

Chapter 6

■ Code Implementation

Clean Java code — from basic to production-ready

■ Java Stack using ArrayDeque (Recommended)

Stack — ArrayDeque (Production)

JAVA

```
import java.util.Deque;
import java.util.ArrayDeque;

// Java's recommended Stack implementation
// ArrayDeque is faster than Stack (no synchronization)
Deque stack = new ArrayDeque<>();

stack.push(10); // push to top
stack.push(20);
stack.push(30);

stack.peek(); // returns 30, does not remove
stack.pop(); // returns 30, removes it
stack.isEmpty(); // false
stack.size(); // 2
```

■ Java Queue using LinkedList

Queue — LinkedList (Production)

JAVA

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

Queue queue = new LinkedList<>();

queue.offer(1); // enqueue – use offer() not add()
queue.offer(2); // add() throws exception on failure
queue.offer(3); // offer() returns false on failure
```

```
queue.peek(); // returns 1 (front), no removal
queue.poll(); // returns 1, removes it
queue.size(); // 2
queue.isEmpty(); // false
```

■ Valid Parentheses — Classic Stack Problem

Valid Parentheses — LeetCode 20

JAVA

```
public boolean isValid(String s) {
    Deque stack = new ArrayDeque<>();

    for (char c : s.toCharArray()) {
        // Push all openers onto stack
        if (c == '(' || c == '{' || c == '[') {
            stack.push(c);
        } else {
            // Stack empty means unmatched closer
            if (stack.isEmpty()) return false;

            char top = stack.pop();
            // Check if top matches current closer
            if (c == ')' && top != '(') return false;
            if (c == '}' && top != '{') return false;
            if (c == ']' && top != '[') return false;
        }
    }

    // Stack must be empty at end
    return stack.isEmpty();
}

// Time: O(n) | Space: O(n)
```

■ Min Stack — Track Minimum in O(1)

Min Stack — LeetCode 155

JAVA

```
class MinStack {
```

```
private Deque stack = new ArrayDeque<>();
private Deque minStack = new ArrayDeque<>();

public void push(int val) {
    stack.push(val);
    // minStack tracks current minimum at each level
    int currentMin = minStack.isEmpty() ? val
    : Math.min(val, minStack.peek());
    minStack.push(currentMin);
}

public void pop() {
    stack.pop();
    minStack.pop(); // keep in sync!
}

public int top() { return stack.peek(); }
public int getMin() { return minStack.peek(); }
}

// All operations O(1) time, O(n) space
```