# ArrayList

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.
* 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.
* Java ArrayList allows random access because array works at the index basis.

# LinkedList

Java LinkedList class uses doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

* Java LinkedList class can contain duplicate elements.
* Java LinkedList class maintains insertion order.
* Java LinkedList class is non synchronized.
* In Java LinkedList class, manipulation is fast because no shifting needs to be occurred.
* Java LinkedList class can be used as list, stack or queue.

# Difference between ArrayList and LinkedList

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses **dynamic array** to store the elements. | LinkedList internally uses **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses array. If any element is removed from the array, all the bits are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses doubly linked list so no bit shifting is required in memory. |
| 3) ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |

# Difference between List and Set

List can contain duplicate elements whereas Set contains unique elements only.

# HashSet

Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

* HashSet stores the elements by using a mechanism called **hashing.**
* HashSet contains unique elements only.

# LinkedHashSet

Java LinkedHashSet class is a Hash table and Linked list implementation of the set interface. It inherits HashSet class and implements Set interface.

* Contains unique elements only like HashSet.
* Provides all optional set operations, and permits null elements.
* Maintains insertion order.

LinkedHashSet<String> al = new LinkedHashSet<String>();

void clear()

**boolean contains(Object o)**

boolean add(Object o)

boolean isEmpty()

boolean remove(Object o)

Object clone()

Iterator iterator()

int size()

# TreeSet

Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements NavigableSet interface. The objects of TreeSet class are stored in ascending order.

* Contains unique elements only like HashSet.
* Access and retrieval times are quiet fast.
* Maintains ascending order.

TreeSet<String> al=new TreeSet<String>();

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean addAll(Collection c) | It is used to add all of the elements in the specified collection to this set. |
| boolean contains(Object o) | It is used to return true if this set contains the specified element. |
| boolean isEmpty() | It is used to return true if this set contains no elements. |
| boolean remove(Object o) | It is used to remove the specified element from this set if it is present. |
| void add(Object o) | It is used to add the specified element to this set if it is not already present. |
| void clear() | It is used to remove all of the elements from this set. |
| Object clone() | It is used to return a shallow copy of this TreeSet instance. |
| Object first() | It is used to return the first (lowest) element currently in this sorted set. |
| Object last() | It is used to return the last (highest) element currently in this sorted set. |
| int size() | It is used to return the number of elements in this set. |

# Queue

Java Queue interface orders the element in FIFO(First In First Out) manner. In FIFO, first element is removed first and last element is removed at last.

PriorityQueue<String> queue = new PriorityQueue<String>();

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean add(object) | It is used to insert the specified element into this queue and return true upon success. |
| boolean offer(object) | It is used to insert the specified element into this queue. |
| Object remove() | It is used to retrieves and removes the head of this queue. |
| Object poll() | It is used to retrieves and removes the head of this queue, or returns null if this queue is empty. |
| Object element() | It is used to retrieves, but does not remove, the head of this queue. |
| Object peek() | It is used to retrieves, but does not remove, the head of this queue, or returns null if this queue is empty. |

# ArrayDeque

The ArrayDeque class provides the facility of using deque and resizable-array. It inherits AbstractCollection class and implements the Deque interface.

* Unlike Queue, we can add or remove elements from both sides.
* Null elements are not allowed in the ArrayDeque.
* ArrayDeque is not thread safe, in the absence of external synchronization.
* ArrayDeque has no capacity restrictions.
* ArrayDeque is faster than LinkedList and Stack.

Deque<String> deque = **new** ArrayDeque<String>();

|  |  |
| --- | --- |
| boolean add(object) | It is used to insert the specified element into this deque and return true upon success. |
| boolean offer(object) | It is used to insert the specified element into this deque. |
| Object remove() | It is used to retrieves and removes the head of this deque. |
| Object poll() | It is used to retrieves and removes the head of this deque, or returns null if this deque is empty. |
| Object element() | It is used to retrieves, but does not remove, the head of this deque. |
| Object peek() | It is used to retrieves, but does not remove, the head of this deque, or returns null if this deque is empty. |

offerFirst() and pollLast()

# HashMap

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

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

### 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.

|  |  |
| --- | --- |
| **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. |

Difference between HashSet and HashMap

HashSet contains only values whereas HashMap contains entry(key and value).

**static int** sockMerchant(**int** arraySize, **int**[] array) {  
 HashMap <Integer, Integer> colors = **new** HashMap<Integer, Integer>();  
  
 **for** (**int** color : array) {  
 Integer frequency = colors.get(color);  
 **if** (frequency == **null**) {  
 colors.put(color, 1);  
 } **else** {  
 colors.put(color, frequency + 1);  
 }  
 }  
  
 **int** pairs = 0;  
  
 **for** (Integer frequency : colors.values()) {  
 pairs = pairs + (frequency / 2);  
 }  
  
 **return** pairs;

# LinkedHashMap

Java LinkedHashMap class is Hash table and Linked list implementation of the Map interface, with predictable iteration order. It inherits HashMap class and implements the Map interface.

* A LinkedHashMap contains values based on the key.
* It contains only unique elements.
* It may have one null key and multiple null values.
* It is same as HashMap instead maintains insertion order.

# LinkedHashMap class Parameters

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

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

 LinkedHashMap<Integer,String> hm = **new** LinkedHashMap<Integer,String>();

|  |  |
| --- | --- |
| Object get(Object key) | It is used to return the value to which this map maps the specified key. |
| void clear() | It is used to remove all mappings from this map. |
| boolean containsKey(Object key) | It is used to return true if this map maps one or more keys to the specified value. |

# TreeMap

Java TreeMap class implements the Map interface by using a tree. It provides an efficient means of storing key/value pairs in sorted order.

The important points about Java TreeMap class are:

* A TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
* It contains only unique elements.
* It cannot have null key but can have multiple null values.
* It is same as HashMap instead maintains ascending order.

# TreeMap class Parameters

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

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

TreeMap<Integer,String> hm=**new** TreeMap<Integer,String>();

|  |  |
| --- | --- |
| **Method** | **Description** |
| 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. |
| Object firstKey() | It is used to return the first (lowest) key currently in this sorted map. |
| Object get(Object key) | It is used to return the value to which this map maps the specified key. |
| Object lastKey() | It is used to return the last (highest) key currently in this sorted map. |
| Object remove(Object key) | It is used to remove the mapping for this key from this TreeMap if present. |
| void putAll(Map map) | It is used to copy all of the mappings from the specified map to this map. |
| Set entrySet() | It is used to return a set view of the mappings contained 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. |

# Difference between HashMap and Hashtable

|  |  |
| --- | --- |
| **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. |