Index:

collections

Array list

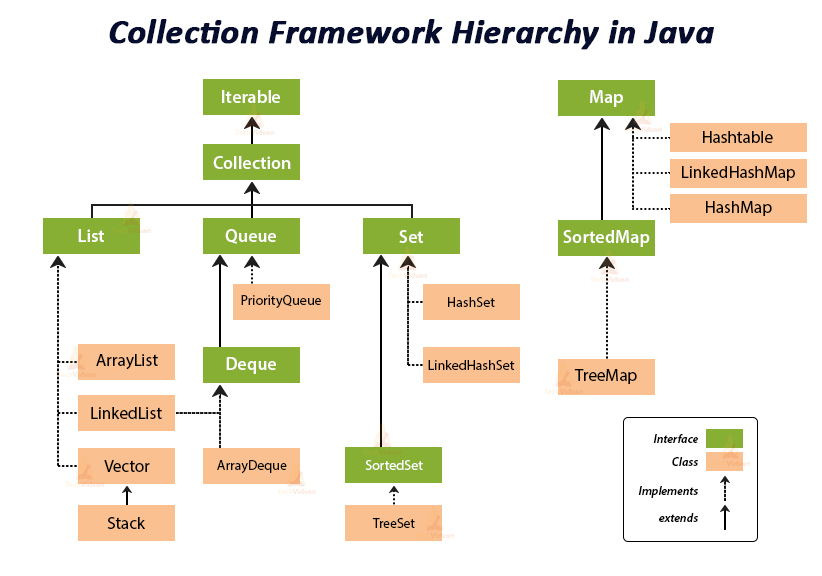
Linkedlist

hashSet

hashMap and internal working

concurrent hashmap

Collection framework:



Sure! Here are the examples with some important points for each data structure:

**ArrayList**

**Important Points:**

* **Dynamic Array**: Resizable array implementation.
* **Index-Based**: Allows random access using indices.
* **Performance**: Fast for get and set operations, slower for add and remove due to shifting elements.

import java.util.ArrayList;

public class ArrayListExample {

public static void main(String[] args) {

ArrayList<String> list = new ArrayList<>();

list.add("Apple");

list.add("Banana");

list.add("Cherry");

System.out.println("Element at index 1: " + list.get(1)); // Banana

list.remove(0); // Removes "Apple"

System.out.println("Updated list: " + list);

}

}

**LinkedList**

**Important Points:**

* **Doubly Linked List**: Each element points to the next and previous elements.
* **Performance**: Fast for add and remove operations, slower for get due to traversal.
* **Use Case**: Ideal for scenarios with frequent insertions and deletions.

import java.util.LinkedList;

public class LinkedListExample {

public static void main(String[] args) {

LinkedList<String> list = new LinkedList<>();

list.add("Dog");

list.add("Cat");

list.add("Rabbit");

System.out.println("Element at index 2: " + list.get(2)); // Rabbit

list.remove(1); // Removes "Cat"

System.out.println("Updated list: " + list);

}

}

**HashSet**

**Important Points:**

* **No Duplicates**: Ensures unique elements.
* **Hashing**: Uses hash table for storage.
* **Performance**: Fast for add, remove, and contains operations.

import java.util.HashSet;

public class HashSetExample {

public static void main(String[] args) {

HashSet<String> set = new HashSet<>();

set.add("Red");

set.add("Green");

set.add("Blue");

System.out.println("Set contains 'Green': " + set.contains("Green")); // true

set.remove("Red"); // Removes "Red"

System.out.println("Updated set: " + set);

}

}

**HashMap**

**Important Points:**

* **Key-Value Pairs**: Stores data in key-value pairs.
* **Hashing**: Uses hash table for storage.
* **Performance**: Fast for put, get, and remove operations.

import java.util.HashMap;

public class HashMapExample {

public static void main(String[] args) {

HashMap<Integer, String> map = new HashMap<>();

map.put(1, "One");

map.put(2, "Two");

map.put(3, "Three");

System.out.println("Value for key 2: " + map.get(2)); // Two

map.remove(1); // Removes key 1 and its value

System.out.println("Updated map: " + map);

}

}

Interview questions on collection frame work?

**Collection Framework Hierarchy**

mathematica

CopyEdit

Iterable

|

Collection

-----------------------------------

| | |

List Set Queue

| | |

ArrayList HashSet PriorityQueue

LinkedList TreeSet LinkedList

Vector

Stack

1. **Q: What is the Java Collection Framework? Why is it used?**  
   **A:**  
   The Java Collection Framework is a unified architecture for representing and manipulating collections. It provides classes like ArrayList, HashSet, and HashMap to store and process data efficiently. It simplifies data handling, improves code quality, and reduces development effort.
2. **Q: What is the difference between ArrayList and LinkedList?**  
   **A:**
   * **ArrayList**:
     + Backed by a dynamic array.
     + Faster for random access (get operations).
   * **LinkedList**:
     + Backed by a doubly linked list.
     + Faster for insertions and deletions in the middle.
3. **Q: What is the difference between List and Set?**  
   **A:**
   * **List**: Allows duplicates and maintains the insertion order.
   * **Set**: Does not allow duplicates and does not guarantee order (except LinkedHashSet and TreeSet).
4. **Q: Explain the difference between HashMap and TreeMap.**  
   **A:**
   * **HashMap**:
     + Stores key-value pairs in no specific order.
     + Allows one null key and multiple null values.
   * **TreeMap**:
     + Maintains keys in natural (ascending) order.
     + Does not allow null keys.
5. **Q: What is the difference between ArrayList and Vector?**  
   **A:**
   * **ArrayList**:
     + Not synchronized (not thread-safe).
     + Faster as it does not require synchronization.
   * **Vector**:
     + Synchronized (thread-safe).
     + Slower due to synchronization overhead.
6. **Q: How does a HashSet ensure uniqueness?**  
   **A:**  
   HashSet uses a HashMap internally. It calculates the hash code of objects and compares them to ensure no two objects have the same hash code or content.
7. **Q: What is the internal working of a HashMap?**  
   **A:**
   * A HashMap uses an array of buckets to store key-value pairs.
   * It calculates the hash code of the key to determine the bucket.
   * In case of collisions (same hash code), it uses a linked list or binary tree (from Java 8) to store multiple entries in a single bucket.
8. **Q: What is the difference between fail-fast and fail-safe iterators?**  
   **A:**
   * **Fail-fast**: Throws a ConcurrentModificationException if the collection is modified while iterating (e.g., ArrayList, HashMap).
   * **Fail-safe**: Does not throw an exception as it works on a copy of the collection (e.g., CopyOnWriteArrayList).
9. **Q: Explain the difference between Comparable and Comparator.**  
   **A:**
   * **Comparable**:
     + Used to define natural ordering of objects.
     + Implemented in the class itself via compareTo().
   * **Comparator**:
     + Used for custom ordering.
     + Implemented as a separate class or lambda via compare().

**Example**:

java

CopyEdit

import java.util.\*;

class Student implements Comparable<Student> {

int id;

String name;

public Student(int id, String name) {

this.id = id;

this.name = name;

}

public int compareTo(Student other) {

return this.id - other.id;

}

}

public class Main {

public static void main(String[] args) {

List<Student> students = new ArrayList<>();

students.add(new Student(2, "Alice"));

students.add(new Student(1, "Bob"));

Collections.sort(students);

}

}

4,what is difference of linked list and arraylist?

A,

| **Aspect** | **List** | **ArrayList** |
| --- | --- | --- |

5,what is hash map vs treemap?  
a,

| **Aspect** | **HashMap** | **TreeMap** |
| --- | --- | --- |
| **Data Structure** | Uses a hash table. | Uses a red-black tree (self-balancing binary tree). |
| **Ordering** | No guarantee of order (entries may appear unordered). | Maintains entries in **sorted order** by keys. |
| **Null Keys** | Allows **one null key** and multiple null values. | Does **not allow null keys**, but allows null values. |
| **Performance** | Faster for most operations: O(1) for put/get. | Slower: O(log n) for put/get due to tree traversal. |
| **Use Case** | Suitable for quick lookups without concern for order. | Suitable when sorted order of keys is required. |

**Example: HashMap**

java

Copy code

import java.util.HashMap;

public class HashMapExample {

public static void main(String[] args) {

HashMap<Integer, String> hashMap = new HashMap<>();

hashMap.put(3, "Three");

hashMap.put(1, "One");

hashMap.put(2, "Two");

hashMap.put(null, "Null Key");

System.out.println("HashMap:");

for (Integer key : hashMap.keySet()) {

System.out.println("Key: " + key + ", Value: " + hashMap.get(key));

}

}

}

**Output** (Order may vary):

yaml

Copy code

HashMap:

Key: null, Value: Null Key

Key: 1, Value: One

Key: 2, Value: Two

Key: 3, Value: Three

**Example: TreeMap**

java

Copy code

import java.util.TreeMap;

public class TreeMapExample {

public static void main(String[] args) {

TreeMap<Integer, String> treeMap = new TreeMap<>();

treeMap.put(3, "Three");

treeMap.put(1, "One");

treeMap.put(2, "Two");

// treeMap.put(null, "Null Key"); // Throws NullPointerException

System.out.println("TreeMap:");

for (Integer key : treeMap.keySet()) {

System.out.println("Key: " + key + ", Value: " + treeMap.get(key));

}

}

}

**Output** (Keys sorted):

yaml

Copy code

TreeMap:

Key: 1, Value: One

Key: 2, Value: Two

Key: 3, Value: Three

6,’what is hashmap vs list?

A,

| **Aspect** | **HashMap** | **ArrayList** |
| --- | --- | --- |
| **Purpose** | A HashMap is a key-value pair data structure. | An ArrayList is a resizable array for storing elements. |
| **Structure** | Uses a hash table internally. | Uses a dynamic array internally. |
| **Data Organization** | Stores data in key-value pairs (Map.Entry<K,V>). | Stores elements sequentially in an indexed manner. |
| **Access Time** | Fast lookup by key: O(1) (average case). | Fast random access by index: O(1). |
| **Order** | Does not guarantee the order of elements. | Maintains insertion order. |
| **Duplicates** | Keys must be unique; values can be duplicated. | Allows duplicate elements. |
| **Null Handling** | Allows one null key and multiple null values. | Allows multiple null elements. |
| **Use Case** | Used when you need key-value mapping and fast lookup. | Used when you need an ordered collection of elements. |
| **Implementation** | Implements the Map interface. | Implements the List interface. |

7,explain internal working of hashmap?

A,

i,what is hashcode?

A **hash code** is a numerical value that is generated by a hash function from an object, commonly used in data structures like hash tables to quickly locate data.

This numerical value is used in arraing index bucket in hash table.

Ex:

String str1 = "Hello"; String str2 = "World"; String str3 = "Hello";

System.out.println("Hash code for str1 (\"Hello\"): " + str1.hashCode());

ii , what is hashing ?

a,

**Hashing** is a process of converting data (like strings, numbers, or objects) into a fixed-size numerical value called a hash value or hash code, using a function known as a **hash function.**

**public class Main {**

**public static void main(String[] args) {**

**HashMap<Employee, String> employeeMap = new HashMap<>();**

**Employee emp1 = new Employee(1, "John");**

**Employee emp2 = new Employee(2, "Jane");**

**employeeMap.put(emp1, "HR");**

**employeeMap.put(emp2, "Finance");**

**System.out.println("John's department: " + employeeMap.get(emp1));**

**}**

**}**

**Iii ,what is hash table?**

**A ,**

**A hash table is a data structure that stores key-value pairs. It uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found efficiently.**

**8,how put method works in hashmap and equals method is used?**

**A,after providing key,value**

**It finds hash code using hash function ex:n%10**

**It will store data in that buket index.**

**If other key got the same index it will form linked list and continue adding.**

**But linked list o(n) traversal.**

**So if elements>8 it will form balanced tree.**

**In balanced tree less hashcode values are left remainin are right.**

**While travelling it checks equal function if exists it replaces value keeping key same.**

**If not exist attach at end.**

**9,when a linked list changed to tree in hashmap?**

**If threshold value>8**

**List changes to tree.**

**10,how delete function works?**

**If equal is true it deletes eventually.**

**11,how contains function works?**

**A,Use equal method and returns true.**

**12,what is collision in hashmap?**

**A,**

**A collision in a hash map occurs when two different keys generate the same hash value and are assigned to the same index in the hash table. This happens because the hash function, which maps keys to indices, may produce the same output for multiple inputs due to the finite size of the hash table.**

**Why Collisions Occur:**

1. **Finite Hash Table Size: A hash table has a limited number of slots, but the number of possible keys is often much larger.**
2. **Hash Function Limitations: A hash function maps keys to a fixed range of indices, so two different keys can map to the same index.**

**How chaining applies:**

**Chaining (Separate Chaining):**

* **Each index in the hash table points to a linked list (or another data structure) that holds all key-value pairs with the same hash value.**
* **If a collision occurs, the new key-value pair is added to the linked list at that index.**
* **Example: hash\_table[index] = [key1, key2, ...].**