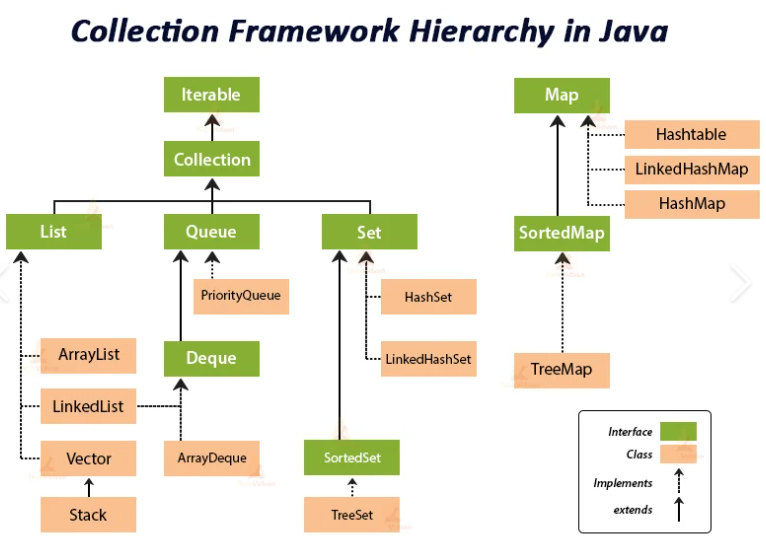
# Collection interview questions

## What is collection framework

Collections provide a way to store, manipulate, and manage data efficiently. In automation testing, collections are often used to handle and organize test data, represent page elements, manage test scenarios, and perform various data-related operations.

This framework includes interfaces (e.g., List, Set, Queue)



## What is difference in collection and collections

* Collection" refers to a high-level interface representing groups of objects in the Java Collections Framework. List, Set, Queue
* // Collection (interface) example

List<String> myList = new ArrayList<>()

* "Collections" refers to a utility class (**java.util.Collections**) providing static methods for operations on collections.
* // Collections (class) example - sorting the list

Collections.sort(myList);

## Why Maps are not part of collection?

| **Aspect** | **Collections** | **Maps** |
| --- | --- | --- |
| **Definition** | Grouping and manipulation of objects. Focus on storage and retrieval of individual elements. | Key-value pairs, associating keys with values. Represent relationships between entities. |
| **Framework** | Part of the Java Collections Framework (**java.util**). | Part of the Java Collections Framework (**java.util**). |
| **Interfaces** | Includes interfaces like List, Set, Queue, etc. | Primarily includes the **Map** interface. |
| **Implementations** | Examples include ArrayList, HashSet, LinkedList, etc. | Examples include HashMap, TreeMap, LinkedHashMap, etc. |
| **Key Characteristics** | Dealing with individual elements. | Associating keys with values, representing relationships. |
| **Common Operations** | Add, remove, iterate, retrieve elements by index, etc. | Put, get, remove, iterate over entries, retrieve values by key, etc. |
| **Example Usage** | List<String> names = new ArrayList<>(); | Map<String, Integer> ages = new HashMap<>(); |
| **Use Cases** | Storing and manipulating individual elements. | Representing relationships, managing key-value associations. |
| **Interface Hierarchy** | Collection (Root Interface) | Map (Separate Interface) |

## What are different types of collection

| **Characteristic** | **List** | **Set** | **Queue** |
| --- | --- | --- | --- |
| **Insertion Order** | Maintains order based on insertion sequence | Does not guarantee any specific order | Maintains order based on the first-in, first-out (FIFO) principle |
| **Duplicates** | Allows duplicate elements | Does not allow duplicate elements | May allow duplicates depending on the implementation |
| **Implementation** | Examples: ArrayList, LinkedList, Vector | Examples: HashSet, LinkedHashSet, TreeSet | Examples: LinkedList, PriorityQueue |
| **Accessing Elements** | Access by index using get(index), forEach loop | No direct methods for index-based access | Methods like poll(), offer(), peek() for element manipulation |
| **Use Cases** | Order of elements matters, duplicates allowed | Uniqueness of elements is essential | Order and the FIFO principle are important, often used in task scheduling |
| **Performance** | Faster random access, slower membership checks | Faster membership checks | Performance considerations may vary based on the implementation |
| **Basic Methods in Selenium** | **get(index)**, **size()**, **addAll()**, **remove(index)**, **clear()** | **add()**, **remove()**, **contains()**, **clear()** | **offer()**, **poll()**, **peek()**, **size()**, **clear()** |

## What is Difference in ArrayList and HashMap

| **Aspect** | **ArrayList** | **HashMap** |
| --- | --- | --- |
| **Purpose** | Ordered collection of elements | Key-value pair storage for efficient data retrieval |
| **Accessing Elements** | Accessed by index, e.g., **get(index)**, **set(index, value)** | Accessed by key, e.g., **get(key)**, **put(key, value)** |
| **Ordering** | Maintains insertion order | Does not guarantee specific order |
| **Type of Elements** | Stores a list of elements, any data type | Stores key-value pairs, any data type for keys and values |
| **Duplicate Elements** | Allows duplicates | Does not allow duplicate keys, values can be duplicated |
| **Iteration** | Iterated using for loops or enhanced for loops | Iterated using iterators, forEach, or entrySet |
| **Performance** | Fast random access, efficient for sequential access | Fast retrieval of values based on keys, efficient for lookup |
| **Java Classes** | **java.util.ArrayList** (implements List interface) | **java.util.HashMap** (implements Map interface) |

## What is difference in

| **Feature** | **HashMap** | **HashTable** | **HashSet** |
| --- | --- | --- | --- |
| **Synchronization** | Not synchronized | Synchronized (thread-safe) | Not synchronized (not thread-safe) |
| **Performance** | Generally has better performance compared to HashTable | Slower due to synchronization | Generally better in a single-threaded environment |
| **Null values** | Allows one null key and multiple null values | Does not allow null keys or values; attempting to insert null results in a **NullPointerException** | Does not allow null values |
| **Thread-safety** | Multiple threads can access and modify a HashMap simultaneously, which means it is not thread-safe. In a multi-threaded environment, concurrent modifications might lead to data corruption. | thread-safe | Not thread-safe |
| **Usage recommendation** | Preferred in single-threaded scenarios or when thread-safety is explicitly managed. Suitable for scenarios where performance is critical and explicit synchronization is not needed. | Suitable for multi-threaded scenarios requiring inherent thread-safety, though ConcurrentHashMap is recommended for improved performance. Use when explicit synchronization is required. | Preferred in single-threaded scenarios or when thread-safety is explicitly managed. Use when the order of elements doesn't matter, and uniqueness of elements is a requirement. |
| **Common Methods** | **put(key, value)**, **get(key)**, **remove(key)**, **containsKey(key)**, **containsValue(value)** | **put(key, value)**, **get(key)**, **remove(key)**, **containsKey(key)**, **containsValue(value)** | **add(element)**, **remove(element)**, **contains(element)**, **size()**, **clear()** |
| **Enumeration/Iterator** | No specific methods for enumeration, can use **keySet()**, **entrySet()**, **values()** | **elements()**, **keys()**, **values()**, **entrySet()** (Enumeration and Iterator are available) | **iterator()**, **forEach()**, **spliterator()** |
| **Iterating Entries** | Can use **entrySet()** or **forEach()** methods | Can use **entrySet()** or **forEach()** methods | Iterating through elements using **iterator()** or enhanced for loop |
| **Size** | **size()** method returns the number of key-value mappings | **size()** method returns the number of key-value mappings | **size()** method returns the number of elements |
| **Clearing** | **clear()** method removes all key-value mappings | **clear()** method removes all key-value mappings | **clear()** method removes all elements |
| **Clone** | **clone()** method creates a shallow copy of the HashMap | **clone()** method creates a shallow copy of the HashTable | **clone()** method creates a shallow copy of the HashSet |
| **Selenium Code Snippet** | ```java | ```java | ```java |
|  | // Example of using HashMap in Selenium | // Example of using HashTable in Selenium | // Example of using HashSet in Selenium |
|  | Map<String, String> userData = new HashMap<>(); | Hashtable<String, String> userData = new Hashtable<>(); | Set<String> elements = new HashSet<>(); |
|  | userData.put("username", "exampleUser"); | userData.put("username", "exampleUser"); | elements.add("button1"); |
|  | userData.put("password", "password123"); | userData.put("password", "password123"); | elements.add("inputField2"); |
|  |  |  | elements.add("checkbox3"); |
|  | // Accessing values | // Accessing values | // Performing actions with elements |
|  | String username = userData.get("username"); | String password = userData.get("password"); | for (String element : elements) { |
|  | System.out.println("Username: " + username); | System.out.println("Password: " + password); | driver.findElement(By.id(element)).click(); |
|  | ``` | ``` | } |

## What are common methods used in collection

You can mentioned Sort, min() max() reverse(), Search() in give Collection

import java.util.\*;

public class CollectionsExample {

public static void main(String[] args) {

// Example List

List<String> myList = new ArrayList<>();

myList.add("Orange");

myList.add("Apple");

myList.add("Banana");

// Sorting

System.out.println("Original List: " + myList);

Collections.sort(myList);

System.out.println("Sorted List: " + myList);

// Shuffling The shuffle() function randomizes the order of the elements in the array.

Collections.shuffle(myList);

System.out.println("Shuffled List: " + myList);

// Binary Search Binary search is a search algorithm that finds the position of a target value within a sorted array or list.

Collections.sort(myList); // Binary search requires a sorted list

int index = Collections.binarySearch(myList, "Apple");

System.out.println("Index of 'Apple': " + index);

// Min and Max

String minElement = Collections.min(myList);

String maxElement = Collections.max(myList);

System.out.println("Min Element: " + minElement);

System.out.println("Max Element: " + maxElement);

// Reversing

Collections.reverse(myList);

System.out.println("Reversed List: " + myList);

// Filling After the operation, all elements in the list will be replaced with the specified value

Collections.fill(myList, "Mango");

System.out.println("List after filling with 'Mango': " + myList);

// Copying

List<String> copyList = new ArrayList<>(myList.size());

Collections.copy(copyList, myList);

System.out.println("Copied List: " + copyList);

// Empty List and Singleton

List<String> emptyList = Collections.emptyList();

System.out.println("Empty List: " + emptyList);

Set<String> singletonSet = Collections.singleton("SingleValue");

System.out.println("Singleton Set: " + singletonSet);

}

}

* How to iterate List

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

public class ListExampleWithIterator {

public static void main(String[] args) {

// Creating a List

List<String> myList = new ArrayList<>();

// Adding elements

myList.add("Element 1");

myList.add("Element 2");

myList.add("Element 3");

// Accessing elements using Iterator

System.out.println("List Elements using Iterator:");

Iterator<String> iterator = myList.iterator();

while (iterator.hasNext()) {

String element = iterator.next();

System.out.println(element);

}

// removing an element

myList.remove("Element 2");

// displaying the list after removal

System.out.println("\nList after removing 'Element 2':");

for (String element : myList) {

System.out.println(element);

}

}

}

## How to iterate Set

import java.util.HashSet;

import java.util.Iterator;

import java.util.Set;

public class SetExampleWithIterator {

public static void main(String[] args) {

// Creating a Set

Set<String> mySet = new HashSet<>();

// Adding elements

mySet.add("Element 1");

mySet.add("Element 2");

mySet.add("Element 3");

// Accessing elements using Iterator

System.out.println("Set Elements using Iterator:");

Iterator<String> iterator = mySet.iterator();

while (iterator.hasNext()) {

String element = iterator.next();

System.out.println(element);

}

// Removing an element

mySet.remove("Element 2");

// Displaying the set after removal

System.out.println("\nSet after removing 'Element 2':");

for (String element : mySet) {

System.out.println(element);

}

}

}

## How to iterate hmaps

// Logic for HashMap Iteration

Map<String, String> myHashMap = new HashMap<>();

myHashMap.put("Key1", "Value1");

myHashMap.put("Key2", "Value2");

myHashMap.put("Key3", "Value3");

// Iterating through HashMap using Iterator

Iterator<Map.Entry<String, String>> iterator = myHashMap.entrySet().iterator();

while (iterator.hasNext()) {

Map.Entry<String, String> entry = iterator.next();

// Process each entry (entry.getKey() for key, entry.getValue() for value)

}

## What is difference in entry set and key set

import java.util.HashMap;

import java.util.Map;

import java.util.Set;

import java.util.Map.Entry;

public class EntrySetVsKeySetExample {

public static void main(String[] args) {

Map<String, Integer> myMap = new HashMap<>();

myMap.put("One", 1);

myMap.put("Two", 2);

myMap.put("Three", 3);

// Using entrySet()

Set<Entry<String, Integer>> entrySet = myMap.entrySet();

for (Entry<String, Integer> entry : entrySet) {

System.out.println("Key: " + entry.getKey() + ", Value: " + entry.getValue());

}

// Using keySet()

Set<String> keySet = myMap.keySet();

for (String key : keySet) {

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

}

}}

In the above example, **entrySet()** allows direct access to both keys and values, while **keySet()** provides access only to the keys, and **get(key)** is used to retrieve the corresponding values.

## Use of Arraylist in Selenium

// Logic for storing buttons in an ArrayList

List<WebElement> allButtons = driver.findElements(By.tagName("button"));

// Logic for storing visible links in an ArrayList

List<WebElement> visibleLinks = new ArrayList<>();

List<WebElement> allLinks = driver.findElements(By.tagName("a"));

for (WebElement link : allLinks) {

if (link.isDisplayed()) {

visibleLinks.add(link);

}

}

// Logic for storing dropdown options in an ArrayList

List<String> dropdownOptions = new ArrayList<>();

WebElement dropdown = driver.findElement(By.id("dropdown"));

Select select = new Select(dropdown);

List<WebElement> options = select.getOptions();

for (WebElement option : options) {

dropdownOptions.add(option.getText());

}

// Logic for storing window handles in an ArrayList

List<String> windowHandles = new ArrayList<>(driver.getWindowHandles());

// Logic for storing frame elements in an ArrayList

List<WebElement> allFrames = driver.findElements(By.tagName("iframe"));