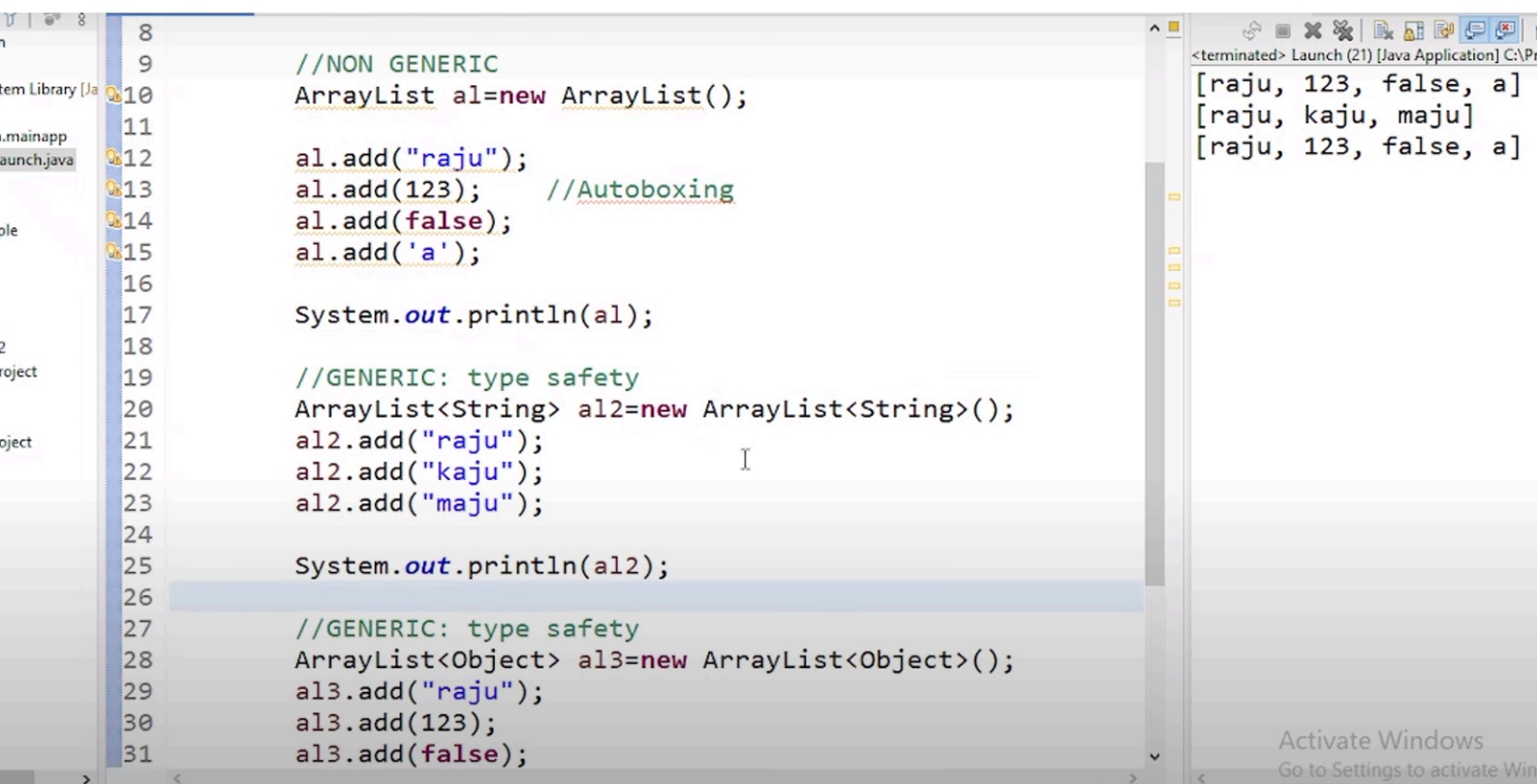


# Lec 38 Collections P3:

This lesson covers three important data structures in Java: Stack, Queue, and PriorityQueue. It explains how each structure organizes and manages data, when to use them, and their underlying mechanisms, making it easier to choose the right one for different programming problems.



The screenshot shows a Java IDE with a code editor on the left and a console on the right. The code editor contains three ArrayList implementations: a non-generic one, a generic one for String, and a generic one for Object. The console shows the output of these implementations, which are printed as arrays.

```
8
9 //NON GENERIC
10 ArrayList al=new ArrayList();
11
12 al.add("raju");
13 al.add(123); //Autoboxing
14 al.add(false);
15 al.add('a');
16
17 System.out.println(al);
18
19 //GENERIC: type safety
20 ArrayList<String> al2=new ArrayList<String>();
21 al2.add("raju");
22 al2.add("kaju");
23 al2.add("maju");
24
25 System.out.println(al2);
26
27 //GENERIC: type safety
28 ArrayList<Object> al3=new ArrayList<Object>();
29 al3.add("raju");
30 al3.add(123);
31 al3.add(false);
```

Output in console:

```
<terminated> Launch (21) [Java Application] C:\Pr
[raju, 123, false, a]
[raju, kaju, maju]
[raju, 123, false, a]
```

- Above code shows types of implementation of arraylist missed in previous notes.

## Stack Basics and Usage

A Stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle. You insert elements using the push method and remove them with pop. Only the

top element can be accessed or removed, not the middle or bottom. Stacks are useful when you need to access the most recently added data first, such as undo operations in text editors.

## **Stack Operations in Java**

In Java, you can create a Stack using the built-in Stack class, supporting both generic and non-generic types. Main methods include push (to insert), pop (to remove the top element), peek (to view the top element without removing), and search (to find the position of an element from the top). Stacks allow duplicates and nulls, but do not support random access or indexing.

## **Stack Example and Behavior**

When you push several items (e.g., "Raju", "Kaju", "Maju") onto a Stack, the last item pushed ("Maju") is at the top and will be the first removed with pop. Using peek shows the top element without removing it. The remove method is generally not used in stacks; pop is preferred for removing elements.

## **Introduction to Queue**

A Queue is a linear data structure that follows the First-In-First-Out (FIFO) principle. Elements are added at the rear and removed from the front, similar to people standing in line. Java provides implementations like `LinkedList` and `ArrayDeque` for queues. Queues are ideal for scenarios like processing tasks in order, such as print jobs or customer service systems.

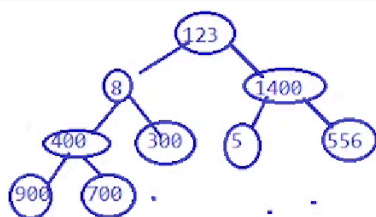
## Queue Operations in Java

In Java, queues are implemented using interfaces and classes like `LinkedList` and `ArrayDeque`. Main methods include `add` (to insert), `poll` (to remove from the front), and `peek` (to view the front element). Queues do not support indexing or random access, but allow duplicates and nulls.

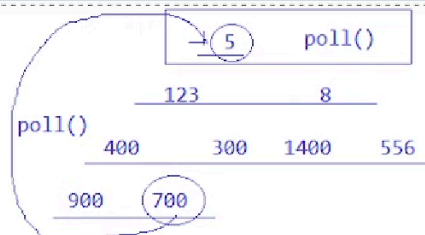
## PriorityQueue Concept

A `PriorityQueue` retrieves elements based on priority rather than insertion order. By default, it uses natural ordering (ascending for numbers), so the smallest element has the highest priority and is removed first. The order can be customized (e.g., descending) using comparators. `PriorityQueues` do not allow nulls and require all elements to be of the same, comparable type.

- Priority queue uses complete binary tree internally.
- In Full Binary tree, all nodes have 2 childs.
- A complete binary tree is a special type of binary tree where all the levels of the tree are filled completely except the lowest level nodes which are filled from as left as possible.
- Priority Queue do not allow nulls and require all elements to be of **Comparable** type.
- When we poll from PQ, then it is not a valid complete binary tree, so the last element (700 in below code) gets on the top of the polled value.
- It takes 700 and not 900 on top because → to maintain the condition of complete binary tree.
- After this, again the **heapify** function arranges the whole tree to maintain the order (Min or Max heap).



HEAPIFY  
(MIN)



[5, 123, 8, 400, 300, 1400, 556, 900, 700]

```

pq.add(123);
pq.add(8);
pq.add(1400);
pq.add(400);
pq.add(300);
pq.add(5);
pq.add(556);
pq.add(900);
pq.add(700);
  
```

8

123      556

400    300    1400    700

900

[8, 123, 556, 400, 300, 1400, 700, 900]

# PriorityQueue Internal Structure: Binary Trees and Heaps

Internally, PriorityQueue uses a complete binary tree, typically structured as a min-heap (parent is smaller than children). This allows efficient insertion and removal of the highest-priority element (the root). Removing an element involves replacing the root with the last element and re-heapifying to maintain the heap property.

## Min-Heap vs Max-Heap

A min-heap keeps the smallest element at the top, so it's removed first. A max-heap keeps the largest element at the top. Java's PriorityQueue uses a min-heap by default, but you can use a comparator to change to a max-heap if needed.

## PriorityQueue and Data Types

PriorityQueue requires all elements to be of the same type and comparable, because it sorts and compares them internally. Mixing data types (like integers and strings) causes errors. Classes used in a PriorityQueue must implement the Comparable interface or be supplied with a Comparator.

# Introduction to Deque (Double-Ended Queue)

A Deque (double-ended queue) allows insertion and removal of elements from both ends. This flexibility supports both stack (LIFO) and queue (FIFO) operations. Deques do not support random access or indexing, but allow duplicates and nulls.--- This summary provides the foundational understanding of stacks, queues, priority queues, and deques, highlighting their unique behaviors, Java implementations, and the reasoning behind their design choices.

```
8      Deque<Object> dq=new ArrayDeque<Object>();
9      dq.addFirst("raju");
10     dq.addFirst("kaju");
11     dq.addLast("xraju");
12     dq.addLast("xkaju");
13
14     System.out.println(dq);
15
16     dq.clear();
17
18     //      dq.pollFirst();
19     //      dq.pollLast();
20
21     //      dq.removeFirst();
22     //      dq.removeLast();
23
24
```

Console ×  
<terminated> Launch (22) [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (Oct 18, 2024, 9:06:55 PM – 9:06:55 PM) [pid: 3296]  
[kaju, raju, xraju, xkaju]  
Exception in thread "main" java.util.NoSuchElementException  
at java.base/java.util.ArrayDeque.removeFirst(ArrayDeque.java:362)  
at coll2.Launch.main(Launch.java:21)

Activate Windows  
Go to Settings to activate Windows.

- In above code, if we use removeFirst / removeLast methods, then they give exception on empty deque.
- But in pollFirst / pollLast methods, we don't get any exception.

