Table of Contents

[1. Java Core Summary 2](#_Toc199860165)

[1.1 Data Types 2](#_Toc199860166)

[1.2 Operators 2](#_Toc199860167)

[1.3 Control Flow 2](#_Toc199860168)

[1.4 OOP Concepts 2](#_Toc199860169)

[1.5 Keywords & Constructs 3](#_Toc199860170)

[1.6 Exception Handling 3](#_Toc199860171)

[1.7 Strings 3](#_Toc199860172)

[1.8 Arrays 3](#_Toc199860173)

[1.9 Collections Framework 3](#_Toc199860174)

[1.10 Java 8 Highlights 4](#_Toc199860175)

[2. Data Structures 4](#_Toc199860176)

[2.1 Array 4](#_Toc199860177)

[2.2 ArrayList 4](#_Toc199860178)

[2.3 LinkedList 4](#_Toc199860179)

[2.4 Stack (LIFO) 4](#_Toc199860180)

[2.5 Queue (FIFO) 4](#_Toc199860181)

[2.6 HashMap 4](#_Toc199860182)

[2.7 HashSet 4](#_Toc199860183)

[2.8 Trees 5](#_Toc199860184)

[2.9 Graph 5](#_Toc199860185)

[3. Algorithms 5](#_Toc199860186)

[3.1 Searching 5](#_Toc199860187)

[3.2 Sorting Algorithms 5](#_Toc199860188)

[3.3 Recursion 5](#_Toc199860189)

[3.4 Time & Space Complexity Table 6](#_Toc199860190)

[Advance: 6](#_Toc199860191)

# 1. Java Core Summary

## 1.1 Data Types

* **Primitive:** byte (1B), short (2B), int (4B), long (8B), float (4B), double (8B), boolean (1B), char (2B)
* **Non-Primitive:** String, Array, Class, Interface, Enum

## 1.2 Operators

|  |  |
| --- | --- |
| **Operator Type** | **Operators** |
| Arithmetic | + , - , \* , / , % |
| Logical | && , || , ! |
| Relational | == , != , < , > , <= , >= |
| Assignment | = , += , -= , \*= , /= , %= , etc. |
| Bitwise | & , | , ^ , ~ , << , >> |

## 1.3 Control Flow

* **Conditionals:** if, else, switch
* **Loops:** for, while, do-while, for-each
* **Jump:** break, continue, return

## 1.4 OOP Concepts

* **Encapsulation:** Hide data using private fields + public methods
* **Abstraction:** Hide implementation using abstract class or interface
* **Inheritance:** extends for code reuse
* **Polymorphism:**
  + Compile-time: Method Overloading
  + Run-time: Method Overriding
* **Abstract vs Interface:**
  + Abstract: can have fields and methods (some defined)
  + Interface: no fields, only method declarations (Java 8+ allows default/static)

## 1.5 Keywords & Constructs

* **this:** current object
* **super:** parent class
* **final:** constant, non-overridable method, or sealed class
* **static:** class-level, not instance-level
* **Constructor:** no return type, runs at object creation

## 1.6 Exception Handling

* **Syntax:** try-catch-finally
* **Checked:** IOException, SQLException (must handle)
* **Unchecked:** NullPointerException, ArrayIndexOutOfBounds

## 1.7 Strings

* **Immutable**
* **Methods:** length(), charAt(), substring(), equals(), indexOf()
* **Compare:** **==** (reference), .equals() (value)
* **StringBuilder**: mutable alternative

## 1.8 Arrays

int[] arr = {1, 2, 3};

int[][] mat = new int[2][3];

* Fixed size, contiguous memory
* **Methods:** Arrays.toString(), Arrays.sort()

## 1.9 Collections Framework

|  |  |  |
| --- | --- | --- |
| **Interface** | **Class** | **Features** |
| List | ArrayList | Ordered, duplicates allowed |
| Set | HashSet | No duplicates, unordered |
| Map | HashMap | Key-value pairs, unique keys |
| Queue | LinkedList | FIFO, supports null elements |

* **Key methods**: add(), remove(), get(), contains(), put()

## 1.10 Java 8 Highlights

* **Lambda:** (x, y) -> x + y
* **Stream API:** filter/map/collect
* **Functional Interface:** Single abstract method (@FunctionalInterface)

# 2. Data Structures

## 2.1 Array

* Access O(1), Search O(n), Insert/Delete O(n)

## 2.2 ArrayList

* Dynamic array, Resize on demand
* Access O(1), Insert/Delete O(n)

## 2.3 LinkedList

* Node-based structure, Doubly-linked
* Insert/Delete O(1) at ends, O(n) elsewhere

## 2.4 Stack (LIFO)

* **Methods:** push(), pop(), peek()
* **Use cases**: Undo, DFS, Expression parsing

## 2.5 Queue (FIFO)

* **Methods**: add(), remove(), peek()
* **Use cases:** BFS, Scheduling

## 2.6 HashMap

* Key-value pair, Avg O(1), Worst O(n)
* **Use cases:** frequency count, caching

## 2.7 HashSet

* Unique values only, O(1) ops

## 2.8 Trees

* **Binary Tree:** ≤2 children
* **BST:** left < root < right
* **Traversals:** Inorder (LNR), Preorder (NLR), Postorder (LRN)
* Avg O(log n), Worst O(n)

## 2.9 Graph

* **Types:** Directed, Undirected, Weighted
* **Representations:** Adjacency List/Matrix
* **Traversals:** BFS (queue), DFS (stack/recursion)

# 3. Algorithms

## 3.1 Searching

* **Linear Search:** O(n), unsorted data
* **Binary Search:** O(log n), requires sorted array

## 3.2 Sorting Algorithms

* **Bubble Sort:** Simple, compare-adjacent.
* **Selection Sort:** Select min each pass.
* **Insertion Sort:** Build sorted list.
* **Merge Sort:** Divide & Conquer
* **Quick Sort:** Pivot-based.

## 3.3 Recursion

* Self-calling function
* Needs base case
* **Used in:** DFS, Factorial, Fibonacci, Backtracking

int fact(int n) {

if (n == 0) return 1;

return n \* fact(n - 1);

}

## 3.4 Time & Space Complexity Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Algorithms | Best | Average | Worst | Space | Stable |
| Bubble Sort | O(n) | O(n²) | O(n²) | O(1) | Yes |
| Selection Sort | O(n²) | O(n²) | O(n²) | O(1) | No |
| Insertion Sort | O(n) | O(n²) | O(n²) | O(1) | Yes |
| Merge Sort | O(n log n) | O(n log n) | O(n log n) | O(n) | Yes |
| Quick Sort | O(n log n) | O(n log n) | O(n²) | O(log n) | No |
| Linear Search | O(1) | O(n) | O(n) | O(1) |  |
| BinarySearch | O(1) | O(log n) | O(log n) | O(1) |  |
| Stack/Queue | O(1) | O(1) | O(1) | O(n) |  |

# Advance:

* Advanced Java features like concurrency (threads, synchronization).
* Deep dive into JVM internals and memory management.
* Advanced data structures (like balanced trees, tries, heaps, graphs in depth).
* Complex algorithms (dynamic programming, graph algorithms like Dijkstra, Floyd-Warshall).
* Design patterns, system design topics.
* Multithreading, Streams API in more depth

Thank You