Day 1 Notes – DSA Basics

1. Arrays – Fundamentals

What is an Array?

- An array is a **row of boxes** that stores multiple values of the **same data type**.
- Each box stores one value.

Size of Array

- Array size is **fixed** at the time of creation.
- Once created, the size **cannot change**.

Memory Storage

- Arrays are stored in **continuous (contiguous) memory**.
- This allows fast access using index.

Index

- Index is the **position number** of an element in the array.
- Index always starts from **0**.

Example:

```
Index:  0   1   2   3
Array: [10, 20, 25, 30]
```

Access:

```
arr[0] → 10
arr[2] → 25
```

Declaring and Initializing an Array

```
int[] arr = new int[5];
```

```
int[] arr = {10, 20, 25, 30, 40};
```

Updating Values

```
arr[2] = 35; // replaces value at index 2
```

Traversing an Array

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

Traversing an Array using Scanner

```java
import java.util.Scanner;

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

        int n = sc.nextInt(); // size of array
        int[] arr = new int[n];

        // input elements
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        // traverse and print elements
        for (int i = 0; i < n; i++) {
            System.out.println(arr[i]);
        }
    }
}
```

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

```
---

## 2. Input in Java

### Input Stream
- An **input stream** is a flow of data from a source to the program.
- `System.in` is the standard input stream (keyboard/console).

### Scanner
- `Scanner` is a Java class used to read input from input streams.
- It reads data **from a buffer**, not directly from the keyboard.

Import:
```java
import java.util.Scanner;
```
```

Create Scanner:

```
Scanner sc = new Scanner(System.in);
```

---

3. Buffer

- A **buffer** is temporary memory where input is stored after submission.
- Input is submitted **line-wise** and always ends with a newline (`\n`).

Example input typed:

```
10 20 30
```

Actual buffer content:

```
10 20 30\n
```

---

4. Tokens

- A **token** is the smallest unit of input separated by **whitespace**.
- Whitespace includes:
- space ( )
- tab (`\t`)
- newline (`\n`)

Example:

```
java is fun
```

Tokens: - `java` - `is` - `fun`

---

5. Scanner Methods

Token-based Methods (leave newline behind)

These read **one token** and stop at whitespace:

- `next()` → one word
- `nextInt()` → one integer
- `nextDouble()` → one decimal
- `nextLong()` → one long

Example:

```
int a = sc.nextInt();
int b = sc.nextInt();
```

Input:

```
10 20
```

Line-based Method

- `nextLine()` reads the **entire line**.
- It **consumes the newline (`\n`)**.

Example:

```
String line = sc.nextLine();
```

---

6. Newline Confusion Explained

Input:

```
java
```

Code:

```
String word = sc.next();
String line = sc.nextLine();
```

What happens: - `next()` reads `"java"` - Newline (`\n`) remains in buffer - `nextLine()` sees `\n` and returns empty string

---

7. Consuming Leftover Newline

Used to clear buffer before reading a line.

```java
int n = sc.nextInt();
sc.nextLine(); // consume leftover newline
String s = sc.nextLine();
```

---

8. Full Working Example

```java
import java.util.Scanner;

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

        int n = sc.nextInt();
        sc.nextLine(); // consume newline

        String name = sc.nextLine();

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

---

9. Scanner Creation – Why `new Scanner(System.in)`?

Why do we write `new Scanner(System.in)`?

`Scanner` needs to know **where to read input from**. `System.in` represents the **standard input stream**, usually the **keyboard / console**.

```java
Scanner sc = new Scanner(System.in);
```

Meaning: Create a Scanner that reads input from the keyboard.

Components Breakdown

1️⃣ `System.in`

- An **input stream**
- Connected to the keyboard
- Provides **raw bytes only**
- Does not understand `int`, `String`, etc.

Think of it as a **pipe** through which input flows.

2️⃣ `Scanner`

- A helper class
- Converts raw input into `int`, `double`, `String`
- Cannot read by itself → needs an input source

3️⃣ **Together**

```
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
```

Flow: Keyboard → System.in → Scanner → int → variable

Why not `new Scanner()`?

Java forces you to specify the **input source** explicitly: - keyboard - file - network

---

Scanner with File (example)

```
Scanner sc = new Scanner(new File("data.txt"));
```

Why Scanner is Slow (CP perspective)

- Reads character by character
- Uses regex internally
- Performs extra checks (locale, format)

For large input → may cause TLE

---

10. How Input Reaches Your Program (Flow)

1. User submits input as a line
2. OS stores it in STDIN buffer
3. Java accesses via `System.in`
4. Scanner reads from buffer
5. Values stored in variables

```
Keyboard → OS Buffer → System.in → Scanner → Variables
```

---

11. Interview-Ready One-Liners

- Arrays store elements in continuous memory and have fixed size
- Index represents the position of an element starting from 0
- Scanner reads input from a buffer, not directly from the keyboard
- `System.in` is the standard input stream
- Scanner converts raw input into data types
- Token-based methods stop at whitespace and leave newline behind
- `nextLine()` reads a full line and consumes the newline
- Buffer is temporary memory storing input before reading
- Token is a unit of input separated by whitespace

---

12. Mental Model (Remember This)

- Buffer → input waiting room
- Token → one word/number
- `next()` → one token
- `nextLine()` → full line

✅ **Day 1 Complete: Arrays basics + Scanner fundamentals**