```java
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Random;
import java.util.stream.IntStream;

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

        Random rnd = new Random();
        rnd.setSeed(23);

        // - Preference lists for m1, m2, …, mn
        int[][] M = {
        {2, 1, 3, 0},
        {0, 1, 3, 2},
        {0, 1, 2, 3},
        {0, 1, 2, 3},
        };

        //- Preference lists for w1, w2, …, wn
        int[][] W = {
        {0, 2, 1, 3},
        {2, 0, 3, 1},
        {3, 2, 1, 0},
        {2, 3, 1, 0},
        };

        //test(5,233);
        System.out.println(Arrays.toString(galeShapley(M,W)));


    }

    public static int[] galeShapley(int[][] M, int[][] W){
        int[] pairM = new int [M.length];
        int[] pairW = new int [W.length];

        // memorize w's index in m's preference list to avoid revisited same w
        int[] rankM = new int [M.length];

        Arrays.fill(pairM, -1);
        Arrays.fill(pairW, -1);
        Arrays.fill(rankM, -1);

        // using Queue to track free m
        Queue<Integer>freeM = new LinkedList<Integer>();
        for (int i = 0; i <  W.length; i++){
            freeM.offer(i);
        }

        while (!freeM.isEmpty()){
            int m = freeM.poll();
            // start searching qualified w from last unmatch w index + 1
            for(int i = rankM[m] + 1; i < W.length; i++){
```

```
                int w = M[m][i];
                if (pairW[w] == -1){
                    // if w is free, match (m, w)
                    System.out.printf("%d proposes to %d [%d, %d]  "+
                                      "Accepted\n", m , w, w, pairW[w] );
                    pairM[m] = w;
                    pairW[w] = m;

                    // note down starting point for next iteration
                    rankM[m] = i;
                    break;

                }else{
                    for(int j = 0; j < W.length; j++){
                        if (W[w][j] == m){
                            // unmatch w with pairW[w]
                            freeM.offer(pairW[w]);

                            // match w with m
                            System.out.printf("%d proposes to %d [%d, %d]  "+
                                              "Accepted\n", m , w, w, pairW[w]
);
                            pairM[m] = w;
                            pairW[w] = m;

                            // note down starting point for next iteration
                            rankM[m] = findIndex(M[m], w);
                            i = W.length;
                            break;
                        }
                        if (W[w][j]  == pairW[w]){
                            System.out.printf("%d proposes to %d [%d, %d]  "+
                                              "Rejected\n", m , w, w, pairW[w]
);
                            break;
                        }
                    }
                }
            }
        }
        return pairM;
    }


    public static int[] permutation(int n, Random rand) {
        int[] arr = IntStream.range(0, n).toArray();
        for (int i = 0; i < n; i++) {
            int j = rand.nextInt(i + 1);
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
        return arr;
        }


    public static int[][] generatePef(int n, Random rand){
        int[][] pef = new int[n][];
```

```
        for(int i = 0; i < n; i++){
            int[] cur = permutation(n, rand);
            pef[i] = cur;
        }
        return pef;
    }

    public static int findIndex(int[] arr, int t) {
        if (arr == null) return -1;
        int len = arr.length;
        int i = 0;
        while (i < len) {
            if (arr[i] == t){
                return i;
            }
            else{
                i++;
            }
        }
        return -1;
    }

    public static void test(int n, int seed){

        Random rnd = new Random();
        rnd.setSeed(seed);

        int[][] pefm = generatePef(n, rnd);
        int[][] pefw = generatePef(n, rnd);
        System.out.println(Arrays.deepToString(pefm));
        System.out.println(Arrays.deepToString(pefw));
        int[] rndRes = galeShapley(pefm, pefw);
        System.out.println(Arrays.toString(rndRes));
    }

}


/*
output & printout:
0 proposes to 2 [2, -1]  Accepted
1 proposes to 0 [0, -1]  Accepted
2 proposes to 0 [0, 1]  Accepted
3 proposes to 0 [0, 2]  Rejected
3 proposes to 1 [1, -1]  Accepted
1 proposes to 1 [1, 3]  Rejected
1 proposes to 3 [3, -1]  Accepted
[2, 3, 0, 1]
*/
```