```
import java.util.ArrayList;
import java.util.List;
import java.util.PriorityQueue;
import java.util.*;

public class App { // Puzzle

 public int dimension = 3;

 // Moves
 int[] row = { 1, 0, -1, 0 };
 int[] col = { 0, -1, 0, 1 };

 PriorityQueue<Node> pq = new PriorityQueue<Node>(1000, (a, b) -> (a.cost + a.level) - (b.cost + b.level));
 HashSet<Node> visited = new HashSet<>();

 public int calculateCost(int[][] initial, int[][] goal) {
  int count = 0;
  int n = initial.length;
  for (int i = 0; i < n; i++) {
   for (int j = 0; j < n; j++) {
    if (initial[i][j] != 0 && initial[i][j] != goal[i][j]) {
     count++;
    }
   }
  }
  return count;
 }

 public void printMatrix(int[][] matrix) {
  for (int i = 0; i < matrix.length; i++) {
   for (int j = 0; j < matrix.length; j++) {
    System.out.print(matrix[i][j] + " ");
   }
   System.out.println();
  }
 }

 public boolean isSafe(int x, int y) {
  return (x >= 0 && x < dimension && y >= 0 && y < dimension);
 }

 public void printPath(Node root) {
  if (root == null) {
   return;
  }
  printPath(root.parent);
  if (root.parent != null) {
   System.out.println("  |");
   System.out.println("  V");
  }
  System.out.println();
  printMatrix(root.matrix);
  System.out.println();
 }
```

```
public boolean isSolvable(int[][] matrix) {
 int count = 0;
 List<Integer> array = new ArrayList<Integer>();

 for (int i = 0; i < matrix.length; i++) {
  for (int j = 0; j < matrix.length; j++) {
   array.add(matrix[i][j]);
  }
 }

 Integer[] anotherArray = new Integer[array.size()];
 array.toArray(anotherArray);

 for (int i = 0; i < anotherArray.length - 1; i++) {
  for (int j = i + 1; j < anotherArray.length; j++) {
   if (anotherArray[i] != 0 && anotherArray[j] != 0 && anotherArray[i] > anotherArray[j]) {
    count++;
   }
  }
 }

 return count % 2 == 0;
}

public void addToQueue(Node node) {
 if (!visited.contains(node))
  pq.add(node);
}

public void solvePuzzle(int[][] initial, int[][] goal, int x, int y) {
 Node root = new Node(initial, x, y, x, y, 0, null);
 root.cost = calculateCost(initial, goal);
 addToQueue(root);

 while (!pq.isEmpty()) {
  Node min = pq.poll();
  if (min.cost == 0) {
   printPath(min);
   return;
  }

  visited.add(min);

  for (int i = 0; i < 4; i++) {
   if (isSafe(min.x + row[i], min.y + col[i])) {
    Node child = new Node(min.matrix, min.x, min.y, min.x + row[i], min.y + col[i], min.level + 1, min);
    child.cost = calculateCost(child.matrix, goal);
    addToQueue(child);
   }
  }
 }
}

public int[] findTilePosition(int initial[][]) {
```

```
  int res[] = new int[2];
  for (int i = 0; i < initial.length; i++) {
   for (int j = 0; j < initial[0].length; j++) {
    if (initial[i][j] == 0) {
     res[0] = i;
     res[1] = j;
    }
   }
  }
  return res;
 }

 public static void main(String[] args) {
  Scanner scanner = new Scanner(System.in);
  int initial[][] = new int[3][3];
  int goal[][] = new int[3][3];

  System.out.println("Enter Initial Matrix: ");
  for (int i = 0; i < initial.length; i++) {
   for (int j = 0; j < initial[0].length; j++) {
    initial[i][j] = scanner.nextInt();
   }
  }

  System.out.println("Enter Goal Matrix: ");
  for (int i = 0; i < initial.length; i++) {
   for (int j = 0; j < initial[0].length; j++) {
    goal[i][j] = scanner.nextInt();
   }
  }
  System.out.println();

  App puzzle = new App();

  int res[] = puzzle.findTilePosition(initial);

  if (puzzle.isSolvable(initial)) {
   puzzle.solvePuzzle(initial, goal, res[0], res[1]);
  } else {
   System.out.println("Puzzle is not solvable");
  }
 }

}

class Node {

 public Node parent;
 public int[][] matrix;
 public int x, y; // blank title coordinates
 public int cost; // misplaced tiles - h value
 public int level; // level - g value

 public Node(int[][] matrix, int x, int y, int newX, int newY, int level, Node parent) {
  this.parent = parent;
```

```
 this.matrix = new int[matrix.length][];
 for (int i = 0; i < matrix.length; i++) {
  this.matrix[i] = matrix[i].clone();
 }

 int temp = this.matrix[x][y];
 this.matrix[x][y] = this.matrix[newX][newY];
 this.matrix[newX][newY] = temp;

 this.cost = Integer.MAX_VALUE;
 this.level = level;
 this.x = newX;
 this.y = newY;
}

public boolean equals(Object o) {
 if (this == o)
  return true;
 Node node = (Node) o;
 return Arrays.equals(matrix, node.matrix);
}

public int hashCode() {
 return Arrays.hashCode(matrix);
}

}
```