Java Collections Framework

Title: Java Collections Framework

Subtitle: Exploring Lists, Sets, Maps, Iterators, and Streams

Presenter Name: Rama Shanker

Date: 23/01/2025

Agenda

Overview of the Java Collections Framework

Key Interfaces: List, Set, and Map

Iterators and Streams

Lab: Hands-On Exercises

CRUD Operations

Using Streams

Overview of Java Collections Framework

Definition: A standardized architecture to store and manipulate groups of objects.

Key Features:

Unified and efficient data manipulation

Dynamic resizing

Thread-safe variants

Core Interfaces - List

Definition: Ordered collection (sequence), allows duplicates.

Implementations:

ArrayList (dynamic array, fast read access)

LinkedList (doubly linked list, efficient insertion and deletion)

Vector (synchronized, legacy class)

Common Operations:

```
List<String> list = new ArrayList<>();
list.add("Apple");
list.get(0);
list.set(0, "Orange");
list.remove(0);
```

Core Interfaces - Set

Definition: Collection that does not allow duplicate elements.

Implementations:

HashSet (unordered, backed by hash table)

LinkedHashSet (ordered by insertion order)

TreeSet (sorted, backed by tree structure)

Key Features: Efficient lookups and unique elements.

Core Interfaces - Map

Definition: Key-value pairs where keys are unique.

Implementations:

HashMap (unordered, fast)

LinkedHashMap (ordered by insertion)

TreeMap (sorted by keys)

Hashtable (synchronized, legacy class)

Common Operations:

```
Map<String, Integer> map = new HashMap<>();
map.put("A", 1);
map.get("A");
map.remove("A");
```

Iterators

Definition: Object for traversing elements in a collection.

Common Types:

Iterator (supports all collections)

ListIterator (supports bidirectional traversal in List)

Example

```
Iterator<String> iterator = list.iterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next());
}
```

Streams

Definition: Functional, high-level abstraction for processing sequences of elements.

Common Operations:

Intermediate: filter(), map(), sorted()

Terminal: forEach(), collect(), reduce()

Example

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
numbers.stream()
        .filter(n -> n % 2 == 0)
        .forEach(System.out::println);
```

CRUD Operations

Objective: Practice Create, Read, Update, and Delete operations on collections.

Task:

Use ArrayList to store student names.

Add, retrieve, update, and delete entries.

Code Example:

```
List<String> students = new ArrayList<>();
students.add("Alice"); // Create
System.out.println(students.get(0)); // Read
students.set(0, "Bob"); // Update
students.remove(0); // Delete
```

Lab Exercise 2 - Using Streams

Objective: Apply stream operations for advanced data processing.

Task:

Filter even numbers from a list.

Sort names alphabetically.

Compute the sum of a list of numbers.

Code Example:

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
numbers.stream()
       .filter(n -> n % 2 == 0)
       .forEach(System.out::println);

List<String> names = Arrays.asList("Zoe", "Anna", "Bob");
names.stream()
     .sorted()
     .forEach(System.out::println);

int sum = numbers.stream().mapToInt(Integer::intValue).sum();
System.out.println("Sum: " + sum);
```

Conclusion

Summary:

Java Collections Framework provides a robust structure for data storage and manipulation.

Iterators and Streams simplify traversal and processing.