Q-1
To calculate the sum of the first 10 multiples of a given number `n`, you can use a direct mathematical approach. The sum of the first 10 multiples of `n` can be computed using the formula for the sum of an arithmetic series.

### Explanation:

The multiples of `n` are:
- \(n \times 1, n \times 2, n \times 3, \dots, n \times 10 \)

The sum can be calculated as:
\[n \times (1 + 2 + 3 + \dots + 10) \]

The sum of the first 10 natural numbers (1 through 10) is:
\[\text{Sum} = \frac{10 \times (10 + 1)}{2} = 55 \]

Thus, the sum of the first 10 multiples of `n` is:
\[\text{Result} = n \times 55 \]

This is the most optimal solution, as it reduces the problem to a single multiplication after computing the sum of the first 10 natural numbers.

### Python Implementation:
```python
def sum_of_multiples(n):
    return n * 55

# Input
n = int(input("Enter a number: "))

# Output
result = sum_of_multiples(n)
print(result)
```

### Java Implementation:
```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int n = sc.nextInt();

        int result = n * 55;

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

### C++ Implementation:
```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;

    int result = n * 55;

    cout << result << endl;

    return 0;
}
```

### Explanation:
- **Optimal Solution**: The approach is optimal because it leverages the known formula for the sum of the first `k` natural numbers to minimize the number of operations to a single multiplication.
- **Complexity**: The time complexity is \(O(1) \), which is constant time, and the space complexity is also \(O(1) \).

Q-2

Sure! Here's how you can convert the given C++ code to Python and Java.

### Python Version:
```python
from collections import deque
import heapq
```

```
def max_sliding_window(arr, k):
    result = []
    window = deque()

    for i in range(len(arr)):
        while window and window[0] <= i - k:
            window.popleft()

        while window and arr[window[-1]] <= arr[i]:
            window.pop()

        window.append(i)

        if i >= k - 1:
            result.append(arr[window[0]])

    return result

# Input
n = int(input())
arr = [int(input()) for _ in range(n)]
k = int(input())

# Processing
result = max_sliding_window(arr, k)

# Output
print(" ".join(map(str, result)))
```

### Java Version:
```java
import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();
        int[] arr = new int[n];
        for(int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        int k = sc.nextInt();
        List<Integer> result = new ArrayList<>();
        Deque<Integer> window = new LinkedList<>();

        for(int i = 0; i < n; i++) {
            while(!window.isEmpty() && window.peek() <= i - k) {
                window.poll();
            }

            while(!window.isEmpty() && arr[window.peekLast()] <= arr[i]) {
                window.pollLast();
            }

            window.offer(i);

            if(i >= k - 1) {
                result.add(arr[window.peek()]);
            }
        }

        for(int i : result) {
            System.out.print(i + " ");
        }

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

C++

```
/******************************************************************************

                              Online C++ Compiler.
               Code, Compile, Run and Debug C++ program online.
Write your code in this editor and press "Run" button to compile and execute it.

*******************************************************************************/

#include <bits/stdc++.h>
using namespace std;
```

```
int main()
{

   int n;
   cin>>n;
   vector<int>arr;
   for(int i=0;i<n;i++){
       int element;
       cin>>element;
       arr.push_back(element);

   }
   int k;
   cin>>k;
   vector<int>result;
   multiset<int>window;

   for(int i=0;i<k;i++){

       window.insert(arr[i]);

   }


   for(int i=k;i<=n;i++){
       result.push_back(*window.rbegin());

       if(i<arr.size()){
           window.erase(window.find(arr[i-k]));
           window.insert(arr[i]);

       }
   }

   for(auto i:result){
       cout<<i<<" ";
   }

   cout<<endl;
```

```
    return 0;
}
```

### Explanation:

- **Python Version**:
  - Used `deque` from `collections` to keep track of indices of the current sliding window.
  - We maintain the indices in the `deque` in such a way that the maximum element's index is always at the front.
  - At each step, we remove elements from the front if they are out of the current window, and from the back if they are smaller than the current element, ensuring that the largest element in the current window is always at the front.

- **Java Version**:
  - Similar logic as Python but implemented using Java's `Deque` and `ArrayList`.
  - We maintain a sliding window using the deque and store the maximum element's index in the window.
  - The result is stored in an `ArrayList` and printed at the end.

These conversions closely follow the logic in the provided C++ code while adapting to the syntax and idioms of Python and Java.