INTERVIEW PREPARATION CHEAT SHEET

DSA Core Topics (Basic Level)

1. **Kadane's Algorithm** - Find maximum subarray sum
2. **Palindrome Check** - Check if a string is a palindrome
3. **Cycle Detection in Linked List** - Floyd's Cycle Detection Algorithm
4. **Linked List Reversal** - Iterative & Recursive approaches
5. **Intersection of Two Linked Lists** - Identify merging node
6. **Stack Using Queues** - Implement stack using two queues
7. **Binary Tree Inorder Traversal** - Recursive & iterative methods
8. **Binary Tree Height** - Recursive height calculation
9. **Level Order Traversal** - BFS approach in tree
10. **Cycle in Undirected Graph** - DFS-based detection
11. **Topological Sort** - DFS-based in DAG
12. **Shortest Path (Unweighted Graph)** - BFS method

Java Fundamentals (Basic Level)

1. **== vs .equals()** - Compare reference vs value
2. **Java Memory Model** - Heap, Stack, Method Area
3. **Exception Hierarchy** - Throwable > Error / Exception
4. **Checked vs Unchecked Exceptions**
5. **Final Keyword** - Variable, Method, Class
6. **Garbage Collection** - How Java handles memory
7. **ArrayList vs LinkedList** - Use cases & differences
8. **Wrapper Classes** - Autoboxing and unboxing
9. **String vs StringBuffer vs StringBuilder**
10. **Pass by Value** - Java always uses pass-by-value
11. **Thread Lifecycle** - New → Runnable → Running → Terminated
12. **Static Block/Method** - Execution before main

OOPs Concepts (Basic Level)

1. **4 Pillars of OOP** - Encapsulation, Inheritance, Abstraction, Polymorphism
2. **Abstract Class vs Interface**
3. **Overloading vs Overriding**
4. **Encapsulation Example** - Private fields with public getters/setters
5. **Inheritance in Java** - extendskeyword
6. **Polymorphism** - Compile-time & runtime
7. **Abstraction in Banking System** - Hiding implementation
8. **Access Modifiers** - Private, Default, Protected, Public
9. **Constructor Chaining** - Using this()or super()
10. **super Keyword** - Access parent class members
11. **Default Methods in Interface** - Java 8 feature
12. **Why No Multiple Inheritance with Classes** - Diamond Problem

Pseudo-code & Logic Building

1. **Second Largest in Array** - Traverse & compare

2. **Balanced Parentheses** - Stack-based logic
3. **First Non-Repeating Character** - Queue & HashMap
4. **Rotate Matrix 90°** - Transpose + reverse
5. **LRU Cache** - LinkedHashMap or DLL + HashMap
6. **Infix to Postfix** - Stack implementation
7. **Count Set Bits** - Brian Kernighan's algorithm
8. **Anagram Detection** - Frequency map comparison
9. **Custom String Tokenizer** - Manual split logic
10. **Longest Common Prefix** - Vertical scanning
11. **Power of Two Check** - Bitwise AND method
12. **Elevator Simulation Logic** - State machine-based

Scenario-Based Questions

1. **Parking Lot System** - Classes: Vehicle, Slot, ParkingLot
2. **Food Delivery App** - Flow from order to delivery
3. **Rate Limiter Design** - Token bucket/Leaky bucket
4. **Chat App Logic** - User sessions, real-time messages
5. **Fraud Detection System** - Pattern matching, alerts
6. **URL Shortener** - Hashing + redirection
7. **Inventory Caching** - Cache + DB fallback
8. **Attendance System (OOP)** - Class: Student, Attendance
9. **Duplicate Login Sessions** - Token/session ID tracking
10. **Compiler Logic Design** - Lexical → Parsing → CodeGen
11. **File Versioning System** - Timestamps & rollback
12. **Hotel Booking System** - Rooms, Availability, Booking Flow

Critical Thinking – Programming Based

1. **Why are Strings immutable in Java? What problems does this solve?**
2. **If recursion uses a stack, how would you convert a recursive solution to iterative?**
3. **What trade-offs do you consider when choosing between an ArrayList and a LinkedList?**
4. **Explain the impact of time complexity when using nested loops.**
5. **Why might a HashMap give unexpected results with custom objects as keys?**
6. **What is a memory leak in Java and how can it happen despite GC?**
7. **How would you debug a program with intermittent bugs (non-reproducible)?**
8. **How does lazy initialization help in Singleton design pattern?**
9. **Why is modular code important in team-based development?**
10. **Why should equals() and hashCode() be overridden together?**
11. **How would you design your code to be testable and maintainable?**
12. **If your app crashes only in production, how would you trace the issue without logs?**