 2 letters.

Example 1:

input1: 010010001

The decoded string (original word) will be: ZYX

Example 2:

input1: 0000100000000000000000010000000000010000000001000000000001

The decoded string (original word) will be: WIPRO

Note: The decoded string must always be in UPPER case.

SOLUTION :

```
import java.util.Scanner;

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

      StringBuilder decodedString = new StringBuilder();
      int count = 0;

      for (int i = 0; i < encodedString.length(); i++) {
         if (encodedString.charAt(i) == '0') {
         count++;
         } else { char decodedChar = (char) ('Z' - count
             + 1); decodedString.append(decodedChar);
            count = 0;
         }
      }

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

OUTPUT :

	Input	Expected	Got	
✓	010010001	ZYX	ZYX	✓
✓	00001000000000000000000100000000000100000000010000000000001	WIPRO	WIPRO	✓

Passed all tests! ✓

2.

Given two char arrays input1[] and input2[] containing only lower case alphabets, extracts the alphabets which are present in both arrays (common alphabets).

Get the ASCII values of all the extracted alphabets.

Calculate sum of those ASCII values. Lets call it sum1 and calculate single digit sum of sum1, i.e., keep adding the digits of sum1 until you arrive at a single digit.

Return that single digit as output.

Note:

1. Array size ranges from 1 to 10.
2. All the array elements are lower case alphabets.
3. Atleast one common alphabet will be found in the arrays.

Example 1:

input1: {'a', 'b', 'c'}

input2: {'b', 'c'}

output: 8

Explanation:

'b' and 'c' are present in both the arrays.

ASCII value of 'b' is 98 and 'c' is 99.

98 + 99 = 197

1 + 9 + 7 = 17

1 + 7 = 8

For example:

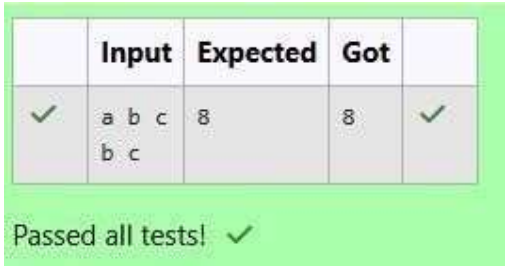
Input	Result
a b c b c	8

SOLUTION :

```
import java.util.HashSet; import java.util.Set; public class
CommonAlphabetSum {

  public static int singleDigitSum(int num) { int sum = 0; while (num > 0) { sum
    += num % 10; num /= 10;
    }
    if (sum > 9) { return singleDigitSum(sum); }
```

```
        return sum;
    }

    public static int calculateCommonAlphabetSum(char[] input1, char[] input2) {
        Set<Character> set1 = new HashSet<>(); for (char c : input1) { set1.add(c);
        }

        int sum = 0; for
        (char c : input2) { if
            (set1.contains(c))  {
            sum += c;
            }
        }

        return singleDigitSum(sum);
    }

    public static void main(String[] args)
        { char[] input1 = {'a', 'b', 'c'};
        char[] input2 = {'b', 'c', 'd'};

        int result = calculateCommonAlphabetSum(input1, input2);
    System.out.println(result); }
}
```

OUTPUT :



3.

Write a function that takes an input String (sentence) and generates a new String (modified sentence) by reversing the words in the original String, maintaining the words position.

In addition, the function should be able to control the reversing of the case (upper or lowercase) based on a case_option parameter, as follows:

If case_option = 0, normal reversal of words i.e., if the original sentence is "Wipro TechNologies BangaLore", the new reversed sentence should be "orpiW seigoloNhceT eroLagnaB".

If case_option = 1, reversal of words with retaining position's case i.e., if the original sentence is "Wipro TechNologies BangaLore", the new reversed sentence should be "Orpiw SeigOlonhcet ErolaGnab".

Note that positions 1, 7, 11, 20 and 25 in the original string are uppercase W, T, N, B and L.

Similarly, positions 1, 7, 11, 20 and 25 in the new string are uppercase O, S, O, E and G.

NOTE:

1. Only space character should be treated as the word separator i.e., "Hello World" should be treated as two separate words, "Hello" and "World". However, "Hello,World", "Hello;World", "Hello-World" or "Hello/World" should be considered as a single word.
2. Non-alphabetic characters in the String should not be subjected to case changes. For example, if case option = 1 and the original sentence is "Wipro TechNologies, Bangalore" the new reversed sentence should be "Orpiw ,seiGolonhceT Erolagnab". Note that comma has been treated as part of the word "Technologies," and when comma had to take the position of uppercase T it remained as a comma and uppercase T took the position of comma. However, the words "Wipro and Bangalore" have changed to "Orpiw" and "Erolagnab".
3. Kindly ensure that no extra (additional) space characters are embedded within the resultant reversed String.

Examples:

S. No.	input1	input2	output
1	Wipro Technologies Bangalore	0	orpiW seigolonhceT erolagnaB
2	Wipro Technologies, Bangalore	0	orpiW ,seigolonhceT erolagnaB
3	Wipro Technologies Bangalore	1	Orpiw Seigolonhcet Erolagnab
4	Wipro Technologies, Bangalore	1	Orpiw ,seigolonhceT Erolagnab

For example:

Input	Result
Wipro Technologies Bangalore 0	orpiW seigolonhceT erolagnaB
Wipro Technologies, Bangalore 0	orpiW ,seigolonhceT erolagnaB
Wipro Technologies Bangalore 1	Orpiw Seigolonhcet Erolagnab
Wipro Technologies, Bangalore 1	Orpiw ,seigolonhceT Erolagnab

SOLUTION :

```
import java.util.Scanner; public

class WordReverser {

   public static String reverseWordsWithCase(String sentence, int caseOption) {
      // Split the sentence into words based on spaces String[] words =
      sentence.split(" ");

      // StringBuilder to store the result
      StringBuilder result = new StringBuilder();

      // Process each word for
      (String word : words) {
         // Reverse the word
         String reversedWord = new StringBuilder(word).reverse().toString();

         if (caseOption == 0) {
            // If caseOption is 0, no case conversion, just reverse the word
            result.append(reversedWord).append(" ");
         } else if (caseOption == 1) {
            // If caseOption is 1, adjust the case while maintaining original letter
positions
```

```
            result.append(applyCaseConversion(reversedWord, word)).append(" ");
        }
    }

    // Remove the trailing space and return the result return result.toString().trim();
  }

  private static String applyCaseConversion(String reversedWord, String
originalWord) {
    // StringBuilder to store the adjusted word
    StringBuilder adjustedWord = new StringBuilder();

    // Iterate over each character in the reversed word for
    (int i = 0; i < reversedWord.length(); i++) { char
    reversedChar = reversedWord.charAt(i); char
    originalChar = originalWord.charAt(i);

      if (Character.isLowerCase(originalChar)) {
        // If the original character was lowercase, the reversed character should be
uppercase adjustedWord.append(Character.toLowerCase(reversedChar));
      } else if (Character.isUpperCase(originalChar)) {
        // If the original character was uppercase, the reversed character should be
lowercase adjustedWord.append(Character.toUpperCase(reversedChar));
      } else {
        //          Non-alphabetic          characters          remain          unchanged
      adjustedWord.append(reversedChar); }
    }

    return adjustedWord.toString();
  }

  public static void main(String[] args) {
    // Create a Scanner object to get input from the user Scanner

    scanner = new Scanner(System.in); // Get sentence input from

    the user

    String sentence = scanner.nextLine(); //

    Get case option input from the user int

    caseOption = scanner.nextInt();

    // Validate the case option
    if (caseOption != 0 && caseOption != 1) {
```

```
            System.out.println("Invalid case option. Please enter 0 or 1.");
        } else {
            // Call the function and print the result
            String result = reverseWordsWithCase(sentence, caseOption);
            System.out.println(result);
        }

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

OUTPUT :

	Input	Expected	Got	
✓	Wipro Technologies Bangalore 0	orpiW seigolonhceT erolagnaB	orpiW seigolonhceT erolagnaB	✓
✓	Wipro Technologies, Bangalore 0	orpiW ,seigolonhceT erolagnaB	orpiW ,seigolonhceT erolagnaB	✓
✓	Wipro Technologies Bangalore 1	Orpiw Seigolonhcet Erolagnab	Orpiw Seigolonhcet Erolagnab	✓
✓	Wipro Technologies, Bangalore 1	Orpiw ,seigolonhceT Erolagnab	Orpiw ,seigolonhceT Erolagnab	✓

Passed all tests! ✓

STUDENT MANAGEMENT SYSTEM

Submitted by

SUDHAKAR V(231001219)

SUDHARSAN D (231001220)

VIGNESH KUMAR S (231001241)

MINI PROJECT REPORT

for

JAVA(CS23333)

DEPARTMENT OF INFORMATION TECNOLOY RAJALAKSMI ENGINEERING COLLEGE

BONAFIDE CERTIFICATE

Certified that this project report titled **"STUDENT MANAGEMENT SYSTEM"** is the **BONAFIDE** work of **SUDHAKAR V (231001219), SUDHARSAN D (231001220),VIGNESH KUMAR S(231001241)** who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr.Valamathi

Head of The Department

Department of Information Technology

Rajalakshmi Engineering College

SIGNATURE

Mrs.K E Narayana

Assistant Professor,

Department of Information Technology

Rajalakshmi Engineering College

Submitted to Project Viva-Voce Examination held on

Internal Examiner

External Examiner

ABSTRACT

The **Student Management System** is a comprehensive desktop-based application designed to streamline and automate the management of student records in an educational institution. Built using **Java Swing** for the graphical user interface and **Oracle Database** for secure and scalable data storage, this system provides an efficient and user-friendly solution for managing student information.

The project focuses on core operations such as adding, updating, deleting, and viewing student details, including **Student ID**, **First Name**, **Last Name**, **Date of Birth (DOB)**, **Gender**, **CGPA**, **Address**, and **Email**. The integration of advanced Java libraries, such as **JCalendar** for date selection and **Oracle JDBC Connector** for database connectivity, ensures a seamless and robust user experience.

Purpose and Motivation

Educational institutions often struggle with outdated or manual processes for handling student information. Such systems are prone to inefficiency, errors, and data loss, leading to challenges in maintaining accurate and consistent records. The primary motivation of this project is to replace these traditional methods with a fully automated system that is secure, scalable, and easy to use.

Key Features and Functionalities

1. **Add New Records**: The system allows administrators to input student details through a graphical form with built-in validation to ensure the correctness of data.
2. **Update Records**: Existing student records can be modified with updated information while maintaining data integrity.
3. **Delete Records**: Administrators can remove outdated or irrelevant student data directly through the system.
4. **View Records**: Displays all stored student details in a tabular format with options for sorting and filtering.
5. **Validation**: Includes input validations for fields such as email, DOB, and CGPA to ensure proper data entry.
6. **Database Security**: Oracle Database is utilized to securely store and manage data, offering robust query execution and backup mechanisms.

Significance of the Project

The **Student Management System** enhances productivity by automating repetitive tasks, reducing the chances of errors, and improving data accessibility. Its modular and scalable design ensures adaptability for future enhancements, such as integrating features like data export to Excel or remote database access.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER - 1 INTRODUCTION

Motivation of the Project

The motivation behind developing the Student Management System stems from the growing need for an efficient, secure, and user-friendly solution to manage student records in educational institutions. With the increasing number of students, courses, and associated data, traditional manual or semi-digital systems have proven to be inadequate. This project aims to overcome the limitations of existing methods and provide a robust and scalable alternative.

1. Addressing Limitations of Existing Systems

- **Data Accuracy**: Manual systems are prone to human errors, leading to inaccuracies in student records. This project is motivated by the need to eliminate such errors through automation and validation mechanisms.
- **Operational Efficiency**: Tasks like adding, updating, and retrieving student records are time-consuming in traditional systems. The proposed system seeks to automate these operations, reducing the administrative workload.
- **Data Accessibility**: In manual or semi-digital systems, accessing student records is often slow and limited to specific locations or files. This project aims to enable real-time data access, making information retrieval faster and more efficient.

2. Improving Data Security • Educational institutions handle sensitive student information, including personal details, academic records, and contact information. However, manual and semi-digital systems lack robust security measures, leaving data vulnerable to loss or unauthorized access. The motivation for this project lies in implementing a secure database system (Oracle) to ensure data confidentiality, integrity, and availability.

Project Objectives

The **Student Management System** project has been designed with the following key objectives in mind:

1. Providing a User-Friendly Interface • The system aims to simplify student record management by offering an intuitive and visually appealing **Graphical User Interface (GUI)** built using **Java Swing**.

- The interface is designed for non-technical users, ensuring ease of navigation and usability.
- Key features include form-based input for data entry, table views for displaying records, and buttons for CRUD (Create, Read, Update, Delete) operations.

2. Automating CRUD Operations

- One of the primary goals is to automate repetitive administrative tasks to save time and effort.
- CRUD functionalities allow administrators to:
  - **Create**: Add new student records. ○ **Read**: View all existing student records in a tabular format. ○ **Update**: Modify specific details of a student (e.g., CGPA, email, address).
  - **Delete**: Remove outdated or invalid student records.
- Automation eliminates the need for manual record-keeping, reducing human errors and improving efficiency.

3. Ensuring Data Security

- The system incorporates a **robust Oracle Database** to store and manage student records securely.
- Secure access to the database is ensured through **Oracle JDBC Connector**, which restricts unauthorized access.
- Sensitive data like student personal details and email addresses are protected using the database's in-built security features.

4. Maintaining Data Integrity • The system implements validation mechanisms to ensure the accuracy and reliability of data.

- Examples of data validation:
  - **Email Validation**: Ensures the entered email address is in the correct format.
  - **CGPA Range**: Limits the CGPA to a valid range (e.g., 0.0 to 10.0).
  - **Date Validation**: Uses **JCalendar** to prevent invalid date entries for DOB.
- Consistency in data entry reduces the risk of errors and ensures the integrity of the stored information.

5. Enhancing Scalability • The system is designed to handle increasing volumes of data as the institution grows.

- Features like modular database design and efficient query execution make it adaptable for future requirements (e.g., adding new fields like emergency contact or course history).

Proposed System

The **Proposed System** focuses on delivering a modern and efficient solution to overcome the limitations of the existing manual or semi-digital systems. Below are the key aspects of the proposed system:

1. Automation of Tasks • The system automates critical operations related to student record management, such as adding, updating, deleting, and retrieving data.

- Eliminates repetitive manual work, reducing administrative workload and improving productivity.
- Tasks are executed in real-time, ensuring instant updates to the database.

2. Seamless Data Handling

- The proposed system uses **Oracle Database** to manage and store student information securely and efficiently.
- Features include:
    - **Structured Tables**: The database is organized into well-defined tables with fields for each student attribute (e.g., ID, Name, DOB, Gender, CGPA, Address, Email).
    - **Real-Time Access**: Data retrieval and updates occur instantaneously, allowing for smooth interaction between the user interface and the database. ○ **Dynamic Updates**: Changes made to student records (e.g., updating an address or deleting a record) are reflected immediately.

3. Robust Database for Secure Storage

- The **Oracle Database** serves as the backbone of the system, ensuring secure and reliable data storage.
- Key features of the database design:
    - **Primary Keys**: Ensures each student record is unique (e.g., Student ID as the primary key). ○ **Constraints**: Implements validation rules at the database level to prevent invalid entries.
    - **Data Backup**: Oracle's built-in features allow for regular backups, reducing the risk of data loss. ○ **Role-Based Access Control**: Ensures that only authorized users can access or modify the database, enhancing security.

4. User-Friendly Features

- The **Java Swing** interface provides an easy-to-use environment for interacting with the system.
- Features include:
    - **JCalendar**: Integrated for date selection, ensuring accurate DOB input.
    - **Buttons and Forms**: Simplifies CRUD operations with clearly labeled buttons like "Add," "Update," "Delete," and "View."
    - **Table View**: Displays all student records in a structured format, with options for sorting and filtering.

5. Validation Mechanisms

- The system incorporates both front-end and back-end validation to ensure accurate data handling.
- Examples of validations:

- **Front-End Validation**: Checks user inputs before sending data to the database (e.g., email format, DOB selection). ○ **Back-End Validation**: Adds an extra layer of protection by applying constraints within the database schema.

CHAPTER 2 SYSTEM DESIGN

Chapter 2 focuses on the detailed architecture, flow, and technical requirements of the **Student Management System**. It provides a comprehensive overview of the system's structure and design. Below is a detailed explanation of each heading under this chapter:

2.1 Introduction

This section introduces the concept of system design and explains its importance in the development process:

- **Purpose of System Design**: Ensures that the application meets the defined objectives and functions effectively by creating a blueprint of the system.
- **Key Focus Areas**:
  - How various components (e.g., user interface, database) interact.
  - Ensuring scalability, security, and usability.
- **Relevance to the Project**: In the context of the **Student Management System**, system design involves planning the architecture, database schema, user interface, and data flow.

2.2 System Architecture

This section provides an overview of the system's architecture, explaining the interaction between different layers:

- **Three-Layered Architecture**:
  1. **Presentation Layer (Frontend)**:
     - Built using **Java Swing**.
     - Provides the graphical user interface for interacting with the system (e.g., adding, updating, or viewing student records).
  2. **Application Layer (Logic)**:
     - The core logic written in **Java** connects the frontend to the backend.
     - Handles operations like input validation, database queries, and processing CRUD operations.
  3. **Data Layer (Backend)**:
     - An **Oracle Database** stores all student-related data securely.

        - Managed through **Oracle JDBC Connector**, which allows communication between the application and the database.
- **Flow**: User inputs data → Application processes the input → Data is stored/retrieved from the database.

2.3 System Requirements

This section specifies the hardware and software requirements for developing and running the system:

Hardware Requirements

- **Processor**: Minimum Dual Core or higher for faster processing.
- **RAM**: Minimum 4GB; recommended 8GB for smooth application execution.
- **Storage**: At least 500MB for program files and database storage.

Software Requirements

- **Operating System**: Windows 10 or higher / Linux.
- **Development Tools**:
    - **JDK (Java Development Kit)**: For coding and compiling the application. ○ **Java Swing**: For creating the GUI. ○ **JCalendar Library**: For date selection in forms.
    - **Oracle JDBC Connector**: To enable database connectivity.
- **Database**: Oracle 11g or higher for data storage.

2.4 Database Design

This section explains the structure of the database and its role in managing student records efficiently:

- **Database Schema**: Defines the tables, fields, and relationships within the database.
- **Student Table**:
    - **Columns**:
        - Student_ID (Primary Key): Unique identifier for each student.
        - First_Name, Last_Name: For storing the student's full name.
        - DOB: Date of Birth (validated using JCalendar).
        - Gender: Stores the gender of the student (e.g., Male, Female, Other).
        - CGPA: Stores the cumulative grade point average (validated for correct range).
        - Address: Stores the residential address.
        - Email: Stores the student's email (validated for correct format).
    - **Relationships**: Single-table structure for simplicity but can be extended for relationships (e.g., linking courses or departments).
- **Constraints**:
    - **Primary Key**: Ensures each record is unique (e.g., Student_ID). ○ **Not Null**:

Mandatory fields such as Name, DOB, and CGPA. ○ **Check Constraints**: Validates CGPA and DOB fields.

2.5 Data Flow Design

This section illustrates how data flows through the system and ensures efficient interaction between components:

- **Data Flow Diagram (DFD)**: ○ **Level 0**: High-level overview of the system (inputs and outputs). ○ **Level 1**:
  Explains the core processes like adding, updating, and deleting student records.
- **Workflow**:

1. User interacts with the frontend (e.g., fills a form or clicks a button).
2. The application logic validates the input.
3. The Oracle JDBC Connector sends SQL queries to the database.
4. Data is stored, retrieved, or modified in the database.
5. The output is displayed on the frontend in real-time.

CHAPTER 3

PROJECT DESCRIPTION

Chapter 3 provides an in-depth explanation of the **Student Management System**, including the methodologies used, detailed module descriptions, and results and discussions. Below is a breakdown of each heading:

3.1 Methodologies

This section explains the methodologies and techniques applied during the development of the project.

1. Software Development Lifecycle (SDLC) • **Model Used**: Waterfall Model ○ Requirements were gathered first, followed by design, implementation, testing, deployment, and maintenance. ○ Sequential approach ensured that each stage was completed before moving to the next.

2. Object-Oriented Programming (OOP) Principles

The project used **OOP concepts** for better structure and reusability:

- **Encapsulation**: Restricted direct access to student attributes by using getter and setter methods.
- **Inheritance**: Reused functionality in different parts of the system.
- **Polymorphism**: Allowed methods like saveStudent() and updateStudent() to handle multiple types of operations.

- **Abstraction**: Simplified database connectivity by abstracting it into utility classes.

3. Java Swing for GUI

- Used for creating a dynamic and interactive graphical user interface.
- Included components such as forms, buttons, and tables.

4. Oracle Database

- Managed relational data with a well-structured schema.
- SQL queries were used for CRUD operations.

5. Integration with Oracle JDBC

- Established communication between the Java application and Oracle Database. • Handled database connections, queries, and result sets efficiently.

3.2 Module Descriptions

This section details the various modules in the **Student Management System** and their functionalities.

1. Login Module

- **Purpose**: Authenticates users before granting access to the system.
- **Key Features**:
  - Username and password validation. o

    Security measures to prevent unauthorized access.

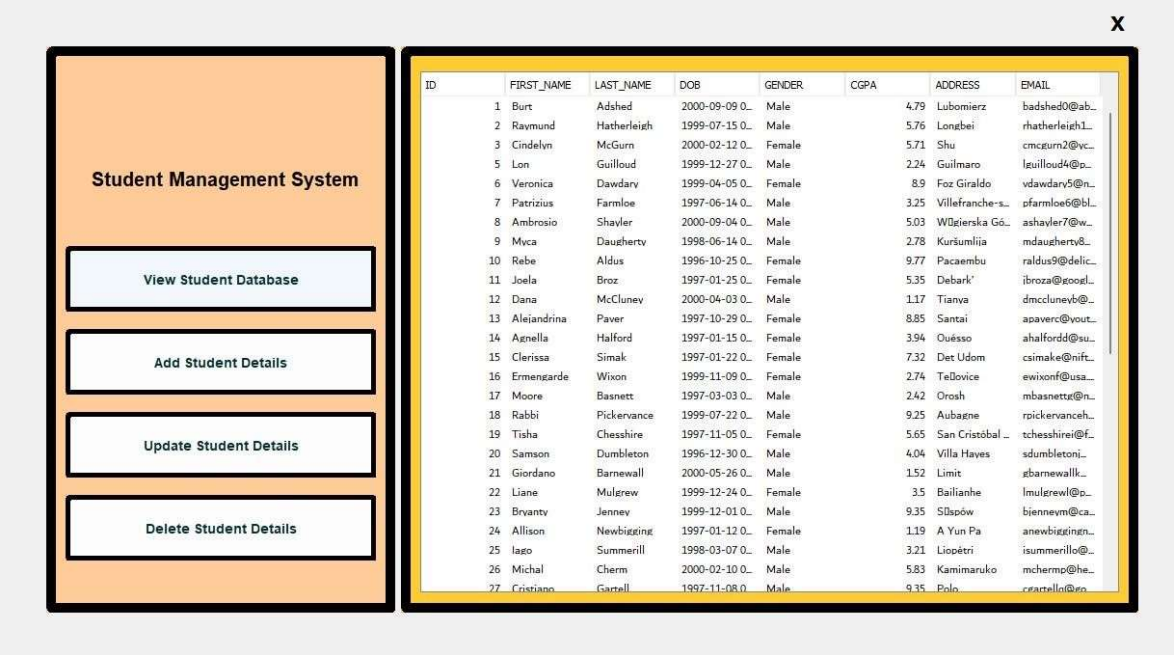


Fig :3.1 Main page

2. Add Student Module

- **Purpose:** Enables users to input new student records.
- **Features:**
  - Form-based input for student details like ID, first name, last name, DOB, gender, CGPA, address, and email.
  - Validation for mandatory fields and formats (e.g., CGPA range, email format).
  - Data is stored in the Oracle database**.**

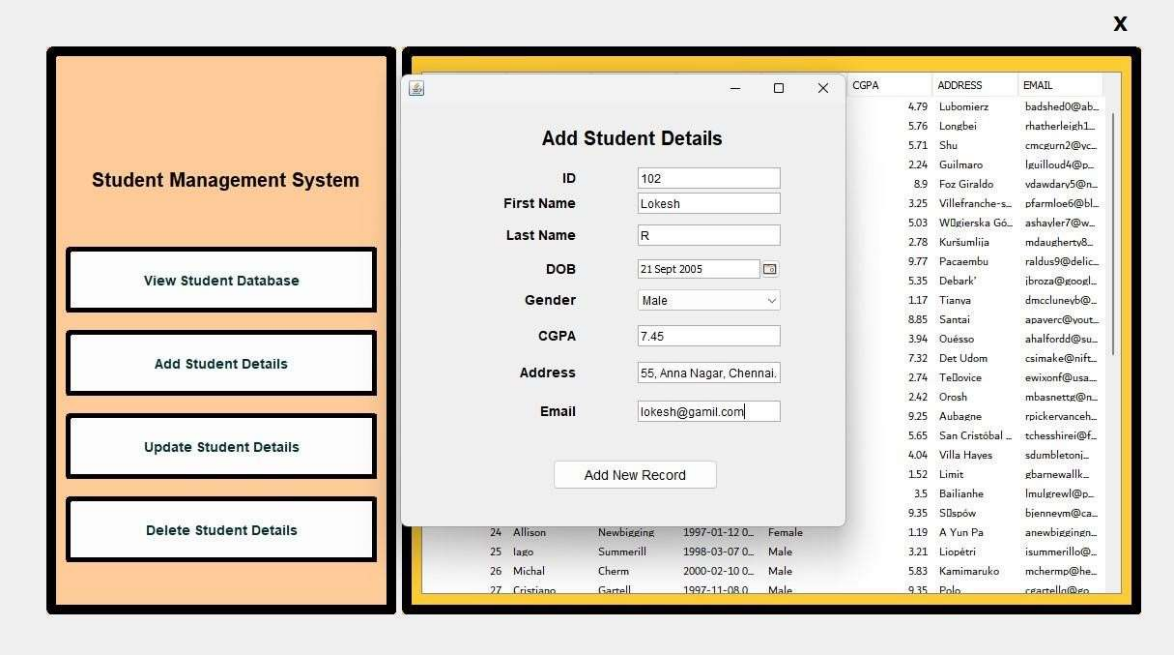


Fig3.2 Adding Data

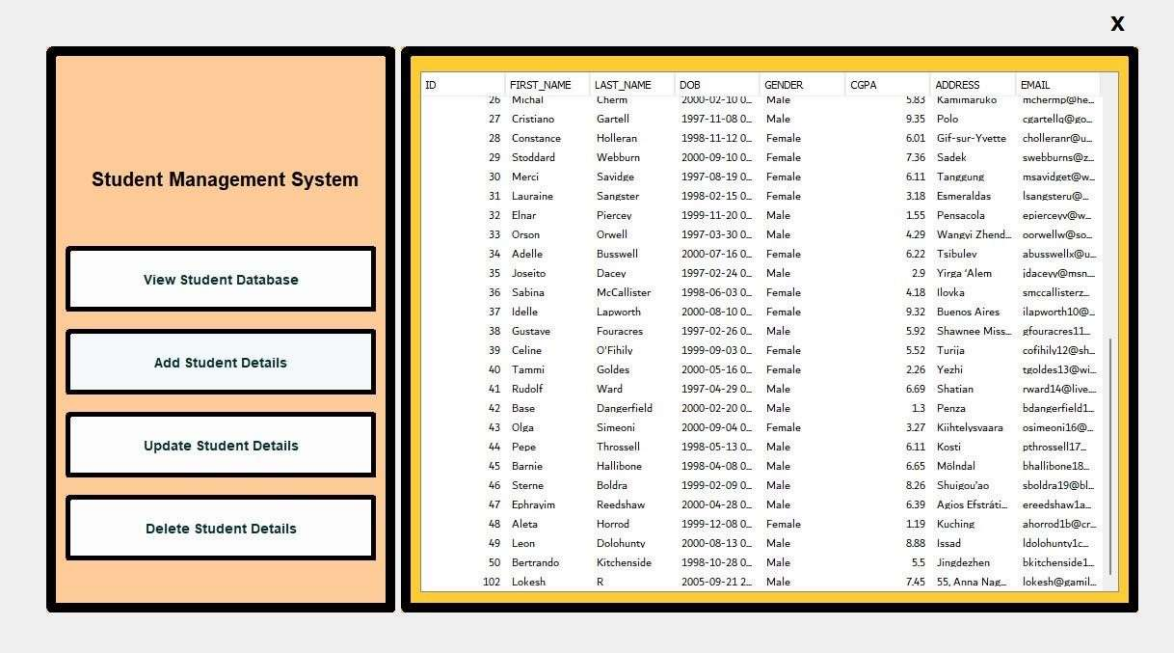


Fig 3.3 After Adding Data

3. Update Student Module

- **Purpose**: Allows modification of existing student records.
- **Features**:

- Search functionality to locate a student by ID or name.
- Editable fields for updating information such as address, CGPA, or email.
- Realtime reflection of changes in the database.

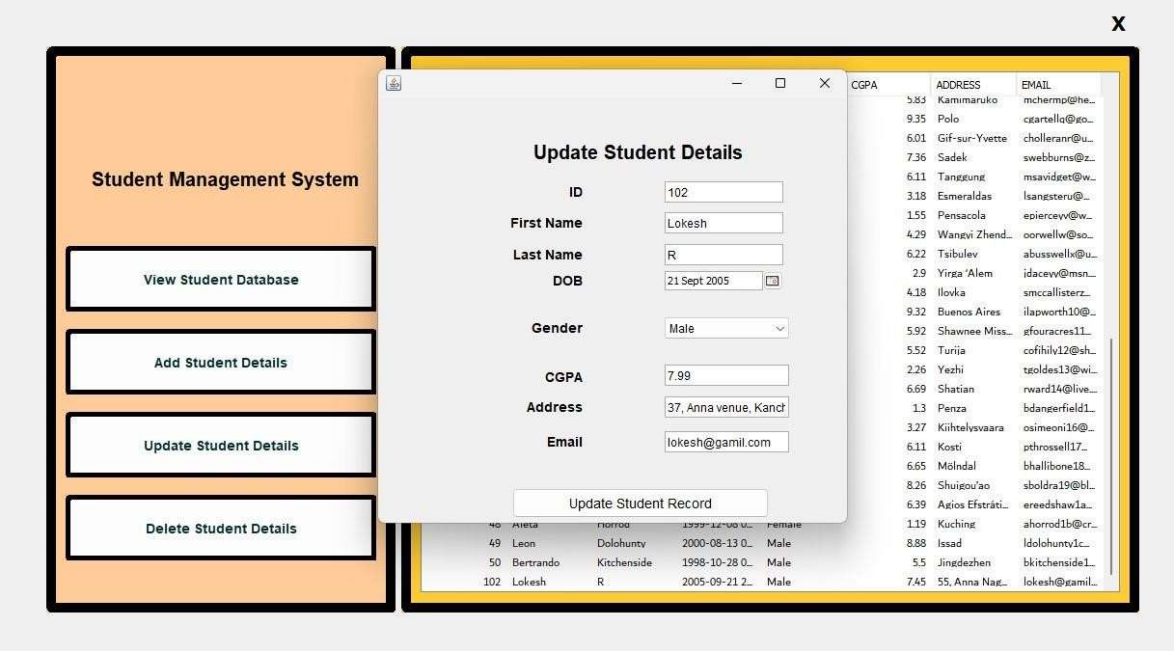


Fig 3.4 Updating

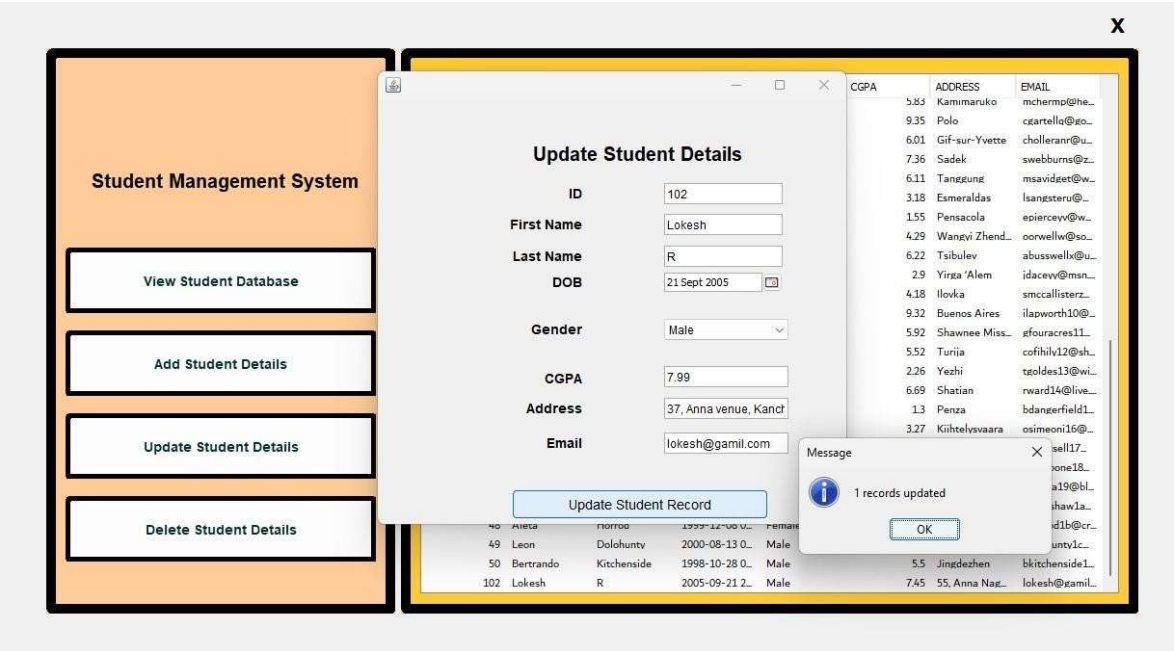


Fig 3.5 After Updating

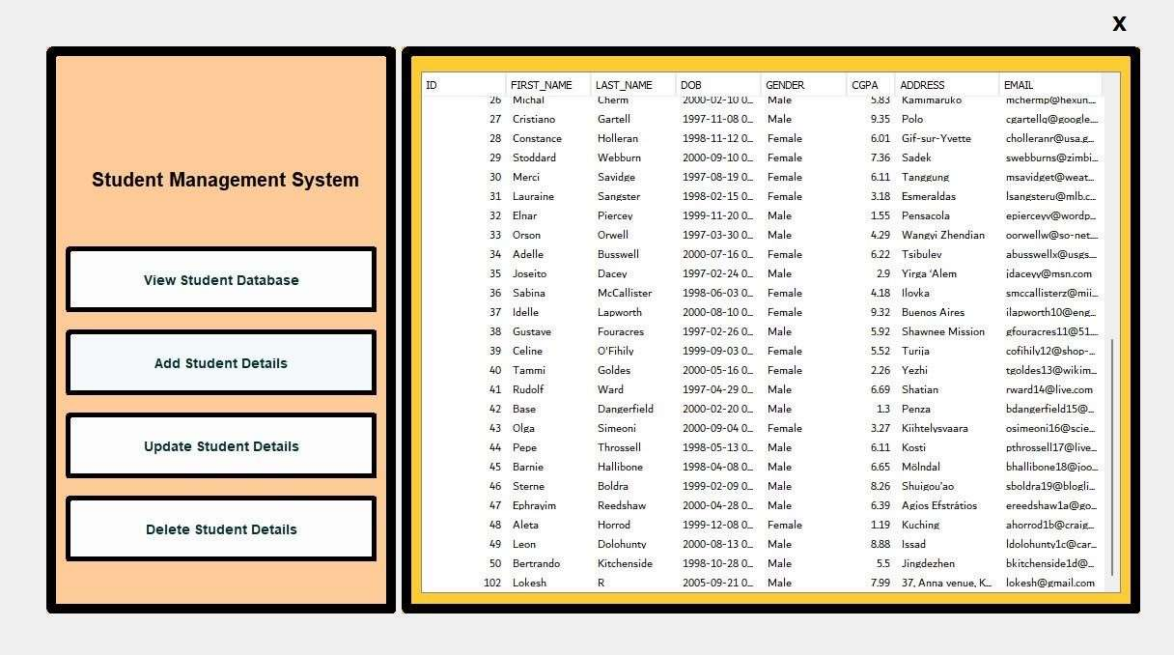


Fig 3.6 Updated

4. Delete Student Module

- **Purpose:** Removes outdated or invalid records.
- **Features:**
  - Users can search for a student by ID or name.
  - Confirmation dialog to prevent accidental deletions. o Deletion cascades across related tables if relationships exist.

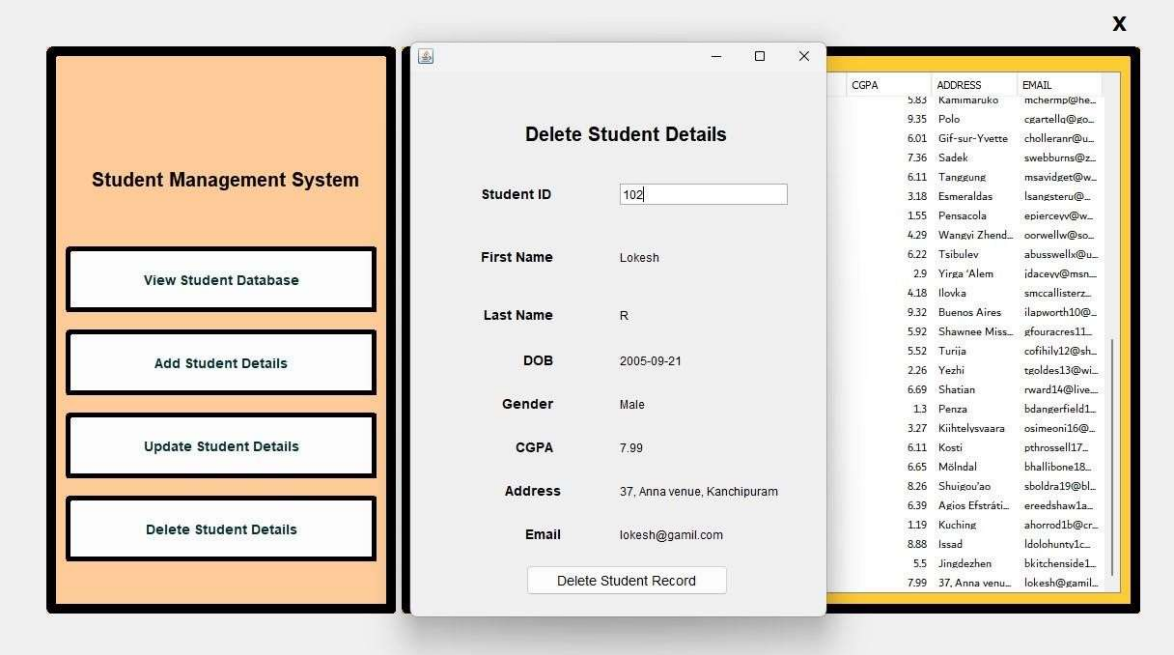


Fig 3.7 Delete data

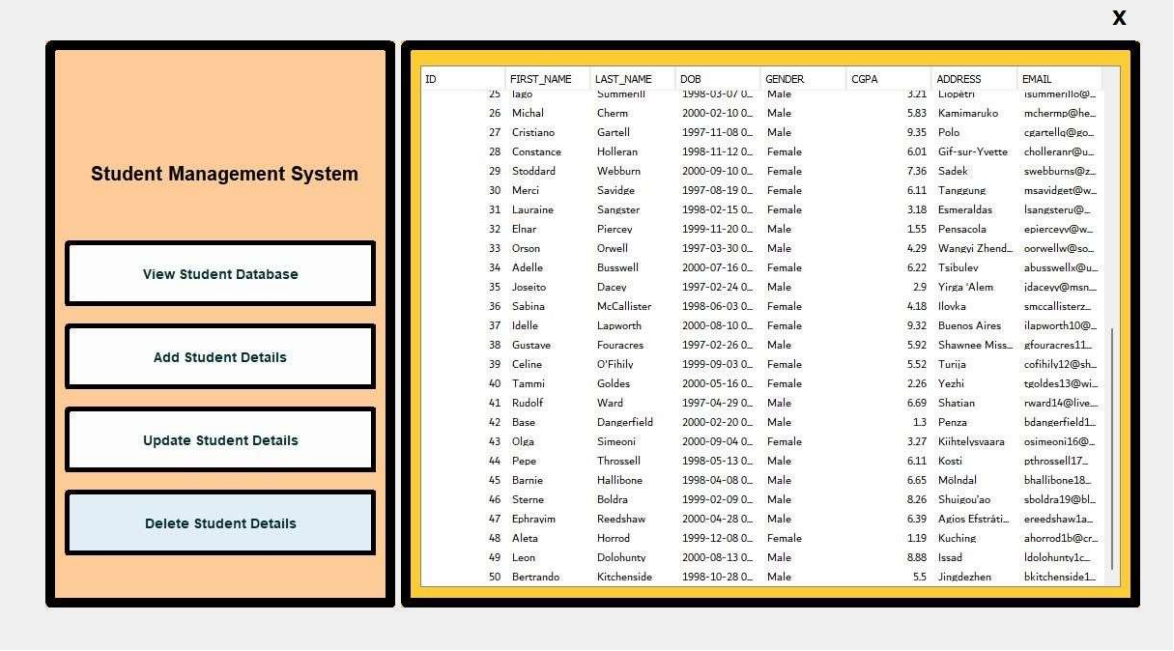


Fig 3.8 After deleting data

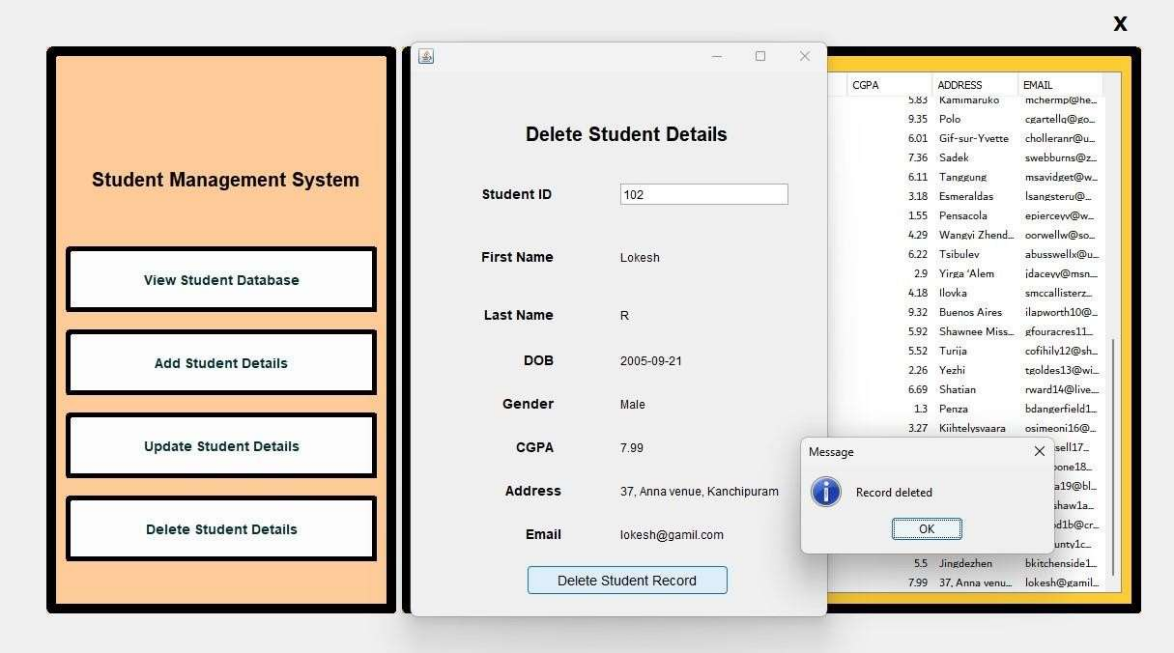


Fig 3.9 Deleted data

3.3 Results and Discussion

This section discusses the outcomes of the project and evaluates its effectiveness.

1. System Functionality

- The system successfully implements CRUD operations:
    - Adding, viewing, updating, and deleting student records. ○ All changes are instantly reflected in the

  Oracle database.

2. User Experience

- The Java Swing GUI provides a user-friendly interface:
    - Easy navigation and clear labeling of buttons. ○ Minimal errors due to validation mechanisms.

3. Data Security

- The use of Oracle Database ensures secure storage of data.
- Authentication mechanisms prevent unauthorized access to the system.

4. Performance

- The system performs efficiently with quick response times for CRUD operations.
- Scalable to handle a larger number of students as needed.

5. Challenges Encountered

- Database Connectivity Issues: Debugging JDBC configuration required additional time.
- Validation Complexity: Ensuring robust validation for all fields was challenging but resolved.
- UI Design: Designing a clean and intuitive interface using Java Swing involved iterative improvements.

6. Future Scope

- Integration with cloud-based databases for enhanced scalability.
- Addition of analytics and reporting tools for better decision-making.
- Integration of role-based access control (e.g., admin, teacher, student)

CHAPTER 4 CONCLUSION

4.1 Libraries Needed

This section describes the external libraries and tools integrated into the project:

1. **JCalendar Library:**
    - Purpose: Provides a graphical calendar widget for selecting dates (e.g., Date of Birth).

- Integration: The library is used in the student form to simplify and validate date input.

2. **Oracle JDBC Connector:**
    - Purpose: Facilitates the connection between the Java application and Oracle Database.
    - Functionality:
    - Executes SQL commands (e.g., INSERT, UPDATE, DELETE, SELECT) to interact with the database.
    - Ensures secure and efficient communication between the application and backend.
3. **Oracle Database:**
    - Purpose: Serves as the backend to store, retrieve, and manipulate student data.
    - Features:
    - Stores data in a structured manner.
    - Ensures data integrity and supports large-scale data handling.

Connectivity of Frontend and Backend in the Student Management System

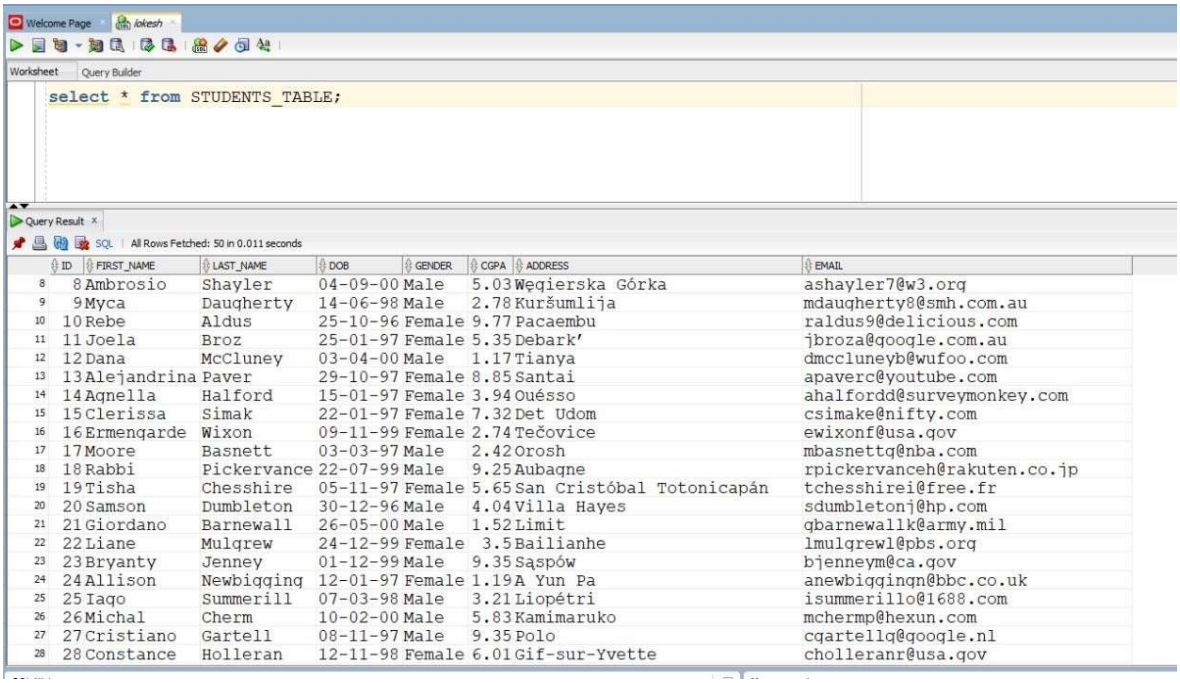
In this project, the connection between the frontend (Java Swing) and the backend (Oracle Database) is established using **Java Database Connectivity (JDBC)**. Here's a brief explanation of the process:

1. **Frontend (Java Swing)**:
    - The user interacts with the system through a graphical user interface (GUI) built using Java Swing.
    - Actions like adding, updating, deleting, or viewing student records trigger events in the frontend.
2. **Backend (Oracle Database)**:
    - The Oracle Database stores student details, such as ID, name, DOB, CGPA, etc.
    - All data operations (CRUD) are performed directly on the database.
3. **Connectivity via JDBC**:
    - **JDBC Driver**: The Oracle JDBC driver (ojdbc.jar) is used to establish a connection between the Java application and the Oracle Database.
    - **Connection Steps**:

- Load the JDBC driver using Class.forName("oracle.jdbc.driver.OracleDriver").
- Establish a connection using DriverManager.getConnection() with the database URL, username, and password.
- SQL queries (e.g., INSERT, UPDATE, DELETE, SELECT) are executed using PreparedStatement or Statement objects.
- Results are fetched from the database and displayed in the Swing GUI (e.g., in a JTable).

4. **Example Workflow**:
    - **Adding Data**:
    - The user fills out a form and clicks "Add."
    - The input is validated in the frontend.
    - A SQL INSERT query is sent via JDBC to the database to store the record.
    - **Retrieving Data**:
    - The system sends a SQL SELECT query to fetch records.
    - The results are displayed in a table on the GUI.

This integration ensures seamless communication between the user-facing application and the database, enabling efficient data management.



	ID	FIRST_NAME	LAST_NAME	DOB	GENDER	CGPA	ADDRESS	EMAIL
8	8	Ambrosio	Shayler	04-09-00	Male	5.03	Węgierska Górka	ashayler7@w3.org
9	9	Myca	Daugherty	14-06-98	Male	2.78	Kuršumlija	mdaugherty8@smh.com.au
10	10	Rebe	Aldus	25-10-96	Female	9.77	Pacaembu	raldus9@delicious.com
11	11	Joela	Broz	25-01-97	Female	5.35	Debark'	jbroza@google.com.au
12	12	Dana	McCluney	03-04-00	Male	1.17	Tianya	dmccluneyb@wufoo.com
13	13	Alejandrina	Paver	29-10-97	Female	8.85	Santai	apaverc@youtube.com
14	14	Agnella	Halford	15-01-97	Female	3.94	Ouésso	ahalfordd@surveymonkey.com
15	15	Clerissa	Simak	22-01-97	Female	7.32	Det Udom	csimake@nifty.com
16	16	Ermengarde	Wixon	09-11-99	Female	2.74	Tečovice	ewixonf@usa.gov
17	17	Moore	Basnett	03-03-97	Male	2.42	Orosh	mbasnettg@nba.com
18	18	Rabbi	Pickervance	22-07-99	Male	9.25	Aubagne	rpickervanceh@rakuten.co.jp
19	19	Tisha	Chesshire	05-11-97	Female	5.65	San Cristóbal Totonicapán	tchesshirei@free.fr
20	20	Samson	Dumbleton	30-12-96	Male	4.04	Villa Hayes	sdumbletonj@hp.com
21	21	Giordano	Barnewall	26-05-00	Male	1.52	Limit	gbarnewallk@army.mil
22	22	Liane	Mulgrew	24-12-99	Female	3.5	Bailianhe	lmulgrewl@pbs.org
23	23	Bryanty	Jenney	01-12-99	Male	9.35	Sąspów	bjenneym@ca.gov
24	24	Allison	Newbigging	12-01-97	Female	1.19	A Yun Pa	anewbiggingn@bbc.co.uk
25	25	Iago	Summerill	07-03-98	Male	3.21	Liopétri	isummerillo@1688.com
26	26	Michal	Cherm	10-02-00	Male	5.83	Kamimaruko	mchermp@hexun.com
27	27	Cristiano	Gartell	08-11-97	Male	9.35	Polo	cgartellq@google.nl
28	28	Constance	Holleran	12-11-98	Female	6.01	Gif-sur-Yvette	cholleranr@usa.gov

Fig 4.1 Oracle Database

Conclusion of Student Management System

The Student Management System is a robust application that simplifies the process of managing student data within educational institutions. Developed using Java Swing for the frontend and Oracle Database as the backend, the system successfully addresses the inefficiencies and challenges of traditional, manual processes for handling student records. This project integrates modern technologies to deliver a scalable, efficient, and user-friendly solution for student data management.

Key Accomplishments 1. **Streamlined Student**

Data Management:

The system enables administrators to efficiently manage student data, including adding, updating, deleting, and viewing records. By replacing traditional manual processes, the project ensures faster and more accurate data handling. 2. **Improved Data Security and Integrity:**

By using Oracle Database as the backend, the project ensures that student data is stored securely and remains consistent. The use of validations and error handling minimizes the chances of incorrect or duplicate entries.

3. **User-Friendly Interface:**
The graphical user interface built with Java Swing provides an intuitive and straightforward way for users to interact with the system. Features like drop-down menus, forms, and a tabular display of data enhance usability and accessibility. 4.

Efficient Data Connectivity:
The integration of the Oracle JDBC Connector ensures seamless communication between the frontend and backend. The use of prepared SQL queries makes the system both secure and efficient in performing CRUD operations.

5. **Extensibility:**
The modular design of the application lays the foundation for future enhancements. New features, such as advanced search, data export, or integration with online systems, can be easily added without disrupting the existing functionality.

Challenges Addressed

The project addresses several challenges associated with traditional methods of managing student records:

- **Time-Consuming Processes:** Manual data entry, updating, and retrieval are slow and prone to errors. The system automates these tasks, significantly reducing time and effort.
- **Data Inconsistency:** Centralized storage ensures that all data is consistent and updated in real-time.
- **Lack of Security:** Unlike paper-based or spreadsheet systems, this project offers improved security by storing data in a password-protected database.

- **Human Errors:** Real-time validation of inputs (e.g., ensuring a valid email format or CGPA range) minimizes errors during data entry

Benefits of the System

1. **Automation:** Automating CRUD operations reduces the workload on administrators, allowing them to focus on more critical tasks.
2. **Data Validation:** Validation mechanisms ensure that only correct and meaningful data is stored in the system, improving the quality of records.
3. **Accessibility:** Users can easily retrieve and manage student data through an organized and interactive interface.
4. **Reliability:** The use of Java and Oracle ensures that the application is both stable and capable of handling large amounts of data without performance issues.

Future Enhancements

While the current system achieves its primary objectives, there is scope for improvement and expansion:

1. Search and Filter Options: Adding advanced search and filtering capabilities would help administrators quickly find specific student records.
2. Export and Reporting: The system could generate reports in formats such as PDF or Excel for academic analysis or record-keeping.
3. Online Integration: Integrating the system with online portals or cloud-based services could provide access to data from remote locations.
4. Role-Based Access Control: Adding user roles (e.g., admin, teacher, viewer) would enhance security and limit access to sensitive data.
5. Mobile Application: A mobile version of the system could improve accessibility for administrators on the go.

Conclusion

In conclusion, the Student Management System demonstrates the power of integrating programming and database technologies to solve real-world problems. By automating and simplifying the management of student records, the project has significant implications for educational institutions, helping them to operate more efficiently and effectively.

The project's success lies in its ability to meet the objectives of data accuracy, security, and usability. It provides a centralized solution that streamlines administrative processes while ensuring that data is well-organized and secure. The use of Java Swing for the frontend and

Oracle Database for the backend demonstrates how robust and scalable applications can be built using these technologies.

Finally, this project serves as a foundation for future development and innovation. With enhancements like online integration, data analytics, and advanced reporting, the system can evolve into a comprehensive tool that caters to broader educational management needs. It is a testament to how technology can improve operational efficiency and drive progress in educational institutions.