

LAB MANUAL

Subject Name: Java Programming

Subject Code: CS3312

Year: II

Semester: III

Batch: 2024-2025

INDEX

Ex. No.	Date	Title	Marks	Staff Sign
1		SEARCHING AND SORTING TECHNIQUES		
2		STACK AND QUEUE DATA STRUCTURES		
3		INHERITANCE		
4		ABSTRACT CLASS		
5		INTERFACE		
6		EXCEPTION HANDLING		
7		MULTI-THREADED PROGRAMMING		
8		JAVA I/O FILE OPERATIONS		
9		GENERIC PROGRAMMING		
10		MINI PROJECT		

Ex. No. 1. Solve problems by using sequential search, binary search, and quadratic sorting algorithms (selection, insertion)

a) Sequential Search

Aim:

To write a java program for implementing Sequential Search.

Algorithm:

1. Declare the class Sequential Search.
2. Traverse the array
3. Match the key element with array element
4. If key element is found, return the index position of the array element
5. If key element is not found, return -1
6. Print the output.

Source Code:

```
public class SequentialSearch {  
    public static int sequentialSearch(int[] arr, int target) {  
        for (int i = 0; i < arr.length; i++) {  
            if (arr[i] == target) {  
                return i;  
            }  
        }  
        return -1; // Element not found  
    }  
    public static void main(String[] args) {  
        int[] arr = { 10, 20, 30, 40, 50 };  
        int target = 30;  
        int index = sequentialSearch(arr, target);  
        if (index != -1) {  
            System.out.println("Element found at index: " + index);  
        }  
        else {  
            System.out.println("Element not found.");  
        } } }  
}
```

Output:

Element found at index: 2

b) Binary Search

Aim:

To write a java program for implementing Binary Search.

Algorithm:

1. Import the required packages.
2. Declare the class Binary Search
3. Sort the array
4. Choose low and high values
5. Choose Midpoint or middle element
6. Binary Search
 - a. If element_to_be_found is equal to midpoint, the midpoint index will be returned.
 - b. If element_to_be_found is greater than midpoint, search through the elements on the right side of the midpoint.
 - c. If element_to_be_found is less than midpoint, search through the elements on the left side midpoint.
7. Once the element is found, print the output.

Source Code:

```
import java.util.Arrays;

public class BinarySearch {

    public static int binarySearch(int[] arr, int target) {

        int left = 0;

        int right = arr.length - 1;

        while (left <= right) {

            int mid = left + (right - left) / 2;

            if (arr[mid] == target) {

                return mid;

            } else if (arr[mid] < target) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return -1; // Element not found

    }

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

```

int[] arr = { 10, 20, 30, 40, 50 };
int target = 30;
Arrays.sort(arr); // Binary search requires a sorted array
int index = binarySearch(arr, target);
if (index != -1) {
    System.out.println("Element found at index: " + index);
} else {
    System.out.println("Element not found.");
}
}
}

```

Output:

Element found at index: 2

c) Selection Sort

Aim:

To write a java program for implementing Selection Sort.

Algorithm:

1. Import the required packages.
2. Declare the class Selection Sort.
3. Set MIN to location 0
4. Search the minimum element in the list
5. Swap with value at location MIN
6. Increment MIN to point to next element
7. Repeat until list is sorted

Source Code:

```

import java.util.Arrays;

public class SelectionSort {

    public static void selectionSort(int[] arr) {
        for (int i = 0; i < arr.length - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
        }
    }
}

```

```

        }
        int temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
    }
}

public static void main(String[] args) {
    int[] arr = { 64, 25, 12, 22, 11 };
    System.out.println("Unsorted array: " + Arrays.toString(arr));
    selectionSort(arr);
    System.out.println("Sorted array: " + Arrays.toString(arr));
}
}

```

Output:

Unsorted array: [64, 25, 12, 22, 11]

Sorted array: [11, 12, 22, 25, 64]

d) Insertion Sort

Aim:

To write a java program for implementing Insertion Sort.

Algorithm:

1. Import the required packages.
2. Declare the class Insertion Sort.
3. If the element is the first element, assume that it is already sorted. Return 1.
4. Pick the next element, and store it separately in a key.
5. Now, compare the key with all elements in the sorted array.
6. If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right.
7. Insert the value.
8. Repeat until the array is sorted.

Source Code:

```

import java.util.Arrays;

public class InsertionSort {
    public static void insertionSort(int[] arr) {
        for (int i = 1; i < arr.length; i++) {
            int key = arr[i];

```

```

        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

public static void main(String[] args) {
    int[] arr = { 64, 25, 12, 22, 11 };
    System.out.println("Unsorted array: " + Arrays.toString(arr));
    insertionSort(arr);
    System.out.println("Sorted array: " + Arrays.toString(arr));
}
}

```

Output:

Unsorted array: [64, 25, 12, 22, 11]

Sorted array: [11, 12, 22, 25, 64]

Result:

The programs sequential search, binary search, and quadratic sorting algorithms (selection, insertion) are executed successfully.

Ex. No. 2. Develop stack and queue data structures using classes and objects.

Aim:

To develop stack and queue data structures using classes and objects in java.

A) Stack Data Structure:

Algorithm:

1. Create a class named "Stack" with private instance variables:
 - a. arr: an integer array to store stack elements.
 - b. top: an integer representing the index of the top element in the stack.
 - c. capacity: an integer representing the total capacity of the stack.
2. Define a constructor for the Stack class:
 - a. Initialize the capacity with the specified size.
 - b. Initialize the arr with a new integer array of the given capacity.
 - c. Initialize top as -1 to indicate an empty stack.
3. Implement the "push" method:
 - a. Check if the stack is full ($\text{top} == \text{capacity} - 1$).
 - b. If it is, print "Stack Overflow" and terminate the program.
 - c. Otherwise, increment top by 1 and insert the element 'x' into $\text{arr}[\text{top}]$.
4. Implement the "pop" method:
 - a. Check if the stack is empty ($\text{top} == -1$).
 - b. If it is, print "STACK EMPTY" and terminate the program.
 - c. Otherwise, return the element at $\text{arr}[\text{top}]$ and decrement top by 1 to remove the top element.
5. Implement the "getSize" method:
 - a. Return the size of the stack, which is $(\text{top} + 1)$.
6. Implement the "isEmpty" method:
 - a. Check if top is equal to -1 and return true if it is, indicating an empty stack; otherwise, return false.
7. Implement the "isFull" method:
 - a. Check if top is equal to $\text{capacity} - 1$ and return true if it is, indicating a full stack; otherwise, return false.
8. Implement the "printStack" method:
 - a. Loop through the elements in the stack from index 0 to top:
 - b. Print each element followed by a comma and space.
 - c. This method is used to display the elements of the stack.
9. In the main method:
 - a. Create an instance of the Stack class with a specified size (e.g., 5).
 - b. Push elements (e.g., 1, 2, 3) onto the stack.
 - c. Print the contents of the stack using the "printStack" method.
 - d. Pop an element from the stack.
 - e. Print the contents of the stack again to show the result after popping.
10. End of the program.

SOURCE CODE:

// Stack implementation in Java

```
class Stack {  
  
    // store elements of stack
```



```
private int arr[];
// represent top of stack
private int top;
// total capacity of the stack
private int capacity;
// Creating a stack
Stack(int size) {
    // initialize the array
    // initialize the stack variables
    arr = new int[size];
    capacity = size;
    top = -1;
}
// push elements to the top of stack
public void push(int x) {
    if (isFull()) {
        System.out.println("Stack OverFlow");
        // terminates the program
        System.exit(1);
    }
    // insert element on top of stack
    System.out.println("Inserting " + x);
    arr[++top] = x;
}
// pop elements from top of stack
public int pop() {
    // if stack is empty
    // no element to pop
    if (isEmpty()) {
        System.out.println("STACK EMPTY");
        // terminates the program
        System.exit(1);
    }
}
```

```

    }

    // pop element from top of stack
    return arr[top--];
}

// return size of the stack
public int getSize() {
    return top + 1;
}

// check if the stack is empty
public Boolean isEmpty() {
    return top == -1;
}

// check if the stack is full
public Boolean isFull() {
    return top == capacity - 1;
}

// display elements of stack
public void printStack() {
    for (int i = 0; i <= top; i++) {
        System.out.print(arr[i] + ", ");
    }
}

public static void main(String[] args) {
    Stack stack = new Stack(5);
    stack.push(1);
    stack.push(2);
    stack.push(3);
    System.out.print("Stack: ");
    stack.printStack();

    // remove element from stack
    stack.pop();
    System.out.println("\nAfter popping out");
}

```

```
        stack.printStack();  
    }  
}
```

OUTPUT:

Inserting 1

Inserting 2

Inserting 3

Stack: 1, 2, 3,

After popping out

1, 2,

B) Queue Data Structure:

Algorithm:

- 1) Create a class named "Queue" with the following instance variables:
 - a) SIZE: An integer representing the maximum size of the queue.
 - b) items: An integer array to store queue elements.
 - c) front: An integer representing the front index of the queue.
 - d) rear: An integer representing the rear index of the queue.
- 2) Define a constructor for the Queue class:
 - a) Initialize front and rear to -1 to indicate an empty queue.
- 3) Implement the "isFull" method:
 - a) Check if both front and rear are at their respective limits (front == 0 && rear == SIZE - 1).
 - b) If they are, return true, indicating the queue is full; otherwise, return false.
- 4) Implement the "isEmpty" method:
 - a) Check if front is -1.
 - b) If it is, return true, indicating the queue is empty; otherwise, return false.
- 5) Implement the "enQueue" method:
 - a) Check if the queue is full using the "isFull" method.
 - b) If it is, print "Queue is full."
 - c) Otherwise, if the front is -1, set it to 0 (marking the front element).
 - d) Increment rear by 1 and insert the element at items[rear].
 - e) Print "Insert <element>" to indicate the element insertion.
- 6) Implement the "deQueue" method:
 - a) Declare a variable 'element' to store the removed element.
 - b) Check if the queue is empty using the "isEmpty" method.
 - c) If it is, print "Queue is empty" and return -1.
 - d) Otherwise, remove the element at items[front] and assign it to 'element'.
 - e) If front is greater than or equal to rear, it means the queue had only one element. In this case, reset front and rear to -1.
 - f) Otherwise, increment front to mark the next element as the front.
 - g) Print "<element> Deleted" to indicate the element removal and return 'element'.

- 7) Implement the "display" method:
 - a) Check if the queue is empty using the "isEmpty" method.
 - b) If it is, print "Empty Queue."
 - c) Otherwise, print the front index, followed by displaying all elements in the queue from front to rear.
 - d) Finally, print the rear index.
- 8) In the main method:
 - a) Create an instance of the Queue class (e.g., 'q').
 - b) Attempt to dequeue an element from the queue (which should print "Queue is empty").
 - c) Enqueue elements (1 to 5) into the queue using the "enqueue" method.
 - d) Attempt to enqueue the 6th element (which should print "Queue is full").
 - e) Display the contents of the queue using the "display" method.
 - f) Dequeue an element (1) using the "dequeue" method.
 - g) Display the contents of the updated queue.
- 9) End of the program.

SOURCE CODE:

```
public class Queue {  
    int SIZE = 5;  
    int items[] = new int[SIZE];  
    int front, rear;  
    Queue() {  
        front = -1;  
        rear = -1;  
    }  
    // check if the queue is full  
    boolean isFull() {  
        if (front == 0 && rear == SIZE - 1) {  
            return true;  
        }  
        return false;  
    }  
    // check if the queue is empty  
    boolean isEmpty() {  
        if (front == -1)  
            return true;  
        else  
            return false;  
    }  
}
```

```

}

// insert elements to the queue
void enQueue(int element) {
    // if queue is full
    if (isFull()) {
        System.out.println("Queue is full");
    }
    else {
        if (front == -1) {
            // mark front denote first element of queue
            front = 0;
        }
        rear++;
        // insert element at the rear
        items[rear] = element;
        System.out.println("Insert " + element);
    }
}

// delete element from the queue
int deQueue() {
    int element;
    // if queue is empty
    if (isEmpty()) {
        System.out.println("Queue is empty");
        return (-1);
    }
    else {
        // remove element from the front of queue
        element = items[front];
        // if the queue has only one element
        if (front >= rear) {
            front = -1;
        }
    }
}

```

```

        rear = -1;
    }
    else {
        // mark next element as the front
        front++;
    }
    System.out.println( element + " Deleted");
    return (element);
}
}
// display element of the queue
void display() {
    int i;
    if (isEmpty()) {
        System.out.println("Empty Queue");
    }
    else {
        // display the front of the queue
        System.out.println("\nFront index-> " + front);
        // display element of the queue
        System.out.println("Items -> ");
        for (i = front; i <= rear; i++)
            System.out.print(items[i] + " ");
        // display the rear of the queue
        System.out.println("\nRear index-> " + rear);
    }
}

public static void main(String[] args) {
    // create an object of Queue class
    Queue q = new Queue();
    // try to delete element from the queue
    // currently queue is empty

```

```
// so deletion is not possible
q.deQueue();
// insert elements to the queue
for(int i = 1; i < 6; i++) {
    q.enqueue(i);
}
// 6th element can't be added to queue because queue is full
q.enqueue(6);
q.display();
// deQueue removes element entered first i.e. 1
q.deQueue();
// Now we have just 4 elements
q.display();
}
}
```

OUTPUT:

Queue is empty

Insert 1

Insert 2

Insert 3

Insert 4

Insert 5

Queue is full

Front index-> 0

Items ->

1 2 3 4 5

Rear index-> 4

1 Deleted

Front index-> 1

Items ->

2 3 4 5

Rear index-> 4

Result:

The Stack and Queue data structures using classes and objects has been executed successfully.

Ex. No. 3. Develop a java application with an Employee class with Emp_name, Emp_id, Address, Mail_id, Mobile_no as members. Inherit the classes, Programmer, Assistant Professor, Associate Professor and Professor from employee class. Add Basic Pay (BP) as the member of all the inherited classes with 97% of BP as DA, 10 % of BP as HRA, 12% of BP as PF, 0.1% of BP for staff club funds. Generate pay slips for the employees with their gross and net salary.

Aim:

To develop a java application which generates pay slips for the employees with their gross and net salary.

Algorithm:

1. Import the necessary Java libraries.
2. Create a class Employee with fields for employee information: emp_name, emp_id, address, mail_id, and mobile_no.
3. Define a constructor for the Employee class to initialize its fields.
4. Create subclasses for different types of employees (Programmer, AssistantProfessor, AssociateProfessor, and Professor) that extend the Employee class. Each subclass should have an additional field basicPay.
5. Define constructors for each subclass to initialize the basicPay field and call the superclass constructor to initialize the common employee information fields.
6. Implement methods getGrossSalary() and getNetSalary() in each subclass to calculate gross and net salaries based on the given formulas.
7. Create a PaySlipGenerator class with the main method.
8. Inside the main method:
 - a. Create instances of different employee types (Programmer, AssistantProfessor, AssociateProfessor, and Professor) with their respective basic pay.
 - b. Use DecimalFormat to format the salary values with two decimal places.
 - c. Print the pay slip for each employee type, including their name, ID, gross salary, and net salary.

SOURCE CODE:

```
import java.text.DecimalFormat;

class Employee {
    String emp_name;
    int emp_id;
    String address;
    String mail_id;
    String mobile_no;

    public Employee(String emp_name, int emp_id, String address, String mail_id, String
mobile_no) {
        this.emp_name = emp_name;
        this.emp_id = emp_id;
        this.address = address;
```

```

        this.mail_id = mail_id;

        this.mobile_no = mobile_no;
    }
}

class Programmer extends Employee {
    double basicPay;

    public Programmer(String emp_name, int emp_id, String address, String mail_id, String
mobile_no, double basicPay) {
        super(emp_name, emp_id, address, mail_id, mobile_no);
        this.basicPay = basicPay;
    }

    public double getGrossSalary() {
        double da = 0.97 * basicPay;
        double hra = 0.10 * basicPay;
        return basicPay + da + hra;
    }

    public double getNetSalary() {
        double pf = 0.12 * basicPay;
        double staffClubFunds = 0.001 * basicPay;
        return getGrossSalary() - pf - staffClubFunds;
    }
}

class AssistantProfessor extends Employee {
    double basicPay;

    public AssistantProfessor(String emp_name, int emp_id, String address, String mail_id,
String mobile_no, double basicPay) {
        super(emp_name, emp_id, address, mail_id, mobile_no);
        this.basicPay = basicPay;
    }

    public double getGrossSalary() {
        double da = 0.97 * basicPay;
        double hra = 0.10 * basicPay;
        return basicPay + da + hra;
    }
}

```

```

    }

    public double getNetSalary() {
        double pf = 0.12 * basicPay;
        double staffClubFunds = 0.001 * basicPay;
        return getGrossSalary() - pf - staffClubFunds;
    }
}

class AssociateProfessor extends Employee {
    double basicPay;

    public AssociateProfessor(String emp_name, int emp_id, String address, String mail_id,
String mobile_no, double basicPay) {
        super(emp_name, emp_id, address, mail_id, mobile_no);
        this.basicPay = basicPay;
    }

    public double getGrossSalary() {
        double da = 0.97 * basicPay;
        double hra = 0.10 * basicPay;
        return basicPay + da + hra;
    }

    public double getNetSalary() {
        double pf = 0.12 * basicPay;
        double staffClubFunds = 0.001 * basicPay;
        return getGrossSalary() - pf - staffClubFunds;
    }
}

class Professor extends Employee {
    double basicPay;

    public Professor(String emp_name, int emp_id, String address, String mail_id, String
mobile_no, double basicPay) {
        super(emp_name, emp_id, address, mail_id, mobile_no);
        this.basicPay = basicPay;
    }

    public double getGrossSalary() {

```

```

        double da = 0.97 * basicPay;

        double hra = 0.10 * basicPay;

        return basicPay + da + hra;
    }

    public double getNetSalary() {
        double pf = 0.12 * basicPay;

        double staffClubFunds = 0.001 * basicPay;

        return getGrossSalary() - pf - staffClubFunds;
    }
}

public class PaySlipGenerator {

    public static void main(String[] args) {

        DecimalFormat decimalFormat = new DecimalFormat("0.00");

// Creating employees with their basic pay

        Programmer programmer = new Programmer("John Doe", 101, "Address 1",
"john.doe@example.com", "1234567890", 50000.00);

        AssistantProfessor assistantProfessor = new AssistantProfessor("Alice Smith", 201,
"Address 2", "alice.smith@example.com", "9876543210", 60000.00);

        AssociateProfessor associateProfessor = new AssociateProfessor("Bob Johnson", 301,
"Address 3", "bob.johnson@example.com", "4567890123", 70000.00);

        Professor professor = new Professor("Emily Brown", 401, "Address 4",
"emily.brown@example.com", "7890123456", 80000.00);


// Generating pay slips

        System.out.println("Pay Slip for Programmer:");

        System.out.println("Employee Name: " + programmer.emp_name);

        System.out.println("Employee ID: " + programmer.emp_id);

        System.out.println("Gross Salary: $" +
decimalFormat.format(programmer.getGrossSalary()));

        System.out.println("Net Salary: $" +
decimalFormat.format(programmer.getNetSalary()));

        System.out.println();

        System.out.println("Pay Slip for Assistant Professor:");

        System.out.println("Employee Name: " + assistantProfessor.emp_name);

```

```
        System.out.println("Employee ID: " + assistantProfessor.emp_id);

        System.out.println("Gross Salary: $" +
decimalFormat.format(assistantProfessor.getGrossSalary()));

        System.out.println("Net Salary: $" +
decimalFormat.format(assistantProfessor.getNetSalary()));

        System.out.println();

        System.out.println("Pay Slip for Associate Professor:");

        System.out.println("Employee Name: " + associateProfessor.emp_name);

        System.out.println("Employee ID: " + associateProfessor.emp_id);

        System.out.println("Gross Salary: $" +
decimalFormat.format(associateProfessor.getGrossSalary()));

        System.out.println("Net Salary: $" +
decimalFormat.format(associateProfessor.getNetSalary()));

        System.out.println();

        System.out.println("Pay Slip for Professor:");

        System.out.println("Employee Name: " + professor.emp_name);

        System.out.println("Employee ID: " + professor.emp_id);

        System.out.println("Gross Salary: $" +
decimalFormat.format(professor.getGrossSalary()));

        System.out.println("Net Salary: $" + decimalFormat.format(professor.getNetSalary()));
    }
}
```

OUTPUT:

```
C:\Users\Dell\Desktop>javac PaySlipGenerator.java

C:\Users\Dell\Desktop>java PaySlipGenerator
Pay Slip for Programmer:
Employee Name: John Doe
Employee ID: 101
Gross Salary: $103500.00
Net Salary: $97450.00

Pay Slip for Assistant Professor:
Employee Name: Alice Smith
Employee ID: 201
Gross Salary: $124200.00
Net Salary: $116940.00

Pay Slip for Associate Professor:
Employee Name: Bob Johnson
Employee ID: 301
Gross Salary: $144900.00
Net Salary: $136430.00

Pay Slip for Professor:
Employee Name: Emily Brown
Employee ID: 401
Gross Salary: $165600.00
Net Salary: $155920.00

C:\Users\Dell\Desktop>|
```

Result:

A java application which generates pay slips for the employees with their gross and net salary has been successfully implemented.

Ex. No. 4. Write a Java Program to create an abstract class named Shape that contains two integers and an empty method named printArea(). Provide three classes named Rectangle, Triangle and Circle such that each one of the classes extends the class Shape. Each one of the classes contains only the method printArea() that prints the area of the given shape.

Aim:

To develop a java program which prints the area of the given shape.

ALGORITHM:

1. Define an abstract class named `Shape`:
 - Declare two integer variables `side1` and `side2`.
 - Define an abstract method `printArea()`.
2. Define a class named `Rectangle` that extends `Shape`:
 - Implement the `Rectangle` constructor that takes length and width as parameters and calls the superclass constructor.
 - Override the `printArea()` method to calculate and print the area of the rectangle (length * width).
3. Define a class named `Triangle` that extends `Shape`:
 - Implement the `Triangle` constructor that takes base and height as parameters and calls the superclass constructor.
 - Override the `printArea()` method to calculate and print the area of the triangle ($0.5 * \text{base} * \text{height}$).
4. Define a class named `Circle` that extends `Shape`:
 - Implement the `Circle` constructor that takes radius as a parameter and calls the superclass constructor with 0 as the second side.
 - Override the `printArea()` method to calculate and print the area of the circle ($\pi * \text{radius} * \text{radius}$).
5. Create a `ShapeMain` class:
 - In the `main()` method:
 - Create an instance of `Rectangle`, passing length and width as parameters.
 - Call the `printArea()` method for the `Rectangle` instance.
 - Create an instance of `Triangle`, passing base and height as parameters.
 - Call the `printArea()` method for the `Triangle` instance.
 - Create an instance of `Circle`, passing the radius as a parameter.
 - Call the `printArea()` method for the `Circle` instance.

SOURCE CODE:

```
abstract class Shape {
    protected int side1;
    protected int side2;
    public Shape(int side1, int side2) {
        this.side1 = side1;
        this.side2 = side2;
    }
    // Abstract method to be implemented by subclasses
    public abstract void printArea();
}

class Rectangle extends Shape {
    public Rectangle(int length, int width) {
        super(length, width);
    }
    @Override
    public void printArea() {
        int area = side1 * side2;
        System.out.println("Area of Rectangle: " + area);
    }
}

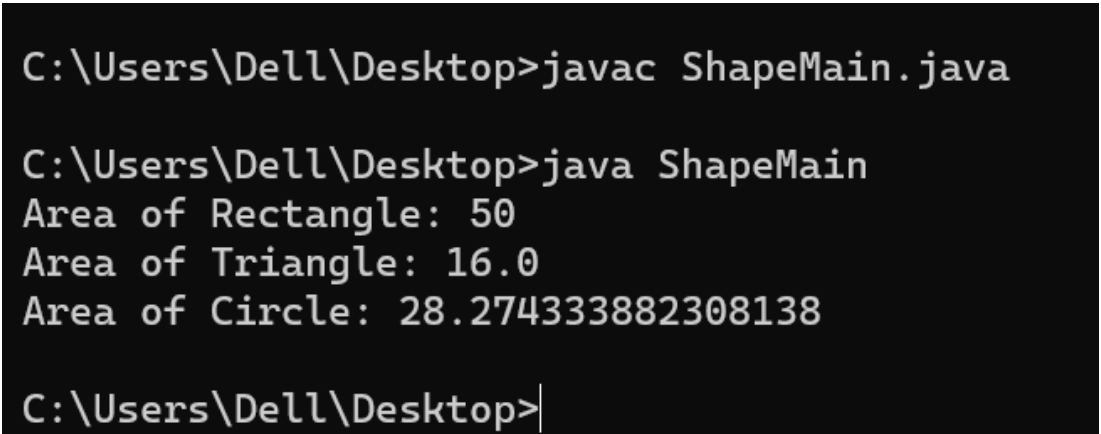
class Triangle extends Shape {
    public Triangle(int base, int height) {
        super(base, height);
    }
    @Override
    public void printArea() {
        double area = 0.5 * side1 * side2;
        System.out.println("Area of Triangle: " + area);
    }
}

class Circle extends Shape {
```



```
public Circle(int radius) {  
    super(radius, 0);  
}  
@Override  
public void printArea() {  
    double area = Math.PI * side1 * side1;  
    System.out.println("Area of Circle: " + area);  
}  
}  
public class ShapeMain {  
    public static void main(String[] args) {  
        Rectangle rectangle = new Rectangle(5, 10);  
        rectangle.printArea(); // Output: Area of Rectangle: 50  
        Triangle triangle = new Triangle(4, 8);  
        triangle.printArea(); // Output: Area of Triangle: 16.0  
        Circle circle = new Circle(3);  
        circle.printArea(); // Output: Area of Circle: 28.274333882308138  
    }  
}
```

OUTPUT:



```
C:\Users\Dell\Desktop>javac ShapeMain.java  
  
C:\Users\Dell\Desktop>java ShapeMain  
Area of Rectangle: 50  
Area of Triangle: 16.0  
Area of Circle: 28.274333882308138  
  
C:\Users\Dell\Desktop>|
```

Result:

Java program for printing the area of given shapes has been successfully executed.

Ex. No. 5. Solve the above problem using an interface.

Aim:

To develop a java program that prints the area of a given shape using an interface.

Algorithm:

- 1) Create an interface AreaCalculatable with a single abstract method printArea().
- 2) Define a Rectangle class that implements the AreaCalculatable interface:
 - a) Include private instance variables length and width.
 - b) Create a constructor to initialize these variables.
 - c) Implement the printArea() method to calculate and print the area of the rectangle using the formula $\text{area} = \text{length} * \text{width}$.
- 3) Define a Triangle class that also implements the AreaCalculatable interface:
 - a) Include private instance variables base and height.
 - b) Create a constructor to initialize these variables.
 - c) Implement the printArea() method to calculate and print the area of the triangle using the formula $\text{area} = 0.5 * \text{base} * \text{height}$.
- 4) Define a Circle class that implements the AreaCalculatable interface:
 - a) Include a private instance variable radius.
 - b) Create a constructor to initialize this variable.
 - c) Implement the printArea() method to calculate and print the area of the circle using the formula $\text{area} = \pi * \text{radius} * \text{radius}$.
- 5) Create a ShapeMain class with the main method:
 - a) Inside the main method, create instances of Rectangle, Triangle, and Circle, passing appropriate values to their constructors.
 - b) Call the printArea() method on each of these instances to calculate and print their respective areas.

SOURCE CODE:

```
// Interface for shapes that can calculate area
interface AreaCalculatable {
    void printArea();
}

// Rectangle class implementing AreaCalculatable
class Rectangle implements AreaCalculatable {
    private int length;
    private int width;

    public Rectangle(int length, int width) {
        this.length = length;
        this.width = width;
    }

    @Override
```

```
public void printArea() {
    int area = length * width;
    System.out.println("Area of Rectangle: " + area);
}
}

// Triangle class implementing AreaCalculatable
class Triangle implements AreaCalculatable {
    private int base;
    private int height;
    public Triangle(int base, int height) {
        this.base = base;
        this.height = height;
    }
    @Override
    public void printArea() {
        double area = 0.5 * base * height;
        System.out.println("Area of Triangle: " + area);
    }
}

// Circle class implementing AreaCalculatable
class Circle implements AreaCalculatable {
    private int radius;
    public Circle(int radius) {
        this.radius = radius;
    }
    @Override
    public void printArea() {
        double area = Math.PI * radius * radius;
        System.out.println("Area of Circle: " + area);
    }
}
```

```
public class ShapeMain {  
    public static void main(String[] args) {  
        Rectangle rectangle = new Rectangle(5, 10);  
        rectangle.printArea();  
        Triangle triangle = new Triangle(4, 8);  
        triangle.printArea();  
        Circle circle = new Circle(3);  
        circle.printArea();  
    }  
}
```

OUTPUT:

Area of Rectangle: 50

Area of Triangle: 16.0

Area of Circle: 28.274333882308138

Result:

Java program for printing the area of a given shape using an interface has been successfully executed.

Ex. No. 6. Implement exception handling and creation of user defined exceptions.

Aim: To implement exception handling and to create user defined exceptions in java.

Algorithm:

1. Create a Custom Exception Class
 - i) Create a custom exception class called NegativeNumberException by extending the Exception class.
 - ii) In the NegativeNumberException class, define a constructor that accepts a message and calls the superclass constructor with the message.
2. Implement Exception Handling
 - a) Create a class called ExceptionHandlingExample.
 - b) Inside the ExceptionHandlingExample class:
 - i) Define a method called calculateSquareRoot that accepts a double number.
 - ii) In the calculateSquareRoot method:
 - 2.1. Check if number is less than 0.
 - 2.2. If it is, throw a NegativeNumberException with a message "Cannot calculate square root of a negative number."
 - 2.3. If not, calculate the square root of number using Math.sqrt() and return the result.
 - 2.3.1. In the main method:
 - 2.4. Initialize a double variable number with a value (you can change it to test different scenarios).
 - 2.5. Use a try-catch block to handle exceptions:
 - 2.6. Try to calculate the square root using calculateSquareRoot(number).
 - 2.7. Catch a NegativeNumberException and display a custom error message with e.getMessage().
 - 2.8. Catch an ArithmeticException and display an error message for arithmetic issues.
 - 2.9. Catch a generic Exception and display an error message for unexpected exceptions.
 - 2.9.1. Add a finally block to indicate that execution has completed.
3. Compile and Run
 - i) Compile the Java program.
 - ii) Run the program and change the number value to test different scenarios.

SOURCE CODE:

```
class NegativeNumberException extends Exception {  
    public NegativeNumberException(String message) {  
        super(message);  
    }  
}  
  
public class ExceptionHandlingExample {  
    public static double calculateSquareRoot(double number) throws NegativeNumberException  
    {  
        if (number < 0) {  
            throw new NegativeNumberException("Cannot calculate square root of a negative number.");  
        }  
    }  
}
```

```
    }  
    return Math.sqrt(number);  
}  
public static void main(String[] args) {  
    double number = -5.0; // Change this value to test different scenarios  
    try {  
        double result = calculateSquareRoot(number);  
        System.out.println("Square root of " + number + " is: " + result);  
    }  
    catch (NegativeNumberException e) {  
        System.err.println("Error: " + e.getMessage());  
    }  
    catch (ArithmeticException e) {  
        System.err.println("Arithmetic error: " + e.getMessage());  
    }  
    catch (Exception e) {  
        System.err.println("An unexpected exception occurred: " + e.getMessage());  
    }  
    finally {  
        System.out.println("Execution completed.");  
    }  
}
```

Output:

```
C:\Users\Dell>cd desktop  
  
C:\Users\Dell\Desktop>javac ExceptionHandlingExample.java  
  
C:\Users\Dell\Desktop>java ExceptionHandlingExample  
Error: Cannot calculate square root of a negative number.  
Execution completed.  
  
C:\Users\Dell\Desktop>|
```

Result:

User defined exceptions has been successfully created in java.

Ex. No. 7. Write a java program that implements a multi-threaded application that has three threads. First thread generates a random integer every 1 second and if the value is even, the second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of the cube of the number.

Aim: To develop a java program that implements a multi-threaded application which has three threads.

Algorithm:

Step 1: Create Thread Classes

1.1. Create a class NumberGenerator implementing the Runnable interface.

1.2. In the NumberGenerator class:

- i. Initialize a Random object for generating random integers.
- ii. Accept a BlockingQueue as a parameter for communication.
- iii. Inside a continuous loop:
- iv. Generate a random integer between 0 and 9.
- v. Print the generated number.
- vi. Put the generated number into the BlockingQueue.
- vii. Sleep for 1 second to introduce a delay.

1.3. Create a class SquareCalculator implementing the Runnable interface.

1.4. In the SquareCalculator class:

- i. Accept a BlockingQueue as a parameter for communication.
- ii. Inside a continuous loop:
- iii. Take a number from the BlockingQueue.
- iv. Calculate the square of the number.
- v. Print the square result.

1.5. Create a class CubeCalculator implementing the Runnable interface.

1.6. In the CubeCalculator class:

- i. Accept a BlockingQueue as a parameter for communication.
- ii. Inside a continuous loop:
- iii. Take a number from the BlockingQueue.
- iv. Calculate the cube of the number.
- v. Print the cube result.

Step 2: Create the Main Class

2.1. Create a class called MultiThreadedNumberProcessor.

2.2. In the main method:

- i. Create a LinkedBlockingQueue named numberQueue to facilitate communication between threads.
- ii. Create three threads: generatorThread, squareThread, and cubeThread, passing appropriate instances of NumberGenerator, SquareCalculator, and CubeCalculator with the numberQueue.
- iii. Start all three threads.

Step 3: Run the Program

3.1. Compile the Java program.

3.2. Run the program.

3.3. Observe random numbers being generated, their squares being calculated and printed, and their cubes being calculated and printed in parallel.

SOURCE CODE:

```
import java.util.Random;

import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;

class NumberGenerator implements Runnable {
    private Random random = new Random();
    private BlockingQueue<Integer> numberQueue;

    public NumberGenerator(BlockingQueue<Integer> numberQueue) {
        this.numberQueue = numberQueue;
    }

    @Override
    public void run() {
        try {
            while (true) {
                int randomNumber = random.nextInt(10); // Generate a random integer between 0 and 9
                System.out.println("Generated number: " + randomNumber);
                numberQueue.put(randomNumber);
                Thread.sleep(1000); // Sleep for 1 second
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

class SquareCalculator implements Runnable {
    private BlockingQueue<Integer> numberQueue;

    public SquareCalculator(BlockingQueue<Integer> numberQueue) {
        this.numberQueue = numberQueue;
    }
}
```

```

    }

    @Override
    public void run() {
        try {
            while (true) {
                int number = numberQueue.take();
                int square = number * number;
                System.out.println("Square of " + number + " is: " + square);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

class CubeCalculator implements Runnable {
    private BlockingQueue<Integer> numberQueue;

    public CubeCalculator(BlockingQueue<Integer> numberQueue) {
        this.numberQueue = numberQueue;
    }

    @Override
    public void run() {
        try {
            while (true) {
                int number = numberQueue.take();
                int cube = number * number * number;
                System.out.println("Cube of " + number + " is: " + cube);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class MultiThreadedNumberProcessor {
    public static void main(String[] args) {
        BlockingQueue<Integer> numberQueue = new LinkedBlockingQueue<>();
        Thread generatorThread = new Thread(new NumberGenerator(numberQueue));
    }
}

```

```
Thread squareThread = new Thread(new SquareCalculator(numberQueue));
Thread cubeThread = new Thread(new CubeCalculator(numberQueue));
generatorThread.start();
squareThread.start();
cubeThread.start();
}
}
```

Output:

```
C:\Users\Dell>cd desktop
C:\Users\Dell\Desktop>javac MultiThreadedNumberProcessor.java
C:\Users\Dell\Desktop>java MultiThreadedNumberProcessor
Generated number: 1
Square of 1 is: 1
Generated number: 5
Cube of 5 is: 125
Generated number: 8
Square of 8 is: 64
Generated number: 9
Cube of 9 is: 729
Generated number: 3
Square of 3 is: 9
Generated number: 6
Cube of 6 is: 216
Generated number: 8
Square of 8 is: 64
Generated number: 7
Cube of 7 is: 343
Generated number: 7
Square of 7 is: 49
Generated number: 0
Cube of 0 is: 0
Generated number: 2
Square of 2 is: 4
Generated number: 9
Cube of 9 is: 729
```

Result:

Java program to implement a multi-threaded application which has three threads has been successfully executed.

Ex. No. 8. Write a program to perform file operations.

Aim: To write a java program that perform file operations.

Algorithm:

Step 1: Specify the File Path

1.1. Define a String variable filePath to specify the path of the file (e.g., "sample.txt").

Step 2: Create and Write to a File

2.1. Create a method writeToFile(filePath, content):

- i. Accept the filePath and content to write as parameters.
- ii. Create a FileWriter object with the specified filePath.
- iii. Use the FileWriter to write the content to the file.
- iv. Close the FileWriter to save the changes.
- v. Print a success message.

Step 3: Read and Display the Contents of the File

3.1. Create a method readFromFile(filePath):

- i. Accept the filePath as a parameter.
- ii. Create a FileReader object with the specified filePath.
- iii. Initialize a StringBuilder to store the file contents.
- iv. Read each character from the FileReader and append it to the StringBuilder until the end of the file.
- v. Close the FileReader.
- vi. Return the contents as a String.

Step 4: Append Additional Content to the File

4.1. Create a method appendToFile(filePath, content):

- i. Accept the filePath and content to append as parameters.
- ii. Create a FileWriter object with the specified filePath in append mode (use true as the second parameter).
- iii. Use the FileWriter to append the content to the file.
- iv. Close the FileWriter to save the changes.
- v. Print a success message.

Step 5: Delete the File

5.1. Create a method deleteFile(filePath):

- i. Accept the filePath as a parameter.
- ii. Create a File object with the specified filePath.
- iii. Use the delete method on the File object to delete the file.
- iv. Print a success message if the deletion is successful, or an error message if it fails.

Step 6: Main Program

6.1. Create a main method:

- i. Call the `writeToFile` method to create and write to a file.
- ii. Call the `readFromFile` method to read and display the contents of the file.
- iii. Call the `appendToFile` method to append additional content to the file.
- iv. Call the `deleteFile` method to delete the file.

Step 7: Run the Program

7.1. Compile the Java program.

7.2. Run the program to perform the specified file operations.

SOURCE CODE:

```
import java.io.*;

public class FileOperationsExample {
    public static void main(String[] args) {
        // Specify the file path
        String filePath = "sample.txt";

        // Create and write to a file
        writeToFile(filePath, "Hello, World!");

        // Read and display the contents of the file
        String fileContents = readFromFile(filePath);
        System.out.println("File Contents: " + fileContents);

        // Append additional content to the file
        appendToFile(filePath, "\nThis is an appended line.");

        // Read and display the updated contents of the file
        fileContents = readFromFile(filePath);
        System.out.println("Updated File Contents: " + fileContents);

        // Delete the file
        deleteFile(filePath);
    }

    // Method to write content to a file
    public static void writeToFile(String filePath, String content) {
        try {
            FileWriter writer = new FileWriter(filePath);
            writer.write(content);
            writer.close();

            System.out.println("File created and written successfully.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
    } catch (IOException e) {
        e.printStackTrace();
    }
}

// Method to read content from a file
public static String readFromFile(String filePath) {
    StringBuilder content = new StringBuilder();
    try {
        FileReader reader = new FileReader(filePath);
        int character;
        while ((character = reader.read()) != -1) {
            content.append((char) character);
        }
        reader.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return content.toString();
}

// Method to append content to an existing file
public static void appendToFile(String filePath, String content) {
    try {
        FileWriter writer = new FileWriter(filePath, true); // true for appending
        writer.write(content);
        writer.close();
        System.out.println("Content appended to the file.");
    } catch (IOException e) {
        e.printStackTrace();
    }
}

// Method to delete a file
public static void deleteFile(String filePath) {
```

```
File file = new File(filePath);  
if (file.delete()) {  
    System.out.println("File deleted successfully.");  
} else {  
    System.out.println("Failed to delete the file.");  
} }}
```

Output:

```
C:\Users\Dell\Desktop>javac FileOperationsExample.java  
  
C:\Users\Dell\Desktop>java FileOperationsExample  
File created and written successfully.  
File Contents: Hello, World!  
Content appended to the file.  
Updated File Contents: Hello, World!  
This is an appended line.  
File deleted successfully.  
  
C:\Users\Dell\Desktop>
```

Result:

A java program that performs file operations has been successfully implemented.

Ex. No. 9. Develop an application to demonstrate the features of generics classes.

Aim: To develop a java application to demonstrate the features of generic classes.

Algorithm:

Step 1: Create a Generic Class

1.1. Define a generic class `MyGenericContainer<T>` that takes a type parameter `T`.

1.2. Inside the `MyGenericContainer` class:

- i. Declare a private data member `data` of type `T` to store the generic data.
- ii. Create a constructor that accepts an initial value of type `T` and initializes the data member.
- iii. Implement getter and setter methods to access and modify the data member.
- iv. Create a method `displayType` to display the type of the stored data.

Step 2: Create an Application Class

2.1. Create a public class named `GenericClassDemo`.

2.2. In the main method:

- i. Create an instance of `MyGenericContainer` for `Integer` and initialize it with an integer value (e.g., 42).
- ii. Print the stored integer data and display its type.
- iii. Create an instance of `MyGenericContainer` for `String` and initialize it with a string value (e.g., "Hello, Generics!").
- iv. Print the stored string data and display its type.
- v. Create an instance of `MyGenericContainer` for `Double` and initialize it with a double value (e.g., 3.14).
- vi. Print the stored double data and display its type.

Step 3: Run the Program

3.1. Compile the Java program.

3.2. Run the program to create generic containers with different data types (`Integer`, `String`, and `Double`) and display their values and types.

SOURCE CODE:

```
class MyGenericContainer<T> {  
    private T data;  
    public MyGenericContainer(T data) {  
        this.data = data;  
    }  
    public T getData() {  
        return data;  
    }  
    public void setData(T data) {
```



```

        this.data = data;
    }
    public void displayType() {
        System.out.println("Type of data: " + data.getClass().getName());
    }
}

public class GenericClassDemo {
    public static void main(String[] args) {
        // Create a generic container for an Integer
        MyGenericContainer<Integer> intContainer = new MyGenericContainer<>(42);
        System.out.println("Integer data: " + intContainer.getData());
        intContainer.displayType();
        // Create a generic container for a String
        MyGenericContainer<String> strContainer = new MyGenericContainer<>("Hello, Generics!");
        System.out.println("String data: " + strContainer.getData());
        strContainer.displayType();
        // Create a generic container for a Double
        MyGenericContainer<Double> doubleContainer = new MyGenericContainer<>(3.14);
        System.out.println("Double data: " + doubleContainer.getData());
        doubleContainer.displayType();
    }
}

```

Output:

```

C:\Users\Dell\Desktop>javac GenericClassDemo.java

C:\Users\Dell\Desktop>java GenericClassDemo
Integer data: 42
Type of data: java.lang.Integer
String data: Hello, Generics!
Type of data: java.lang.String
Double data: 3.14
Type of data: java.lang.Double

C:\Users\Dell\Desktop>

```

Result:

The java application using generic classes has been created successfully.

Ex. No. 10. Develop a mini project for any application using Java concepts.

Aim: To develop a library management system using java concepts.

Algorithm:

1. Create a class named "Book" with private fields:

- title (String)
- author (String)
- isAvailable (boolean)

Implement the following methods for the "Book" class:

- Constructor: Initialize title, author, and isAvailable.
- getTitle(): Get the title of the book.
- getAuthor(): Get the author of the book.
- isAvailable(): Check if the book is available.
- borrowBook(): Set isAvailable to false.
- returnBook(): Set isAvailable to true.
- toString(): Override to display book details.

2. Create a class named "Library" with a private list of books (List<Book>).

Implement the following methods for the "Library" class:

- Constructor: Initialize the list of books.
- addBook(Book book): Add a book to the library.
- listBooks(): List all books in the library.
- borrowBook(int bookIndex): Borrow a book by index.
- returnBook(int bookIndex): Return a book by index.

3. Create a public class named "LibraryManagementSystem" with the main method:

- Create a Library object.
- Create a Scanner object for user input.
- Enter a loop to display a menu and perform actions until the user chooses to exit.
- Display a menu with the following options:
 1. Add Book
 2. List Books
 3. Borrow Book
 4. Return Book
 5. Exit

- Read the user's choice.
- Based on the choice:
 - If 1:
 - Prompt the user for book title and author.
 - Create a new Book object and add it to the library.
 - If 2:
 - List all books in the library.
 - If 3:
 - Prompt the user for the index of the book to borrow.
 - Borrow the selected book if it's available.
 - If 4:
 - Prompt the user for the index of the book to return.
 - Return the selected book if it's borrowed.
 - If 5:
 - Exit the program with a goodbye message.
- If the choice is invalid, display an error message.
- End the loop when the user chooses to exit.

4. Run the program.

SOURCE CODE:

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

class Book {
    private String title;
    private String author;
    private boolean isAvailable;
    public Book(String title, String author) {
        this.title = title;
        this.author = author;
        this.isAvailable = true;
    }
    public String getTitle() {
```

```

        return title;
    }
    public String getAuthor() {
        return author;
    }
    public boolean isAvailable() {
        return isAvailable;
    }
    public void borrowBook() {
        isAvailable = false;
    }
    public void returnBook() {
        isAvailable = true;
    }
    @Override
    public String toString() {
        return "Title: " + title + ", Author: " + author + ", Available: " + (isAvailable ? "Yes" :
"No");
    }
}

class Library {
    private List<Book> books;
    public Library() {
        books = new ArrayList<>();
    }
    public void addBook(Book book) {
        books.add(book);
    }

    public void listBooks() {
        if (books.isEmpty()) {
            System.out.println("No books in the library.");
        }
    }
}

```

```

    } else {
        System.out.println("Books in the library:");
        for (Book book : books) {
            System.out.println(book);
        }
    }
}

public void borrowBook(int bookIndex) {
    if (bookIndex >= 0 && bookIndex < books.size()) {
        Book book = books.get(bookIndex);
        if (book.isAvailable()) {
            book.borrowBook();
            System.out.println("You have borrowed the book: " + book.getTitle());
        } else {
            System.out.println("Sorry, the book is already borrowed.");
        }
    } else {
        System.out.println("Invalid book index.");
    }
}

public void returnBook(int bookIndex) {
    if (bookIndex >= 0 && bookIndex < books.size()) {
        Book book = books.get(bookIndex);
        if (!book.isAvailable()) {
            book.returnBook();
            System.out.println("You have returned the book: " + book.getTitle());
        } else {
            System.out.println("The book is already available in the library.");
        }
    } else {
        System.out.println("Invalid book index.");
    }
}

```



```

        library.borrowBook(borrowIndex);

        break;

    case 4:

        System.out.print("Enter the index of the book to return: ");

        int returnIndex = scanner.nextInt();

        library.returnBook(returnIndex);

        break;

    case 5:

        System.out.println("Exiting Library Management System. Goodbye!");

        System.exit(0);

    default:

        System.out.println("Invalid choice. Please enter a valid option.");

    }
}
}}

```

Output:

```
C:\Users\Dell\Desktop>javac LibraryManagementSystem.java
```

```
C:\Users\Dell\Desktop>java LibraryManagementSystem
```

```
Library Management System Menu:
```

1. Add Book
2. List Books
3. Borrow Book
4. Return Book
5. Exit

```
Enter your choice: 1
```

```
Enter book title: 2 States
```

```
Enter author name: Chetan Bhagat
```

```
Book added successfully.
```

```
Library Management System Menu:
```

1. Add Book
2. List Books
3. Borrow Book
4. Return Book
5. Exit

```
Enter your choice: 3
```

```
Enter the index of the book to borrow: 1
```

```
Invalid book index.
```

```
Library Management System Menu:
```

1. Add Book
2. List Books
3. Borrow Book
4. Return Book
5. Exit

```
Enter your choice: |
```

Result:

A mini project on library management system using java concepts has been implemented successfully.



