Assignment No 6

Q1.Answer

Collections in java is a framework that provides an architecture to store and manipulate the group of objects. All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

WHAT IS COLLECTION IN JAVA Collection represents a single unit of objects i.e. a group.

Q2.Answer:

1. ***Collection :*** *"Collection" is an interface defined in the Java Collections Framework, which is part of the Java Standard Library. It serves as the root interface in the collection hierarchy and represents a group of objects known as elements. The Collection interface provides common methods for working with collections, such as adding, removing, and querying elements. It is a general-purpose interface and has various sub interfaces and implementations, such as List, Set, and Queue.*

Example:

```
Collection<String> myCollection = new ArrayList<>();
myCollection.add("element1"); myCollection.add("element2");
System.out.println(myCollection); // Output: [element1, element2]
```

2. ***collections*** *: "collections" usually refers to the plural form of "collection" and is commonly used to describe multiple instances of collections or various collection-related classes, methods, or utilities. It is a more general term and can encompass different types of collections, including instances of classes that implement the Collection interface, such as List, Set, Queue, etc.*

Example:

List<String> stringList = new ArrayList<>(); Set<Integer> integerSet = new HashSet<>(); Map<String, Integer> stringIntegerMap = new HashMap<>();

Q3.Answer:

1. *Reusability: The Collection framework provides a set of standard interfaces, implementations, and algorithms that can be reused across different applications. This promotes code reusability, as developers can use these pre-defined components instead of reinventing the wheel.*
2. *Abstraction and Polymorphism: The framework is designed around abstract interfaces, such as Collection, List, Set, Queue, etc. This allows developers to write code that is agnostic to specific implementations, promoting abstraction and polymorphism. It becomes easier to switch or substitute different collection implementations based on specific needs.*
3. *Standardized APIs: The Collection framework offers a comprehensive set of methods and algorithms for working with collections. These standardized APIs provide common operations for adding, removing, querying, sorting, iterating, and manipulating elements in collections. Developers can leverage these APIs, saving time and effort in implementing common collection-related functionalities.*
4. *Performance Efficiency: The framework includes various collection implementations optimized for different use cases. For example,* ***ArrayList*** *provides fast random access, while* ***LinkedList*** *offers efficient insertion and removal. By choosing the appropriate collection implementation, developers can optimize their code for specific performance requirements.*

Q4.Answer:

Collection:

- *It is the root interface in the Collection hierarchy.*
- *Represents a group of objects known as elements.*
- *Defines basic operations such as adding, removing, querying, and iterating over elements.*
- *Common methods include* ***add****,* ***remove****,* ***contains****,* ***size****,* ***is Empty****,* ***iterator****, etc.*
- *Sub interfaces include List, Set, and Queue.*

2. *List:*

- *Extends the Collection interface.*
- *Represents an ordered collection of elements.*
- *Allows duplicate elements and maintains insertion order.*
- *Common methods include* ***add****,* ***remove****,* ***get****,* ***set****,* ***index Of****,* ***sub List****, etc.*
- *Notable implementations include ArrayList, LinkedList, and Vector.*

3. *Set:*

- *Extends the Collection interface.*
- *Represents a collection of unique elements, with no duplicates allowed.*
- *Does not maintain a specific order of elements.*
- *Common methods include* ***add****,* ***remove****,* ***contains****,* ***size****,* ***is Empty****, etc.*
- *Notable implementations include HashSet, TreeSet, and LinkedHashSet.*

*4.**Queue**:*

- *Extends the Collection interface.*
- *Represents a collection designed for holding elements prior to processing.*
- *Follows the First-In-First-Out (FIFO) order.*
- *Common methods include **add**, **remove**, **peek**, **element**, **offer**, etc.*
- *Notable implementations include LinkedList, PriorityQueue, and ArrayDeque.*

4. Map:

- *Not a subinterface of Collection.*
- *Represents a collection of key-value pairs.*
- *Each key in a Map must be unique, and each key is associated with a value.*
- *Common methods include **put**, **get**, **remove**, **contains Key**, **keySet**, **entry Set**, etc.*

Q5. Answer:

List:

- *Allows duplicate elements: You can have multiple occurrences of the same element in a List.*
- *Maintains insertion order: The order of elements in a List is determined by the order in which they are added.*
- *Provides positional access: You can access elements in a List using their index.*
- *Examples: ArrayList, LinkedList, Vector.*

Set:

- *Does not allow duplicate elements: Each element in a Set must be unique.*
- *Does not maintain a specific order: The elements in a Set can be in any arbitrary order.*
- *Does not provide positional access: You cannot directly access elements in a Set using an index.*
- *Examples: HashSet, TreeSet, LinkedHashSet.*

Q6.Answer:

Iterator:

Iterator is used for traversing List and Set.

We can traverse in only forward direction using Iterator.

We cannot obtain indexes while using Iterator

We can remove element from the collection but we cannot add element to collection while traversing it using Iterator, it throws Concurrent Modification Exception when you try to do it.

We cannot replace the existing element value when using Iterator.

ListIterator:

We can use ListIterator to traverse List only, we cannot traverse Set using ListIterator.

Using ListIterator, we can traverse a List in both the directions (forward and Backward).

We can obtain indexes at any point of time while traversing a list using ListIterator. The methods nextIndex() and previousIndex() are used for this purpose.

We can add/Remove element at any point of time while traversing a list using ListIterator.

We can add a new element to any position in the collection by using add(E e) method

By using set(E e) method of ListIterator we can replace the last element returned by next() or previous() methods.

Q7Answer:

- ***Comparable***

1) Comparable provides single sorting sequence. In other words, we can sort the collection on the basis of single element such as id or name or price etc.

2) Comparable affects the original class i.e. actual class is modified..

3) Comparable provides compareTo () method to sort elements.

4) Comparable is found in java.lang package.

5) We can sort the list elements of Comparable type by Collections.sort(List) method.

- ***Comparator***

1. *Comparator provides multiple sorting sequence. In other words, we can sort the collection on the basis of multiple elements such as id, name and price etc.*
2. *Comparator doesn't affect the original class i.e. actual class is not modified.*
3. *Comparator provides compare() method to sort elements.*
4. *Comparator is found in java.util package.*
5. *We can sort the list elements of Comparator type by Collections. sort(List, Comparator) method.*

Q8.Answer:

A collision, or more specifically, a hash code collision in a HashMap, is a situation where two or more key objects produce the same final hash value and hence point to the same bucket location or array index.

Q9.Answer:

HashMap

1. *HashMap doesn't maintain any sorting order*
2. *Internally it uses hash table*
3. *Contains one null key and many null values*
4. *HashMap implements Map interface*
5. *Time Complexity - O(1)*

TreeMap

1. *TreeMap elements are sorted according to natural sorting order or Comparator*
2. *Internally is uses Red Black Tree*
3. *Cannot contains null keys but may contains many null values*
4. *It implements NavigableMap interface*
5. *Time Complexity - O(log n).*

Q10.Answer:

LinkedHashMap is a class in Java that extends the HashMap class and provides an ordered and predictable iteration order. It combines the functionality of a hash table and a doubly-linked list to maintain the order of elements.

Key Points:

1. *Ordered Iteration: Unlike HashMap, LinkedHashMap preserves the insertion order or access order of elements during iteration.*
2. *Hash Table and Linked List: Internally, it uses a hash table for fast access and lookup by keys, while a linked list maintains the order of elements.*
3. *Performance: The time complexity of basic operations is O(1) on average, but slightly slower than HashMap due to the linked list maintenance.*
4. *Iteration Modes: It supports two iteration modes - insertion order (default) and access order. Access order allows iterating based on the most recently accessed elements.*
5. *Null Keys and Values: LinkedHashMap allows null values and a single null key, handled similarly to HashMap.*