Lec 39 Collections P4:

The session explains different types of Set collections in Java—HashSet, LinkedHashSet, and TreeSet—focusing on how they store unique data, manage order, and handle data removal efficiently. It also covers how hash functions work internally to make operations fast, and when to use each type of Set based on the need for order or uniqueness.

Introduction to Collection Types

Lists, stacks, queues, and sets are different data structures in Java's collection framework. Sets are used to store unique elements, unlike lists that can store duplicates and nulls.

Choosing Between ArrayList and LinkedList

ArrayList is suitable for storing and reading data with possible duplicates and nulls in a non-synchronized environment, especially when no data manipulation is needed after insertion. LinkedList is better when frequent data deletion or manipulation is required.

Synchronized Collections and Vectors

When multiple threads access a collection (synchronized environment), Vector is used as it is thread-safe, unlike ArrayList and LinkedList.

Specialized Collections: PriorityQueue and Deque

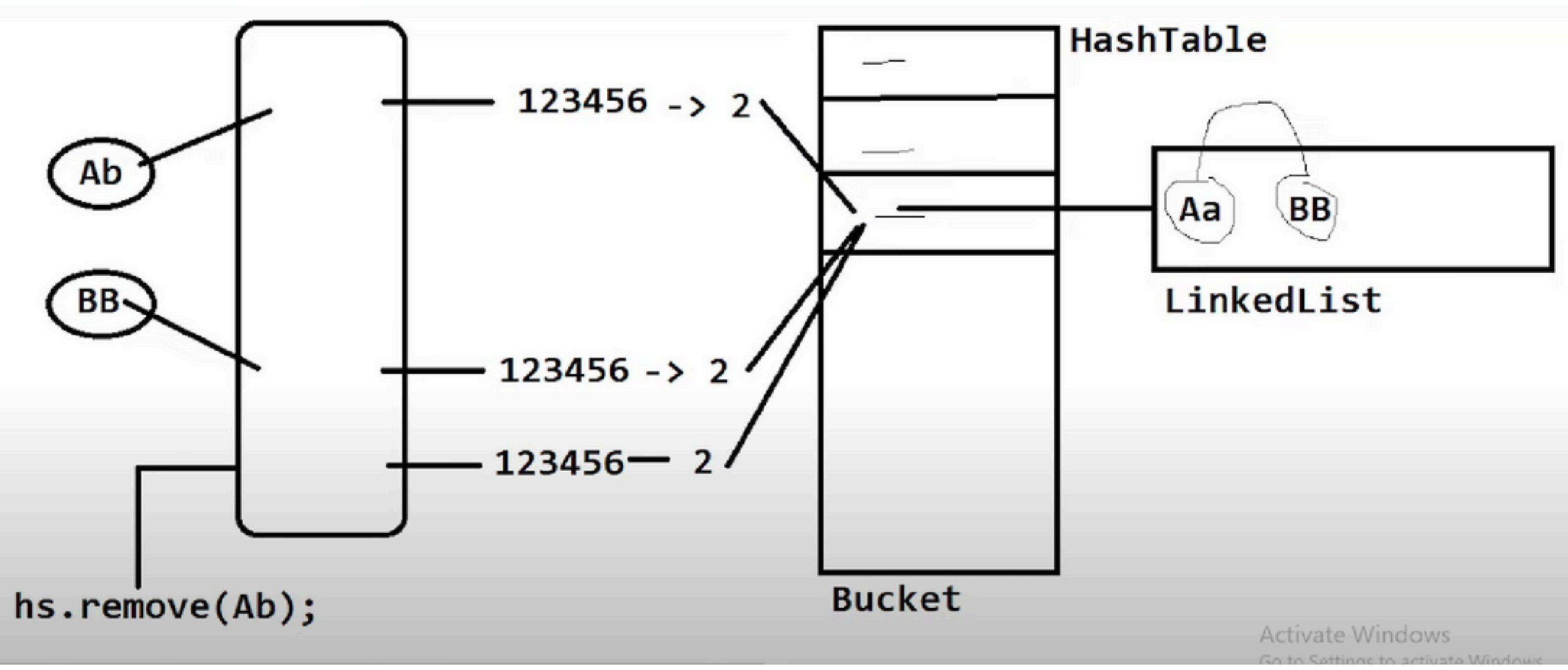
PriorityQueue is used for default sorting and priority-based data removal, while Deque allows insertion and removal from both ends, useful for scenarios like browser navigation.

Removal Operations and Time Complexity

Removing by index in ArrayList is fast (constant time), but removing by value requires traversing the list (linear time). LinkedList avoids shifting elements but still needs traversal to find an object.

Introduction to Set, HashSet, LinkedHashSet, TreeSet

Sets enforce unique data storage, disallowing duplicates. HashSet, LinkedHashSet, and TreeSet are implementations for different needs: HashSet (random order), LinkedHashSet (insertion order), and TreeSet (sorted order).



Properties of HashSet

HashSet does not allow duplicates, allows one null, does not maintain insertion order, and is not synchronized. It uses a hash function to determine where to store each element internally.

How HashSet Stores Data Internally

HashSet uses a hash function to convert objects into numeric hash codes, which determine the storage index in an internal array (default size is 16). The index is calculated using the hash code, but users cannot access elements by index.

Hash Collisions and Buckets

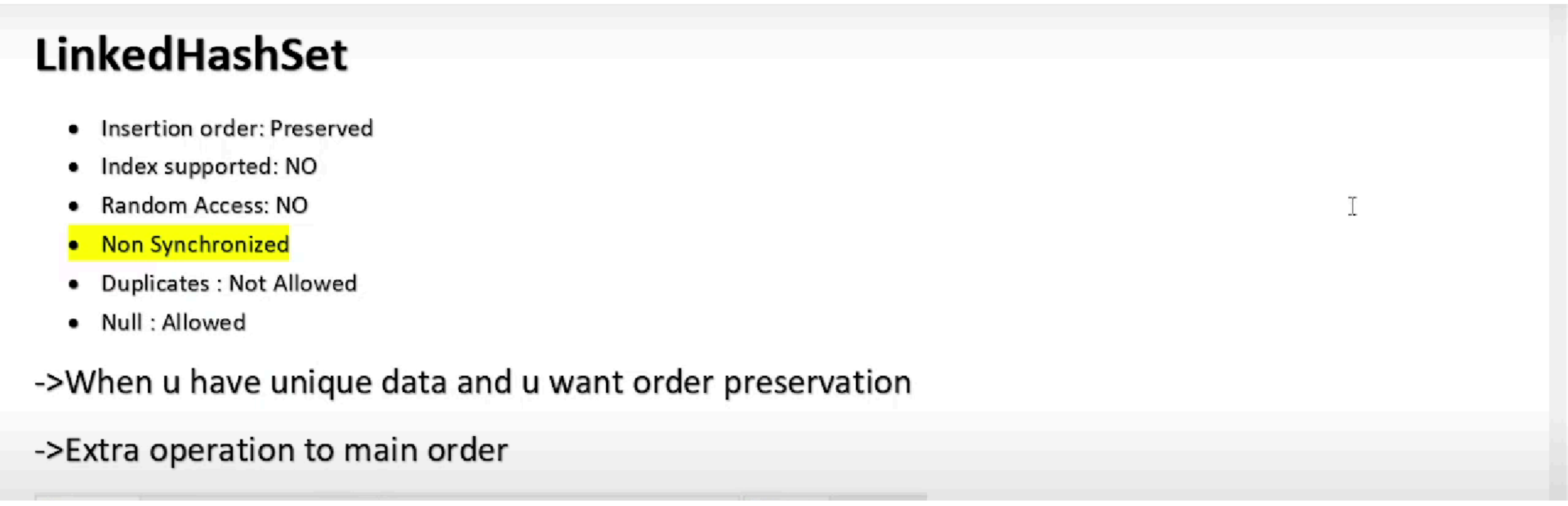
When two objects have the same hash code (collision), they are stored in the same bucket as a linked list. Frequent collisions can reduce efficiency, which is why duplicates are not allowed in sets.

Hash Table and Buckets Explained

Internally, a HashSet is a hash table made of buckets, each storing elements with the same index. The hash code and index calculation enable direct access and quick removal.

LinkedHashSet and Order Preservation

LinkedHashSet maintains the order in which elements are inserted, unlike HashSet. This is achieved by combining a hash table and a linked list, but it is less efficient due to extra operations needed to preserve order.



LinkedHashSet

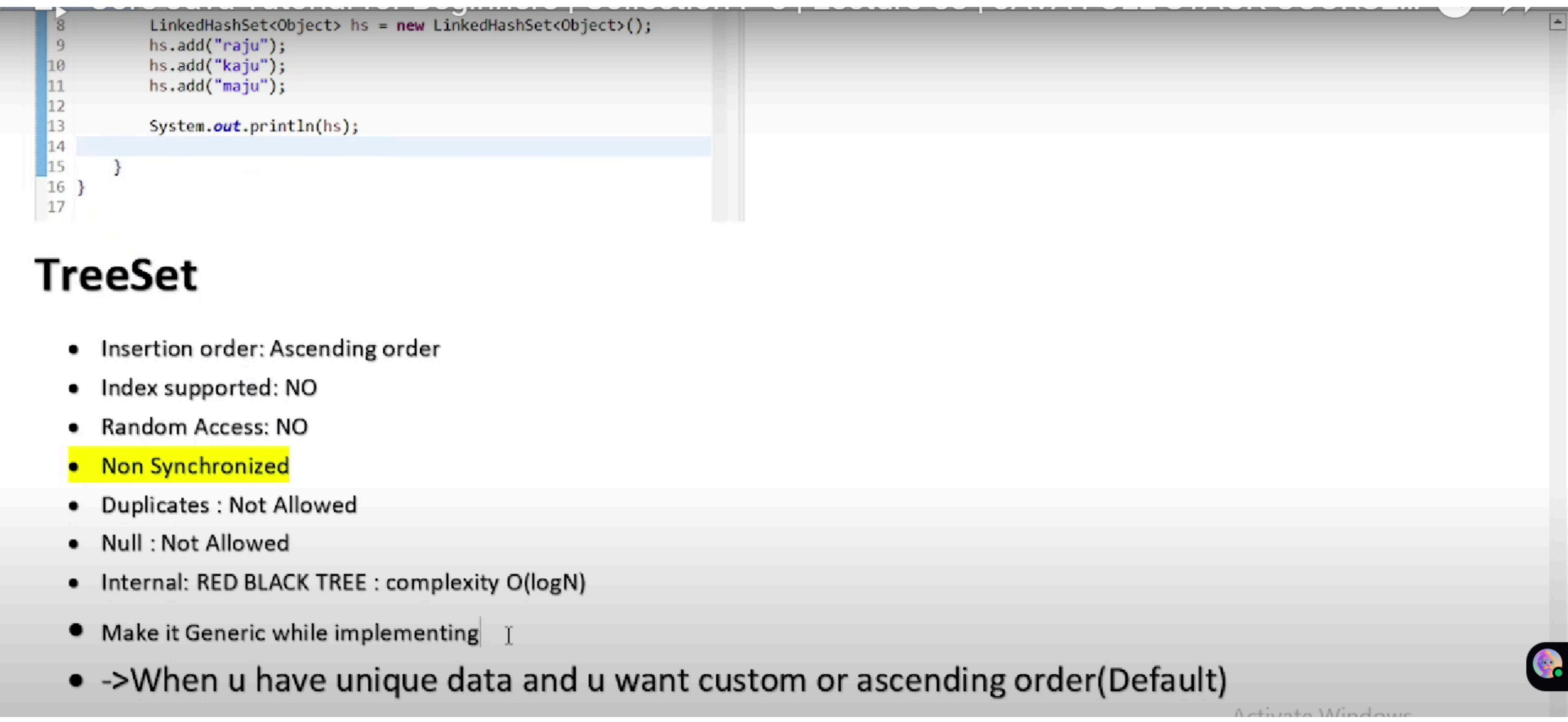
- Insertion order: Preserved
- Index supported: NO
- Random Access: NO
- Non Synchronized
- Duplicates : Not Allowed
- Null : Allowed

->When u have unique data and u want order preservation

->Extra operation to main order

TreeSet and Sorted Order

TreeSet stores unique elements in sorted (ascending) order using a tree structure (Red-Black tree). Nulls are not allowed, and all elements must be comparable. TreeSet is chosen when natural or custom sorting is needed.



```
        LinkedHashSet<Object> hs = new LinkedHashSet<Object>();
        hs.add("raju");
        hs.add("kaju");
        hs.add("maju");

        System.out.println(hs);

    }
}
```

TreeSet

- Insertion order: Ascending order
- Index supported: NO
- Random Access: NO
- Non Synchronized
- Duplicates : Not Allowed
- Null : Not Allowed
- Internal: RED BLACK TREE : complexity O(logN)
- Make it Generic while implementing
- ->When u have unique data and u want custom or ascending order(Default)

Comparing PriorityQueue and TreeSet

PriorityQueue allows duplicates and is based on a heap structure, while TreeSet only stores unique values and uses a tree. Both can provide sorted order, but their use cases differ based on data uniqueness and sorting needs.

Summary and Best Practices

Use HashSet for unique, unordered data; LinkedHashSet for unique, insertion-ordered data; and TreeSet for unique, sorted data. HashSet is fastest, LinkedHashSet maintains order with slight efficiency loss, and TreeSet sorts data but requires comparable elements.