



# Java Collections and Maps

By Kenneth Rodriguez

# What are Collections and Maps?

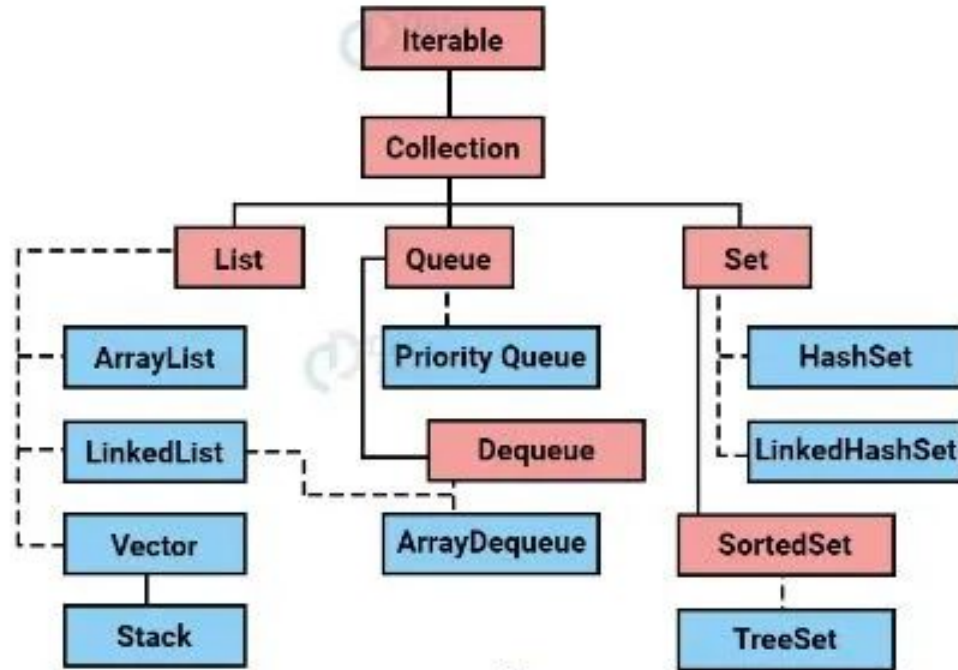
Collections and Maps are interfaces in the core Java library that allow developers to use data structures without building them from scratch. These sets of classes and interfaces are designed to represent and manipulate groups of objects.

**Collections**, for instance, consist of data structures that contain a list or set of objects, along with methods for searching, sorting, and other common operations.

**Maps**, on the other hand, store key–value pairs, where each key is unique. This allows the map to associate data with a unique identifier.

Knowing when and how to implement these interfaces is an important skill, because they allow developers to perform more complex algorithms.

# Collections

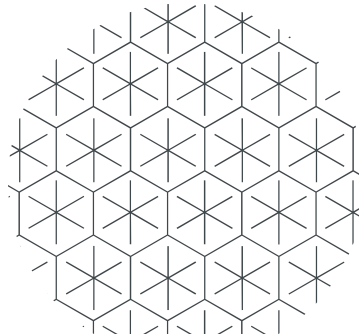


# ArrayList

The ArrayList is similar to a regular array, with the main difference that ArrayList can add new objects to the list, as well as remove them.

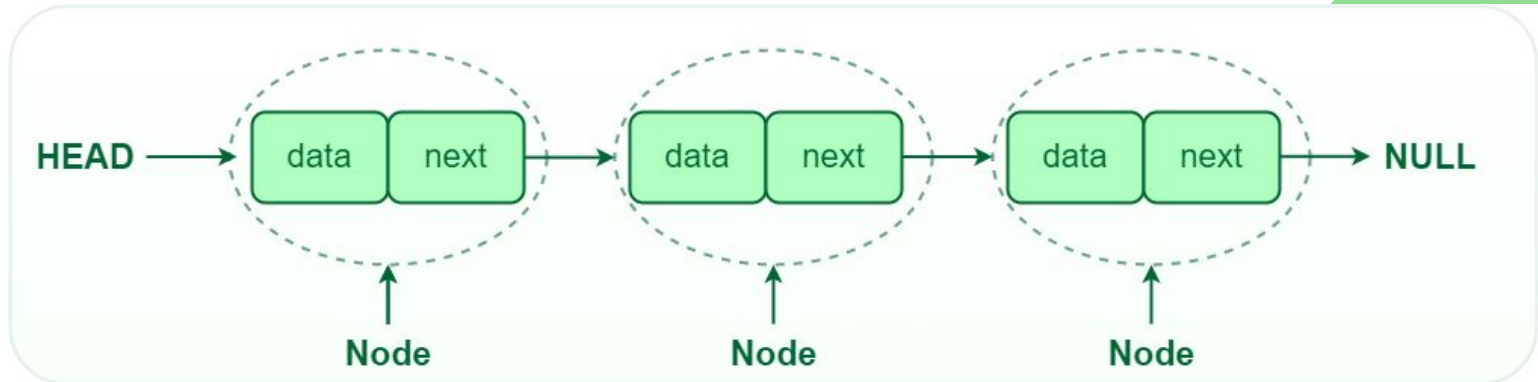
An ArrayList can contain repeating objects, and these objects have an order.

This class is perfect if you want to have random access to any element in the list.



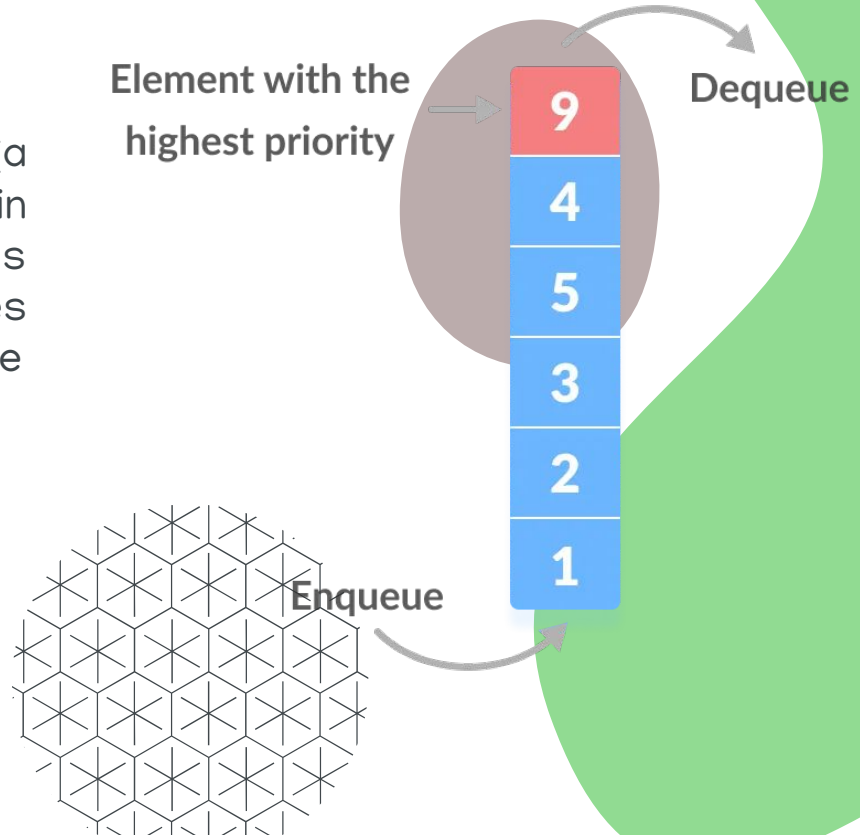
# LinkedList

LinkedList is a list where objects in the list can't be accessed directly. Instead, the LinkedList points toward the first item in the list. This first item then provides the reference of the next item, and so on.



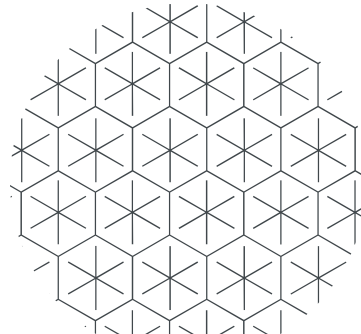
# PriorityQueue

A PriorityQueue is similar to the standard queue (a “First-In-First-Out” structure). The main difference is that each item in a PriorityQueue has an associated value or “priority” which determines the order in which items are removed from the queue.



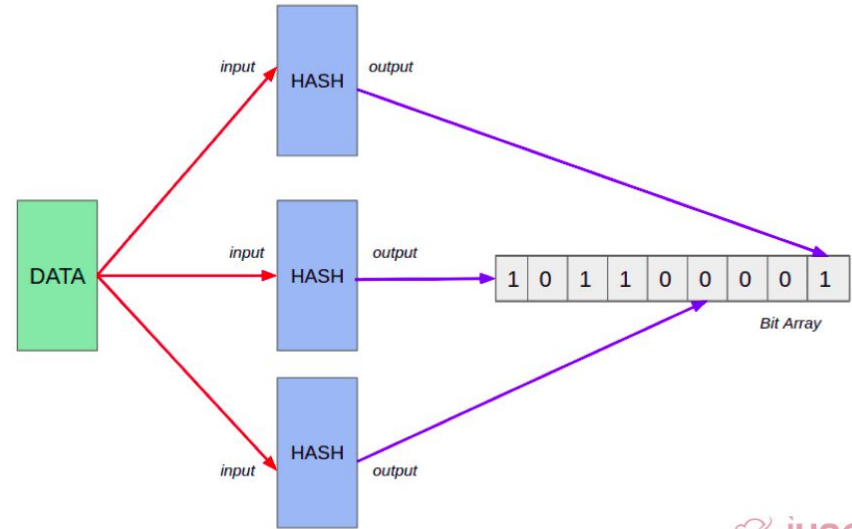
# ArrayDeque

An ArrayDeque is akin to a combination between an Array and a Queue. In a traditional Queue, items can only be appended at the end of the structure. However, in an ArrayDeque, items can be appended from either side of the queue, and they can also be removed from either of them.



# HashSet

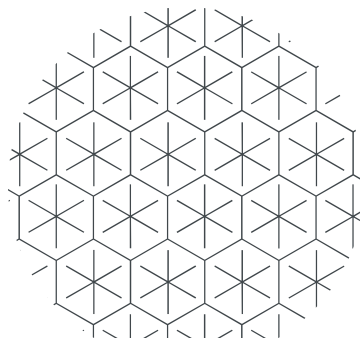
A HashSet is a set (meaning that items cannot be repeated) that implements a hash table. A HashSet is an efficient implementation of a Set, but the hash table implementation means that there is a small chance of collision, or two different values being interpreted as the same, and one of them would therefore not be included in the resulting Set.





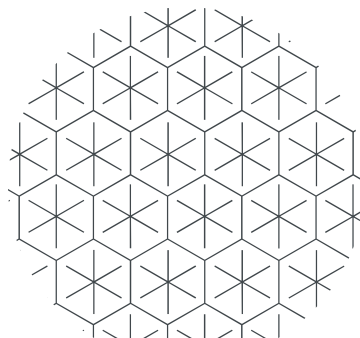
# LinkedHashSet

A LinkedHashSet is essentially an ordered Set (sorted by insertion), allowing the structure to preserve the order in which new values are added to the Set.

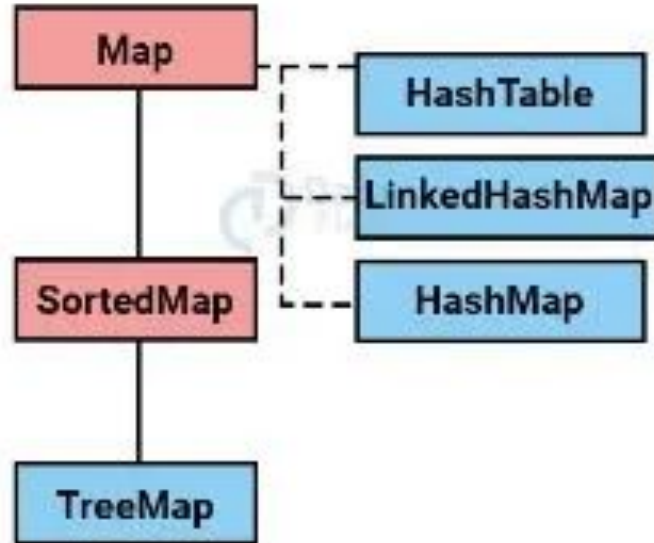


# TreeSet

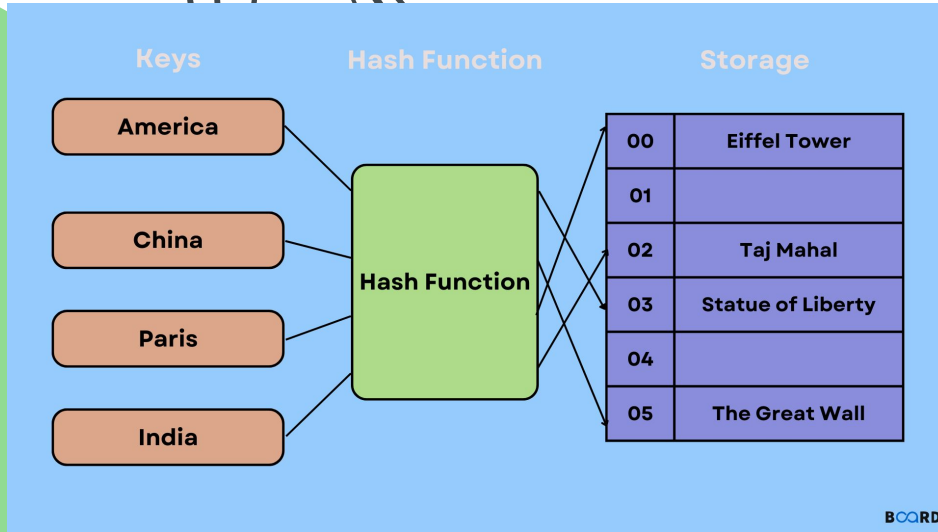
A TreeSet is an ordered Set, meaning that values cannot be repeated, but they are all sorted inside this data structure.



# Maps



# HashMap

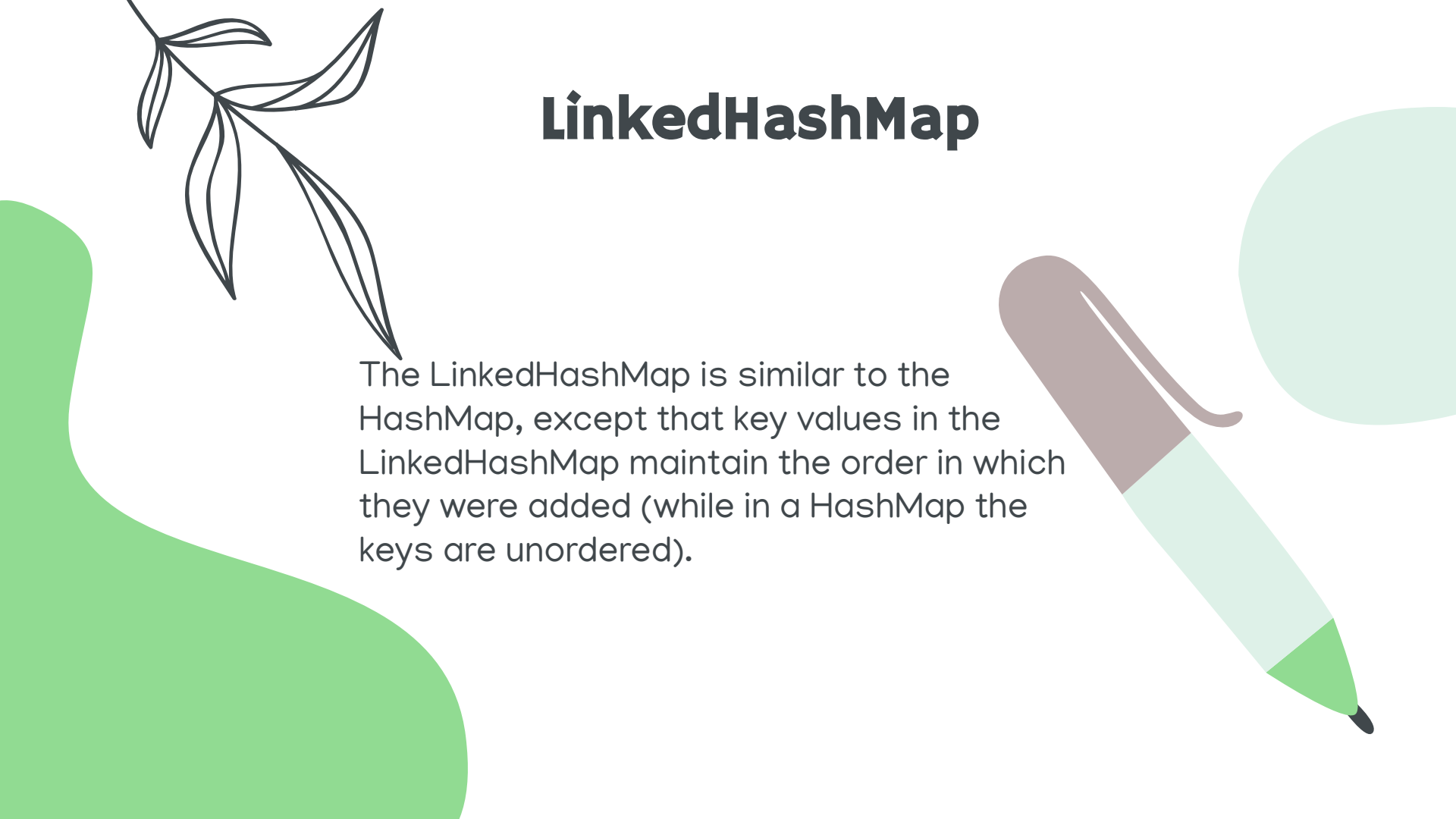


A HashMap is the most common implementation of the Map interface. It provides an efficient lookup time for key values.



# LinkedHashMap



The LinkedHashMap is similar to the HashMap, except that key values in the LinkedHashMap maintain the order in which they were added (while in a HashMap the keys are unordered).





# HashTable

A HashTable is very similar to the HashMap. However, the main difference is that a HashTable is thread safe, which means that it can be shared with several threads and parallel processes.





# TreeMap

A TreeMap is similar to the LinkedHashMap, because they both have ordered key values. However, while the LHM sorts keys by insertion order, the TreeMap sorts keys by value.