## **Representation of Arrays in memory**

## 

## Arrays in Java (and in most languages) are contiguous blocks of memory. This means:

## Each element is stored next to the other in memory.

## The base address (location of the first element) is used to compute the address of any element using this formula:

## address = base\_address + (index × size\_of\_element)

## **Advantages of Arrays**

1. **Fast Random Access**You can access any element in **O(1)** time because of the memory layout.
2. **Simplicity**Easy to declare and use. Suitable for small and fixed-size collections.
3. **Cache-Friendly**Because of contiguous memory, CPUs cache array elements better.

## **Time Complexity of Array Operations**

| Operation | Time Complexity | Explanation |
| --- | --- | --- |
| Access | O(1) | Direct jump using index |
| Search | O(n) | You have to scan each element until found (linear search) |
| Insert at End | O(1) (if space) | If space is available, just put it at the end |
| Insert at Index | O(n) | Need to shift all elements after the index |
| Delete at Index | O(n) | Need to shift elements to fill the gap |
| Traverse | O(n) | Visit each element once |

## **Limitations of Arrays**

## **Fixed Size** Once declared, the size **cannot be changed**. You need to re-create a new array to resize.

## **Costly Insertions/Deletions (Middle)** Adding/removing elements in the middle involves **shifting elements**, which is **slow (O(n))**.

## **Wasted Memory** If the array is declared larger than needed, unused memory is wasted.

## **When to Use Arrays**

## Use arrays when:

## The **size is known in advance** and doesn't change often.

## You need **fast access by index**.

## You want to store **simple, primitive data types** or objects in order.

## 

## 