Implement binary search tree and perform following operations:

Insert, b. Delete, c. Display (Inorder) d. Search e. BFS (Level wise print)

```
import java.util.*;

public class BST

{

static class Node{

int data;

Node left;

Node right;

Node(int data){

this.data = data;

}

}

public static Node insert(Node root, int val)

{

if(root == null){

root = new Node(val);

return root;

}

if(root.data&gt; val){

//left subtree

root.left = insert(root.left, val);

}

else {

root.right =insert(root.right, val);

}

return root;

}

public static void inorder(Node root){

if(root == null){

return ;
```

```
}
inorder(root.left);
System.out.print(root.data+ &quot; &quot;);
inorder(root.right);
}
public static boolean search(Node root, int key){
if(root == null){
return false;
}
if(root.data &gt; key){ //let subtree
return search(root.left,key);

}
else if(root.data &lt; key){
return search(root.right, key);
}
else if(root.data == key){
return true;
}
return false;
}
public static Node delete(Node root, int val){
if(root.data &gt; val){
root.left = delete(root.left, val);
}
else if(root.data &lt; val){
root.right = delete(root.right, val);
}
else{ //root.data == val
// case 1
if(root.left == null &amp;&amp; root.right == null){
```

```
return null;
}
// case 2
else if(root.left == null){
return root.right;
}
else if(root.right == null){
return root.left;
}
// case 3
else {
root.data = minValue(root.right);
root.right = delete(root.right, root.data );
}
}
return root;
}

static int minValue(Node root) {
int minval = root.data;
//find minval
while (root.left != null) {
minval = root.left.data;
root = root.left;
}
return minval;
}

public static void printLevelOrder(Node data)
{
Queue&lt;Node&gt; queue = new
```

```
LinkedList&lt;Node&gt;();

queue.add(data);

while (!queue.isEmpty()) {

// poll() removes the present head.

Node tempNode = queue.poll();

System.out.print(tempNode.data + &quot; &quot;);

// Enqueue left child

if (tempNode.left != null) {

queue.add(tempNode.left);

}

if (tempNode.right != null) {

queue.add(tempNode.right);

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n;

Node root = null;

int a[] = new int[100];

int ch;

do {

System.out.println(&quot;Enter your Choice\n1.Insert \n2.Delete \n3.search \n4.Display
\n5.BFS&quot;);

ch = sc.nextInt();

switch (ch) {

case 1:

System.out.println(&quot;enter no. of elements&quot;);

n = sc.nextInt();

for (int i = 0; i &lt; n; i++) {

a[i] = sc.nextInt();
root = insert(root, a[i]);
}
break;
case 2:
System.out.println(&quot;enter the value that you have to delete&quot;);
int val = sc.nextInt();
delete(root, val);
break;
case 3:
System.out.println(&quot;enter the value that you have to search&quot;);
int ser = sc.nextInt();

if (search(root, ser)) {
System.out.println(ser + &quot; is Found&quot;);
} else {
System.out.println(&quot;Value not found&quot;);
}
break;
case 4:
System.out.println(&quot;Display inorder Traversal&quot;);
inorder(root);
break;
case 5:
System.out.println(&quot;Level order Traversal-Breadth First Search&quot;);
printLevelOrder(root);
break;
default:
System.out.println(&quot;Something went wrong&quot;);
}
}
while (ch != 6);
```

```
}
}
```

Implement Circular linked list for employee data and perform the various operations:

a. Inserting At Beginning of the list

b. Inserting At End of the list

c. Inserting At Specific location in the list

d. Deleting from Beginning of the list

e. Deleting from End of the list

f. Deleting a Specific Node

g. Display

```
import java.util.Scanner;

public class CircularDoublyLinkedList {
// Circular doubly linked list
public static class Node {
int data;
Node next;
Node prev;
public Node(int data) {
this.data = data;
this.next = null;
this.prev = null;
}
}
public static Node head;
public static Node tail;
public static int size;
public static void addFirst(int data) {
Node newNode = new Node(data);
size++;

if (head == null) {

head = tail = newNode;

head.next = head;

head.prev = head;

} else {

newNode.next = head;

newNode.prev = tail;

head.prev = newNode;

tail.next = newNode;

head = newNode;

}

print();

}

public static void addLast(int data) {

Node newNode = new Node(data);

size++;

if (head == null) {

head = tail = newNode;

head.next = head;

head.prev = head;

} else {

newNode.next = head;

newNode.prev = tail;

tail.next = newNode;

head.prev = newNode;

tail = newNode;

}

print();

}
```

```
public static void addMiddle(int data, int idx) {
if (idx == 1) {
addFirst(data);
}
else if(idx&gt;size){
System.out.println(&quot;Invalid index&quot;);
}
else {
Node newNode = new Node(data);
Node temp = head;
int i = 1;
while (i &lt; idx - 1 &amp;&amp; temp != null) {
temp = temp.next;
i++;
}
if (temp != null ) {
newNode.next = temp.next;
newNode.prev = temp;
temp.next.prev = newNode;
temp.next = newNode;
size++;
}
else {
System.out.println(&quot;Invalid index&quot;);
}
print();
}
}
public static int delFirst() {
if (head == null) {
System.out.println(&quot;Linked List is empty&quot;);
```

```
print();
return -1;
}
if (head == tail) {
int val = head.data;
head = tail = null; // Only one element in the list
size = 0 ;
print();
return val;
} else {
int val = head.data;
head = head.next;
tail.next = head;
head.prev = tail;
size--;
print();
return val;
}
}
public static int delLast() {

if (head == null) {
System.out.println(&quot;Linked List is empty&quot;);
print();
return -1 ;
}
if (head == tail) {
int val = tail.data;
head = tail = null; // Only one element in the list
size = 0 ;
print();
```

```
return val;
} else {
int val = tail.data;
tail = tail.prev;
tail.next = head;
head.prev = tail;
size--;
print();
return val;
}
}
public static int delVal(int idx) {
if (idx == 1) {
return delFirst();
}
if (idx &gt; size) {
System.out.println(&quot;Incorrect Index&quot;);
return -1;
}
Node temp = head;
int i = 1;
while (i &lt; idx - 1 &amp;&amp; temp.next != head) {
temp = temp.next;
i++;
}
if (temp.next == tail) {
return delLast();
}
Node toDelete = temp.next;
int val = toDelete.data;
temp.next = toDelete.next;
```

```
toDelete.next.prev = temp;
size--;
print();
return val;
}

public static boolean search(int value){
Node temp = head;
do {
if (temp.data == value){
return true;
}
temp = temp.next;
} while (temp != head);
return false;
}
public static void modify(int data, int repl) {
if (head == null) {
System.out.println(&quot;Linked List is empty&quot;);
return;
}
Node temp = head;
boolean found = false;
do {
if (temp.data == data) {
temp.data = repl;
found = true;
System.out.println(data + &quot; replaced by &quot;+ repl);
break;
}
temp = temp.next;
```

```
}
while (temp != head);
print();
if (!found) {
System.out.println(&quot;Value Not Found&quot;);
}
}
public static void print() {
if (head == null) {
System.out.println(&quot;Linked List is empty&quot;);
return;
}
Node temp = head;
do {
System.out.print(temp.data + &quot; --&gt; &quot;);
temp = temp.next;
} while (temp != head);
System.out.println(&quot;head&quot;);
}
public static void main(String[] args) {
CircularDoublyLinkedList l1 = new CircularDoublyLinkedList();
Scanner sc = new Scanner(System.in);
System.out.println(&quot;1.addFirst \n2.addLast \n3.addMiddle \n4.DelFirst \n5.DelLast
\n6.DelValue
\n7.Search \n8.Modify \n9.Print Size \n10.Print \n11.Exit&quot;);
int choice =0 ;

while (choice!=11) {
System.out.print(&quot;Enter the Choice : &quot;);
choice = sc.nextInt();
switch (choice) {
```

```
case 1: {

System.out.print(&quot;Enter the value to Insert : &quot;);

int data = sc.nextInt();

addFirst(data);

break;

}

case 2: {

System.out.print(&quot;Enter the value to Insert : &quot;);

int data = sc.nextInt();

addLast(data);

break;

}

case 3: {

System.out.print(&quot;Enter the value to Insert : &quot;);

int data = sc.nextInt();

System.out.print(&quot;Enter the index of value : &quot;);

int idx = sc.nextInt();

addMiddle(data,idx);

break;

}

case 4: {

System.out.println(&quot;Deleted : &quot; + delFirst());

break;

}

case 5: {

System.out.println(&quot;Deleted : &quot; + delLast());

break;

}

case 6: {

System.out.print(&quot;Enter the index of value to Delete : &quot;);

int val = sc.nextInt();
System.out.println(&quot;Deleted : &quot; +delVal(val));
break;
}
case 7: {
System.out.print(&quot;Enter the value to Search : &quot;);
int val = sc.nextInt();
if (search(val)==true){
System.out.println(&quot;Key Found&quot;);
}
else {
System.out.println(&quot;Not found&quot;);
}
break;
}
case 8 : {
System.out.print(&quot;Enter the value to be replaced : &quot; );
int data = sc.nextInt();

System.out.print(&quot;Enter the value to be replaced by &quot; + data +&quot; : &quot;);
int repl = sc.nextInt();
modify(data,repl);
break;
}
case 9 :{
System.out.println(&quot;Size : &quot;+size );
break;
}
case 10: {
print();
break;
}
```

```
default:{
System.out.println(&quot;enter the correct choice&quot;);
}
}
}
}
}
```

Create a Singly linked list for employee data and perform

a. insertion b. deletion c. search d. modify

```
import java.util.Scanner;
public class Lab1_LinkedList {
public static class Node {
int data;
Node next;
public Node(int data) {
this.data = data;
this.next = null;
}
}
public static Node head;
public static int size;
public static void addFirst(int data) {
Node newNode = new Node(data);
if (head == null) {
head = newNode;
size++;
}
else {
newNode.next = head;
head = newNode;
size++;
```

```
}
print();
}
public static void addLast(int data) {
Node newNode = new Node(data);
size++;
if (head == null) {
head = newNode;
} else {
Node temp = head;
while (temp.next != null) {
temp = temp.next;
}
temp.next = newNode;
}
print();
}
public static void addMiddle(int idx, int data) {
if (idx == 0) {
addFirst(data);
} else {
Node newNode = new Node(data);
size++;

Node temp = head;
int i = 0;
while (i &lt; idx - 1 &amp;&amp; temp != null) {
temp = temp.next;
i++;
}
if (temp != null) {
```

```
newNode.next = temp.next;
temp.next = newNode;
}
}
print();
}
public static int delFirst() {
if (head == null) {
System.out.println(&quot;List is Empty&quot;);
return -1;
} else {
int val = head.data;
head = head.next;
size--;
print();
return val;
}
}
public static int delLast() {
if (head == null) {
System.out.println(&quot;LL is empty&quot;);
return -1;
} else if (head.next == null) {
int val = head.data;
head = null;
size =0;
return val;
} else {
Node prev = head;
Node last = head.next;
while (last.next != null) {
```

```
last = last.next;
prev = prev.next;
}
int val = last.data;
prev.next = null;
size--;
print();
return val;
}
}
public static void delVal(int val) {
if (head == null) {
System.out.println(&quot;List is Empty&quot;);
return;

}
if (head.data == val) {
head = head.next;
size--;
return;
}
Node temp = head;
while (temp != null &amp;&amp; temp.next != null) {
if (temp.next.data == val) {
// Node valueDel = temp.next;
temp.next = temp.next.next; // we can only use this single statment instead of this 3
statment because deleted Node can be handled by garbage collector
//valueDel.next =null; //therefore it will not need to add one more Node
size--;
return;
}
```

```
temp = temp.next;
}
print();
}
public static void print() {
Node temp = head;
while (temp != null) {
System.out.print(temp.data + &quot;--&gt;&quot;);
temp = temp.next;
}
System.out.println(&quot;null&quot;);
}
public static boolean Search(int val){
Node temp = head ;
while (temp!=null){
if (temp.data == val){
return true;
}
temp = temp.next;
}
return false;
}
public static void Modify(int data , int repl){
Node temp = head;
while (temp!= null){
if (temp.data == data){
temp.data = repl;
print();
}
else {
System.out.println(&quot;Value Not found&quot;);
```

```
}

temp = temp.next;

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Lab1_LinkedList ll = new Lab1_LinkedList();


System.out.println(&quot;1.addFirst \n2.addLast \n3.addMiddle \n4.DelFirst \n5.DelLast
\n6.DelValue

\n7.Search \n8.Print \n9.Modify \n10.print Size \n11.Exit&quot;);

int choice =0 ;

while (choice!=11) {

System.out.print(&quot;Enter the Choice : &quot;);

choice = sc.nextInt();

switch (choice) {

case 1: {

System.out.print(&quot;Enter the value to Insert : &quot;);

int data = sc.nextInt();

ll.addFirst(data);

break;

}

case 2: {

System.out.print(&quot;Enter the value to Insert : &quot;);

int data = sc.nextInt();

ll.addLast(data);

break;

}

case 3: {

System.out.print(&quot;Enter the value to Insert : &quot;);

int data = sc.nextInt();
System.out.print(&quot;Enter the index of value : &quot;);
int idx = sc.nextInt();
ll.addMiddle(idx, data);
break;
}
case 4: {
System.out.println(&quot;Deleted : &quot; + ll.delFirst());
break;
}
case 5: {
System.out.println(&quot;Deleted : &quot; + ll.delLast());
break;
}
case 6: {
System.out.print(&quot;Enter the value to Delete : &quot;);
int val = sc.nextInt();
ll.delVal(val);
break;
}
case 7: {
System.out.print(&quot;Enter the value to Search : &quot;);
int val = sc.nextInt();
Search(val);
if (Search(val)==true){
System.out.println(&quot;Key Found&quot;);
}
else {
System.out.println(&quot;Not found&quot;);
}
break;
}
```

```
case 8: {
ll.print();
break;


}
case 9 : {
System.out.print(&quot;Enter the value to be replaced : &quot; );
int data = sc.nextInt();
System.out.print(&quot;Enter the value to be replaced by : &quot; + data );
int repl = sc.nextInt();
ll.Modify(data,repl);
break;
}
case 10 :{
System.out.println(&quot;Size : &quot;+size );
break;
}
default:{
System.out.println(&quot;enter the correct choice&quot;);
}
}
}
}
}
```

Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject. Use heap data structure. Analyze the algorithm.

```
import java.util.Scanner;


public class MarksHeap {
```

```
// Method to insert an element into a min-heap
public static void insertMinHeap(int[] heap, int n, int value) {
    heap[n] = value;
    int i = n;
    while (i > 0 && heap[(i - 1) / 2] > heap[i]) {
        // Swap heap[i] with heap[(i - 1) / 2]
        int temp = heap[i];
        heap[i] = heap[(i - 1) / 2];
        heap[(i - 1) / 2] = temp;
        i = (i - 1) / 2;
    }
}

// Method to insert an element into a max-heap
public static void insertMaxHeap(int[] heap, int n, int value) {
    heap[n] = value;
    int i = n;
    while (i > 0 && heap[(i - 1) / 2] < heap[i]) {
        // Swap heap[i] with heap[(i - 1) / 2]
        int temp = heap[i];
        heap[i] = heap[(i - 1) / 2];
        heap[(i - 1) / 2] = temp;
        i = (i - 1) / 2;
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the number of students: ");
    int n = scanner.nextInt();
```

```
        int[] minHeap = new int[n];
        int[] maxHeap = new int[n];

        System.out.println("Enter the marks obtained by the students:");
        for (int i = 0; i < n; i++) {
            int mark = scanner.nextInt();
            insertMinHeap(minHeap, i, mark);
            insertMaxHeap(maxHeap, i, mark);
        }

        int minMark = minHeap[0];  // The root of the min-heap
        int maxMark = maxHeap[0];  // The root of the max-heap

        System.out.println("Minimum mark: " + minMark);
        System.out.println("Maximum mark: " + maxMark);
    }
}
```

Implement IsSpell utility (1. Create Hashtable and write in file 2. Enter Word 3. Search word 4. Exit)

```
import java.io.*;
import java.util.*;
public class IsSpellUtility {
public static void main(String[] args) {
Hashtable&lt;String, Integer&gt; dictionary = new Hashtable&lt;&gt;();
Scanner scanner = new Scanner(System.in);
// Load dictionary from file
loadDictionary(dictionary);
while (true) {
System.out.println(&quot;1. Enter Word&quot;);
System.out.println(&quot;2. Search Word&quot;);
System.out.println(&quot;3. Remove Word&quot;);
System.out.println(&quot;4. Exit&quot;);
System.out.print(&quot;Enter your choice: &quot;);
int choice = scanner.nextInt();
scanner.nextLine(); // Consume newline
switch (choice) {
case 1:
System.out.print(&quot;Enter the word to add to dictionary: &quot;);
String wordToAdd = scanner.nextLine();
addWordToDictionary(dictionary, wordToAdd);
saveDictionaryToFile(dictionary);
break;
case 2:
System.out.print(&quot;Enter the word to search in dictionary: &quot;);
String wordToSearch = scanner.nextLine();
searchWordInDictionary(dictionary, wordToSearch);
break;
case 3:
System.out.print(&quot;Enter the word to remove from dictionary: &quot;);
String wordToRemove = scanner.nextLine();
removeWordFromDictionary(dictionary, wordToRemove);
saveDictionaryToFile(dictionary);
break;
case 4:
System.out.println(&quot;Exiting program...&quot;);
System.exit(0);
break;
default:
System.out.println(&quot;Invalid choice. Please enter a valid option.&quot;);
}
}
}
```

```
private static void loadDictionary(Hashtable&lt;String, Integer&gt; dictionary) {
try (BufferedReader reader = new BufferedReader(new FileReader(&quot;dictionary.txt&quot;))) {

String line;
while ((line = reader.readLine()) != null) {
dictionary.put(line.trim().toLowerCase(), 1);
}
} catch (IOException e) {
System.out.println(&quot;Error loading dictionary: &quot; + e.getMessage());
}
}
private static void addWordToDictionary(Hashtable&lt;String, Integer&gt; dictionary, String word) {
dictionary.put(word.trim().toLowerCase(), 1);
System.out.println(&quot;Word added to dictionary.&quot;);
}
private static void searchWordInDictionary(Hashtable&lt;String, Integer&gt; dictionary, String word)
{
if (dictionary.containsKey(word.trim().toLowerCase())) {
System.out.println(&quot;Word found in dictionary.&quot;);
} else {
System.out.println(&quot;Word not found in dictionary.&quot;);
}
}
private static void removeWordFromDictionary(Hashtable&lt;String, Integer&gt; dictionary, String
word) {
if (dictionary.containsKey(word.trim().toLowerCase())) {
dictionary.remove(word.trim().toLowerCase());
System.out.println(&quot;Word removed from dictionary.&quot;);
} else {
System.out.println(&quot;Word not found in dictionary.&quot;);
}
}
```

```
private static void saveDictionaryToFile(Hashtable&lt;String, Integer&gt; dictionary) {
try (BufferedWriter writer = new BufferedWriter(new FileWriter(&quot;dictionary.txt&quot;))) {
for (String word : dictionary.keySet()) {
writer.write(word);
writer.newLine();
}
} catch (IOException e) {
System.out.println(&quot;Error saving dictionary to file: &quot; + e.getMessage());
}
}
}
```

A customer wants to travel from source A to destination B. He books a cab from source A to reach destination B. Calculate a shortest path by avoiding real time traffic to reach destination B.

```
import java.util.Arrays;


class Graph {
private int vertices;
private int[][] adjacencyMatrix;
// Constructor
public Graph(int vertices) {
this.vertices = vertices;
adjacencyMatrix = new int[vertices][vertices];
// Initialize all edges to infinity (no connection) except for self-loops
for (int i = 0; i &lt; vertices; i++) {
Arrays.fill(adjacencyMatrix[i], Integer.MAX_VALUE);
adjacencyMatrix[i][i] = 0;
}
}
// Adding edges to the graph
public void addEdge(int src, int dest, int weight) {
```

```
adjacencyMatrix[src][dest] = weight;
adjacencyMatrix[dest][src] = weight; // If the graph is undirected
}
// Dijkstra&#39;s algorithm to find the shortest path
public void dijkstra(int source) {
int[] distances = new int[vertices];
boolean[] visited = new boolean[vertices];
// Initialize distances as infinity and visited as false
Arrays.fill(distances, Integer.MAX_VALUE);
distances[source] = 0; // Distance to the source itself is 0
// Loop to find the shortest path for all vertices
for (int i = 0; i &lt; vertices - 1; i++) {
// Find the unvisited vertex with the smallest distance
int u = getMinDistanceVertex(distances, visited);
// Mark the picked vertex as visited
visited[u] = true;
// Update distances of adjacent vertices of the picked vertex
for (int v = 0; v &lt; vertices; v++) {
// Update distances[v] if:
// 1. v is not visited
// 2. There is an edge from u to v
// 3. Total weight of path from source to v through u is smaller than the current value of
distances[v]
if (!visited[v] &amp;&amp; adjacencyMatrix[u][v] != Integer.MAX_VALUE &amp;&amp;
distances[u] != Integer.MAX_VALUE &amp;&amp;
distances[u] + adjacencyMatrix[u][v] &lt; distances[v]) {
distances[v] = distances[u] + adjacencyMatrix[u][v];
}
}
}
```

```
// Print the shortest distances from the source
printSolution(distances, source);
}
// Helper method to find the vertex with the minimum distance value
private int getMinDistanceVertex(int[] distances, boolean[] visited) {
int minDistance = Integer.MAX_VALUE;
int minIndex = -1;
for (int v = 0; v &lt; vertices; v++) {
if (!visited[v] &amp;&amp; distances[v] &lt; minDistance) {
minDistance = distances[v];
minIndex = v;
}
}
return minIndex;
}
// Helper method to print the solution
private void printSolution(int[] distances, int source) {
System.out.println(&quot;Shortest paths from node &quot; + source + &quot;:&quot;);
for (int i = 0; i &lt; distances.length; i++) {
System.out.println(&quot;To node &quot; + i + &quot; is &quot; + distances[i]);
}
}
public static void main(String[] args) {
Graph graph = new Graph(6);
graph.addEdge(0, 1, 4);
graph.addEdge(0, 2, 1);
graph.addEdge(2, 1, 2);
graph.addEdge(1, 3, 1);
graph.addEdge(2, 3, 5);
graph.addEdge(3, 4, 3);
graph.addEdge(4, 5, 1);
```

```
int source = 0; // Choose a source node

graph.dijkstra(source); // Run Dijkstra&#39;s algorithm

}

}
```

Implement student database (Roll number, Name of student, Gr. Number, Class etc.) using text or binary files in JAVA.

```
import java.io.*;class StudentRecords{static BufferedReader br = new BufferedReader(new

InputStreamReader(System.in));

public void addRecords() throws IOException

{

// Create or Modify a file for Database

PrintWriter pw = new PrintWriter(new BufferedWriter(new
FileWriter(&quot;studentRecords.txt&quot;,true)));

String name, Student_class;

int roll_no, grnumber;

String s;

boolean addMore = false;

// Read Data

do

{

System.out.print(&quot;\nEnter name: &quot;);

name = br.readLine();


System.out.print(&quot;Class &quot;);

Student_class = br.readLine();

System.out.print(&quot;Roll number: &quot;);

roll_no= Integer.parseInt(br.readLine());


System.out.print(&quot;Grade number: &quot;);

grnumber= Integer.parseInt(br.readLine());

// Print to File
```

```
pw.println(name);

pw.println(Student_class);

pw.println(roll_no);

pw.println(grnumber);


System.out.print(&quot;\nRecords added successfully !\n\nDo you want to add more records ? (y/n)
: &quot;);

s = br.readLine();

if(s.equalsIgnoreCase(&quot;y&quot;))

{

addMore = true;

System.out.println();

}

else

addMore = false;

}

while(addMore);

pw.close();

showMenu();

}

public void readRecords() throws IOException

{

try

{

// Open the file

BufferedReader file = new BufferedReader(new

FileReader(&quot;studentRecords.txt&quot;));

String name;

int i=1;

// Read records from the file

while((name = file.readLine()) != null)
```

```
{
System.out.println(&quot;S.No. : &quot; +(i++));
System.out.println(&quot;-------------&quot;);
System.out.println(&quot;\nName: &quot; +name);
System.out.println(&quot;Student Class : &quot;+file.readLine());
System.out.println(&quot;Roll number : &quot;+file.readLine());
System.out.println(&quot;Grade Number: &quot;+file.readLine());
System.out.println();
}
file.close();
showMenu();

}
catch(FileNotFoundException e)
{
System.out.println(&quot;\nERROR : File not Found !!!&quot;);
}
}
public void clear() throws IOException
{
// Create a blank file
PrintWriter pw = new PrintWriter(new BufferedWriter(new
FileWriter(&quot;studentRecords.txt&quot;)));
pw.close();
System.out.println(&quot;\nAll Records cleared successfully !&quot;);
for(int i=0;i&lt;999999999;i++); // Wait for some time
showMenu();
}
public void showMenu() throws IOException
{
```

```
System.out.print(&quot;1 : Add Records\n2 : Display Records\n3 : Clear All Records\n4 :
Exit\n\nYour

Choice : &quot;);

int choice = Integer.parseInt(br.readLine());

switch(choice)

{

case 1: addRecords();

break;

case 2: readRecords();

break;

case 3: clear();

break;

case 4: System.exit(1);

break;

default: System.out.println(&quot;\nInvalid Choice !&quot;);

break;

} }

public static void main(String args[]) throws IOException

{

StudentRecords call = new StudentRecords();

call.showMenu();

}}
```

You are given an undirected weighted graph with nodes and edges. The nodes are numbered

from and to. Find the total weight of the minimum spanning tree, as well as one specific minimum

spanning tree using Prims/Kruskal’s algorithm. Note that there may be multiple different minimum

spanning trees. You need to construct any one of them.

```
import java.util.*;

class Graph {
```

```
class Edge implements Comparable&lt;Edge&gt; {
int src, dest, weight;
public int compareTo(Edge compareEdge) {
return this.weight - compareEdge.weight;
}
};
// Union
class subset {
int parent, rank;
};
int vertices, edges;

Edge edge[];
// Graph creation
Graph(int v, int e) {
vertices = v;
edges = e;
edge = new Edge[edges];
for (int i = 0; i &lt; e; ++i)
edge[i] = new Edge();
}
int find(subset subsets[], int i) {
if (subsets[i].parent != i)
subsets[i].parent = find(subsets, subsets[i].parent);
return subsets[i].parent;
}
void Union(subset subsets[], int x, int y) {
int xroot = find(subsets, x);
int yroot = find(subsets, y);
if (subsets[xroot].rank &lt; subsets[yroot].rank)
subsets[xroot].parent = yroot;
```

```
else if (subsets[xroot].rank &gt; subsets[yroot].rank)
subsets[yroot].parent = xroot;
else {
subsets[yroot].parent = xroot;
subsets[xroot].rank++;
}
}
// Applying Krushkal Algorithm
void KruskalAlgo() {
Edge result[] = new Edge[vertices];
int minCost = 0;
int e = 0;
int i = 0;
for (i = 0; i &lt; vertices; ++i)
result[i] = new Edge();
// Sorting the edges
Arrays.sort(edge);
subset subsets[] = new subset[vertices];
for (i = 0; i &lt; vertices; ++i)
subsets[i] = new subset();
for (int v = 0; v &lt; vertices; ++v) {
subsets[v].parent = v;
subsets[v].rank = 0;
}
i = 0;
while (e &lt; vertices - 1) {
Edge next_edge = new Edge();
next_edge = edge[i++];

int x = find(subsets, next_edge.src);
int y = find(subsets, next_edge.dest);
if (x != y) {
result[e++] = next_edge;
Union(subsets, x, y);
}
}
for (i = 0; i &lt; e; ++i){
System.out.println(result[i].src + &quot; - &quot; + result[i].dest + &quot;: &quot; + result[i].weight);
minCost += result[i].weight;
}
System.out.println(&quot;Total cost of MST: &quot; + minCost);
}
public static void main(String[] args) {
int vertices = 6; // Number of vertices
int edges = 8; // Number of edges
Graph G = new Graph(vertices, edges);
G.edge[0].src = 0;
G.edge[0].dest = 1;
G.edge[0].weight = 4;
G.edge[1].src = 0;
G.edge[1].dest = 2;
G.edge[1].weight = 4;
G.edge[2].src = 1;
G.edge[2].dest = 2;
G.edge[2].weight = 2;
G.edge[3].src = 2;
G.edge[3].dest = 3;
G.edge[3].weight = 3;
G.edge[4].src = 2;
G.edge[4].dest = 5;
G.edge[4].weight = 2;
G.edge[5].src = 2;
```

```
G.edge[5].dest = 4;
G.edge[5].weight = 4;
G.edge[6].src = 3;
G.edge[6].dest = 4;
G.edge[6].weight = 3;
G.edge[7].src = 5;
G.edge[7].dest = 4;
G.edge[7].weight = 3;
G.KruskalAlgo();
}
}
```

Consider a friends' network on face book social web site. Model it as a graph to represent each node

as a user and a link to represent the friend relationship between them. Store data such as date of birth,

number of comments for each user.

1. Find who is having maximum friends

2. Find who has post maximum and minimum comments

3. Find users having birthday in this month

```
import java.util.Calendar;
import java.util.Scanner;
class SocialNetworkGraph {
static class DOB {
int day, month, year;
DOB(int day, int month, int year) {
this.day = day;
this.month = month;
this.year = year;
}
}
```

```
static class Node {
Link e;
Node next;
String name;
int comments;
DOB dob;
boolean visited;
Node(String name, int comments, DOB dob) {
this.name = name;
this.comments = comments;
this.dob = dob;
this.visited = false;
}
}
static class Link {
Link next;
Node ptr;
Link(Node p) {
this.ptr = p;
this.next = null;
}
}
private Node head, insert;
void create() {
Scanner scanner = new Scanner(System.in);
System.out.print(&quot;Enter number of nodes: &quot;);
int n = scanner.nextInt();
for (int i = 0; i &lt; n; i++) {
System.out.println(&quot;Enter details of person &quot; + (i + 1));
System.out.print(&quot;Enter name: &quot;);
String name = scanner.next();
```

```
System.out.print(&quot;Enter date of birth (day month year): &quot;);
int day = scanner.nextInt();
int month = scanner.nextInt();
int year = scanner.nextInt();
DOB dob = new DOB(day, month, year);
System.out.print(&quot;Enter number of comments: &quot;);

int comments = scanner.nextInt();
if (i == 0)
head = insert = new Node(name, comments, dob);
else {
insert.next = new Node(name, comments, dob);
insert = insert.next;
}
}
for (Node i = head; i != null; i = i.next) {
System.out.println(&quot;Who are friends of &quot; + i.name + &quot;?&quot;);
for (Node j = head; j != null; j = j.next) {
if (j == i)
continue;
System.out.print(&quot;Is &quot; + j.name + &quot; a friend? (y/n): &quot;);
char c = scanner.next().charAt(0);
if (c == &#39;y&#39;) {
Link temp;
if (i.e == null) {
i.e = new Link(j);
continue;
}
for (temp = i.e; temp.next != null; temp = temp.next);
temp.next = new Link(j);
}
}
}
}
void display() {
for (Node i = head; i != null; i = i.next) {
System.out.print(&quot;\nName: &quot; + i.name + &quot; DOB: &quot; + i.dob.day +
&quot;/&quot; + i.dob.month + &quot;/&quot; +
i.dob.year + &quot; Comments: &quot; + i.comments + &quot;\nFriends: &quot;);
for (Link temp = i.e; temp != null; temp = temp.next)
System.out.print(temp.ptr.name + &quot; &quot;);
}
}
void friends() {
int min = Integer.MAX_VALUE, max = 0;
Node[] S = new Node[30];
int top = 0;
S[top] = head;
head.visited = true;
while (top &gt; -1) {
Node temp = S[top--];
int n = 0;
for (Link l = temp.e; l != null; l = l.next) {
n++;
if (!l.ptr.visited) {
S[++top] = l.ptr;
l.ptr.visited = true;
}
}
if (max &lt; n) max = n;
if (min &gt; n) min = n;
```

```
}
System.out.println(&quot;Maximum: &quot; + max + &quot; Minimum: &quot; + min);
}
void comments() {
int min = Integer.MAX_VALUE, max = 0;
Node[] Q = new Node[30];
int f = -1, r = 0;
Q[r++] = head;
head.visited = true;
while (f != r - 1) {
Node temp = Q[++f];
if (max &lt; temp.comments) max = temp.comments;
if (min &gt; temp.comments) min = temp.comments;
for (Link l = temp.e; l != null; l = l.next) {
if (!l.ptr.visited) {
Q[r++] = l.ptr;
l.ptr.visited = true;
}
}
}
System.out.println(&quot;Maximum: &quot; + max + &quot; Minimum: &quot; + min);
}
void resetVisited() {
for (Node t = head; t != null; t = t.next)
t.visited = false;
}
void birthdays() {
Calendar now = Calendar.getInstance();
int month = now.get(Calendar.MONTH) + 1;
boolean flag = false;
System.out.println(&quot;Birthdays in current month:&quot;);
```

```
for (Node i = head; i != null; i = i.next) {

if (i.dob.month == month) {

System.out.println(i.name + &quot; &quot; + i.dob.day + &quot;-&quot; + i.dob.month + &quot;-
&quot; + i.dob.year);

flag = true;

}

}

if (!flag) System.out.println(&quot;No birthdays in this month!&quot;);

}

public static void main(String[] args) {

SocialNetworkGraph graph = new SocialNetworkGraph();

Scanner scanner = new Scanner(System.in);

int choice = 0;

while (choice != 6) {

System.out.print(&quot;\n1.Create \n2.Display \n3.Friends \n4.Comments \n5.Birthdays \n6.Exit:\n

&quot;);

choice = scanner.nextInt();

switch (choice) {

case 1:

graph.create();

break;

case 2:

graph.display();


break;

case 3:

graph.resetVisited();

graph.friends();

break;

case 4:

graph.resetVisited();
graph.comments();
break;
case 5:
graph.birthdays();
break;
}
}
}
}
```

Read the marks obtained by students of second year in an online examination of particular subject.

Find out maximum and minimum marks obtained in that subject. Use heap data structure. Analyze the

algorithm.

```
import java.util.Scanner;

public class MarksHeap {

    // Method to insert an element into a min-heap
    public static void insertMinHeap(int[] heap, int n, int value) {
        heap[n] = value;
        int i = n;
        while (i > 0 && heap[(i - 1) / 2] > heap[i]) {
            // Swap heap[i] with heap[(i - 1) / 2]
            int temp = heap[i];
            heap[i] = heap[(i - 1) / 2];
            heap[(i - 1) / 2] = temp;
            i = (i - 1) / 2;
        }
    }
```

```
// Method to insert an element into a max-heap
public static void insertMaxHeap(int[] heap, int n, int value) {
    heap[n] = value;
    int i = n;
    while (i > 0 && heap[(i - 1) / 2] < heap[i]) {
        // Swap heap[i] with heap[(i - 1) / 2]
        int temp = heap[i];
        heap[i] = heap[(i - 1) / 2];
        heap[(i - 1) / 2] = temp;
        i = (i - 1) / 2;
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the number of students: ");
    int n = scanner.nextInt();

    int[] minHeap = new int[n];
    int[] maxHeap = new int[n];

    System.out.println("Enter the marks obtained by the students:");
    for (int i = 0; i < n; i++) {
        int mark = scanner.nextInt();
        insertMinHeap(minHeap, i, mark);
        insertMaxHeap(maxHeap, i, mark);
    }

    int minMark = minHeap[0];  // The root of the min-heap
    int maxMark = maxHeap[0];  // The root of the max-heap
```

```
        System.out.println("Minimum mark: " + minMark);

        System.out.println("Maximum mark: " + maxMark);

    }

}
```

Implement IsSpell utility (1. Create Hashtable and write in file 2. Enter Word 3. Search word 4. Exit)

```
import java.io.*;

import java.util.*;

public class IsSpellUtility {

public static void main(String[] args) {

Hashtable&lt;String, Integer&gt; dictionary = new Hashtable&lt;&gt;();

Scanner scanner = new Scanner(System.in);

// Load dictionary from file

loadDictionary(dictionary);

while (true) {

System.out.println(&quot;1. Enter Word&quot;);

System.out.println(&quot;2. Search Word&quot;);

System.out.println(&quot;3. Remove Word&quot;);

System.out.println(&quot;4. Exit&quot;);

System.out.print(&quot;Enter your choice: &quot;);

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline

switch (choice) {

case 1:

System.out.print(&quot;Enter the word to add to dictionary: &quot;);

String wordToAdd = scanner.nextLine();

addWordToDictionary(dictionary, wordToAdd);

saveDictionaryToFile(dictionary);

break;

case 2:

System.out.print(&quot;Enter the word to search in dictionary: &quot;);
String wordToSearch = scanner.nextLine();
searchWordInDictionary(dictionary, wordToSearch);
break;
case 3:
System.out.print(&quot;Enter the word to remove from dictionary: &quot;);
String wordToRemove = scanner.nextLine();
removeWordFromDictionary(dictionary, wordToRemove);
saveDictionaryToFile(dictionary);
break;
case 4:
System.out.println(&quot;Exiting program...&quot;);
System.exit(0);
break;
default:
System.out.println(&quot;Invalid choice. Please enter a valid option.&quot;);
}
}
}
private static void loadDictionary(Hashtable&lt;String, Integer&gt; dictionary) {
try (BufferedReader reader = new BufferedReader(new FileReader(&quot;dictionary.txt&quot;))) {

String line;
while ((line = reader.readLine()) != null) {
dictionary.put(line.trim().toLowerCase(), 1);
}
} catch (IOException e) {
System.out.println(&quot;Error loading dictionary: &quot; + e.getMessage());
}
}
private static void addWordToDictionary(Hashtable&lt;String, Integer&gt; dictionary, String word) {
dictionary.put(word.trim().toLowerCase(), 1);
```

```
System.out.println(&quot;Word added to dictionary.&quot;);
}
private static void searchWordInDictionary(Hashtable&lt;String, Integer&gt; dictionary, String word)
{
if (dictionary.containsKey(word.trim().toLowerCase())) {
System.out.println(&quot;Word found in dictionary.&quot;);
} else {
System.out.println(&quot;Word not found in dictionary.&quot;);
}
}
private static void removeWordFromDictionary(Hashtable&lt;String, Integer&gt; dictionary, String
word) {
if (dictionary.containsKey(word.trim().toLowerCase())) {
dictionary.remove(word.trim().toLowerCase());
System.out.println(&quot;Word removed from dictionary.&quot;);
} else {
System.out.println(&quot;Word not found in dictionary.&quot;);
}
}
private static void saveDictionaryToFile(Hashtable&lt;String, Integer&gt; dictionary) {
try (BufferedWriter writer = new BufferedWriter(new FileWriter(&quot;dictionary.txt&quot;))) {
for (String word : dictionary.keySet()) {
writer.write(word);
writer.newLine();
}
} catch (IOException e) {
System.out.println(&quot;Error saving dictionary to file: &quot; + e.getMessage());
}
}
}
```

Construct an expression tree from postfix/prefix expression and perform recursive and non- recursive In-order, pre-order and post-order traversals.

```
package tree;

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

import java.util.Stack;


public class expressionEval {

static Node root;

static class Node{

char data;

Node left, right;

public Node(char data) {

this.data=data;

left=right=null;

}

}

public static void main(String[] args) {

Scanner in=new Scanner(System.in);

int continues=1;

while(continues==1) {

System.out.println(&quot;Menu:&quot;);

System.out.println(&quot;1:Enter expression&quot;);

System.out.println(&quot;2:Recursive Inorder&quot;);

System.out.println(&quot;3:Non-Recursive Inorder&quot;);

System.out.println(&quot;4:Recursive Preorder&quot;);

System.out.println(&quot;5:Non-Recursive Preorder&quot;);

System.out.println(&quot;6:Recursive Postorder&quot;);

System.out.println(&quot;7:Non-Recursive Postorder&quot;);

System.out.println(&quot;8:BFS&quot;);

System.out.println(&quot;&lt;Other Keys&gt;:Exit&quot;);

System.out.print(&quot;Enter choice: &quot;);
```

```
int choice;

try {

choice=in.nextInt();

}

catch (Exception e) {

choice=9;

}

System.out.println(&quot;\n&quot;);

switch(choice) {

case 1:

System.out.print(&quot;Enter expression: &quot;);

String str=in.next();

convert(str);

break;

case 2:

System.out.println(&quot;Recursive Inorder: &quot;);


recurinorder(root);

break;

case 3:

System.out.println(&quot;Non-Recursive Inorder: &quot;);

nonrecurinorder(root);

break;

case 4:

System.out.println(&quot;Recursive Preorder:&quot;);

recurpreorder(root);

break;

case 5:

System.out.println(&quot;Non-recursive Preorder:&quot;);

nonrecurpreorder(root);

break;
```

```
case 6:
System.out.println(&quot;Recursive Postorder:&quot;);
recurpostorder(root);
break;
case 7:
System.out.println(&quot;Non-Recursive Postorder:&quot;);
nonrecurpostorder(root);
break;
case 8:
System.out.println(&quot;BFS:&quot;);
bfs(root);
break;
default:
continues=0;
break;
}
System.out.println(&quot;\n\n&quot;);
}
in.close();
}
public static boolean isOperator(char c) {
return (c == &#39;+&#39; || c == &#39;-&#39; || c == &#39;*&#39; || c == &#39;/&#39;);
}
public static void convert(String postfix) {
Stack&lt;Node&gt; stack=new Stack&lt;Node&gt;();
for(char ch: postfix.toCharArray()) {
if(!isOperator(ch)) {
Node temp=new Node(ch);
stack.push(temp);
}
else {
```

```
Node op1, op2, temp;
temp=new Node(ch);
op1=stack.pop();
op2=stack.pop();
temp.left=op2;
temp.right=op1;

stack.push(temp);
}
}
root=stack.pop();
}
public static void recurinorder(Node root) {
if(root==null) {
return;
}
else {
Node temp=root;
recurinorder(temp.left);
System.out.print(temp.data+&quot; &quot;);
recurinorder(temp.right);
}
}
public static void recurpreorder(Node root) {
if(root==null) {
return;
}
else {
Node temp=root;
System.out.print(temp.data+&quot; &quot;);
recurpreorder(temp.left);
recurpreorder(temp.right);
}
}
public static void recurpostorder(Node root) {
if(root==null) {
return;
}
else {
Node temp=root;
recurpostorder(temp.left);

recurpostorder(temp.right);
System.out.print(temp.data+&quot; &quot;);
}
}
public static void nonrecurinorder(Node root) {
Stack&lt;Node&gt; stack = new Stack&lt;&gt;();
Node current = root;

while (current != null || !stack.isEmpty()) {
while (current != null) {
stack.push(current);
current = current.left;
}
current = stack.pop();
System.out.print(current.data + &quot; &quot;);
current = current.right;
}
}
public static void nonrecurpreorder(Node root) {
Stack&lt;Node&gt; stack = new Stack&lt;&gt;();
```

```
stack.push(root);
while (!stack.isEmpty()) {
Node current = stack.pop();
System.out.print(current.data + &quot; &quot;);
if (current.right != null) {
stack.push(current.right);
}
if (current.left != null) {
stack.push(current.left);
}
}
}
public static void nonrecurpostorder(Node root) {
Stack&lt;Node&gt; stack1 = new Stack&lt;&gt;();
Stack&lt;Node&gt; stack2 = new Stack&lt;&gt;();
stack1.push(root);
while (!stack1.isEmpty()) {
Node current = stack1.pop();
stack2.push(current);
if (current.left != null) {
stack1.push(current.left);
}
if (current.right != null) {
stack1.push(current.right);
}
}
while (!stack2.isEmpty()) {
System.out.print(stack2.pop().data + &quot; &quot;);
}
}
public static void bfs(Node root) {
```

```
if (root == null) {
System.out.println(&quot;The tree is empty.&quot;);
return;
}

Queue&lt;Node&gt; queue = new LinkedList&lt;&gt;();
queue.add(root);
while (!queue.isEmpty()) {
Node current = queue.poll();
System.out.print(current.data + &quot; &quot;);
if (current.left != null) {
queue.add(current.left);
}
if (current.right != null) {
queue.add(current.right);
}
}
}
}
```

A customer wants to travel from source A to destination B. He books a cab from source A to reach destination B. Calculate a shortest path by avoiding real time traffic to reach destination B.

```
import java.util.Arrays;

class Graph {
private int vertices;
private int[][] adjacencyMatrix;
// Constructor
public Graph(int vertices) {
this.vertices = vertices;
adjacencyMatrix = new int[vertices][vertices];
```

```
// Initialize all edges to infinity (no connection) except for self-loops
for (int i = 0; i &lt; vertices; i++) {
Arrays.fill(adjacencyMatrix[i], Integer.MAX_VALUE);
adjacencyMatrix[i][i] = 0;
}
}
// Adding edges to the graph
public void addEdge(int src, int dest, int weight) {
adjacencyMatrix[src][dest] = weight;
adjacencyMatrix[dest][src] = weight; // If the graph is undirected
}
// Dijkstra&#39;s algorithm to find the shortest path
public void dijkstra(int source) {
int[] distances = new int[vertices];
boolean[] visited = new boolean[vertices];
// Initialize distances as infinity and visited as false
Arrays.fill(distances, Integer.MAX_VALUE);
distances[source] = 0; // Distance to the source itself is 0
// Loop to find the shortest path for all vertices
for (int i = 0; i &lt; vertices - 1; i++) {
// Find the unvisited vertex with the smallest distance
int u = getMinDistanceVertex(distances, visited);
// Mark the picked vertex as visited
visited[u] = true;
// Update distances of adjacent vertices of the picked vertex
for (int v = 0; v &lt; vertices; v++) {
// Update distances[v] if:
// 1. v is not visited
// 2. There is an edge from u to v
// 3. Total weight of path from source to v through u is smaller than the current value of
distances[v]
if (!visited[v] &amp;&amp; adjacencyMatrix[u][v] != Integer.MAX_VALUE &amp;&amp;
distances[u] != Integer.MAX_VALUE &amp;&amp;
distances[u] + adjacencyMatrix[u][v] &lt; distances[v]) {
distances[v] = distances[u] + adjacencyMatrix[u][v];
}
}
}

// Print the shortest distances from the source
printSolution(distances, source);
}
// Helper method to find the vertex with the minimum distance value
private int getMinDistanceVertex(int[] distances, boolean[] visited) {
int minDistance = Integer.MAX_VALUE;
int minIndex = -1;
for (int v = 0; v &lt; vertices; v++) {
if (!visited[v] &amp;&amp; distances[v] &lt; minDistance) {
minDistance = distances[v];
minIndex = v;
}
}
return minIndex;
}
// Helper method to print the solution
private void printSolution(int[] distances, int source) {
System.out.println(&quot;Shortest paths from node &quot; + source + &quot;:&quot;);
for (int i = 0; i &lt; distances.length; i++) {
System.out.println(&quot;To node &quot; + i + &quot; is &quot; + distances[i]);
}
}
public static void main(String[] args) {
```

```
Graph graph = new Graph(6);
graph.addEdge(0, 1, 4);
graph.addEdge(0, 2, 1);
graph.addEdge(2, 1, 2);
graph.addEdge(1, 3, 1);
graph.addEdge(2, 3, 5);
graph.addEdge(3, 4, 3);
graph.addEdge(4, 5, 1);
int source = 0; // Choose a source node
graph.dijkstra(source); // Run Dijkstra&#39;s algorithm
}
}
```

Implement student database (Roll number, Name of student, Gr. Number, Class etc.) using text or binary files in JAVA.

```
import java.io.*;class StudentRecords{static BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
public void addRecords() throws IOException
{
// Create or Modify a file for Database
PrintWriter pw = new PrintWriter(new BufferedWriter(new
FileWriter(&quot;studentRecords.txt&quot;,true)));
String name, Student_class;
int roll_no, grnumber;
String s;
boolean addMore = false;
// Read Data
do
{
System.out.print(&quot;\nEnter name: &quot;);
name = br.readLine();
System.out.print(&quot;Class &quot;);

Student_class = br.readLine();

System.out.print(&quot;Roll number: &quot;);

roll_no= Integer.parseInt(br.readLine());


System.out.print(&quot;Grade number: &quot;);

grnumber= Integer.parseInt(br.readLine());

// Print to File

pw.println(name);

pw.println(Student_class);

pw.println(roll_no);

pw.println(grnumber);


System.out.print(&quot;\nRecords added successfully !\n\nDo you want to add more records ? (y/n)
: &quot;);

s = br.readLine();

if(s.equalsIgnoreCase(&quot;y&quot;))

{

addMore = true;

System.out.println();

}

else

addMore = false;

}

while(addMore);

pw.close();

showMenu();

}

public void readRecords() throws IOException

{

try
```

```
{
// Open the file
BufferedReader file = new BufferedReader(new
FileReader(&quot;studentRecords.txt&quot;));
String name;
int i=1;
// Read records from the file
while((name = file.readLine()) != null)
{
System.out.println(&quot;S.No. : &quot; +(i++));
System.out.println(&quot;-------------&quot;);
System.out.println(&quot;\nName: &quot; +name);
System.out.println(&quot;Student Class : &quot;+file.readLine());
System.out.println(&quot;Roll number : &quot;+file.readLine());
System.out.println(&quot;Grade Number: &quot;+file.readLine());
System.out.println();
}
file.close();
showMenu();

}
catch(FileNotFoundException e)
{
System.out.println(&quot;\nERROR : File not Found !!!&quot;);
}
}
public void clear() throws IOException
{
// Create a blank file
PrintWriter pw = new PrintWriter(new BufferedWriter(new
FileWriter(&quot;studentRecords.txt&quot;)));
```

```
pw.close();

System.out.println(&quot;\nAll Records cleared successfully !&quot;);

for(int i=0;i&lt;999999999;i++); // Wait for some time

showMenu();

}

public void showMenu() throws IOException

{

System.out.print(&quot;1 : Add Records\n2 : Display Records\n3 : Clear All Records\n4 :
Exit\n\nYour

Choice : &quot;);

int choice = Integer.parseInt(br.readLine());

switch(choice)

{

case 1: addRecords();

break;

case 2: readRecords();

break;

case 3: clear();

break;

case 4: System.exit(1);

break;

default: System.out.println(&quot;\nInvalid Choice !&quot;);

break;

} }

public static void main(String args[]) throws IOException

{

StudentRecords call = new StudentRecords();

call.showMenu();

}}
```