**Collection Framework and Maps**

**Assignment**

1. What is the Collection framework in Java?

Ans: The Collections framework is a unified architecture provided by Java that allows users to manipulate and organize groups of objects. It provides interfaces and classes that help with the storage, retrieval, manipulation, and traversal of collections of objects. The framework consists of several key interfaces, such as List, Set, and Queue, as well as their corresponding implementation classes like ArrayList, HashSet, and LinkedList.

1. What is the difference between ArrayList and LinkedList?

Ans:

|  |  |  |
| --- | --- | --- |
|  | **ArrayList** | **LinkedList** |
| 1. | This class uses a dynamic array to store the elements in it. With the introduction of [generics](https://www.geeksforgeeks.org/generics-in-java/), this class supports the storage of all types of objects. | This class uses a [doubly linked list](https://www.geeksforgeeks.org/doubly-linked-list/) to store the elements in it. Similar to the ArrayList, this class also supports the storage of all types of objects. |
| 2. | Manipulating ArrayList takes more time due to the internal implementation. Whenever we remove an element, internally, the array is traversed and the memory bits are shifted. | Manipulating LinkedList takes less time compared to ArrayList because, in a doubly-linked list, there is no concept of shifting the memory bits. The list is traversed and the reference link is changed. |
| 3. | Inefficient memory utilization. | Good memory utilization. |
| 4. | It can be one, two or multi-dimensional. | It can either be single, double or circular LinkedList. |
| 5. | Insertion operation is slow. | Insertion operation is fast. |
| 6. | This class implements a [List interface](https://www.geeksforgeeks.org/list-interface-java-examples/). Therefore, this acts as a list. | This class implements both the List interface and the [Deque interface](https://www.geeksforgeeks.org/deque-interface-java-example/). Therefore, it can act as a list and a deque. |
| 7. | This class works better when the application demands storing the data and accessing it. | This class works better when the application demands manipulation of the stored data. |
| 8. | Data access and storage is very efficient as it stores the elements according to the indexes. | Data access and storage is slow in LinkedList. |
| 9. | Deletion operation is not very efficient. | Deletion operation is very efficient. |
| 10. | It is used to store only similar types of data. | It is used to store any types of data. |
| 11. | Less memory is used. | More memory is used. |
| 12. | The memory is allocated at compile-time only. | The memory is allocated at run-time. |
| 13. | This is known as static memory allocation. | This is known as dynamic memory allocation. |

1. What is the difference between Iterator and ListIterator?

Ans:

|  |  |
| --- | --- |
| **Iterator** | **ListIterator** |
| 1. Can traverse elements present in Collection only in the forward direction. | 1. Can traverse elements present in Collection both in forward and backward directions. |
| 1. Helps to traverse Map, List and Set. | 1. Can only traverse List and not the other two. |
| 1. Indexes cannot be obtained by using Iterator. | 1. It has methods like nextIndex() and previousIndex() to obtain indexes of elements at any time while traversing List. |
| 1. Cannot modify or replace elements present in Collection | 1. We can modify or replace elements with the help of set(E e) |
| 1. Cannot add elements and it throws ConcurrentModificationException. | 1. Can easily add elements to a collection at any time. |
| 1. Certain methods of Iterator are next(), remove() and hasNext(). | 1. Certain methods of ListIterator are next(), previous(), hasNext(), hasPrevious(), add(E e). |

1. What is the difference between Iterator and Enumeration?

Ans:

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Key** | **Iterator** | **Enumeration** |
| 1 | Basic | In Iterator,  we can read and remove element while traversing element in the collections. | Using Enumeration, we can only read element during traversing element in the collections. |
| 2. | Access | It can be used with any class of the collection framework. | It can be used only with legacy class of the collection framework such as a Vector and HashTable. |
| 3. | Fail-Fast and Fail -Safe | Any changes in the collection, such as removing element from the collection during a thread is iterating collection then it throw concurrent modification exception. | Enumeration  is Fail safe in nature. It doesn’t throw concurrent modification exception |
| 4. | Limitation | Only forward direction iterating is possible | Remove operations can not be performed using Enumeration. |
| 5. | Methods | It has following methods −: \*hasNext() \*next() \*remove() | It has following methods −: \*hasMoreElements() \*nextElement() |

1. What is the difference between List and Set?

Ans:

|  |  |  |
| --- | --- | --- |
| **S.No** | **List** | **Set** |
| 1. | The list implementation allows us to add the same or duplicate elements. | The set implementation doesn't allow us to add the same or duplicate elements. |
| 2. | The insertion order is maintained by the List. | It doesn't maintain the insertion order of elements. |
| 3. | List allows us to add any number of null values. | Set allows us to add at least one null value in it. |
| 4. | The List implementation classes are LinkedList and ArrayList. | The Set implementation classes are TreeSet, HashSet and LinkedHashSet. |
| 5. | We can get the element of a specified index from the list using the get() method. | We cannot find the element from the Set based on the index because it doesn't provide any get method(). |
| 6. | It is used when we want to frequently access the elements by using the index. | It is used when we want to design a collection of distinct elements. |
| 7. | The method of List interface listiterator() is used to iterate the List elements. | The iterator is used when we need to iterate the Set elements. |

1. What is the difference between HashSet and TreeSet?

Ans:

|  |  |  |
| --- | --- | --- |
| **Parameters** | **HashSet** | **TreeSet** |
| Ordering or Sorting | It does not provide a guarantee to sort the data. | It provides a guarantee to sort the data. The sorting depends on the supplied Comparator. |
| Null Objects | In HashSet, only an element can be null. | It does not allow null elements. |
| Comparison | It uses hashCode() or equals() method for comparison. | It uses compare() or compareTo() method for comparison. |
| Performance | It is faster than TreeSet. | It is slower in comparison to HashSet. |
| Implementation | Internally it uses HashMap to store its elements. | Internally it uses TreeMap to store its elements. |
| Data Structure | HashSet is backed up by a hash table. | TreeSet is backed up by a Red-black Tree. |
| Values Stored | It allows only heterogeneous value. | It allows only homogeneous value. |

1. What is the difference between Array and ArrayList?

Ans:

|  |  |  |
| --- | --- | --- |
| **Basis** | **Array** | **ArrayList** |
| Definition | An array is a dynamically-created object. It serves as a container that holds the constant number of values of the same type. It has a contiguous memory location. | The ArrayList is a class of Java Collections framework. It contains popular classes like Vector, HashTable, and HashMap. |
| Static/ Dynamic | Array is static in size. | ArrayList is dynamic in size. |
| Resizable | An array is a fixed-length data structure. | ArrayList is a variable-length data structure. It can be resized itself when needed. |
| Initialization | It is mandatory to provide the size of an array while initializing it directly or indirectly. | We can create an instance of ArrayList without specifying its size. Java creates ArrayList of default size. |
| Performance | It performs fast in comparison to ArrayList because of fixed size. | ArrayList is internally backed by the array in Java. The resize operation in ArrayList slows down the performance. |
| Primitive/ Generic type | An array can store both objects and primitives type. | We cannot store primitive type in ArrayList. It automatically converts primitive type to object. |
| Iterating Values | We use for loop or for each loop to iterate over an array. | We use an iterator to iterate over ArrayList. |
| Type-Safety | We cannot use generics along with array because it is not a convertible type of array. | ArrayList allows us to store only generic/ type, that's why it is type-safe. |
| Length | Array provides a length variable which denotes the length of an array. | ArrayList provides the size() method to determine the size of ArrayList. |
| Adding Elements | We can add elements in an array by using the assignment operator. | Java provides the add() method to add elements in the ArrayList. |
| Single/ Multi-Dimensional | Array can be multi-dimensional. | ArrayList is always single-dimensional. |

1. What is a Map in Java?

Ans: In Java, a Map is an interface in the java.util package that represents a collection of key-value pairs. It is also known as an associative array or a dictionary.

A Map does not allow duplicate keys, but it allows multiple values associated with a single key. Each key-value pair is treated as a separate entry in the map. Keys are used to retrieve the associated values.

The main advantage of using a Map is efficient and fast lookup of values based on their keys. It provides various methods to manipulate and access the key-value pairs, such as inserting a key-value pair, removing a key-value pair, retrieving a value by key, checking if a key exists, and iterating over the keys or values.

The Map interface is implemented by several classes in Java, such as HashMap, TreeMap, LinkedHashMap, and ConcurrentHashMap, each with its own characteristics and performance trade-offs. You can choose the appropriate implementation based on your requirements, such as ordering, concurrency, or memory usage.

1. What are the commonly used implementations of Map in Java?

Ans: Some commonly used implementations of the Map interface in Java are:

1. HashMap: It is the most commonly used implementation of a Map in Java. It stores key-value pairs in an unordered manner. HashMap provides constant-time performance for basic operations like get() and put(). However, it does not guarantee the order of the elements.
2. TreeMap: It is an implementation of the SortedMap interface and stores key-value pairs in a sorted order based on the natural order of the keys or a custom comparator. TreeMap is slower than HashMap for basic operations but offers additional functionality like range-based queries and key-based operations.
3. LinkedHashMap: It maintains the insertion order of the keys, meaning the elements are stored in the order they were added to the map. LinkedHashMap offers the same performance as HashMap for basic operations but has slightly higher memory overhead due to maintaining the order.
4. ConcurrentHashMap: It is a thread-safe implementation of a Map that provides high concurrency for read and write operations. ConcurrentHashMap achieves this by internally partitioning the map into segments, allowing multiple threads to operate on different segments simultaneously.
5. EnumMap: It is an implementation of a Map where the keys are enum constants. EnumMap is highly efficient and provides constant-time performance for most operations.
6. WeakHashMap: It is an implementation of a Map where the keys are weak references. WeakHashMap is useful in situations where the keys are dependent on external factors and should be automatically removed when they are no longer referenced.

These are just a few commonly used implementations of the Map interface in Java. Depending on your specific requirements, you may choose the appropriate implementation.

1. What is the difference between HashMap and TreeMap?

Ans:

|  |  |  |
| --- | --- | --- |
| **Basis** | **HashMap** | **TreeMap** |
| Definition | 1. Java HashMap is a hashtable based implementation of Map interface. | 1. Java TreeMap is a Tree structure-based implementation of Map interface. |
| Interface Implements | 1. HashMap implements Map, Cloneable, and Serializable interface. | 1. TreeMap implements NavigableMap, Cloneable, and Serializable interface. |
| Null Keys/ Values | 1. HashMap allows a single null key and multiple null values. | 1. TreeMap does not allow null keys but can have multiple null values. |
| Homogeneous/ Heterogeneous | 1. HashMap allows heterogeneous elements because it does not perform sorting on keys. | 1. TreeMap allows homogeneous values as a key because of sorting. |
| Performance | 1. HashMap is faster than TreeMap because it provides constant-time performance that is O(1) for the basic operations like get() and put(). | 1. TreeMap is slow in comparison to HashMap because it provides the performance of O(log(n)) for most operations like add(), remove() and contains(). |
| Data Structure | 1. The HashMap class uses the hash table. | 1. TreeMap internally uses a Red-Black tree, which is a self-balancing Binary Search Tree. |
| Comparison Method | 1. It uses equals() method of the Object class to compare keys. The equals() method of Map class overrides it. | 1. It uses the compareTo() method to compare keys. |
| Functionality | 1. HashMap class contains only basic functions like get(), put(), KeySet(), etc. . | 1. TreeMap class is rich in functionality, because it contains functions like: tailMap(), firstKey(), lastKey(), pollFirstEntry(), pollLastEntry(). |
| Order of elements | 1. HashMap does not maintain any order. | 1. The elements are sorted in natural order (ascending). |
| Uses | 1. The HashMap should be used when we do not require key-value pair in sorted order. | 1. The TreeMap should be used when we require key-value pair in sorted (ascending) order. |

1. How do you check if a key exists in a Map in Java?

Ans: To check if a key exists in a Map in Java, you can use the following methods:

1. `containsKey(Object key)`: This method is available in the Map interface and returns true if the map contains the specified key, otherwise returns false.

Example:

java

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

map.put("A", 1);

map.put("B", 2);

boolean keyExists = map.containsKey("A"); // true

boolean keyDoesNotExist = map.containsKey("C"); // false

1. `get(Object key)`: This method is also available in the Map interface and returns the value to which the specified key is mapped, or null if the key is not present in the map.

Example:

java

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

map.put("A", 1);

map.put("B", 2);

Integer value = map.get("A"); // 1

Integer nonExistentValue = map.get("C"); // null

Note: It is important to note that if the value associated with the key can be null, the `get()` method alone cannot be used to determine if the key exists or not, as it will return null for both existing and non-existing keys with null values. In such cases, `containsKey()` should be used instead.