Data Structure	Definition	Key Characteristics	Example	Java Implementation	Practical Application	Where to Learn More
Array	A collection of elements stored in contiguous memory locations.	Fixed size (usually), direct access via index, efficient for sequential access.	[1, 2, 3, 4]	int[] arr = {1, 2, 3, 4};	Dynamic programming, binary search, storing ordered data.	Java Arrays Documentation
Linked List	A linear data structure where each element (node) points to the next.	Dynamic size, insertion and deletion are efficient (if you have a reference), sequential access.	1 -> 2 -> 3 -> NULL	LinkedList<Integer> list = new LinkedList<>();	Implementing stacks, queues, dynamic memory allocation, avoids fixed size.	Java LinkedList Class
Stack	A collection of elements that follows the LIFO (Last In First Out) principle.	LIFO order, efficient push and pop operations at one end, easy to implement.	push(1), push(2), pop() -> 2	Stack<Integer> stack = new Stack<>();	Function call management, syntax parsing, undo/redo operations.	Java Stack Class
Queue	A collection of elements that follows the FIFO (First In First Out) principle.	FIFO order, efficient enqueue and dequeue operations, easy to implement.	enqueue(1), enqueue(2), dequeue() -> 1	Queue<Integer> queue = new LinkedList<>();	Scheduling algorithms, breadth-first search, handling requests in servers.	Java Queue Interface
Deque	A generalized form of queue supporting insertion and deletion at both ends (Double-ended Queue).	Flexible insertion/deletion at both ends, can be used as a stack or queue.	addFirst(1), addLast(2)	Deque<Integer> deque = new LinkedList<>();	Sliding window problems, palindrome checking, implementing queues or stacks.	Java Deque Interface
Hash Table	A data structure that maps keys to values using a hash function.	Fast lookups, insertions, and deletions (average case), unordered storage, may have collisions.	{key: value} e.g., {1: 'one', 2: 'two'}	HashMap<Integer, String> map = new HashMap<>();	Caching, implementing dictionaries, fast lookups by key.	Java HashMap Class
Binary Tree	A tree structure where each node has at most two children (left and right).	Hierarchical data representation, can have different properties based on the structure.	1 / \ 2 3	class TreeNode { int val; TreeNode left, right; }	Hierarchical data representation, parsing expressions.	Binary Tree Implementation in Java
Binary Search Tree	A binary tree where left < root < right for all nodes.	Ordered structure, efficient for search, insertion, deletion (in average cases).	2 / \ 1 3	class TreeNode { int val; TreeNode left, right; }	Searching, database indexing, maintaining sorted data.	Java Binary Search Tree
Heap	A special tree-based structure satisfying the heap property (min or max heap).	Partial order, efficient for finding min or max element, used for priority queues.	10 / \ 5 8	PriorityQueue<Integer> heap = new PriorityQueue<>();	Priority queues, heap sort, graph algorithms like Dijkstra's.	Java PriorityQueue Class
Graph	A collection of nodes (vertices) and edges that connect them.	Represent relationships between entities, can be directed or undirected, cyclic or acyclic.	A -> B -> C	Map<Integer, List<Integer>> graph = new HashMap<>();	Social networks, transportation networks, shortest path problems, network analysis.	Graph Implementation in Java
Trie (Prefix Tree)	A tree-like data structure for storing strings based on shared prefixes.	Efficient prefix-based searching, useful for string operations.	"cat", "car" stored in a prefix tree	class TrieNode { Map<Character, TrieNode> children; boolean isEndOfWord; }	Autocomplete, spell checkers, IP routing, dictionary implementation.	Trie Implementation in Java
Segment Tree	A tree for storing intervals or segments for range queries.	Efficient range queries (e.g., sum, min, max), pre-processed data.	sum(0,3)	class SegmentTree { int[] tree; }	Range query problems (range sum, min, max), computational geometry.	Segment Tree in Java
Fenwick Tree (Binary Indexed Tree)	A tree for efficient prefix sum queries.	Efficient prefix sum queries and updates, compact representation.	sum(0,5)	class FenwickTree { int[] tree; }	Frequency analysis, range queries, cumulative frequency tables.	Fenwick Tree in Java
Set	A collection of unique elements (no duplicates).	Ensures uniqueness, unordered collection of elements.	{1, 2, 3}	Set<Integer> set = new HashSet<>();	Ensuring uniqueness, mathematical set operations, graph implementations.	Java Set Interface