BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 4

DCA2202

JAVA PROGRAMMING

Unit 8
Collections

Table of Contents

1. INTRODUCTION

Millions of users prefer the Java programming language as a coding language. Java is a fundamental object-oriented, dependable, efficient, and secure programming language.

Collections are used in every programming language. A collection's structure is included in the Java platform. A collection is a representation objects that group multiple elements into a single unit. Before the Collections Framework, it was hard for programmers to write algorithms that worked for different kinds of collections. Java came with some Collection classes, like Vector, Stack, Hashtable, and Array, but they had their disadvantages.

In JDK 1.2, Java developers introduced the Collections Framework, which is an important framework to help you achieve all of your data operations. A collections framework is a unifying architecture for describing and processing collections that allows them to be modified regardless of implementations.

STUDY NOTE

Java is a broad, object-oriented programming language based on classes that are supposed to have fewer implementation requirements.

Java Collection Framework enables the user to perform various data manipulation operations like storing data, searching, sorting, insertion, deletion, and updating of data on the group of elements.

Java Collections are the one-stop solutions for all the data manipulation jobs. Java collection responds as a single object, and a Java Collection Framework provides various Interfaces and Classes.

Here, you must learn and understand the Hierarchy of Java collections and various descendants, or classes and interfaces involved in the Java Collections. After the Hierarchy of Java collections, you should also get to know the various methods applied to the Collections in Java to perform the data manipulation operations.

1.1 Learning Objectives

After studying this chapter, you will be able to:

- *Understand the need of collections and its hierarchy*
- *Get to know about different collection interfaces*
- *Explain list interface, set interface, and map interface*
- *Discuss about miscellaneous concepts in java collections*

2. COLLECTIONS OVERVIEW

Collections are a specified set of classes or data structures for storing groups or collections of items. These data models, unlike arrays, are robust and can quickly grow or decrease without needing to be redefined. They are simple to use due to capacity planning and established algorithms for sorting, searching, and manipulating data.

A framework is a collection of classes and objects that come with a pre-built structure. There is no need to create a system to add new functionality or class. On the other hand, an ideal object-oriented design always comprises a structure with a variety of subclasses. All of these do the same activity.

Collections are divided into three categories. There are three of them: List, Set, and Map.

1. **A list** is a type of data structure that is used to store items in a specific order. The elements are saved and returned in the same sequence in which they were added. Duplicate elements are allowed in lists. Java ArrayList and Java LinkedList are two well-known List implementations.
2. **Set** saves just the unique elements. Because it is unpredictable and distinct, it will only retain one component if we add any duplicates. HashSet Class in Java and Java Set are two well-known classes that implement Set. TreeSet - Java TreeSet Examples
3. **A map** is a type of data structure that stores key-value pairs. The keys of any map are distinctive, although the values are not. Java HashMap Implementation and Java TreeMap are two classes that implement Map.

STUDY NOTE

A collection is a set of classes or data structures that are used to store groups or collections of stuff.

2.1 Need For Collection Classes In Java

A class is a blueprint or prototype that is defined by the user and used to build things. It denotes the collection of properties or methods that all objects of the same type share.

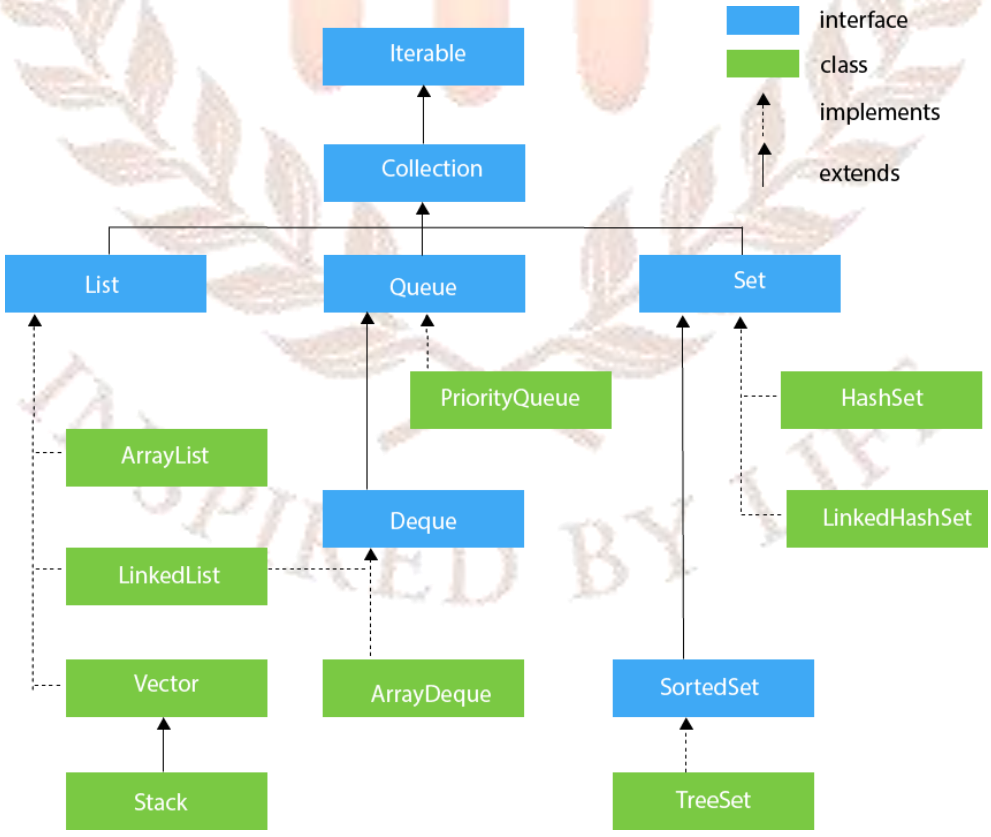
In Java, the collections class is a helpful utility class for working with collections. The collections class contains polymorphism collection techniques and "wrappers," which

produce a new collection supported by a systematic and organized. It belongs to the Java Collections Framework.

Why do you need collection classes? There are several reasons:

- Collection classes in Java provide a more high-level interface than arrays.
- The size of an array is fixed. The size of collections (see ArrayList) is variable.
- Constructing complex data structures (e.g., hash tables) properly on top of raw arrays is difficult. It is provided for free by the basic HashMap.
- You can pick between alternative implementations for the same Set of services, such as ArrayList vs. LinkedList, HashMap vs. TreeMap, synchronized, and so on.
- Furthermore, arrays allow for covariance: changing an array element is not assured to succeed due to typing issues that can only be detected at runtime. In arrays, generics solve this difficulty.

2.2 The Collection Hierarchy



Source: *sciencetecheasy.com*

Fig 1: The Collection Hierarchy

The comprehensive collection framework hierarchy comprises four fundamental interfaces: Collection, List, Set, Map, and two specific interfaces for sorting called SortedSet and SortedMap. The Java.util package contains all of the collection framework's interfaces and classes.

- **Extends**: The keyword extends used to create inheritance between two classes and two interfaces.
- **Implements:** The term implements are used to create inheritance across classes and interfaces.

The Collection interface, which is the root interface of all archives in the API, is the most fundamental interface of the collections system (Application programming interface). In Java, this is at the apex of the collection hierarchy. It allows you to add and remove elements from the Collection using basic methods.

The Iterable interface is augmented by the Collection interface. There is only one procedure in the iterable interface called Iterator (). The iterator method's purpose is to retrieve the iterator object. We may loop over the components of the array that use this iterator object.

1. **List Interface:**
   - This interface represents a collection of elements that are grouped chronologically.
   - Lists preserve element order, which means that the order in which we add elements is preserved, and the same sequence gets preserved while retrieving elements.
   - At any point in the List, we can insert elements. In Java, the List enables the storage of duplicate elements.
   - The list interface is implemented by three concrete subclasses: ArrayList, Vector, and LinkedList.
2. **Set Interface:**
   - This interface is made up of a variety of parts, each of which is distinct. It's used to keep a collection of one-of-a-kind pieces.

STUDY NOTE

There are four basic interfaces: Collection, List, Set, and Map, as well as two specific sorting interfaces called SortedSet and SortedMap.

- While storing objects, the Set interface does not keep track of their order, and we may not get the same order when retrieving them. A set's elements can be arranged in any sequence.
- Duplicate items are not permitted in Set.

3. **SortedSets Interface**:
   - This interface extends a set whose Iterator traverses its elements in the order in which they appear in the Set.
   - TreeSet implements the sorted interface.
4. **Queue Interface:**
   - A queue is an ordered collection of homogeneous elements, with the addition of new elements at one end (back) and removal of elements at the other (front), as a line at a supermarket or any store.
   - This interface defines a specific type of List in which only the head entries are eliminated.
5. **Deque Interface**: A Deque (also known as a double-ended queue) is a type of queue interface.
   - In Java SE 6, this interface was introduced to the collection framework.
   - The Deque interface enhances the queue interface and implements deque using its method.
6. **Map Interface:** Map interface is a part of Java Collection Framework, but it doesn't inherit Collection Interface. It describes an object that stores and retrieves elements in Key/Value pairs, with a Key determining their placement within the Map.
   - For storing key-value pairs, Map employs a hashing algorithm.
   - It does not allow multiple keys to be stored, but duplicate values are permitted.
7. **Sorted Map Interface:**
   - The elements of this Map are maintained in their order of things in this interface. TreeMap classes follow the Map interface, which is extended by it.

Self-Assessment Questions - 1

1. The collection _____.
   a) Extends collections class
   b) Extends Iterable class
   c) Implements serializable interface
   d) Implements traversable interface
2. List, Set and Queue are extends collection. [True/ False]
3. _____ is the root and fundamental interface of collections in java.
4. Interface that is part of collection framework, but doesn't inherit the collection interface:
   a) set
   b) list
   c) map
   d) none

3. LIST: THE INTERFACE

An ordered collection is represented via a list interface (sometimes called a sequence). Identical elements can appear in lists. In most cases, the user of a List interface has complete control over, in which each list element is entered and can retrieve elements by corresponding integer index (position).

The List interface is used to keep track of the Collection's order. It is a Collection's child interface and a sorted collection of objects that can store duplicate values. The List supports positional retrieval and addition of members because the insertion order is preserved.

The following are some of the additional operations enabled by the List interface over the Collection interface:

- Positional access tries to control elements in a list according to their numerical position.
- Search looks in the List for a specific object and returns its numerical position.
- Iteration is a semantic extension of Iterator that takes advantage of the List's sequential nature.
- Range-view allows you to execute arbitrary range operations on your data.

General-Purpose List Implementations

General-purpose and special-purpose implementations are the two types of list implementations.

ArrayList and LinkedList are two general-purpose List implementations. You'll presumably use ArrayList in almost all cases because it provides constant-time positional reference and is just straight quick.

It doesn't have to allocate a node object for each List element, and it may use System.arraycopy to move numerous elements at once. Consider ArrayList to be a Vector without the synchronization.

Contemplate employing LinkedList if you often add members to the beginning of the List or iterate through the List to delete elements from its inside. In LinkedList, these actions take constant effort, but in an ArrayList, they take linear time. However, you will pay a high price

in terms of performance. In LinkedList, positional access takes linear time, while in an ArrayList, it takes constant time.

Moreover, the LinkedList constant factor is substantially worse. If you think you'll use a LinkedList, test your application's performance with both LinkedList and ArrayList before deciding; ArrayList is probably quicker.

The main constituent is the only tuning option available for ArrayList. It represents the number of elements the ArrayList can store when it has to expand, there are no tuning options for LinkedList, and it has seven optional actions, one of which is a clone. AddFirst, getFirst, removeFirst, addLast, getLast, and removeLast are the other six. The Queue interface is also implemented by LinkedList.

Special-Purpose List Implementations:

CopyOnWriteArrayList is a copy-on-write array-backed List implementation. This implementation is comparable to CopyOnWriteArraySet in essence. Including during iteration, no synchronization is required, and iterators are assured never to raise ConcurrentModificationException. This method is ideal for maintaining event-handler lists where changes are few, but traversal is regular and possibly time-consuming.

A Vector would probably be quicker than an ArrayList synchronized with Collections.synchronizedList if you ever need synchronization. However, because Vector has many legacy functions, you should always traverse it using the List interface. Otherwise, you won't be able to replace the implementation afterward.

STUDY NOTE

The List interface is used to maintain track of the order of the collection. It is the child interface of a Collection.

If the length of your List is constant — that is, you'll don't use delete, add, or any other batch action except contains all — you have a third alternative to consider. For more details, see the Arrays.asList part in the Convenience Implementations part.

3.1 Implementations Of List Interface

ArrayList

The ArrayList implementation of the List interface is configurable. It supports all optional list operations and allows all elements, even NULL. It is nearly identical to Vector, with the

exception that it is not synchronized. To avoid unintended unsynchronized access to the ArrayList, use the Collections.synchronizedList method.

Here are some key features of ArrayList:

- Provides positional access in real-time.
- Allows for quick iteration and random access.
- It's an organized (by index) but not categorized collection.
- The most typical method of implementation.

LinkedList

A linked list is a collection of data pieces that are organized linearly. It's a data structure made up of a series of nodes presented in order. There are two types of members in each node: data and reference. The term "reference" refers to a link to the next nodes in the series.

STUDY NOTE

HashSet, TreeSet and LinkedHashSet are the three examples of Set Interface Implementations.

This data structure enables efficient inserting and removing elements from any location in the sequence throughout the iteration.

Here are some key features of Linked List:

- The time it takes to gain access is linear.
- It is not possible to have faster access, such as random access.
- The main advantage of a linked list over a traditional array is that it is dynamic, meaning that list components can be quickly added or removed without having to reallocate or reorganise the entire structure.

Program: To demonstrate usage of ArrayList and LinkedList.

```
package unext.java_examples.Collections_example1;

import java.util.*;
public class ArrayLIst_LinkedList {

        public static void main(String args[]){

         List<String> al=new ArrayList<String>();//creating arraylist
         al.add("AAA");//adding object in arraylist
         al.add("BBB");
         al.add("CCC");
         al.add("DDD");

         List<String> ll=new LinkedList<String>();//creating linkedlist
         ll.add("One");//adding object in linkedlist
         ll.add("Two");
         ll.add("Three");
         ll.add("Four");

         System.out.println("arraylist: "+al);
         System.out.println("linkedlist: "+ll);
        }
}
```

3.1.1 ArrayList_LinkedList.java

Output:

```
arraylist: [AAA, BBB, CCC, DDD]
linkedlist: [One, Two, Three, Four]
```

Self-Assessment Questions - 2

5. Which interface handle sequence?
   a) set
   b) list
   c) map
   d) collection

6. Which of the following is an outdated class but still in use?
   a) Arraylist
   b) Vector
   c) Hashtable
   d) Both b and c are true

4. SET: THE INTERFACE

The Set interface is a collection of components that can't have duplicates. It represents sets like the ones below by modelling the mathematical set abstraction.

- A poker hand is made up of cards
- Courses that make up a student's schedule
- The operations that occur on a computer

The Set interface inherits only the methods from the Collection interface and adds a limitation to prevent duplicate elements.

"add" Operation of Set Interference

It is an optional procedure that adds the provided element to the Set if it is not already there. If the Set already includes the element, calling this function returns negative and leaves the Set untouched. This, together with the function Object () { [native code] } limitation, guarantees that sets never have identical elements.

"equals" operation of Set Interference

Set also adds a stricter constraint on the behaviour of the equals and HashCode operations, enabling Set instances to be effectively contrasted despite different implementation types. If two Set instances have the same elements, they are equivalent.

4.1. Implementations Of Set Interference

The following are examples of Set interface implementations:

- **HashSet:** Its items are stored in a hash table, and it is the best performing and fastest implementation. However, the sequence of repetition is not guaranteed. It is the most widely utilised method of implementation.
- **TreeSet:** It keeps its components in a red-black tree and sorts them depending on their contents. The elements will be arranged in ascending order, as per natural order. It is more time-consuming than HashSet.

- **LinkedHashSet:** This is a hash table with a linked list that runs through it. It keeps track of insertion order, or the order in which elements are added to the Set.

4.2 Limitations Of Using Hashset

It's important to remember that HashSet - "iteration is linear in the sum of the number of entries and the number of buckets (the capacity)".

As a result, selecting a capacity that is too high at the outset can waste both space and time. Selecting an annual production that is too low, on the other hand, costs time by replicating the database schema each time the capability is required to be increased.

If you don't provide an initial capacity, 16 is used as the default. Choosing an integer value as the maximum capacity has had certain advantages in the past. However, this is no longer the case because capacity is always rounded up to a power of two internally.

The following piece of code creates a HashSet with a capacity of 64.

Set<String> STR = new HashSet<String>(64);

Program: To demonstrate usage of hashset interface.

```
package unext.java_examples.Collections_Example2;

import java.util.*;
public class Hashset {

        public static void main(String args[]){
         //Creating HashSet and adding elements
           HashSet<String> hs=new HashSet();
                  hs.add("One");
                  hs.add("Two");
                  hs.add("Three");
                  hs.add("Four");
                  hs.add("Five");
                  hs.add("six");
                  Iterator<String> i=hs.iterator();
                  while(i.hasNext())
                  {
                  System.out.println(i.next());
                  }
        }
}
```

4.2.1 Hashset.java

Output:

Five

six

One

Four

Two

Three

5. MAP: THE INTERFACE

The Map interface, which specifies a mapping between a key and a result, is one of Java's most fascinating subjects. In Java, it's sometimes mistaken as a subtype of the Collection interface.

STUDY NOTE

A Map is a Java object that maps keys to data and is optimised for performance.

In Java, a Map is an entity that maps keys to data and is optimized for speed. Data is saved in key-value pairs, with each key being distinct. The name map comes from the fact that each key corresponds to a value. Map entries are a type of key-value pair.

- Because the Map interface is not a simple subclass of the Collection interface, it differs from the other collection types in terms of attributes and behaviours.
- It offers three different collection views: a set of keys, a set of key-value mappings, and a collection of values.
- There can be no duplicate keys in a Map, and each key can only map to one value. Some implementations (HashMap and LinkedHashMap) allow for null keys and values, whereas others do not (TreeMap).

For key-value association mapping, such as vocabularies, maps are ideal. The maps are used to do key lookups or to obtain and update elements based on their keys.

"Sorted" Map Interface

The SortedMap interface is a "Map" interface that keeps its mappings in increasing key sequence. It is the SortedSet's Map counterpart, and it improves on the "Map" interface.

Sorted maps are utilized for inherently ordered sets of key or monetary authorities, such as dictionaries and phone books.

"NavigableMap" Interface

The NavigableMap interface modifies the SortedMap interface and includes popular navigation methods such as lowerEntry, floorEntry, ceilingEntry, higherEntry, and others that provide the nearest good result for the supplied search targets. It improves on the SortedMap interface.

5.1 Implementation Of Map Interface

HashMap, TreeMap, and LinkedHashMap are the three general-purpose Map implementations. LinkedHashMap provides near-HashMap efficiency with insertion-order iteration; HashMap for top speed and no care for iteration order; TreeMap for SortedMap operations or key-ordered Collection-view iteration. Map's position is similar to Set's in this regard.

Everything that pertains to set implementations also applies to Map implementations.

- **HashMap:** Implementation of the "Map" interface based on hash tables. Null values and the null key are allowed. If you want maximum performance and don't worry about iteration order, this is the class to use. It is the most widely utilized method of implementation.
- **TreeMap:** It is a NavigableMap implementation based on a Red-Black tree. The keys in TreeMap are ordered according to their natural order. Whenever you need SortedMap methods or key-ordered Collection-view iteration, use this class.
- **LinkedHashMap:** It implements the Map interface as a hash table and linked List. Although it is slower to add and remove components than HashMap, you can expect quicker repetition with LinkedHashMap.

Program – To demonstrate working of MAP interface.

```
package unext.java_examples.Collections_Example3;

import java.util.*;
public class map {
    public static void main(String args[])
    {
        Map<String, Integer> hm
            = new HashMap<String, Integer>();

        hm.put("First", 100);
        hm.put("Second", 200);
        hm.put("Third", 300);
        hm.put("Fourth", 400);

        // accessing map elements
        for (Map.Entry<String, Integer> me : hm.entrySet()) {
            System.out.print(me.getKey() + ":");
            System.out.println(me.getValue());
        }
    }
}
```

5.1.1: map.java

Output:

```
Second:200
Third:300
First:100
Fourth:400
```

Self-Assessment Questions - 3

7. All methods must be implemented of an interface. [True/ False]
8. Which of the following is true about methods in an interface in Java?
   a) An interface can contain only abstract method
   b) We can define a method in an interface
   c) Private and protected access modifiers can also be used to declare methods in Java.
   d) None of the above
9. A list interface is also called as a _____ interface.

6. MISCELLANEOUS

In java, there are some legacy classes defined by java.util. They are vector, stack, hashtable, properties and soon. All these concepts are discussed in this section. Along with that, object-oriented programming languages require the ability to compare objects. Comparable objects also discussed in this section.

6.1 Comparable Objects

Every class that implements the Comparable interface must organize its objects in a specific order. The inherent order of the class is what it's called. To provide a logical comparison, the class's compareTo method must be implemented.

STUDY NOTE

Comparable is for items with a natural ordering, which means the object must understand how it will be sorted.

When you first begin to work with Java objects, you'll notice that you may compare them using the operators == and !=. An object is something you see on a computer screen, such as a button. You might inquire as to whether the item you just mouse-clicked is a certain button on your screen. This is accomplished using Java's equality operator.

A similar object can compare itself to another. The class must implement Java.lang package. To compare its instances, it has a comparable interface.

Compare To(Object obj): public int It's used to update the actual object to the one that's been specified. It reappears.

- If the current object is greater than the requested object, the value will be a positive integer.
- If the current object is less than the requested object, the value will be negative.
- If the current object is equal to the requested object, the value is zero.

Assume a Movie class with members who include things like rating, name, and year. Let's say we want to arrange a list of movies by release year. With the Movie class, we can extend the Comparable interface and modify the Comparable interface's compareTo() method.

Program: To demonstrate comparable interface

```
package unext.java_examples.Collections_Example4;

import java.util.*;
class Employee implements Comparable<Employee>{
	int id;
	String name;
	Employee(int id,String name){
		this.id=id;
		this.name=name;
	}
	public int compareTo(Employee e){
		if(id==e.id)
			return 0;
		else if(id>e.id)
			return 1;
		else
			return -1;
	}
}
//Creating a test class to sort the elements
public class compare{
	public static void main(String args[]){
		ArrayList<Employee> al=new ArrayList<Employee>();
		al.add(new Employee(15,"Emp-1"));
		al.add(new Employee(10,"Emp-2"));
		al.add(new Employee(11,"Emp-3"));

		Collections.sort(al);
		for(Employee e:al){
			System.out.println(e.id+" "+e.name+" ");
		}
	}
}
```

6.1.1: campare.java

Output:

```
10 Emp-2
11 Emp-3
15 Emp-1
```

Comparable VS Comparator

Comparable is intended for items with a natural ordering, which means that the object must know how it will be sorted. For instance, student roll numbers. Comparator interface sorting, on the other hand, is handled by a different class.

Comparable compares "this" references to object provided, whereas Comparator in Java compares two separate class objects provided.

If a class in Java implements the Comparable interface, then its Collection, whether a List or an Array, can be ordered efficiently using Collections. Arrays or sort() The items will be sorted using the sort() function in the natural order indicated by the CompareTo method.

The fact that we can only perform one comparison while using comparative is a key distinguishing feature. We can develop as many bespoke comparators as you wish for a particular type, each based on a particular view of what sorting entails. In the comparative example, we could only sort by one property, namely year, however in the comparator. We could utilize many attributes such as rating, name, and so on.

STUDY NOTE

Three main iterators in Java are: Enumeration, Iterator and ListIterator.

6.2 Iterators

An iterator across a set of items. In the Java Collections Framework, Iterator replaces Enumeration. Enumerations and iterators differ in two ways:

During the iteration, iterators allow the caller to eliminate values from the encompassing Collection with well-defined consequences.

- The names of the methods have been improved.
- The Java Collections Framework includes this interface.

The Collection framework in Java uses iterators to get elements one by one. In Java, there are many three iterators.

- Enumeration
- Iterator
- ListIterator

Enumeration

It's a user interface for retrieving components from legacy collections (Vector, Hashtable). The first Iterator in JDK 1.0 is Enumeration; rests are added in JDK 1.2 with extra capability.

The input channels to a SequenceInputStream are likewise specified using enumerations. By invoking the elements() function of the vector class on any vector object, we may build an Enumeration object.

The following are some of the constraints of Enumeration:

- Only legacy classes (Vector, Hashtable) can be enumerated. As a result, it isn't a universal iterator.
- Enumeration cannot be used to conduct "remove operations".
- It is only feasible to iterate in the forward direction.

Iterator

It's a global iterator because it may be used with any Collection object. We can execute both read and remove activities using Iterator. It's an enhanced version of Enumeration that adds the ability to remove an element from the List.

Iterator should be used to enumerate items in all Collection framework defined interfaces such as Set, List, Queue, and Deque, as well as all Map interface implemented classes. For the complete collection framework, the Iterator is now the only pointer accessible.

The Iterator () method of the Collection interface can be used to generate an iterator object.

There are, however, some limits to Iterator, which are stated below:

- It is only feasible to iterate in the forward direction.
- The Iterator does not support the replacement or insertion of new elements.

ListIterator

It only applies to List collection implementation classes like ArrayList, LinkedList, and so on. It can iterate in both directions. When we wish to enumerate List elements, we must use ListIterator. This cursor has additional methods and capabilities than an iterator. The ListIterator() method of the List interface can be used to construct a ListIterator object.

"l" denotes any List object, and ltr denotes a type. The "l" refers to the ListIterator interface. The ListIterator interface adds to the Iterator interface's functionality. As a result,

ListIterator has access to all three Iterator interface methods. There are also six additional methods.

6.3 Collections Utility Class

In basic terms, we may argue that we won't have to build our method too often if we use the Collections class utility methods. Furthermore, it makes our code shorter and more understandable, which is a good practice when thinking about programming.

Static methods that function on or return collections make up the Collections utility class. It includes polymorphic collection algorithms, "wrappers," which return a new collection supported by a systematic and organized collection, and more.

6.4 Array Utility Class

The Java.util.Arrays class provides easy-to-use array manipulation capabilities, including sorting and searching. There's also a technique for converting an array to a List collection. It's a utility class that contains several utility methods for working with arrays.

This module defines static methods for creating and accessing Java arrays dynamically. It only has static methods and methods from the Object class.

A container that holds a fixed number of values of a single type is called an array. It is required to provide an array size at the time of initialization, and the size we are providing is fixed. Here are some key aspects to remember regarding Java arrays.

- Arrays in Java are index-based collections.
- The array's files are recorded in memory in order from left to right.
- The array's default values are as follows. In the case of primitives, "0" is used, "null" is used in the case of Objects, and "false" is used in the case of Booleans.
- The size of an array is fixed.
- An array is a consistent data structure that can store uniform items.

STUDY NOTE

Three main iterators in Java are: Enumeration, Iterator and ListIterator.

- The real benefit is that it allows us to represent multiple values with the same name. As a result, code readability is improved.

6.5 Vector And Stack

Vector

In a resizable array way, the Vector class also represents a list of items. Its data structure is a dynamic array as well.

- It's a class from the past.
- It enables the insertion of duplicate items.
- It accepts an unlimited amount of null values.
- It keeps the order of insertion.
- It allows for the creation of diverse objects.
- Vector is the greatest option for retrieval.
- It's everything in sync. The Vector's methods are all synchronized. As a result, it is thread-safe.

Vector is a term that refers to a group of AbstractList is a class that implements List and RandomAccess. Vector uses the RandomAccess interface, which explains why its retrieval methods are so quick.

Stack

Vector has a child class called Stack.

- It operates on the LIFO (Last in, First Out) principle.
- In Java, it's the stack implementation.
- There are just 5 methods in Stack.

STUDY NOTE

HashTable has an added benefit on HashMap. The former can be synchronized. It has a subtype called Properties.

6.6 Hashtable

A hash table is implemented using the Hashtable class, which maps keys to values. As a key or a variable, any non-null object could be used. The objects used as keys should interface the hashCode and equals methods to store and recover entries from a Hashtable effectively.

Features of Hashtable

- It's comparable to HashMap, but with the added benefit of being synchronized.
- In a hash table, a key/value pair is stored.
- We provide an entity to be used as a key and the value we wish to associate with that key in Hashtable. The key is then hashed, and the hash code is used as the index for storing the value within the table.
- Hashtable class has a default capacity of 11 and a load factor of 0.75.
- HashMap does not support Enumeration, and Hashtable does not provide fail-fast Enumeration.

Parameters of HashTable

Let's look at the Java.util. Hashtable class's parameters.

- K: It refers to the kind of keys maintained.
- V: It refers to the mapped value type.

PROPERTIES:

Hashtable has a subtype called Properties. It's used to track lists of values with a String as the key and a string as the value.

Many additional Java classes use the Properties class. For example, when collecting core responsibility, it is the kind of object returned by System.getProperties().

If the data in a properties file is modified, no recompilation is needed: You don't need to restart the java class if any information in the properties file changes. It's used to keep track of information that needs to be updated frequently.

Program: To demonstrate the usage of Hash table.

```
package unext.java_examples.Collections_Example5;

import java.util.*;
public class Hashtable_Example{
       public static void main(String args[]){
                Hashtable<Integer,String> hm=new Hashtable<Integer,String>();
                hm.put(1,"A");
                hm.put(2,"B");
                hm.put(3,"C");
                hm.put(4,"D");
                for(Map.Entry m:hm.entrySet()){
                 System.out.println(m.getKey()+" "+m.getValue());
                }
       }
}
```

6.6.1 Hashtable_Example.class

Output:

```
4 D
3 C
2 B
1 A
```

Self-Assessment Questions - 4

10. The three main iterators in Java are _____, _____, _____.
11. The Iterator supports the replacement or insertion of new elements. [True/ False]
12. Java programs are:
    a) Faster than others
    b) Platform independent
    c) Not scalable
    d) Not reusable
13. Which of the following package stores all standard classes?
    a) lang
    b) java
    c) util
    d) java.packages
14. _____ is a subtype of Hashtable.

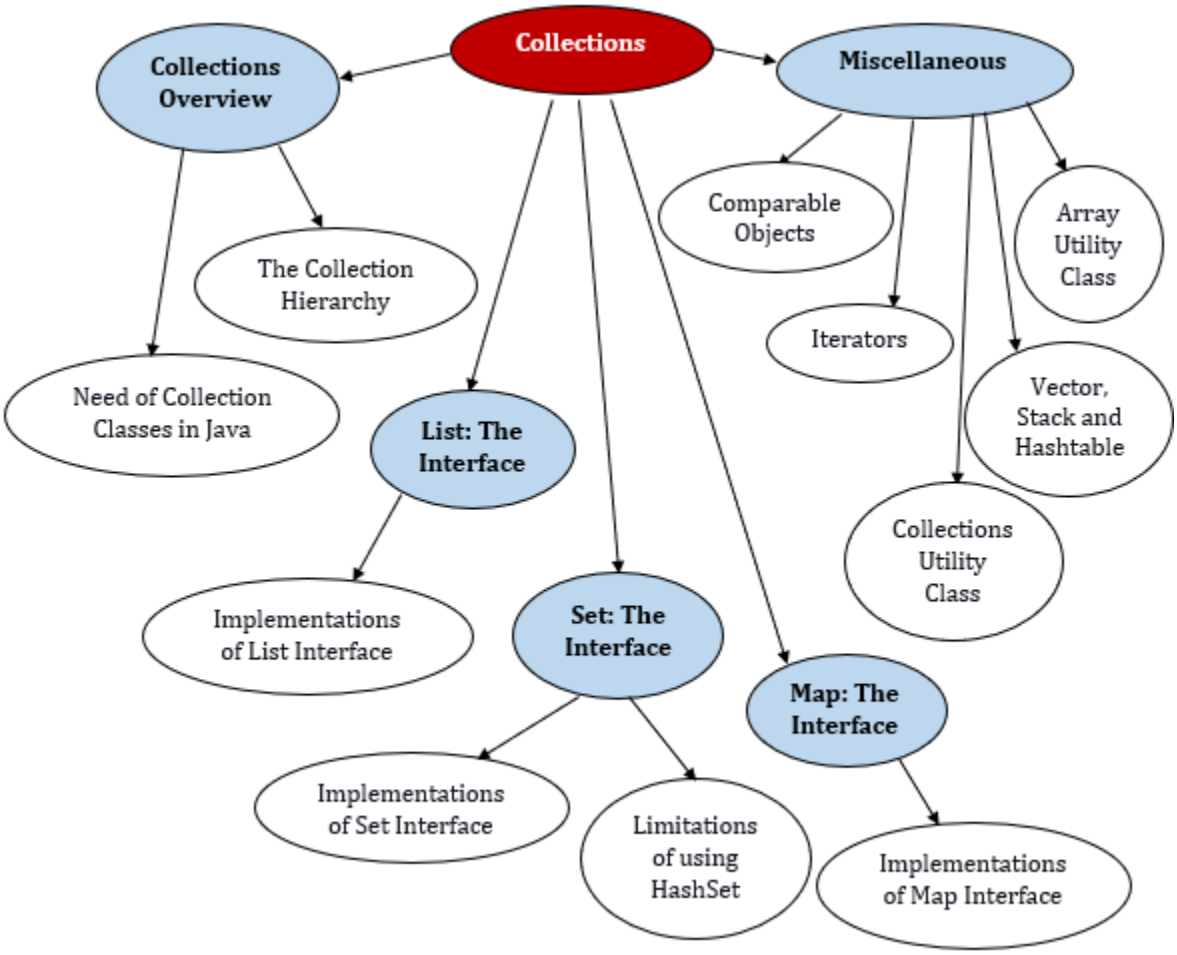


Fig 2: Conceptual Map

7. SUMMARY

- Java is a computer programming language that allows programmers to write computer commands using English-based instructions instead of input data.
- A collections framework is a unified architecture for describing and processing collections that allows them to be changed across implementations.
- There are three different types of collections. List, Set, and Map.
- The collections class in Java is a handy utility class for dealing with collections.
- The List interface is an ordered group of nodes that can store duplicate key and is Collection's child interface.
- Set Interface represents the unordered set of elements which doesn't allow us to store the duplicate items.
- We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.
- Map interface is a part of Java Collection Framework, but it doesn’t inherit Collection Interface.
- It describes an object that stores and retrieves elements in Key/Value pairs, with a Key determining their placement within the Map.
- Queue interface maintains the first-in-first-out order. It can be defined as an ordered list that is used to hold the elements which are about to be processed.
- There are various classes like PriorityQueue, Deque, and ArrayDeque which implements the Queue interface.

8. GLOSSARY

Hierarchy: In the hierarchy, every element can have one or more items beneath it. The Object class is at the top of the Java class hierarchy.

Implements: Optionally, a Java programming language keyword is provided in the class declaration to indicate any interfaces that the present class implements.

Collections: Collections are a specified set of classes or data structures for storing groups or collections of items.

List: A list is a type of data structure that is used to store items in a specific order. The elements are saved and returned in the same sequence in which they were added.

Set: Set saves just the unique elements. Because it is unpredictable and distinct, it will only retain one component if we add any duplicates.

Map: Map is a type of data structure that stores key-value pairs. The keys of any map are distinctive, although the values are not.

ArrayList: ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types.

Queue Interface: Queue interface maintains the first-in-first-out order. It can be defined as an ordered list that is used to hold the elements which are about to be processed.

Deque Interface: Extends the Queue interface. Deque stands for a double-ended queue which enables us to perform the operations at both the ends.

SortedSet Interface: SortedSet is the alternate of Set interface that provides a total ordering on its elements. The elements of the SortedSet are arranged in the increasing (ascending) order.

9. CASE STUDY

CASE STUDY: VIEWS IN JAVA COLLECTION API

The package java.util provides collections – sets, lists and maps. At the design of java.util, it is often called as the Java 'collections API'. The API organises its classes with a hierarchy of interfaces. The Java collections framework consists of:

- Collection interfaces representing different types of collections (Set, List, etc.)
- General purpose implementations (like ArrayList or HashSet)
- Abstract implementations to support custom implementations
- Algorithms defined in static utility methods that operate on collections (like Collections.sort(List list))
- Infrastructure interfaces that support collections (like Iterator).

Views are a sophisticated mechanism, very useful now and then, but dangerous. They break many of our basic conceptions about what sort of behaviour can occur in a well-formed object-oriented program. Three sorts of views are often identified, consistent with their purpose:

- Functionality extension. Some views are provided to enhance the functionality of an object without adding new methods to its class. Iterators fall in this category.
- Decoupling. Some views provide a subset of the functionality of the underlying collection. The keySet method on Map, for instance, returns a set that consists of the keys of the map. It therefore allows part of the code that is only concerned with the keys, and not with the values, to be decoupled from the remaining part of the specification of Map.
- Coordinate Transformation. The view provided by the subList method of List gives a kind of coordinate transformation.

Views are dangerous for two reasons:

- First, things change underneath you: call remove on an iterator and its underlying collection changes; call remove on a map and its key set view changes. This is a form of

abstract aliasing in which a mutation to one object causes another object, of a different type to change.

- Second, the specification of a method that returns a view often limits the kinds of mutation that are allowed. To make sure that your code works, you'll need to understand this specification.

There are some useful strategies that mitigate the complexity of views:

- Limit the scope in which the view is accessible.
- Prevent mutation of a view or underlying object by wrapping it using a method of the Collections class.

***Source**: web.mit.edu*

Discussion Questions:

1. Why views are needed and discuss different types of views.
2. Why views are dangerous? Explain with valid example.

10. TERMINAL QUESTIONS

SHORT ANSWER QUESTIONS

Q1. Explain List interface

Q2. What are the limitations of Hashset?

Q3. List the categories of collection class.

Q4. Define Properties class in Java collections.

LONG ANSWER QUESTIONS

Q1. Why do you need collection classes? Justify.

Q2. Elaborate hierarchy of Collection framework.

Q3. Mention and describe different kinds of iterators in Java collections.

Q4. Explain difference between comparable and comparator interfaces.

11. ANSWERS

SELF ASSESSMENT QUESTIONS

1. Extends Iterable interface
2. True
3. Collection
4. C. Map
5. B. List
6. D. Both b and c are true
7. True
8. An interface can contain only abstract method
9. Sequence
10. Enumeration, Iterator, ListIterator
11. False
12. Platform independent
13. Java
14. Properties

TERMINAL QUESTIONS

SHORT ANSWER QUESTIONS

Answer 1: List Interface: This interface represents a collection of elements that are grouped chronologically. Lists preserve element order, which means that the order in which we add elements is preserved, and the same sequence gets preserved while retrieving elements. At any point in the List, we can insert elements. In Java, the List enables the storage of duplicate elements. The list interface is implemented by three concrete subclasses: ArrayList, Vector, and LinkedList.

Answer 2: It's important to remember that HashSet - "iteration is linear in the sum of the number of entries and the number of buckets (the capacity)".

As a result, selecting a capacity that is too high at the outset can waste both space and time. Selecting an annual production that is too low, on the other hand, costs time by replicating the database schema each time the capability is required to be increased.

If you don't provide an initial capacity, 16 is used as the default. Choosing an integer value as the maximum capacity has had certain advantages in the past. However, this is no longer the case because capacity is always rounded up to a power of two internally.

Answer 3: Collections are divided into three categories. They are:

- List,
- Set, and
- Map.

Answer 4: Properties: Hashtable has a subtype called Properties. It's used to track lists of values with a String as the key and a string as the value.

Many additional Java classes use the Properties class. For example, when collecting core responsibility, it is the kind of object returned by System.getProperties().

LONG ANSWER QUESTIONS

Answer 1: Need for collection classes:

- Collection classes in Java provide a more high-level interface than arrays.
- The size of an array is fixed. The size of collections (see ArrayList) is variable.
- Constructing complex data structures (e.g., hash tables) properly on top of raw arrays is difficult. It is provided for free by the basic HashMap.
- You can pick between alternative implementations for the same Set of services, such as ArrayList vs. LinkedList, HashMap vs. TreeMap, synchronized, and so on.
- Furthermore, arrays allow for covariance: changing an array element is not assured to succeed due to typing issues that can only be detected at runtime. In arrays, generics solve this difficulty.

Answer 2: The comprehensive collection framework hierarchy comprises four fundamental interfaces: Collection, List, Set, Map, and two specific interfaces for sorting called SortedSet and SortedMap.

1. **List Interface:**

- This interface represents a collection of elements that are grouped chronologically.
- Lists preserve element order, which means that the order in which we add elements is preserved, and the same sequence gets preserved while retrieving elements.
- At any point in the List, we can insert elements. In Java, the List enables the storage of duplicate elements.
- The list interface is implemented by three concrete subclasses: ArrayList, Vector, and LinkedList.

2. **Set Interface:**

- This interface is made up of a variety of parts, each of which is distinct. It's used to keep a collection of one-of-a-kind pieces.
- While storing objects, the Set interface does not keep track of their order, and we may not get the same order when retrieving them. A set's elements can be arranged in any sequence.
- Duplicate items are not permitted in Set.

3. **Map Interface:** Map interface is a part of Java Collection Framework, but it doesn't inherit Collection Interface. It describes an object that stores and retrieves elements in Key/Value pairs, with a Key determining their placement within the Map.

- For storing key-value pairs, Map employs a hashing algorithm.
- It does not allow multiple keys to be stored, but duplicate values are permitted.

4. **SortedSets Interface:**

    - This interface extends a set whose Iterator traverses its elements in the order in which they appear in the Set.
    - TreeSet implements the sorted interface.

5. **Sorted Map Interface:**

   The elements of this Map are maintained in their order of things in this interface. TreeMap classes follow the Map interface, which is extended by it.

Answer 3: The Collection framework in Java uses iterators to get elements one by one. In Java, there are many three iterators.

- Enumeration
- Iterator
- ListIterator

Enumeration

It's a user interface for retrieving components from legacy collections (Vector, Hashtable). The first Iterator in JDK 1.0 is Enumeration; rests are added in JDK 1.2 with extra capability. The input channels to a SequenceInputStream are likewise specified using enumerations. By invoking the elements() function of the vector class on any vector object, we may build an Enumeration object.

Iterator

It's a global iterator because it may be used with any Collection object. We can execute both read and remove activities using Iterator. It's an enhanced version of Enumeration that adds the ability to remove an element from the List. Iterator should be used to enumerate items in all Collection framework defined interfaces such as Set, List, Queue, and Deque, as well as all Map interface implemented classes.

ListIterator

It only applies to List collection implementation classes like ArrayList, LinkedList, and so on. It can iterate in both directions. When we wish to enumerate List elements, we must use ListIterator. This cursor has additional methods and capabilities than an iterator. The ListIterator() method of the List interface can be used to construct a ListIterator object.

Answer 4: Comparable is intended for items with a natural ordering, which means that the object must know how it will be sorted. For instance, student roll numbers. Comparator interface sorting, on the other hand, is handled by a different class.

Comparable compares "this" references to object provided, whereas Comparator in Java compares two separate class objects provided.

If a class in Java implements the Comparable interface, then its Collection, whether a List or an Array, can be ordered efficiently using Collections. Arrays or sort() The items will be sorted using the sort() function in the natural order indicated by the CompareTo method.

The fact that we can only perform one comparison while using comparative is a key distinguishing feature. We can develop as many bespoke comparators as you wish for a particular type, each based on a particular view of what sorting entails. In the comparative example, we could only sort by one property, namely year, however in the comparator. We could utilise many attributes such as rating, name, and so on.

12. SUGGESTED BOOKS AND E-REFERENCES

BOOKS:

- Rafael H. Bordini, L. B. (2006), A Survey of Programming Languages and Platforms for Multi- Agent Systems, Informatica, 33-44.
- Singh, T. (2012), New Learning Methodology for Student of Java Programming Language, International Journal of Engineering Research and Development, 17-19.
- Joshua Bloch, (2018), Effective Java, 3rd Edition.
- Brian Goetz with Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, and Doug Lea, (2006), Java Concurrency in Practice, 1st Edition.

E-REFERENCES:

- Java Collections Framework, viewed on 30th Sep, 2021, https://www.javatpoint.com/collections-in-java
- Java – Collections Framework, viewed on 30th Sep, 2021, https://www.tutorialspoint.com/java/java_collections.htm
- Comparable vs Comparator in Java, viewed on 30th Sep, 2021, https://www.geeksforgeeks.org/comparable-vs-comparator-in-java/
- Java Iterator: Learn To Use Iterators In Java With Examples, viewed on 30th Sep, 2021, https://www.softwaretestinghelp.com/java/learn-to-use-java-iterator-with-examples/