I**D: hrajranj**

**Day 12 – 25th June 2025**

**Task 1:**

**What is the binary 8-bit representation of A?**

**Ans:**

ASCII value of A is 65

65/2=1

32/2=0

16/2=0

8/2=0

4/2=0

2/2=0

1/2=1

It will be written from bottom to top  A==   01000001

**Task 2:**

**What is the binary value of a?**

**Ans:**

ASCII code for a is 97

97/2= 1

48/2= 0

24/2= 0

12/2= 0

 6/2= 0

 3/2= 1

 1/2= 1

It will be written from  bottom to top  a==   01100001

**Task 3:**

**Types of Computer memory with examples. Explain.**

**Ans:**

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

This is the memory directly accessed by the CPU. It’s fast but temporary — used during the execution of programs.

A. RAM (Random Access Memory)

Loses data when power is off

CPU can both read from and write to RAM

Stores current instructions and data

Much faster than secondary memory

Example:

When you open a Word document, it loads into RAM.

DDR4 RAM, LPDDR5 (laptops, phones)

Types:

SRAM (Static RAM) – faster, used in cache

DRAM (Dynamic RAM) – slower, needs refreshing

B. ROM (Read-Only Memory)

Retains data even when power is off

Can't be modified (in most cases)

Permanent

Used to store firmware (like BIOS)

Example:

BIOS/UEFI that runs before your OS starts

Embedded systems (microwave controllers, etc.)

Types:

PROM (Programmable ROM)

EPROM (Erasable PROM)

EEPROM (Electronically Erasable PROM, like Flash)

**2. Secondary Memory (Storage)**

This memory is not directly accessed by CPU, but stores data permanently. It is slower but larger and cheaper than primary memory.

A. Hard Disk Drive (HDD)

Uses magnetic storage

Stores OS, apps, and files

Slower but high capacity (1 TB, 2 TB)

B. Solid State Drive (SSD)

Uses flash memory (no moving parts)

Much faster than HDD

Used in modern laptops/phones

C. Optical Drives

CD/DVD/Bluray

Slower and becoming obsolete

D. USB Flash Drives / Pen Drives

Portable

Uses Flash EEPROM

**3. Cache Memory**

A small, ultra-fast memory inside or very close to the CPU

Stores frequently used instructions/data

Reduces time CPU takes to access RAM

Example:

CPU accessing a variable in a tight loop

L1, L2, L3 cache (in increasing size and decreasing speed)

**4. Virtual Memory**

Not physical memory, but a portion of storage (HDD/SSD) used as RAM extension

Helps run larger programs than RAM can hold

Managed by the operating system (using a swap file or page file)

**5. Registers**

Smallest and fastest memory

Built into the CPU

Stores intermediate data like loop counters, addresses

**Task 4:**

**What do you understand by data structures...?**

**Ans:**

A data structure is just a way to organize and store data so we can use it easily and efficiently. Example: A to-do list where tasks are ordered (structured data). It helps us put data in order so we can find, use, or change it quickly.

**Task 5:**

**What are the operations on data structures?**

**Ans:**

Operations on data structures: 1. Insertion 2. Deletion 3. Traversal 4. Searching 5. Sorting 6. Updation 7. Merging 8. Splitting 9. Accessing elements 10. Resizing (for dynamic structures)

**Task 6:**

**What are static and dynamic arrays key points summarizing in a table**

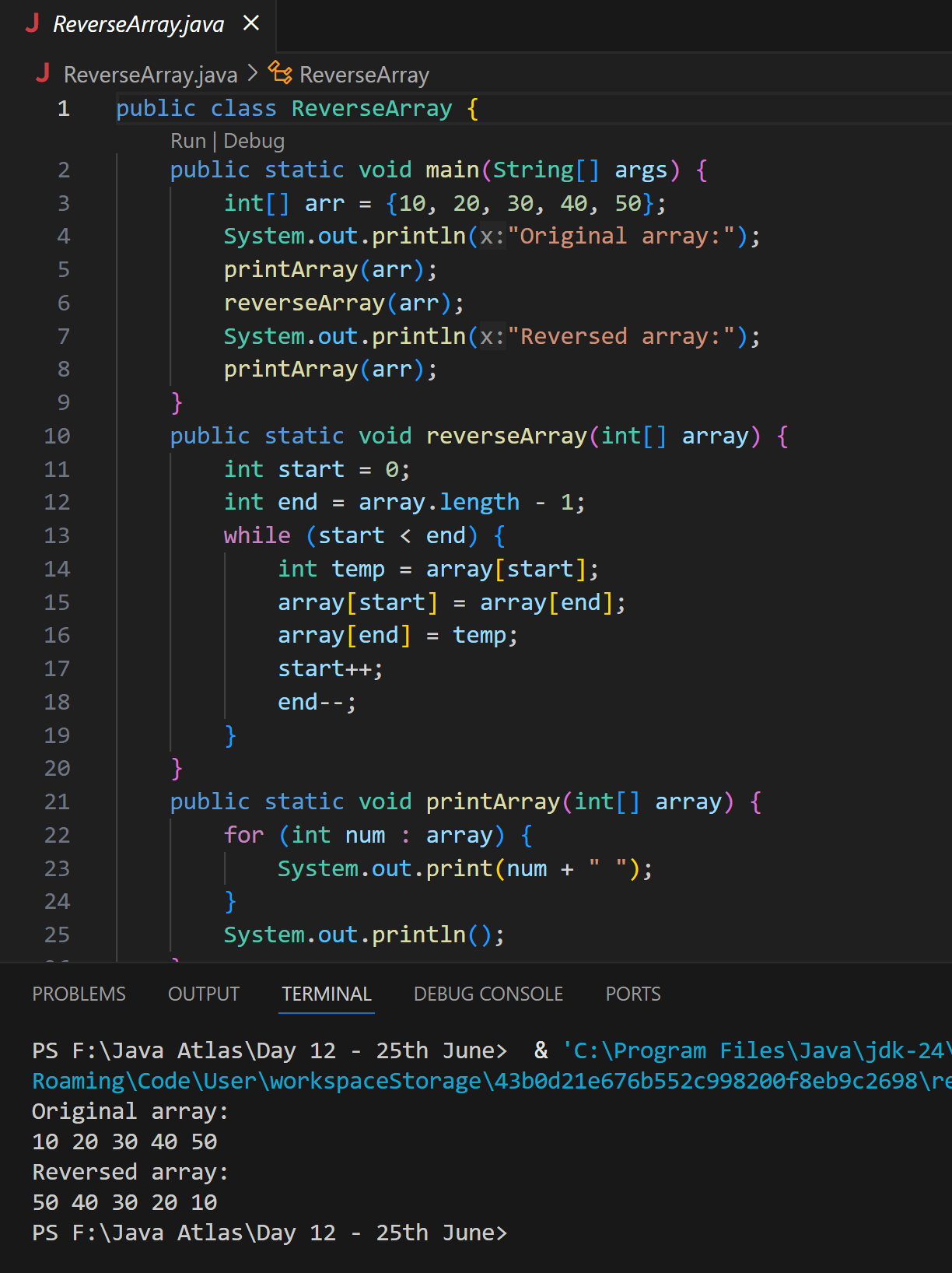
**Ans:**

|  |  |  |
| --- | --- | --- |
| Feature | Static Array | Dynamic Array |
| Size | Fixed at compile time | Can grow/shrink at runtime |
| Flexibility | Not flexible | Highly flexible |
| Memory Allocation | Compile-time | Runtime |
| Performance | Faster (no resizing) | Slightly slower (due to resizing logic) |
| Storage | Stored in contiguous memory | May reallocate to new memory locations |
| Example | int[] arr = new int[5]; | ArrayList<Integer> list = new ArrayList<>(); |
| Resizing | Not allowed | Allowed using internal mechanisms |

**Task 7:**

**Reverse an array. write a code.**

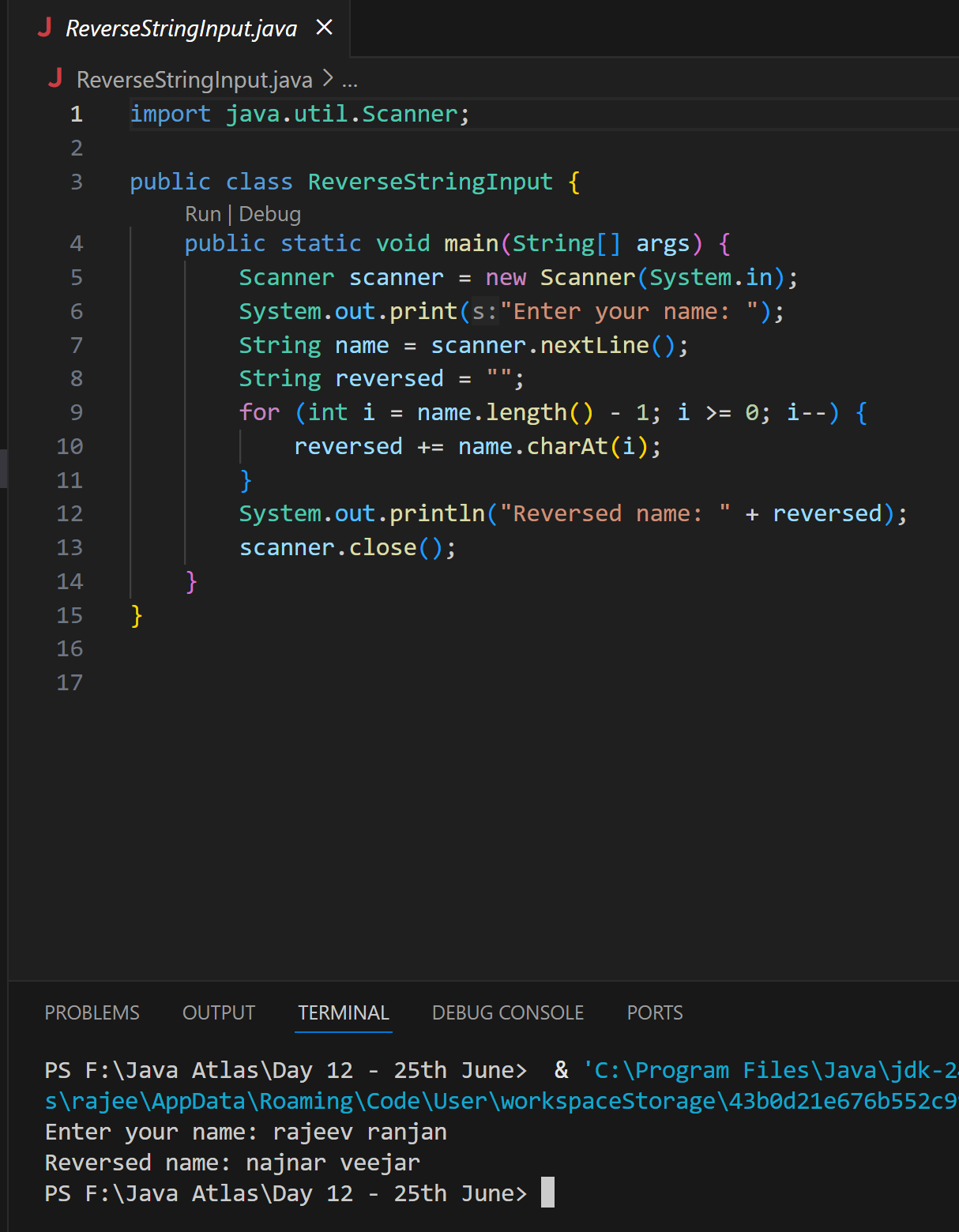
**Ans:**



**Task 8:**

**Reverse a string. write a code.**

**Ans:**

****

**Task 9:**

Leetcode and Hackerrank … accounts ..

**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] + " ");

   }

}

**What is the above code snippet doing...?**

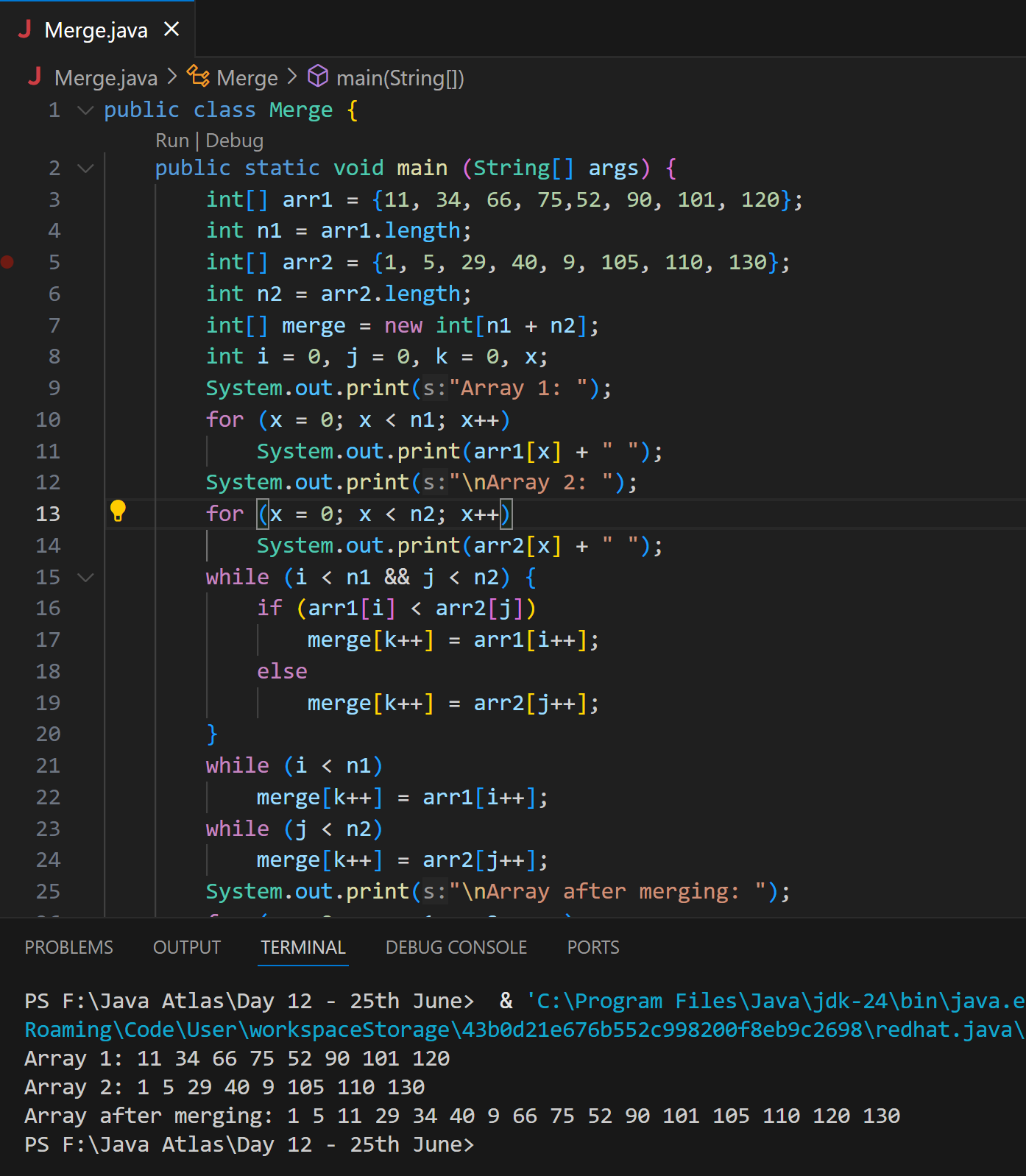
**Ans:**

The program merges two sorted arrays (arr1 and arr2) into a single sorted array (merge).

It compares elements from both arrays using a loop.

The smaller element is placed into the merge array. After one array is fully traversed, the remaining elements of the other are copied.

Finally, it prints all three arrays: arr1, arr2, and the merged result.



**Task 11:**

**What do you know about hash table?**

**Ans:** A hash table is a data structure that stores data in key-value pairs, and allows for very fast access, usually in O(1) time on average.

Working:

1. The key is passed through a hash function.

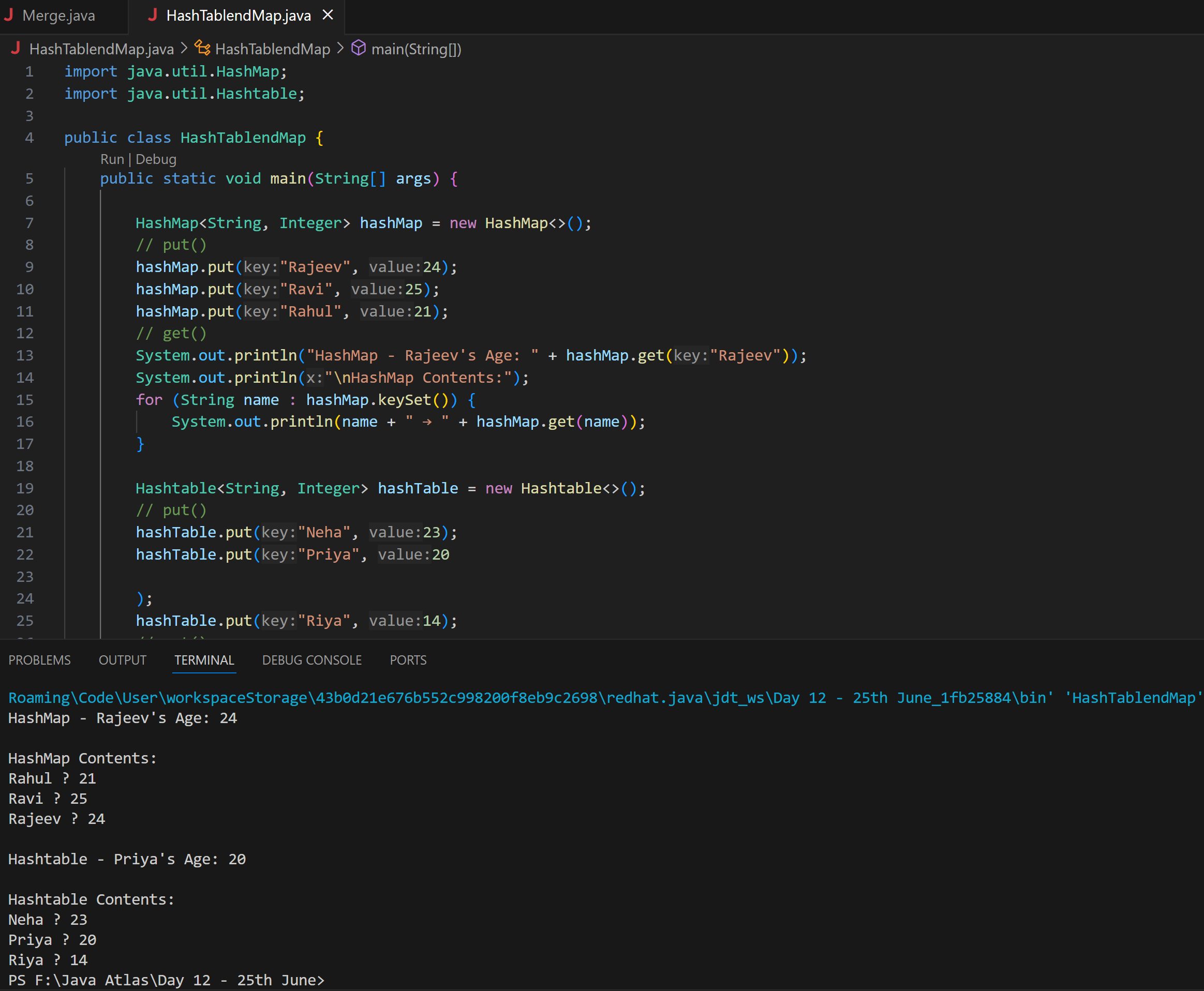
2. The hash function returns an index in an array.

3. The value is stored at that index.

**Task 12:**

**Wap to create a hash table and display them.**

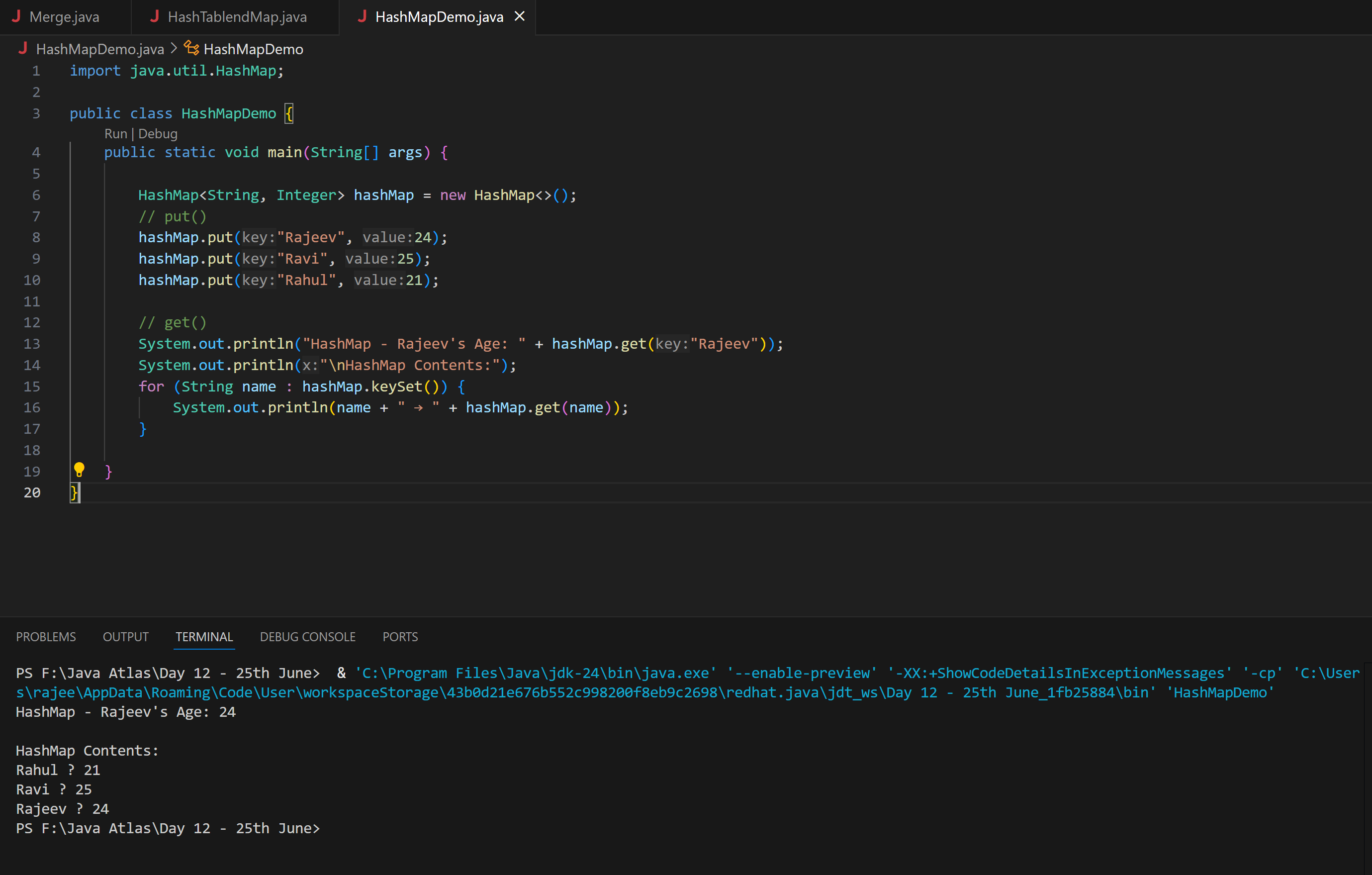
**Ans:**

****

**Task 13:**

**Wap to create a hash map and display them.**

**Ans:**

****

**Task 14:**

**Hash table advantages and disadvantages**

**Ans:**

HashMap

Advantages:

* Fast lookup, insertion, and deletion (average O(1) time).
* Allows one null key and multiple null values.
* Not synchronized, so faster in single-threaded environments.
* Part of the Java Collections Framework (modern and flexible).
* Can be easily converted to a thread-safe version using Collections.synchronizedMap() or use ConcurrentHashMap.

Disadvantages:

* Not thread-safe by default (use with care in multi-threaded environments).
* No guaranteed order (use LinkedHashMap if you want insertion order).
* Can consume more memory (buckets, keys, values, etc.).
* Performance degrades in case of many hash collisions.

Hashtable

Advantages:

* Thread-safe: All methods are synchronized.
* Good for legacy code that requires synchronization.
* Supports basic key-value operations similar to HashMap.

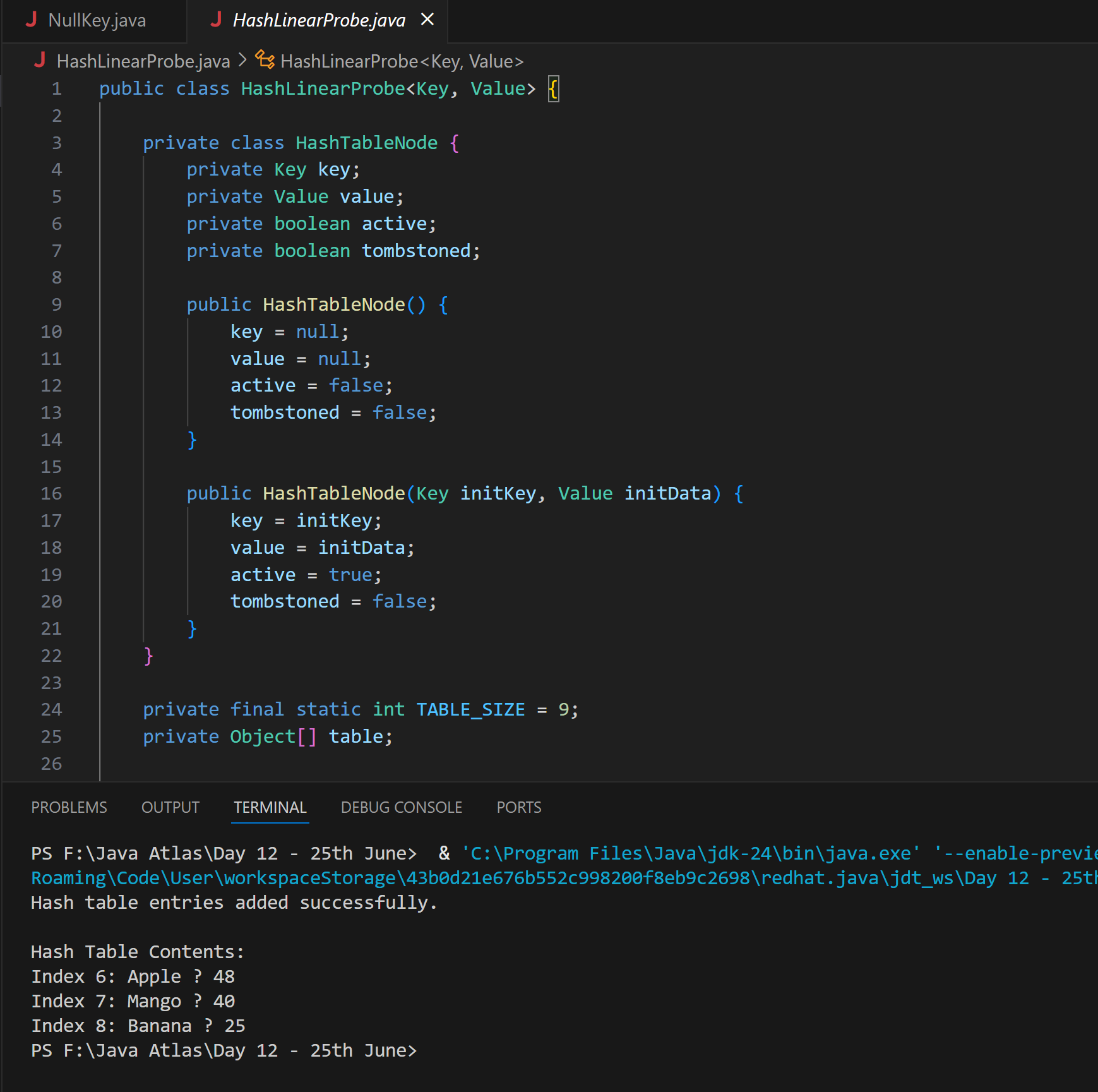
Disadvantages:

* Slower performance due to synchronization overhead.
* Does not allow null key or values.
* Considered legacy, replaced by HashMap and ConcurrentHashMap in modern code.
* Not part of the newer Collections Framework enhancements.
* Less flexible and extensible.

**Task 15:**

**Linear probing in Hash table**

**Ans:**

****

**Task 16:**

**Methods of Hash table plz list them.**

**Ans:**

1. put(K key, V value)
2. get(Object key)
3. remove(Object key)
4. containsKey(Object key)
5. containsValue(Object value)
6. isEmpty()
7. size()
8. clear()
9. keySet()
10. values()
11. entrySet()
12. equals(Object o)
13. hashCode()
14. clone()
15. toString()

**Task 17:**

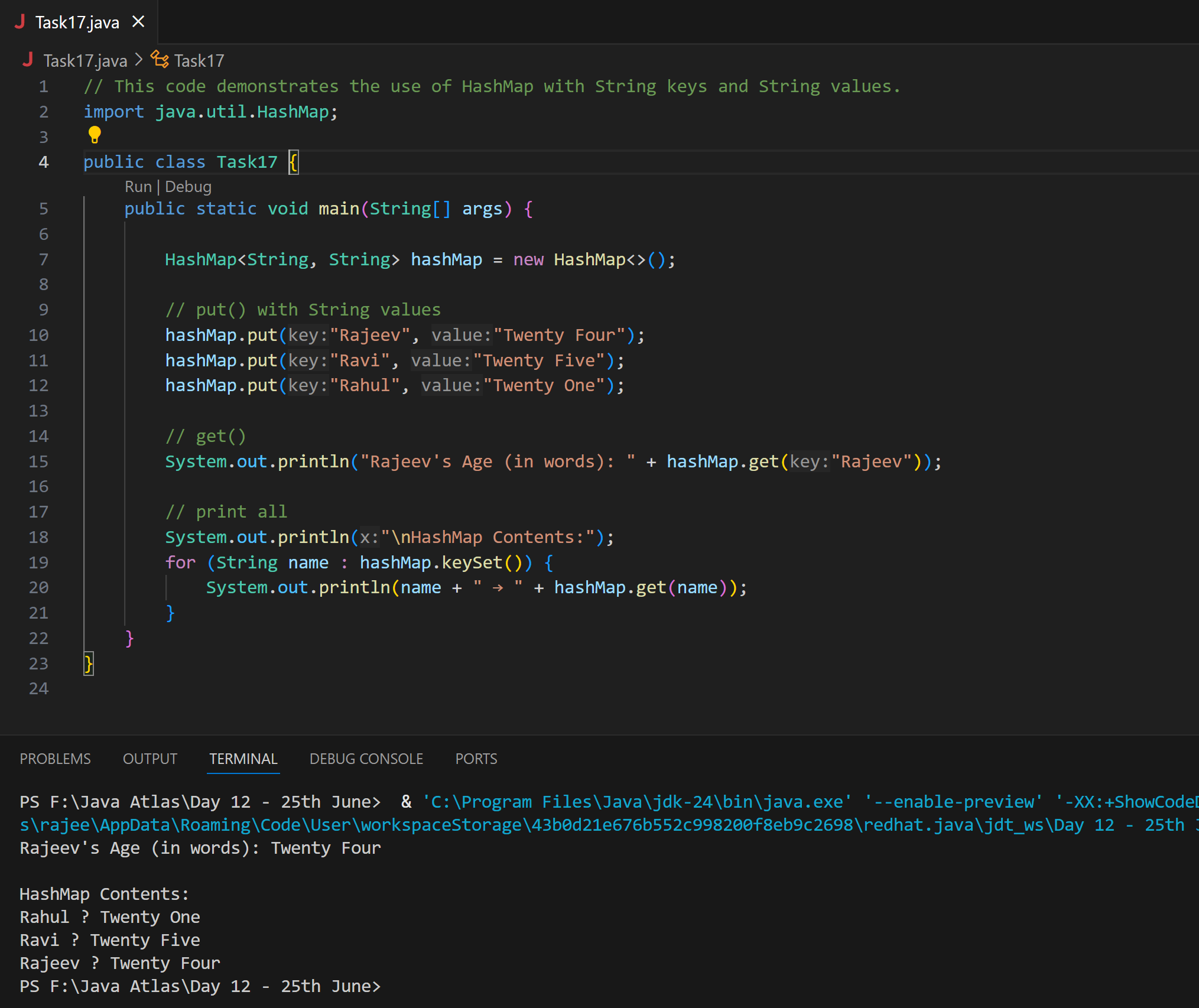
In Task 13 of hash Map .. we were using string and integer / integer and string

Like HashMap<Integer, String>

Can you change to String and string and c if it works

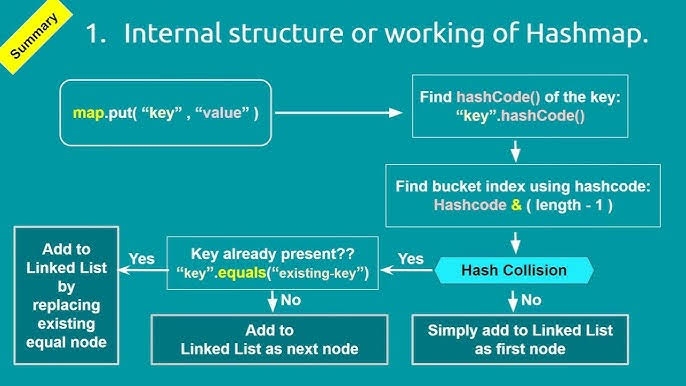
Like this HashMap<String, String>   ? will this work?

Ans:



**Task 18:**

**Explain the internal working of a HashMap. With diagram.**



HashMap is a data structure which stores data in key-value pairs.

Internal working:

1. Hashing
2. Index calculation
3. Storing data
4. Handling collision
5. Retrieving data

1. Hashing

When we call a method to insert data, Java calls key.hashCode() to generate an integer hash code. Suppose, when we call put("A", 1), Java first calls: key.hashCode(); This returns an integer (e.g., 65 for "A").

2. Index Calculation

That hash code is converted into a bucket index: index = hash(key) % array.length. So if array length = 8, then index = 65 % 8 = 1

3. Storing data/ Bucket Access

If the bucket already has entries (collision), use linear probing, If the bucket at index 1 is empty, it stores the key-value pair there.

4. Collision Handling

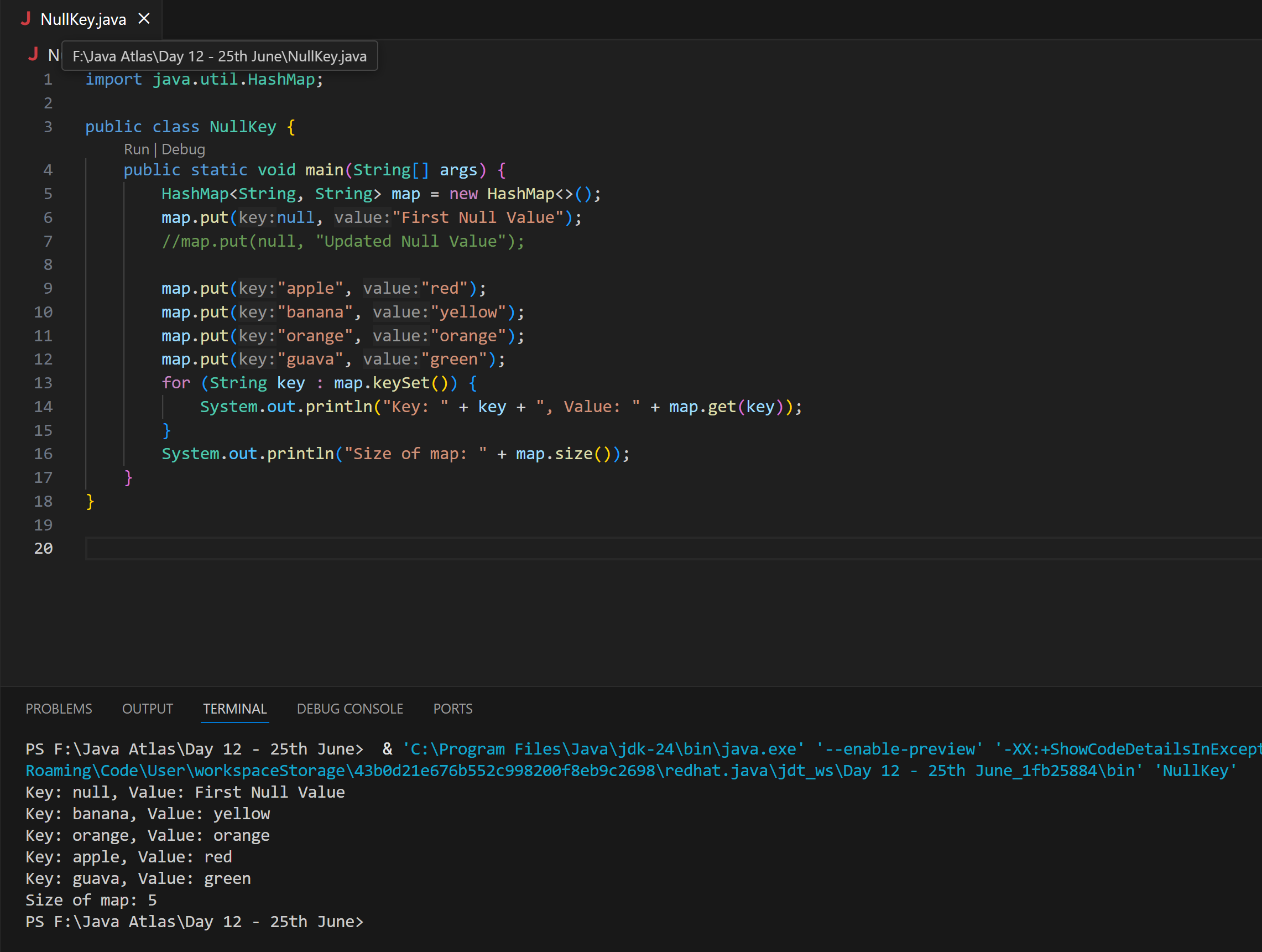
If another key also lands at index 1 (same hash % size), that’s a collision. Java stores the new entry as a linked list or a tree (if too many entries). This helps keep time complexity close to O(1).

5. Retrieving a Value

When you do map.get("A"): Java again hashes "A" → finds index, goes to that bucket - Searches the linked list (or tree) for the key "A", Returns the value 1

**Task 19:**

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

****

**Task 20:**

**How many methods are there to create a hash Map?**

**Ans:**

1. new HashMap<>()

 → Creates an empty HashMap with default size and settings.

2. new HashMap<>(initialCapacity)

 → Creates a HashMap with a given starting bucket size to reduce resizing.

3. new HashMap<>(initialCapacity, loadFactor)

 → Sets both the starting size and the threshold for resizing the map.

4. new HashMap<>(existingMap)

 → Copies all entries from another map into a new HashMap.

5. Map.of(...) (immutable, Java 9+)

 → Quickly creates a small read-only map with fixed key-value pairs.

6. Map.ofEntries(...) (immutable, Java 9+)

 → Builds a read-only map using Map.entry(key, value) format.

7. Double brace initialization new HashMap<>() {{ ... }}

 → Shortcut to add entries at creation, but not recommended for real use.

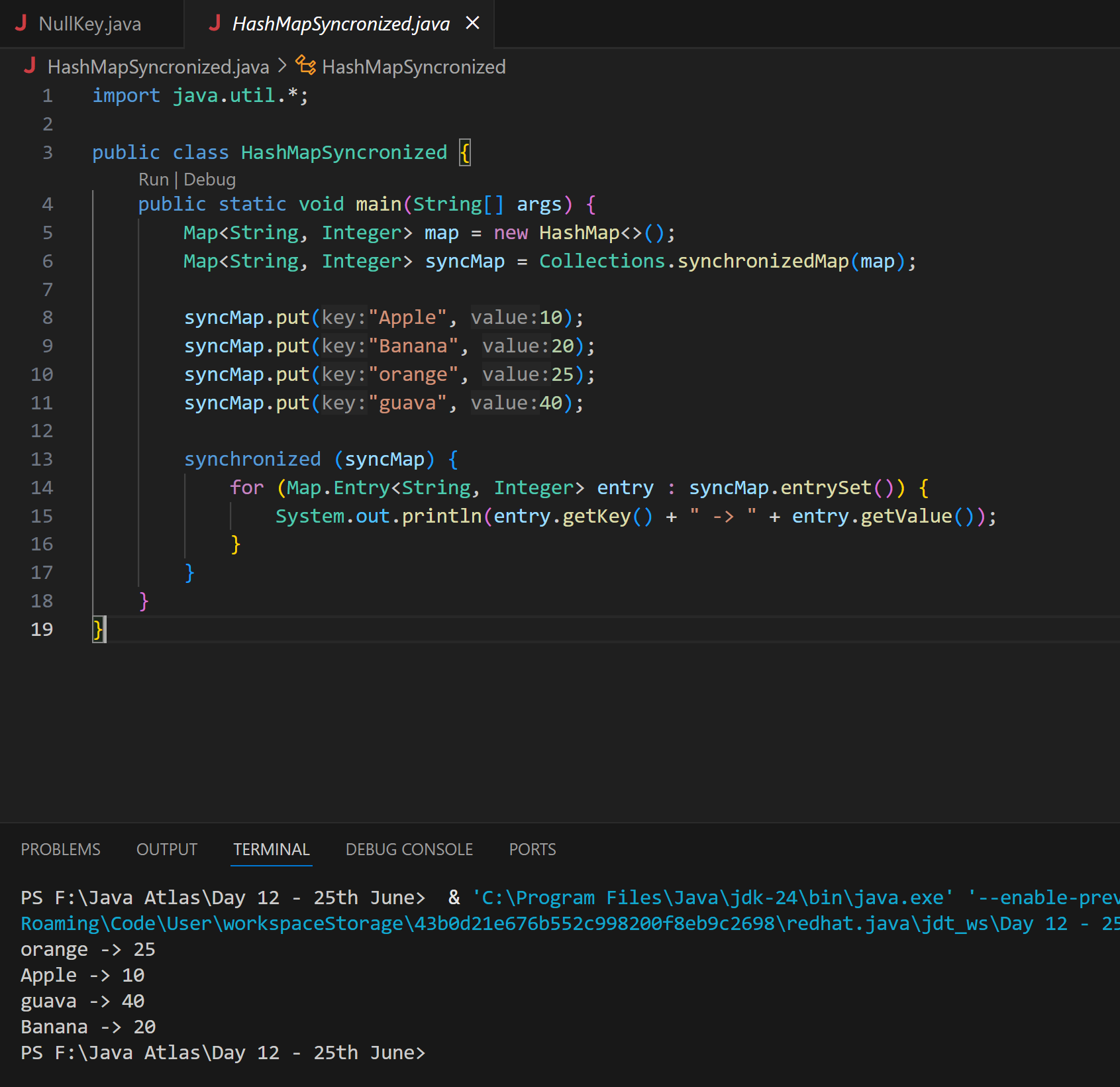
8. Using Java Streams + Collectors.toMap()

 → Converts data from arrays, lists, or streams into a HashMap.

**Task 21:**

**Wap to make a Hashmap synchronized.**

**Ans:**

****