**Day 12 Docs**

**Task - 01:**

**What do you understand about data structures?**

A data structure is a way to store, organize, and manage data so we can use it efficiently. Data structures are like tools in a toolbox. You pick the right tool depending on the job. An array is like a simple box, but sometimes you need a drawer (stack), a queue (line), or even a network (graph).

I understand that data structures are essential for solving problems efficiently in programming. Without them, managing data would be slow, messy, and hard — especially in large or complex applications.

**Task -02:**

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

### 1. Linear Data Structures

These store data in a sequence (one after another).

* Array
* Linked List (Singly, Doubly, Circular)
* Stack
* Queue (Simple, Circular, Priority Queue, Deque)

### 2. Non-Linear Data Structures

These store data in a hierarchical or networked format.

* Tree  
  + Binary Tree
  + Binary Search Tree (BST)
  + AVL Tree
  + Heap (Min Heap / Max Heap)
  + B-Tree, B+ Tree
  + Trie (Prefix Tree)
* Graph  
  + Directed Graph
  + Undirected Graph
  + Weighted / Unweighted Graph

### 3. Hash-Based Structures

Used for fast access using keys.

* Hash Table
* Hash Map
* Hash Set

### 4. Specialized or Abstract Data Types (ADTs)

Not tied to a specific implementation but describe behavior.

* List
* Map / Dictionary
* Set
* Priority Queue
* Deque (Double-Ended Queue**)**

**Task -03:**

**What all operations can we do in Data structures?**

### 1. Insertion

* Add a new element to the data structure.
* Example: Add a number to an array or a node to a linked list.

### 2. Deletion

* Remove an element from the data structure.
* Example: Remove the top element from a stack or delete a node from a tree.

### 3. Traversal

* Visit each element in the structure (one by one).
* Example: Going through all items in a list or tree.

### 4. Searching

* Find a specific element in the structure.
* Example: Look for a value in an array or search a key in a hash table.

### 5. Sorting

* Arrange elements in a specific order (ascending/descending).
* Example: Sorting a list of numbers.

### 6. Updating (Modification)

* Change the value of an existing element.
* Example: Change a score in an array from 60 to 80.

### 7. Merging

* Combine two data structures into one.
* Example: Merging two sorted arrays into a single sorted array.

### 8. Splitting

* Divide a data structure into parts.
* Example: Split a linked list into two halves.

### 9. Access

* Get the value of an element at a specific position (common in arrays).
* Example: Access the 3rd element of a list.

### 10. Searching by Key/Value

* Mostly used in hash tables or maps.
* Example: Get the phone number by entering a name**.**

**Task - 04:**

**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 creation | Can grow or shrink at runtime |
| Performance | Fast for accessing elements | Slightly slower during resizing (copying data) |
| Memory | Allocated at once; may waste unused space | Allocated as needed; more memory efficient |
| Flexibility | Less flexible (size can't change) | More flexible (can adjust to data growth) |
| Limitations | Can’t resize; must know size in advance | Resize may be costly; overhead in memory mgmt. |
| Examples | C-style arrays (int arr[10]) | Python lists, Java ArrayList, C++ vector |

**Task 05:**

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

* 'a' in ASCII = 97 (decimal)
* 97 in binary = 01100001

| Bit Position (from left) | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Power of 2 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| Value (for 97) | 0 | 1 | 1 | 0 | 0 | 0 | 0 |  |

**Task - 06:**

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

## Main Types of Computer Memory

Computer memory is broadly divided into two types:

## 1. Primary Memory (Main Memory)

Used directly by the CPU to store data temporarily while programs are running.

| Type | Description | Examples |
| --- | --- | --- |
| RAM | (Random Access Memory) – temporary, fast | DDR4, DDR5 |
| ROM | (Read Only Memory) – permanent, read-only | BIOS, firmware |
| Cache | Very fast memory near CPU | L1, L2, L3 cache |
| Registers | Tiny memory inside CPU for immediate use | Instruction register, accumulator |

## 2. Secondary Memory (Storage)

Used for long-term storage of data and programs.

| Type | Description | Examples |
| --- | --- | --- |
| HDD | Hard Disk Drive – slower, large storage | 1TB internal hard drives |
| SSD | Solid State Drive – faster, no moving parts | NVMe SSD, SATA SSD |
| Optical Discs | Stores data using laser tech | CD, DVD, Blu-ray |
| Flash Drives | Portable memory | Pen drives, USB flash drives |
| Memory Cards | Small external storage | SD cards, microSD |

## 3. Tertiary Memory (Backup & Archival)

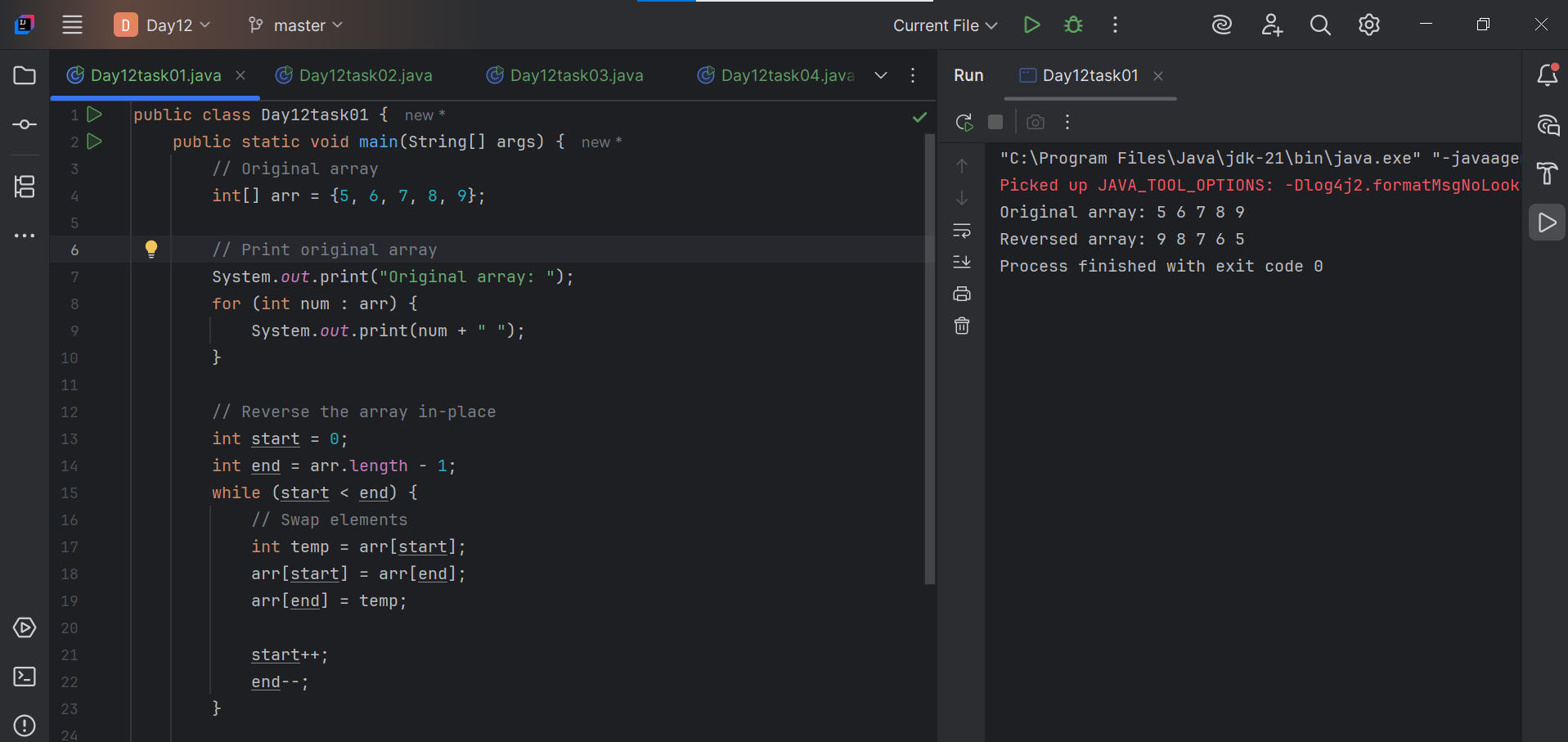
Used for large-scale backup or rare access.

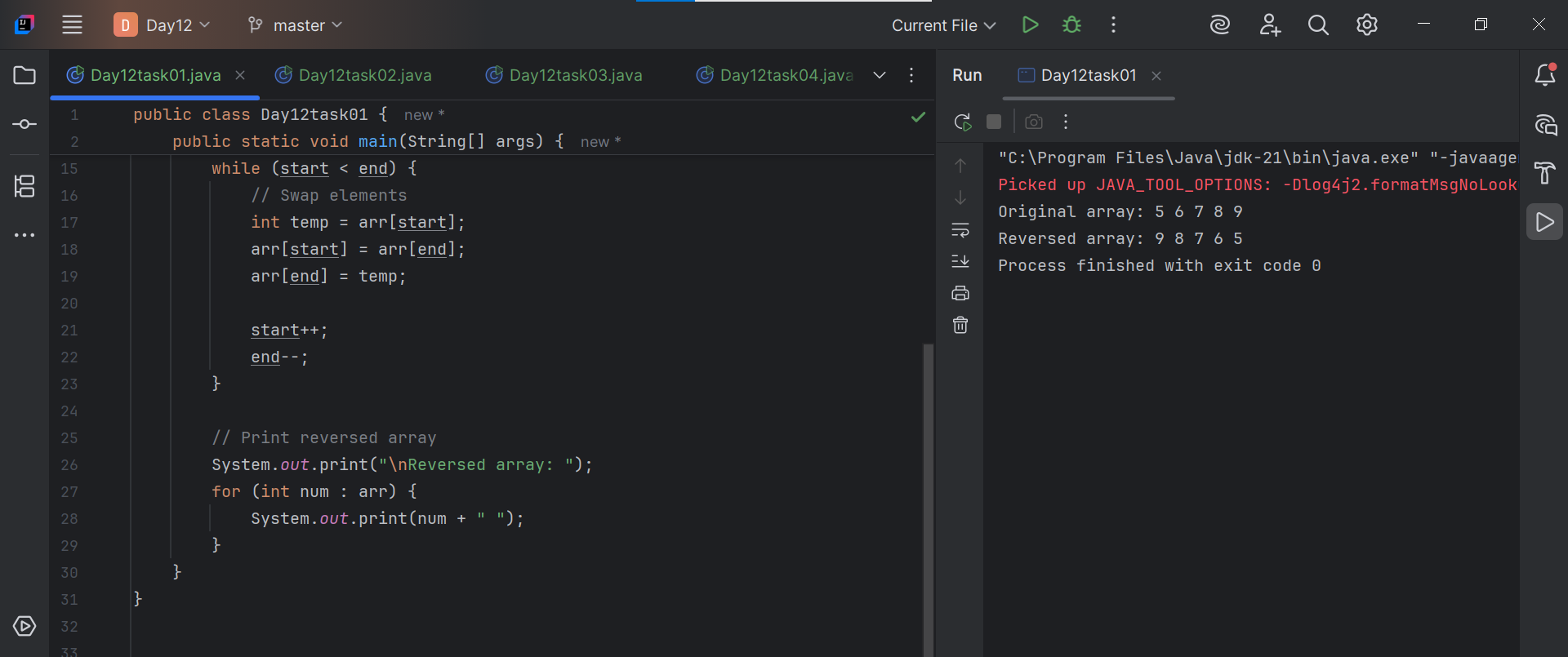
| Type | Description | Examples |
| --- | --- | --- |
| Magnetic Tape | Sequential storage for backups | Used in data centers |
| Cloud Storage | Internet-based remote storage | Google Drive, Dropbox |

## 4. Virtual Memory

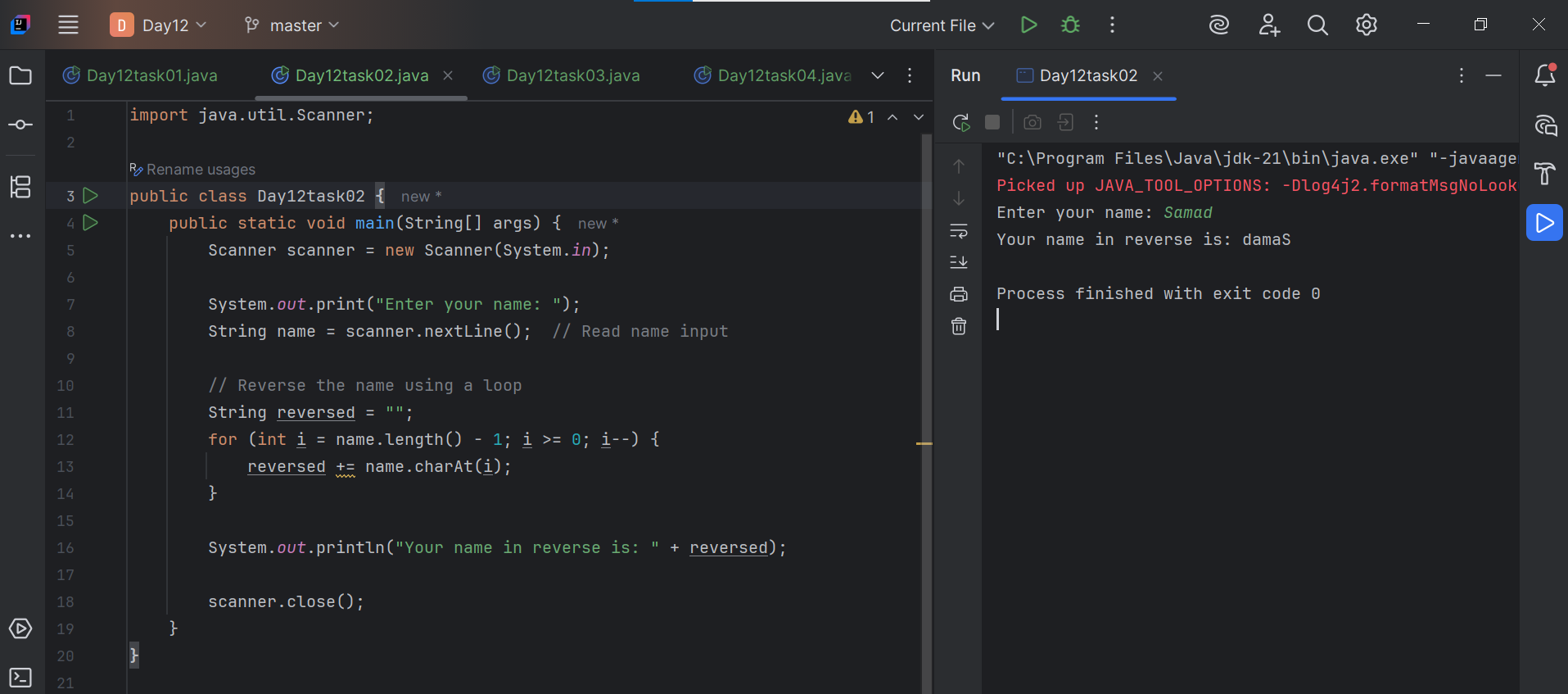
* Space on the hard disk used when RAM is full.
* Slower than RAM, but helps run large programs.

**Task - 07:**

****

****

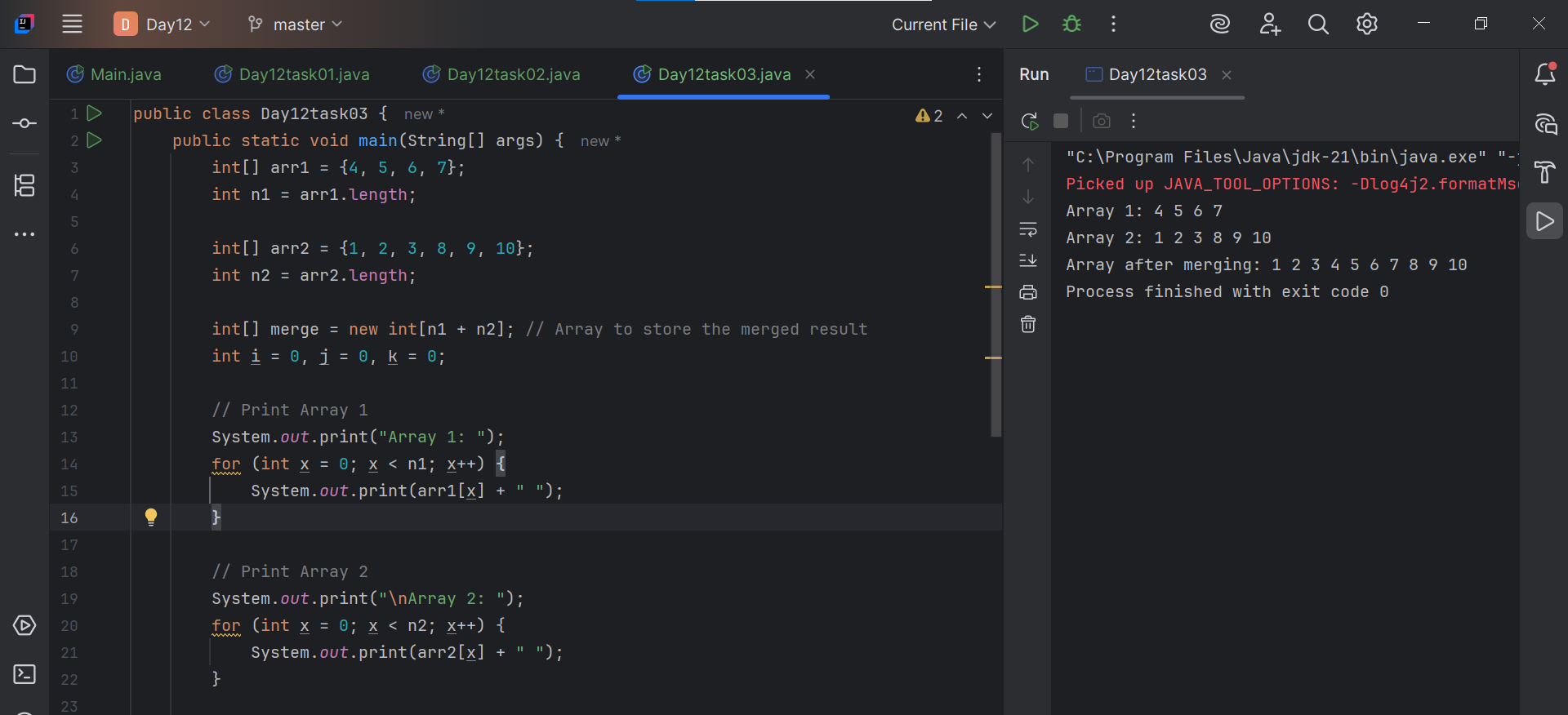
**Task - 08:**

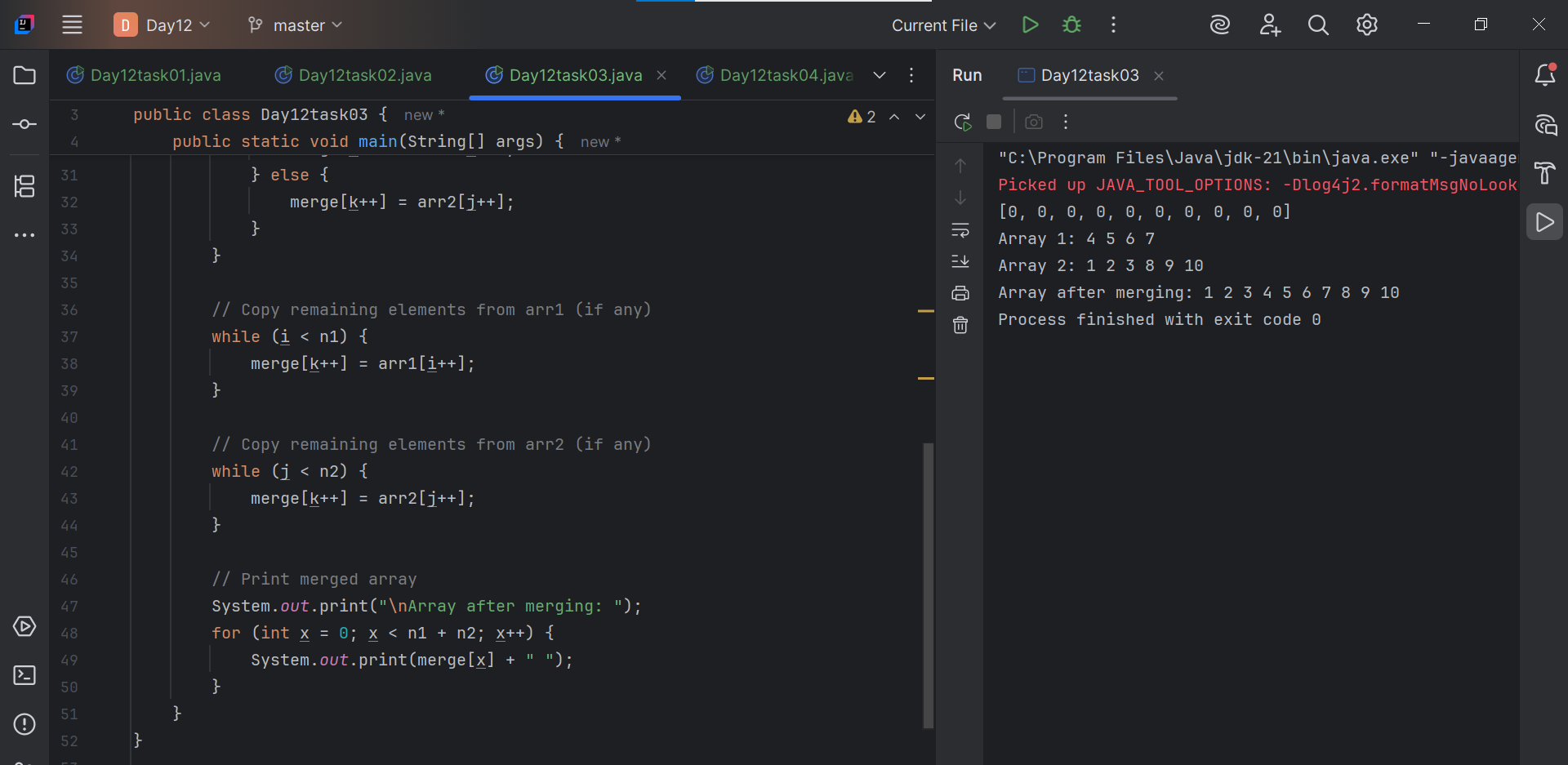
****

**Task - 09:**

Yes I have both Leetcode and Hackerrank … account

**Task -10:**

****

****

**Task - 11:**

**What do you understand by a Hash table?**

A hash table is a data structure that stores data in key-value pairs.  
 It's like a smart locker where you store and find things very fast using a unique key.

Example:

Think of a school register:

* You search a student by their ID number (key).
* You quickly get their name, marks, or info (value).

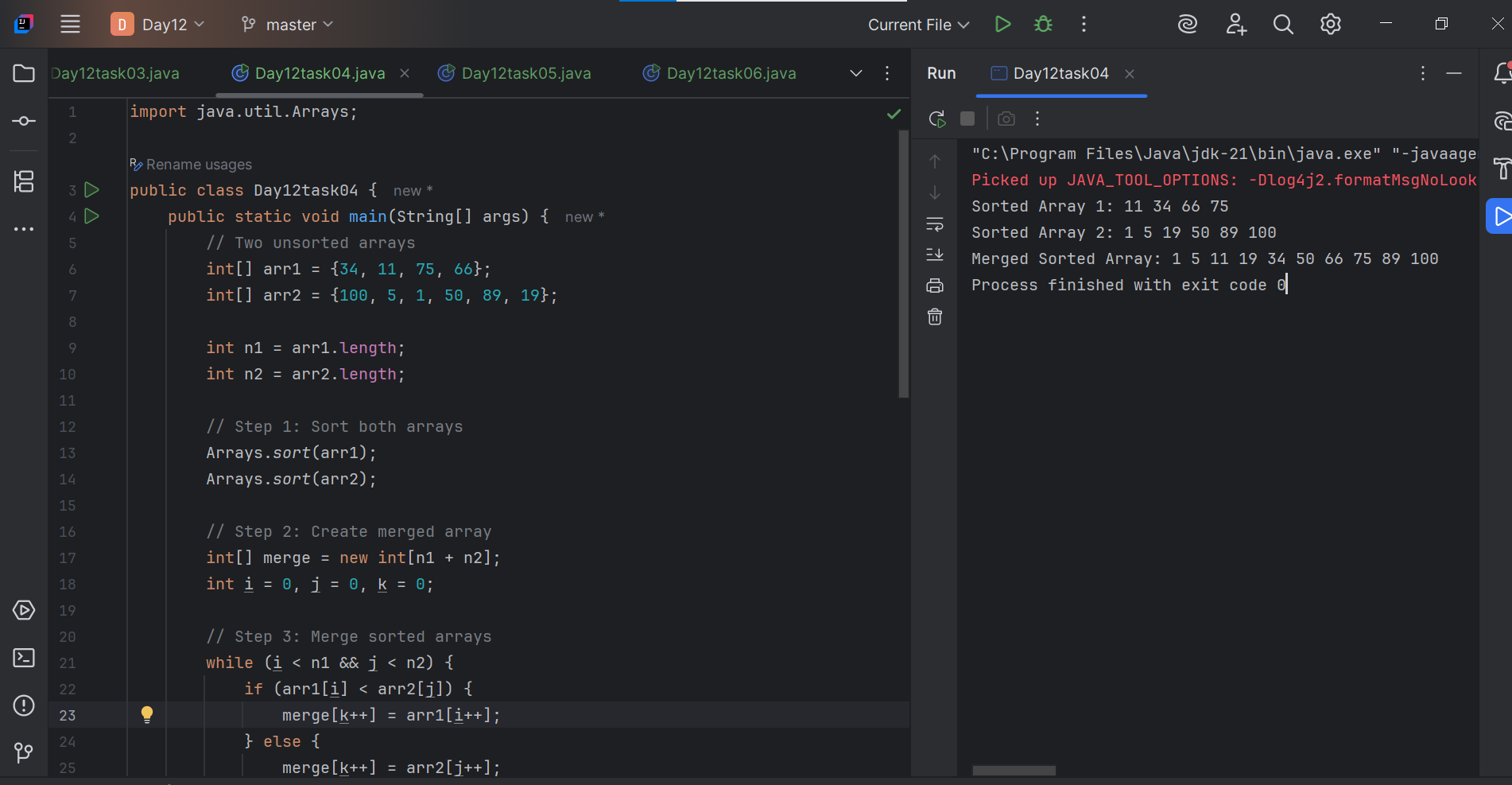
The ID number is used to find the right spot directly—just like a hash table.

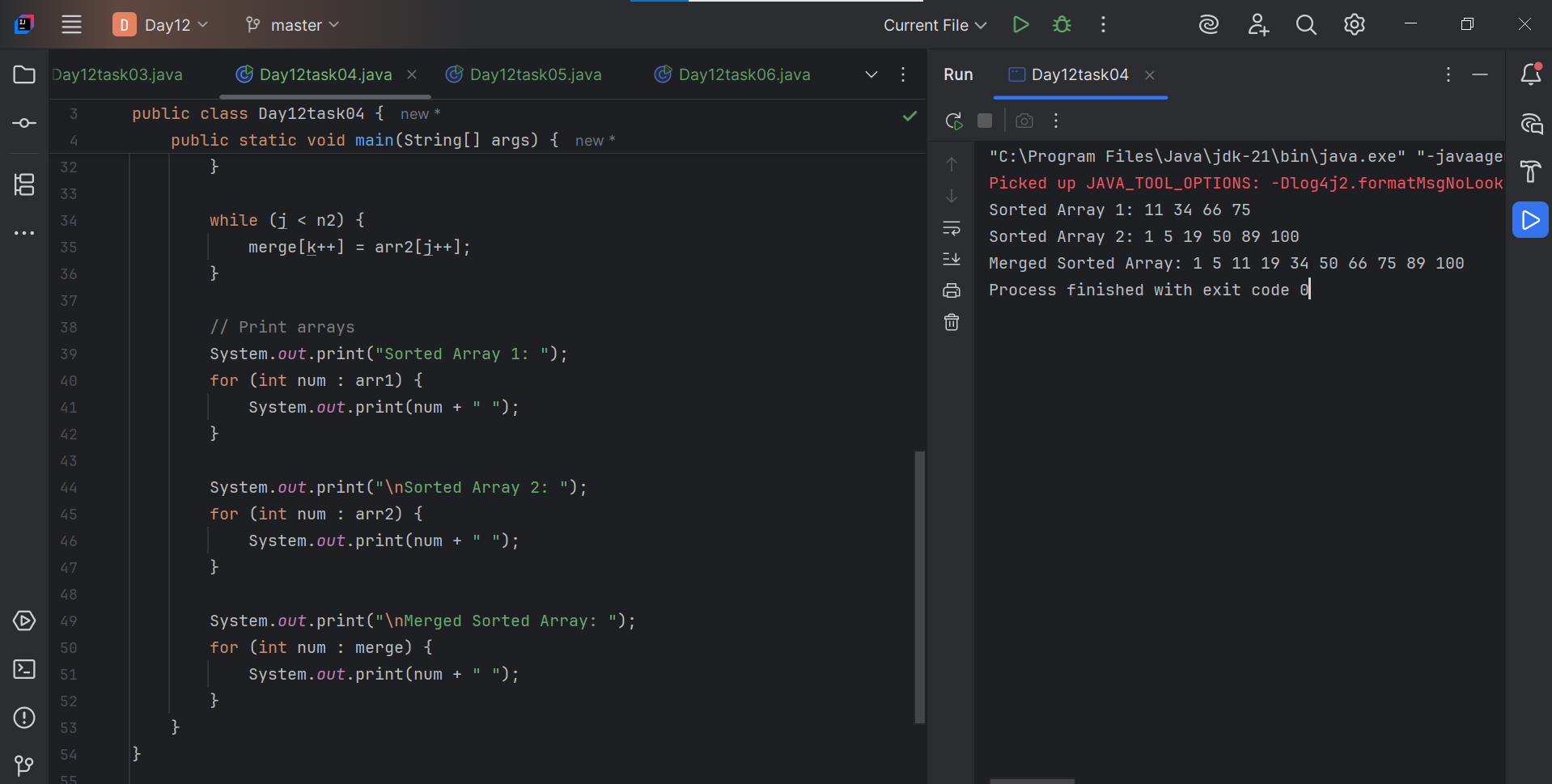
How It Works:

1. You give a key (like a name or ID).
2. A hash function converts the key into an index (a number).
3. The value is stored at that index in an array.

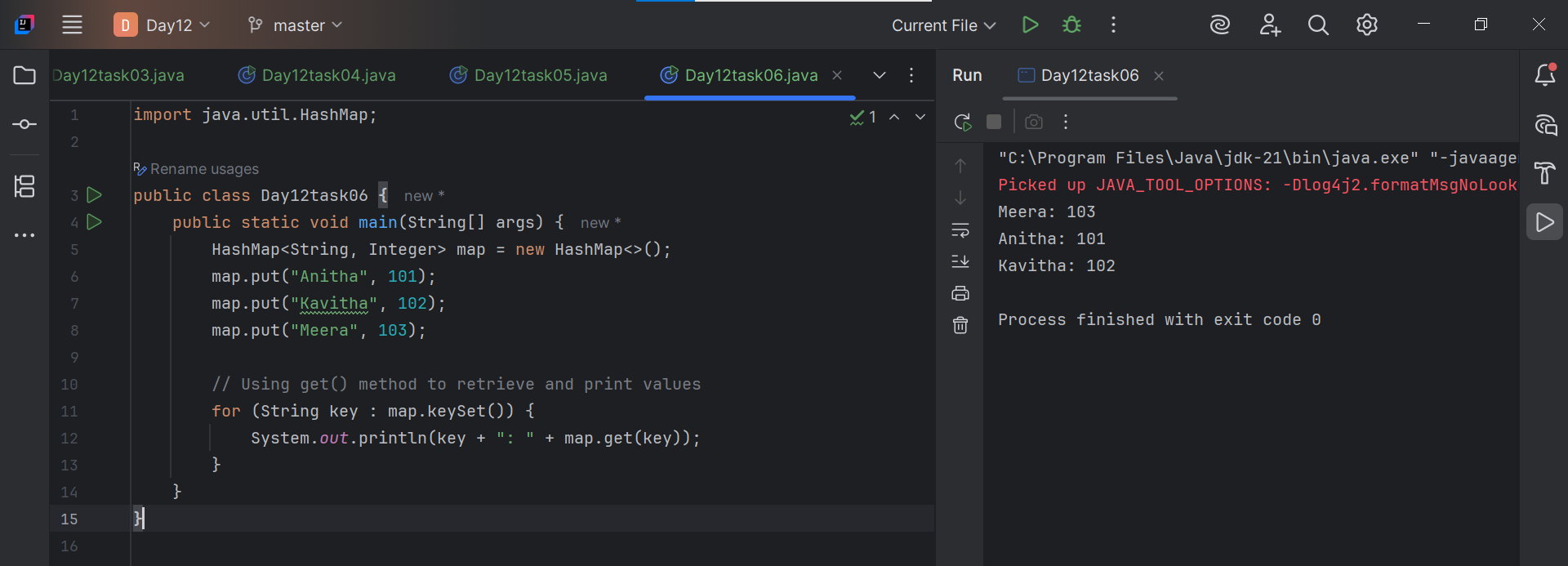
**Task - 12:**

****

****

****

**Task - 13:**

****

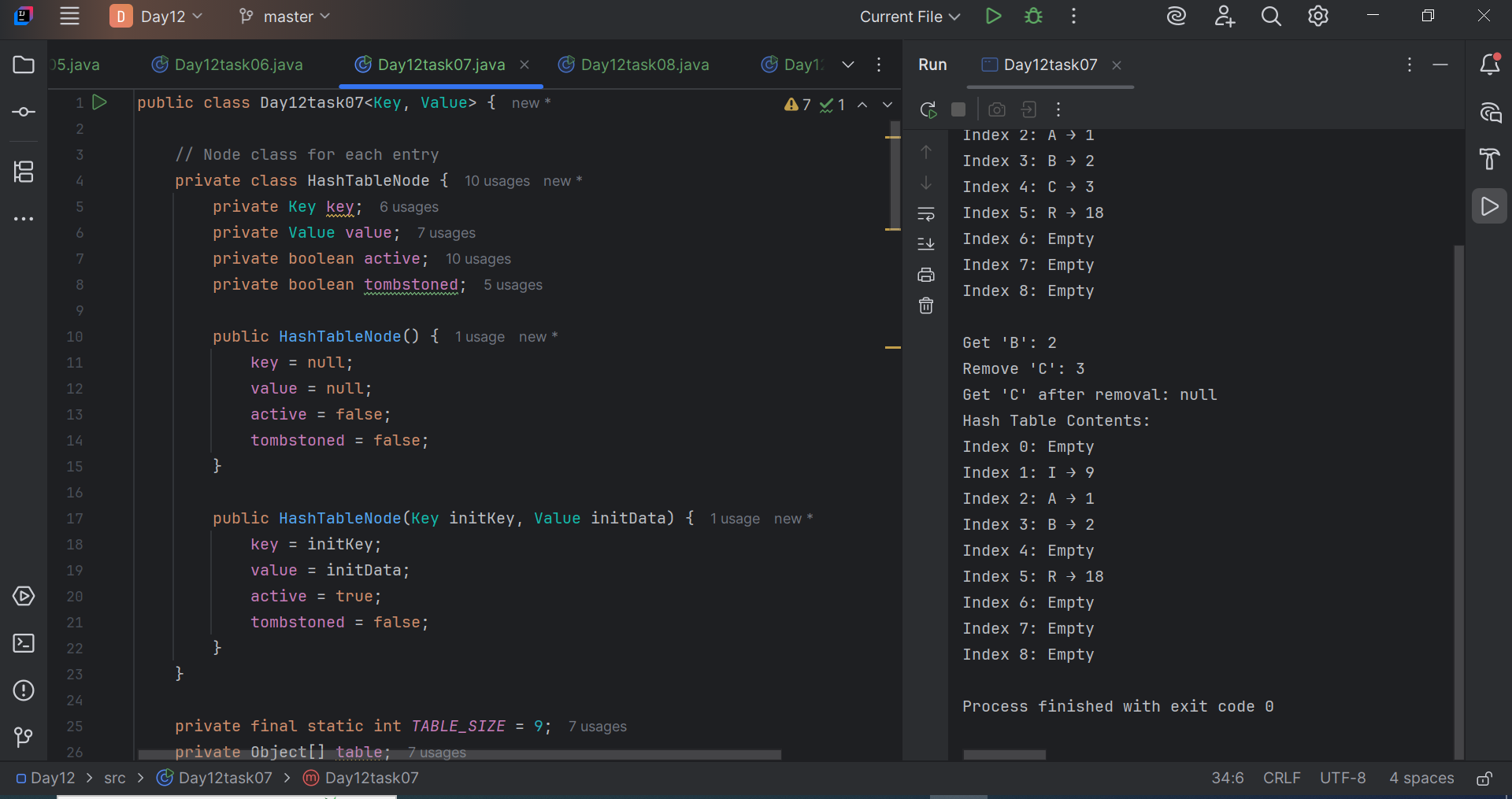
**Task -14:**

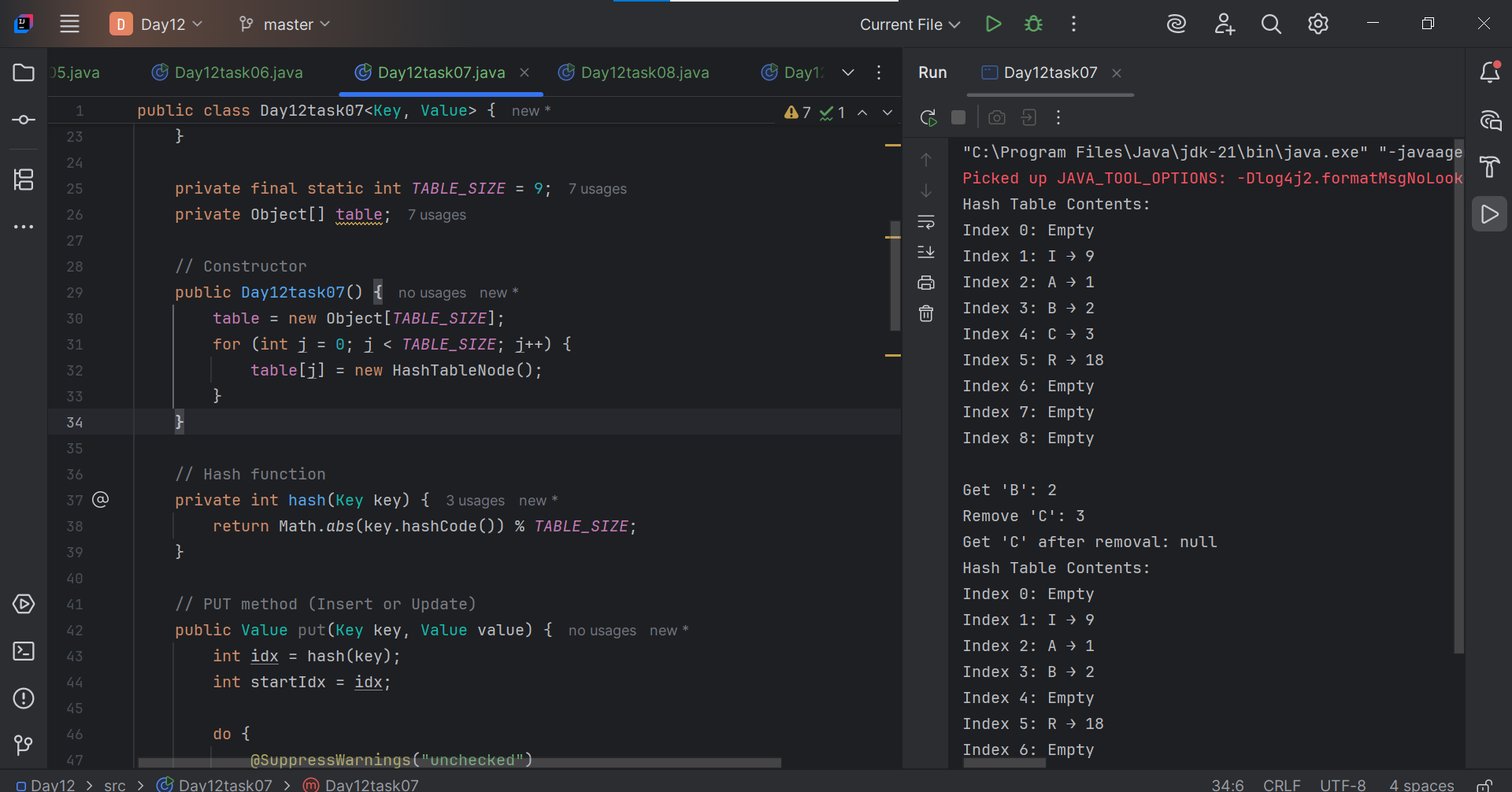
**Difference between Hash Table and Hash Map :**

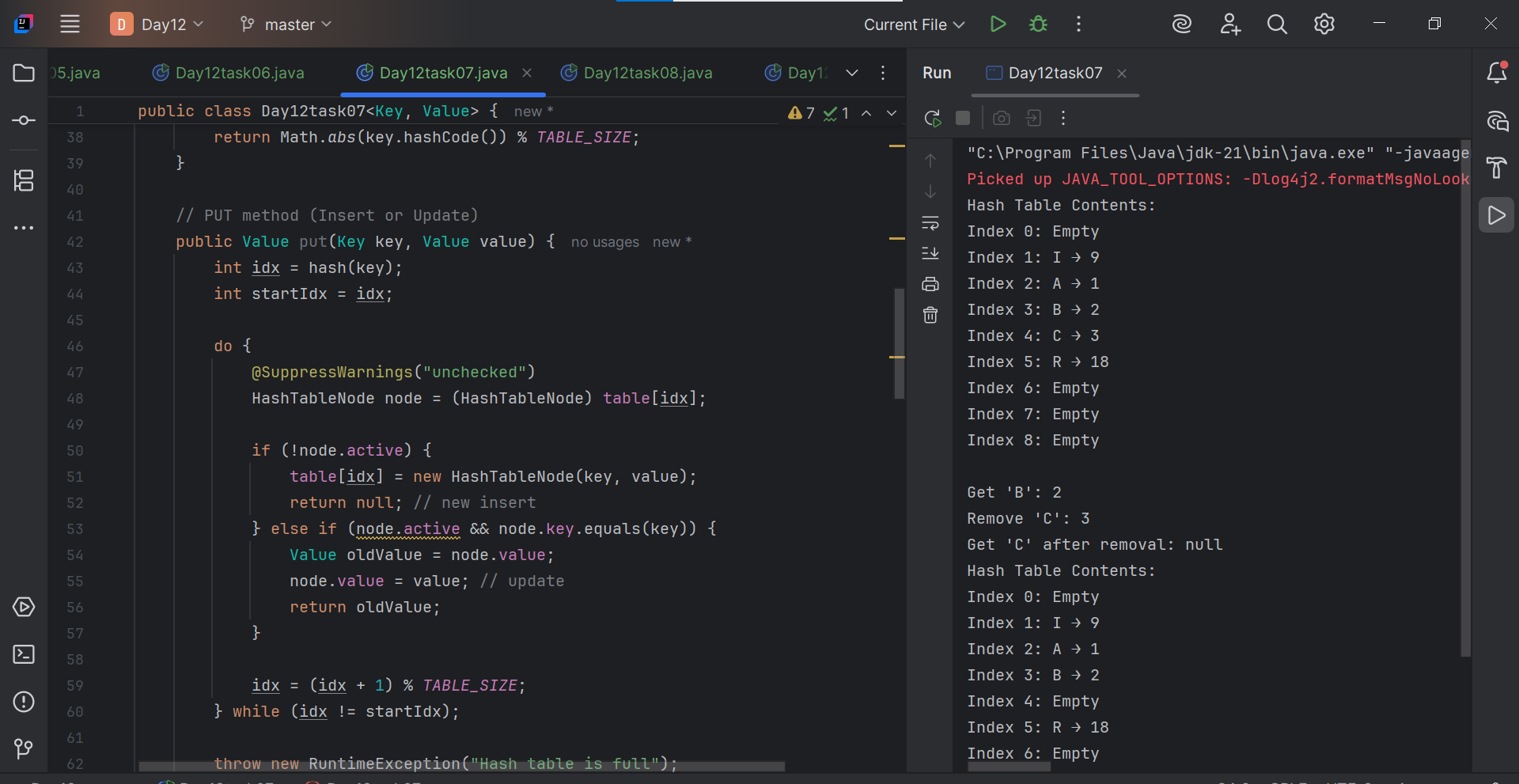
### HashMap vs HashTable – Simple Explanation

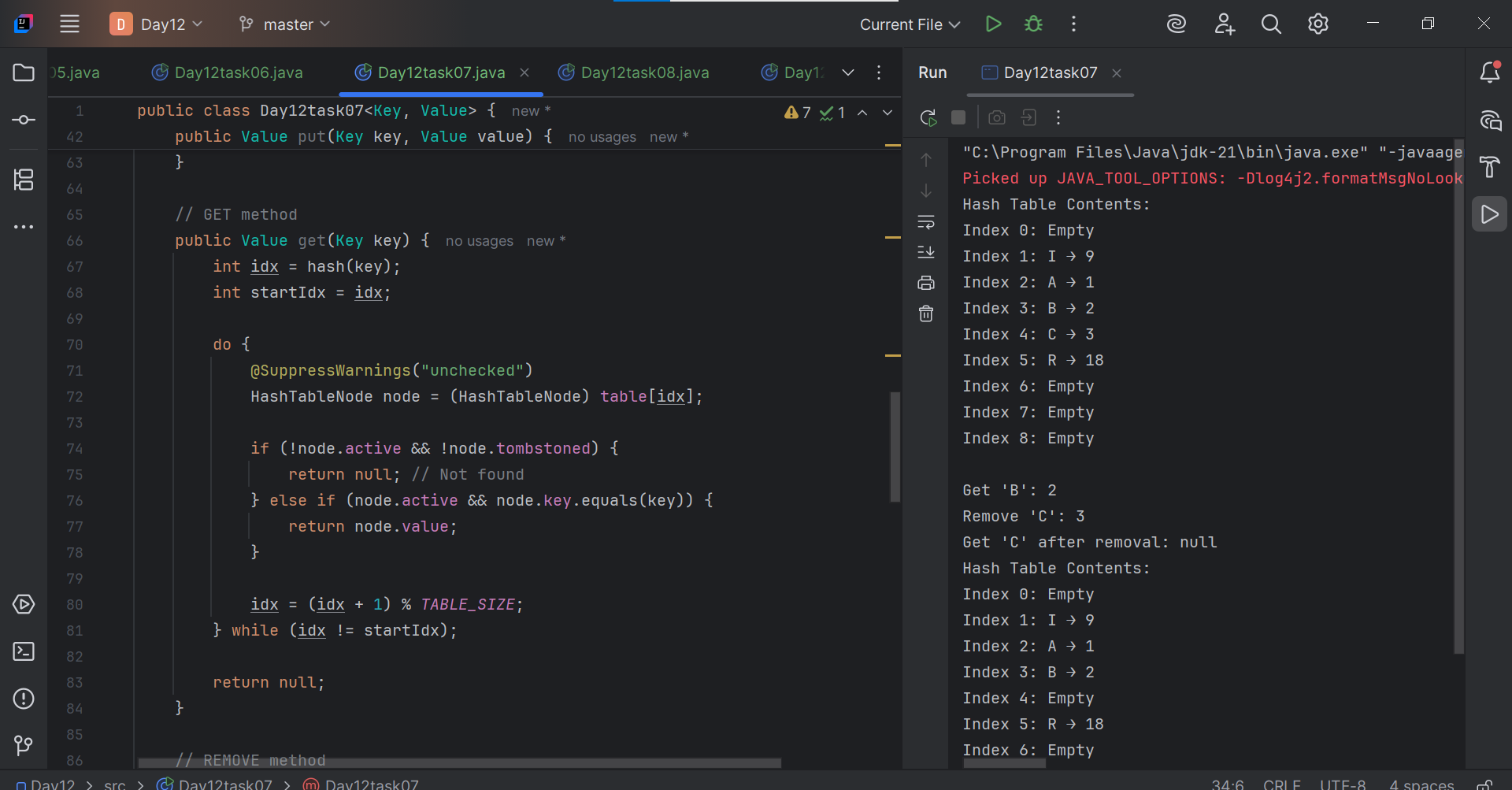
| Feature | HashMap | Hashtable |
| --- | --- | --- |
| Introduced In | Java 1.2 (Newer, part of Collections) | Java 1.0 (Older, legacy) |
| Thread-Safe | No (not safe for multiple threads) | Yes (safe for multiple threads) |
| Performance | Faster (no locking) | Slower (because of synchronization) |
| Null Allowed? | Yes – allows 1 null key & many null values | No – doesn’t allow null keys or values |
| Preferred When | Working in single-threaded environment | Need automatic thread safety (old code) |
| Part of  Thread-Safety | Java Collections Framework  Not synchronized (not thread-safe) | Legacy class (not in Collections)  Synchronized (thread-safe) |

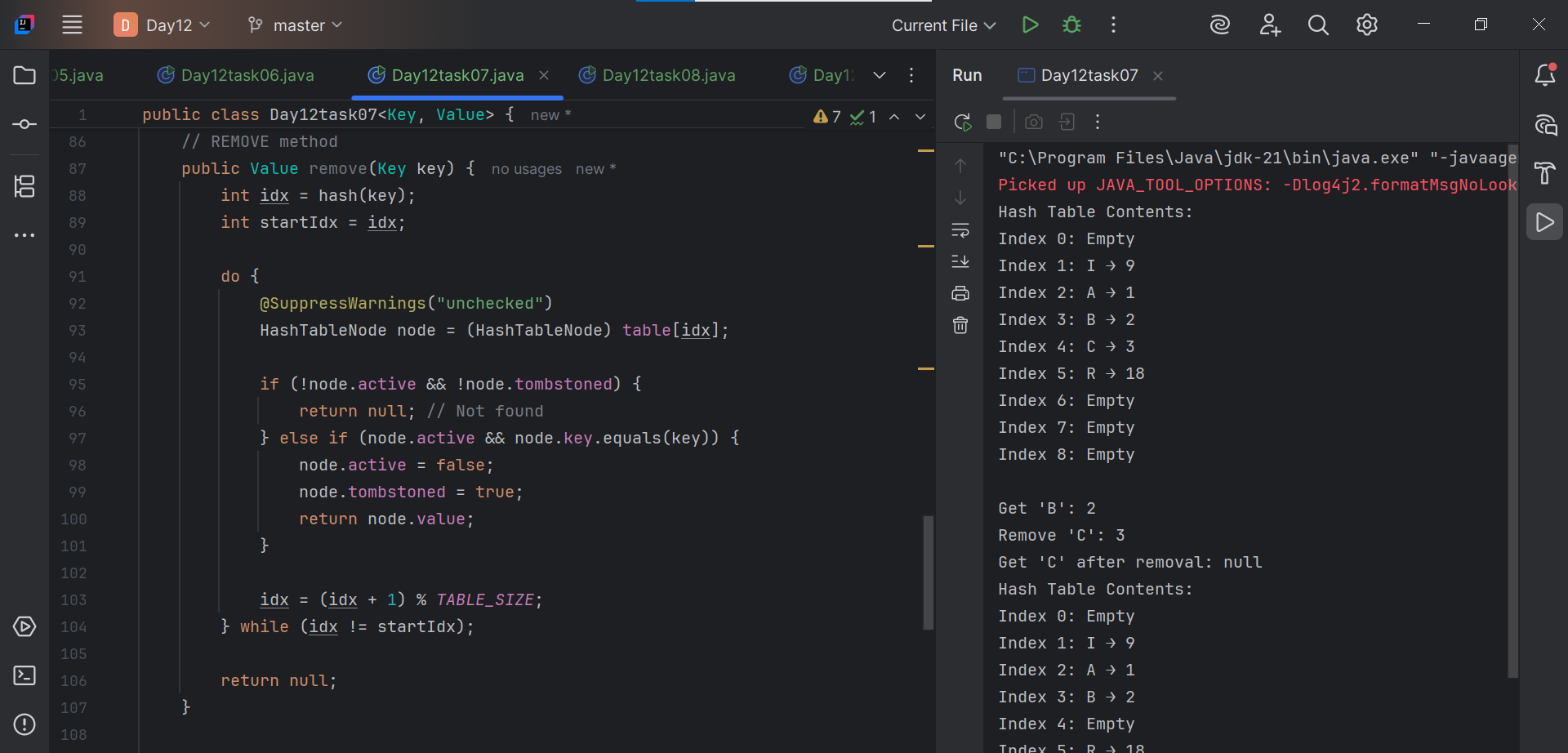
**Task - 15:**

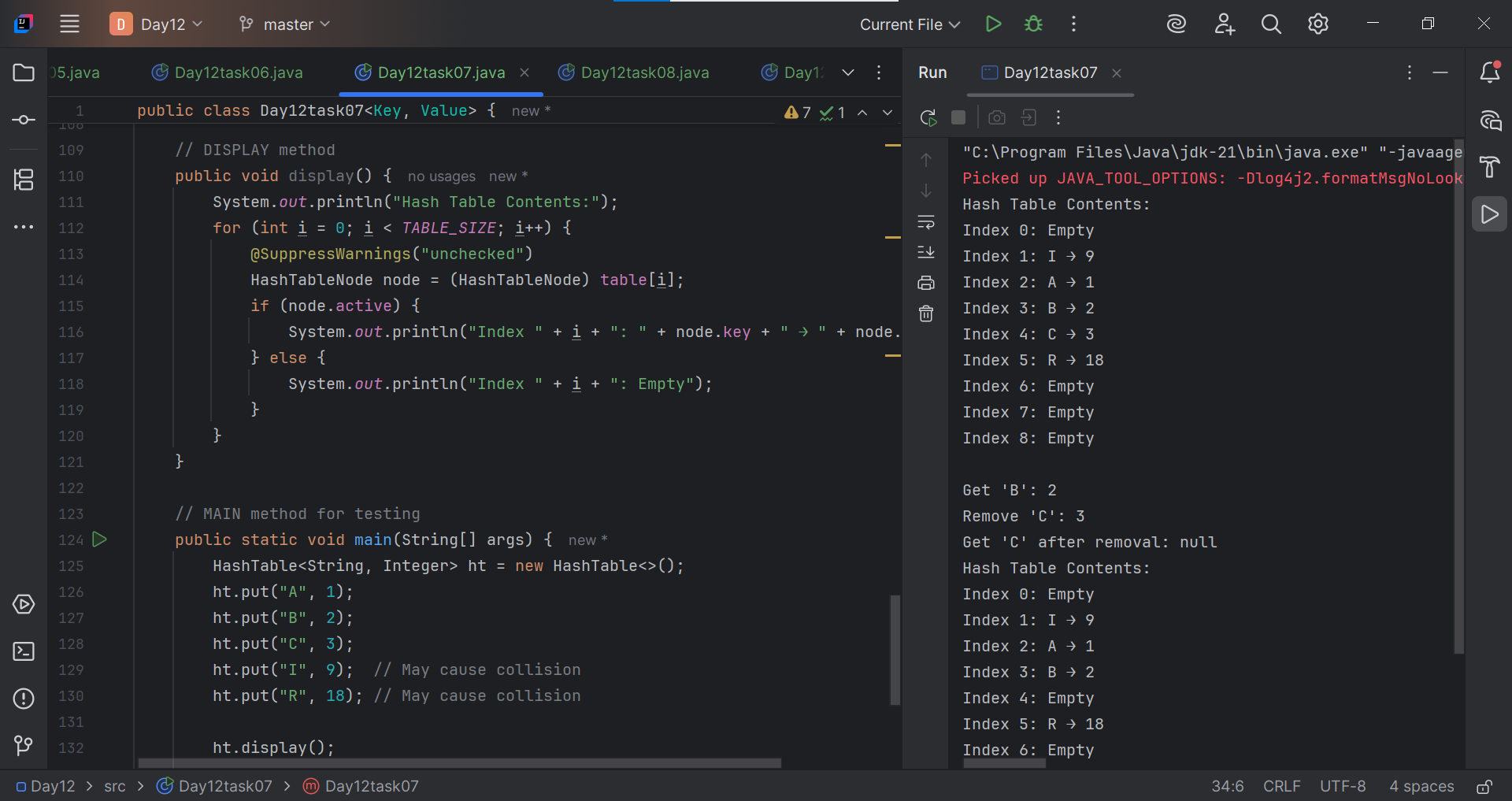
****

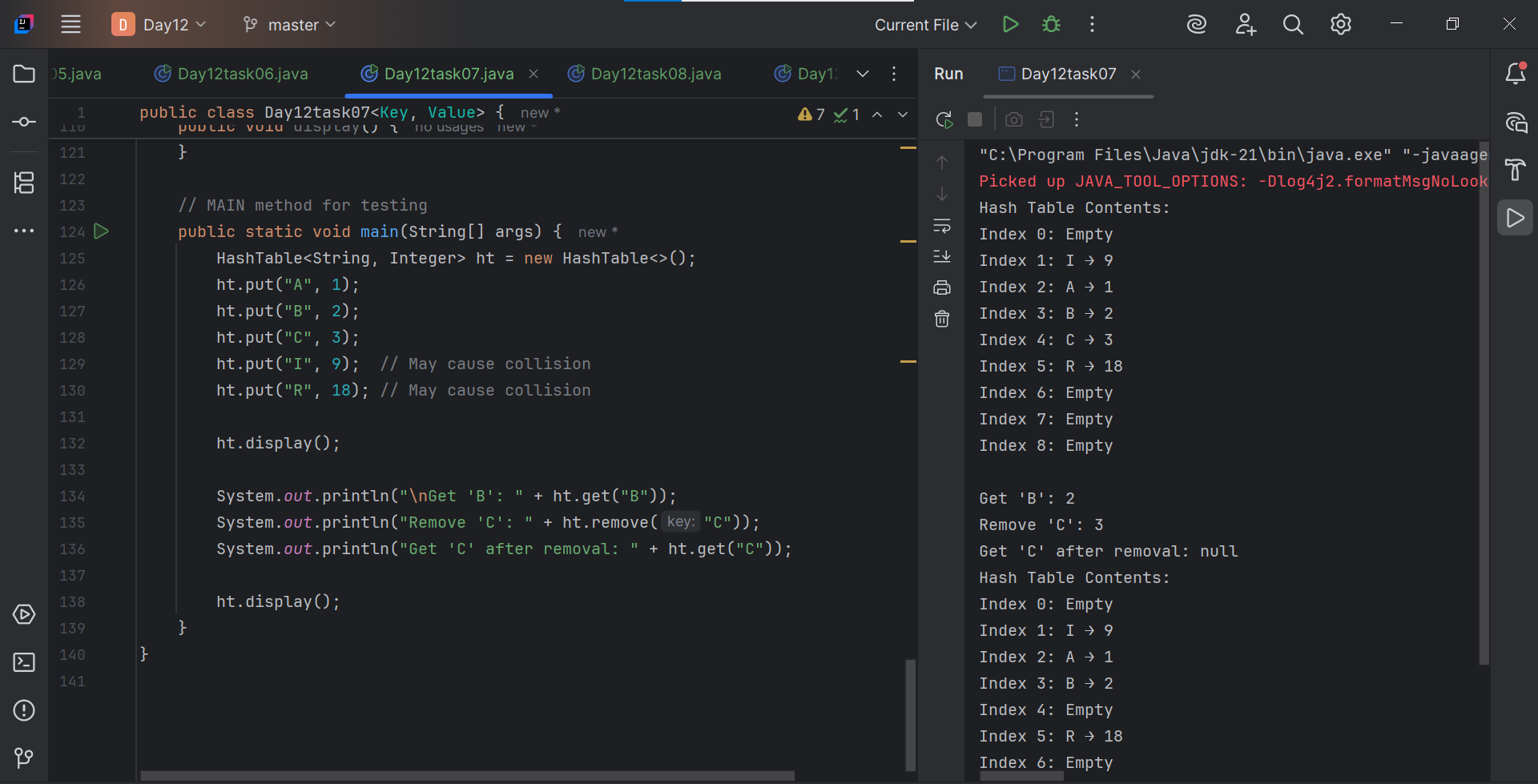
****

****

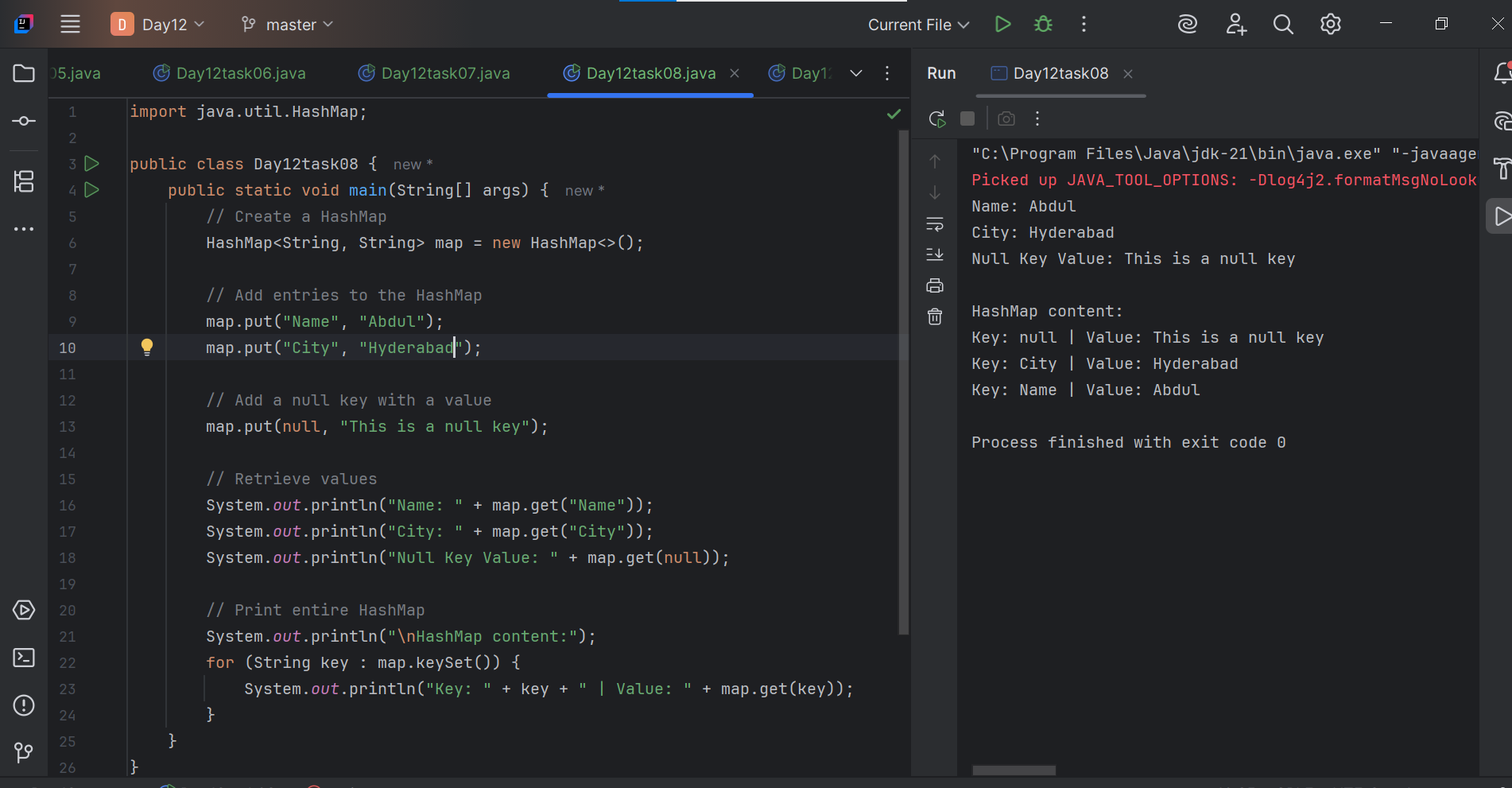
****

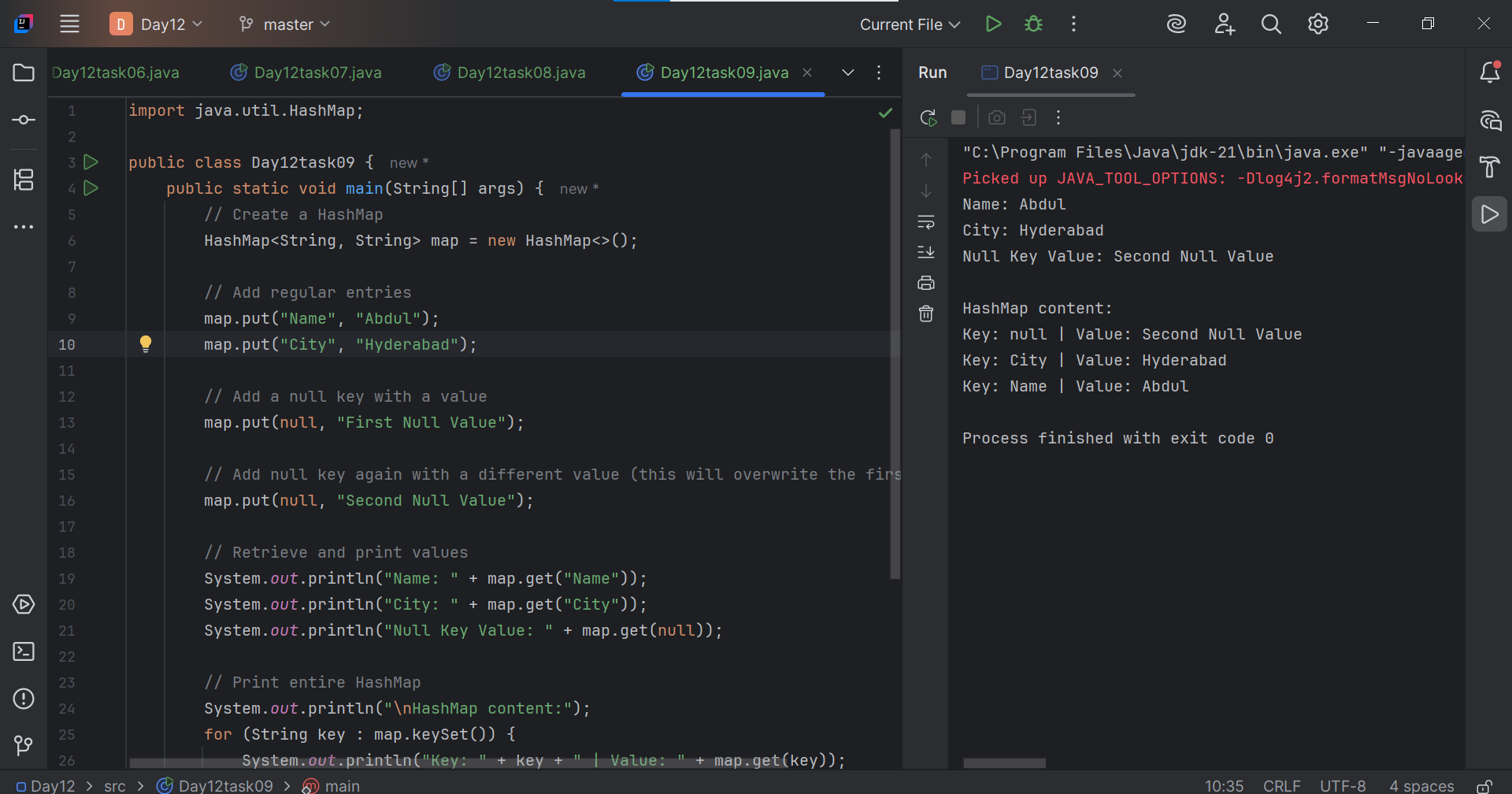
****

****

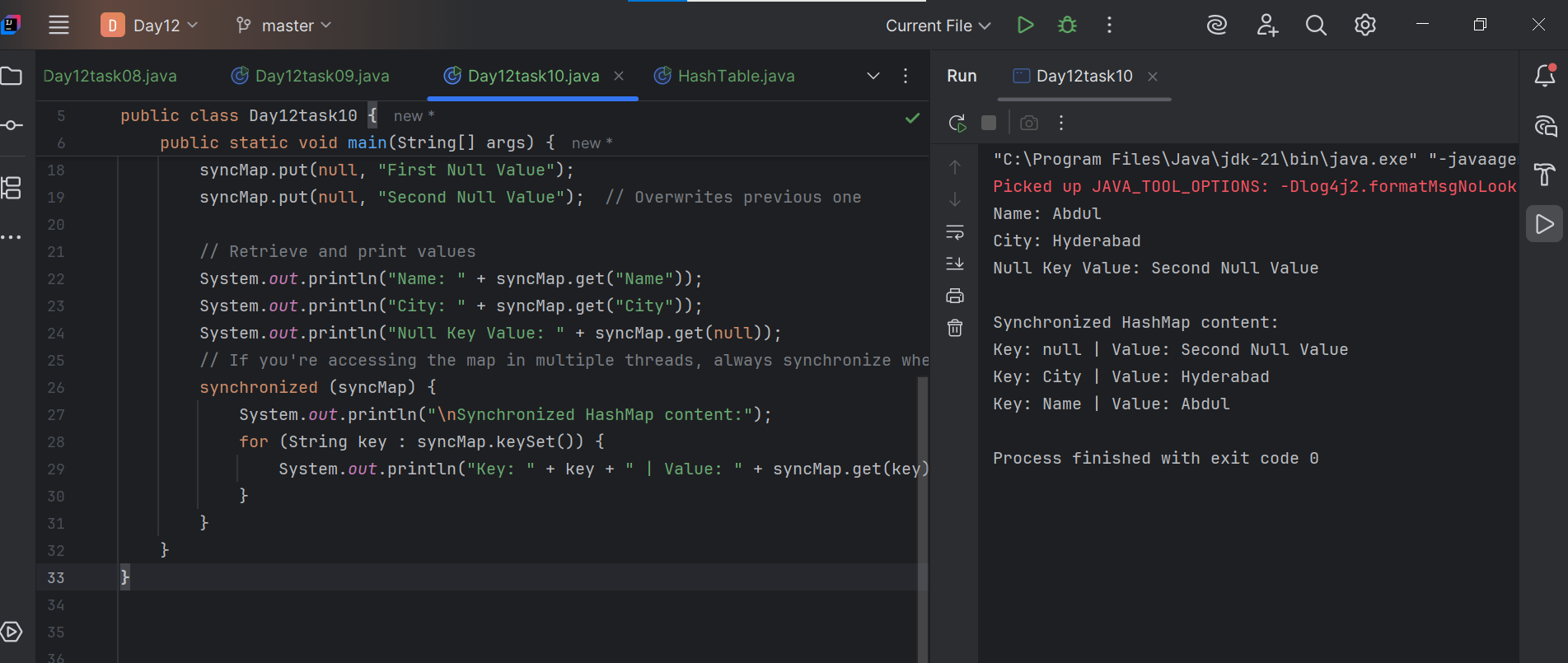
****

**Task - 16:**

****

****

****

****