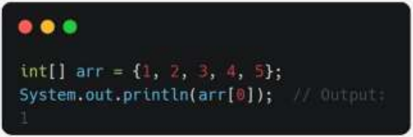
DSA IN JAVA

Save

1. Arrays

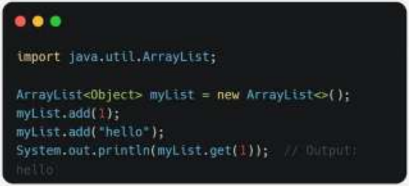
Arrays are collections of elements of the same data type, accessed by an index.



```
int[] arr = {1, 2, 3, 4, 5};
System.out.println(arr[0]);  // Output:
1
```

2. Lists

Lists are ordered collections that can hold elements of different data types.

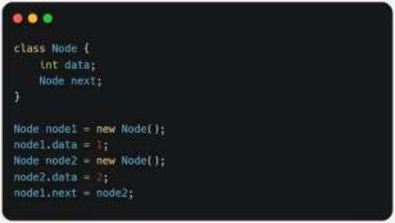


```
import java.util.ArrayList;

ArrayList<Object> myList = new ArrayList<>();
myList.add(1);
myList.add("hello");
System.out.println(myList.get(1));  // Output:
hello
```

3. Linked Lists

Linked lists are linear data structures where each element is a separate object linked together.

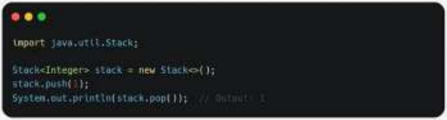


```
class Node {
    int data;
    Node next;
}

Node node1 = new Node();
node1.data = 1;
Node node2 = new Node();
node2.data = 2;
node1.next = node2;
```

4. Stacks

Stacks follow the Last In First Out (LIFO) principle.

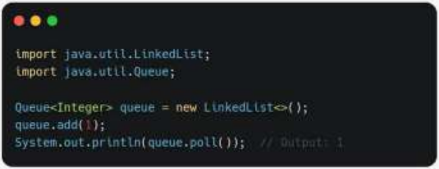


```
import java.util.Stack;

Stack<Integer> stack = new Stack<>();
stack.push(1);
System.out.println(stack.pop()); // Output: 1
```

5. Queues

Queues follow the First In First Out (FIFO) principle.

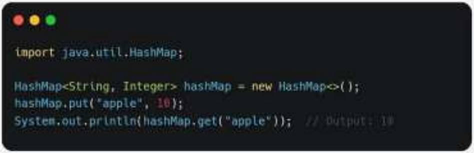


```java
import java.util.LinkedList;
import java.util.Queue;

Queue<Integer> queue = new LinkedList<>();
queue.add(1);
System.out.println(queue.poll());  // Output: 1
```

6. Hash Tables

Hash tables store key-value pairs and provide efficient lookup.

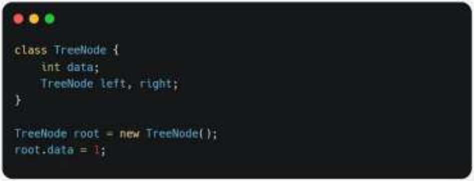


```java
import java.util.HashMap;

HashMap<String, Integer> hashMap = new HashMap<>();
hashMap.put("apple", 10);
System.out.println(hashMap.get("apple"));  // Output: 10
```

7. Trees

Trees are hierarchical data structures with nodes connected by edges.

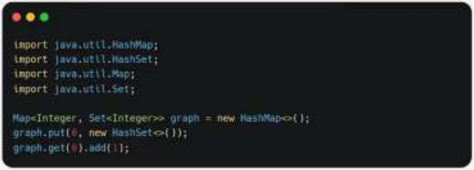


```
class TreeNode {
    int data;
    TreeNode left, right;
}

TreeNode root = new TreeNode();
root.data = 1;
```

8. Graphs

Graphs consist of vertices and edges that connect them.

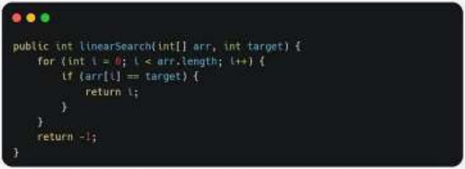


```
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

Map<Integer, Set<Integer>> graph = new HashMap<>();
graph.put(0, new HashSet<>());
graph.get(0).add(1);
```

9. Linear Search

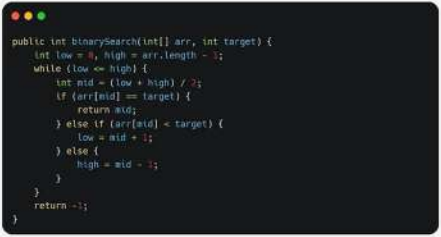
Linear search iterates through each element in a list to find the target value.



```
public int linearSearch(int[] arr, int target) {
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == target) {
            return i;
        }
    }
    return -1;
}
```

10. Binary Search

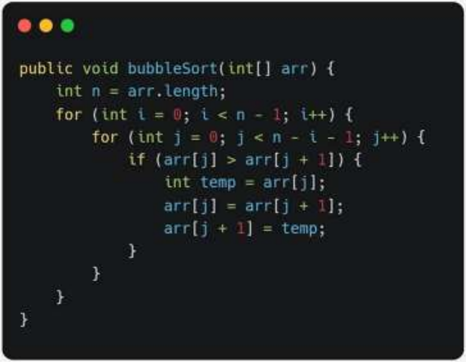
Binary search finds the target value by repeatedly dividing the search interval in half.



```
public int binarySearch(int[] arr, int target) {
    int low = 0, high = arr.length - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == target) {
            return mid;
        } else if (arr[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1;
}
```

11. Bubble Sort

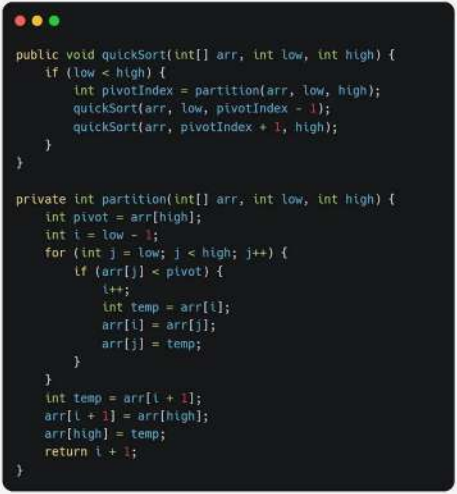
Bubble sort repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order.



```java
public void bubbleSort(int[] arr) {
    int n = arr.length;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
```

12. Quick Sort

Quick sort selects a pivot element and partitions the array around the pivot.

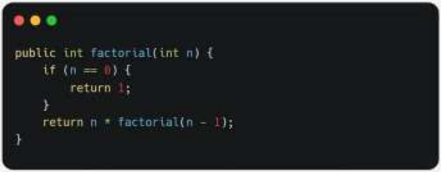


```java
public void quickSort(int[] arr, int low, int high) {
    if (low < high) {
        int pivotIndex = partition(arr, low, high);
        quickSort(arr, low, pivotIndex - 1);
        quickSort(arr, pivotIndex + 1, high);
    }
}

private int partition(int[] arr, int low, int high) {
    int pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;
    return i + 1;
}
```

13. Recursion

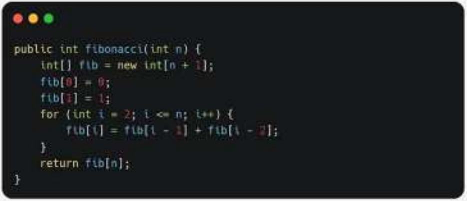
Recursion is a programming technique where a function calls itself.



```java
public int factorial(int n) {
    if (n == 0) {
        return 1;
    }
    return n * factorial(n - 1);
}
```

14. Dynamic Programming

Dynamic programming breaks down complex problems into simpler subproblems.



```java
public int fibonacci(int n) {
    int[] fib = new int[n + 1];
    fib[0] = 0;
    fib[1] = 1;
    for (int i = 2; i <= n; i++) {
        fib[i] = fib[i - 1] + fib[i - 2];
    }
    return fib[n];
}
```