Let me create a focused 7-day plan to master data structures and algorithms with Java, structured for one hour daily study sessions.

Day 1: Arrays and Basic Sorting
Morning Focus (30 minutes):
Understanding array operations and manipulations, including:

- Multi-dimensional arrays
- Array rotation techniques
- Sliding window patterns
- Kadane's algorithm for maximum subarray
- Prefix sum array applications

Afternoon Practice (30 minutes):
Implementation of fundamental sorting algorithms:

- Bubble Sort with optimization
- Selection Sort with comparator
- Insertion Sort and its variants
- Understanding time/space complexity
- Best/worst case scenarios

Project of the Day:
Build a sorting visualizer that demonstrates how different sorting algorithms work on various input sizes.

Day 2: Search Algorithms and LinkedLists
Morning Focus (30 minutes):
Search algorithm implementations:

- Binary search variants
- Two pointer techniques
- Dutch national flag algorithm
- Quick select algorithm
- Jump search implementation

Afternoon Practice (30 minutes):
LinkedList operations and patterns:

- Singly vs Doubly linked lists
- Floyd's cycle detection

- Reverse linkages
- Merge sorted lists
- Skip list concept

Project of the Day:
Implement a music playlist system using doubly linked lists with skip functionality.

Day 3: Stacks and Queues
Morning Focus (30 minutes):
Stack applications:

- Balanced parentheses
- Next greater element
- Min stack design
- Expression evaluation
- Stack using queues

Afternoon Practice (30 minutes):
Queue implementations:

- Circular queue
- Priority queue operations
- Deque applications
- Queue using stacks
- Sliding window maximum

Project of the Day:
Build a task scheduler with priority handling and task dependencies.

Day 4: Trees and Binary Search Trees
Morning Focus (30 minutes):
Tree traversal patterns:

- Level order traversal
- Boundary traversal
- Vertical order traversal
- Mirror operations
- Lowest common ancestor

Afternoon Practice (30 minutes):
BST operations:

- Insertion and deletion
- Balancing techniques
- Range queries
- Predecessor/Successor
- BST validation

Project of the Day:
Create a file system navigator using tree structures.

Day 5: Advanced Tree Structures
Morning Focus (30 minutes):
AVL and Red-Black trees:

- Self-balancing mechanisms
- Rotation operations
- Height balancing
- Tree transformations
- Performance analysis

Afternoon Practice (30 minutes):
Heap operations:

- Min/Max heap implementation
- Heap sort
- K-way merging
- Median maintenance
- Heap optimization

Project of the Day:
Implement a real-time ranking system using heap structures.

Day 6: Graphs and Graph Algorithms
Morning Focus (30 minutes):
Graph representations:

- Adjacency matrix/list
- BFS/DFS applications
- Cycle detection
- Topological sorting
- Connected components

Afternoon Practice (30 minutes):
Path finding algorithms:

- Dijkstra's algorithm
- Bellman-Ford algorithm
- Floyd-Warshall algorithm
- Minimum spanning trees
- Network flow basics

Project of the Day:
Build a social network analyzer with relationship strength metrics.

Day 7: Dynamic Programming and Advanced Concepts
Morning Focus (30 minutes):
DP fundamentals:

- Memoization techniques
- Tabulation methods
- State transition
- Space optimization
- Subsequence patterns

Afternoon Practice (30 minutes):
Advanced concepts:

- Trie data structure
- Segment trees
- Fenwick trees
- Disjoint sets
- String algorithms

Project of the Day:
Create a word suggestion system using Trie with frequency tracking.

Daily Practice Structure:

1. Theory Review (10 minutes):
    - Read concept documentation
    - Watch visualization videos
    - Understand complexity analysis
2. Implementation (30 minutes):

    - Code basic operations
    - Handle edge cases
    - Test with various inputs
    - Optimize solutions
3. Problem Solving (20 minutes):
    - Solve related LeetCode problems
    - Practice interview questions
    - Analyze different approaches

Additional Tips for Success:

- Keep a code repository of implementations
- Draw diagrams for visual understanding
- Practice with real-world examples
- Focus on one concept at a time
- Review previous day's concepts
- Write clean, documented code
- Test edge cases thoroughly

Would you like me to provide specific coding exercises or detailed implementations for any of these topics?