Day 12 - 27th June 2025

**Task 1:**

What do you understand about data structures?

Data structure defines how data is organized, managed, and stored. These data elements (data structures) provide an efficient way of storing and organizing data in the computer making possible utilizing them more efficiently.

**Task 2:**

What are the types of data structures you know .. list them out..

linear and non-linear types. Linear data structures include arrays, linked lists, stacks, and queues. Non-linear data structures include trees and graphs. Hash tables are also a common data structure, often categorized separately.

**Task 3:**

What all operations can we do in Data structures?

Data structures support several fundamental operations, including traversing, searching, inserting, deleting, sorting, and merging. These operations allow for efficient storage, manipulation, and retrieval of data within a data structure.

**Task 4:**

What are static and dynamic arrays? Explain or summerize key points in a table

Size, performance, memory, flexibility, limitations

| **Aspect** | **Static Arrays** | **Dynamic Arrays** |
| --- | --- | --- |
| Size | Fixed size declared at compilation | Can grow or shrink during runtime |
| Memory Allocation | Allocated during compile time | Allocated during runtime (heap memory) |
| Memory Efficiency | More efficient in memory usage | May use extra memory for flexibility |
| Memory Location | Stack memory | Heap memory |
| Speed of Access | Faster access time | Slightly slower access time |
| Size Modification | Cannot be modified after creation | Can be modified through methods like push, pop, resize |
| Memory Management | Automatic (managed by system) | Manual in some languages, automatic in others |
| Risk of Overflow | Can have buffer overflow if bounds not checked | Usually handles overflow by automatic resizing |
| Implementation Complexity | Simpler to implement | More complex due to resizing mechanisms |
| Use Case | When size is known and fixed | When size needs to be flexible |
| Example Declaration | int arr[5]; (C++) | vector<int> arr; (C++) or ArrayList<Integer> (Java) |

**Task 5:**

What is the binary value of a?

Hint ascii value is 97..

The ASCII value for the lowercase letter 'a' is 97 in decimal.

To convert 97 to its 8-bit binary representation, we follow the successive division by 2 method:

1. 97/2=48 remainder 1

2. 48/2=24 remainder 0

3. 24/2=12 remainder 0

4. 12/2=6 remainder 0

5. 6/2=3 remainder 0

6. 3/2=1 remainder 1

7. 1/2=0 remainder 1

Reading the remainders from bottom to top gives us the 7-bit binary: 1100001.

Since we need an 8-bit representation, we pad a leading zero to the left:

01100001

**Task 6:**

Types of Computer memory with examples.. Explain .

Computer memory is used to store data and instructions either temporarily or permanently. It plays a crucial role in a computer's performance and is typically divided into primary and secondary memory.

🔹 1. Primary Memory (Main Memory)

Primary memory is directly accessible by the CPU. It is fast and volatile (loses data when power is off).

a. RAM (Random Access Memory)

Temporary memory used to store data and instructions that the CPU needs while working.

Volatile in nature.

Example: DDR4 RAM, LPDDR5 RAM in smartphones/laptops.

b. ROM (Read Only Memory)

Stores permanent instructions for booting up the computer (firmware).

Non-volatile.

Example: BIOS, EEPROM, Flash ROM.

c. Cache Memory

A small, high-speed memory located inside or close to the CPU.

Stores frequently accessed data and instructions.

Example: L1, L2, and L3 cache in processors.

d. Registers

Smallest and fastest type of memory, located inside the CPU.

Stores temporary data like instructions, memory addresses, etc.

Example: Program Counter (PC), Accumulator.

🔹 2. Secondary Memory (Storage Devices)

Secondary memory stores data permanently until deleted or overwritten. It is not directly accessed by the CPU; data must be loaded into RAM.

a. Hard Disk Drive (HDD)

Magnetic storage device used for storing large amounts of data.

Example: 1TB HDD used in desktop PCs.

b. Solid State Drive (SSD)

Uses flash memory, faster than HDD, more reliable.

Example: 512GB NVMe SSD in modern laptops.

c. Optical Discs

Uses laser technology to read/write data.

Example: CD, DVD, Blu-ray.

d. Flash Drives

Portable USB-based storage using flash memory.

Example: USB Pendrive, SD Card.

e. Magnetic Tape

Used for archival and backup.

Example: LTO (Linear Tape-Open) tapes used in enterprises.

🔹 3. Tertiary and Off-line Storage

Used for data archiving and backup; not immediately available to the CPU.

Example: Cloud Storage, External Hard Drives.

**Task 7:**

Reverse an array. write a code.

import java.util.Arrays;

public class ReverseArray {

public static void main(String[] args) {

int[] arr = {10, 20, 30, 40, 50};

System.out.println("Original array: " + Arrays.toString(arr));

// Reverse the array in-place

int start = 0;

int end = arr.length - 1;

while (start < end) {

// Swap elements

int temp = arr[start];

arr[start] = arr[end];

arr[end] = temp;

start++;

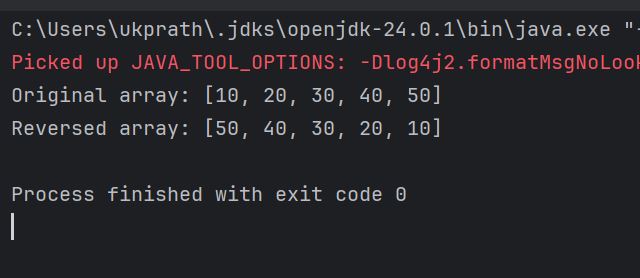
end--;

}

System.out.println("Reversed array: " + Arrays.toString(arr));

}

}



**Task 8:**

Reverse a string .. write a code.

import java.util.Scanner;

public class ReverseString {

public static void main(String[] args) {

// Create Scanner object to take user input

Scanner scanner = new Scanner(System.in);

// Prompt the user

System.out.print("Enter a string: ");

String input = scanner.nextLine();

// Reverse the string using a loop

String reversed = "";

for (int i = input.length() - 1; i >= 0; i--) {

reversed += input.charAt(i); // Append characters in reverse order

}

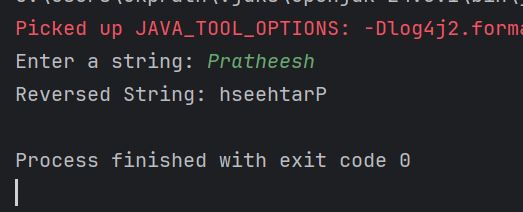
// Display the reversed string

System.out.println("Reversed String: " + reversed);

scanner.close(); // Close the scanner

}

}



**Task 9 : leetcode or anyother account opening**

**Task 10: understand the code and write**

public class Example {  
 public static void main (String[] args) {  
 int[] arr1 = {11, 34, 66, 75};  
 int n1 = arr1.length;  
 int[] arr2 = {1, 5, 19, 50, 89, 100};  
 int n2 = arr2.length;  
 int[] merge = new int[n1 + n2];  
 int i = 0, j = 0, k = 0, x;  
 System.out.print("Array 1: ");  
 for (x = 0; x < n1; x++)  
 System.out.print(arr1[x] + " ");  
 System.out.print("\nArray 2: ");  
 for (x = 0; x < n2; x++)  
 System.out.print(arr2[x] + " ");  
 while (i < n1 && j < n2) {  
 if (arr1[i] < arr2[j])  
 merge[k++] = arr1[i++];  
 else  
 merge[k++] = arr2[j++];  
 }  
 while (i < n1)  
 merge[k++] = arr1[i++];  
 while (j < n2)  
 merge[k++] = arr2[j++];  
 System.out.print("\nArray after merging: ");  
 for (x = 0; x < n1 + n2; x++)  
 System.out.print(merge[x] + " ");  
 }  
 }

* arr1 and arr2 are two **sorted arrays**.
* n1 and n2 are their lengths.
* merge[] is a new array with size n1 + n2 to hold the **merged result**.This is the **core merging loop**.like [0,0,0,0,0,0,0,0,0,0]
* It compares elements of arr1 and arr2 and inserts the **smaller** into the merge[] array.
* Uses three pointers:
  + i for arr1
  + j for arr2
  + k for merge

After one array finishes, the remaining elements of the other array are **copied as-is** into the merged array.

Prints the fully **merged and sorted** array.

**hometask**

**Task 10 rewrite the code in such a way that it has to take unsorted list and then ,merge in an array the sorted list.**

**import java.util.Arrays;**

**public class hometask {**

**public static void main (String[] args) {**

**int[] arr1 = {14, 11, 34, 66, 75};**

**Arrays.sort(arr1);**

**int n1 = arr1.length;**

**int[] arr2 = {4, 1, 5, 19, 50, 89, 100};**

**Arrays.sort(arr2);**

**int n2 = arr2.length;**

**int[] merge = new int[n1 + n2];**

**int i = 0, j = 0, k = 0, x;**

**System.out.print("Array 1: ");**

**for (x = 0; x < n1; x++)**

**System.out.print(arr1[x] + " ");**

**System.out.print("\nArray 2: ");**

**for (x = 0; x < n2; x++)**

**System.out.print(arr2[x] + " ");**

**while (i < n1 && j < n2) {**

**if (arr1[i] < arr2[j])**

**merge[k++] = arr1[i++];**

**else**

**merge[k++] = arr2[j++];**

**}**

**while (i < n1)**

**merge[k++] = arr1[i++];**

**while (j < n2)**

**merge[k++] = arr2[j++];**

**System.out.print("\nArray after merging: ");**

**for (x = 0; x < n1 + n2; x++)**

**System.out.print(merge[x] + " ");**

**}**

**}**

**Task 11:**

What do you understand by Hash table?

A hash table is a type of data structure in which information is stored in an easy-to-retrieve and efficient manner. In the key-value method, keys are assigned random indexes where their values are stored in an array. The index is the information of where exactly in the array the value is stored.

key could be your SID, your telephone number,

social security number, account number, ...

Must have unique keys

Each key is associated with–mapped to–a value

Convert a String key into an integer that will be in the range of 0 through the maximum capacity-1

**Task 12**

Understand the below Hash table code and try to print values using get method of Hash table

public class task012HashTable {

public static void main(String[] args) {

Hashtable<String, Integer> ht = new Hashtable<>();

ht.put("Pratheesh", 101);

ht.put("bala", 102);

ht.put("allwin", 103);

// use get method of Ht

for (Map.Entry<String, Integer> e : ht.entrySet())

System.out.println(e.getKey() + " " + e.getValue());

}

}

**Task13**

import java.util.HashMap;

import java.util.Map;

public class task013HashMap {

public static void main(String[] args) {

// Create a HashMap

HashMap<String, Integer> hm = new HashMap<>();

// Add key-value pairs

hm.put("Pratheesh", 101);

hm.put("bala", 102);

hm.put("allwin", 103);

// // Access values using get()

// System.out.println("Using get() method:");

// System.out.println("Pratheesh ID: " + hm.get("Pratheesh"));

// System.out.println("bala ID: " + hm.get("bala"));

// System.out.println("allwin ID: " + hm.get("allwin"));

// Or use entrySet() to loop over all key-value pairs

System.out.println("\nUsing entrySet():");

for (Map.Entry<String, Integer> e : hm.entrySet()) {

System.out.println(e.getKey() + " : " + e.getValue());

}

}

}

**Similarities**

hash table and Hash Map have linked list internally.

Collisions occur in Hash Table and hash Maps.

Collision in Hash map can handle separate chaining, Open addressing etc..

Task 14:

Difference between Hash Table and Hash Map

| **Feature** | [**HashMap**](https://chatgpt.com/w) | [**Hashtable**](https://chatgpt.com/w) |
| --- | --- | --- |
| **Thread Safety** | ❌ Not thread-safe (non-synchronized) | ✅ Thread-safe (synchronized methods) |
| **Performance** | ✅ Faster (no synchronization overhead) | ❌ Slower (due to synchronization) |
| **Allows null Keys/Values** | ✅ 1 null key, multiple null values | ❌ Does not allow null keys/values |
| **Synchronization** | Manual (if needed, use Collections.synchronizedMap()) | Built-in synchronization |
| **Introduced In** | [Java 1.2 (part of Collections Framework)](https://chatgpt.com/w) | Java 1.0 (legacy class) |
| **Iterator Type** | [Fail-fast — throws ConcurrentModificationException](https://chatgpt.com/w) | [Fail-safe — no such exception](https://chatgpt.com/w) |
| **Use in Multithreading** | [Must use external synchronization or ConcurrentHashMap](https://chatgpt.com/w) | Safe for multi-threaded use |
| **Package** | java.util | java.util |
| Implementation Complexity | Simpler to implement | More complex due to resizing mechanisms |
| Use Case | When size is known and fixed | When size needs to be flexible |
| Example Declaration | int arr[5]; (C++) | vector<int> arr; (C++) or ArrayList<Integer> (Java) |

## **HashMap**

### **✅ Advantages:**

1. **Faster performance – Not synchronized, so it performs better in single-threaded environments.**
2. **Allows one null key and multiple null values – Offers more flexibility.**
3. **Modern and part of the Java Collections Framework – Introduced in Java 1.2.**
4. **More versatile – Can be converted into a synchronized map using Collections.synchronizedMap() if needed.**

### **❌ Disadvantages:**

1. **Not thread-safe – Needs manual synchronization in multi-threaded environments.**
2. **Fail-fast iterator – Throws ConcurrentModificationException if modified while iterating.**
3. **Requires extra care in concurrent scenarios – Better alternatives like ConcurrentHashMap are preferred for thread safety.**

## **🟢 Hashtable**

### **✅ Advantages:**

1. **Thread-safe by default – All methods are synchronized, making it safe in multi-threaded environments.**
2. **Legacy-compatible – Available since Java 1.0; still used in some older codebases.**

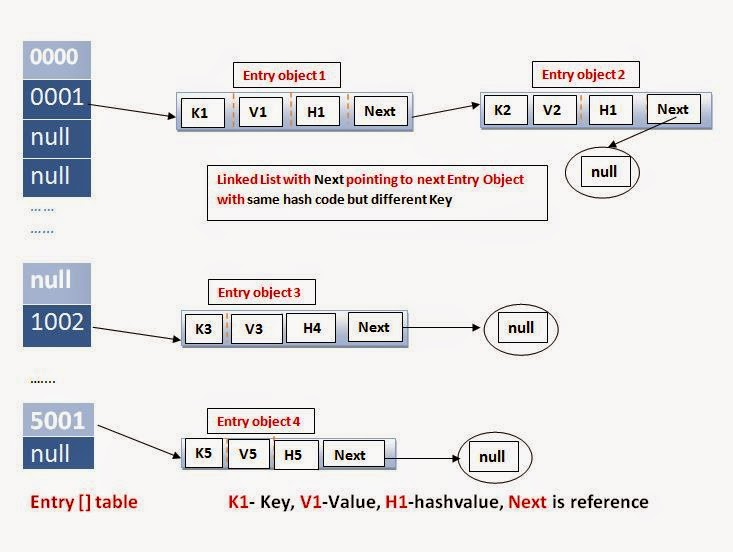
### **❌ Disadvantages:**

1. **Slower performance – Due to method-level synchronization, it is less efficient in single-threaded applications.**
2. **Does not allow null keys or values – Any attempt results in NullPointerException.**
3. **Outdated – Considered a legacy class; modern alternatives like ConcurrentHashMap or synchronized HashMap are preferred.**
4. **Less flexible – Compared to the more modern and customizable HashMap.**

Hash table methods List .. for your ref..

* put(K key, V value): Inserts a key-value mapping into the Hashtable. If the key already exists, the old value is replaced with the new one.
* get(Object key): Returns the value associated with the specified key. Returns null if the key is not found.
* remove(Object key): Removes the key-value mapping for the specified key from the Hashtable.
* containsKey(Object key): Returns true if the Hashtable contains a mapping for the specified key, otherwise returns false.
* containsValue(Object value): Returns true if the Hashtable maps one or more keys to the specified value, otherwise returns false.
* isEmpty(): Returns true if the Hashtable contains no key-value mappings, otherwise returns false.
* size(): Returns the number of key-value mappings in the Hashtable.
* clear(): Removes all key-value mappings from the Hashtable.
* keySet(): Returns a Set view of the keys contained in the Hashtable.
* values(): Returns a Collection view of the values contained in the Hashtable.
* entrySet(): Returns a Set view of the key-value mappings contained in the Hashtable.
* rehash(): Increases the size of the Hashtable and rehashes all of its keys. This method is protected and typically handled internally by the Hashtable for performance optimization.
* clone(): Returns a shallow copy of the Hashtable instance.

Hash table internal structure

****

**Task 16**

import java.util.HashMap;

public class task016 {

public static void main(String[] args) {

HashMap<String, String> map = new HashMap<>();

// Add entries

map.put("A", "Apple");

map.put("B", "Banana");

// Add null key

map.put(null, "NullKeyValue");

// Add another value for null key (overwrites)

map.put(null, "UpdatedNullKey");

// Add null values

map.put("C", null);

map.put("C", "grape");

// Print map

for (String key : map.keySet()) {

System.out.println("Key: " + key + " → Value: " + map.get(key));

}

}

}

Task 17 make a Hashmap synchronized..

import java.util.Collections;

import java.util.HashMap;

import java.util.Map;

public class task017 {

public static void main(String[] args) {

// Create a normal HashMap

Map<String, String> map = new HashMap<>();

// Make it synchronized

Map<String, String> syncMap = Collections.synchronizedMap(map);

// Use synchronized map

syncMap.put("A", "Apple");

syncMap.put("B", "Banana");

syncMap.put(null, "Banana");

syncMap.put(null, "NullKey"); // Still allows null key

// Access within synchronized block if doing iteration in multi-threaded context

synchronized (syncMap) {

for (Map.Entry<String, String> entry : syncMap.entrySet()) {

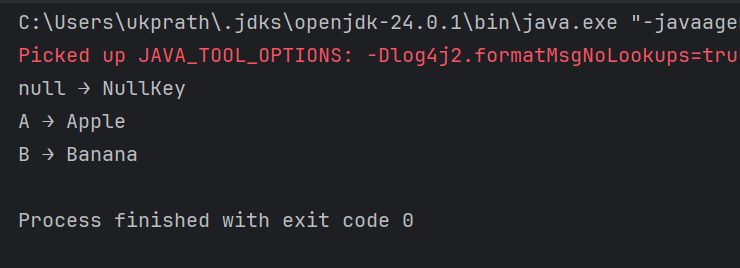
System.out.println(entry.getKey() + " → " + entry.getValue());

}

}

}

}



Different methods to create a hashmap in java :

1) Constructing a hashmap with default capacity

ex:

HashMap<String, Integer> hm1 = new HashMap<String, Integer>();

2) Constructing a hashmap with a capacity 10

ex:

HashMap<String, Integer> hm2 = new HashMap<String, Integer>(10);

3)copy one map to another map

ex:

HashMap<String, Integer> hm3 = new HashMap<String, Integer>( hm2);

4)

Specifying load factor along with the capacity

ex:

HashMap<String, Integer> hm4= new HashMap<String, Integer>(10, 0.75f);

Initial capacity ===10

Load factor === 0.75f