# Cheat Sheet: Array Basics, Prefix Sum, Sliding Window

## 1. Introduction

Topics Covered:

- Basics of arrays
- Reading and writing array elements
- Looping through arrays
- Changing arrays into Lists or Streams
- Sorting, searching, and merging arrays
- **Array Basics, Prefix Sum, and Sliding Window (Java DSA)**

## 2. What Is an Array?

In Java, an array is a container that holds a fixed number of values, all of the same type. Each value is stored in a numbered slot called an index, starting from 0.

You can have arrays of basic types like `int`, `float`, and `boolean`, and also arrays of objects like `String`, `Object`, or custom classes.

## 3. Setting Up an Array

### 3.1 Declaration

There are two ways to declare an array:

```
int[] myArray;
int myArray[];
```

### 3.2 Initialization

You can create and fill an array like this:

```
int[] myArray = new int[10];  // Creates an array of 10 ints, all initialized to 0
```

Or create and assign values directly:

```
int[] myArray = new int[] {1, 2, 3, 4, 5};
```

The size is automatically set based on the number of values you give.

## 4. Array Length

An array's length is fixed once it's created. To find out how many elements an array has, use:

```
myArray.length
```

For example:

```
System.out.println(myArray.length);  // Outputs: 5
```

You can also use:

```
Array.getLength(myArray);
```

This works when working with arrays as generic objects.

## 5. Accessing Elements

Use the index to access a specific item:

```
myArray[0] = 10;
System.out.println(myArray[0]);
```

**Note**: If you use a negative index or an index that's too large, Java will throw an `ArrayIndexOutOfBoundsException`.

## 6. Looping Over an Array

Use a `for` loop:

```
for (int i = 0; i < myArray.length; i++) {
    System.out.println(myArray[i]);
}
```

Or a `for-each` loop:

```
for (int element : myArray) {
    System.out.println(element);
}
```

The for-each loop is easier if you don't need the index

## 7. Varargs (Variable Arguments)

You can pass a flexible number of values to a method using `...` (varargs):

```
void printItems(String... items) {
    for (String item : items) {
        System.out.println(item);
```

```
    }
}

printItems("Milk", "Eggs", "Bread");
```

You can also pass an array instead:

```
String[] groceries = {"Milk", "Eggs"};
printItems(groceries);
```

## 8. Convert Array to List

Method 1: Manual way

```
List<Integer> list = new ArrayList<>();
for (int num : myArray) {
    list.add(num);
}
```

Method 2: Built-in method

```
Integer[] nums = {1, 2, 3};
List<Integer> list = Arrays.asList(nums);
```

This method doesn't work with primitive arrays like `int[]`, and you can't add/remove elements.

## 9. Convert Array to Stream

You can turn an array into a Stream:

```
String[] items = {"Milk", "Bread"};
Stream<String> stream = Arrays.stream(items);
```

Or only a part of the array:

```
Stream<String> subStream = Arrays.stream(items, 1, 2); // From index 1 to 2
(exclusive)
```

## 10. Sorting Arrays

Sort numbers:

```
int[] nums = {5, 2, 9};
Arrays.sort(nums); // {2, 5, 9}
```

Sort objects:

```
String[] names = {"Tom", "Anna"};
Arrays.sort(names); // Alphabetical order
```

Sort using a custom rule:

```
Arrays.sort(names, Comparator.reverseOrder());
```

## 11. Searching in Arrays

Linear search (simple loop):

```
for (int i = 0; i < nums.length; i++) {
    if (nums[i] == 4) {
        System.out.println("Found at index " + i);
        break;
    }
}
```

Binary search (for sorted arrays):

```
int index = Arrays.binarySearch(nums, 4);
```

## 12. Concatenating Arrays

Method 1: Manual loop

```
int[] result = new int[arr1.length + arr2.length];
for (int i = 0; i < result.length; i++) {
    result[i] = (i < arr1.length) ? arr1[i] : arr2[i - arr1.length];
}
```

Method 2: Using `Arrays.setAll`

```
Arrays.setAll(result, i -> (i < arr1.length ? arr1[i] : arr2[i -
arr1.length]));
```

Method 3: Using `System.arraycopy`

```
System.arraycopy(arr1, 0, result, 0, arr1.length);
System.arraycopy(arr2, 0, result, arr1.length, arr2.length);
```

## 13. Array Basics, Prefix Sum, and Sliding Window (Java DSA)1. Array Basics

**Definition:**

An array is a fixed-size container that holds elements of the same data type in contiguous memory.

**Declaration & Initialization:**

```
int[] arr = new int[5];              // Default values
int[] nums = {1, 2, 3, 4, 5};        // Initialized directly
```

**Access & Update:**

```
int val = nums[2];      // Access index 2
nums[1] = 10;           // Update index 1
```

**Traversal:**

```
for (int i = 0; i < nums.length; i++) System.out.println(nums[i]);

for (int num : nums) System.out.println(num);  // Enhanced for-loop
```

## 2. Prefix Sum (Cumulative Sum)

**Idea:**
Prefix sum stores the sum of elements up to a certain index to allow range sum queries in O(1) time.

**Prefix Array Creation:**

```
int[] prefix = new int[arr.length];
prefix[0] = arr[0];
for (int i = 1; i < arr.length; i++) {
    prefix[i] = prefix[i - 1] + arr[i];
}
```

**Range Sum Query [l, r]:**

```
int sum = prefix[r] - (l > 0 ? prefix[l - 1] : 0);
```

**Use Cases:**

- Range sum queries
- Difference arrays
- Subarray problems

## 3. Sliding Window

**Idea:**
Used to find subarrays of a fixed size with optimal time by avoiding recalculations.

**Fixed Window Size (e.g., max sum of subarray of size k):**

```
int maxSum = 0, windowSum = 0, k = 3;

for (int i = 0; i < k; i++) windowSum += arr[i];
maxSum = windowSum;

for (int i = k; i < arr.length; i++) {
    windowSum += arr[i] - arr[i - k];
    maxSum = Math.max(maxSum, windowSum);
}
```

**Variable Window Size (Two-pointer):**
Useful when the window size depends on a condition (e.g., sum ≤ target).

## 14. Conclusion

Arrays in Java are powerful tools for storing and processing fixed-size data collections. Once you learn how to declare, use, loop, sort, and convert arrays, they become a strong foundation for learning more complex data structures and collections.