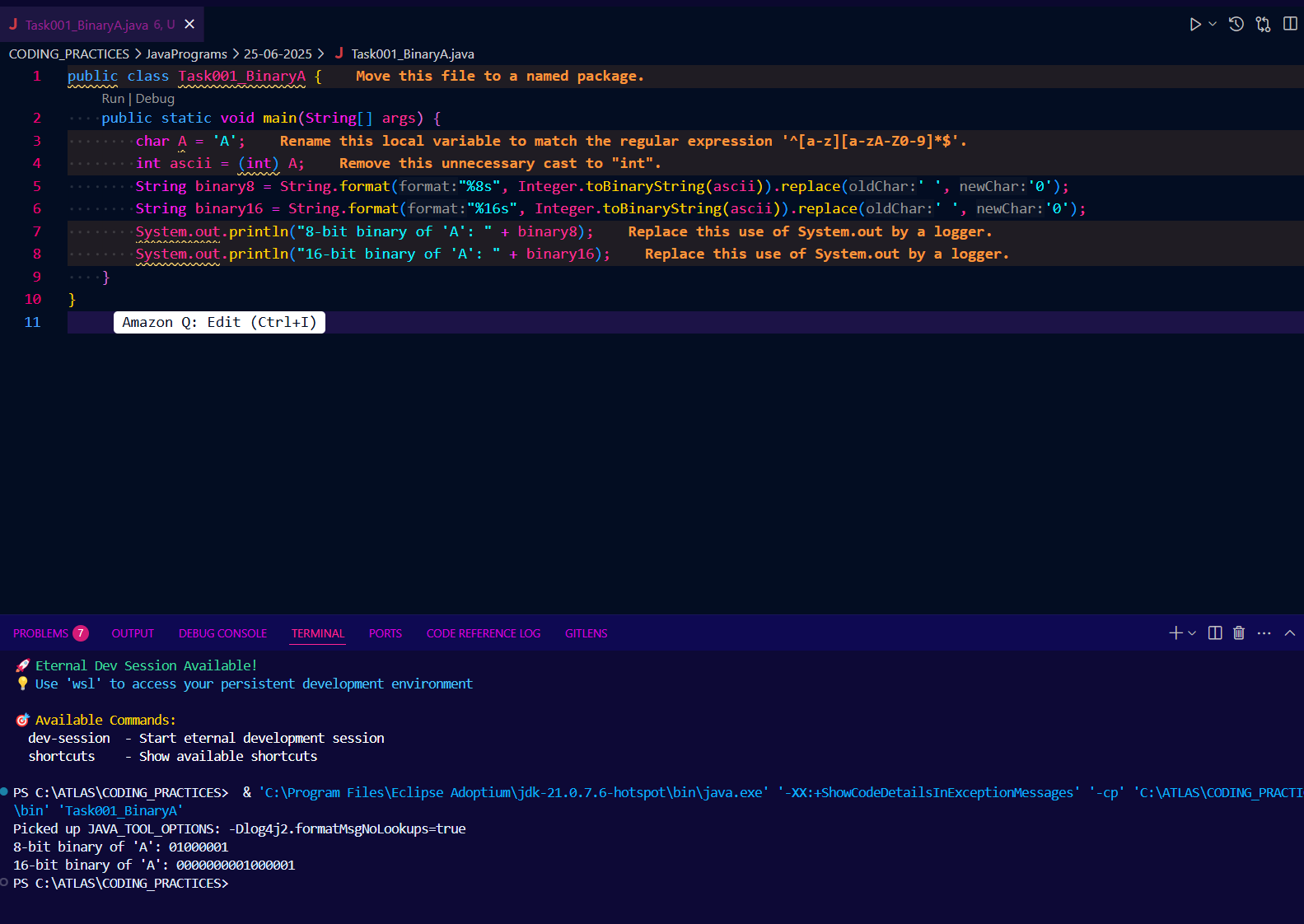
Day 12 - 25th june 2025 Data Structures

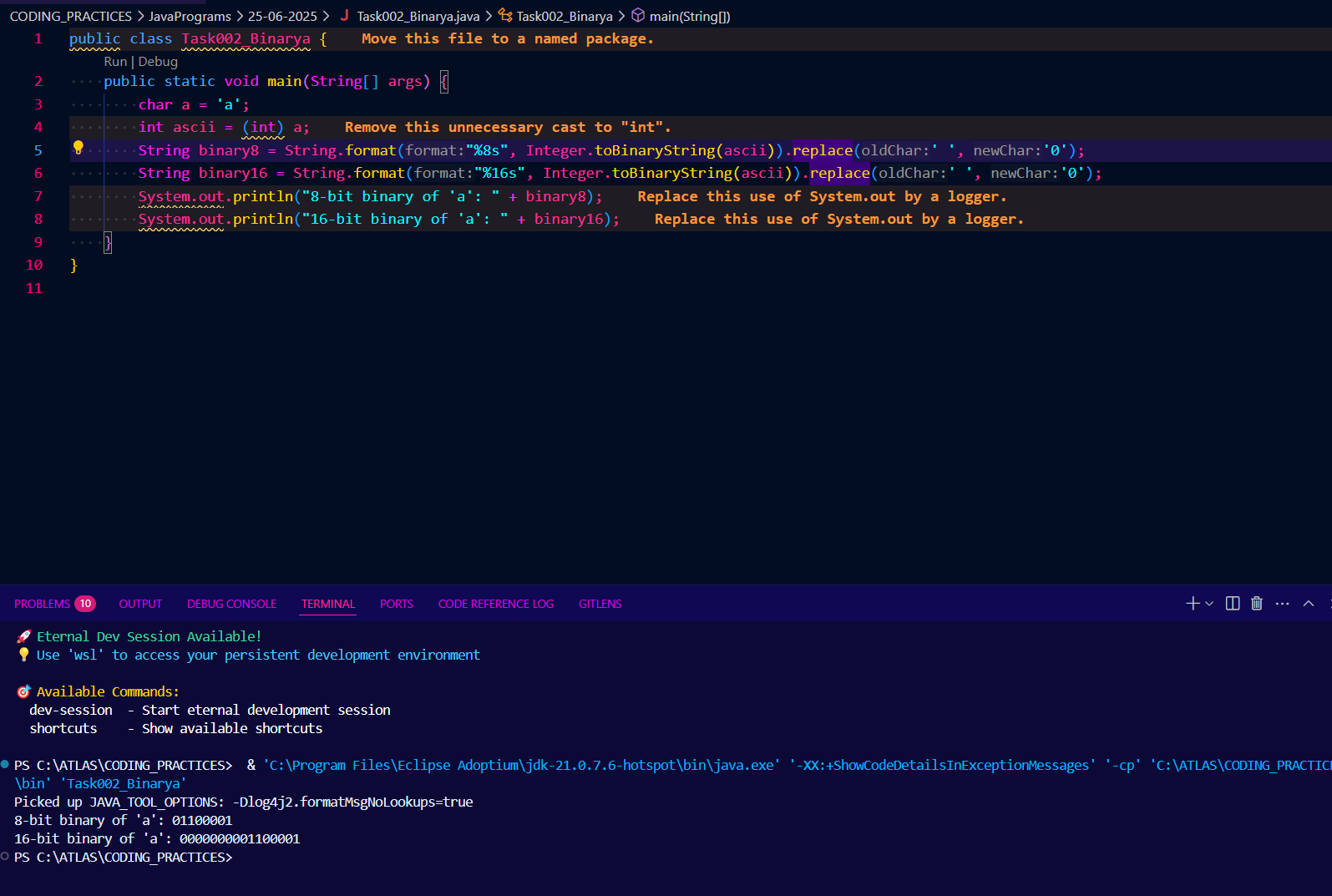
Task 1:What is the binary 8 bit representation of A?

Solution : 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 2: What is the binary value of a? (Hint ascii value is 97)

Solution : 

Task 3:Types of Computer memory with examples.. Explain ..

Solution :

1. Primary Memory (Main Memory)

- RAM (Random Access Memory): Volatile, fast, used for temporary storage while programs run. Example: DDR4 RAM in laptops.

- ROM (Read Only Memory): Non-volatile, stores firmware. Example: BIOS chip.

2. Secondary Memory

- Hard Disk Drive (HDD): Non-volatile, large capacity, slower. Example: 1TB HDD.

- Solid State Drive (SSD): Non-volatile, faster than HDD. Example: 512GB SSD.

- Optical Disks: CDs, DVDs for backup/storage.

3. Cache Memory

- Small, very fast memory between CPU and RAM. Example: L1, L2, L3 cache in processors.

4. Registers

- Smallest, fastest memory inside CPU for immediate data processing.

5. Flash Memory

- Used in USB drives, SD cards, SSDs.

6. Virtual Memory

- Uses part of secondary storage as extra RAM.

Task 4: What do you understand by data structures..?

Solution :

Data structures are specialized formats for organizing, processing, and storing data efficiently.

They define the relationship between data and the operations that can be performed on them.

Examples: Arrays, Linked Lists, Stacks, Queues, Trees, Graphs, Hash Tables.

Data structures help in efficient data access, modification, and management in programs.

Task 5:What are the operations on data structures ?

Solution : Operations vary in complexity and efficiency depending on the data structure used.Common operations on data structures include:

1. Insertion: Adding new elements.

2. Deletion: Removing elements.

3. Traversal: Accessing each element (e.g., for display).

4. Searching: Finding an element.

5. Sorting: Arranging elements in a specific order.

6. Updating: Modifying an element's value.

Task 6: What are static and dynamic arrays key points summarize in a table?(Size, performance, memory, flexibility)

Solution :

Feature Static Array Dynamic Array

Size Fixed at creation Can grow/shrink at runtime

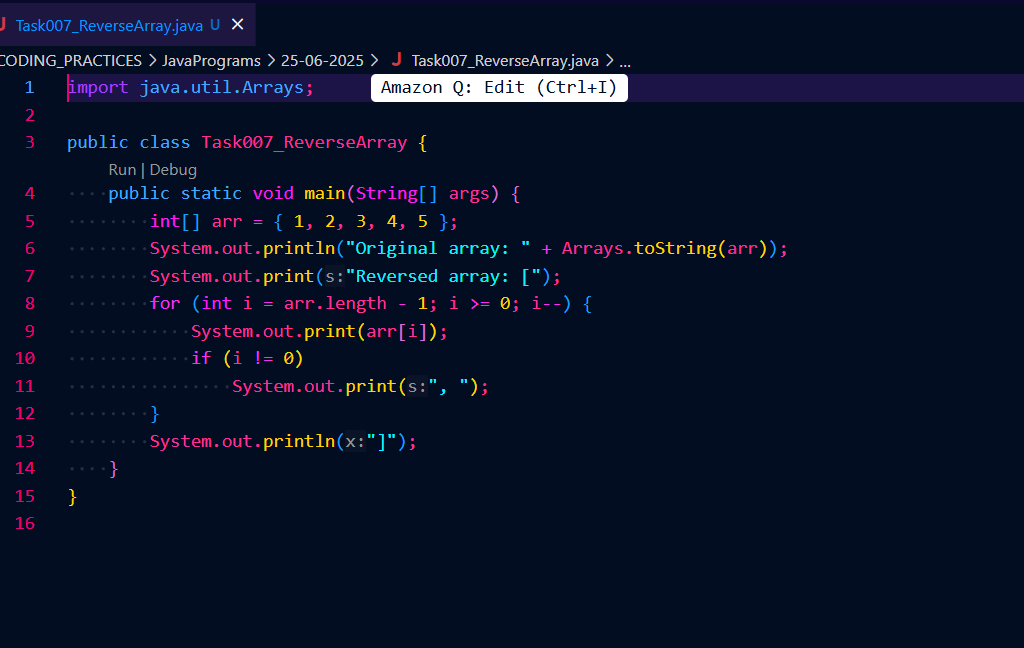
Performance Fast access, no resizing Slightly slower (resize cost)

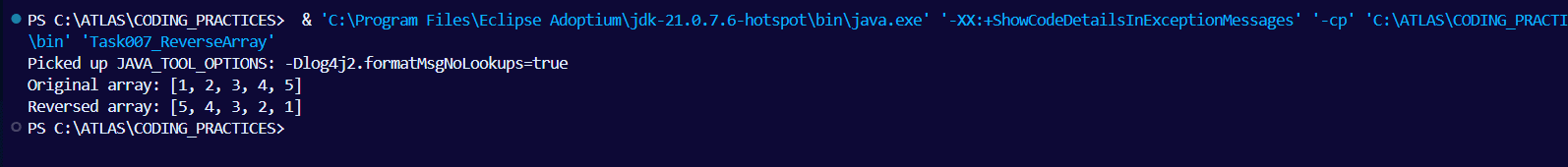
Memory Contiguous, pre-allocated May allocate new memory

Flexibility Less flexible More flexible

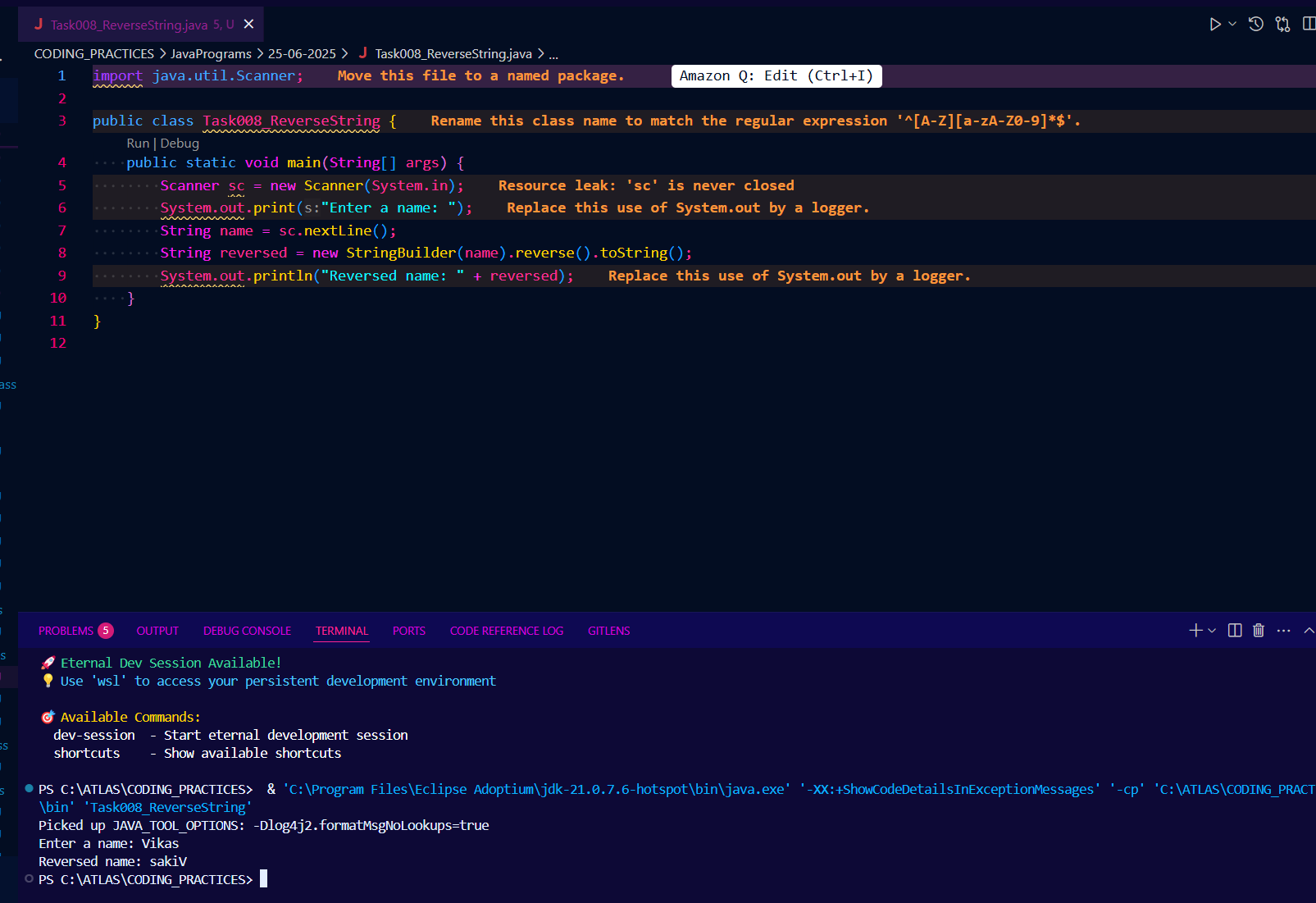
Example int[] arr = new int[10]; ArrayList<Integer> list;

Task 7: Reverse an array. write a code.(Hint : take a list of nos and display in reverse order)

Solution : 



Task 8: Reverse a string .. write a code.(Hint: take a name from the user and display the name in reverse order..)

Solution : 

Task 9: Leetcode and Hackerrank … accounts

By Zain – link given for practice

AlgoMaster.io - Master Software Engineering Interviews

Task 10: What is the above code snippet doing..?

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] + " ");

}

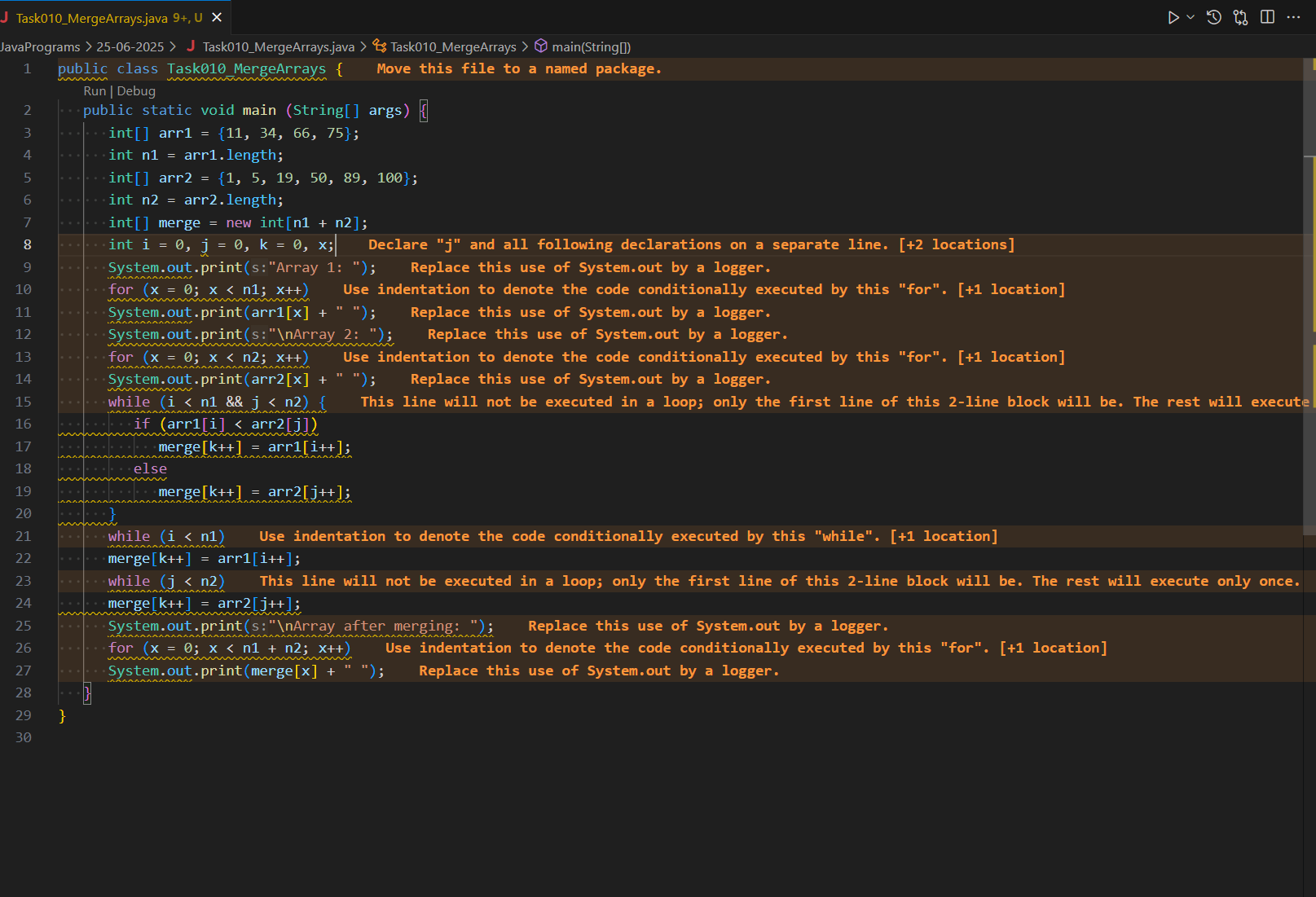
}

Solution : The code merges two sorted integer arrays (arr1 and arr2) into a single sorted array (merge).

It prints both input arrays, then iterates through both, comparing elements and adding the smaller one to the merged array.

After one array is exhausted, it copies the remaining elements from the other array.

Finally, it prints the merged, sorted array.



Task 11: What do you know about hash table?

Solution :

A hash table is a data structure that stores key-value pairs.

It uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found.

Hash tables provide fast insertion, deletion, and lookup operations.

They handle collisions using techniques like chaining or open addressing.

In Java, Hashtable and HashMap are common implementations.

Task 12:Wap to create a hash table and display them..

Hint 👍

Import java.util.Hashtable;

Import java.util.Map;

import java.util.Hashtable;

import java.util.Map;

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);

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

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

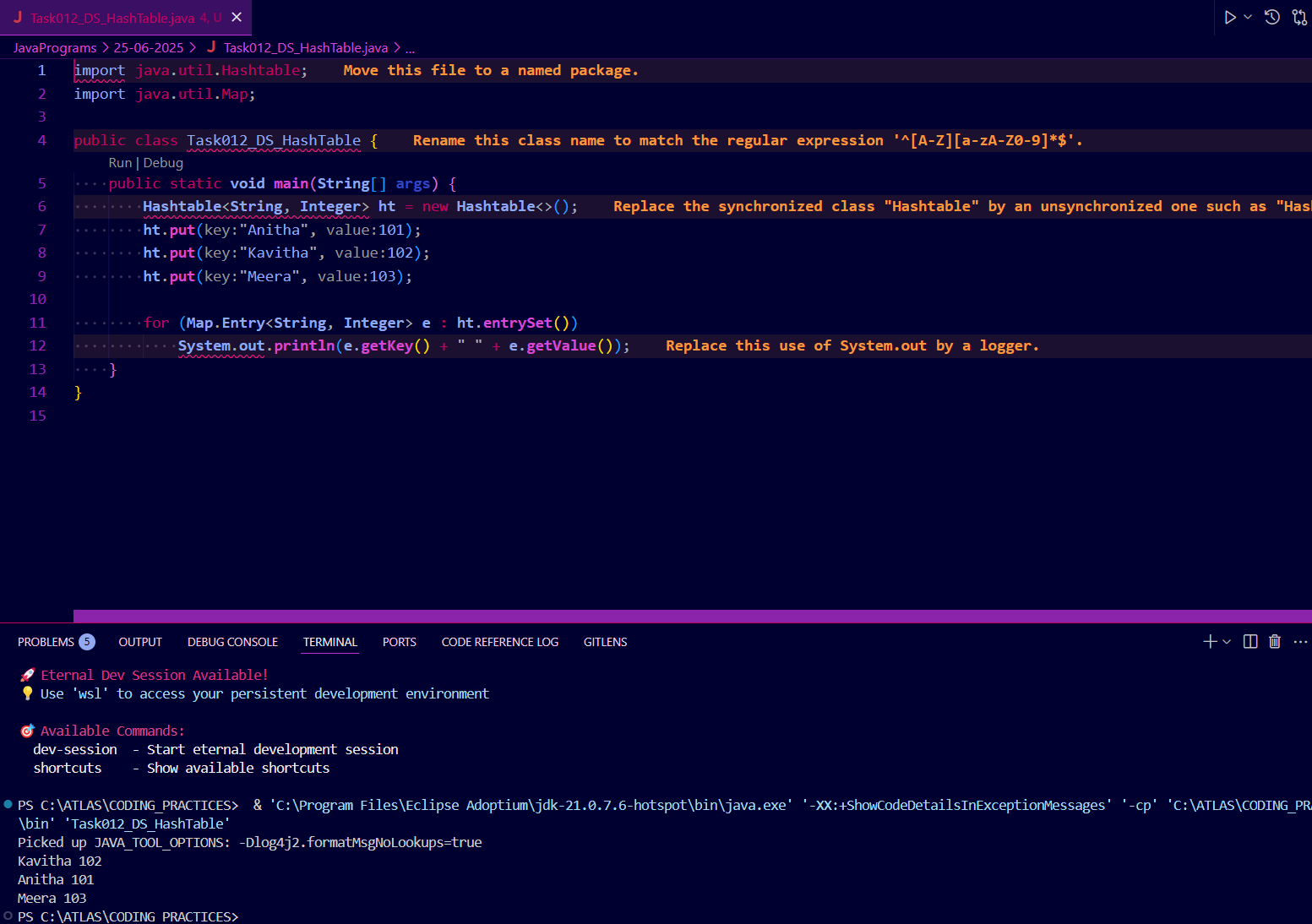
}

}

// Map is an interface

// hash table --> slower , sync , thread safe, no null value accepted

// hash map --> faster while retrieving, asynchro , only one null key and multiple null values..

Solution : 

Task 13:Wap to create a hash map and display them..

import java.io.\*;

import java.util.\*;

class Task013\_DS\_HashMap {

public static void main(String args[]) {

HashMap<Integer, String> hmobj1 = new HashMap<>();

HashMap<Integer, String> hmobj2 = new HashMap<Integer, String>();

hmobj1.put(10, "Anitha");

hmobj1.put(20, "Saritha");

hmobj1.put(30, "Ankitha");

hmobj2.put(44, "John");

hmobj2.put(55, "Steve");

hmobj2.put(66, "Jack");

System.out.println("Mapping HashMap hmobj1: " + hmobj1);

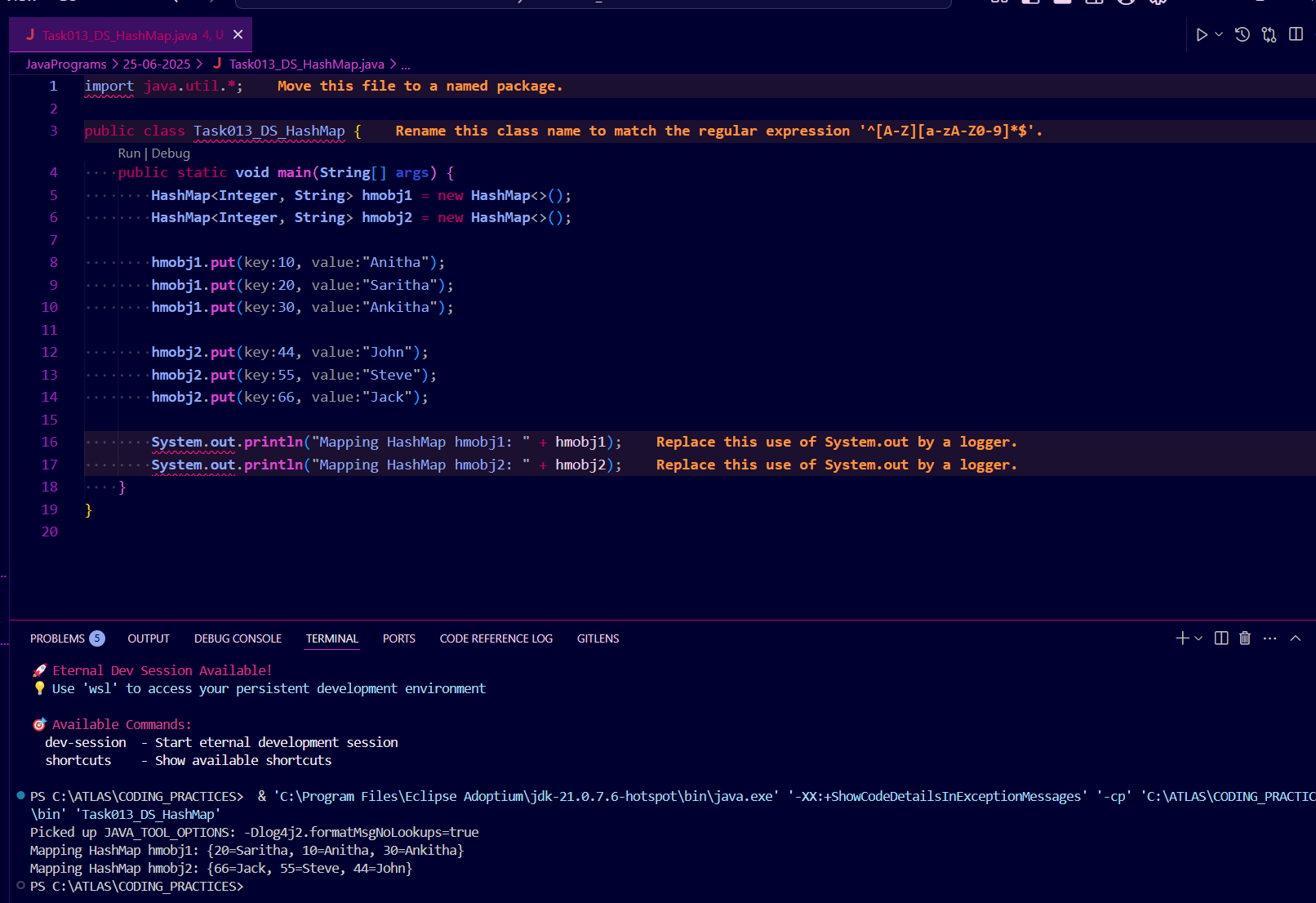
System.out.println("Mapping HashMap hmobj2: " + hmobj2);

}

}

//set -- arrayList , replace the values , updates the previous value

//put -- hash table , insert the value, puts a new value

Solution : 

Task 14: Hash table advantages and disadvantages

Solution : Hash table advantages and disadvantages

Advantages:

- Fast data access (O(1) average time for search, insert, delete)

- Efficient for large datasets

- Simple implementation for key-value pairs

- Useful for implementing caches, symbol tables, etc.

Disadvantages:

- Performance degrades with many collisions

- Fixed size (unless rehashed)

- Not ordered

- Hash functions can be tricky to design

- Uses more memory than some alternatives

- In Java, Hashtable is synchronized (slower than HashMap)

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();

}

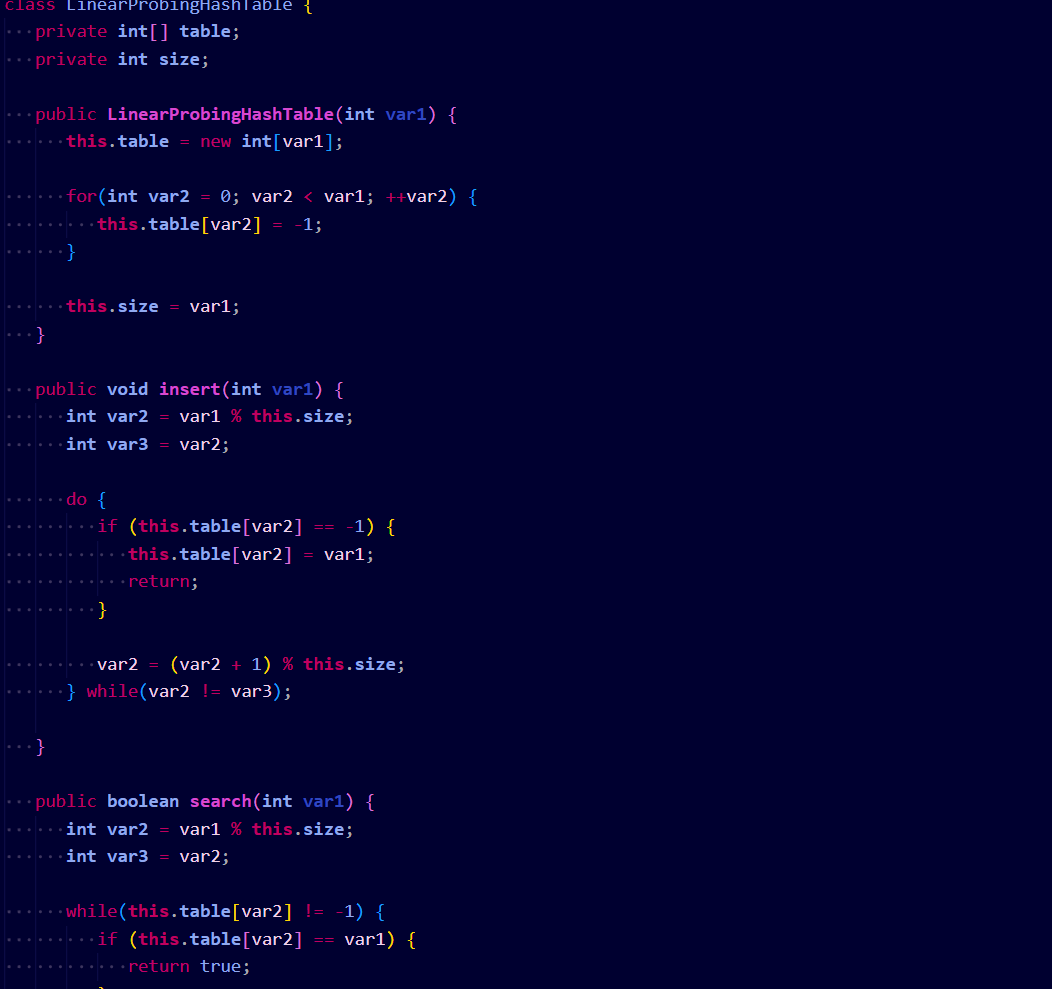
public Value put(Key key, Value value) // TBA

Solution :

Linear probing is a collision resolution technique in open addressing hash tables.

If a collision occurs, the algorithm checks the next slot (index + 1) until an empty slot is found.

This reduces the need for linked lists but can cause clustering.





Task 16: Methods of Hash table plz list them(Hash table methods List .. for your ref)

Solution : Methods of Hash table

- put(K key, V value)

- get(Object key)

- remove(Object key)

- containsKey(Object key)

- containsValue(Object value)

- isEmpty()

- size()

- clear()

- keySet()

- values()

- entrySet()

- rehash() (protected)

- clone()

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.

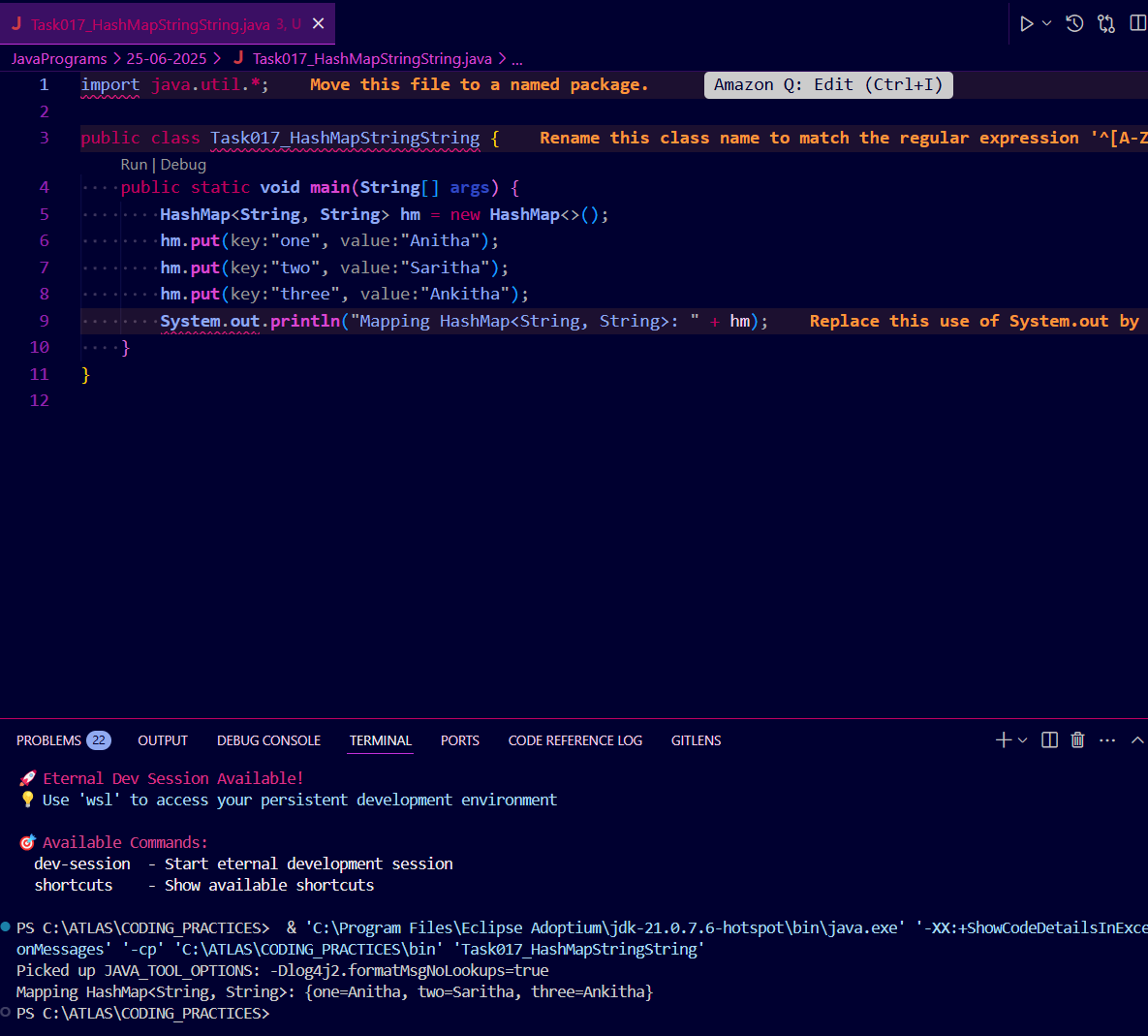
rehash(): Increases the size of the Hashtable and rehashes all of its keys. This method is protected and typically handled internally by the Hashtable for performance optimization.

clone(): Returns a shallow copy of the Hashtable instance.

===========================

Task 17: In Task 13 of hash Map .. we were using string and integer / integer and string

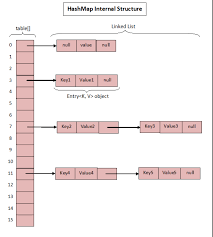
Like HashMap<Integer, String> .Can you change to String and string and c if it works Like this HashMap<String, String> ? will this work?

Solution : 

Task 18:Explain the internal working of a HashMap. With diagram..

Solution : A HashMap in Java stores key-value pairs using a hash table. Internally, it uses an array of buckets, where each bucket is a linked list (or a tree for high-collision buckets in Java 8+). The key's hashCode() determines the bucket index. If multiple keys hash to the same bucket (collision), they are stored in the list/tree at that bucket.

Diagram:



- Each bucket can have a linked list/tree of entries.

- When a key is added, its hashCode is used to find the bucket.

- If the bucket is empty, the entry is added.

- If not, the list/tree is traversed to check for duplicates or add at the end.

- On get(), the hashCode is used to find the bucket, then the list/tree is searched for the key.

This structure allows O(1) average time for put/get, but can degrade to O(n) if many collisions occur.

Task 19: 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..

Sample 👍

import java.util.\*;

import java.io.\*;

public class Task019\_Ds\_HashMapNull {

public static void main(String[] args) {

HashMap hmap=new HashMap();

hmap.put(101,"Prasunamba");

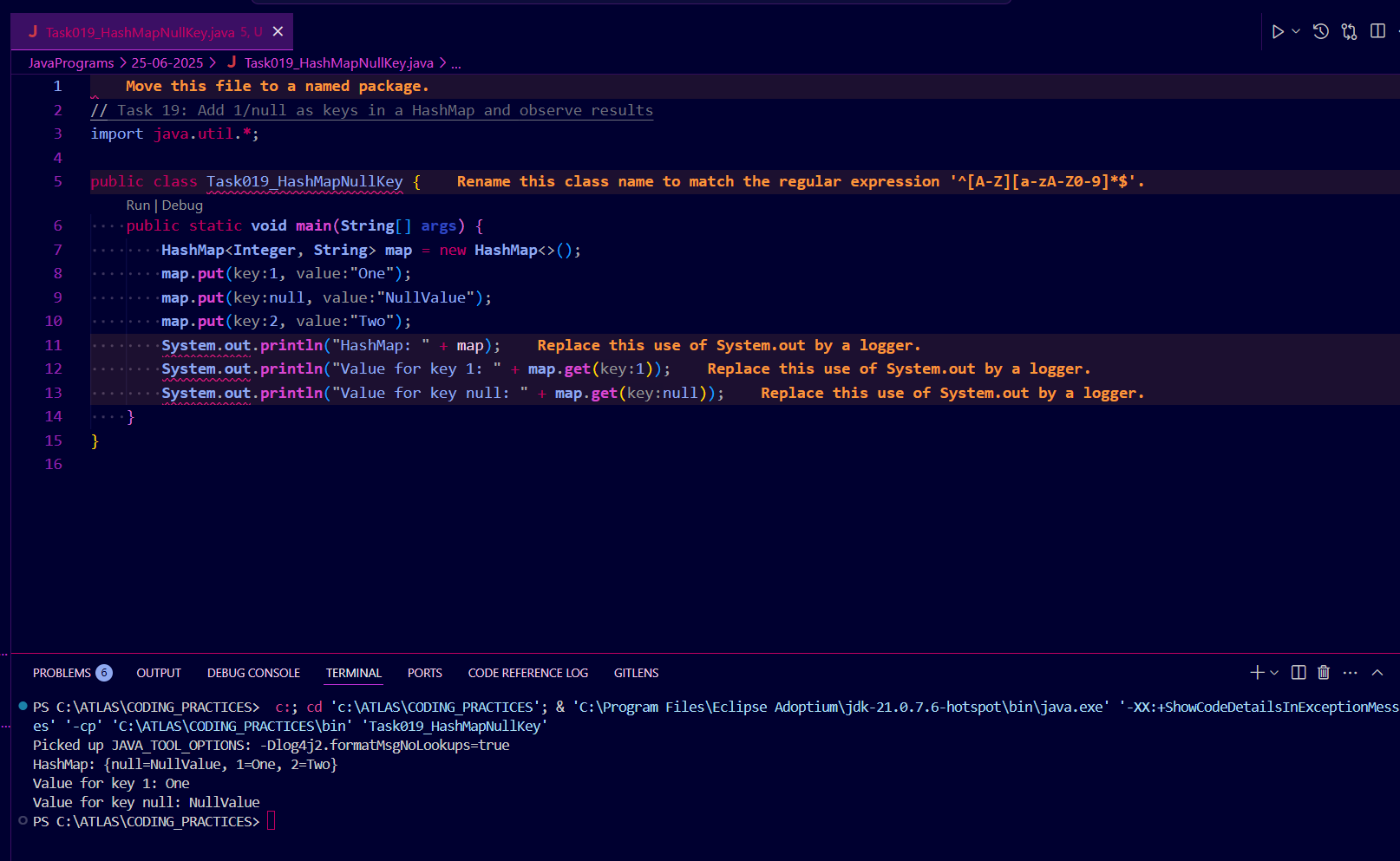
hmap.put(null,"Meher");

hmap.put(null,".MK");

System.out.println(hmap);

}

}

Solution : 

Task 20: How many methods are there to create a hash Map?

Solution : Methods to Create a HashMap in Java:

So, there are 4 main ways to create a HashMap in Java.

­

1. Using the default constructor:

HashMap<K, V> map = new HashMap<>();

2. With initial capacity:

HashMap<K, V> map = new HashMap<>(int initialCapacity);

3. With initial capacity and load factor:

HashMap<K, V> map = new HashMap<>(int initialCapacity, float loadFactor);

4. From another Map:

HashMap<K, V> map = new HashMap<>(Map<? extends K, ? extends V> m);

5 min -> 17.35 to 17.40

Task020\_DS\_HashMapCreateMethods:

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

Task 21:

import java.util.Collections;

import java.util.HashMap;

import java.util.Map;

public class task021\_DS\_HashMap\_SyncMap {

public static void main(String[] args) {

HashMap<String, Integer> Hm1 = new HashMap<String, Integer>();

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

}

}

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

Home Tasks:

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

Q) Do hash table have linked list internally?

A) Hash Table Internals :

- Hash table uses an array of buckets to store key-value pairs.

- Each key's hashCode() determines the bucket index.

- Collisions are handled by chaining (linked list/tree) or open addressing.

- Rehashing occurs when load factor exceeds threshold.

- Provides O(1) average time for put/get operations.

Q) Do collisions occur in hash Maps? What are they?

A) Hash Table Collisions :

- Collision: When two keys hash to the same bucket.

- Handled by chaining (linked list/tree) or open addressing (probing).

- Good hash functions and resizing reduce collisions.

Q)What is load factor and how the capacity increases?

Q)Map and set

A)Map vs Set :

- Map: Stores key-value pairs. Keys are unique. Example: HashMap, TreeMap.

- Set: Stores unique values only, no duplicates. Example: HashSet, TreeSet.

- Map allows value lookup by key; Set does not.

Add on Task: Hash Table - Linear Probing

import java.util.Scanner;

class LinearProbingHashTable {

private int currentSize, maxSize;

private String[] keys;

private String[] vals;

public LinearProbingHashTable(int capacity) {

currentSize = 0;

maxSize = capacity;

keys = new String[maxSize];

vals = new String[maxSize];

}

public void makeEmpty() {

currentSize = 0;

keys = new String[maxSize];

vals = new String[maxSize];

}

public int getSize() {

return currentSize;

}

public boolean isFull() {

return currentSize == maxSize;

}

public boolean isEmpty() {

return getSize() == 0;

}

public boolean contains(String key) {

return get(key) != null;

}

private int hash(String key) {

return key.hashCode() % maxSize;

}

public void insert(String key, String val) {

int tmp = hash(key);

int i = tmp;

do {

if (keys[i] == null) {

keys[i] = key;

vals[i] = val;

currentSize++;

return;

}

if (keys[i].equals(key)) {

vals[i] = val;

return;

}

i = (i + 1) % maxSize;

} while (i != tmp);

}

public String get(String key) {

int i = hash(key);

while (keys[i] != null)

{

if (keys[i].equals(key))

return vals[i];

i = (i + 1) % maxSize;

}

return null;

}

public void remove(String key) {

if (!contains(key))

return;

int i = hash(key);

while (!key.equals(keys[i]))

i = (i + 1) % maxSize;

keys[i] = vals[i] = null;

for (i = (i + 1) % maxSize; keys[i] != null; i = (i + 1) % maxSize) {

String tmp1 = keys[i], tmp2 = vals[i];

keys[i] = vals[i] = null;

currentSize--;

insert(tmp1, tmp2);

}

currentSize--;

}

public void printHashTable() {

System.out.println("\nHash Table: ");

for (int i = 0; i < maxSize; i++)

if (keys[i] != null)

System.out.println(keys[i] +" "+ vals[i]);

System.out.println();

}

}

public class LinearProbingHashTableTest {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

System.out.println("Hash Table Test\n\n");

System.out.println("Enter size");

LinearProbingHashTable lpht = new LinearProbingHashTable(scan.nextInt() );

char ch;

do {

System.out.println("\nHash Table Operations\n");

System.out.println("1. insert ");

System.out.println("2. remove");

System.out.println("3. get");

System.out.println("4. clear");

System.out.println("5. size");

int choice = scan.nextInt();

switch (choice) {

case 1 :

System.out.println("Enter key and value");

lpht.insert(scan.next(), scan.next() );

break;

case 2 :

System.out.println("Enter key");

lpht.remove( scan.next() );

break;

case 3 :

System.out.println("Enter key");

System.out.println("Value = "+ lpht.get( scan.next() ));

break;

case 4 :

lpht.makeEmpty();

System.out.println("Hash Table Cleared\n");

break;

case 5 :

System.out.println("Size = "+ lpht.getSize() );

break;

default :

System.out.println("Wrong Entry \n ");

break;

}

lpht.printHashTable();

System.out.println("\nDo you want to continue (Type y or n) \n");

ch = scan.next().charAt(0);

} while (ch == 'Y'|| ch == 'y');

}

}

Solution :

