

Chapter 8

Common Mistakes

Every beginner makes these — understand WHY to never repeat them

1. Not Checking isEmpty() Before pop()/peek()

Calling pop() on an empty stack throws NoSuchElementException in Java.

Java	JAVA
<pre>// WRONG – will throw exception if stack is empty int val = stack.pop(); // ■ Danger! // CORRECT – always guard with isEmpty() if (!stack.isEmpty()) { int val = stack.pop(); // ✓ Safe }</pre>	

2. Using Stack class instead of ArrayDeque

java.util.Stack extends Vector, which is synchronized (thread-safe but slow). In 99% of problems, you don't need thread safety. Use ArrayDeque.

Java	JAVA
<pre>// WRONG – Stack is outdated and slow Stack stack = new Stack<>(); // ■ // CORRECT – ArrayDeque is the modern, faster choice Deque stack = new ArrayDeque<>(); // ✓</pre>	

3. Using add() instead of offer() for Queue

queue.add() throws IllegalStateException if capacity is exceeded. queue.offer() returns false gracefully. Always prefer offer().

Java	JAVA
<pre>Queue q = new LinkedList<>();</pre>	

```
q.add(1); // ■ throws exception if bounded queue full
q.offer(1); // ✓ returns false if full – safe

q.remove(); // ■ throws if empty
q.poll(); // ✓ returns null if empty – safe
```

4. Forgetting to Empty Stack2 Before Pouring in Queue-via-Stacks

When implementing a Queue using two stacks, pour Stack1 into Stack2 ONLY when Stack2 is empty. Pouring when S2 has elements destroys order.

Java	JAVA
<pre>int dequeue() { // WRONG: always pour – destroys FIFO order // while (!s1.isEmpty()) s2.push(s1.pop()); ■ // CORRECT: only pour when s2 is empty if (s2.isEmpty()) { while (!s1.isEmpty()) s2.push(s1.pop()); // ✓ } return s2.pop(); }</pre>	

5. Circular Queue Index Overflow

Without modulo wrap, rear keeps incrementing until `ArrayIndexOutOfBoundsException`.

Java	JAVA
<pre>// WRONG – rear goes past array bounds arr[++rear] = x; // ■ rear overflows // CORRECT – circular wrap with modulo rear = (rear + 1) % capacity; // ✓ arr[rear] = x;</pre>	