Assignment 2: Basics to Backtracking - Java Solutions

1. Print N-bit Binary Numbers having more 1s than 0s

```
class NBitBinaryNumbers {
    static void generate(int n, int ones, int zeros, String output) {
        if (n == 0) {
            System.out.println(output);
            return;
        }
        generate(n - 1, ones + 1, zeros, output + "1");
        if (ones > zeros) {
            generate(n - 1, ones, zeros + 1, output + "0");
        }
    }

    public static void main(String[] args) {
        int N = 3;
        System.out.println("N-bit binary numbers having more 1s than 0s:");
        generate(N, 0, 0, "");
    }
}
```

2. Delete Middle Element of a Stack using Recursion

```
import java.util.Stack;

class DeleteMiddleOfStack {
    static void deleteMiddle(Stack<Integer> stack, int size, int current) {
        if (stack.isEmpty() || current == size) return;

        int x = stack.pop();
        deleteMiddle(stack, size, current + 1);

        if (current != size / 2) stack.push(x);
    }

    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        for (int i = 1; i <= 5; i++) stack.push(i);

        System.out.println("Original Stack: " + stack);
        deleteMiddle(stack, stack.size(), 0);
        System.out.println("After Deleting Middle: " + stack);
    }
}
```

3. Letter Combinations of a Phone Number (LeetCode 17)

```
import java.util.*;

class LetterCombinationsPhone {
    static final String[] KEYS = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};

    static void backtrack(String digits, int index, String current, List<String> result) {
        if (index == digits.length()) {
            if (!current.isEmpty()) result.add(current);
            return;
        }

        String letters = KEYS[digits.charAt(index) - '0'];
        for (char c : letters.toCharArray()) {
            backtrack(digits, index + 1, current + c, result);
        }
    }

    public static void main(String[] args) {
        String digits = "23";
        List<String> result = new ArrayList<>();
        backtrack(digits, 0, "", result);
        System.out.println("Letter combinations: " + result);
    }
}
```

4. Combinations (LeetCode 77)

```
import java.util.*;

class Combinations {
    static void combineHelper(int n, int k, int start, List<Integer> current, List<List<Integer>> res
        if (current.size() == k) {
            result.add(new ArrayList<>(current));
            return;
        }

        for (int i = start; i <= n; i++) {
            current.add(i);
            combineHelper(n, k, i + 1, current, result);
            current.remove(current.size() - 1);
        }
    }

    public static void main(String[] args) {
        int n = 4, k = 2;
        List<List<Integer>> result = new ArrayList<>();
        combineHelper(n, k, 1, new ArrayList<>(), result);
        System.out.println("Combinations: " + result);
    }
}
```

5. Letter Tile Possibilities (LeetCode 1079)

```
import java.util.*;

class LetterTilePossibilities {
    static int count = 0;

    static void backtrack(String tiles, boolean[] used, String current) {
        if (!current.isEmpty()) count++;

        for (int i = 0; i < tiles.length(); i++) {
            if (used[i]) continue;
            if (i > 0 && tiles.charAt(i) == tiles.charAt(i - 1) && !used[i - 1]) continue;

            used[i] = true;
            backtrack(tiles, used, current + tiles.charAt(i));
            used[i] = false;
        }
    }

    public static void main(String[] args) {
        String tiles = "AAB";
        char[] chars = tiles.toCharArray();
        Arrays.sort(chars);
        backtrack(new String(chars), new boolean[chars.length], "");
        System.out.println("Number of possible sequences: " + count);
    }
}
```