Back-Tracking

Q1. Maze

Recursion, return void

```java
package com.inclass.backtracking;

import java.util.ArrayList;
import java.util.List;

public class Maze {
    public static void main(String[] args) {
        // start
        int x1 = 1, y1 = 1;
        // end
        int x2 = 3, y2 = 3;
        /* N -> North, S -> South, E -> East, W -> West
        Ne -> North-east, Nw -> North-west, Se -> South-east, Sw -> South-west */
        recVoid(x1 - x2, y1 - y2, "");
    }

    static void recVoid (int x, int y, String str) {
        if (x == 0 && y == 0) {
            System.out.print(str + " ");
            return;
        }
        if (x > 0) {
            recVoid(x - 1, y, str + "N");
        } if (x < 0) {
            recVoid(x + 1, y, str + "S");
        } if (y > 0) {
            recVoid(x, y - 1, str + "W");
        } if (y < 0) {
            recVoid(x, y + 1, str + "E");
        }
    }
}
```

Recursion, return List

```java
package com.inclass.backtracking;

import java.util.ArrayList;
import java.util.List;

public class Maze {
    public static void main(String[] args) {
        // start
        int x1 = 1, y1 = 1;
        // end
        int x2 = 3, y2 = 3;
        /* N -> North, S -> South, E -> East, W -> West
        Ne -> North-east, Nw -> North-west, Se -> South-east, Sw -> South-west */
        System.out.println("\n" + recList(x1 - x2, y1 - y2, ""));
```

```
    }

    static List<String> recList (int x, int y, String str) {
        List<String> list = new ArrayList<>();
        if (x == 0 && y == 0) {
            list.add(str);
            return list;
        }
        if (x > 0) {
            list.addAll(recList(x - 1, y, str + "N"));
        } if (x < 0) {
            list.addAll(recList(x + 1, y, str + "S"));
        } if (y > 0) {
            list.addAll(recList(x, y - 1, str + "W"));
        } if (y < 0) {
            list.addAll(recList(x, y + 1, str + "E"));
        }
        return list;
    }
}
```

Q2. Maze with diagonals

```
package com.inclass;

import java.util.ArrayList;
import java.util.List;

public class Diagonal {
    public static void main(String[] args) {
        // start
        int x1 = 1, y1 = 1;
        // end
        int x2 = 3, y2 = 3;
        /* N -> North, S -> South, E -> East, W -> West
        Ne -> North-east, Nw -> North-west, Se -> South-east, Sw -> South-west */
        diagonal(x1, x2, y1, y2);
    }

    static void diagonal (int x1, int x2, int y1, int y2) {
        System.out.println(recList(x1 - x2, y1 - y2, ""));
    }

    static List<String> recList (int x, int y, String str) {
        List<String> list = new ArrayList<>();
        if (x == 0 && y == 0) {
            list.add(str);
            return list;
        }

        if (x > 0) {
            list.addAll(recList(x - 1, y, str + "N"));
            if (y < 0) {
                list.addAll(recList(x - 1, y + 1, str + "Ne"));
            } if (y > 0) {
                list.addAll(recList(x - 1, y - 1, str + "Nw"));
            }
        } if (y < 0) {
```

```
            list.addAll(recList(x, y + 1, str + "E"));
        }

        if (x < 0) {
            list.addAll(recList(x + 1, y, str + "S"));
            if (y > 0) {
                list.addAll(recList(x + 1, y - 1, str + "Sw"));
            } if (y < 0) {
                list.addAll(recList(x + 1, y + 1, str + "Se"));
            }
        } if (y > 0) {
            list.addAll(recList(x, y - 1, str + "W"));
        }

        return list;
    }
}
```

Q3. Maze with single obstacle

```
package com.inclass;

import java.util.ArrayList;
import java.util.List;

public class SingleObstacle {
    public static void main(String[] args) {
        // start
        int x1 = 1, y1 = 1;
        // end
        int x2 = 3, y2 = 3;
        // obstacle
        int xx = 2, yy = 2;
        /* N -> North, S -> South, E -> East, W -> West
        Ne -> North-east, Nw -> North-west, Se -> South-east, Sw -> South-west */
        singleObstacle(x1, x2, xx, y1, y2, yy);
    }

    static void singleObstacle (int x1 , int x2, int xx , int y1 , int y2 ,int yy) {
        System.out.println(recList(x1 - x2, y1 - y2, xx - x2, yy - y2, ""));
    }

    static List<String> recList(int x, int y, int xx, int yy, String str) {
        List<String> list = new ArrayList<>();
        if (x == 0 && y == 0) {
            list.add(str);
            return list;
        } if (x == xx && y == yy) {
            return list;
        }
        if (x > 0) {
            list.addAll(recList(x - 1, y, xx, yy, str + "N"));
        } if (x < 0) {
            list.addAll(recList(x + 1, y, xx, yy, str + "S"));
        } if (y > 0) {
            list.addAll(recList(x, y - 1, xx, yy, str + "W"));
        } if (y < 0) {
```

```
            list.addAll(recList(x, y + 1, xx, yy, str + "E"));
        }
        return list;
    }
}
```

Q4. Maze with multiply obstacles

```
package com.inclass;

import java.util.ArrayList;
import java.util.List;

public class MultipleObstacles {
    public static void main(String[] args) {
        /* N -> North, S -> South, E -> East, W -> West
        Ne -> North-east, Nw -> North-west, Se -> South-east, Sw -> South-west */
        boolean[][] maze = new boolean[][]{
                {true, true, true},
                {true, true, false},
                {true, true, true}
        };
        int x1, y1, x2, y2;

        // start
        x1 = 1; y1 = 1;
        // target
        x2 = 3; y2 = 3;
        multipleObstracles(x1, x2, y1, y2, maze);

        // start
        x1 = 3; y1 = 3;
        // target
        x2 = 1; y2 = 1;
        multipleObstracles(x1, x2, y1, y2, maze);

        // start
        x1 = 3; y1 = 1;
        // target
        x2 = 1; y2 = 3;
        multipleObstracles(x1, x2, y1, y2, maze);

        // start
        x1 = 1; y1 = 3;
        // target
        x2 = 3; y2 = 1;
        multipleObstracles(x1, x2, y1, y2, maze);
    }

    static void multipleObstracles (int x1, int x2, int y1, int y2, boolean[][] maze) {
        System.out.println(recList(maze, x1 - x2, y1 -y2, "", x1 - x2 > 0));
    }

    static List<String> recList (boolean[][] arr, int x, int y, String str, Boolean bool) {
        List<String> list = new ArrayList<>();
        if (Math.abs(x) == 0 && Math.abs(y) == 0) {
            list.add(str);
```

```
            return list;
        }

        if (!bool) {
            if (y < 0) {
                if (!arr[arr.length + y - 1][arr.length + x - 1]){
                    return list;
                }
                list.addAll(recList(arr, x, y + 1, str + "S", bool));
            } if (y > 0) {
                if (!arr[y][arr.length + x - 1]) {
                    return list;
                }
                list.addAll(recList(arr, x, y - 1, str + "N", bool));
            } if (x != 0) {
                list.addAll(recList(arr, x + 1, y, str + "E", bool));
            }
        }

        if (bool) {
            if (y < 0) {
                if (!arr[arr.length + y - 1][x]){
                    return list;
                }
                list.addAll(recList(arr, x, y + 1, str + "S", bool));
            } if (y > 0) {
                if (!arr[y][x]) {
                    return list;
                }
                list.addAll(recList(arr, x, y - 1, str + "N", bool));
            } if (x != 0) {
                list.addAll(recList(arr, x - 1, y, str + "W", bool));
            }
        }
        return list;
    }
}
```

Q5. Maze with all directions

```
package com.inclass;

import java.util.ArrayList;
import java.util.List;

public class AllDirections {
    public static void main(String[] args) {
        /* N -> North, S -> South, E -> East, W -> West
        Ne -> North-east, Nw -> North-west, Se -> South-east, Sw -> South-west */
        int x1, y1, x2, y2;

        // start
        x1 = 1; y1 = 1;
        // target
        x2 = 3; y2 = 3;
        allDirections(x1, x2, y1, y2);

        // start
```

```
        x1 = 1; y1 = 3;
        // target
        x2 = 3; y2 = 1;
        allDirections(x1, x2, y1, y2);

        // start
        x1 = 3; y1 = 1;
        // target
        x2 = 1; y2 = 3;
        allDirections(x1, x2, y1, y2);

        // start
        x1 = 3; y1 = 3;
        // target
        x2 = 1; y2 = 1;
        allDirections(x1, x2, y1, y2);
    }

    static void allDirections(int x1, int x2, int y1, int y2) {
        boolean[][] arr = new boolean[][]{
                {true, true, true},
                {true, true, true},
                {true, true, true}
        };
        System.out.println(recList(arr, x1 - x2, y1 - y2, "", x1 - x2 > 0, y1 - y2 > 0));
    }

    static List<String> recList (boolean[][] arr, int x, int y, String str, boolean bool1,
boolean bool2) {
        List<String> list = new ArrayList<>(0);
        if (x == 0 && y == 0) {
            list.add(str);
            return list;
        } if (!bool1) {
            if (!bool2) {
                if (-x < 0 || -x ≥ arr.length || -y < 0 || -y ≥ arr[0].length ||
!arr[arr[0].length + y - 1][arr.length + x - 1]) {
                    return list;
                }
                arr[arr[0].length + y - 1][arr.length + x - 1] = false;
                list.addAll(recList(arr, x + 1, y, str + "E", bool1, bool2));
                list.addAll(recList(arr, x - 1, y, str + "W", bool1, bool2));
                list.addAll(recList(arr, x, y + 1, str + "S", bool1, bool2));
                list.addAll(recList(arr, x, y - 1, str + "N", bool1, bool2));
                arr[arr[0].length + y - 1][arr.length + x - 1] = true;
            } if (bool2) {
                if (-x < 0 || -x ≥ arr.length ||  y < 0 || y ≥ arr[0].length || !arr[y]
[arr.length + x - 1]) {
                    return list;
                }
                arr[y][arr.length + x - 1] = false;
                list.addAll(recList(arr, x + 1, y, str + "E", bool1, bool2));
                list.addAll(recList(arr, x - 1, y, str + "W", bool1, bool2));
                list.addAll(recList(arr, x, y + 1, str + "S", bool1, bool2));
                list.addAll(recList(arr, x, y - 1, str + "N", bool1, bool2));
                arr[y][arr.length + x - 1] = true;
            }
        } if (bool1) {
            if (!bool2) {
                if (x < 0 || x ≥ arr.length || -y < 0 || -y ≥ arr[0].length ||
```

```
!arr[arr[0].length + y - 1][x]) {
                    return list;
                }
                arr[arr[0].length + y - 1][x] = false;
                list.addAll(recList(arr, x + 1, y, str + "E", bool1, bool2));
                list.addAll(recList(arr, x - 1, y, str + "W", bool1, bool2));
                list.addAll(recList(arr, x, y + 1, str + "S", bool1, bool2));
                list.addAll(recList(arr, x, y - 1, str + "N", bool1, bool2));
                arr[arr[0].length + y - 1][x] = true;
            } if (bool2) {
                if (x < 0 || x ≥ arr.length ||  y < 0 || y ≥ arr[0].length || !arr[y][x]) {
                    return list;
                }
                arr[y][x] = false;
                list.addAll(recList(arr, x + 1, y, str + "E", bool1, bool2));
                list.addAll(recList(arr, x - 1, y, str + "W", bool1, bool2));
                list.addAll(recList(arr, x, y + 1, str + "S", bool1, bool2));
                list.addAll(recList(arr, x, y - 1, str + "N", bool1, bool2));
                arr[y][x] = true;
            }
        }
        return list;
    }

    static List<String> recList1 (boolean[][] arr, int x, int y, String str) {
        List<String> list = new ArrayList<>(0);
        if (x == 0 && y == 0) {
            list.add(str);
            return list;
        }
        if (x < 0 || x ≥ arr.length || y < 0 || y ≥ arr[0].length || !arr[x][y]) {
            return list;
        }
        arr[x][y] = false;
        list.addAll(recList1(arr, x + 1, y, str + "E"));
        list.addAll(recList1(arr, x - 1, y, str + "W"));
        list.addAll(recList1(arr, x, y + 1, str + "S"));
        list.addAll(recList1(arr, x, y - 1, str + "N"));
        arr[x][y] = true;
        return list;
    }

    static List<String> recList2(boolean[][] arr, int x, int y, String str) {
        List<String> list = new ArrayList<>();
        if (x == 0 && y == 0) {
            list.add(str);
            return list;
        }

        if (!arr[Math.abs(x)][Math.abs(y)]) {
            return list;
        }
        arr[Math.abs(x)][Math.abs(y)] = false;
        if (x < 0) {
            list.addAll(recList2(arr, x + 1, y, str + "E"));
        } if (x > 0) {
            list.addAll(recList2(arr, x - 1, y, str + "W"));
        } if (y < 0) {
            list.addAll(recList2(arr, x, y + 1, str + "S"));
        } if (y > 0) {
```

```
            list.addAll(recList2(arr, x, y - 1, str + "N"));
        }
        arr[Math.abs(x)][Math.abs(y)] = true;
        return list;
    }
}
```

Q6. Maze with all the paths

```
package com.inclass;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class AllPath {
    public static void main(String[] args) {
        /* N --> North, S --> South, E --> East, W --> West
        Ne --> North-east, Nw --> North-west, Se --> South-east, Sw --> South-west */
        int x1, y1, x2, y2;

        // start
        x1 = 1; y1 = 1;
        // target
        x2 = 3; y2 = 3;
        allPath(x1, x2, y1, y2);

        // start
        x1 = 1; y1 = 3;
        // target
        x2 = 3; y2 = 1;
        allPath(x1, x2, y1, y2);

        // start
        x1 = 3; y1 = 1;
        // target
        x2 = 1; y2 = 3;
        allPath(x1, x2, y1, y2);

        // start
        x1 = 3; y1 = 3;
        // target
        x2 = 1; y2 = 1;
        allPath(x1, x2, y1, y2);

    }

    static void allPath (int x1, int x2, int y1, int y2) {
        recList(new int[Math.abs(y1 - y2) + 1][Math.abs(x1 - x2) + 1], x1 - x2, y1 - y2, " ", x1
- x2 > 0 , y1 - y2 > 0, 0);
        System.out.println("═══════════════════════════════");
    }

    static void recList (int[][] arr, int x, int y, String str, boolean bool1, boolean bool2,
int count) {
        if (x == 0 && y == 0) {
```

```
            System.out.print("→");
            System.out.println(Arrays.toString(arr[0]));
            for(int i = 1; i < arr.length; i++) {
                System.out.print("  ");
                System.out.println(Arrays.toString(arr[i]));
            }
            System.out.println("  " + str);
            arr = new int[arr[0].length][arr.length];
            return;
        } if (!bool1) {
            if (!bool2) {
                if (-x < 0 || -x ≥ arr.length || -y < 0 || -y ≥ arr[0].length ||
arr[arr[0].length + y - 1][arr.length + x - 1] ≠ 0){
                    return;
                }
                arr[arr[0].length + y - 1][arr.length + x - 1] = ++count;
                recList(arr, x + 1, y, str + "E", bool1, bool2, count);
                recList(arr, x - 1, y, str + "W", bool1, bool2, count);
                recList(arr, x, y + 1, str + "S", bool1, bool2, count);
                recList(arr, x, y - 1, str + "N", bool1, bool2, count);
                arr[arr[0].length + y - 1][arr.length + x - 1] = 0;
                --count;
                return;
            } if (bool2) {
                if (-x < 0 || -x ≥ arr.length ||  y < 0 || y ≥ arr[0].length || arr[y]
[arr.length + x - 1] ≠ 0) {
                    return;
                }
                arr[y][arr.length + x - 1] = ++count;
                recList(arr, x + 1, y, str + "E", bool1, bool2, count);
                recList(arr, x - 1, y, str + "W", bool1, bool2, count);
                recList(arr, x, y + 1, str + "S", bool1, bool2, count);
                recList(arr, x, y - 1, str + "N", bool1, bool2, count);
                arr[y][arr.length + x - 1] = 0;
                --count;
                return;
            }
        } if (bool1) {
            if (!bool2) {
                if (x < 0 || x ≥ arr.length || -y < 0 || -y ≥ arr[0].length ||
arr[arr[0].length + y - 1][x] ≠ 0) {
                    return;
                }
                arr[arr[0].length + y - 1][x] = ++count;
                recList(arr, x + 1, y, str + "E", bool1, bool2, count);
                recList(arr, x - 1, y, str + "W", bool1, bool2, count);
                recList(arr, x, y + 1, str + "S", bool1, bool2, count);
                recList(arr, x, y - 1, str + "N", bool1, bool2, count);
                arr[arr[0].length + y - 1][x] = 0;
                --count;
                return;
            } if (bool2) {
                if (x < 0 || x ≥ arr.length ||  y < 0 || y ≥ arr[0].length || arr[y][x] ≠ 0)
{
                    return;
                }
                arr[y][x] = ++count;
                recList(arr, x + 1, y, str + "E", bool1, bool2, count);
                recList(arr, x - 1, y, str + "W", bool1, bool2, count);
                recList(arr, x, y + 1, str + "S", bool1, bool2, count);
```

```
                recList(arr, x, y - 1, str + "N", bool1, bool2, count);
                arr[y][x] = 0;
                --count;
                return;
            }
        }
    }

}
```

Q7. N Queens

- Solution 1:

```
package com.inclass;

import java.util.ArrayList;
import java.util.List;

public class NQueen2 {
    public static void main(String[] args) {
        int n = 5;
        for (List<String> list : solveNQueens(n)) {
            System.out.println(list);
        }
        /*for (List<Integer> list: nQueens(new ArrayList<>(), n, -1, -1)) {
            System.out.println(list);
        }*/
    }


    static List<List<String>> solveNQueens (int n) {
        return converter(nQueens(new ArrayList<>(), n , -1, -1), n);
    }

    static List<List<Integer>> nQueens (List<Integer> board, int n, int i, int j) {
        List<List<Integer>> answers = new ArrayList<>();
        if (i == -1) {
            while (j++ < n - 1 && i < n) {
                answers.addAll(nQueens(board, n, i + 1, j));
            }
            return answers;
        }
        if (!safety(board, i, j)) {
            return new ArrayList<>();
        }
        board.add(j);
        j = -1;
        if (i == n - 1) {
            answers.add(new ArrayList<>(board));
            board.remove(board.size() - 1);
            return answers;
        }
        while (j++ < n - 1 && i < n) {
            answers.addAll(nQueens(board, n, i + 1, j));
        }
        board.remove(board.size() - 1);
```

```
        return answers;
    }

    static boolean safety(List<Integer> board, int i, int j) {
        for (int k = 0; k < board.size(); k++) {
            if (Math.abs(Math.subtractExact(i, k)) == Math.abs(Math.subtractExact(j,
board.get(k))) || i == k || j == board.get(k)) {
                return false;
            }
        }
        return true;
    }

    static List<List<String>> converter(List<List<Integer>> list, int n) {
        List<List<String>> board = new ArrayList<>();
        for (int i = 0; i < list.size(); i++) {
            board.add(new ArrayList<>());
            for (int j = 0; j < list.get(i).size(); j++) {
                String str = "";
                for (int k = 0; k < list.get(i).get(j); k++) {
                    str += ".";
                }
                str += "Q";
                for (int k = list.get(i).get(j) + 1; k < n; k++) {
                    str += ".";
                }
                board.get(i).add(str);
            }
        }
        return board;
    }
}
```

– Solution 2:

```
package com.inclass;

public class NQueens3 {
    public static void main(String[] args) {
        int n = 4;
        boolean[][] board = new boolean[n][n];
        System.out.println(queens(board, 0));
    }

    static int queens(boolean[][] board, int row) {
        if (row == board.length) {
            display(board);
            System.out.println();
            return 1;
        }

        int count = 0;

        // placing the queen and checking for every row and col
        for (int col = 0; col < board.length; col++) {
            // place the queen if it is safe
            if(isSafe(board, row, col)) {
                board[row][col] = true;
```

```
                count += queens(board, row + 1);
                board[row][col] = false;
            }
        }

        return count;
    }

    private static boolean isSafe(boolean[][] board, int row, int col) {
        // check vertical row
        for (int i = 0; i < row; i++) {
            if (board[i][col]) {
                return false;
            }
        }

        // diagonal left
        int maxLeft = Math.min(row, col);
        for (int i = 1; i ≤ maxLeft; i++) {
            if(board[row-i][col-i]) {
                return false;
            }
        }

        // diagonal right
        int maxRight = Math.min(row, board.length - col - 1);
        for (int i = 1; i ≤ maxRight; i++) {
            if(board[row-i][col+i]) {
                return false;
            }
        }

        return true;
    }

    private static void display(boolean[][] board) {
        for(boolean[] row : board) {
            for(boolean element : row) {
                if (element) {
                    System.out.print("Q ");
                } else {
                    System.out.print("X ");
                }
            }
            System.out.println();
        }
    }
}
```

Q8. N Knights

```
package com.inclass;

import java.util.List;

public class NKnights {
    public static void main(String[] args) {
        int m = 3;
        int n = 3;
```

```java
        int k = 5;
        boolean[][] board = new boolean[m][n];
        nKnights(board, 0, 0, k);
    }

    static void nKnights (boolean[][] board, int row, int col, int k) {
        if (k == 0) {
            display(board);
            System.out.println();
            return;
        }

        if (row == board.length - 1 && col == board.length) {
            return;
        }

        if (col == board.length) {
            nKnights(board, row + 1, 0, k);
            return;
        }

        if (isSafe(board, row, col)) {
            board[row][col] = true;
            nKnights(board, row, col + 1, k - 1);
            board[row][col] = false;
        }

        nKnights(board, row, col + 1, k);
    }

    static void display (boolean[][] board) {
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[i].length; j++) {
                if (board[i][j]) {
                    System.out.print("K ");
                } else {
                    System.out.print("_ ");
                }
            }
            System.out.println();
        }
        System.out.println();
    }

    static boolean isSafe (boolean[][] board, int i, int j) {
        for (int p = 0; p < board.length; p++) {
            for (int q = 0; q < board[p].length; q++) {
                if (board[p][q]) {
                    int a = Math.abs(Math.subtractExact(p, i));
                    int b = Math.abs(Math.subtractExact(q, j));
                    if (a == 2 && b == 1 || a == 1 && b == 2) {
                        return false;
                    }
                }
            }
        }
        return true;
    }
```

```
}
```