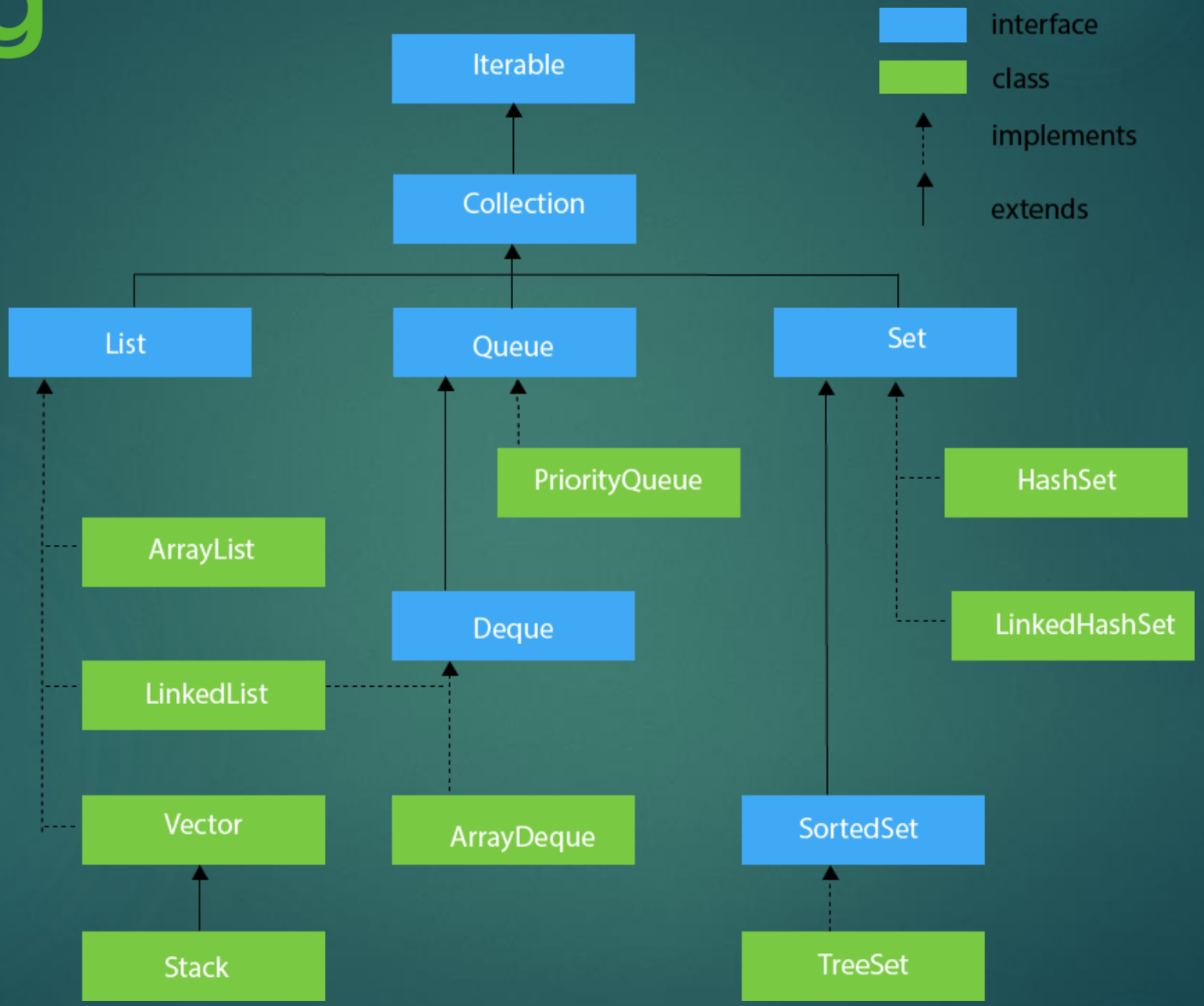
spring®

EraaSoft




By Eslam Khder


# Set

In Java, both `HashSet` and `LinkedHashSet` implement the `Set` interface and are used to store unique elements. However, there are some key differences between the two in terms of ordering and performance characteristics.

By Eslam Khder

# Set

## 1. Ordering of Elements:

- `HashSet` : Does **not guarantee any specific order** of elements. It stores elements in an unordered manner. Internally, it uses a hash table, which doesn't maintain the order in which elements are inserted.
- `LinkedHashSet` : Maintains the **insertion order** of elements. It uses a combination of a hash table (like `HashSet` ) and a linked list to maintain the order in which elements were added to the set. This means the order of iteration will be the same as the order of insertion.

A **HashTable** in Java is a data structure that stores key-value pairs and allows efficient data retrieval based on the key. It is part of the **Java Collections Framework** and is found in the `java.util` package.

By Eslam Khder


# Set

## 2. Performance:

- `HashSet` : Typically, **faster** than `LinkedHashSet` for operations like `add()` , `remove()` , and `contains()` because it does not maintain any ordering, thus it has less overhead.
- `LinkedHashSet` : Slightly **slower** than `HashSet` because it maintains the linked list to preserve the insertion order, which adds some overhead to the operations. However, the difference is generally minimal unless you're working with large sets.


# Set

### 3. Memory Overhead:

- `HashSet` : Has less memory overhead since it only stores the hash table.
- `LinkedHashSet` : Uses more memory because it also maintains a linked list to preserve the order.

### 4. Use Case:

- `HashSet` : When you do not care about the order of elements and only care about uniqueness and fast access, `HashSet` is ideal.
- `LinkedHashSet` : When you need to maintain the order of insertion but still want the performance benefits of a hash-based set, `LinkedHashSet` is more appropriate.


# Set

## Summary of Differences:

| Feature | `HashSet` | `LinkedHashSet` |
|---|---|---|
| **Order** | No guaranteed order | Maintains insertion order |
| **Performance** | Faster (due to no linked list) | Slower (due to linked list overhead) |
| **Memory Overhead** | Lower (no linked list) | Higher (due to linked list) |
| **Use Case** | When order is not important | When order matters (insertion order) |

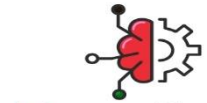
By Eslam Khder

# Vector


A `Vector` in Java is a **resizable array-based** data structure that implements the **List interface** and is **synchronized**, meaning it is **thread-safe** for concurrent access.

By Eslam Khder

# Vector



## Key Features of `Vector`

1. **Dynamic Resizing:**
   - Unlike arrays, a `Vector` **automatically grows** when needed.
   - Default growth: **Doubles in size** when capacity is exceeded.
2. **Thread-Safe (Synchronized):**
   - Methods are **synchronized**, meaning only one thread can modify it at a time.
   - This makes it **slower** than `ArrayList` in a single-threaded environment.
3. **Implements `List` & Uses an Array:**
   - Internally, `Vector` uses an **array** for storage, similar to `ArrayList`.

By Eslam Khder

# Vector

🔹 `Vector` **vs.** `ArrayList`

| Feature | `Vector` 🚀 | `ArrayList` 🏎️ |
|---|---|---|
| **Thread-Safe?** | ✅ Yes (Synchronized) | ❌ No (Not Synchronized) |
| **Performance** | 🔻 Slower | ✅ Faster |
| **Growth Strategy** | Doubles in size | Increases by 50% |
| **Use Case** | Multi-threaded apps | Single-threaded apps |

By Eslam Khder


# PriorityQueue

## PriorityQueue in Java

A **PriorityQueue** in Java is a special type of queue where elements are processed based on their priority instead of the order they were added (FIFO).

By Eslam Khder



# ArrayDeque

ArrayDeque  is a resizable, array-based implementation of the Deque interface in Java. It allows elements to be added or removed from both ends (front and back), making it more versatile than Stack and Queue.

By Eslam Khder

# ArrayDeque

## Key Features of `ArrayDeque`

- **Implements `Deque` Interface** → Supports operations at both ends.
- **No Capacity Restrictions** → Automatically resizes when full.
- **Does Not Allow Null Elements** → Unlike some other collections.

By Eslam Khder


| Feature | `HashMap` | `Hashtable` |
| --- | --- | --- |
| **Thread Safety** | Not thread-safe (must synchronize manually if needed) | Thread-safe (synchronized methods) |
| **Performance** | Faster (no synchronization overhead) | Slower (due to synchronization) |

By Eslam Khder