Java Backend Development Live-85

lecture-4

Class Agenda

- Java Collections Framework.
- Important Interfaces and their implementation.
- equals() and hashcode()
- HashMap
- Generics in Java
- Comparable vs Comparator
- Functional Interfaces and Lambda Expressions

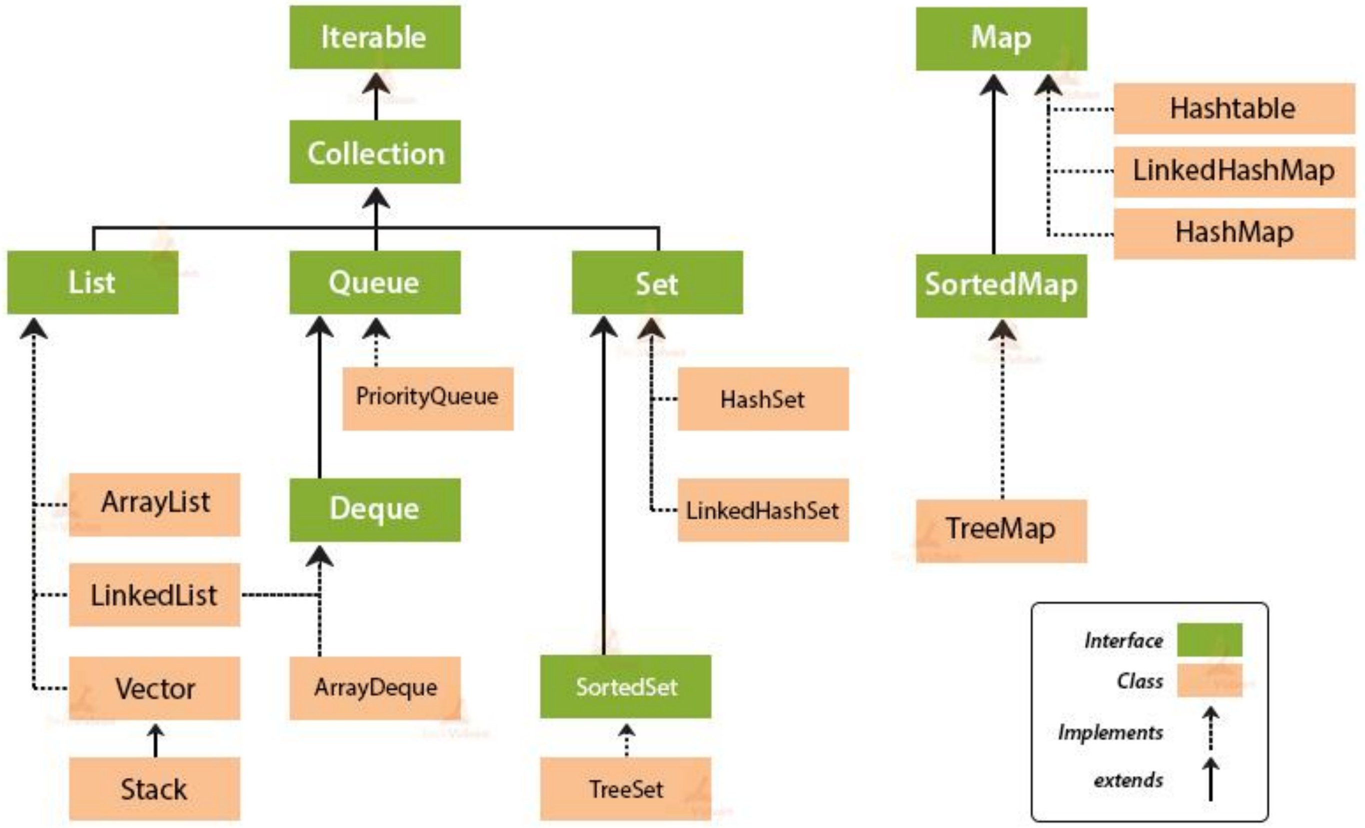
Java Collections Framework

- The Java language API provides many of the data structures from this framework for you.
- It defines a “collection” as “an object that represents a Group of elements (references to objects) It is not specified whether they are Ordered / not ordered Duplicated / not duplicated”.
- It defines a collections framework as “**a unified architecture for representing and manipulating collections,** allowing them to be manipulated independent of the details of their representation.”

Why Java Collection Framework ?

- Provides useful data structures and algorithms.
- Decreases extra effort required to learn, use, and design new API’s
- Supports reusability of standard data structures and algorithms

Collection Framework Hierarchy in Java



Keyword Analyzer Code

Develop keyword analyzer code with following basic features. This code can be used in any App providing search functionality based on keyword.

- Record keywords.
- Return list of all the recorded keywords.

List

A List is an ordered Collection of elements which may contain duplicates. It is an interface that extends the Collection interface. Lists are further classified into the following:

- ArrayList
- LinkedList
- Vectors

List implementations

ArrayList	**LinkedList**
Random Access: get(n) Constant time O(1)	Random Access: get(n) Linear time O(n)
Insert (beginning) and delete while iterating Linear time O(n)	Insert (beginning) and delete while iterating Constant time O(1)

Vectors: Similar to ArrayList but these are synchronized.

Practical uses of List

- Listing of product on amazon/flipkart.
- Listing of jobs on naukri.com
- Listing of questions of GeeksForGeeks

Queue

Queue in Java follows a FIFO approach i.e. it orders the elements in First In First Out manner. In a queue, the first element is removed first and last element is removed in the end.

Important methods:

- add() / offer()
- poll()
- peek()

Queue implementations

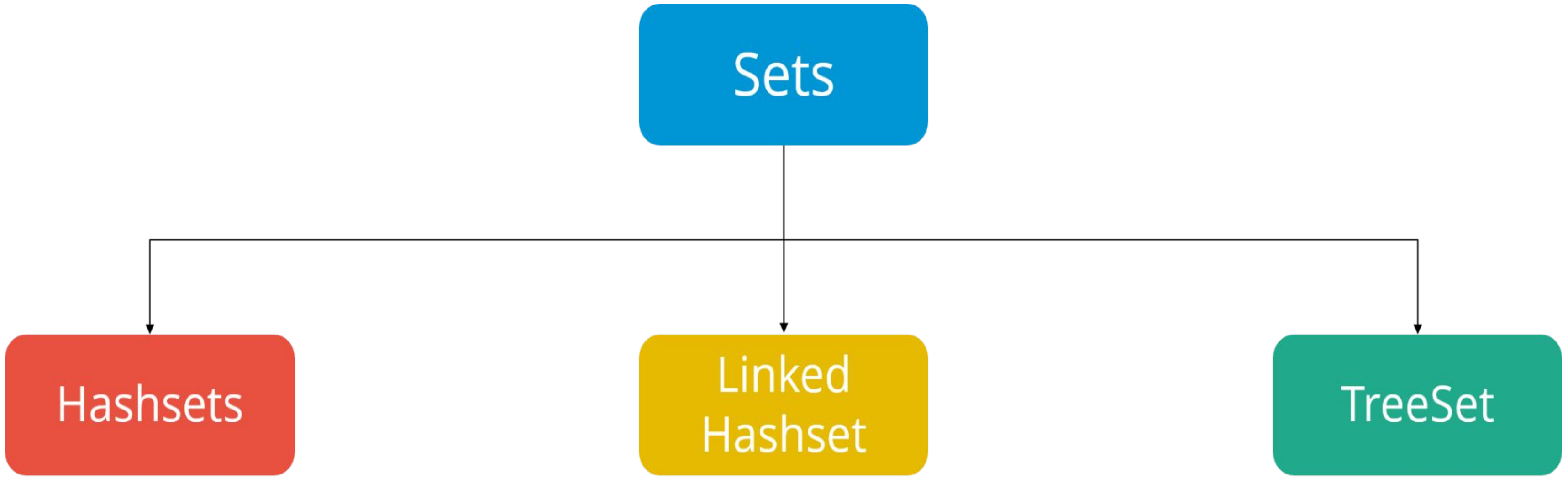
- **LinkedList**
  - head is the first element of the list
  - FIFO: Fist-In-First-Out
- **PriorityQueue**
  - The elements are ordered according to their natural ordering, or by a Comparator provided at the queue construction time

Practical uses of Queue

- Queue on ticket counter, waiting or RAC queue.
- Breadth First Search(BFS): Shortest distance between nodes.
- Level order traversal of tree.

Set Interface

A Set refers to a collection that cannot contain duplicate elements.



equals() and hashcode()

- equals() and hashCode() are **bound together by a joint contract** that specifies if two objects are considered equal using the equals() method, then they must have identical hashcode values.
- To be truly safe:
  - If override equals(), override hashCode()
  - Objects that are equals have to return identical hashcodes

Set implementations

- HashSet implements Set.
  - Hash tables as internal data structure (faster)
- LinkedHashSet extends HashSet
  - Elements are traversed by iterator according to the insertion order.
- TreeSet implements SortedSet.
  - R-B trees as internal data structure (computationally expensive)

Practical uses of Set

Find unique visitors.

Check username already exist.

Map Interface

- Data is stored in key-value pairs and every key is unique. Each key maps to a value hence the name map.
- Designed for the faster lookups.
- Analogous to Set.

Map implementations

- HashMap implements Map
  - No order
- LinkedHashMap extends HashMap
  - Insertion order
- TreeMap implements SortedMap
  - Ascending key order

Practical uses of Map

- Total hits on GeeksForGeeks country wise.
- API Rate Limiting.
- Find frequency of all char in a String.

Arrays and Collections class

Arrays and Collections classes provides several static methods that can be used to perform many tasks directly on arrays and collections.

- Fill an array/collection with a particular value.
- Sort an Arrays/Collections.
- Search in an Arrays/Collections.
- And many more.

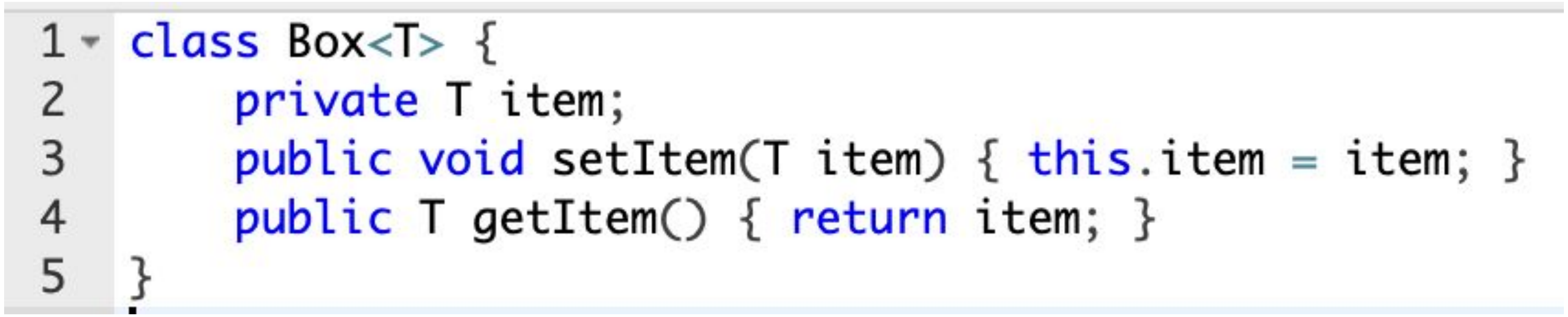
How HashMap Works?

- **Hashing**:
  - Converts a key into a hash code using hashCode().
  - Hash code is used to find the bucket index.
- **Buckets**:
  - HashMap stores entries in an array of buckets.
  - Each bucket is a LinkedList or TreeNode (from Java 8 onwards).
- **Put Operation**:
  - Step 1: Compute hash code for the key.
  - Step 2: Locate the bucket index.
  - Step 3: Check for collisions (entries with the same hash code).
  - Step 4: Store the entry (key-value pair).

- **Get Operation**:
    - Step 1: Compute the hash code for the key.
    - Step 2: Locate the bucket.
    - Step 3: Search for the key in the bucket.
- **Collisions Handling (Chaining):**
    - Colliding entries are stored in a LinkedList or a tree structure (Java 8+ optimizes using TreeMap if the list size exceeds a threshold).

Generics in Java

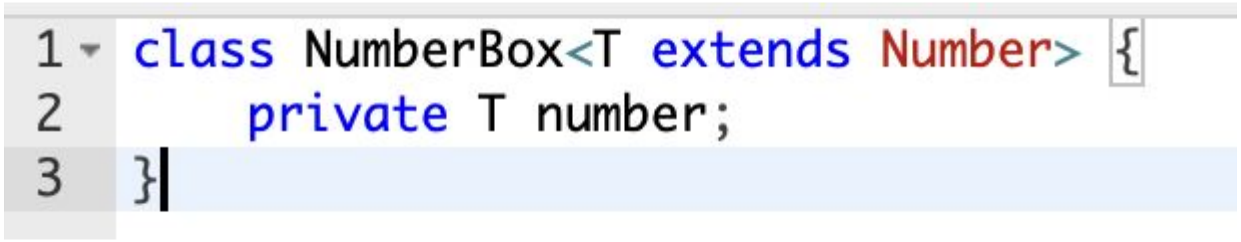
- **Type Safety**: Ensures compile-time type checking.
- Allow classes, interfaces, and methods to operate on a **type parameter** (like `T`, `E`, `K`, `V`).
- **Code Reusability**: Write a single method or class that works with any type.



```java
class Box<T> {
    private T item;
    public void setItem(T item) { this.item = item; }
    public T getItem() { return item; }
}
```

Bounded Generics

- Restrict the types that can be used as type parameters.
- `T extends Class`: Accepts `Class` or its subclasses
- Example:



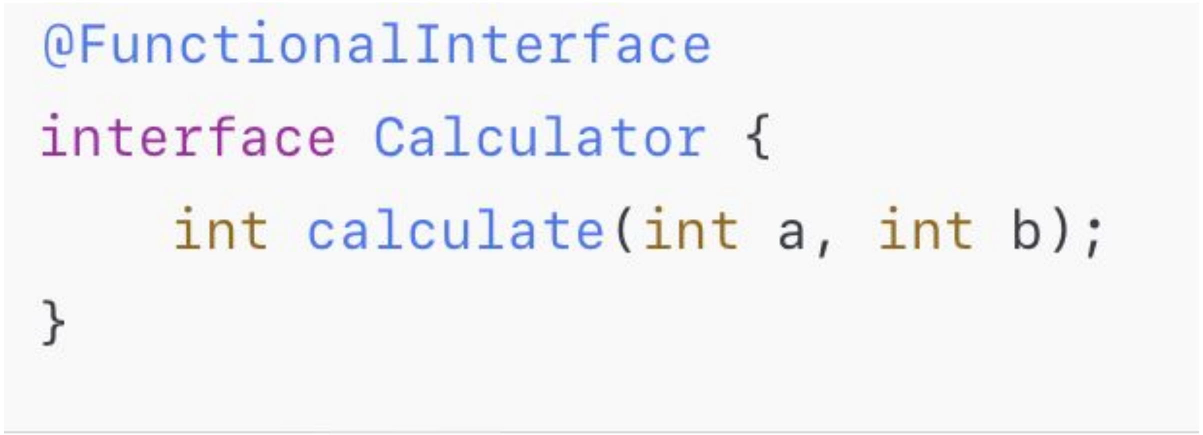
```java
class NumberBox<T extends Number> {
    private T number;
}
```

Comparable vs Comparator

- **Comparable** Interface is used to define the natural ordering of objects.
  - Used when the class itself defines the comparison logic.
  - Single sorting sequence.
  - Method: `int compareTo(T o)`
- **Comparator** Interface is used to define custom sorting logic.
  - Allows multiple sorting sequences.
  - Externalizes the comparison logic.
  - Method: `int compare(T o1, T o2)`

Functional Interfaces

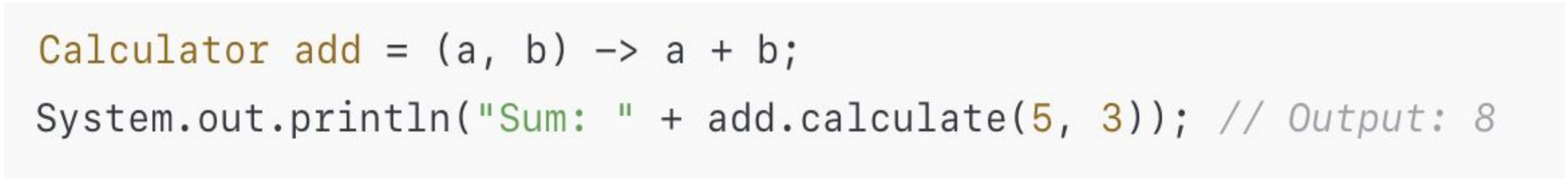
- Introduced in **Java 8**.
- An interface with **exactly one abstract method**.
- Enables the use of **lambda expressions**.
- Annotated with @FunctionalInterface (optional but recommended).



```java
@FunctionalInterface
interface Calculator {
    int calculate(int a, int b);
}
```

Lambda Expression

- A **concise way to write anonymous functions**.
- Introduced in **Java 8** to support functional programming.
- **Simplified Syntax**: No need for boilerplate code.
- **Improves Readability**: Cleaner code for functional interfaces.



```java
Calculator add = (a, b) -> a + b;
System.out.println("Sum: " + add.calculate(5, 3)); // Output: 8
```