1)
```
import java.util.HashMap;

public class Main {
    public static int[] findSum(int[] arr, int n) {
        HashMap<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < arr.length; i++) {
            int complement = n - arr[i];
            if (map.containsKey(complement)) {
                return new int[] { arr[i], complement };
            }
            map.put(arr[i], i);
        }
        return arr;
    }
}
```

2)
```
public class Main {
    public static int findMinimum(int[] arr) {
        int min = arr[0];
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] < min) {
                min = arr[i];
            }
        }
        return min;
    }
}
```

3)
```
class Node<T> {
    T data;
    Node<T> next;

    Node(T data) {
        this.data = data;
        this.next = null;
    }
}

class LinkedList<T> {
    Node<T> head;

    void insertAtEnd(T data) {
        Node<T> newNode = new Node<>(data);
        if (head == null) {
            head = newNode;
            return;
        }
        Node<T> current = head;
        while (current.next != null) {
            current = current.next;
        }
        current.next = newNode;
    }
}
```

4)
```
public class Main {
    public static String reverseWords(String str) {
        String[] words = str.split("\\s+");
        StringBuilder sb = new StringBuilder();
        for (int i = words.length - 1; i >= 0; i--) {
            sb.append(words[i]);
            sb.append(" ");
        }
        return sb.toString().trim();
    }
}
```

5)
```
public class Main {
    public static List<String> findPalindromes(String s) {
        List<String> result = new ArrayList<>();
        for (int i = 0; i < s.length(); i++) {
            for (int j = i + 1; j <= s.length(); j++) {
                String sub = s.substring(i, j);
                if (sub.length() > 1 && isPalindrome(sub)) {
                    result.add(sub);
                }
            }
        }
        return result;
    }

    private static boolean isPalindrome(String s) {
        int i = 0;
        int j = s.length() - 1;
        while (i < j) {
            if (s.charAt(i++) != s.charAt(j--)) {
                return false;
            }
        }
        return true;
    }
}
```

6)
```
public class Main {
    public static int totalFruit(int[] tree) {
        int max = 0;
        int start = 0;
        int end = 0;
        Map<Integer, Integer> count = new HashMap<>();
        while (end < tree.length) {
            if (count.size() <= 2) {
                count.put(tree[end], end++);
            }
            if (count.size() > 2) {
                int min = Integer.MAX_VALUE;
                for (int val : count.values()) {
                    min = Math.min(min, val);
                }
                start = min + 1;
                count.remove(tree[min]);
            }
            max = Math.max(max, end - start);
        }
        return max;
    }
}
```

7)

```
public class Main {
    public static int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];
            if (map.containsKey(complement)) {
                return new int[] { map.get(complement), i };
            }
            map.put(nums[i], i);
        }
        return new int[] {};
    }
}
```

8)

```
public class Main {
    public static int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];
            if (map.containsKey(complement)) {
                return new int[] { map.get(complement), i };
            }
            map.put(nums[i], i);
        }
        return new int[] {};
    }
}
```

9)

```
public class Main {
    public static int[] nextGreaterElement(int[] nums) {
        int n = nums.length;
        int[] result = new int[n];
        Arrays.fill(result, -1);
        Stack<Integer> stack = new Stack<>();
        for (int i = 0; i < n; i++) {
            while (!stack.isEmpty() && nums[stack.peek()] < nums[i]) {
                result[stack.pop()] = nums[i];
            }
            stack.push(i);
        }
        return result;
    }
}
```