ARRAYS

1. Declaration

```
int[] arr1;      // Preferred

int arr2[];     // Valid but less preferred
```

Initialization

```
arr1 = new int[5];    // Default values (0 for int, null for objects)

int[] arr3 = {1, 2, 3, 4, 5};   // Inline initialization
```

2. Accessing Elements

```
int value = arr3[2]; // Access element at index 2 (value = 3)

arr3[2] = 10;       // Modify element at index 2
```

3. Length of Array

```
int length = arr3.length; // Length property (no parentheses)
```

4. Iterating Through Arrays

Using for loop:

```
     for (int i = 0; i < arr3.length; i++) {
System.out.println(arr3[i]); }
```

Using for-each loop:

```
    for (int value : arr3) {
System.out.println(value);  }
```

5. Common Array Operations

Sorting:

```
import java.util.Arrays;

Arrays.sort(arr3); // Ascending order
```

Binary Search (on sorted array):

```
int index = Arrays.binarySearch(arr3, 10); // Returns index or -1 if not found
```

Copying:

```
int[] newArr = Arrays.copyOf(arr3, arr3.length); // Full copy

int[] partialArr = Arrays.copyOfRange(arr3, 1, 4); // From index 1 to 3
```

Equality Check:

```
boolean isEqual = Arrays.equals(numbers, newArray);
```

6. Multidimensional Arrays

1. Declaration and Initialization:

```
int[][] matrix = new int[3][3]; // 3x3 matrix
int[][] predefinedMatrix = {   {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9}  };
```

2. Accessing Elements:

```
int value = matrix[1][2]; // Access element at second row, third column
```

3. Modifying Elements:

```
matrix[0][0] = 10; // Change element at first row, first column to 10
```

4. Traversing Multidimensional Arrays:

```
for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[i].length; j++) {
               System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }
```

ARRAYLIST

Part 1: Declaration, Initialization, Adding, and Accessing

1. Declaration and Initialization

```
ArrayList<String> languages = new ArrayList<>();

System.out.println("Initial List: " + languages);
```

2. Adding Elements

```
languages.add("Java");
```

//Adding at specific index

```
languages.add(1, "Kotlin");
```

3. Accessing Elements

```
languages.get(2);
```

Part 2: Updating, Removing, Size, and Searching

4. Updating Elements

```
languages.set(1, "JavaScript");
```

5. Removing Elements

```
languages.remove("C++"); // ELEMENT

languages.remove(2); // INDEX ELEMENT
```

6. Checking Size and Emptiness

```
languages.size();

languages.isEmpty();
```

7. Searching for Elements

```
languages.contains("Java");

languages.indexOf("Ruby");
```

Part 3: Iterating Through ArrayList

8. Iterating Through the ArrayList

1. for (int i = 0; i < languages.size(); i++) {

```
  System.out.println(languages.get(i));

}
```

2. Using Enhanced For Loop:");

```
for (String lang : languages) {

  System.out.println(lang);

}
```

3. Iterating Using Iterator:");

```
Iterator<String> iterator = languages.iterator();

while (iterator.hasNext()) {

  System.out.println(iterator.next());

}
```

Part 4: SORTING

```
Collections.sort(newList);

Collections.sort(newList, Collections.reverseOrder());
```

11. Clearing the ArrayList

```
newList.clear();
```

12. Working with Numbers (Example of Integer ArrayList)

```
numbers.remove(2); // Removes element at index 2

System.out.println("Iterating Over Numbers:");

for (int num : numbers) {

    System.out.println(num);

}
```

STRING

1. String Declaration and Initialization

```
String str1 = "Hello, World!";  // Using literal

String str2 = new String("Hello, Java!");  // Using new keyword
```

4. String Immutability

Strings in Java are **immutable**, meaning once a `String` object is created, its value cannot be changed.

```
String str = "Hello";

str = "World";  // Creates a new string object and reassigns it
```

The original `"Hello"` string is not modified. A new string `"World"` is created and assigned to the reference variable `str`.

5. StringBuilder (Mutable String)

If you need to modify a string frequently, it's better to use **StringBuilder** because it's mutable, unlike `String`.

```
StringBuilder sb = new StringBuilder("Hello");
sb.append(" World!");  // Modifies the string directly
System.out.println(sb);  // Output: Hello World!
```

`StringBuilder` is faster than using `String` for repeated modifications because it doesn't create a new object every time.

NOTE : String[] languages = str.split(",");

str.split("\\d+"); // on digit one or more

str.split("\\s+"); // spaces

2. Common String Methods

1. Length of the String

```
int length = str.length();
```

2. Concatenation of Strings

```
String result = str1 + str2;  // Concatenation using '+'
```

3. Accessing Characters

```
char ch = str.charAt(7);  // Accessing character at index 7
```

4. Substring

```
String str = "Hello, Java!";

String subStr = str.substring(7, 11);  // Extract substring from index 7 to 10

System.out.println(subStr);  // Output: Java
```

5. Changing Case

```
String str = "Hello, World!";

String upper = str.toUpperCase();  // Converts to uppercase

String lower = str.toLowerCase();  // Converts to lowercase
```

6. Trimming Whitespaces

```
String trimmed = str.trim();  // Removes leading and trailing spaces
```

7. Replacing Characters

```
String replaced = str.replace("Java", "Programming");
```

8. Comparing Strings

```
boolean isEqual = str.equals(str3);  // Checks if strings are equal
```

9. Using `equalsIgnoreCase()` Method

```
String str1 = "java";
String str2 = "JAVA";
boolean isEqualIgnoreCase = str1.equalsIgnoreCase(str2);  // Compares ignoring case
System.out.println(isEqualIgnoreCase);  // Output: true
```

STRINGBUILDER

2. Commonly Used Methods in StringBuilder

1. Append

```
sb.append(" World");

System.out.println(sb); // Output: Hello World
```

2. Insert

```
sb.insert(5, ",");

System.out.println(sb); // Output: Hello, World
```

3. Replace

```
sb.replace(7, 12, "Java");

System.out.println(sb); // Output: Hello, Java
```

4. Delete

```
sb.delete(5, 6);

System.out.println(sb); // Output: Hello Java
```

5. Reverse

```
sb.reverse();

System.out.println(sb); // Output: avaJ olleH
```

6. Length

```
System.out.println(sb.length()); // Output: 10
```

7. charAt(int index)

```
StringBuilder sb = new StringBuilder("Hello");

System.out.println(sb.charAt(1)); // Output: e
```

8. substring(int start, int end)

```
StringBuilder sb = new StringBuilder("Hello, World");

System.out.println(sb.substring(7, 12));

 // Output: World
```

9. capacity()

```
StringBuilder sb = new StringBuilder("Hello");

System.out.println(sb.capacity()); // Output: 21
(Default: 16 + length of initial string)
```

1. Declaration and Initialization

```
StringBuilder sb = new StringBuilder();

System.out.println(sb); // Output: (empty string)

StringBuilder sb = new StringBuilder("Hello");

System.out.println(sb); // Output: Hello
```

3. Convert StringBuilder to String

```
String str = sb.toString();
```

3. Convert String to StringBuilder

```
StringBuilder sb = new StringBuilder(str);
```

```
public class PalindromeCheck {

  public static void main(String[] args) {

    String str = "madam";


    // Using StringBuilder

    StringBuilder sb = new StringBuilder(str);

    sb.reverse();

    if (str.equals(sb.toString())) {

      System.out.println(str + " is a palindrome.");

    } else {

  System.out.println(str + " is not a palindrome.");

    }

  }

}
```

HASHMAP

1. Creating a HashMap

```
HashMap<KeyType, ValueType> map = new
HashMap<>();
```

2. Adding Elements

```
map.put(K key, V value);
```

3. Accessing Elements

```
V value = map.get(Object key);
```

4. Removing Elements

```
map.remove(Object key);
```

5. Checking for Key/Value

```
boolean containsKey(Object key);

boolean containsValue(Object value);
```

6. Size and Emptiness

```
map.size(); // int

map.isEmpty(); // true or false
```

7. Iterating Over Elements

Iterating Over Keys:

```
for (K key : map.keySet()) {
   // Access value with map.get(key)
}
```

Iterating Over Values:

```
for (V value : map.values()) {
   // Access value
}
```

Iterating Over Key-Value Pairs:

```
for (Map.Entry<K, V> entry : map.entrySet())
{
   K key = entry.getKey();
   V value = entry.getValue();
}
```

8. Clearing All Elements

```
map.clear();
```

9. Default Value

```
V value = map.getOrDefault(K key, V defaultValue);
```

10. Traverse the array and count occurrences

```
 for (int num : arr)
{
 map.put(num, map.getOrDefault(num, 0) + 1);
}
```

```
import java.util.HashMap;
public class HashMapExample {
   public static void main(String[] args) {
      // Creating a HashMap to store the name and age of people
      HashMap<String, Integer> map = new HashMap<>();
      // Adding key-value pairs to the HashMap
      map.put("Alice", 25);
      // Print the entire HashMap
      System.out.println("HashMap: " + map);
      // Accessing a value using a key
      System.out.println("Age of Alice: " + map.get("Alice"));
      // Check if a key exists
      if (map.containsKey("Bob")) {
         System.out.println("Bob is in the map with age: " +
map.get("Bob"));
      } else {
         System.out.println("Bob is not in the map.");
      }
      // Remove a key-value pair
      map.remove("David");
      System.out.println("HashMap after removing David: " + map);
      // Iterate over the HashMap using for-each loop
      System.out.println("Iterating over the HashMap:");
      for (String key : map.keySet()) {
         System.out.println(key + " : " + map.get(key));
      }
      // Check if a value exists
      if (map.containsValue(30)) {
         System.out.println("There is someone with age 30.");
      }
      // Get the size of the HashMap
      System.out.println("Size of the HashMap: " + map.size());
   }
}
```

HASHSET, STACK, QUEUE

```
import java.util.HashSet;

    // Create a HashSet

    HashSet<String> set = new HashSet<>();

    // Adding elements to the set

    set.add("Apple");  // Duplicate value, won't be added

    // Print the HashSet (order may not be the same)

    System.out.println("HashSet: " + set);

    // Check if an element exists

    set.contains("Banana")

    // Remove an element

    set.remove("Cherry");

     // Get the size of the set

    set.size();

    // Iterate over the HashSet using for-each loop

     for (String fruit : set) {
       System.out.println(fruit);
    }
     // Check if the set is empty

    set.isEmpty();

    // Clear the HashSet

    set.clear();
```

MATH FUNCTIONS

```
import java.lang.Math;
Math.pow(2, 3); // 2 raised to the power 3
Math.sqrt(16); // Square root of 16
Math.max(1, 2); // Maximum of 1 and 2
Math.min(1, 2); // Minimum of 1 and 2
```

STACK

```
import java.util.Stack;
Stack<Integer> stack = new Stack<>();
stack.push(1); // Push element
stack.pop(); // Pop element
stack.peek(); // Get top element
stack.isEmpty(); // Check if stack is empty
stack.size(); // size of stack
```

QUEUE

CREATING QUEUE

```
Queue<Type> queue = new LinkedList<>();
Queue<Type> queue = new PriorityQueue<>();
Queue<Type> queue = new ArrayDeque<>();

import java.util.Queue;
import java.util.LinkedList;

public class QueueExample {
  public static void main(String[] args) {
    // Create a Queue using LinkedList
    Queue<String> queue = new LinkedList<>();

    // Add elements to the queue
    queue.offer("Apple");
    queue.offer("Banana");
    queue.offer("Cherry");

    // Peek at the front element
System.out.println(queue.peek());  // Output: Apple

    // Remove the front element
    System.out.println(queue.poll());  // Output: Apple

    // Check if the queue is empty
    queue.isEmpty();
    // Get the size of the queue
   queue.size();
    // Check if the queue is empty now
    queue.isEmpty();
```