Day 12

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.

ASCII, ANSI ,  Binary code conversions, 8 bit representation , 16 bit representation..

A ===>             how the character A is stored?

A ====> ASCII = American Standard code for information interchange    value 65 ====> binary code

ASCII values are given ===  by ASNI === American Standard National Institute

Special  characters before 65

A  ===> 65 …. 66 = B…..

Small characters …  97 = a. 98 = b

A

65 ===> binary code?

65 / 2 == 1

32 / 2 == 0

16 / 2 == 0

8   / 2 == 0

4   / 2 == 0

2   / 2 == 0

1  /     == 1       = ⇒ 1 0 0 0 0 0 1

Bottom to top take the remainders

For A : 65

0 1 0 0 0 0 0 1 => 8 bit rep

0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 => 16 bit rep

Task 4:

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

Size, performance, memory, flexibility, limitations

comparison of Static and Dynamic Arrays in a table format:

| **Aspect** | **Static Arrays** | **Dynamic Arrays** |
| --- | --- | --- |
| **Size** | Fixed size at declaration | Can grow/shrink as needed |
| **Performance** | - Faster access (O(1)) <br> - No resize overhead | - Same access speed (O(1)) <br> - Additional overhead during resizing |
| **Memory** | - Fixed memory allocation <br> - No extra space | - Uses extra space for growth <br> - May have unused space |
| **Flexibility** | - Cannot add/remove elements <br> - Size cannot change | - Can add/remove elements <br> - Automatically resizes |
| **Limitations** | - May waste memory if underused <br> - Array overflow if full | - Resizing is costly <br> - May use more memory than needed |

Key Points:

1. Static Arrays:
   * Good for fixed-size data
   * Better memory efficiency when size is known
   * Used in embedded systems
2. Dynamic Arrays:
   * Good for variable-size data
   * More flexible for growing collections
   * Common in modern programming (like ArrayList in Java, Vector in C++)

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 ..?

Let's break down the types of computer memory with examples:

1. Primary Memory (Volatile)

a) RAM (Random Access Memory)

- Example: 16GB DDR4 SDRAM

- Used for temporary data and running programs

- Loses data when power is off

- Fast access times

b) Cache Memory

- Example: L1 (64KB), L2 (256KB), L3 (8MB) on an Intel Core i7 processor

- Very fast, small amount of memory close to the CPU

- Stores frequently accessed data

2. Secondary Memory (Non-volatile)

a) Hard Disk Drives (HDD)

- Example: Seagate BarraCuda 2TB HDD

- Magnetic storage, slower but high capacity

- Stores OS, programs, and user data

b) Solid State Drives (SSD)

- Example: Samsung 970 EVO Plus 1TB NVMe SSD

- Faster than HDDs, no moving parts

- Used for faster boot and load times

c) Optical Drives

- Examples: DVD-RW, Blu-ray

- Used for data distribution and backups

d) USB Flash Drives

- Example: SanDisk Ultra 64GB USB 3.0

- Portable storage, plug-and-play

3. ROM (Read-Only Memory)

- Example: BIOS/UEFI chip on motherboard

- Contains essential startup instructions

- Non-volatile, retains data when power is off

4. Virtual Memory

- Uses part of the HDD/SSD as an extension of RAM

- Example: Windows page file (pagefile.sys)

5. Graphics Memory (VRAM)

- Example: 8GB GDDR6 on NVIDIA GeForce RTX 3070

- Specialized memory for storing image data

6. Registers

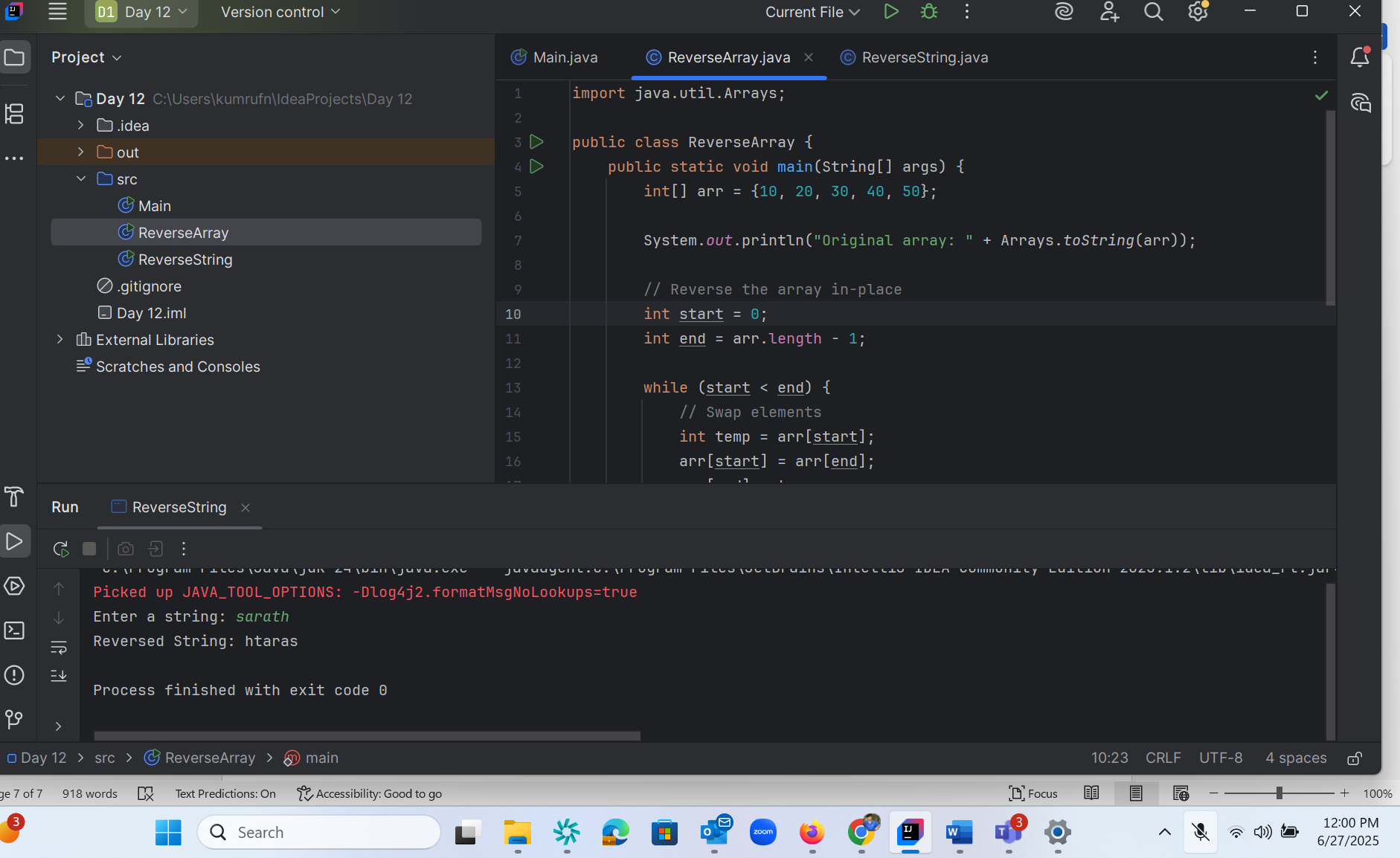
- Extremely fast memory within the CPU

- Example: Accumulator, Instruction Register

Each type serves a specific purpose, balancing speed, capacity, and cost. The combination forms the memory hierarchy in modern computers, from fastest/smallest (registers) to slowest/largest (secondary storage).

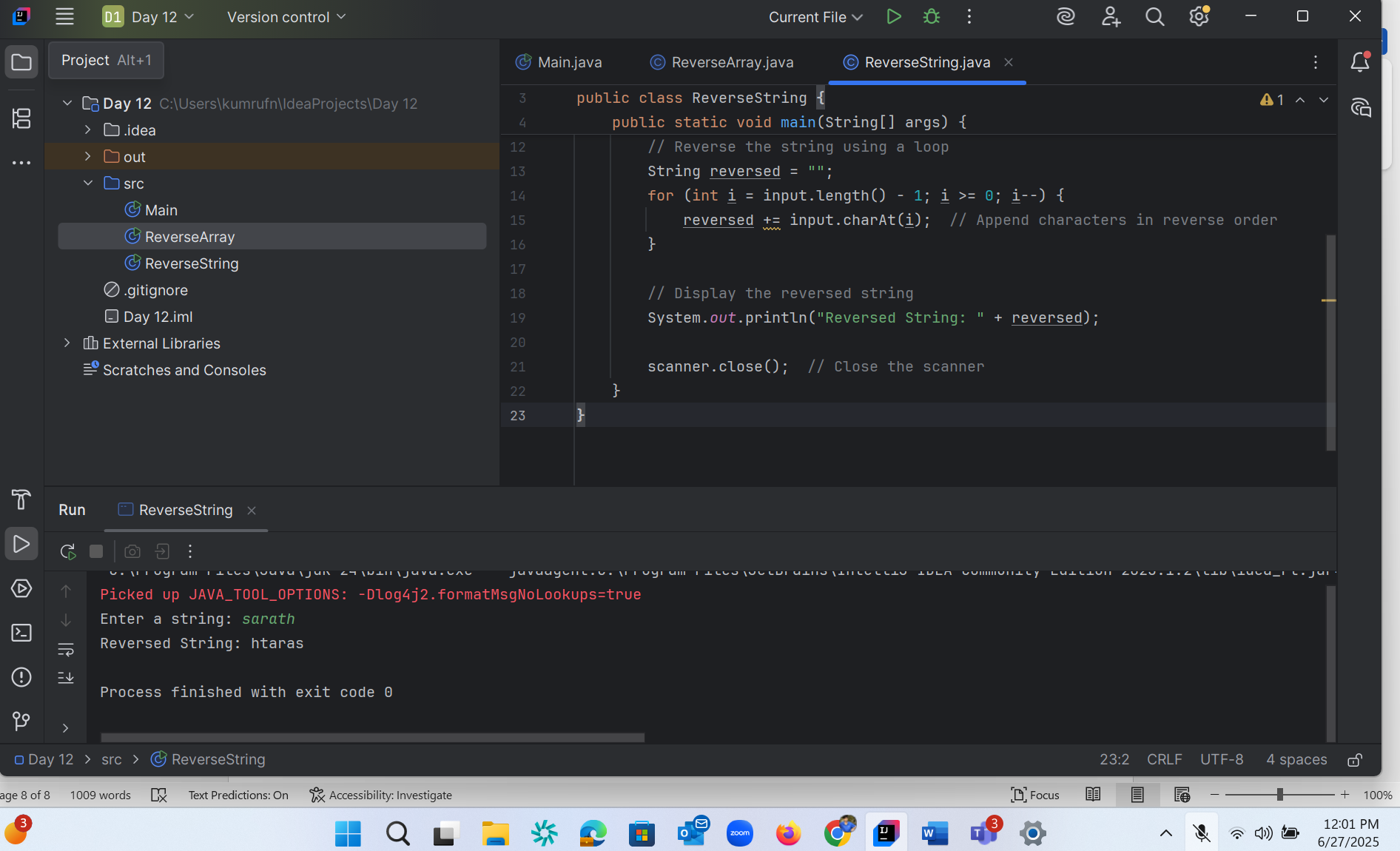
Task 7 :

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 :

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:

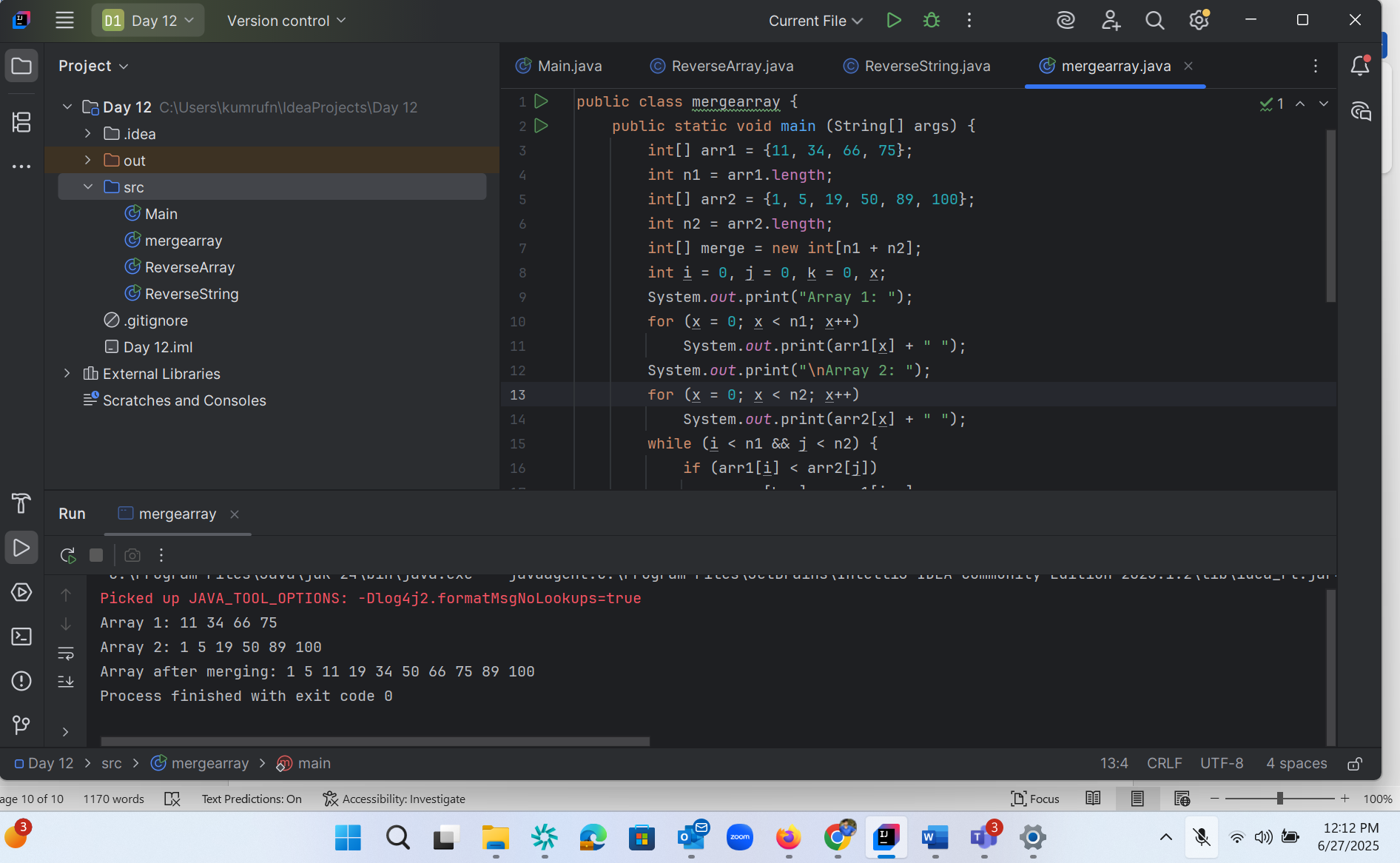
Login to Leetcode or Hacker rank :

Created LEET code : mail id – [Sarathkumar647@gmail.com](mailto:Sarathkumar647@gmail.com)

Password : Regular one

Task 10 :

public class mergearray {  
 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] + " ");  
 }  
}



Home task :

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.

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.

A HashTable is a data structure that:

1. Stores key-value pairs
2. Uses a hash function to compute an index from the key
3. Allows fast insertion and retrieval of data (usually O(1) time complexity)

Think of it like a smart dictionary:

* You give it a word (key)
* It quickly finds the definition (value)

Key benefits:

* Fast lookups
* Efficient for large datasets
* Flexible (can use various types as keys)

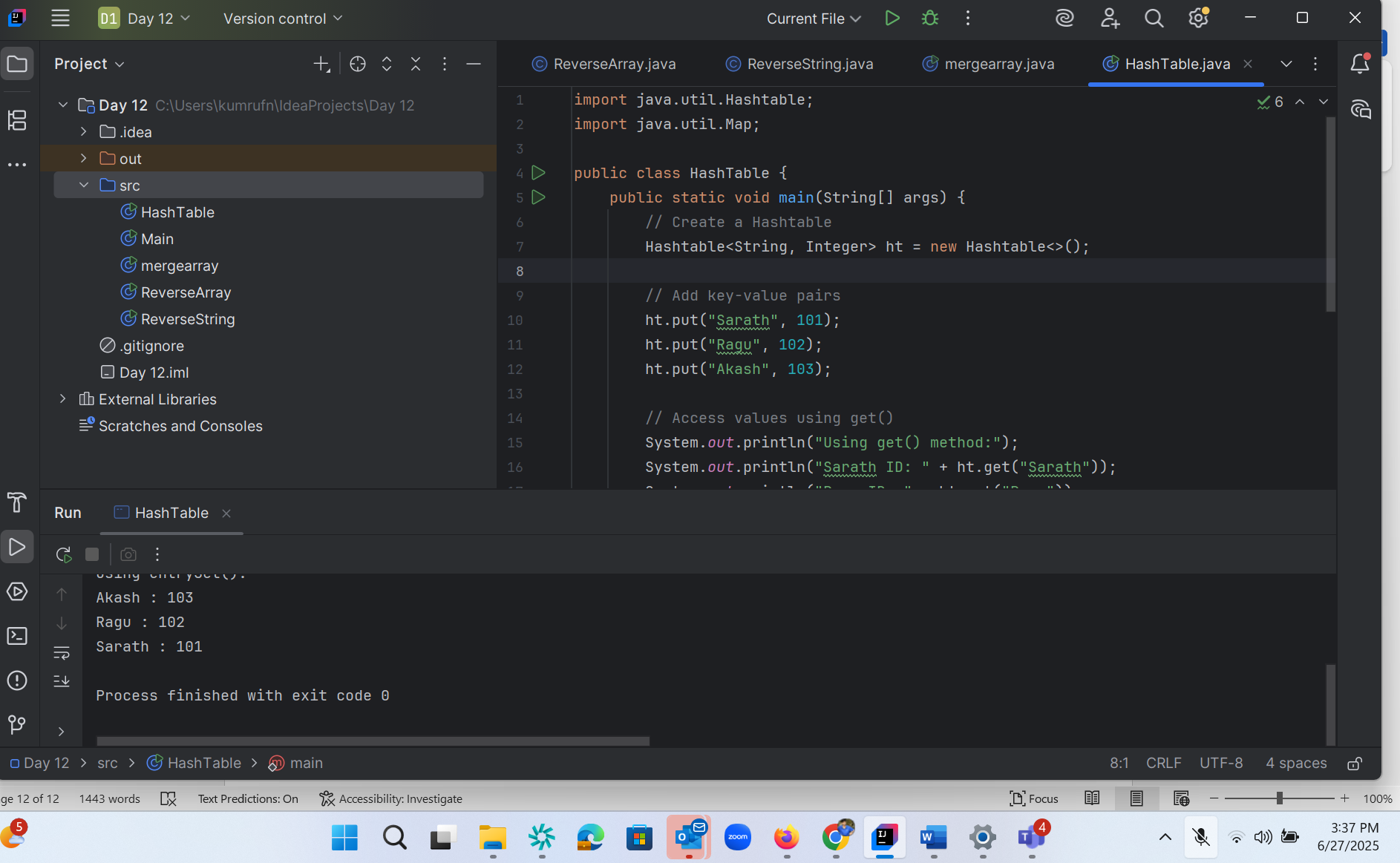
Common uses:

* Caching
* Database indexing
* Implementing associative arrays

HashTables are fundamental in computer science and widely used in real-world applications for their speed and efficiency.

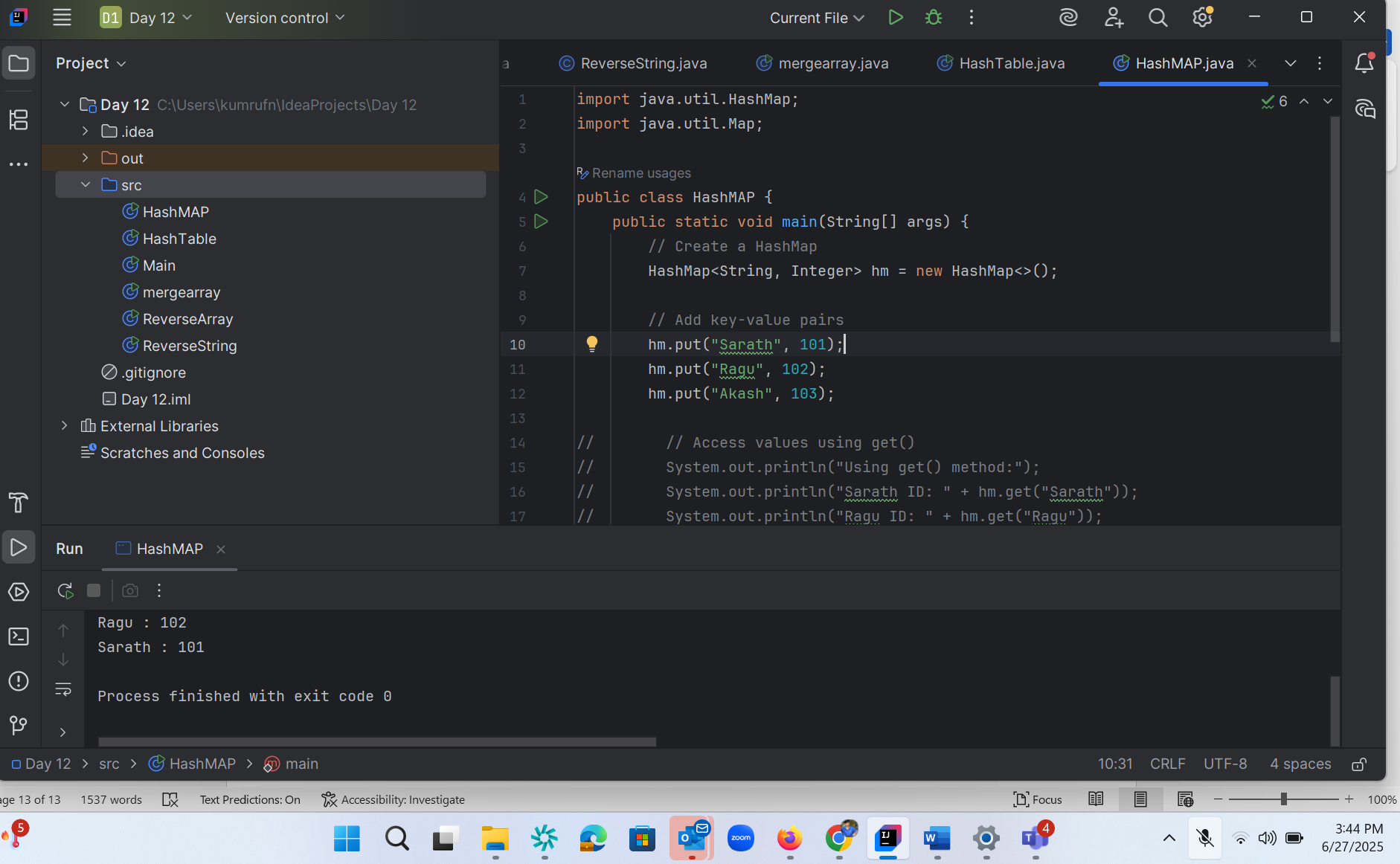
Task 12 :

import java.util.Hashtable;  
import java.util.Map;  
  
public class HashTable {  
 public static void main(String[] args) {  
 // Create a Hashtable  
 Hashtable<String, Integer> ht = new Hashtable<>();  
  
 // Add key-value pairs  
 ht.put("Sarath", 101);  
 ht.put("Ragu", 102);  
 ht.put("Akash", 103);  
  
 // Access values using get()  
 System.*out*.println("Using get() method:");  
 System.*out*.println("Sarath ID: " + ht.get("Sarath"));  
 System.*out*.println("Ragu ID: " + ht.get("Ragu"));  
 System.*out*.println("Akash ID: " + ht.get("Akash"));  
  
 // Or use entrySet() to loop over all key-value pairs  
 System.*out*.println("\nUsing entrySet():");  
 for (Map.Entry<String, Integer> e : ht.entrySet()) {  
 System.*out*.println(e.getKey() + " : " + e.getValue());  
 }  
 }  
}



Task 13 :

import java.util.HashMap;  
import java.util.Map;  
  
public class HashMAP {  
 public static void main(String[] args) {  
 // Create a HashMap  
 HashMap<String, Integer> hm = new HashMap<>();  
  
 // Add key-value pairs  
 hm.put("Sarath", 101);  
 hm.put("Ragu", 102);  
 hm.put("Akash", 103);  
  
// // Access values using get()  
// System.out.println("Using get() method:");  
// System.out.println("Sarath ID: " + hm.get("Sarath"));  
// System.out.println("Ragu ID: " + hm.get("Ragu"));  
// System.out.println("Akash ID: " + hm.get("Akash"));  
  
 // 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());  
 }  
 }  
}



Task 14:

Difference between Hash Table and Hash Map?

HashTable:

1. Synchronized (thread-safe)
2. Doesn't allow null keys or values
3. Slower due to synchronization
4. Legacy class (since Java 1.0)
5. All methods are synchronized

HashMap:

1. Not synchronized (not thread-safe by default)
2. Allows one null key and multiple null values
3. Generally faster
4. Introduced later (Java 1.2)
5. Methods are not synchronized

When to use:

* Use HashMap for single-threaded applications or when you need null keys/values
* Use HashTable (or ConcurrentHashMap) for thread-safe operations in multi-threaded environments

In modern Java applications, HashMap is generally preferred unless thread-safety is specifically required. For thread-safe operations, ConcurrentHashMap is often a better choice than HashTable due to its better performance in concurrent scenarios.

**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.**

Task 15 :

Task 15:

Linear probing in Hash table

public class HashTable<Key, Value> {

private class HashTableNode {

private Key key;

private Value value;

private boolean active;

private boolean tombstoned; // Allow reuse of removed slots

public HashTableNode() {

// All nodes in array will begin initialized this way

key = null;

value = null;

active = false;

tombstoned = false;

}

public HashTableNode(Key initKey, Value initData) {

key = initKey;

value = initData;

active = true;

tombstoned = false;

}

}

private final static int TABLE\_SIZE = 9;

private Object[] table;

public HashTable() {

// Since HashNodeTable has generics, we can not have

// a new HashNodeTable[], so use Object[]

table = new Object[TABLE\_SIZE];

for (int j = 0; j < TABLE\_SIZE; j++)

table[j] = new HashTableNode();

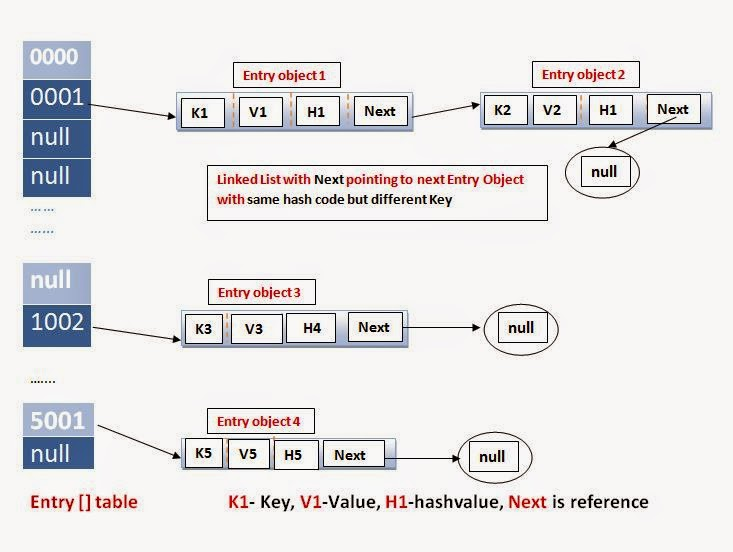
}

public Value put(Key key, Value value)

 Hash table methods List .. for your ref..

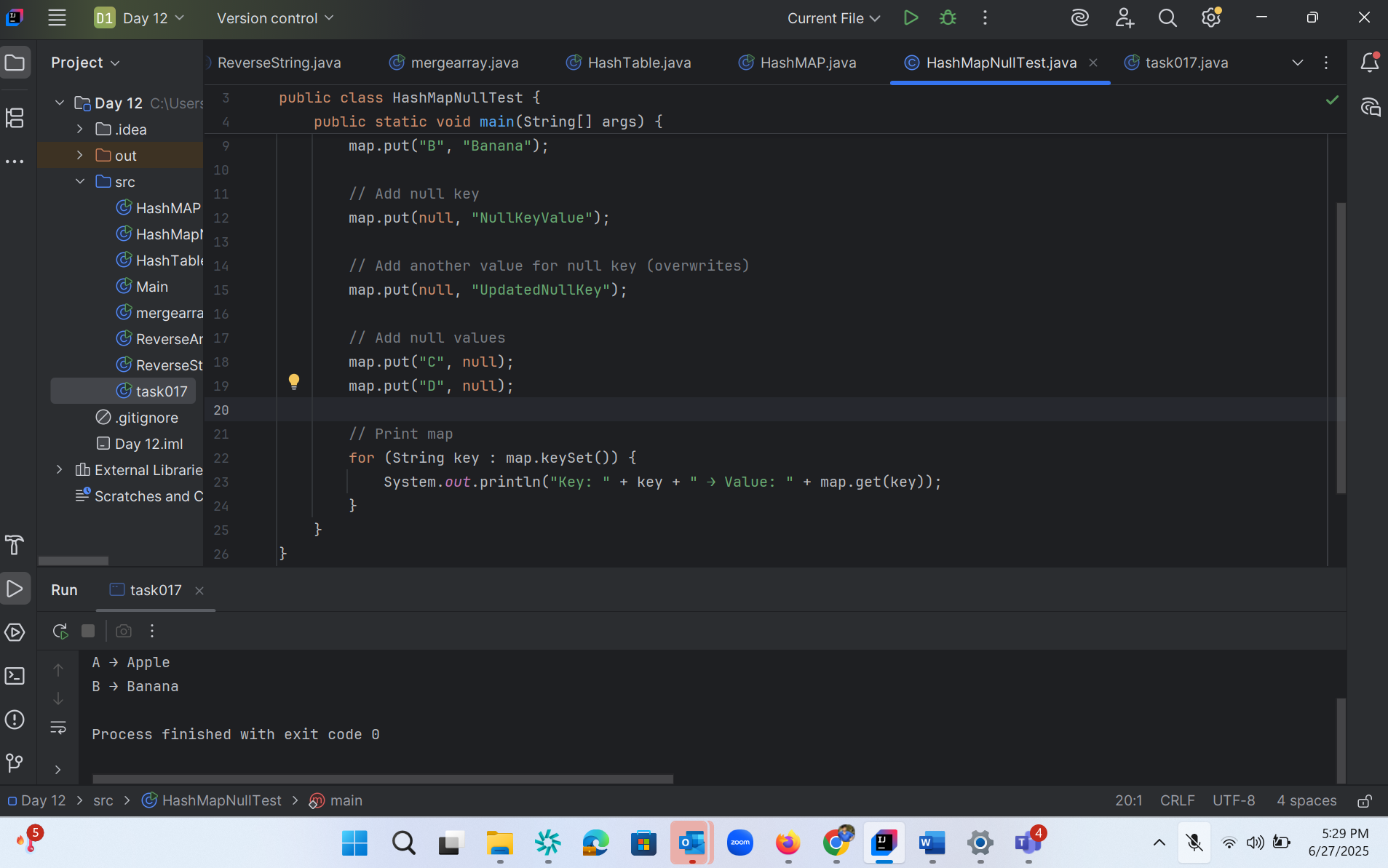
* put(K key, V value): Inserts a key-value mapping into the Hashtable. If the key already exists, the old value is replaced with the new one.
* get(Object key): Returns the value associated with the specified key. Returns null if the key is not found.
* remove(Object key): Removes the key-value mapping 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 HashMapNullTest {  
 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("D", null);  
  
 // Print map  
 for (String key : map.keySet()) {  
 System.*out*.println("Key: " + key + " → Value: " + map.get(key));  
 }  
 }  
}



Task 16.1

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, "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());  
 }  
 }  
 }  
}

