Day 12 – 27/06/2025

Q1. What do you understand about data structures?  
Ans. Data structures are ways to organize and store data in a computer so that it can be efficiently accessed, modified, and manipulated. They provide a way to manage large amounts of data, enabling efficient data retrieval, insertion, deletion, and manipulation.

Q2. What are the types of data structures you know?  
Ans.

1. Arrays
2. Linked lists
3. Stacks
4. Queues
5. Hash maps
6. Trees
7. Graphs

Q3. What all operations can we do in Data structures?  
Ans.

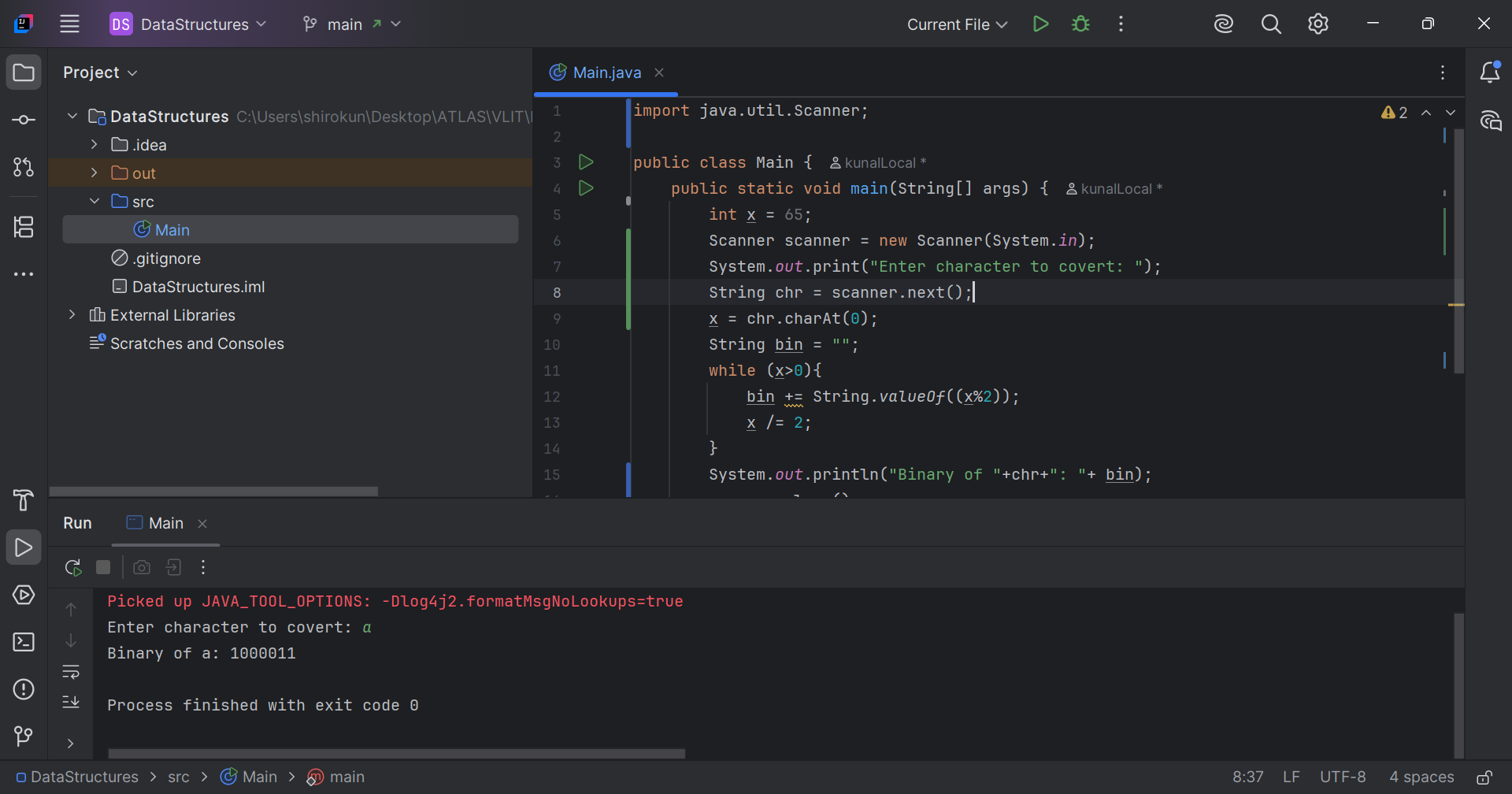
1. Insertion
2. Deletion
3. Traversal
4. Search
5. Update
6. Sorting
7. Merging

Q4. What are static and dynamic arrays? Explain or summarize key points in a table.  
Ans.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Size | Performance | Memory | Flexibility | Limitations |
| Static | Fixed size determined at compile-time. | Fast access and modification due to contiguous memory allocation. | Memory is allocated at compile-time and remains fixed. | Less flexible due to fixed size. | May lead to wasted memory if not fully utilized or may not be enough if more elements are added. |
| Dynamic | Size can be changed dynamically at runtime. | Performance may be affected by resizing and reallocation. | Memory is allocated and deallocated dynamically. | More flexible as size can be adjusted dynamically. | May incur performance overhead due to resizing and reallocation. |

Q5. What is the binary value of a?  
Ans. ASCII code of ‘a’ is 97.

import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 int x = 65;  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Enter character to covert: ");  
 String chr = scanner.next();  
 x = chr.charAt(0);  
 String bin = "";  
 while (x>0){  
 bin += String.*valueOf*((x%2));  
 x /= 2;  
 }  
 System.*out*.println("Binary of "+chr+": "+ bin);  
 scanner.close();  
 }  
}

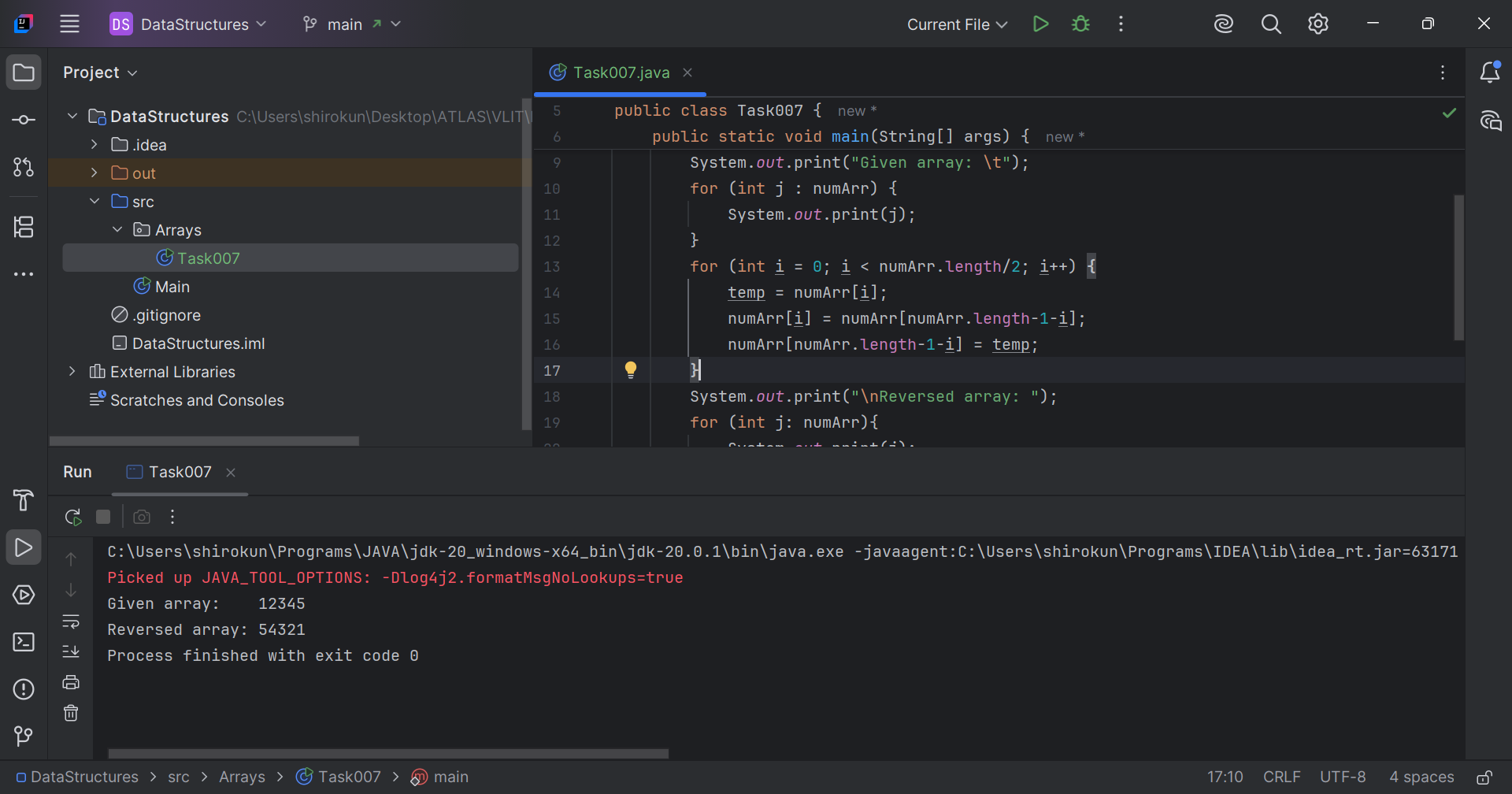


Q6. Explain different computer types.  
Ans.

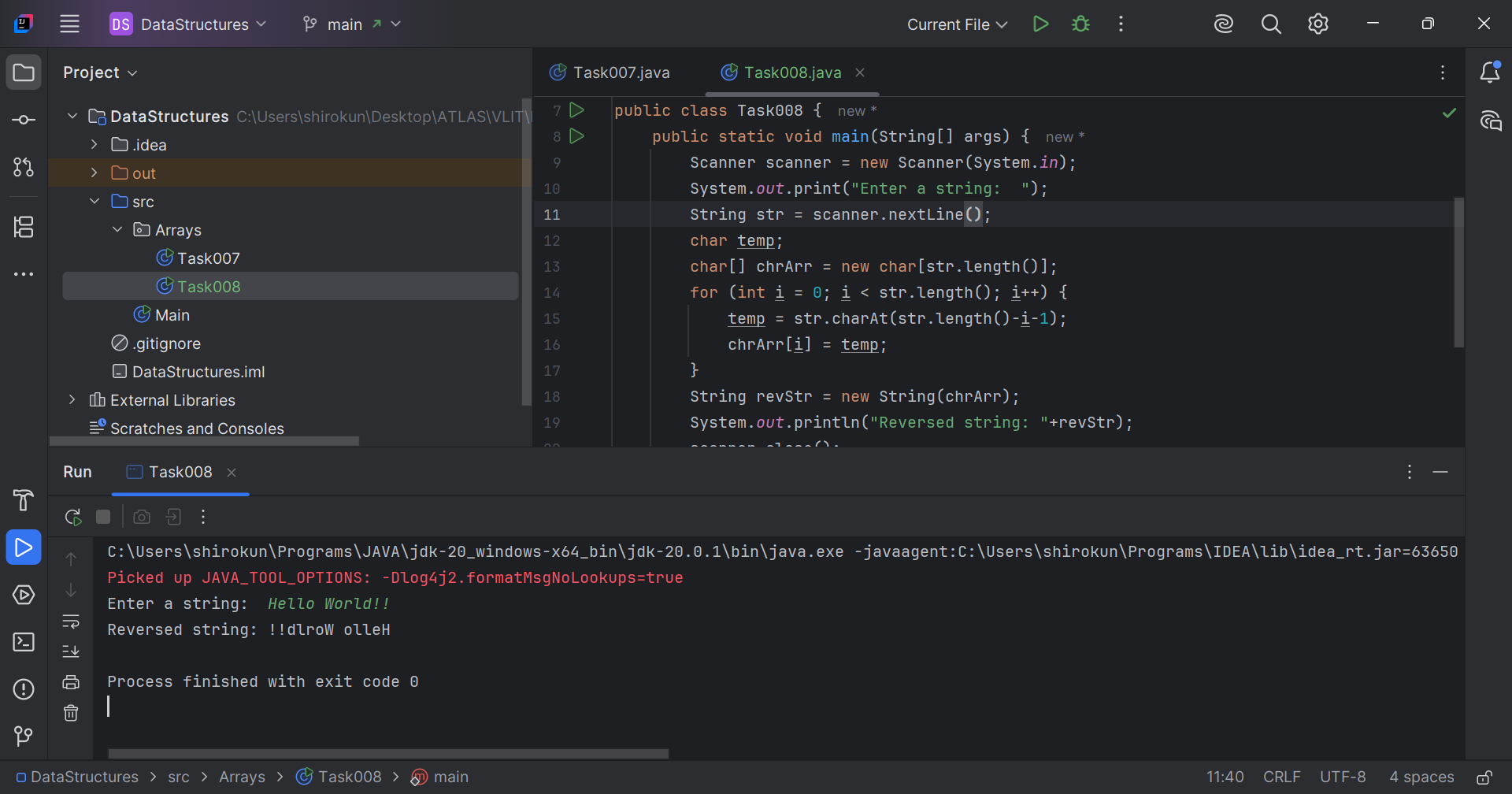
1. Primary memory
   1. RAM – Volatile memory that loses its contents when power is turned off.   
      Examples: DRAM, SRAM.
   2. ROM – Non-volatile memory that retains its contents even when power is turned off.  
      Examples: BIOS, Firmware.
2. Secondary memory
   1. HDD – Non-volatile storage device that uses magnetic disks to store data.
   2. SSD – Non-volatile storage device that uses flash memory to store data.
   3. Flash drive – Portable, non-volatile storage device that uses flash memory.
   4. CD/DVD – Optical storage media that use lasers to read and write data.
3. Cache memory
   1. Level 1 – small, fast cache built into the CPU.
   2. Level 2 – larger, slower cache that is often located on the CPU or on a separate chip.
   3. Level 3 – shared cache that is often used in multi-core processors.
4. Virtual memory
   1. Paging – technique that uses secondary storage to extend the amount of available RAM.
   2. Swapping – technique that moves entire processes between RAM and secondary storage.

// Task007: Reverse an array.  
  
package Arrays;  
  
public class Task007 {  
 public static void main(String[] args) {  
 int[] numArr = {1, 2, 3, 4, 5};

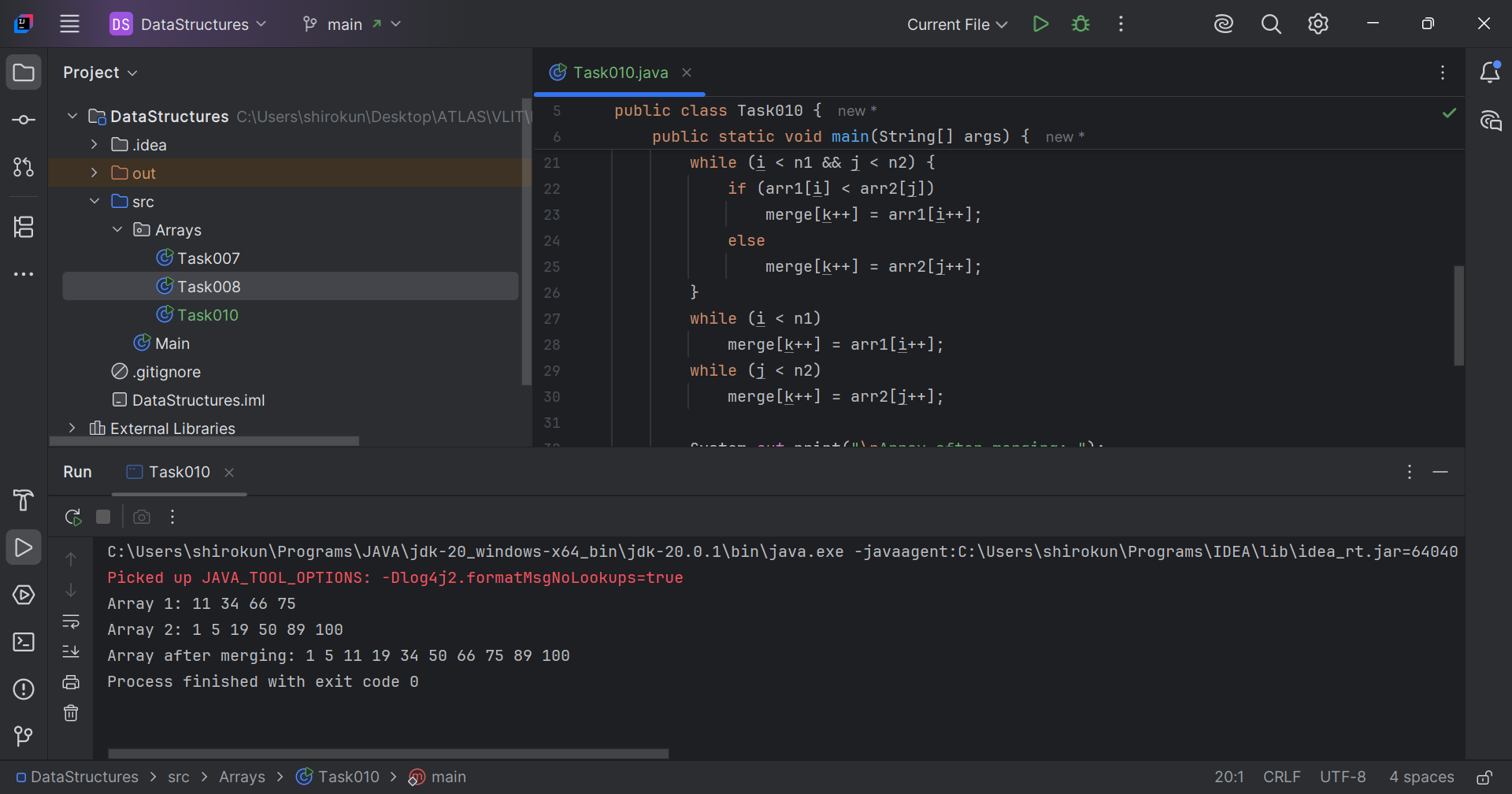
int temp;  
 System.*out*.print("Given array: \t");  
 for (int j : numArr) {  
 System.*out*.print(j);  
 }  
 for (int i = 0; i < numArr.length/2; i++) {  
 temp = numArr[i];  
 numArr[i] = numArr[numArr.length-1-i];  
 numArr[numArr.length-1-i] = temp;  
 }  
 System.*out*.print("\nReversed array: ");  
 for (int j: numArr){  
 System.*out*.print(j);  
 }  
 }  
}



// Task008: Reverse a string, input from user.  
  
package Arrays;  
  
import java.util.Scanner;  
  
public class Task008 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Enter a string: ");  
 String str = scanner.nextLine();  
 char temp;  
 char[] chrArr = new char[str.length()];  
 for (int i = 0; i < str.length(); i++) {  
 temp = str.charAt(str.length()-i-1);  
 chrArr[i] = temp;  
 }  
 String revStr = new String(chrArr);  
 System.*out*.println("Reversed string: "+revStr);  
 scanner.close();  
 }  
}

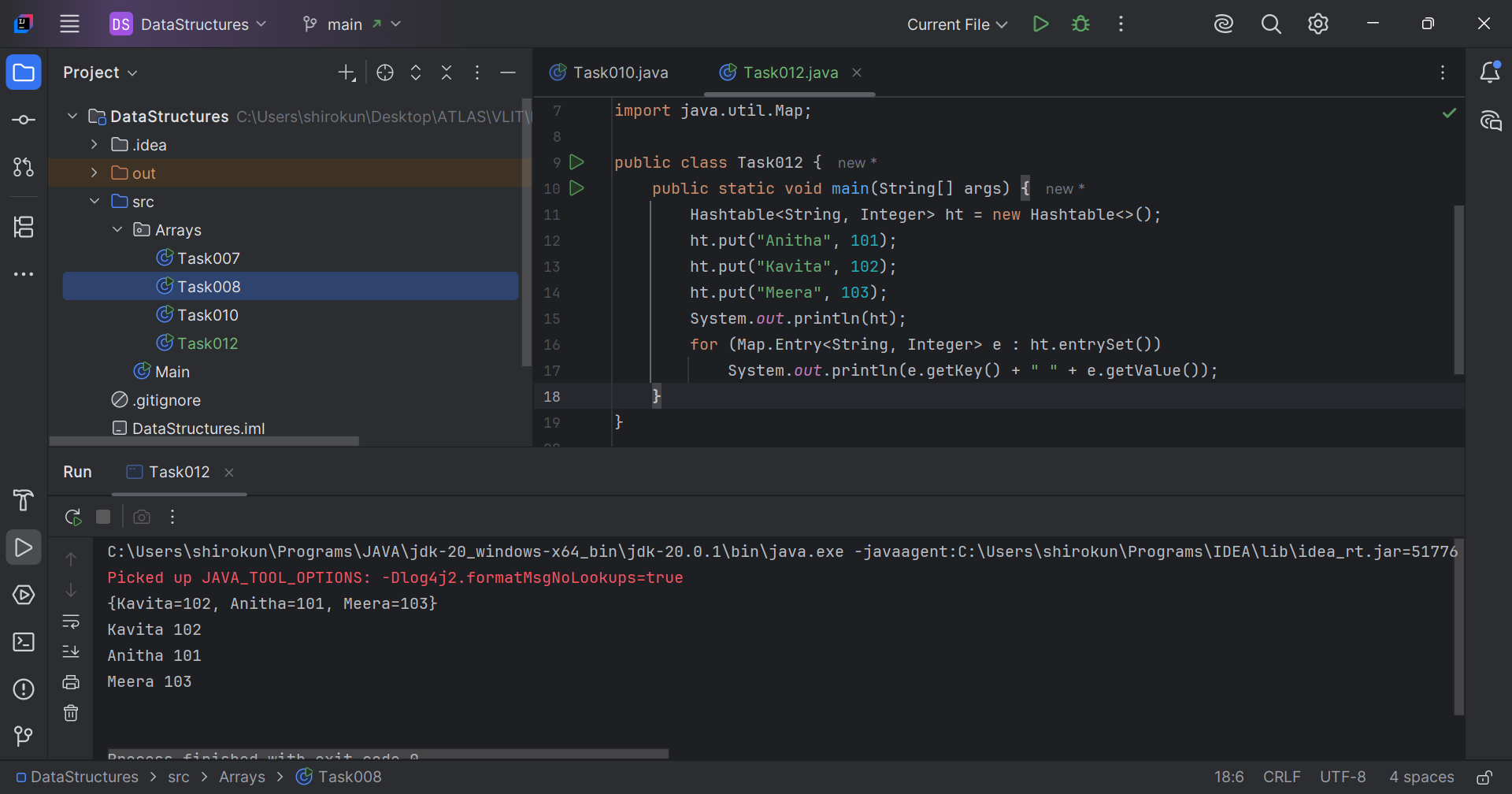


// Task010: Checkout the code. Merging two sorted arrays in sorted manner.  
  
package Arrays;  
  
public class Task010 {  
 public static void main(String[] args) {  
 int[] arr1 = {11, 34, 66, 75};  
 int n1 = arr1.length;  
 int[] arr2 = {1, 5, 19, 50, 89, 100};  
 int n2 = arr2.length;  
 int[] 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] + " ");  
 }  
}

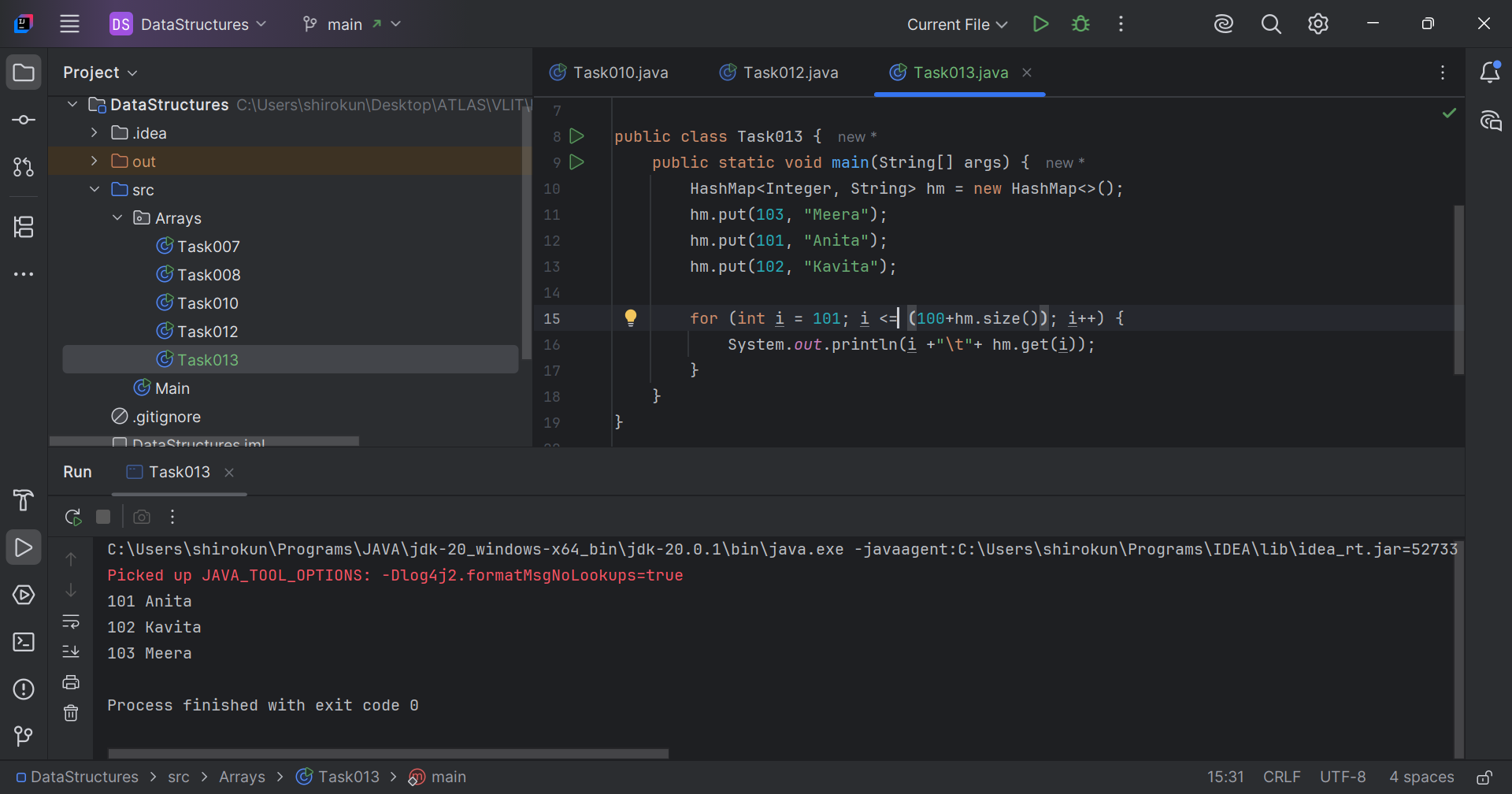


Q11. What do you understand by hash table?  
Ans. A hash table is a data structure that stores key-value pairs in a way that allows for efficient lookup, insertion, and deletion of elements.

// Task012: Understand the below Hash table code and  
// try to print values using get method of Hash table  
  
package Arrays;  
  
import java.util.Hashtable;  
import java.util.Map;  
  
public class Task012 {  
 public static void main(String[] args) {  
 Hashtable<String, Integer> ht = new Hashtable<>();  
 ht.put("Anitha", 101);  
 ht.put("Kavita", 102);  
 ht.put("Meera", 103);  
 System.*out*.println(ht);  
 for (Map.Entry<String, Integer> e : ht.entrySet())  
 System.*out*.println(e.getKey() + " " + e.getValue());  
 }  
}



// Task013: Understand the below Hash map code and  
// try to print values using get method of Hash map.  
  
package Arrays;  
  
import java.util.HashMap;  
  
public class Task013 {  
 public static void main(String[] args) {  
 HashMap<Integer, String> hm = new HashMap<>();  
 hm.put(103, "Meera");  
 hm.put(101, "Anita");  
 hm.put(102, "Kavita");  
  
 for (int i = 101; i <= (100+hm.size()); i++) {  
 System.*out*.println(i +"\t"+ hm.get(i));  
 }  
 }  
}



Q14. Difference between Hash Table and Hash Map.  
Ans.   
Hash Table:

* + Synchronized: Thread-safe, suitable for multithreaded environments.
  + Legacy class: Introduced in Java 1.0, less flexible.
  + No null keys or values: Does not allow null keys or values.
  + Fail-fast iterators: Throws ConcurrentModificationException if modified during iteration.

Hash Map:

* + Not synchronized: Not thread-safe, requires external synchronization.
  + More flexible: Allows null keys and values, provides more methods.
  + Better performance: Generally faster than Hash Table due to lack of synchronization overhead.
  + Fail-fast iterators: Also throws ConcurrentModificationException if modified during iteration.

// Task016: 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.  
  
package Arrays;  
  
import java.util.Collections;  
import java.util.HashMap;  
import java.util.Map;  
  
public class Task016 {  
 public static void main(String[] args) {  
 HashMap<Integer, String> hashMap = new HashMap<>();  
 hashMap.put(null, "Lok");  
 hashMap.put(102, "Jan");  
 hashMap.put(103, "People");  
 System.*out*.println(hashMap);  
 Map<Integer, String> syncMap = Collections.*synchronizedMap*(hashMap);  
 syncMap.put(null, "Human");  
 System.*out*.println(syncMap);  
 }  
}

