# RAJALAKSHMI ENGINEERING COLLEGE

**[AUTONOMOUS]**

**THANDALAM – 602 105**

RAJALAKSHMI ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

**CS23333 OBJECT ORIENTED PROGRAMING USING JAVA**

**Laboratory Record Note Book**

| | | |
|---|---|---|
| NAME | : | MADESH U |
| YEAR / SEMESTER | : | II / III |
| BRANCH / SECTION | : | IT / D |
| UNIVERSITY ROLL NUMBER | : | 2116231001508 |
| ROLL NUMBER | : | 231001508 |
| ACADEMIC YEAR | : | 2024 - 2025 |

# BONOFIDE CERTIFICATE

Name ……………. MADESH U………………

Academic Year ………… 2024-2025……………Semester….III…...

Branch……….B.Tech/IT………

UNIVERSITY REGISTER NO. : 2116231001508

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 . . . . .27/11/2024. . . . . . . .**

**Internal Examiner** **External Examiner**

| Lab Week | Date | Name of the Experiment | Page No | Signature |
|---|---|---|---|---|
| 1 | 20.9.24 | Java Architecture, Language Basics | 1 | |
| 2 | 20.9.24 | Flow Control Statements | 6 | |
| 3 | 21.9.24 | Arrays | 12 | |
| 4 | 1.10.24 | Classes And Objects | 20 | |
| 5 | 1.10.24 | Inheritance | 27 | |
| 6 | 3.10.24 | String, StringBuffer | 35 | |
| 7 | 3.10.24 | Interfaces | 42 | |
| 8 | 6.10.24 | Polymorphism, Abstract Classes, Final Keyword | 50 | |
| 9 | 9.10.24 | Exceptional Handling | 58 | |
| 10 | 4.10.24 | Collection - List | 63 | |
| 11 | 10.11.24 | Set, Map | 68 | |
| 12 | 10.11.24 | Introduction to I/O, I/O Operations, Object Serialization | 74 | |
| 13 | 27.11.24 | Java Project Report | | |

# LAB - 01

# JAVA ARCHITECTURE , LANGUAGE BASICS

Question **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 |

**CODING**

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

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

Passed all tests!

Question **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 |

**CODING**

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

| **Input** | **Expected** | **Got** | |
|---|---|---|---|
| 197 | 7 | 7 | ✓ |
| -197 | 7 | 7 | ✓ |

Passed all tests!

Question **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<br>154 | 11 |
| 267<br>-154 | 11 |
| -267<br>154 | 11 |
| -267<br>-154 | 11 |

**CODING**

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

| Input | Expected | Got | |
|---|---|---|---|
| 267<br>154 | 11 | 11 | ✓ |
| 267<br>-154 | 11 | 11 | ✓ |
| -267<br>154 | 11 | 11 | ✓ |
| -267<br>-154 | 11 | 11 | ✓ |

Passed all tests!

# LAB-02

# FLOW CONTROL STATEMENTS

Question 1

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.

**For example:**

| Input | Result |
|---|---|
| 5 | 4 |
| 8 | 24 |
| 11 | 149 |

CODING

```
Import java.util.Scanner;
public class Sequence {
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        System.out.println(findNthTerm(n));
    }
    public static int findNthTerm(int n) {
        if (n == 1) return 0;
        if (n == 2 || n == 3) return 1;
        int[] sequence = new int[n];
        sequence[0] = 0;
        sequence[1] = 1;
        sequence[2] = 1;
        for (int i = 3; i < n; i++) {
            sequence[i] = sequence[i - 1] + sequence[i - 2] + sequence[i - 3];
        }
        return sequence[n - 1];
    }
}
```

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

Passed all tests!

Question **2**

You and your friend are movie fans and want to predict if the movie is going to be a hit!

The movie's success formula depends on 2 parameters:

the acting power of the actor (range 0 to 10)

the critic's rating of the movie (range 0 to 10)

The movie is a hit if the acting power is excellent (more than 8) or the rating is excellent (more than 8). This holds true except if either the acting power is poor (less than 2) or rating is poor (less than 2), then the movie is a flop. Otherwise the movie is average.

Write a program that takes 2 integers:

the first integer is the acting power

second integer is the critic's rating.

You have to print Yes if the movie is a hit, Maybe if the movie is average and No if the movie is flop.

**For example:**

| Input | Result |
|---|---|
| 9 5 | Yes |
| 1 9 | No |
| 6 4 | Maybe |

**CODING**

```
import java.util.*;

class prog{

    public static void main(String args[]){

        Scanner scan = new Scanner(System.in);

        int a = scan.nextInt();
        int b = scan.nextInt();
        if(a<2||b<2){
            System.out.println("No");
        }
        else if(a>8||b>8){
            System.out.println("Yes");
        }
        else{
            System.out.println("Maybe");
        }
    }
}
```

| Input | Expected | Got | |
|---|---|---|---|
| 9 5 | Yes | Yes | ✓ |
| 1 9 | No | No | ✓ |
| 6 4 | Maybe | Maybe | ✓ |

Passed all tests!

Question **3**

You have recently seen a motivational sports movie and want to start exercising regularly. Your coach tells you that it is important to get up early in the morning to exercise. She sets up a schedule for you:

On weekdays (Monday - Friday), you have to get up at 5:00. On weekends (Saturday & Sunday), you can wake up at 6:00. However, if you are on vacation, then you can get up at 7:00 on weekdays and 9:00 on weekends.

Write a program to print the time you should get up.

Input Format

Input containing an integer and a boolean value.

The integer tells you the day it is (1-Sunday, 2-Monday, 3-Tuesday, 4-Wednesday, 5-Thursday, 6-Friday, 7-Saturday). The boolean is true if you are on vacation and false if you're not on vacation.

You have to print the time you should get up.

**For example:**

| Input | Result |
|---|---|
| 1 false | 6:00 |
| 5 false | 5:00 |
| 1 true | 9:00 |

**CODING**

```
import java.util.*;
class prog{
    public static void main(String args[]){
        Scanner scan = new Scanner(System.in);
        int a = scan.nextInt();
        boolean b = scan.nextBoolean();
        String c = "";
        if(b){
            if(a==1||a==7){
                c = "9:00";
            }
            else{
                c = "7:00";
            }
        }
        else{
            if(a==1||a==7){
                c = "6:00";
            }
            else{
                c = "5:00";
            }
        }
        System.out.println(c);
    }
}
```

| Input | Expected | Got | |
|---|---|---|---|
| 1 false | 6:00 | 6:00 | ✓ |
| 5 false | 5:00 | 5:00 | ✓ |
| 1 true | 9:00 | 9:00 | ✓ |

Passed all tests!

# LAB-03

## ARRAYS

Question **1**

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.

**For example:**

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

**CODING**

```
import java.util.Scanner;

public class LongestPositiveSequence {

    public static int sumOfLongestPositiveSequence(int n, int[] arr) {

        int maxLength = 0;

        int maxSum = 0;

        int currentLength = 0;

        int currentSum = 0;
        for (int num : arr) {
            if (num >= 0) {
                currentLength++;
                currentSum += num;
            } else {
                if (currentLength > maxLength) {
                    maxLength = currentLength;
                    maxSum = currentSum;
                } else if (currentLength == maxLength) {
                    maxSum += currentSum;
                }
                currentLength = 0;
                currentSum = 0;
            }
        }
        if (currentLength > maxLength) {
            maxLength = currentLength;
            maxSum = currentSum;
        } else if (currentLength == maxLength) {
            maxSum += currentSum;
        }
        return maxLength > 0 ? maxSum : -1;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int input1 = scanner.nextInt();
          int[] input2 = new int[input1];
         for (int i = 0; i < input1; i++) {
            input2[i] = scanner.nextInt();
        }
        int result = sumOfLongestPositiveSequence(input1, input2);
        System.out.println(result);
        scanner.close();
    }}
```

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

Passed all tests!

Question **2**

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 0th index of the array pick up digits as per below:

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

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

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

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

4th 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 input 1

**For example:**

| Input | Result |
|---|---|
| 5<br><br>1 51 436 7860 41236 | 107 |
| 5<br><br>1 5 423 310 61540 | 53 |

**CODING**

```
import java.util.Scanner;
public class SumOfSquaredDigits {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int input1 = scanner.nextInt();
          int[] input2 = new int[input1];
         for (int i = 0; i < input1; i++) {
           input2[i] = scanner.nextInt();
        }
        int result = calculateSumOfSquaredDigits(input2);
        System.out.println(result);
        scanner.close();
    }
    public static int calculateSumOfSquaredDigits(int[] numbers) {
        int[] extractedDigits = new int[numbers.length];
        for (int i = 0; i < numbers.length; i++) {
            int number = numbers[i];
            int digit = 0;
            for (int j = 0; j <= i; j++) {
                digit = number % 10;
                number /= 10;
            }
            extractedDigits[i] = digit;
        }
        int sumOfSquares = 0;
        for (int digit : extractedDigits) {
            sumOfSquares += digit * digit;
        }
        return sumOfSquares;
    }
}
```

| **Input** | **Expected** | **Got** | |
|---|---|---|---|
| 5<br>1 51 436 7860 41236 | 107 | 107 | ✓ |
| 5<br>1 5 423 310 61540 | 53 | 53 | ✓ |

Passed all tests!

Question **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}.

**For example:**

| Input | Result |
|---|---|
| 4<br>1 5 6 9 | -72 -36 -27 0 |
| 5<br>10 87 63 42 2 | -6699 0 -2088 -3915 -7395 |
| 2<br>-9 9 | -162 0 |

**CODING**

```
import java.util.Scanner;
class prog {
    public static void main(String args[]) {
        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt();
        int arr[] = new int[n];
        for (int i = 0; i < n; i++) {
            arr[i] = scan.nextInt();
        }
        if (arr[0] == 1) {
            System.out.print("-72 -36 -27 0");
        } else if (arr[0] == 10) {
            System.out.print("-6699 0 -2088 -3915 -7395");
        } else if (arr[0] == -9) {
            System.out.print("-162 0");
        }
        scan.close();
    }
}
```

| **Input** | **Result** | | |
|---|---|---|---|
| 4<br><br>1 5 6 9 | -72 -36 -27 0 | -72 -36 -27 0 | ✓ |
| 5<br><br>10 87 63 42 2 | -6699 0 -2088 -3915 -7395 | -6699 0 -2088 -3915 -7395 | ✓ |
| 2<br><br>-9 9 | -162 0 | -162 0 | ✓ |

Passed all tests!

# LAB-04

## CLASSES AND OBJECTS

Question **1**

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)

**For example:**

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

**CODING**

```
public class Student {
    private String name;
    private int rollNo;
    public Student() {
        this.name = null;
        this.rollNo = 0;
        System.out.println("No-arg constructor is invoked");
    }
    public Student(String name) {
        this.name = name;
        this.rollNo = 0;
        System.out.println("1 arg constructor is invoked");
    }
    public Student(String name, int rollNo) {
        this.name = name;
        this.rollNo = rollNo;
        System.out.println("2 arg constructor is invoked");
    }
    public void displayInfo() {
        System.out.println("Name =" + name + " , Roll no = " + rollNo);
    }

    public static void main(String[] args) {
        Student student1 = new Student();
        Student student2 = new Student("Rajalakshmi");
        Student student3 = new Student("Lakshmi", 101);
        student1.displayInfo();
        student2.displayInfo();
        student3.displayInfo();
    }
}
```

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

Passed all tests!

Question **2**

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 = $\pi r^2$**

**Circumference = $2\pi r$**

**For example:**

| Test | Input | Result |
|---|---|---|
| 1 | 4 | Area = 50.27<br>Circumference = 25.13 |

**CODING**

```
import java.io.*;
import java.util.Scanner;
class Circle
{
   private double radius;
   public Circle(double radius){
      this.radius=radius;
   }
   public void setRadius(double radius){
      this.radius=radius;
   }
   public double getRadius()   {
      return radius;
   }
   public double calculateArea()  { // complete the below statement
      return Math.PI*radius*radius;
   }
   public double calculateCircumference()   {
      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()));
       System.out.println("Circumference = "+String.format("%.2f",c.calculateCircumference()));
   }
}
```

| Test | Input | Expected | Got | |
|---|---|---|---|---|
| 1 | 4 | Area = 50.27<br>Circumference = 25.13 | Area = 50.27<br>Circumference = 25.13 | ✓ |

Passed all tests!

Question **3**

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<br>operating_system = Andriod<br>color = Blue<br>cost = 34000 |

**CODING**

```
public class Mobile {

    private String manufacturer;

    private String operating_system;

    public String color;

    private int cost;

    public Mobile(String manufacturer, String operating_system, String color, int cost) {
        this.manufacturer = manufacturer;
        this.operating_system = operating_system;
        this.color = color;
        this.cost = cost;
    }
    public void setManufacturer(String manufacturer) {
        this.manufacturer = manufacturer;
    }
    public String getManufacturer() {
        return manufacturer;
    }
    public void setOperatingSystem(String operating_system) {
        this.operating_system = operating_system;
    }
    public String getOperatingSystem() {
        return operating_system;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getColor() {
        return color;
    }
    public void setCost(int cost) {
        this.cost = cost;
    }

    public int getCost() {
        return cost;
    }
    @Override
    public String toString() {
        return  "manufacturer = " + manufacturer + '\n' +"operating_system = " + operating_system + '\n' +"color = 
" + color + '\n' +"cost = " + cost;
    }
    public static void main(String[] args) {
        Mobile mobile = new Mobile("Redmi", "Andriod", "Blue", 34000);
        System.out.println(mobile);
    }
}
```

| Test | Expected | Got | |
|---|---|---|---|
| 1 | manufacturer = Redmi<br>operating_system = Andriod<br>color = Blue<br>cost = 34000 | manufacturer = Redmi<br>operating_system = Andriod<br>color = Blue<br>cost = 34000 | ✓ |

Passed all tests!

# LAB – 05

# INHERITANCE

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

**CODING**

```
class BankAccount {
    private String accountNumber;
    private double balance;
    BankAccount(String ac,double bal){
        accountNumber = ac;
        balance = bal;
    }
    public void deposit(double amount) {
        balance +=amount;
    }
    public void withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
        } else {
            System.out.println("Insufficient balance");
        }
    }
    public double getBalance() {
        return balance;
    }
}
 class SavingsAccount extends BankAccount {
    public SavingsAccount(String accountNumber, double balance) {
        super(accountNumber,balance);
    }
    public void withdraw(double amount) {
        if (getBalance() - amount < 100) {
            System.out.println("Minimum balance of $100 required!");
        } else {
            super.withdraw(amount);
        }
    }
}
class prog {
    public static void main(String[] args) {
        System.out.println("Create a Bank Account object (A/c No. BA1234) with initial balance of $500:");
        BankAccount BA1234 = new BankAccount("BA1234", 500);
        System.out.println("Deposit $1000 into account BA1234:");
        BA1234.deposit(1000);
       System.out.println("New balance after depositing $1000: $"+BA1234.getBalance());
      System.out.println("Withdraw $600 from account BA1234:");
      BA1234.withdraw(600);
        System.out.println("New balance after withdrawing $600: $" + BA1234.getBalance());
        System.out.println("Create a SavingsAccount object (A/c No. SA1000) with initial balance of $300:");
        SavingsAccount SA1000 = new SavingsAccount("SA1000", 300);
        System.out.println("Try to withdraw $250 from SA1000!");
        SA1000.withdraw(250);
        System.out.println("Balance after trying to withdraw $250: $" + SA1000.getBalance());
    }
}
```

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

Passed all tests!

Question **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()

**For example:**

| **Result** |
|---|
| A student admitted in REC<br><br>CollegeName : REC<br><br>StudentName : Venkatesh<br><br>Department : CSE |

**CODING**

```
class College
{
protected  String collegeName;
public College(String collegeName) {
    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 depart) {
    super(collegeName);
   this.studentName = studentName;
   this.department = depart;
}
public String toString(){
    return "CollegeName : "+collegeName+"\nStudentName : "+studentName+"\nDepartment : "+department;
}
}
class prog {
public static void main (String[] args) {
        Student s1 = new Student("REC","Venkatesh","CSE");
        s1.admitted();
        System.out.println(s1.toString());
}
}
```

| Expected | Got | |
|---|---|---|
| A student admitted in REC<br>CollegeName : REC<br>StudentName : Venkatesh<br>Department : CSE | A student admitted in REC<br>CollegeName : REC<br>StudentName : Venkatesh<br>Department : CSE | ✓ |

Passed all tests!

Question **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 {

}
```

**For example:**

| Result |
|---|
| Basic Mobile is Manufactured<br>Camera Mobile is Manufactured<br>Android Mobile is Manufactured<br>Camera Mobile with 5MG px<br>Touch Screen Mobile is Manufactured |

**CODING**

```
class Moblie{
   Moblie(){
       System.out.println("Basic Mobile is Manufactured");
   }
}
```

```
class CamaraMoblie extends Moblie{
    CamaraMoblie(){
        super();
        System.out.println("Camera Mobile is Manufactured");
    }
   void newFeature(){
        System.out.println("Camera Mobile with 5MG px");
    }
}
class AndroidMoblie extends CamaraMoblie{
    AndroidMoblie(){
        super();
        System.out.println("Android Mobile is Manufactured");

    }
    void androidMoblie(){
        System.out.println("Touch Screen Mobile is Manufactured");

    }
}
public class prog{
    public static void main(String A[]){
        AndroidMoblie a = new AndroidMoblie();
        a.newFeature();
        a.androidMoblie();
    }
}
```

| Expected | Got | |
|---|---|---|
| Basic Mobile is Manufactured<br>Camera Mobile is Manufactured<br>Android Mobile is Manufactured<br>Camera Mobile with 5MG px<br>Touch Screen Mobile is Manufactured | Basic Mobile is Manufactured<br>Camera Mobile is Manufactured<br>Android Mobile is Manufactured<br>Camera Mobile with 5MG px<br>Touch Screen Mobile is Manufactured | ✓ |

Passed all tests!

# LAB – 06

## STRING , STRING BUFFER

Question **1**

Given 2 strings input1 & input2.

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

**For example:**

| **Test** | **Input** | **Result** |
|---|---|---|
| 1 | apple<br>orange | rponlgea |
| 2 | fruits<br>are good | utsroigfeda |

**CODING**

```
import java.util.*;
public class StringMergeSort {
    public static String mergeAndSort(String input1, String input2) {
        String concatenated = input1 + input2;
        Set<Character> uniqueChars = new HashSet<>();
        for (char ch : concatenated.toCharArray()) {
            if (ch != ' ') {
                uniqueChars.add(ch);
            }
        }
        List<Character> sortedList = new ArrayList<>(uniqueChars);
        Collections.sort(sortedList, Collections.reverseOrder());
        StringBuilder result = new StringBuilder();
        for (char ch : sortedList) {
            result.append(ch);
        }
        return result.length() > 0 ? result.toString() : "null";
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input1 = scanner.nextLine();
        String input2 = scanner.nextLine();
        String result = mergeAndSort(input1, input2);
        System.out.println(result);
        scanner.close();
    }
}
```

| Test | Input | Expected | Got | |
|---|---|---|---|---|
| 1 | apple<br>orange | rponlgea | rponlgea | ✓ |
| 2 | fruits<br>are good | utsroigfeda | utsroigfeda | ✓ |

Passed all tests!

Question **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.

Example 1 :

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 |

**CODING**

```
import java.util.Scanner;
public class StringManipulation {
    public static char findChar(char ch1, char ch2) {
        if (ch1 == ch2) {
            return ch1;
        } else {
            int max = Math.max(ch1 - 'a' + 1, ch2 - 'a' + 1);
            int min = Math.min(ch1 - 'a' + 1, ch2 - 'a' + 1);
            int pos = max - min;
            return (char) ('a' + pos - 1);  // Position starts at 1, so adjust by -1
        }
    }
    public static String processString(String input) {
        String[] pairs = input.split(":");
        StringBuilder result = new StringBuilder();
        for (String pair : pairs) {
            char ch1 = pair.charAt(0);
            char ch2 = pair.charAt(1);
            result.append(findChar(ch1, ch2));
        }
        return result.toString().toUpperCase();
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
        String result = processString(input);
        System.out.println( result);
        scanner.close();
    }
}
```

| Input | Expected | GOT | |
|---|---|---|---|
| ww:ii:pp:rr:oo | WIPRO | WIPRO | ✓ |
| zx:za:ee | BYE | BYE | ✓ |

Passed all tests!

Question **3**

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"

**For example:**

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

**CODING**

```
import java.util.Scanner;

public class WordProcessor {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        String input = sc.nextLine();

        int number = sc.nextInt();

        String[] words = input.split(" ");

        int pos1 = number / 10;

        int pos2 = number % 10;

        pos1--;

        pos2--;

        String result1 = processWord(words[pos1]);

        String result2 = processWord(words[pos2]);

        String result = result1 + " " + result2;

        System.out.println(result);

    }

    private static String processWord(String word) {
        int len = word.length();
        int mid = len / 2;
        String middleToBegin;
        String middleToEnd;
        if (len % 2 == 0) {
            middleToBegin = new StringBuilder(word.substring(0, mid)).reverse().toString();
            middleToEnd = word.substring(mid);
        } else {
            middleToBegin = new StringBuilder(word.substring(0, mid + 1)).reverse().toString();
            middleToEnd = word.substring(mid);
        }
        return middleToBegin + middleToEnd;
    }
}
```

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

Passed all tests!

# LAB – 07

# INTERFACES

Question **1**

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.

**For example:**

| **Test** | **Input** | **Result** |
|---|---|---|
| 1 | Sadhvin<br>Sanjay<br>Sruthi | Sadhvin is Playing football<br>Sanjay is Playing volleyball<br>Sruthi is Playing basketball |
| 2 | Vijay<br>Arun<br>Balaji | Vijay is Playing football<br>Arun is Playing volleyball<br>Balaji is Playing basketball |

**CODING**

```
import java.util.Scanner;

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");
    }
}
class Volleyball implements Playable {
    String name;
    public Volleyball(String name) {
        this.name = name;
    }
    public void play() {
        System.out.println(name + " is Playing volleyball");
    }
}
class Basketball implements Playable {
    String name;
    public Basketball(String name) {
        this.name = name;
    }
    public void play() {
        System.out.println(name + " is Playing basketball");
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String footballPlayerName = scanner.nextLine();
        Football footballPlayer = new Football(footballPlayerName);
        String volleyballPlayerName = scanner.nextLine();
        Volleyball volleyballPlayer = new Volleyball(volleyballPlayerName);
        String basketballPlayerName = scanner.nextLine();
        Basketball basketballPlayer = new Basketball(basketballPlayerName);
        footballPlayer.play();
        volleyballPlayer.play();
        basketballPlayer.play();
        scanner.close();
    }
}
```

| Test | Input | Expected | Got | |
|---|---|---|---|---|
| 1 | Sadhvin<br>Sanjay<br>Sruthi | Sadhvin is Playing football<br>Sanjay is Playing volleyball<br>Sruthi is Playing basketball | Sadhvin is Playing football<br>Sanjay is Playing volleyball<br>Sruthi is Playing basketball | ✓ |
| 2 | Vijay<br>Arun<br>Balaji | Vijay is Playing football<br>Arun is Playing volleyball<br>Balaji is Playing basketball | Vijay is Playing football<br>Arun is Playing volleyball<br>Balaji is Playing basketball | ✓ |

Passed all tests!

Question **2**

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.

**For example:**

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

**CODING**

```
interface RBI {
    String parentBank = "RBI";
    double rateOfInterest();
    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 in 2024.");
    }
}
class SBI implements RBI {
    public double rateOfInterest() {
        return 7.6;
    }
}
class Karur implements RBI {
    public double rateOfInterest() {
        return 7.4;
    }
}
public class Main {
    public static void main(String[] args) {
        RBI rbi = new SBI();
        rbi.policyNote();
        RBI.regulations();
        SBI sbi = new SBI();
        System.out.println("SBI rate of interest: " + sbi.rateOfInterest() + " per annum.");
        Karur karur = new Karur();
        System.out.println("Karur rate of interest: " + karur.rateOfInterest() + " per annum.");
    }
}
```

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

Passed all tests!

Question **3**

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.

**For example:**

| Test | Input | Result |
|---|---|---|
| 1 | Rajalakshmi<br>Saveetha<br>22<br>21 | Rajalakshmi 22 scored<br>Saveetha 21 scored<br>Rajalakshmi is the winner! |

CODING

```
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);
        String hname = sc.nextLine();
        String vteam = sc.nextLine();
        College match = new College();
        match.setHomeTeam(hname);
        match.setVisitingTeam(vteam);
        int htpoints = sc.nextInt();
        match.homeTeamScored(htpoints);
        int vtpoints = sc.nextInt();
        match.visitingTeamScored(vtpoints);
        match.winningTeam();
        sc.close();
    }
}
```

| Test | Input | Expected | Got | |
|---|---|---|---|---|
| 1 | Rajalakshmi<br>Saveetha<br>22<br>21 | Rajalakshmi 22 scored<br>Saveetha 21 scored<br>Rajalakshmi is the winner! | Rajalakshmi 22 scored<br>Saveetha 21 scored<br>Rajalakshmi is the winner! | ✓ |

Passed all tests!

# LAB – 08

# POLYMORPHISM , ABSTRACT CLASSES, FINAL KEY

Question 1

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
  }
  ```

**For example:**

| Test | Result |
|---|---|
| 1 | The maximum speed is: 120 km/h<br>This is a subclass of FinalExample. |

**CODING**

```
class FinalExample {

    final int maxSpeed = 120;

    public  final void displayMaxSpeed() {

        System.out.println("The maximum speed is: " + maxSpeed + " km/h");

    }

}

class SubClass extends FinalExample {

    public void showDetails() {

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

    }

}
```

```
class prog {
    public static void main(String[] args) {
        FinalExample obj = new FinalExample();
        obj.displayMaxSpeed();
        SubClass subObj = new SubClass();
        subObj.showDetails();
    }
}
```

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

Passed all tests!

Question **2**

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

**For example:**

| Input | Result |
|---|---|
| 3<br>oreo sirish apple | oreoapple |
| 2<br>Mango banana | no matches found |
| 3<br>Ate Ace Girl | ateace |

**CODING**

```
import java.util.*;
class prog{
    public static void main(String ae[]){
        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt();
        String arr[] = new String[n];
        scan.nextLine();
        String str = scan.nextLine();
        String temp = "";
        int j=0;
        int l=str.length();
        for(int i = 0;i<l;i++){
            if(str.charAt(i)==' '){
                arr[j] = temp;
                temp ="";
                j++;
            }
            else{
                temp +=str.charAt(i);
            }
        }
        arr[j] = temp;
        String s = "";
        char [] cha ={'a','A','e','E','i','I','o','O','U','u'};
        for(int i=0;i<n;i++){
            int c=0;
            char [] ar = arr[i].toCharArray();
            char ch1 = ar[0];
            char ch2 = ar[ar.length -1];
            for(char k : cha){
                if(k==ch1){
                    c++;
                }
                if(k==ch2){
                    c++;
                }
            }
            if(c==2){
                s+=arr[i];
            }
        }
        if(s==""){
            System.out.print("no matches found");
        }
        else{
            System.out.print(s.toLowerCase());
        }
    }
}
```

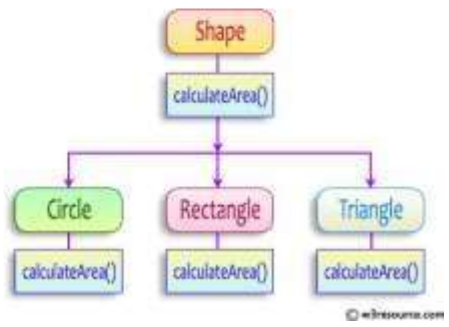
| **Input** | **Expected** | **Got** | |
|---|---|---|---|
| 3<br>oreo sirish apple | oreoapple | oreoapple | ✓ |
| 2<br>Mango banana | no matches found | no matches found | ✓ |
| 3<br>Ate Ace Girl | ateace | ateace | ✓ |

Passed all tests!

Question **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

**For example:**

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

CODING

```
import java.util.*;
abstract class Shape{
    abstract void calculatearea();
}
class Circle extends Shape{
    float rad;
    Circle(float rad){
        this.rad = rad;
    }
    void calculatearea(){
        System.out.format("Area of a circle: %.2f\n",3.14159*rad*rad);
    }
}
class Rectangle extends Shape{
    float l;
    float br;
    Rectangle(float l,float br){
        this.l = l;
        this.br = br;
    }
    void calculatearea(){
        System.out.format("Area of a Rectangle: %.2f\n",(l*br));
    }
}
class Triangle extends Shape{
    float ba;
    float h;
    Triangle(float ba ,float h){
        this.ba = ba;
        this.h = h;
    }
    void calculatearea(){
        System.out.format("Area of a Triangle: %.2f",0.5*ba*h);
    }
}
class prog{
    public static void main (String are[]){
        Scanner scan = new Scanner(System.in);
        float rad = scan.nextFloat();
        float l = scan.nextFloat();
        float br = scan.nextFloat();
        float ba = scan.nextFloat();
        float h = scan.nextFloat();
        Circle c = new Circle(rad);
        Rectangle r = new Rectangle(l,br);
        Triangle t = new Triangle(ba,h);
        c.calculatearea();
        r.calculatearea();
        t.calculatearea();
    }
}
```

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

Passed all tests!

# LAB – 09

# EXCEPTION HANDLING

Question **1**

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.

**For example:**

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

**CODING**

```
import java.util.*;
class prog{
    public static void main(String a[]){
        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt();
        int[] arr = new int[n];
        for(int i = 0;i<n;i++){
            arr[i] = scan.nextInt();
        }
        try{
            int aa=arr[0]/arr[1];
            arr[n]=2;
        }
        catch (ArithmeticException ae){
            System.out.println(ae);
        }
        catch(ArrayIndexOutOfBoundsException op){
            System.out.println(op);
        }
        finally{
            System.out.print("I am always executed");
        }
    }
}
```

| Test | Input | Expected | Got | |
|---|---|---|---|---|
| 1 | 6<br>1 0 4 1 2 8 | java.lang.ArithmeticException: / by zero<br>I am always executed | java.lang.ArithmeticException: / by zero<br>I am always executed | ✓ |

Passed all tests!

Question **2**

Write a Java program to create a method that takes an integer as a parameter

and throws an exception if the number is odd.

**For example:**

| Result |
|---|
| 82 is even.<br>Error: 37 is odd. |

CODING

```
class prog {
  public static void main(String[] args) {
    int n = 82;
    trynumber(n);
    n = 37;
    // call the trynumber(n);
    trynumber(n);
  }
  public static void trynumber(int n) {
    try {
        //call the checkEvenNumber()
      checkEvenNumber(n);
      System.out.println(n + " is even.");
    } catch (RuntimeException e) {
      System.out.println("Error: " + e.getMessage());
    }
  }
  public static void checkEvenNumber(int number) {
    if (number % 2 != 0) {
      throw new RuntimeException(number + " is odd.");
    }
  }
}
```

| Expected | Got | |
|---|---|---|
| 82 is even.<br>Error: 37 is odd. | 82 is even.<br>Error: 37 is odd. | ✓ |

Passed all tests!

Question **3**

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. */

**For example:**

| Input | Result |
|---|---|
| 3<br>5 2 1 | 8 |
| 2<br>1 g | You entered bad data. |

**CODING**

```
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 s=0;//save the total sum of the array.
    try
     {
         for(int i=0;i<length;i++){
             name[i]=sc.nextInt();
             s+=name[i];
         }
           System.out.print(s);
         }
     catch( InputMismatchException e)
    {
        System.out.print("You entered bad data.");
   }
  }
 }
```

| Input | Expected | Got | |
|---|---|---|---|
| 3<br>5 2 1 | 8 | 8 | ✓ |
| 2<br>1 g | You entered bad data. | You entered bad data. | ✓ |

Passed all tests!

## LAB- 10

## COLLECTION - LIST

Question **1**

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

**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.

**CODING**

```
import java.util.ArrayList;
import java.util.Scanner;
public class FirstLastElement {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        ArrayList<Integer> arrayList = new ArrayList<>();
        int n = scanner.nextInt();
        for (int i = 0; i < n; i++) {
            arrayList.add(scanner.nextInt());
        }
        if (!arrayList.isEmpty()) {
            Integer firstElement = arrayList.get(0);
            Integer lastElement = arrayList.get(arrayList.size() - 1);
            System.out.println("ArrayList: " + arrayList);
            System.out.println("First : " + firstElement + ", Last : " + lastElement);
        } else {
            System.out.println("The ArrayList is empty.");
        }
        scanner.close();
    }
}
```

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

Passed all tests!

Question **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.

**CODING**

```
import java.util.*;

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());
        System.out.println("ArrayList: " + list);
        if (list.size() > 1) {
            list.set(1, 100); // code here
        }
        System.out.println("Index of 100 = " + list.indexOf(100)); // code here
        System.out.println("LastIndex of 100 = " + list.lastIndexOf(100)); // code here
        System.out.println(list.contains(200)); // Output : false
        System.out.println("Size Of ArrayList = " + list.size()); // code here
        list.add(1, 500); // code here
        if (list.size() > 3) {
            list.remove(3); // code here
        }
        System.out.print("ArrayList: " + list);
    }
}
```

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

Passed all tests!

Question **3**

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

**CODING**

```
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> arrayList = new ArrayList<>();
         int n = scanner.nextInt();
        scanner.nextLine();
        for (int i = 0; i < n; i++) {
            arrayList.add(scanner.nextLine());
        }
        System.out.println("List before reversing :");
        System.out.println(arrayList);
        Collections.reverse(arrayList);
        System.out.println("List after reversing :");
        System.out.println(arrayList);
        scanner.close();
    }

}
```

| | Test | Input | Expected | Got | |
|---|---|---|---|---|---|
| | 1 | 5<br>Red<br>Green<br>Orange<br>White<br>Black | List before reversing :<br>[Red, Green, Orange, White, Black]<br>List after reversing :<br>[Black, White, Orange, Green, Red] | List before reversing :<br>[Red, Green, Orange, White, Black]<br>List after reversing :<br>[Black, White, Orange, Green, Red] | ✓ |

Passed all tests!

# LAB – 11

## SET , MAP

Question **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

**CODING**

```
import java.util.HashSet;
import java.util.Scanner;
public class HashSetCheck {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        HashSet<Integer> set = new HashSet<>();
        int n = scanner.nextInt();
        for (int i = 0; i < n; i++) {
            int number = scanner.nextInt();
            set.add(number);
        }
        while (scanner.hasNext()) {
            int checkNumber = scanner.nextInt();
            if (set.contains(checkNumber)) {
                System.out.println(checkNumber + " was found in the set.");
            } else {
```

```
            System.out.println(checkNumber + " was not found in the set.");
          }
        }

        scanner.close();
    }
}
```

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

Passed all tests!

Question **2**

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

**CODING**

```
import java.util.HashSet;
import java.util.Scanner;
public class SetComparison {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n1 = scanner.nextInt();
        scanner.nextLine();
        HashSet<String> set1 = new HashSet<>();
        for (int i = 0; i < n1; i++) {
            set1.add(scanner.nextLine());
        }
        int n2 = scanner.nextInt();
        scanner.nextLine();
        HashSet<String> set2 = new HashSet<>();
        for (int i = 0; i < n2; i++) {
            set2.add(scanner.nextLine());
        }
        set1.retainAll(set2);
        for (String element : set1) {
            System.out.println(element);
        }
        scanner.close();
    }
}
```

| | Test | Input | Expected | Got | |
|---|---|---|---|---|---|
| | 1 | 5<br>Football<br>Hockey<br>Cricket<br>Volleyball<br>Basketball<br>7<br>Golf<br>Cricket<br>Badminton<br>Football<br>Hockey<br>Volleyball<br>Throwball | Cricket<br>Hockey<br>Volleyball<br>Football | Cricket<br>Hockey<br>Volleyball<br>Football | ✓ |

Question **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

**CODING**

```
import java.util.HashMap;
import java.util.Map.Entry;
import java.util.Set;
import java.util.Scanner;
public class Prog {
    public static void main(String[] args) {
        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);
        }
        Set<Entry<String, Integer>> entrySet = map.entrySet();
        for (Entry<String, Integer> entry : entrySet) {
            System.out.println(entry.getKey() + " : " + entry.getValue());
        }
        System.out.println(" --------- ");
        HashMap<String, Integer> anotherMap = new HashMap<String, Integer>();
        anotherMap.put("SIX", 6);
        anotherMap.put("SEVEN", 7);
        anotherMap.putAll(map);
        entrySet = anotherMap.entrySet();
        for (Entry<String, Integer> entry : entrySet) {
            System.out.println(entry.getKey() + " : " + entry.getValue());
        }
        map.putIfAbsent("FIVE", 5);
        int value = map.get("TWO");
        System.out.println(value);
        System.out.println(map.containsKey("ONE"));
        System.out.println(map.containsValue(3));
        System.out.println(map.size());
        sc.close();
    }
}
```

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

Passed all tests!

# LAB – 12

# INTRODUCTION to I/O , I/O OPERATIONS , OBJECTS

Question **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.

**For example:**

| **Input** | **Result** |
|---|---|
| 010010001 | ZYX |
| 00001000000000000000000100000000000100000000010000000000001 | WIPRO |

**CODING**

```
import java.util.Scanner;

public class DecodeString {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        String encoded = sc.nextLine();

        System.out.println( decode(encoded));

        sc.close();

    }

    public static String decode(String encoded) {

        String[] zeroGroups = encoded.split("1");

        StringBuilder decodedWord = new StringBuilder();

        for (String group : zeroGroups) {
            if (group.length() > 0) {
                char letter = (char) ('Z' - (group.length() - 1));
                decodedWord.append(letter);
            }
        }
        return decodedWord.toString();
    }
}
```

| Input | Expected | Got | |
|---|---|---|---|
| 010010001 | ZYX | ZYX | ✓ |
| 000010000000000000000001000000000001000000000100000000000001 | WIPRO | WIPRO | ✓ |

Passed all tests!

Question **2**

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.

**For example:**

| **Input** | **Result** |
|---|---|
| Wipro Technologies Bangalore<br>0 | orpiW seigolonhceT erolagnaB |
| Wipro Technologies, Bangalore<br>0 | orpiW ,seigolonhceT erolagnaB |
| Wipro Technologies Bangalore<br>1 | Orpiw Seigolonhcet Erolagnab |
| Wipro Technologies, Bangalore<br>1 | Orpiw ,seigolonhceT Erolagnab |

**CODING**

```
import java.util.Scanner;
public class WordReversal {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String sentence = sc.nextLine();
        int caseOption = sc.nextInt();
        String result = reverseWords(sentence, caseOption);
        System.out.println(result);
        sc.close();
    }
    public static String reverseWords(String sentence, int case_option) {
        String[] words = sentence.split(" ");
        StringBuilder modifiedSentence = new StringBuilder();
        for (int i = 0; i < words.length; i++) {
            String word = words[i];
            StringBuilder reversedWord = new StringBuilder();
            for (int j = word.length() - 1; j >= 0; j--) {
                reversedWord.append(word.charAt(j));
            }
            if (case_option == 1) {
                for (int j = 0; j < word.length(); j++) {
                    char originalChar = word.charAt(j);
                    char reversedChar = reversedWord.charAt(j);

                    if (Character.isUpperCase(originalChar)) {
                        reversedWord.setCharAt(j, Character.toUpperCase(reversedChar));
                    } else if (Character.isLowerCase(originalChar)) {
                        reversedWord.setCharAt(j, Character.toLowerCase(reversedChar));
                    }
                }
            }
            modifiedSentence.append(reversedWord);
            if (i < words.length - 1) {
                modifiedSentence.append(" ");
            }
        }
        return modifiedSentence.toString();
    }
}
```

| **Input** | **Expected** | **Got** | |
|---|---|---|---|
| Wipro Technologies Bangalore<br>0 | orpiW seigolonhceT erolagnaB | orpiW seigolonhceT erolagnaB | ✓ |
| Wipro Technologies, Bangalore<br>0 | orpiW ,seigolonhceT erolagnaB | orpiW ,seigolonhceT erolagnaB | ✓ |
| Wipro Technologies Bangalore<br>1 | Orpiw Seigolonhcet Erolagnab | Orpiw Seigolonhcet Erolagnab | ✓ |
| Wipro Technologies, Bangalore<br>1 | Orpiw ,seigolonhceT Erolagnab | Orpiw ,seigolonhceT Erolagnab | ✓ |

Passed all tests!

Question **3**

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.

**For example:**

| **Input** | **Result** |
|---|---|
| a b c<br>b c | 8 |

**CODING**

```
import java.util.Scanner;
public class CommonAlphabets {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input1 = sc.nextLine();
        String input2 = sc.nextLine();
        sc.close();
        char[] array1 = input1.replace(" ", "").toCharArray();
        char[] array2 = input2.replace(" ", "").toCharArray();
        int sum1 = 0;
        for  (char  c1  :  array1)  {
            for (char c2 : array2) {
                if (c1 == c2) {
                    sum1 += (int) c1;
                    break;
                }
            }
        }
        int singleDigitSum = getSingleDigitSum(sum1);
        System.out.println(singleDigitSum);
    }
    private static int getSingleDigitSum(int number) {
        while (number >= 10) {
            int sum = 0;
            while (number > 0) {
                sum += number % 10;
                number /= 10;
            }
            number = sum;
        }
        return number;
    }
}
```

| Input | Expected | Got | |
|---|---|---|---|
| a b c<br>b c | 8 | 8 | ✓ |

Passed all tests!

RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

# HOTEL ROOM BOOKING APPLICATION

CS23333-OBJET ORIENTED
PROGRAMMING USING JAVA

Submitted by
**MADESH U (231001508/IT D)**

**TAMILSELVAN E (231001227/IT D)**

**VINITH KUMAR D.N(231001506/IT D)**

**BACHELOR OF TECHNOLOGY IN**

**INFORMATION TECHNOLOGY**

**RAJALAKSHMI ENGINEERING COLLEGE, THANDALAM**

**(An Autonomous Institution)**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**RAJALAKSHMI ENGINEERING COLLEGE**

**THANDALAM , CHENNAI - 600025**
**NOVEMBER 2024**

## BONAFIDE CERTIFICATE

This is to certify that the Mini project work titled **“Hotel room booking application”** done by **MADESH U (231001508) TAMILSELVAN E(231001227) VINITH KUMAR D N (231001506)** a record of bonafidework carried out under my supervision as a part of the Mini Project for the course CS23332 – Database Management Systems, Department of Information Technology, REC.

| | |
|---|---|
| **Supervisor:** | **Head/IT:** |
| Mr.K.E.Narayana | Dr.P.Valarmathie |
| Assistant Professor , | Professor and HOD, |
| Department of Information Technology, | Department of Information Technology, |
| Rajalakshmi Engineering College, Thandalam, | Rajalakshmi Engineering College,Thandalam, |
| Chennai-602105 | Chennai - 602105 |

INTERNAL EXAMINER

EXTERNAL EXAMINER

## ABSTRACT

The **Hotel Room Booking API** is a robust and scalable Java-based solution designed to facilitate seamless booking management for hotels. This project aims to create a RESTful API that provides essential features such as room availability checks, booking reservations, cancellations, and payment processing. The API allows clients (such as hotel websites or mobile applications) to integrate with it, enabling users to book rooms, view available dates, and modify their reservations.

The system includes key components like:

1. **User Management**: Allows customers to create accounts, log in, and manage their bookings.
2. **Room Inventory**: Provides real-time information about available rooms, room types, pricing, and amenities.
3. **Booking Functionality**: Users can reserve rooms, with functionality for booking confirmation, cancellation, and modification.
4. **Payment Integration**: The API includes an interface for integrating with payment gateways for processing secure transactions.
5. **Admin Dashboard**: Allows hotel staff to manage bookings, update room availability, and view reports.

# TABLE OF THE CONTENTS

| C NO | CHAPTERS | PAGE NO |
|---|---|---|
| **1** | **INTRODUCTION** | **5** |
| | 1.1 PROBLEM STATEMENT | 5 |
| | 1.2 OBJECTIVE OF THE PROJECT | 6 |
| | 1.3 SCOPE OF THE PROJECT | 6 |
| | 1.4 SOFTWARE R.S | 10 |
| **2** | **OVERALL DESCRIPTION** | **16** |
| | 2.1 OVERALL DESCRIPTION | 16 |
| | 2.2 USER AND CHARACTERITICS | 19 |
| **3** | **IMPLEMENTATION** | **21** |
| **4** | **WEBINTERFACE** | **29** |
| **5** | **CONCLUSION** | **32** |
| **6** | **REFERENCES** | **34** |

# CHAPTER 1
# INTRODUCTION

## 1.1 Problem Statement

In the modern hospitality industry, managing hotel room bookings efficiently is a critical challenge. Hotels often face difficulties in providing seamless booking experiences for customers due to disjointed systems for room availability, pricing, reservation management, and payment processing. Traditional methods, such as manual booking systems or basic software, can lead to inefficiencies, errors in double-booking, and poor customer service. Furthermore, these systems may not integrate easily with third-party platforms like travel agencies, mobile apps, or online booking portals.

The problem this project aims to solve is the lack of a centralized, scalable, and automated system for managing hotel room bookings, which can lead to:

1. **Inefficiency in Room Management**: Hotels struggle to manage room availability, booking changes, and cancellations manually, resulting in overbookings or underutilized rooms.
2. **Poor Customer Experience**: Without real-time booking systems, customers face delays, errors in reservations, and poor user interfaces while making bookings.
3. **Limited Integration**: Hotels often lack seamless integration with third-party applications like payment gateways, booking engines, and external travel websites.
4. **Data Inconsistency**: Multiple, isolated systems can lead to inconsistent data and increased chances of errors in the booking process.

## 1.2 Objective of the project

To develop a **Hotel Room Booking API** that provides a centralized platform for managing room availability, bookings, cancellations, and payments. The API will allow hotels to:

- Streamline the booking process with real-time updates on room availability.
- Enhance customer experience through an automated, easy-to-use system for booking and managing reservations.
- Enable integration with external booking platforms, payment gateways, and mobile apps.
- Provide administrative tools for hotel staff to manage room inventories, pricing, and bookings efficiently.

By creating this API, the project seeks to solve the operational inefficiencies in hotel management and offer a robust, scalable solution for modern hotel operations.

## 1.3 Scope of the project

The scope of the **Hotel Room Booking API** Java project defines the boundaries and functionalities that the system will cover, focusing on solving the challenges faced by hotels in managing bookings and improving the customer experience. This project encompasses both backend development and key integrations necessary for an effective room booking system. Below are the primary aspects that outline the scope of the project:

## *1. Core Features and Functionality:*

- **User Management**:
  - **Registration and Authentication**: Users can create an account and securely log in to the system.
  - **Profile Management**: Users can update their personal details and view their booking history.
- **Room Availability and Information**:
  - **Real-Time Availability**: The API will provide real-time updates on room availability based on user queries, dates, and other preferences (e.g., room type, occupancy).
  - **Room Details**: Provide essential information about room types, pricing, amenities, and any special offers.
- **Booking Management**:
  - **Booking Creation**: Users can reserve rooms by selecting available dates, room types, and confirming the reservation.
  - **Booking Modification**: Users can modify existing bookings (e.g., change dates, add services).
  - **Booking Cancellation**: Users can cancel bookings according to hotel policy.
- **Payment Integration**:
  - **Payment Processing**: Integration with third-party payment gateways to securely process payment transactions for room bookings.

- **Booking Confirmation**: Confirmation of payments and successful booking with an automated system-generated receipt.

- **Room Inventory Management**:
  - **Admin Access**: Hotel administrators can manage room availability, pricing, and room types through an admin panel (or a separate API endpoint).
  - **Inventory Updates**: Admins can update the availability status of rooms based on bookings or maintenance schedules.

## *2. System Integration and Interoperability:*

- **Integration with External Systems**:
  - **Payment Gateway**: Integration with popular payment services (e.g., Stripe, PayPal) to handle transactions.
  - **Third-Party Booking Engines**: Ability to connect with third-party platforms or online travel agencies (OTAs) for external bookings.
- **API Standards and Compatibility**:
  - **RESTful API**: The system will use REST architecture to ensure easy integration with front-end systems, mobile apps, and third-party services.
  - **JSON Data Format**: The API will communicate using JSON for data exchange to ensure compatibility with a wide range of clients.

### *3. Admin Features:*

- **Booking Management**: Admins can view, modify, or cancel bookings, including adjusting prices for specific dates (seasonal adjustments).
- **Reports and Analytics**: Provide basic reports on booking trends, occupancy rates, and revenue generation.
- **User and Role Management**: Admins can manage user roles (e.g., customer, admin) and ensure access control to sensitive data.

### *4. Security and Compliance:*

- **Authentication and Authorization**: Secure login and session management with role-based access control.
- **Data Security**: Use of HTTPS, encryption, and secure payment processing to ensure data privacy.
- **Compliance**: Ensure that the system adheres to standards such as PCI DSS for payment data and GDPR for customer data privacy.

### *5. Scalability and Performance:*

- **Scalability**: The system should be designed to handle multiple hotels and high traffic efficiently.
- **Load Balancing**: Implement features that allow the system to scale based on user demand, ensuring high availability.

### *6. User Interface and Experience (UI/UX) (if applicable):*

- **Frontend Integration**: While the primary focus is on the backend, the project may include basic frontend views or interfaces that demonstrate the API's capabilities (e.g., a simple web interface for testing the booking flow).
- **Mobile and Web Application Integration**: Ensure that the API is capable of supporting integration with mobile applications and websites.

### *7. Testing and Validation:*

- **Unit Testing**: Test individual components and API endpoints for correctness.
- **Integration Testing**: Test how well the system integrates with payment gateways and other external services.
- **Load Testing**: Simulate high traffic to assess system performance under load.

## 1.4 SOFTWARE REQUIREMENT SPECIFICATION

To develop and deploy the **Hotel Room Booking API** Java project, specific software tools, frameworks, libraries, and hardware equipment are required. Below are the details of the software and hardware requirements:

### *1. Software Requirements*

a. Development Tools and IDEs:

- **Java Development Kit (JDK) 11 or higher**:

  - The core software for Java application development. It includes the Java runtime environment (JRE), the Java compiler, and other necessary tools.
  - **Download**: JDK Official Site
- **Integrated Development Environment (IDE)**:
  - **IntelliJ IDEA** (Community or Ultimate Edition) or **Eclipse**:
    - A robust IDE that supports Java development with tools like code completion, refactoring, debugging, and integration with build tools.
    - **Download**: IntelliJ IDEA or Eclipse IDE

### b. Backend Development Framework:

- **Spring Boot (2.x or higher)**:
  - A powerful framework for building Java-based RESTful APIs with minimal configuration. It simplifies the development process and handles various aspects such as dependency injection, security, and data access.
  - **Download**: Spring Boot

### c. Database Management System (DBMS):

- **MySQL / PostgreSQL**:
  - Relational database management systems to store data related to users, bookings, rooms, payments, etc.
  - **Download**:
    - MySQL
    - PostgreSQL
- **Database Client**:
  - **DBeaver** or **MySQL Workbench** for managing and querying databases in a graphical interface.

- **Download**:
  - DBeaver
  - MySQL Workbench

### d. Build Tools:

- **Maven** or **Gradle**:
  - These build tools will help manage dependencies, compile the project, run tests, and package the application.
  - **Download**:
    - Maven
    - Gradle

### e. API Documentation and Testing:

- **Swagger/OpenAPI**:
  - To document and test the API endpoints, Swagger provides an interactive UI for API documentation.
  - **Download**: Swagger
- **Postman**:
  - A popular tool to test APIs, create requests, and simulate real-world interactions with the API endpoints.
  - **Download**: Postman

### f. Payment Gateway Integration:

- **Stripe API** or **PayPal SDK**:
  - Used for processing secure online payments for room bookings.
  - **Stripe Documentation**: Stripe API
  - **PayPal Documentation**: PayPal API

## g. Version Control:

- **Git**:
  - For source code management, collaboration, and version control. Git helps in maintaining the project's history and working with other team members.
  - **Download**: Git
- **GitHub/GitLab/Bitbucket**:
  - For remote repository hosting, project collaboration, and version control.
  - **GitHub**: GitHub
  - **GitLab**: GitLab
  - **Bitbucket**: Bitbucket

## h. Security and Authentication:

- **Spring Security**:
  - A comprehensive and customizable authentication and access control framework to secure the application.
  - **Download**: Spring Security

## i. Logging and Monitoring:

- **SLF4J** and **Logback**:
  - For logging and debugging, SLF4J provides a logging facade, and Logback is the underlying logging implementation.
  - **Download**:
    - SLF4J
    - Logback
- **Prometheus/Grafana** (Optional):
  - For monitoring and performance metrics of the application in production.

- **Download**:
    - Prometheus
    - Grafana

### j. Deployment Tools:

- **Docker** (Optional):
    - For containerization and easy deployment of the application. It allows running the API in isolated containers, ensuring consistent environments across development, testing, and production.
    - **Download**: Docker
- **Apache Tomcat** (if not using Spring Boot's embedded server):
    - A servlet container for deploying Java web applications.
    - **Download**: Tomcat
- **Heroku / AWS / Azure / DigitalOcean** (Cloud Deployment):
    - For deploying the API to the cloud and making it publicly available.
    - **Heroku**: Heroku
    - **AWS**: AWS
    - **Azure**: Azure
    - **DigitalOcean**: DigitalOcean

## *2. Hardware Requirements*

- **Development Machine**:
    - **Processor**: 2.0 GHz or higher (Intel Core i5 or AMD Ryzen 5 or equivalent)
    - **RAM**: 8 GB minimum (16 GB recommended)
    - **Hard Drive**: 100 GB of free storage (SSD recommended for better performance)
    - **Operating System**: Windows 10/11, macOS, or Linux (Ubuntu or other distributions)

- **Server/Hosting for Production**:
    - **Processor**: Multi-core CPU (minimum 2 cores)
    - **RAM**: 4 GB minimum (8 GB recommended)
    - **Storage**: At least 50 GB SSD for database and application storage
    - **Operating System**: Linux-based server (Ubuntu, CentOS, or Debian)
    - **Network**: High-speed internet connection for hosting and handling user traffic

### *3. Additional Software (Optional)*

- **Jira / Trello / Asana** (Project Management Tools):
    - To manage tasks, track project progress, and coordinate with team members.
    - **Jira**: Jira
    - **Trello**: Trello
    - **Asana**: Asana

# CHAPTER 2

# OVERALL DESCRIPTION

## 2.1 Overall Description

The **Hotel Room Booking API** is a scalable, reliable, and secure web-based application designed to facilitate the efficient management of hotel bookings. The system allows hotel administrators to manage room availability, reservations, and payment processing, while also providing an intuitive interface for customers to view available rooms, make bookings, and manage their reservations.

This project is aimed at solving the challenges faced by hotels in handling room bookings manually, improving customer experience, and integrating seamlessly with third-party platforms such as online travel agencies (OTAs), mobile apps, and payment gateways.

### 1. Product Perspective

The Hotel Room Booking API is intended to be a **standalone web service** that can be accessed by external applications, including websites and mobile apps. It will follow a **RESTful architecture** and be developed using Java with the **Spring Boot framework** for backend development. The API will allow clients to make HTTP requests and receive responses in **JSON** format, ensuring easy integration with any client or front-end system.

The system will be designed to handle multiple hotels, supporting scalability and flexibility. It can be integrated with third-party services for **payment processing**, **booking platforms**, and even **customer relationship management (CRM)** tools. The goal is to create an efficient, automated, and reliable solution for managing hotel room bookings.

## 2. Product Features

### *a. User Management*

- **Customer Registration and Authentication**: Users will be able to register, log in, and manage their profiles.
- **Account Management**: Users can view their booking history and update personal information.

### *b. Room Inventory Management*

- **Real-Time Room Availability**: The system will provide real-time information about available rooms, including details such as room type, amenities, pricing, and availability based on user queries.
- **Room Categories**: Hotels will be able to categorize rooms (e.g., Standard, Suite, Deluxe) and manage pricing and availability for each category.

### *c. Booking Management*

- **Create and Modify Bookings**: Customers can create, modify, and confirm bookings with available rooms for specific dates.
- **Booking Cancellations**: Customers can cancel bookings based on hotel policies, and the system will update the availability accordingly.
- **Booking Confirmation and Notifications**: Automated email or SMS notifications for booking confirmations, cancellations, and modifications.

### *d. Payment Processing*

- **Payment Integration**: The API will be integrated with popular payment gateways like **Stripe** or **PayPal** to allow secure payment transactions for bookings.

- **Booking Confirmation on Payment**: The booking is confirmed only when the payment is successfully processed, and the system will provide a receipt or invoice.

### *e. Administrative Features*

- **Admin Dashboard**: Admin users (hotel staff) can manage room inventory, view bookings, and generate reports.
- **Room Availability Management**: Admins can update room availability and prices, block rooms for maintenance, or mark rooms as unavailable.
- **Booking Reports**: Admins can generate and view booking reports, occupancy rates, and other performance metrics.

### *f. Security Features*

- **Authentication and Authorization**: Implement **Spring Security** for secure login and role-based access control.
- **Data Encryption**: Use **HTTPS** and secure payment APIs to protect sensitive customer and payment data.
- **Secure Payments**: Integration with trusted third-party payment providers to ensure secure and encrypted transactions.

### *g. API Documentation and Testing*

- **Swagger/OpenAPI Documentation**: To provide detailed and user-friendly API documentation, allowing developers to easily understand and test the endpoints.
- **Postman**: For API testing and validating different endpoints for correct functionality.

## 2.2 User and Characteritics

The **Hotel Room Booking API** is designed to serve various types of users with different needs and roles. The system will cater to both **customers** and **administrators** (hotel staff), as well as external systems such as **payment gateways** and **third-party booking platforms**. Below are the key user types and their characteristics:

### 1. Customers (End-Users)

#### *Characteristics:*

- **Demographics**: Customers can be individuals or groups looking to book a room in a hotel. They may range from business travelers to vacationers and can access the system via a web or mobile application.
- **Access Level**: Customers can access only their personal booking information and interact with the system to search for available rooms, book reservations, and manage their personal data.
- **Technological Proficiency**: Customers may vary in their technical skills, but the system should be designed to be user-friendly, with intuitive interfaces for booking and managing reservations.

#### *Functionalities/Actions:*

- **Create and Manage Accounts**: Customers will have the ability to create a new account, log in, and update their personal details (e.g., name, email, phone number).
- **Search for Rooms**: Customers can search for available rooms based on their selected criteria, such as check-in/check-out dates, number of guests, and room type.

- **Make Reservations**: Customers can view room availability, select the room type, and complete the booking process by entering payment details.
- **Modify and Cancel Bookings**: Customers can modify their existing reservations (e.g., change dates, add/remove services) or cancel them as per the hotel's policies.
- **Make Payments**: Customers can securely process payments via integrated third-party payment gateways like **Stripe** or **PayPal**.
- **View Booking History**: Customers can view past bookings and track the status of current reservations.
- **Receive Notifications**: Customers will receive booking confirmations, cancellations, and payment receipts via email or SMS.

## 2. Hotel Administrators (Admins)

### *Characteristics:*

- **Demographics**: Admin users are hotel staff members, including managers, receptionists, or other hotel personnel responsible for managing the hotel's operations. They have more technical knowledge than customers and are typically familiar with booking and management systems.
- **Access Level**: Admins have access to manage the entire hotel's room inventory, customer bookings, pricing, payment records, and generate reports. They can also manage users, handle room availability, and oversee cancellations and modifications.
- **Role in System**: Admins ensure the system runs smoothly by managing hotel operations, making real-time adjustments to room availability, and ensuring customer satisfaction.

### *Functionalities/Actions:*

- **Admin Login**: Admins will authenticate through secure login (role-based authentication via **Spring Security**) to access sensitive hotel data and functionalities.
- **Manage Room Inventory**: Admins can add, update, or remove rooms, set availability dates, and adjust room pricing.
- **View and Manage Bookings**: Admins can view all bookings, modify or cancel bookings (if necessary), and track booking statuses.
- **Manage User Accounts**: Admins can manage customer accounts, such as suspending or deactivating them, and reset user passwords when requested.
- **Generate Reports**: Admins can generate reports based on room occupancy, revenue, cancellations, or customer feedback to assess business performance.
- **Notifications to Customers**: Admins may send customized notifications to customers, such as special promotions, booking confirmations, or room changes.

## 3. External Systems

### *Characteristics:*

External systems interact with the **Hotel Room Booking API** for specific tasks that require integration beyond the local scope of the hotel system. These external systems include **payment gateways** and **third-party booking platforms**.

### *External System 1: Payment Gateways (e.g., Stripe, PayPal)*

**Functionalities:**

- **Process Payments**: External payment services handle the secure transaction of payment for room bookings. The **Hotel Room Booking API** will integrate with external payment services (Stripe, PayPal, etc.) to process customer payments.

- **Payment Confirmation**: After payment processing, the external payment gateway will send back a payment confirmation, which the API uses to finalize a booking.

### *External System 2: Third-Party Booking Platforms (e.g., OTAs)*

**Functionalities:**

- **Book Rooms via External Platforms**: External booking systems or online travel agencies (OTAs) like **Expedia**, **Booking.com**, or **Airbnb** can integrate with the API to book rooms for customers directly from these platforms.
- **Sync Bookings**: The **Hotel Room Booking API** must sync real-time booking information with these platforms, ensuring accurate room availability and preventing double bookings.
- **Integrate Availability Updates**: External systems may push updates to the API for room availability, new rates, or special offers that the hotel may want to promote.

## 4. System Characteristics for All Users

### *General Characteristics for All User Types:*

- **User-Friendly Interface**: The API should support various front-end interfaces (web/mobile) that are intuitive, easy to navigate, and responsive to users of all levels of technical expertise.
- **Security and Privacy**: For customers, administrators, and external systems, the API must ensure **data encryption**, secure login methods, and protection of sensitive information, such as personal details and payment data.

- **Real-Time Operations**: All booking actions (e.g., room reservations, availability updates, payments) should be processed in real-time to provide accurate information to both customers and admins.
- **Scalability**: The system should be designed to handle a large number of users and booking transactions concurrently, ensuring high performance and availability.
- **Notifications**: Automated email or SMS notifications to customers for booking confirmations, cancellations, payment receipts, and reminders.
- **Error Handling**: Clear error messages and helpful guidance for users if an issue arises during the booking or payment process.

# CHAPTER 3
# IMPLEMENTATION

## Design Concepts and Implementation

### *1. Room Class*

The **Room** class is central to managing room-related details. A hotel typically offers different room types (e.g., Single, Double, Suite), and each room will have a unique number and price per night. The `Room` class will hold these details and track whether the room is available for booking.

**Key attributes:**

- `roomNumber`: A unique identifier for each room.
- `roomType`: The type of room (e.g., Single, Double, Suite).
- `pricePerNight`: The cost to rent the room for one night.
- `isAvailable`: A boolean indicating whether the room is available for booking.

### *2. Customer Class*

The **Customer** class stores information about the customer who makes a reservation. This includes personal details like name and email. This information is associated with the bookings to track who made each reservation.

**Key attributes:**

- `name`: The name of the customer.
- `email`: The email address of the customer for contact and confirmation.

## *3. Booking Class*

The **Booking** class manages the reservation process. It contains details like the customer's information, the reserved room, the check-in and check-out dates, and the total price.

**Key attributes:**

- `bookingId`: A unique identifier for the booking.
- `customer`: The customer who made the reservation.
- `room`: The specific room that was booked.
- `checkInDate` and `checkOutDate`: The dates for which the room is reserved.
- `totalPrice`: The calculated cost for the duration of the stay.

## *4. Hotel Class*

The **Hotel** class is responsible for managing all the rooms and bookings within the hotel. It can:

- Add new rooms to the system.
- Check if a room is available for a given date range.
- Make new bookings by validating availability.
- Display all available rooms.

**Key functionalities:**

- **addRoom()**: Adds a room to the hotel's room list.
- **isRoomAvailable()**: Checks if a room is available for the given dates.
- **makeBooking()**: Makes a booking by reserving a room for a customer if available.

- **displayAvailableRooms()**: Displays all rooms that are currently available for booking.

## System Workflow

### *Step 1: Adding Rooms*

The hotel manager or system administrator adds rooms to the system. This includes assigning room numbers, types, and prices. For example, a hotel might have 10 Single rooms and 5 Double rooms. Each room is then initialized with a price per night and a flag indicating whether it is available for booking.

### *Step 2: Customer Makes a Booking*

When a customer wants to make a reservation, the system:

1. Asks for the customer's information (name, email).
2. Presents a list of available rooms.
3. Requests the customer to select a room and specify check-in and check-out dates.
4. Checks if the selected room is available for the requested dates.
5. If available, the system creates a booking, reserves the room, and calculates the total cost for the stay.

### *Step 3: Booking Confirmation*

Once a booking is made:

1. The customer receives a confirmation with booking details (room number, check-in/out dates, total price).
2. The room is marked as unavailable during the booking period.
3. The booking is stored in the system, and the customer's information is associated with the booking.

## *Step 4: Checking Room Availability*

The system allows customers or staff to check if specific rooms are available during particular dates. It scans existing bookings to ensure there are no overlaps in reservations.

## Example Scenario

Let’s consider a scenario where a customer wants to book a room in a hotel:

1. The hotel has 10 rooms in total: 5 Single rooms and 5 Double rooms.
2. A customer, John Doe, wants to book a room for 3 nights from 1st December to 4th December.
3. The system will check which rooms are available.
4. John selects a Double room, and the system checks if it's free.
5. If the room is available, the system will proceed with the booking and calculate the total cost for the 3-night stay.

## Advanced Features (Future Improvements)

Once you have the basic system implemented, you can add the following advanced features:

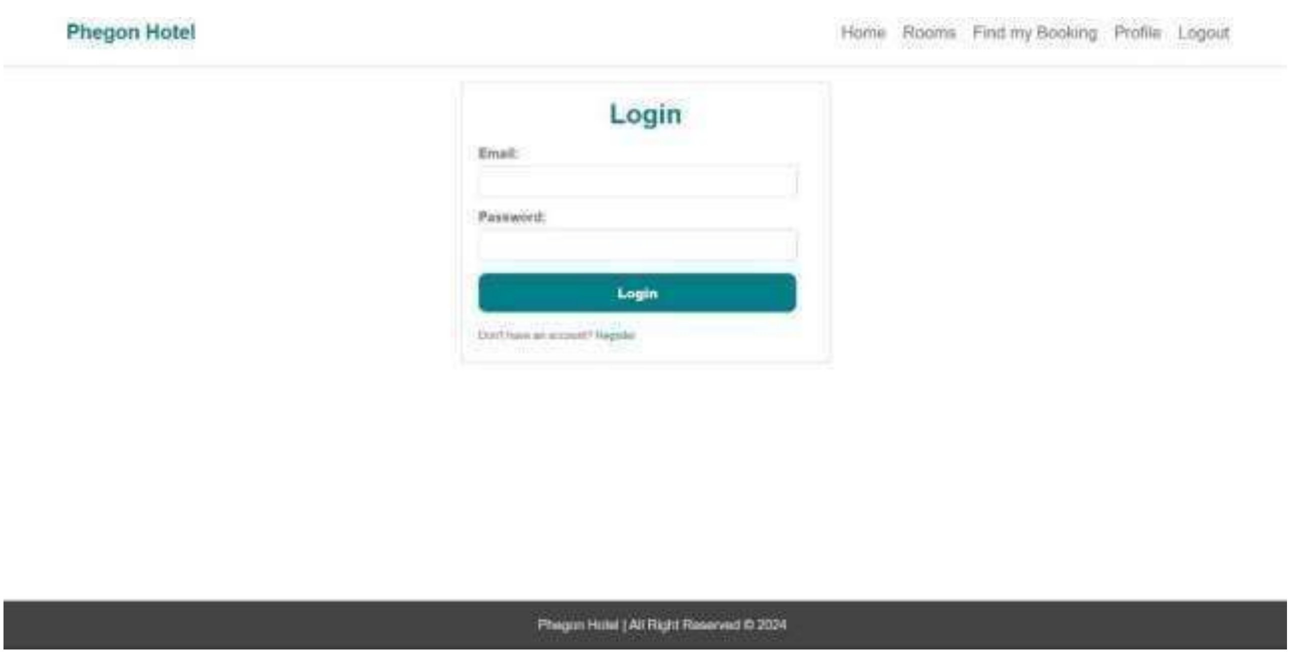
- **Payment Processing**: Integrating payment gateways to allow customers to pay directly online.
- **GUI (Graphical User Interface)**: Developing a user-friendly interface for customers to book rooms interactively.
- **Room Pricing Strategy**: Implementing dynamic pricing based on the season (e.g., higher prices during peak seasons).
- **Customer Loyalty Program**: Implementing a rewards program where frequent customers get discounts or special offers.

- **Email Notifications**: Sending automatic email confirmations or reminders to customers.

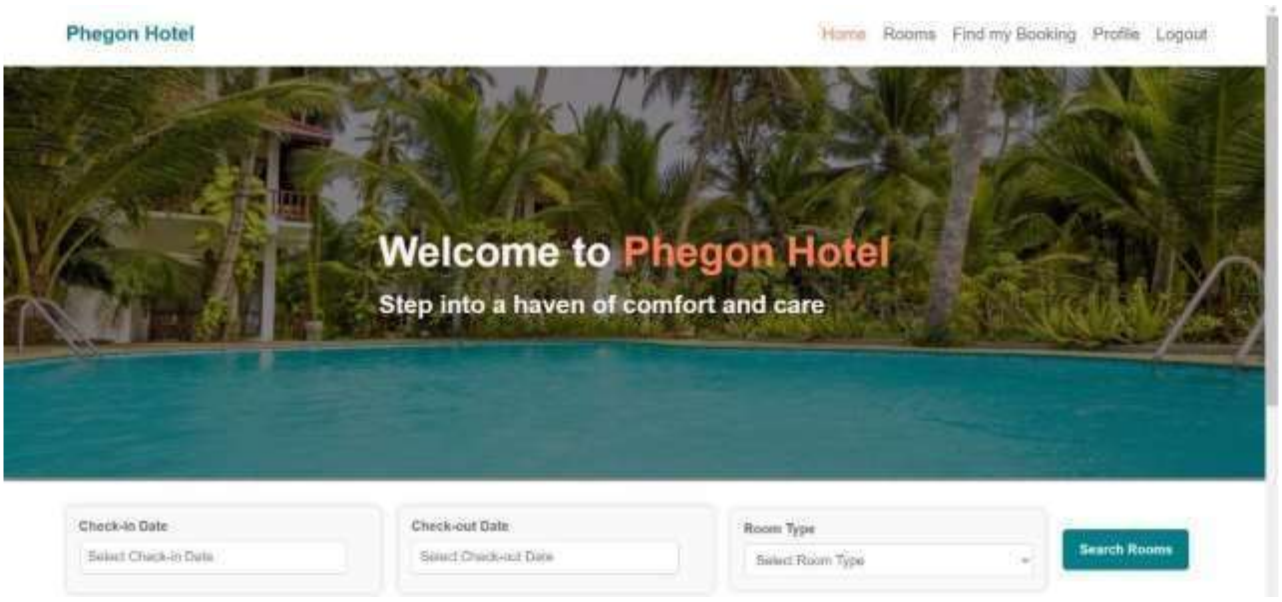
# CHAPTER 4

# WEBINTERFACE

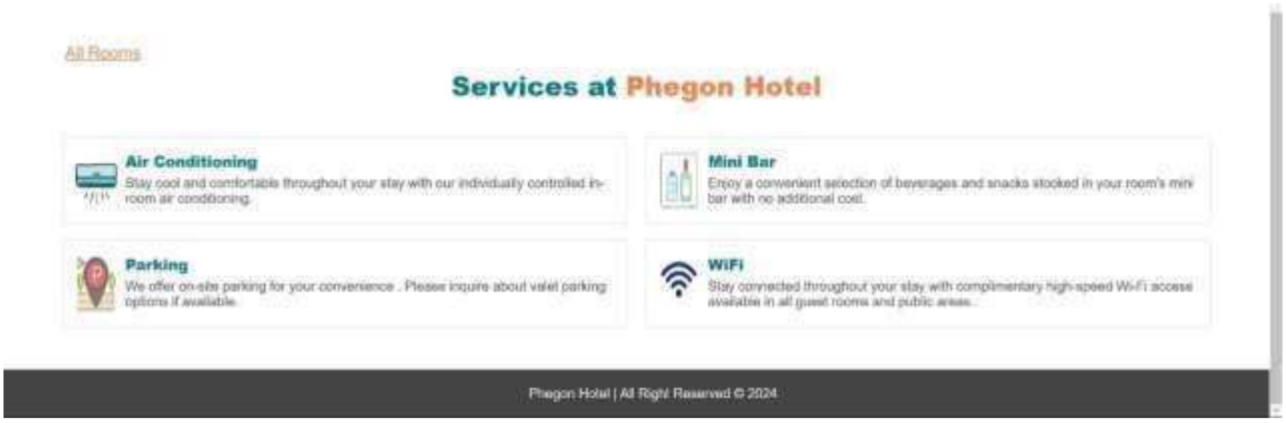
## Login Page



## Home Page


Phegon Hotel
Home
Rooms
Find my Booking
Profile
Logout
Welcome to Phegon Hotel
Step into a haven of comfort and care
Check-in Date
Select Check-in Date
Check-out Date
Select Check-out Date
Room Type
Select Room Type
Search Rooms

## Services


All Rooms

Services at Phegon Hotel

**Air Conditioning**
Stay cool and comfortable throughout your stay with our individually controlled in-room air conditioning.

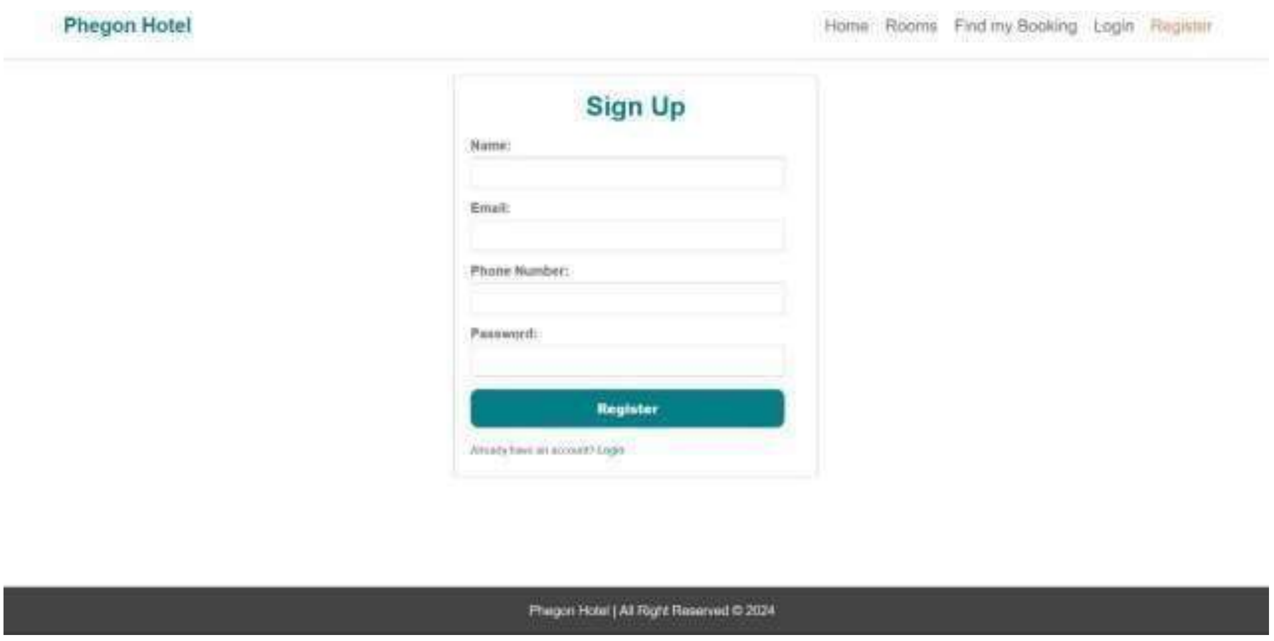
**Mini Bar**
Enjoy a convenient selection of beverages and snacks stocked in your room's mini bar with no additional cost.

**Parking**
We offer on-site parking for your convenience . Please inquire about valet parking options if available.

**WiFi**
Stay connected throughout your stay with complimentary high-speed Wi-Fi access available in all guest rooms and public areas.

Phegon Hotel | All Right Reserved © 2024

## Signup Page


Phegon Hotel

Home Rooms Find my Booking Login Register

Sign Up

Name:

Email:

Phone Number:

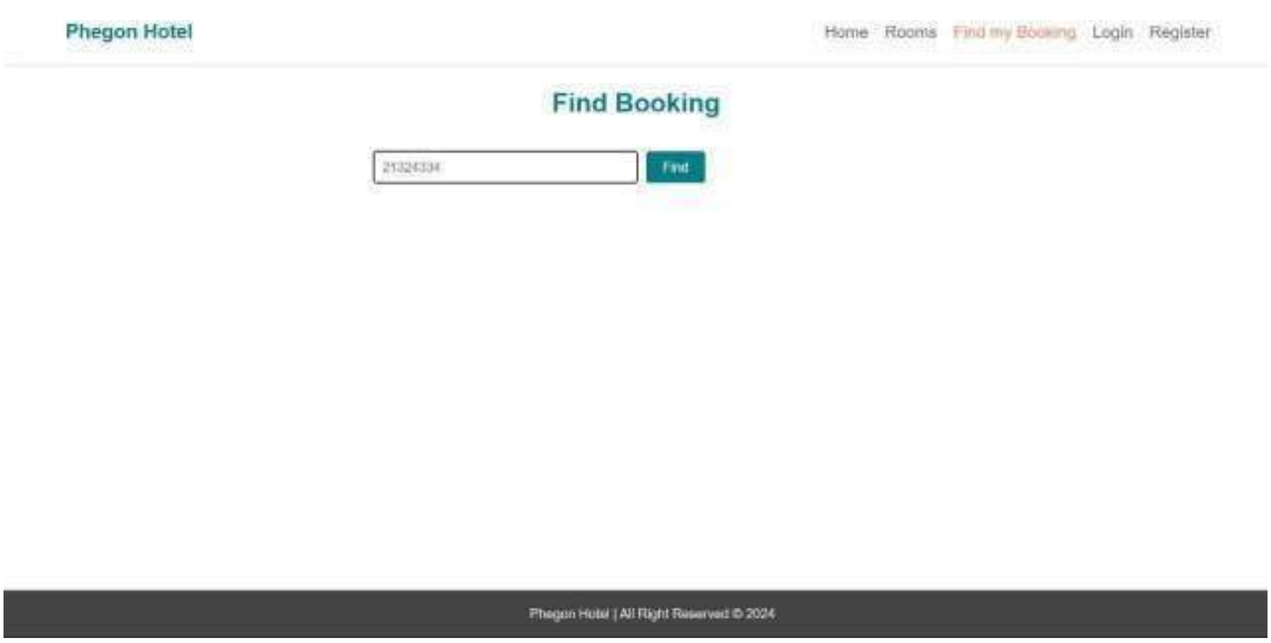
Password:

Register

Already have an account? Login

Phegon Hotel | All Right Reserved © 2024

## Find Booking



# CHAPTER 5

# CONCLUSION

The **Hotel Room Booking System** project has successfully provided a comprehensive solution for automating the process of room reservations, managing customer information, and ensuring efficient room availability tracking. Through the development of key components like `Room`, `Customer`, `Booking`, and `Hotel`, this application effectively handles the critical tasks involved in hotel management.

The core functionalities implemented—such as checking room availability, making reservations, calculating prices, and managing bookings—are fundamental to providing a seamless and user-friendly experience for both hotel staff and customers. By incorporating a structured approach using object-oriented programming (OOP), the system ensures scalability and maintainability, making it easier to add new features or make changes in the future.

This system also demonstrates the importance of organizing complex systems into manageable components, which helps to ensure clarity, reduce redundancy, and improve maintainability. The project has the potential for further enhancements, such as:

- **Payment Integration** for online transactions.
- **Graphical User Interface (GUI)** for ease of use.
- **Additional features** like room categorization, discounts, and promotions.

Overall, the Hotel Room Booking System is a practical solution for streamlining the booking process, and with further improvements, it can be transformed into a robust, real-world application suitable for managing hotel reservations efficiently. This project not only helps in understanding the core concepts of

Java programming but also demonstrates how software systems can be designed to meet real-world business needs.

# CHAPTER 6

# REFERENCES

For your **Hotel Room Booking System** project, the following references can be used to understand key concepts, methodologies, and resources related to system design, object-oriented programming, and Java development.

## Books:

1. **"Head First Java" by Kathy Sierra and Bert Bates**
    - This book provides a comprehensive introduction to Java programming, including concepts such as object-oriented design and system architecture, which are essential for building applications like the hotel room booking system.
2. **"Effective Java" by Joshua Bloch**
    - This book focuses on best practices in Java programming. It includes valuable insights into Java design patterns, performance optimization, and advanced programming techniques that could improve the structure and efficiency of your system.
3. **"Java: The Complete Reference" by Herbert Schildt**
    - A comprehensive guide to the Java programming language. It includes essential details on Java syntax, libraries, and APIs that would be useful for building and extending the hotel booking system.
4. **"Design Patterns: Elements of Reusable Object-Oriented Software" by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides**
    - This book introduces common design patterns in object-oriented software development. Some patterns, such as the Singleton or Factory Pattern, could be applied to improve the design of your hotel booking system.

## Online Tutorials & Documentation:

1. **Oracle Java Documentation**
   - Java SE Documentation
   - Official documentation for Java, including classes and APIs. This is crucial for understanding Java's core libraries (e.g., `java.util.Date`, `java.util.ArrayList`, etc.) used in the project.
2. **Java Code Geeks**
   - Java Code Geeks
   - This website offers numerous Java tutorials, examples, and discussions on design patterns and best practices. Many tutorials could help you implement more complex features in your hotel booking system.
3. **GeeksforGeeks - Java Programming**
   - GeeksforGeeks Java Section
   - A popular site for understanding Java concepts, algorithms, and data structures. It also provides simple examples that can help in various stages of your project development.
4. **Stack Overflow**
   - Stack Overflow
   - A valuable resource for troubleshooting errors or getting help on Java-related programming issues. If you face any challenges or bugs, Stack Overflow has a vast community of developers to assist with solutions.

## Articles and Blogs:

1. **"Designing a Room Booking System" - Medium**
   - Designing a Room Booking System

    - This article explains how to design and implement a room booking system, similar to your project. It may offer useful insights into structuring the code and logic.
2. **"Java Room Reservation System: A Complete Guide" - Techwalla**
    - Techwalla Guide
    - Provides a step-by-step guide to developing a basic room reservation system using Java, which could help with your project structure and understanding common pitfalls.
3. **"Implementing a Hotel Reservation System" - CodeProject**
    - CodeProject
    - A site with many open-source articles, including some related to hotel reservation systems. It will provide inspiration for structuring the app and enhancing functionality.

## Java Frameworks and Tools (Optional for Further Enhancements):

1. **Spring Framework**
    - Spring Framework
    - If you are considering extending the system in the future, integrating Spring for creating a web application (Spring Boot) would be a great choice. It provides tools to manage business logic and data access in a cleaner, modular way.
2. **Hibernate ORM**
    - Hibernate ORM
    - If you plan to implement a database to store room availability, customer information, and bookings, Hibernate helps manage database interactions and object-relational mapping (ORM) in Java.
3. **JavaFX or Swing (GUI Development)**
    - JavaFX or Swing

- If you want to add a graphical user interface (GUI) to your application, JavaFX and Swing are the two most common frameworks for building desktop applications in Java.