COLLECTION

(1) EXPLAIN HEIRARCY OF COLLECTION?

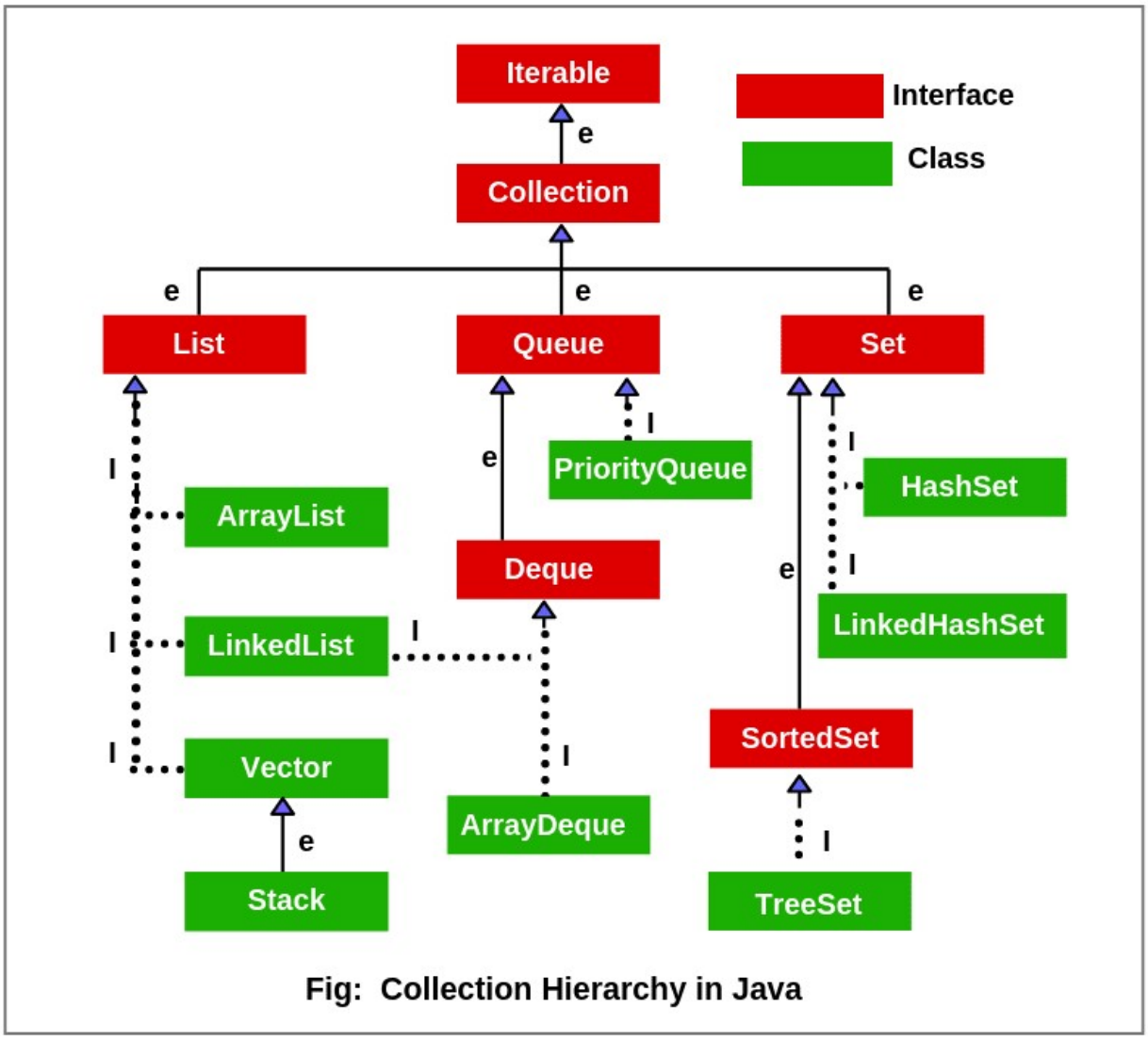


Fig: Collection Hierarchy in Java

(2) EXPLAIN HEIRARCY OF MAP ?

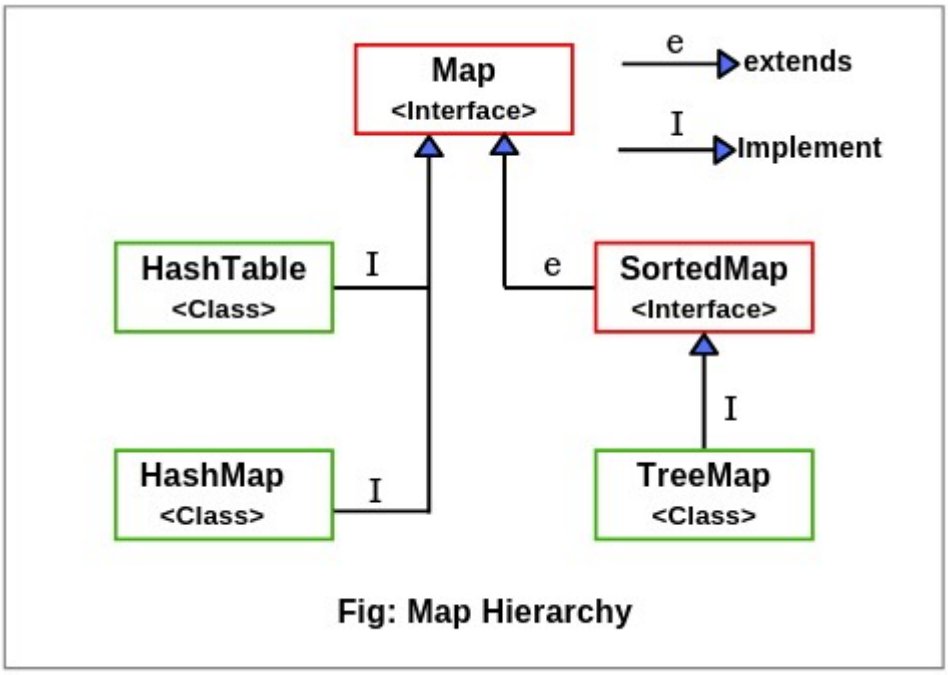


Fig: Map Hierarchy

(3) DIFFERENCE BETWEEN ARRAY AND COLLECTION?

Aspect	Array	Collection
Size	Fixed size, determined when the array is created.	Dynamic size, can grow or shrink at runtime.
Data Type	Can store either primitive types or objects.	Can only store objects (autoboxing supports primitive types).
Flexibility	Less flexible due to fixed size and lack of utility methods.	More flexible, provides utility methods for operations like sorting, searching, etc.
Access Methods	Access elements using indexes (`array[index]`).	Provides various methods like `add()`, `remove()`, `get()`, etc., depending on the type of collection.
Sorting & Searching	Requires manual implementation or `Arrays` utility class.	Many collections offer built-in support (e.g., `TreeSet`, `Collections.sort()`).

(4) DIFFERENCE BETWEEN COLLECTION, COLLECTIONS AND COLLECTION FRAMEWORK?

Aspect	Collection	Collections	Collection Framework
Definition	An interface in `java.util` that represents a group of objects.	A utility class in `java.util` that provides static methods for collection operations.	A unified architecture for storing, managing, and manipulating collections in Java.
Type	Interface	Class	Framework
Key Features	Provides abstract methods like `add()`, `remove()`, `size()`.	Offers utility methods for working with collections.	Combines all collection-related classes and interfaces to provide a cohesive structure.

Examples	Collection<String> coll = new ArrayList<>();	Collections.sort(list);	Includes `Collection`, `Collections`, `List`, `Set`, `Map`, `Queue`, etc.

(5) DIFFERENCE BETWEEN LIST AND SET?

Sno.	List	Set
1	List is index based data structure.	Set is hash code based data structure
2	List can store duplicate elements	Set doesn't allow to store duplicate
3	List can store N number of null values	Set can store only one null value
4	List follow insertion order	Set doesn't follow insertion order
5	We can iterate the list elements using either iterator or list-iterator	We can iterate the set elements using iterator
6	List implemented classes are: arraylist , linkedlist, vector & stack	Set implemented classes are : hash set, linked hashset , tree set.

(6) DIFFERENCE BETWEEN ITERATOR VS LIST-ITERATOR?

Aspect	Iterator	ListIterator
Applicable Collections	Works with all `Collection` types (e.g., `Set`, `List`, `Queue`).	Works only with `List` implementations (`ArrayList`, `LinkedList`).
Traversal Direction	Can traverse elements only in the **forward** direction.	Can traverse elements in **both forward and backward** directions.
Methods	- `hasNext()` - `next()` - `remove()`	- `hasNext()` - `next()` - `hasPrevious()` - `previous()` - `add()` - `set()` - `remove()`
Direction Methods	No method to traverse backward.	`hasPrevious()` and `previous()` methods allow backward traversal.
Declaration	Iterator<E> iterator = list.iterator();	ListIterator<E> listIterator = list.listIterator();

(7) EXPLAIN ENUMERATION CURSOR?

- Works only with legacy classes like: Vector, Stack, Hashtable (for keys or values).
- It is legacy cursor introduced in java 1.0 version
- **Read-Only**: Does not allow modification of elements during traversal.
- methods : hasMoreElements(), nextElement().
- To get enumeration cursor : Enumeration<String> enumeration = **fruits.elements();**

(8) DIFFERENCE BETWEEN ARRAYLIST AND LINKEDLIST WITH ADVANTAGE & DISADVANTAGE?

Aspect	ArrayList	LinkedList
Internal Data Structure	**Dynamic Array**: Elements are stored in a resizable array.	**Doubly Linked List**: Each element is a node with pointers to its previous and next nodes.
Access Speed	Fast random access (`O(1)` for `get(index)`), as it uses an index-based system.	Slower access (`O(n)` for `get(index)`), as traversal is required from the head or tail.
Insertion Speed	Slower for inserting elements at the middle or beginning, as shifting elements is required (`O(n)`).	Faster for insertions at the middle or beginning, as only pointers are updated (`O(1)` for such cases).
Deletion Speed	Slower for deletion from the middle or beginning due to element shifting (`O(n)`).	Faster for deletion from the middle or beginning, as pointers are updated (`O(1)` for such cases).
Memory Usage	Less memory used	More memory required as compare to array list.
Advantages	- Faster random access due to index-based storage. - Efficient for storing and retrieving large datasets. - Uses less memory compared to LinkedList.	- Faster insertions and deletions in the middle or beginning. - No resizing overhead as it dynamically adjusts to the size. - Suitable for real-time systems with frequent structural modifications.
Disadvantages	- Slower insertions and deletions in the middle or beginning. - Resizing overhead for large data. - Inefficient for frequent structural modifications.	- Slower random access due to traversal. - Higher memory usage due to pointers. - Iteration is slower compared to ArrayList.

(9) DIFFERENCE BETWEEN ARRAYLIST VS VECTOR ?

Aspect	ArrayList	Vector
Synchronization	Not synchronized	Synchronized
Performance	Faster due to the absence of synchronization	Slower due to synchronization
Growth Rate	Increases its size by **50%** when it runs out of capacity.	Doubles its size when it runs out of capacity.
Capacity ()	Not present	Present
Methods	----	addElement(), firstElement (), lastElement(), removeAllElement (), capacity ()

(10) EXPLAIN STACK ?

- A **stack** is a linear data structure that follows the **LIFO** (Last In, First Out) principle.
- As it extends `Vector`, it inherits methods like `size()`, `capacity()`.
- The **`Stack`** class internally uses a **dynamic array (Vector)** for storage.
- It is synchronized.
- Methods like :

Method	Description
push(E item)	Adds the specified item to the top of the stack.
pop()	Removes and returns the top item from the stack. Throws `EmptyStackException` if the stack is empty.
peek()	Returns the top item of the stack without removing it. Throws `EmptyStackException` if the stack is empty.
isEmpty()	Returns `true` if the stack is empty, otherwise `false`.
search(Object o)	Returns the 1-based position of the element in the stack. Returns `-1` if the element is not found.

(11) WHY ARE COMPARABLE AND COMPARATOR INTERFACES REQUIRED IN JAVA?

We know that we have Arrays.sort() or Collections.sort() for primitive data sorting however for custom sorting we need comparable or comparator because when we try to use Arrays.sort() or Collections.sort() then it's going to give exception followed by classCastException saying 'employee can't be cast to java.lang.comparable'. to avoid this error we must have to implement either comparable or comparator then use can use same method Arrays.sort() or Collections.sort()

Ex-

```
package org.example;

public class Employee {
    private int id;
    private String name;

    public Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "id=" + id +
                ", name='" + name + '\'' +
                '}';
    }
}
```

```
package org.example;

import java.util.Arrays;

public class Main {
    public static void main(String[] args) {

        // code for using Arrays.sort() with primitive types
//        int [] arr = {5,1,9,6};
//        Arrays.sort(arr);
//        System.out.println(Arrays.toString(arr));


        // same code for using Arrays.sort() with custom object
        Employee[] em = new Employee[4];
        em[0]= new Employee(5,"sam");
        em[1]= new Employee(1,"ali");
        em[2]= new Employee(9,"rahul");
        em[3]= new Employee(6,"meraz");


        Arrays.sort(em);
        System.out.println(Arrays.toString(em));
    }
}
```

(12) DIFFERENCE BETWEEN COMPARABLE VS COMPARATOR INTERFACE?

Feature	Comparable	Comparator
Purpose	Defines the **natural ordering** of objects.	Defines **custom ordering** of objects.
Package	java.lang (automatically imported)	java.util (requires import)
Implementation	Implemented in the same class as the objects being compared.	Implemented in a separate class or as an anonymous/lambda class.
Method to Override	compareTo(T o)	compare(T o1, T o2)
When to Use	When objects have a single, natural ordering.	When multiple or custom orderings are needed.
Invocation	`Collections.sort(list)` or `Arrays.sort(array)`	`Collections.sort(list, comparator)` or `Arrays.sort(array, comparator)`
Default in Java	Example: `String`, `Integer`, `Date` implement `Comparable`.	Requires explicit implementation for sorting logic.

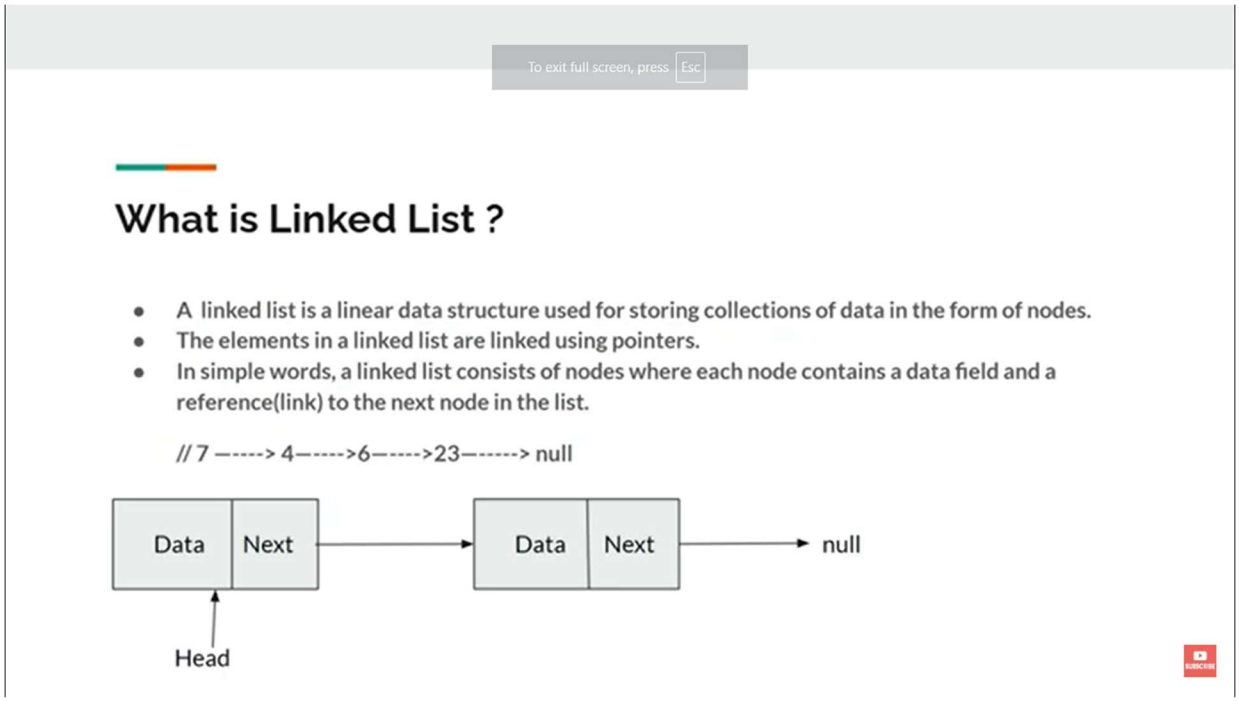
(13) WHAT ARE THE RETURN TYPE OF COMPARETO() AND COMPARE () METHOD?

> negative if 1st parameter is < 2nd parameter

> positive if 1st parameter is > 2nd parameter

> 0 if 1st parameter is = 2nd parameter.

(14) WHAT IS LINKED LIST & HOW MANY TYPES OF IT?



HOW MANY TYPES OF LINKEDLIST?

- SINGLY LINKED LIST
- DOUBLY LINKED LIST
- CIRCULAR LINKED LIST
- DOUBLY CIRCULAR LINKED LIST
- HEADER LINKED LIST

SINGLY LINKED LIST:-

- A singly linked list is a type of linked list that is unidirectional, that is, it can be traversed in only one direction from head to the last node (tail).
- Each element in a linked list is called a node, a single node contains data and a pointer to the next node which helps to maintain list structure.
- The first node is called head, it points to the first node of the list and help us access every other element in the list, the last node , also called tail points to null which help us to determine when

list ends.

(15) WHAT IS HASHING?

Hashing is a technique where we represent any entity into integer form which is done by using hash code method.

(16) WHAT IS COLLUSION?

When two values are being stored at same index number, then collision occur, to over come this problem, we store that value as linked list, mapped to same index number.

(17) EXPLAIN INTERNAL WORKING OF HASHMAP?

HASHMAP KEY CHARACTERISTICS:

- UnOrdered : doesn't maintain any order of it's elements.
- Allow null key & null values: can have one null key and multiple null values.
- Not synchronized : not thread safe , required external synchronization.
- performance : constant-time performance O(1) for basic operations like get and put.

INTERNAL WORKING OF HASHMAP:

there are 4 component of hashmap:

(a) Key

(b) Value

(c) Bucket

(d) Hash function

HOW DATA STORE IN HASHMAP ?

STEPS 1: Hashing the key ?

first, key is passed through a hash function to generate a unique hash code (integer) this hash code helps to determine where key value pair will be stored in the array (called "bucket").

STEPS 2 : calculating the index?

hash code is used to calculate an index in array using int index = hashCode % arraySize; the index decides which bucket will hold this key-value pair for Eg:- if array is 16 , the key's hash code will be divided by 16 and the remainder will be the index.

STEPS 3 : Storing in the bucket?

the key-value pair is stored in the bucket at the calculated index.each bucket can hold multiple key-value pairs. (this is called a collision handling mechanism).

HOW HASHMAP RETRIEVES DATA?

when we call get(key) , the hashmap follows these below steps:

(1) Hashing key: called hash function to calculate hash code.

(2) Finding the index: the hashcode is used to find the index of the bucket where the key-value pair is stored.

(3) searching in the bucket: Once the correct bucket is found,it checks for the key in the bucket ,if it finds the key its returns the associated value.

COLLISION : to handle linked list (threshold=8), when exceed threshold uses Balanced Binary Search Tree (Red Black tree).

HASHMAP RESIZING (REHASHING) ?

hash map has an internal array size, which by default is 16, when the number of elements (key-value) pairs grows and exceeds a certain load factor (0.75) , hash map automatically resizes the array to hold more data, this process called rehashing. the default size of array is 16 , so when more than 12 elements (16 * 0.75) are inserted , the hash map will resize during rehashing the array size will be double, and then all existing entries will be rehashed if no collusion time complexity O(1) otherwise log(n)

(18) DIFFERENCE BETWEEN HASHMAP AND CONCURRENT HASHMAP?

Aspect	HashMap	ConcurrentHashMap
Thread Safety	Not thread-safe	Thread-safe
Synchronization	Must be synchronized externally (e.g., using `Collections.synchronizedMap()` or manual synchronization).	Internally synchronized using a fine-grained locking mechanism (segment-based locking).
Null Keys and Values	Allows **one null key** and **multiple null values**.	Does **not allow null keys or null values**.
Iterator Behavior	The iterator is **fail-fast**: Concurrent modifications throw a `ConcurrentModificationException`.	The iterator is **fail-safe**: Works on a snapshot of the data and does not throw exceptions on concurrent modifications.
Use Case	Suitable for single-threaded applications or scenarios where manual synchronization is handled.	Suitable for multi-threaded applications requiring high-concurrency with thread-safe operations.
Introduced In	JDK 1.2	JDK 1.5

(19) EXPLAIN HASHTABLE ?

Hashtable is a legacy class now more modern alternatives like HashMap and ConcurrentHashMap have been introduced. which is the replacement of hash table.

Key Features of `Hashtable`

1. **Thread Safety**:
    - All methods in `Hashtable` are **synchronized**, making it thread-safe for concurrent access.
    - However, this synchronization can lead to performance overhead compared to non-synchronized classes like `HashMap`.
2. **No `null` Keys or Values**:
    - Unlike `HashMap`, `Hashtable` does **not allow `null` keys or `null` values**.
    - Attempting to insert a `null` key or value will throw a **`NullPointerException`**.
3. **Uses Hashing for Storage**:
    - Keys are hashed to determine their position in the internal hash table, ensuring fast lookups.
4. **Fail-Fast Iterators**:
    - The iterators of `Hashtable` are fail-fast. If the `Hashtable` is modified structurally during iteration (except through the iterator itself), a **`ConcurrentModificationException`** is thrown.
5. **Synchronized Alternative to `HashMap`**:
    - `Hashtable` can be used in multi-threaded environments where thread safety is required without additional synchronization.
6. all map interface methods can be used.

(20) WHAT WAS CHANGED IN JAVA 8, INTERMS OF INTERNAL WORKING OF HASHMAP?

Aspect	Before Java 8	In Java 8
Collision Handling	Used a **linked list** for storing entries in a bucket.	Uses a **linked list** for low-collision buckets, but switches to a **red-black tree** for high-collision buckets.
Performance on Collisions	Linear time complexity (**O(n)**) to traverse a bucket with collisions.	Logarithmic time complexity (**O(log n)**) for operations in tree-buckets.

(21) WHY SET TAKES ONLY UNIQUE ELEMENTS?

1. **Mathematical Definition**:
    - A **set** in mathematics is defined as a collection of distinct objects. Java's `Set` interface follows this principle and does not allow duplicate elements.
2. **Implementation**:
    - `Set` implementations like **`HashSet`**, **`LinkedHashSet`**, and **`TreeSet`** use mechanisms to check for duplicates:
        - **`HashSet`**: Uses a **hash table** to store elements. It relies on the `hashCode()` and `equals()` methods to determine whether two objects are equal.
        - **`LinkedHashSet`**: Same as `HashSet`, but maintains insertion order.
        - **`TreeSet`**: Uses a **self-balancing binary search tree** (e.g., a red-black tree) and compares elements based on their natural order or a provided comparator.

(22) WHY MAP IS NOT A PART OF COLLECTION?

Aspect	Collection	Map
Data Structure	Represents a group of individual elements.	Represents key-value pairs (entries).
Focus	Focuses on storing and processing elements.	Focuses on associating keys with values.
Interfaces	Extends `Iterable` and is part of the Java Collections Framework.	Does not extend `Collection` and is a separate hierarchy.
Usage	Used for single data elements like lists, sets, and queues.	Used for mapping keys to values, like dictionaries in other languages.

(23) DIFFERENCE BETWEEN HASHCODE AND EQUALS METHOD?

Aspect	hashCode()	equals()
Purpose	Returns an integer hash code to represent the object.	Compares two objects for equality.
Return Type	Returns an integer (`int`).	Returns a boolean (`true` or `false`).
Definition	Provides a hash-based representation of the	Defines logical equality

	object.	between two objects.
Usage	Used in hashing-based data structures like `HashMap` and `HashSet`.	Used to compare objects for equality in all contexts.
Relation	If two objects are considered equal using `equals()`, they must have the same `hashCode()`.	Two objects can have the same `hashCode()` but still not be equal.
Default Behavior	By default, derived from the object's memory address.	By default, uses reference equality (==).
Customization	Should be overridden for objects in hash-based collections.	Must be overridden for custom equality checks.
Contract	Must be consistent with `equals()`: if `equals()` returns `true`, `hashCode()` must return the same value.	Must define logical equality based on object properties.

(24) EXPLAIN DIFFERENCE BETWEEN FAIL SAFE AND FAIL FAST?

Aspect	Fail-Fast	Fail-Safe
Definition	Iterators that throw a **`ConcurrentModificationException`** if the collection is modified during iteration.	Iterators that **do not throw exceptions** if the collection is modified during iteration.
Behavior on Modification	Fails immediately when detecting structural modifications.	Works on a **clone** of the collection, so modifications don't affect the iteration.
Thread-Safety	Not thread-safe.	Thread-safe.
Underlying Collections	Applies to most collections in **`java.util`**, like `ArrayList`, `HashMap`, `HashSet`.	Applies to collections in **`java.util.concurrent`**, like `CopyOnWriteArrayList`, `ConcurrentHashMap`.
Exception Thrown	ConcurrentModificationException	No exception thrown.

(25) DIFFERENCE BETWEEN SYNCHRONIZED COLLECTION AND CONCURRENT COLLECTION?

:

- Both synchronized and concurrent collection classes provide thread safety
- The differences between them comes in performance , scalability and how they achieve thread safety
- Synchronized collection like hashmap are much slower than their concurrent parts eg –

concurrenthashmap . main reason for this slowness is locking.