DSA Stack & Queue Solutions (Q1 - Q8)

Q1: Bracket Matching Using Stack and Queue

```
class Q1_BracketMatching {
    public static void main(String[] args) {
        Stack<Character> openBrackets = new Stack<>();
        openBrackets.push('(');
        openBrackets.push('[');
        openBrackets.push('{');

        Queue<Character> closeBrackets = new Queue<>();
        closeBrackets.enqueue(')');
        closeBrackets.enqueue(']');
        closeBrackets.enqueue('}');

        Stack<String> opStack = new Stack<>();

        while (!openBrackets.isEmpty() && !closeBrackets.isEmpty()) {
            char open = openBrackets.pop();
            char close = closeBrackets.dequeue();

            if ((open == '(' && close == ')') ||
                (open == '[' && close == ']') ||
                (open == '{' && close == '}')) {
                opStack.push("" + open + close);
            }
        }
        System.out.println(opStack);
    }
}
```

Q2: Special Character Processing and String Manipulation

```
class Q2_SpecialCharProcessing {
    public static void main(String[] args) {
        Queue<String> ipQueue = new Queue<>();
        ipQueue.enqueue("Ca:t");
        ipQueue.enqueue("Do:g");
        ipQueue.enqueue("Fi:sh");
        ipQueue.enqueue("Parr:ot");
        ipQueue.enqueue("Bi:rd");

        Stack<String> opStack = new Stack<>();

        while (!ipQueue.isEmpty()) {
            String word = ipQueue.dequeue();
            String[] parts = word.split(":");
            if (parts[0].length() == parts[1].length()) {
                opStack.push(word);
            } else {
                String modified = (parts[0] + parts[1]);
                String reversed = new StringBuilder(modified).reverse().toString();
                opStack.push(reversed);
            }
        }
        System.out.println(opStack);
    }
}
```

Q3: Matching Conditions Between Stack and Queue

```
class Q3_MatchingConditions {
    public static void main(String[] args) {
        Stack<Integer> numStack = new Stack<>();
        numStack.push(3);
        numStack.push(3);
        numStack.push(4);
        numStack.push(2);
        numStack.push(1);

        Queue<String> wordQueue = new Queue<>();
        wordQueue.enqueue("Orange");
        wordQueue.enqueue("Eagle");
        wordQueue.enqueue("Dog");
        wordQueue.enqueue("Banana");
        wordQueue.enqueue("Owl");

        Stack<String> opStack = new Stack<>();
        Stack<String> tempStack = new Stack<>();

        while (!wordQueue.isEmpty()) {
            String word = wordQueue.dequeue();
            int vowels = countVowels(word);

            if (!numStack.isEmpty() && vowels == numStack.peek()) {
                opStack.push(word + numStack.pop());
            } else {
                tempStack.push(word);
            }
        }

        // Move from tempStack to opStack in same order
        Stack<String> reverse = new Stack<>();
        while (!tempStack.isEmpty()) reverse.push(tempStack.pop());
        while (!reverse.isEmpty()) opStack.push(reverse.pop());

        System.out.println(opStack);
    }

    static int countVowels(String s) {
        int count = 0;
        for (char c : s.toUpperCase().toCharArray()) {
            if ("AEIOU".indexOf(c) != -1) count++;
        }
        return count;
    }
}
```

Q4: Reversing and Combining Data

```
class Q4_ReversingCombining {
    public static void main(String[] args) {
        Queue<Integer> inputQueue = new Queue<>();
        inputQueue.enqueue(12);
        inputQueue.enqueue(34);
        inputQueue.enqueue(56);
        inputQueue.enqueue(78);
        inputQueue.enqueue(90);

        Stack<Integer> inputStack = new Stack<>();
        inputStack.push(5);
        inputStack.push(4);
        inputStack.push(3);
        inputStack.push(2);
        inputStack.push(1);

        Stack<String> opStack = new Stack<>();

        while (!inputQueue.isEmpty() && !inputStack.isEmpty()) {
            int qVal = inputQueue.dequeue();
            int sVal = inputStack.pop();
            opStack.push(qVal + ":" + sVal);
        }

        System.out.println(opStack);
    }
}
```

Q5: Alternating Stack and Queue Merging

```
class Q5_AlternatingMerge {
    public static void main(String[] args) {
        Stack<Integer> numStack = new Stack<>();
        numStack.push(10);
        numStack.push(20);
        numStack.push(30);
        numStack.push(40);

        Queue<Integer> numQueue = new Queue<>();
        numQueue.enqueue(5);
        numQueue.enqueue(15);
        numQueue.enqueue(25);
        numQueue.enqueue(35);

        Stack<Integer> resultStack = new Stack<>();

        while (!numStack.isEmpty() || !numQueue.isEmpty()) {
            if (!numStack.isEmpty()) resultStack.push(numStack.pop());
            if (!numQueue.isEmpty()) resultStack.push(numQueue.dequeue());
        }

        System.out.println(resultStack);
    }
}
```

Q6: Priority Sorting Using Stacks

```
import java.util.*;

class Q6_PrioritySorting {
    public static void main(String[] args) {
        Stack<Integer> priorityStack = new Stack<>();
        priorityStack.push(3);
        priorityStack.push(1);
        priorityStack.push(4);
        priorityStack.push(2);

        Queue<String> taskQueue = new Queue<>();
        taskQueue.enqueue("TaskA");
        taskQueue.enqueue("TaskB");
        taskQueue.enqueue("TaskC");
        taskQueue.enqueue("TaskD");

        List<String> paired = new ArrayList<>();
        while (!taskQueue.isEmpty() && !priorityStack.isEmpty()) {
            paired.add(taskQueue.dequeue() + priorityStack.pop());
        }

        paired.sort((a, b) -> b.charAt(b.length()-1) - a.charAt(a.length()-1));

        Stack<String> opStack = new Stack<>();
        for (String s : paired) opStack.push(s);

        System.out.println(opStack);
    }
}
```

Q7: Special Character Replacement

```
class Q7_SpecialCharReplacement {
    public static void main(String[] args) {
        Queue<String> ipQueue = new Queue<>();
        ipQueue.enqueue("He@llo");
        ipQueue.enqueue("Wor#ld");
        ipQueue.enqueue("Pyt%hon");
        ipQueue.enqueue("Stac&k");
        ipQueue.enqueue("Que*ue");

        Stack<String> opStack = new Stack<>();
        while (!ipQueue.isEmpty()) {
            String word = ipQueue.dequeue().replaceAll("[^a-zA-Z]", "-");
            int vowels = countVowels(word);
            if (vowels % 2 == 0) {
                opStack.push(word);
            } else {
                ipQueue.enqueue(word);
            }
        }

        System.out.println("OpStack: " + opStack);
        System.out.println("IpQueue: " + ipQueue);
    }

    static int countVowels(String s) {
        int count = 0;
        for (char c : s.toUpperCase().toCharArray()) {
            if ("AEIOU".indexOf(c) != -1) count++;
        }
        return count;
    }
}
```

Q8: Alternating Case Transformation

```
class Q8_AlternatingCase {
    public static void main(String[] args) {
        Queue<String> ipQueue = new Queue<>();
        ipQueue.enqueue("hello");
        ipQueue.enqueue("world");
        ipQueue.enqueue("stack");
        ipQueue.enqueue("queue");
        ipQueue.enqueue("python");

        Stack<String> opStack = new Stack<>();

        while (!ipQueue.isEmpty()) {
            String word = ipQueue.dequeue();
            String transformed = toAltCase(word);
            if (transformed.length() % 2 == 0) {
                opStack.push(transformed);
            } else {
                ipQueue.enqueue(transformed);
            }
        }

        System.out.println("OpStack: " + opStack);
        System.out.println("IpQueue: " + ipQueue);
    }

    static String toAltCase(String s) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < s.length(); i++) {
            sb.append(i % 2 == 0 ? Character.toUpperCase(s.charAt(i)) : Character.toLowerCase(s.char
        }
        return sb.toString();
    }
}
```