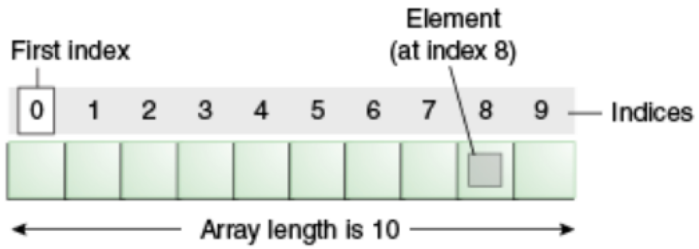
Arrays

Java Array

- Normally, array is a collection of similar type of elements that have contiguous memory location.
- **Java array** is an object the contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.
- Array in java is index based, first element of the array is stored at 0 index.



- Advantage of Java Array
  - ✓ **Code Optimization:** It makes the code optimized, we can retrieve or sort the data easily.
  - ✓ **Random access:** We can get any data located at any index position.
- Disadvantage of Java Array
  - ➢ **Size Limit:** We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

Types of Array in java

- There are two types of array.
  1. Single Dimensional Array
  2. Multidimensional Array

Single Dimensional Array in java

- Syntax to Declare an Array in java

  dataType arr[];

- Instantiation of an Array in java

  ArrayRefVar=**new** datatype[size];

Declaration, Instantiation and Initialization of Java Array

- We can declare, instantiate and initialize the java array together by:

```
int a[]={33,3,4,5};//declaration, instantiation and initialization
```

Multidimensional array in java

- In such case, data is stored in row and column based index (also known as matrix form).

- Syntax to Declare Multidimensional Array in java

  dataType[][] arrayRefVar; (or)

  dataType [][]arrayRefVar; (or)

  dataType arrayRefVar[][]; (or)

  dataType []arrayRefVar[];

Example to instantiate Multidimensional Array in java

- **int**[][] arr=**new int**[3][3];//3 row and 3 column

Collections

What is collections?

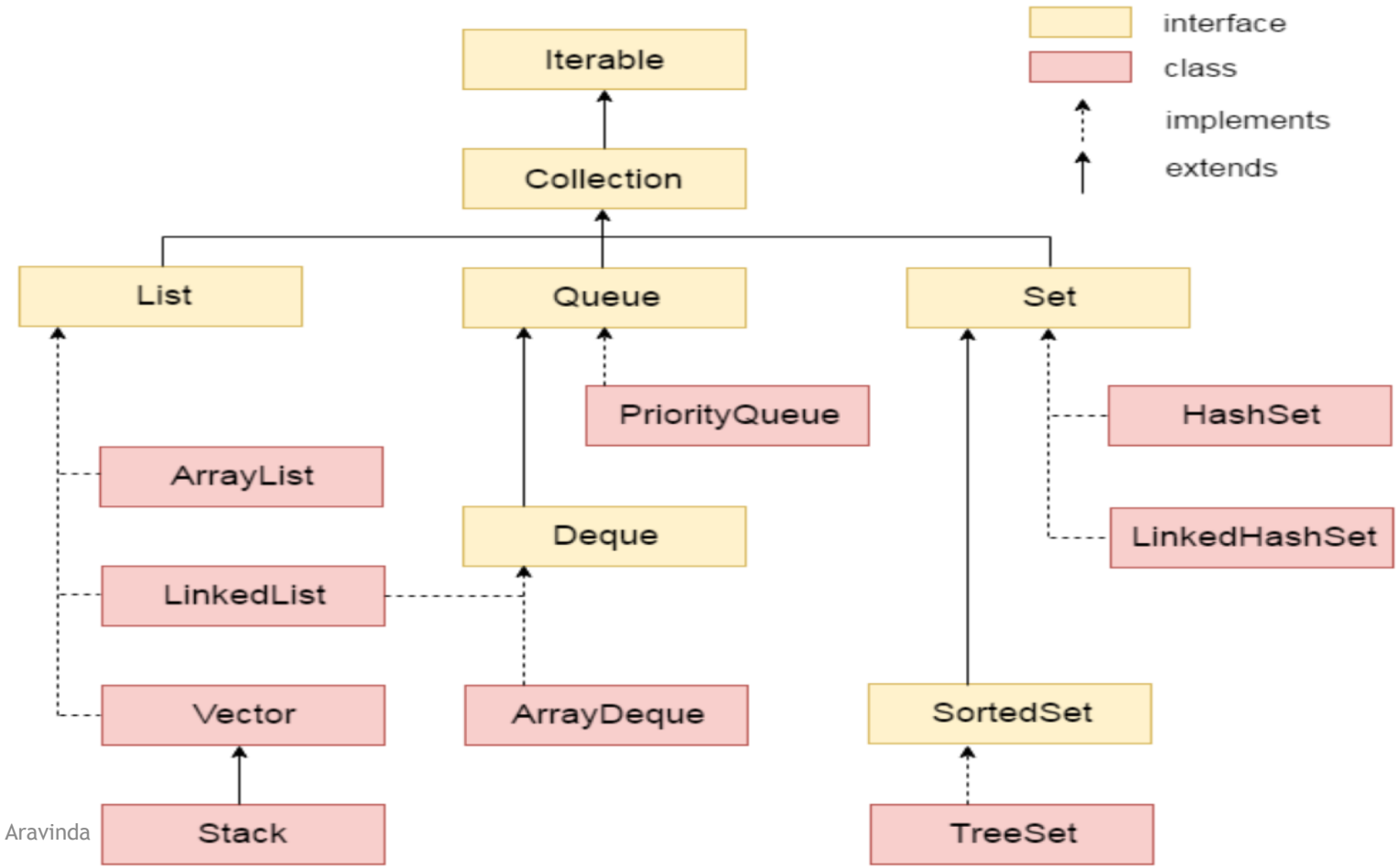
- It is Dynamic Data Structure
- It added after JAVA 1.5
- **Collections in java** is a framework that provides an architecture to store and manipulate the group of objects.
- All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

What is collections?

- Java Collection simply means a single unit of objects.
- Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

Hierarchy of Collection Framework

- Let us see the hierarchy of collection framework.The **java.util** package contains all the classes and interfaces for Collection framework.



Methods of Collection interface

No.	Method	Description
1	public boolean add(Object element)	is used to insert an element in this collection.
2	public boolean addAll(Collection c)	is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	is used to delete an element from this collection.
4	public boolean removeAll(Collection c)	is used to delete all the elements of specified collection from the invoking collection.
5	public boolean retainAll(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
6	public int size()	return the total number of elements in the collection.
7	public void clear()	removes the total no of element from the collection.
8	public boolean contains(Object element)	is used to search an element.
9	public boolean containsAll(Collection c)	is used to search the specified collection in this collection.
10	public Iterator iterator()	returns an iterator.
11	public Object[] toArray()	converts collection into array.
12	public boolean isEmpty()	checks if collection is empty.
13	public boolean equals(Object element)	matches two collection.
14	public int hashCode()	returns the hashcode number for collection.

Iterator interface

- Iterator interface provides the facility of iterating the elements in forward direction only.

Methods of Iterator interface

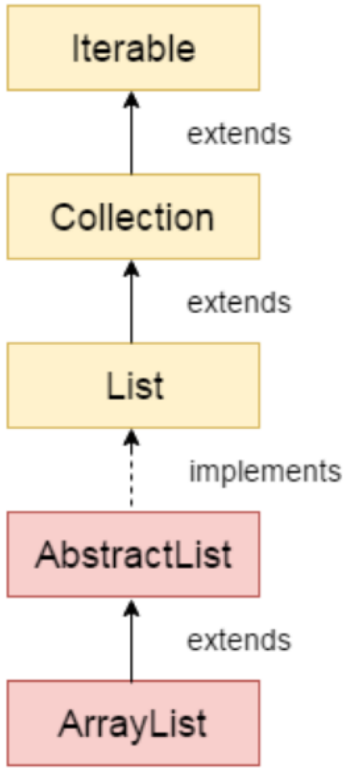
There are only three methods in the Iterator interface. They are:

No.	Method	Description
1	public boolean hasNext()	It returns true if iterator has more elements.
2	public Object next()	It returns the element and moves the cursor pointer to the next element.
3	public void remove()	It removes the last elements returned by the iterator. It is rarely used.

Java ArrayList class

- Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.
  - Java ArrayList class can contain duplicate elements.
  - Java ArrayList class maintains insertion order.
  - Java ArrayList class is non synchronized.
  - Java ArrayList allows random access because array works at the index basis.
  - In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

Hierarchy of ArrayList class



Java List Interface

- List Interface is the subinterface of Collection.It contains methods to insert and delete elements in index basis.It is a factory of ListIterator interface.
- It is the child interface of Collection.
- If we want to represent a group of individual objects as a single entity where duplicates are allowed and insertion order is must be preserved then we should go for list.
- We can differentiate duplicates by using index.
- We can preserve insertion order by using index, hence index play very important role in the list interface.

public interface List<E> **extends** Collection<E>

Methods of Java List Interface

Method	Description
void add(int index,Object element)	It is used to insert element into the invoking list at the index passed in the index.
boolean addAll(int index,Collection c)	It is used to insert all elements of c into the invoking list at the index passed in the index.
object get(int index)	It is used to return the object stored at the specified index within the invoking collection.
object set(int index,Object element)	It is used to assign element to the location specified by index within the invoking list.
object remove(int index)	It is used to remove the element at position index from the invoking list and return the deleted element.
ListIterator listIterator()	It is used to return an iterator to the start of the invoking list.
ListIterator listIterator(int index)	It is used to return an iterator to the invoking list that begins at the specified index.

Java ListIterator Interface

- ListIterator Interface is used to traverse the element in backward and forward direction.

- ListIterator Interface declaration

  public interface ListIterator<E> **extends** Iterator<E>

Methods of Java ListIterator Interface:

Method	Description
boolean hasNext()	This method return true if the list iterator has more elements when traversing the list in the forward direction.
Object next()	This method return the next element in the list and advances the cursor position.
boolean hasPrevious()	This method return true if this list iterator has more elements when traversing the list in the reverse direction.
Object previous()	This method return the previous element in the list and moves the cursor position backwards.

Java Map Interface

- A map contains values on the basis of key i.e. key and value pair. Each key and value pair is known as an entry. Map contains only unique keys.
- Map is useful if you have to search, update or delete elements on the basis of key.

Useful methods of Map interface

Method	Description
Object put(Object key, Object value)	It is used to insert an entry in this map.
void putAll(Map map)	It is used to insert the specified map in this map.
Object remove(Object key)	It is used to delete an entry for the specified key.
Object get(Object key)	It is used to return the value for the specified key.
boolean containsKey(Object key)	It is used to search the specified key from this map.
Set keySet()	It is used to return the Set view containing all the keys.
Set entrySet()	It is used to return the Set view containing all the keys and values.

Map.Entry Interface

- Entry is the sub interface of Map. So we will be accessed it by Map.Entry name. It provides methods to get key

Methods of Map.Entry interface

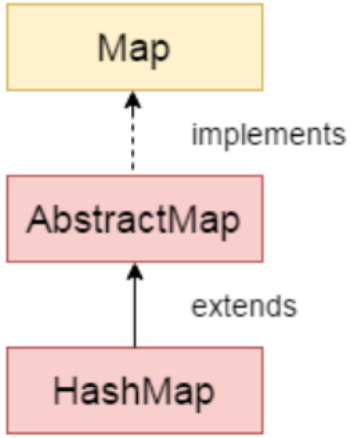
Method	Description
Object getKey()	It is used to obtain key.
Object getValue()	It is used to obtain value.

Java HashMap class

- Java HashMap class implements the map interface by using a hashtable. It inherits AbstractMap class and implements Map interface.

The important points about Java HashMap class are:

1. A HashMap contains values based on the key.
2. It contains only unique elements.
3. It may have one null key and multiple null values.
4. It maintains no order.



HashMap class declaration

Let's see the declaration for java.util.HashMap class.

```
public class HashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, Cloneable, Serializable
```

HashMap class Parameters

Let's see the Parameters for java.util.HashMap class.

- **K**: It is the type of keys maintained by this map.
- **V**: It is the type of mapped values.

Constructors of Java HashMap class

Constructor	Description
HashMap()	It is used to construct a default HashMap.
HashMap(Map m)	It is used to initializes the hash map by using the elements of the given Map object m.
HashMap(int capacity)	It is used to initializes the capacity of the hash map to the given integer value, capacity.
HashMap(int capacity, float fillRatio)	It is used to initialize both the capacity and fill ratio of the hash map by using its arguments.

Methods of Java HashMap class

Method	Description
void clear()	It is used to remove all of the mappings from this map.
boolean containsKey(Object key)	It is used to return true if this map contains a mapping for the specified key.
boolean containsValue(Object value)	It is used to return true if this map maps one or more keys to the specified value.
boolean isEmpty()	It is used to return true if this map contains no key-value mappings.
Object clone()	It is used to return a shallow copy of this HashMap instance: the keys and values themselves are not cloned.
Set entrySet()	It is used to return a collection view of the mappings contained in this map.
Set keySet()	It is used to return a set view of the keys contained in this map.
Object put(Object key, Object value)	It is used to associate the specified value with the specified key in this map.
int size()	It is used to return the number of key-value mappings in this map.
Collection values()	It is used to return a collection view of the values contained in this map.

Java Hashtable class

- Java Hashtable class implements a hashtable, which maps keys to values. It inherits Dictionary class and implements the Map interface.

  The important points about Java Hashtable class are:

  1. A Hashtable is an array of list. Each list is known as a bucket. The position of bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key.
  2. It contains only unique elements.
  3. It may have not have any null key or value.
  4. It is synchronized.

HashMap class declaration

Let's see the declaration for java.util.HashMap class.

```
public class HashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, Cloneable, Serializable
```

HashMap class Parameters

Let's see the Parameters for java.util.HashMap class.

- **K**: It is the type of keys maintained by this map.
- **V**: It is the type of mapped values.

Constructors of Java Hashtable class

Constructor	Description
Hashtable()	It is the default constructor of hash table it instantiates the Hashtable class.
Hashtable(int size)	It is used to accept an integer parameter and creates a hash table that has an initial size specified by integer value size.
Hashtable(int size, float fillRatio)	It is used to create a hash table that has an initial size specified by size and a fill ratio specified by fillRatio.

Methods of Java Hashtable class

Method	Description
void clear()	It is used to reset the hash table.
boolean contains(Object value)	This method return true if some value equal to the value exist within the hash table, else return false.
boolean containsValue(Object value)	This method return true if some value equal to the value exists within the hash table, else return false.
boolean containsKey(Object key)	This method return true if some key equal to the key exists within the hash table, else return false.
boolean isEmpty()	This method return true if the hash table is empty; returns false if it contains at least one key.
void rehash()	It is used to increase the size of the hash table and rehashes all of its keys.
Object get(Object key)	This method return the object that contains the value associated with the key.
Object remove(Object key)	It is used to remove the key and its value. This method return the value associated with the key.
int size()	This method return the number of entries in the hash table.

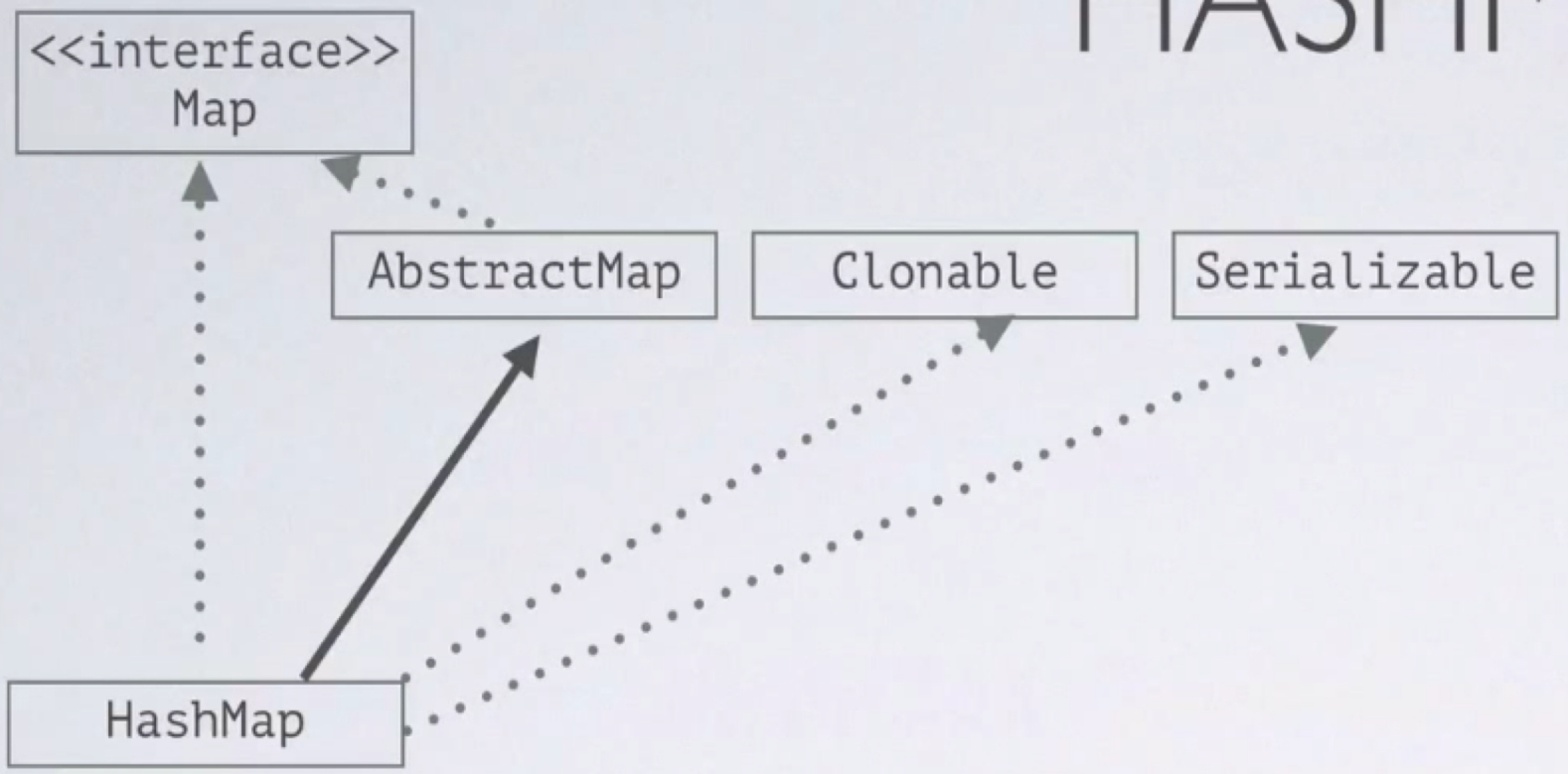
Difference between HashMap and Hashtable

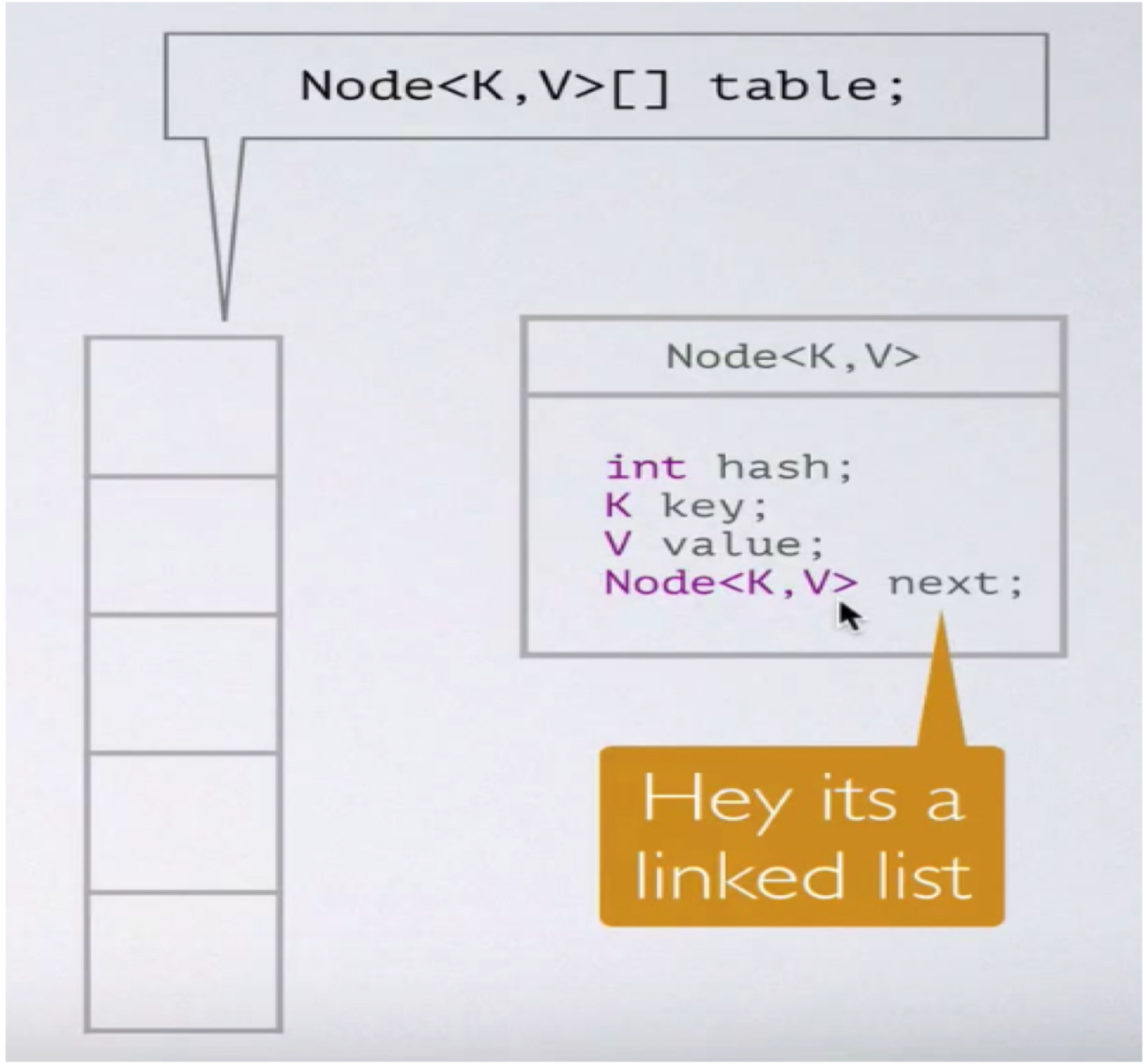
HashMap and Hashtable both are used to store data in key and value form. Both are using hashing technique to store unique keys.

But there are many differences between HashMap and Hashtable classes that are given below.

HashMap	Hashtable
1) HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code.	Hashtable is **synchronized**. It is thread-safe and can be shared with many threads.
2) HashMap **allows one null key and multiple null values**.	Hashtable **doesn't allow any null key or value**.
3) HashMap is a **new class introduced in JDK 1.2**.	Hashtable is a **legacy class**.
4) HashMap is **fast**.	Hashtable is **slow**.
5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap);	Hashtable is internally synchronized and can't be unsynchronized.
6) HashMap is **traversed by Iterator**.	Hashtable is **traversed by Enumerator and Iterator**.
7) Iterator in HashMap is **fail-fast**.	Enumerator in Hashtable is **not fail-fast**.
8) HashMap inherits **AbstractMap** class.	Hashtable inherits **Dictionary** class.

HASHMAP


<<interface>>
Map
AbstractMap
Clonable
Serializable
HashMap


Node<K,V>[] table;
Node<K,V>
int hash;
K key;
V value;
Node<K,V> next;
Hey its a linked list