Day 12 - 27th June 2025

Task 1:

What do you understand about data structures?

Data structure refer to the way data is organized, managed, and stored for efficient access and modification. A data structure is a specialized format for organizing and storing data in a computer so that it can be used efficiently.

Task 2:

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

1. Linear Data Structures: Array, Linked List, stack, queue
2. Non linear data structure: Tree, graph

Task 3:

What all operations can we do in Data structures?

1. Insertion
2. Deletion
3. Traversal
4. Searching
5. Sorting
6. Access
7. Update
8. Merge
9. Splitting

Task 4:

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

Size, performance, memory, flexibility, limitations

| **Feature** | **Static Array** | **Dynamic Array** |
| --- | --- | --- |
| **Size** | Fixed at the time of declaration | Grows or shrinks during runtime |
| **Performance** | Faster access (direct memory allocation) | Slight overhead during resizing operations |
| **Memory** | Allocated at compile-time (contiguous block) | Allocated at runtime (resized as needed) |
| **Flexibility** | Less flexible (size cannot change) | More flexible (can accommodate more/less data) |
| **Limitations** | Wastes memory if underutilized; risk of overflow | Resizing can be costly (copying data to new memory) |
| **Resizing** | Not possible without creating a new array | Automatically resizes (e.g., doubling size) |
| **Example Languages** | C (with int arr[100]) | Python lists, Java ArrayList, C++ vector |

Task 5:

What is the binary value of a?

01100001

Task 6:

Types of Computer memory with examples.. Explain ..

**1. Primary Memory (Main Memory)**

* Directly accessible by the CPU.
* Fast and volatile (except ROM).
* Stores data that is currently in use.

**2. Secondary Memory (Storage Memory)**

* Not directly accessed by the CPU.
* Used for long-term data storage.
* Non-volatile.

Task7:

Code to reverse a array

public class ReverseArray {  
 public static void main(String[] args) {  
 int[] arr = {10, 20, 30, 40, 50};  
  
 System.*out*.println("Original Array:");  
 *printArray*(arr);  
  
 *reverseArray*(arr);  
  
 System.*out*.println("Reversed Array:");  
 *printArray*(arr);  
 }  
  
  
 public static void reverseArray(int[] arr) {  
 int start = 0;  
 int end = arr.length - 1;  
  
 while (start < end) {  
  
 int temp = arr[start];  
 arr[start] = arr[end];  
 arr[end] = temp;  
  
  
 start++;  
 end--;  
 }  
 }  
  
 public static void printArray(int[] arr) {  
 for (int num : arr) {  
 System.*out*.print(num + " ");  
 }  
 System.*out*.println();  
 }  
}

output:

Original Array:

10 20 30 40 50

Reversed Array:

50 40 30 20 10

Process finished with exit code 0

Task 8:

Java program to reverse a string taking input from user

Code:

import java.util.Scanner;  
  
public class reverseString {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*); // Create Scanner object  
  
  
 System.*out*.print("Enter a string to reverse: ");  
 String input = scanner.nextLine();  
  
  
 String reversed = *reverseString*(input);  
  
  
 System.*out*.println("Reversed String: " + reversed);  
 }  
  
 public static String reverseString(String str) {  
 StringBuilder reversed = new StringBuilder(str);  
 return reversed.reverse().toString(); // Using StringBuilder's reverse method  
 }  
}

output:

Enter a string to reverse: jaysree hariharan

Reversed String: narahirah eersyaj

Process finished with exit code 0

Task 10:

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] + " ");  
   }  
}

Explanation:

* It merges two sorted arrays (arr1 and arr2) into a third array merge, maintaining the sorted order.
* Two-pointer technique is used (i for arr1, j for arr2).
* Compare elements at arr1[i] and arr2[j], and insert the smaller one into merge[k].
* After one array is fully traversed, copy the remaining elements from the other array.

Task 11:

What do you understand by Hash table?

A hash table is a data structure that stores key-value pairs and allows for fast insertion, deletion, and lookup operations. It helps in translation of string into an integer.

Task 12

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

import java.util.Hashtable;

public class Task012\_DS\_HashTable {

public static void main(String[] args) {

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

ht.put("Anitha", 101);

ht.put("Kavitha", 102);

ht.put("Meera", 103);

// Printing values using get() method

System.out.println("Printing values using get() method:");

System.out.println("Anitha: " + ht.get("Anitha"));

System.out.println("Kavitha: " + ht.get("Kavitha"));

System.out.println("Meera: " + ht.get("Meera"));

}

}

Output:

Kavitha 102

Anitha 101

Meera 103

Process finished with exit code 0

Task 13:

Wap to create  a hash map and display them..

import java.util.HashMap;  
  
public class Task012\_DS\_Hashmap {  
 public static void main(String[] args) {  
 HashMap<String, Integer> ht = new HashMap<>();  
  
 ht.put("Anitha", 101);  
 ht.put("Kavitha", 102);  
 ht.put("Meera", 103);  
  
 // Printing values using get() method  
 System.*out*.println("Printing values using get() method:");  
 System.*out*.println("Anitha: " + ht.get("Anitha"));  
 System.*out*.println("Kavitha: " + ht.get("Kavitha"));  
 System.*out*.println("Meera: " + ht.get("Meera"));  
 }  
}

output:

Printing values using get() method:

Anitha: 101

Kavitha: 102

Meera: 103

Process finished with exit code 0

Task 14:

Difference between Hash Table and Hash Map

| **Feature** | **HashTable** | **HashMap** |
| --- | --- | --- |
| **Thread Safety** | Thread-safe (synchronized) | Not thread-safe by default |
| **Performance** | Slower due to synchronization | Faster (better in single-threaded apps) |
| **Null Keys / Values** | Does not allow null key or value | Allows one null key, many null values |
| **Introduced In** | JDK 1.0 | JDK 1.2 (part of the Collections Framework) |
| **Part of Collections?** | No | Yes |
| **Iterator Type** | Uses Enumerator (legacy) | Uses Iterator (modern, fail-fast) |
| **Use Case** | Good for multi-threaded legacy apps | Preferred in modern code (use ConcurrentHashMap if thread-safe) |

Try to add 1 null value in the key and run the hash map code..

Also add one more null value to the key and see the result..

make a Hashmap synchronized..

Plz note : Hash Maps are - asynchronous in nature..

Hint   Map<String, Integer> syncMap = Collections.synchronizedMap(Hm1);

Code:

import java.util.\*;  
  
public class HashMapNullAndSyncExample {  
 public static void main(String[] args) {  
 // Step 1: Create a regular HashMap  
 HashMap<String, Integer> hm1 = new HashMap<>();  
  
 // Step 2: Add a null key  
 hm1.put(null, 100);  
 System.*out*.println("After adding 1st null key: " + hm1);  
  
 // Step 3: Add another null key (this will overwrite the previous one)  
 hm1.put(null, 200); // Replaces value for null key  
 System.*out*.println("After adding 2nd null key: " + hm1);  
  
 // Step 4: Make the HashMap synchronized  
 Map<String, Integer> syncMap = Collections.*synchronizedMap*(hm1);  
  
 // Step 5: Access synchronized map (safe in multi-threaded environments)  
 System.*out*.println("Synchronized HashMap: " + syncMap);  
  
 // Bonus: Add a null value  
 syncMap.put("A", null);  
 syncMap.put("B", null);  
 System.*out*.println("With null values added: " + syncMap);  
 }  
}

output:

After adding 1st null key: {null=100}

After adding 2nd null key: {null=200}

Synchronized HashMap: {null=200}

With null values added: {null=200, A=null, B=null}

Process finished with exit code 0

What is load factor and how the capacity increases?

The load factor is a measure of how full a hash table is allowed to get before it automatically resizes (increases its capacity).

Pointers.. And load balancing… plz check what are they why are they used and where are they used?

A **pointer** is a variable that **stores the memory address** of another variable.

**Load balancing** is the process of **distributing workloads evenly** across multiple computing resources (e.g., servers, CPUs, network links).