Q1: Program to Print the Sum of All Elements Present on Even Indices in the Given Array

Eg:-

```
public class SumEvenIndices {
    public static void main(String[] args) {
        int[] arr1 = {3, 20, 4, 6, 9};
        int[] arr2 = {4, 3, 6, 7, 1};

        System.out.println("Sum for arr1: " + sumOfEvenIndices(arr1));  // Output: 16
        System.out.println("Sum for arr2: " + sumOfEvenIndices(arr2));  // Output: 11
    }

    public static int sumOfEvenIndices(int[] arr) {
        int sum = 0;
        for (int i = 0; i < arr.length; i += 2) {  // Iterate through even indices
            sum += arr[i];
        }
        return sum;
    }
}
```

Eg:-

Q2: Program to Traverse Over the Elements of the Array Using a `for-each` Loop and Print All Even Elements

Eg:-

```
public class PrintEvenElements {
    public static void main(String[] args) {
        int[] arr1 = {34, 21, 54, 65, 43};
        int[] arr2 = {4, 3, 6, 7, 1};

        System.out.print("Even elements in arr1: ");
        printEvenElements(arr1);  // Output: 34 54

        System.out.print("Even elements in arr2: ");
        printEvenElements(arr2);  // Output: 4 6
    }

    public static void printEvenElements(int[] arr) {
        for (int element : arr) {  // Traverse using for-each loop
            if (element % 2 == 0) {  // Check if element is even
                System.out.print(element + " ");
            }
        }
```

```
        System.out.println();
    }
}
```
Eg:-

Q3: Program to Calculate the Maximum Element in the Array

Eg:-

```
public class MaxElement {
    public static void main(String[] args) {
        int[] arr1 = {34, 21, 54, 65, 43};
        int[] arr2 = {4, 3, 6, 7, 1};

        System.out.println("Maximum element in arr1: " + findMax(arr1));  // Output: 65
        System.out.println("Maximum element in arr2: " + findMax(arr2));  // Output: 7
    }

    public static int findMax(int[] arr) {
        int max = arr[0];  // Assume the first element is the maximum
        for (int element : arr) {
            if (element > max) {
                max = element;  // Update max if a larger element is found
            }
        }
        return max;
    }
}
```
Eg:-

Q4: Program to Find Out the Second Largest Element in a Given Array

Eg:-

```
public class SecondLargestElement {
    public static void main(String[] args) {
        int[] arr1 = {34, 21, 54, 65, 43};
        int[] arr2 = {4, 3, 6, 7, 1};

        System.out.println("Second largest element in arr1: " + findSecondLargest(arr1));  //
Output: 54
        System.out.println("Second largest element in arr2: " + findSecondLargest(arr2));  //
Output: 6
    }

    public static int findSecondLargest(int[] arr) {
```

```
        if (arr.length < 2) {
            return -1;  // Return -1 if there are not enough elements
        }

        int firstMax = Integer.MIN_VALUE;
        int secondMax = Integer.MIN_VALUE;

        for (int element : arr) {
            if (element > firstMax) {
                secondMax = firstMax;  // Update secondMax before firstMax
                firstMax = element;
            } else if (element > secondMax && element != firstMax) {
                secondMax = element;  // Update secondMax if it's greater than current secondMax
            }
        }

        return secondMax;
    }
}
```

To solve the problem of finding the first peak element in an array, we need to iterate through the array and check each element to see if it is greater than both its immediate left and right neighbors.

Q5: Program to Find the First Peak Element in the Array
Eg:-

```
public class FirstPeakElement {
    public static void main(String[] args) {
        int[] arr1 = {1, 3, 2, 6, 5};
        int[] arr2 = {14, 7, 3, 2, 6, 5};

        System.out.println("First peak element in arr1: " + findFirstPeak(arr1));  // Output: 6
        System.out.println("First peak element in arr2: " + findFirstPeak(arr2));  // Output: 7
    }

    public static int findFirstPeak(int[] arr) {
        // If array has less than 3 elements, it can't have a peak element with both left and right
neighbors
        if (arr.length < 3) {
            return -1;  // No peak can be found
        }

        // Traverse the array from the second element to the second last element
        for (int i = 1; i < arr.length - 1; i++) {
```

```
            // Check if the current element is a peak element
            if (arr[i] > arr[i - 1] && arr[i] > arr[i + 1]) {
                return arr[i];  // Return the first peak element found
            }
        }

        // If no peak element is found, return -1
        return -1;
    }
}
```