Collection Framework Part 1

Why collection framework ?

1. We should know the size in advance of our data structure which not possible in array

```
class ObjectArrays {
public static void main(String[] args){
Object[] a = new Object[10];
System.out.println(a[0]);
}
}
```

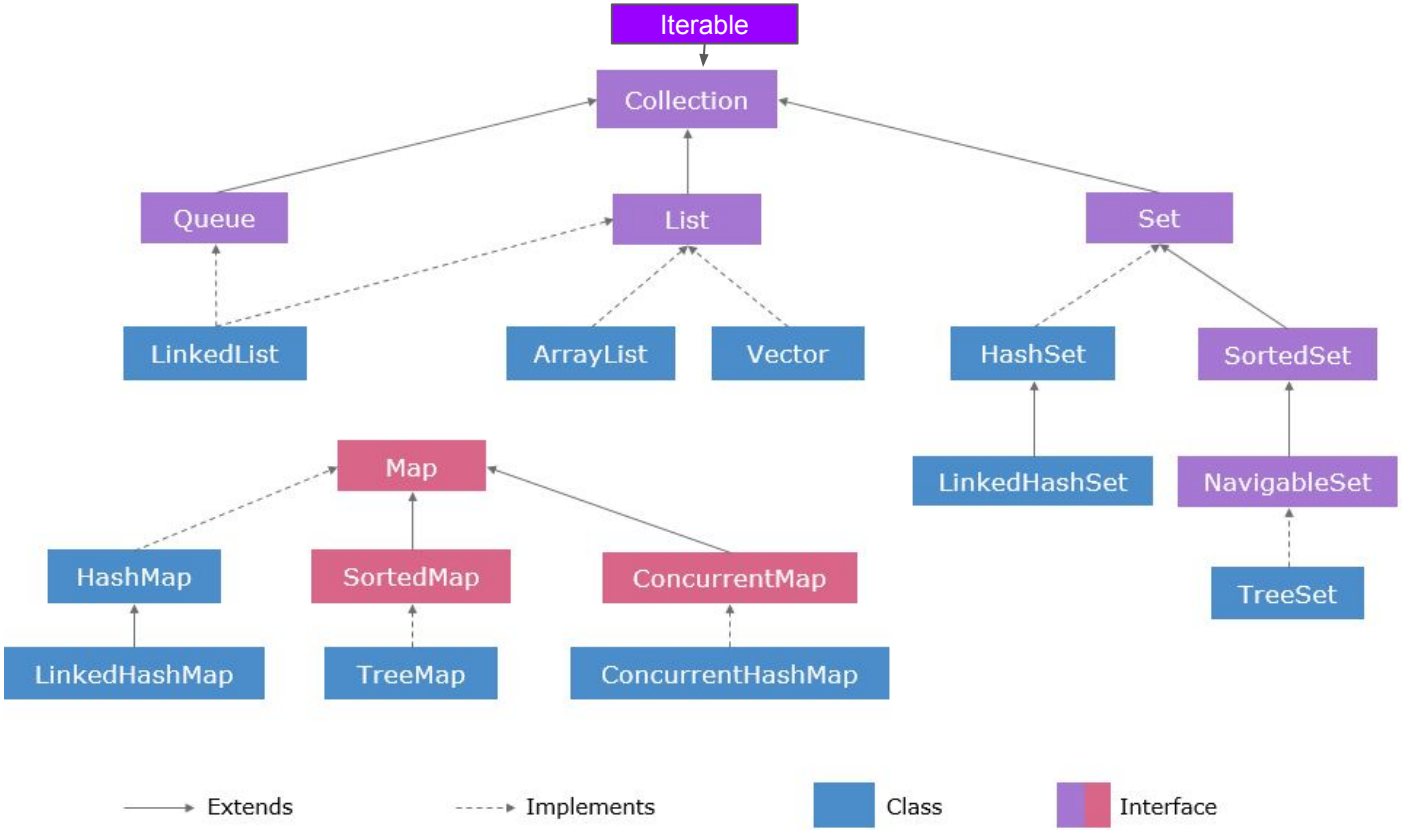

2. Object Arrays don't provide support readymade methods for every requirement. (i.e. Programmer is responsible to write the logic for such kind of requirement).

```
class ObjectArrays {
public static void main(String[] args){
Object[] a = new Object[10];
System.out.println(a[0].add("A")); //Here, add() method will give error it does not provide readymade facility
}
```

Introduction to Collection

- Collections Framework provides a well-designed set of interfaces and classes for storing and manipulating groups of data as a single unit, a collection.
- It provides a convenient API to many of the ADTs like maps, sets, lists, trees, arrays, hash tables, and other collections.
- It provides a standard programming interface to many of the most common abstractions, without burdening the programmer with too many procedures and interfaces.
- Due to Object-oriented design, the classes in the Framework encapsulate both the data structures and the algorithms associated with the ADTs.

Collection Framework



The collections framework offers many interfaces and classes for manipulating and representing collections. All these are in **java.util** package.

Arrays vs Collection

Arrays	Collection
Arrays are fixed in size that is once we create an array we cannot increased or decreased based on our requirement.	Collection are grow able in nature that is based on our requirement. We can increase or decrease of size.
With respect to memory Arrays are not recommended to use.	With respect to memory collection are recommended to use.
With respect to performance Arrays are recommended to use.	With respect to performance collection are not recommended to use.
Arrays can hold only homogeneous data types elements.	Collection can hold both homogeneous and heterogeneous elements.
There is no underlying data structure for arrays and hence readymade method support is not available.	Every collection class is implemented based on some standard data structure and hence for every requirement readymade method support is available being a performance. we can use these method directly and We are not responsible to implement these methods.
Arrays can hold both object and primitive.	Collection can hold only object types but primitive.

Collection methods

- The root of the collection hierarchy is java.util.Collection<E> interface.
- It has the basic methods for manipulating elements in a collection.

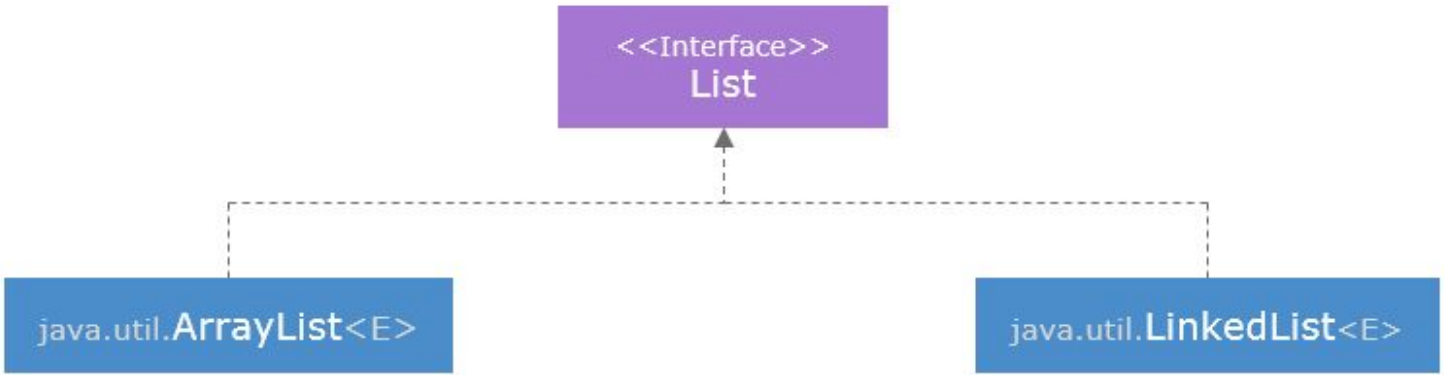
Methods	Description
boolean add(E e)	Adds a new element to the collection.
boolean addAll(Collection e)	Adds all the elements in e to the collection.
void clear()	Deletes all the elements from the collection.
boolean contains(Object O)	Returns true if the collection has the element O.
boolean isEmpty()	Returns true if the collection is empty.
Iterator<E> iterator()	Returns an iterator for the elements in the collection.
boolean remove(0bject o)	Removes the element o from the collection.
int size()	Returns the total number of elements in the collection.
Object[] toArray()	The elements in the collection are returned as an array of Objects.

List Interface

- The java.util.List<E> represents an ordered collection of elements.
- Like arrays, it allows index-based access to its elements. Also, duplicate elements are allowed.

Name	Description
void add(int index, E element)	At the specified index, the element will be added
boolean addAll(int index, Collection c)	Adds all elements in c to the list starting from the specified index
E get(int index)	Gets the element from the given index in the list
E remove(int index)	At the given index the element will be removed
ListIterator<E> listIterator()	A list iterator is returned to iterate through the elements in the list

Classes Implementing the List interface



- Array implementation of the List interface
- Allows random access, and hence fast
- Addition and removal of elements is time consuming

- Doubly Linked List implementation of the List interface
- Allows sequential access from the front and the back, and hence slower
- Addition and removal of elements is easier and faster

ArrayList

- Using ArrayList issues of size limitations and extra processing code for item removal can be overcome. ArrayList is **heterogeneous**.

```
ArrayList food = new ArrayList();
food.add("Noodles"); // Adding the elements
food.add(new Integer(5));
System.out.println(food);
System.out.println(food.get(1)); //1 is the index of element to be displayed
for (int i = 0; i < food.size(); i++) { //accessing elements
        System.out.println(food.get(i));
}
for (String temp: food) {
        System.out.println(temp);
}
```

Generics

- Generics help in creating classes, interfaces, and/or methods in which the type of object on which they operate is mentioned as a parameter.

```
ArrayList<String> nameList = new ArrayList<String>();
```

Accessing Collection Elements

1. Iterator interface
2. Enhanced for loop / for-each
3. for loop

Arraylist methods

Method	Description
void add(int index, Object element)	inserts the specified element to the specified position in the ArrayList
boolean add(Object o)	appends the specified object to the end of the ArrayList
boolean addAll(Collection c)	appends all the elements of the specified collection to the end of the ArrayList
boolean addAll(int index, Collection c)	inserts all the elements of the specified collection into the ArrayList, starting at the specified position
Iterator<E> iterator()	returns an iterator over the elements in the set
void clear()	removes every element of the ArrayList
boolean contains(Object element)	returns True if the specified element is present in the ArrayList
Object get(int index)	returns the element present in the specified position
int indexOf(Object element)	returns the first occurrence of the specified element
boolean isEmpty()	returns True if the ArrayList has no elements
int lastIndexOf(Object element)	returns the last occurrence of the specified element
Object remove(int index)	removes the element present at the specified index of the ArrayList
Object set(int index, Object element)	replaces the element at the specified index in the ArrayList with the specified element
int size()	returns the number of elements in the ArrayList

Arraylist methods

```
list.add("Noodles"); // Adding the elements

list.set(1, "Pizza"); // Modifying the element at a specified index

list.contains(2, "Pasta"); // Checking whether the element is present or not

list.remove(2); // Removing the element from the second index

list.isEmpty();

list.size()

list.addAll(newNames);

Object[] namesArray = list.toArray();

list.clear();
```

Iterator

- This method returns an Iterator type object which is used iterator over elements of the list.
- It is an implementation of the Iterator interface's iterate() method.
- An Iterator is used to traverse elements of a collection in a **forward-only** direction.

Method	Description
boolean hasNext()	returns true if the iteration has more elements
E next()	returns the next element in the iteration
void remove()	removes the current element in the iterator

ListIterator

- A ListIterator is used for traversing a list in both forward and backward directions.

Method	Description
boolean hasPrevious()	returns true if this list iterator has more elements when traversing the list in the reverse direction
E previous()	returns the previous element in the list and moves the cursor position backwards
boolean hasNext()	returns true if the iteration has more elements
E next()	returns the next element in the iteration
void remove()	removes the current element in the iterator

Vector

1. The Vector class in Java is part of the java.util package and implements the List interface.
2. It is a synchronized collection that dynamically resizes as elements are added or removed.
3. Key Features of Vector:
   a. - Supports dynamic array-like functionality.
   b. - Thread-safe (synchronized), but slower compared to ArrayList.
   c. - Allows duplicate elements.


```
Vector<T> food = new Vector<T>();
```

LinkedList

- A LinkedList in Java is a part of the java.util package and implements the List and Deque interfaces. It is a linear data structure in which elements are stored as nodes, and each node contains:
  - **Data**: The actual element value.
  - **Pointer/Reference**: A reference to the next (and/or previous) node in the list.
- Unlike an array, LinkedList provides dynamic memory allocation, allowing for easy insertion and deletion of elements without resizing or shifting.


```
LinkedList<String> myList = new LinkedList<String>();
```

LinkedList methods

Method	Description	Example
`boolean add(E e)`	Adds the specified element to the end of the list.	`list.add("A");`
`E get(int index)`	Returns the element at the specified index.	`list.get(1);`
`boolean remove(Object o)`	Removes the first occurrence of the specified element from the list.	`list.remove("A");`
`E removeFirst()`	Removes and returns the first element of the list.	`list.removeFirst();`
`E removeLast()`	Removes and returns the last element of the list.	`list.removeLast();`
`boolean contains(Object o)`	Returns `true` if the list contains the specified element.	`list.contains("A");`
`int size()`	Returns the number of elements in the list.	`list.size();`
`boolean isEmpty()`	Returns `true` if the list contains no elements.	`list.isEmpty();`
`void clear()`	Removes all elements from the list.	`list.clear();`
`E getFirst()`	Returns the first element of the list.	`list.getFirst();`
`E getLast()`	Returns the last element of the list.	`list.getLast();`
`ListIterator<E> listIterator()`	Returns a list iterator over the elements in the list (from the beginning).	`list.listIterator();`

LinkedList Operations

```
LinkedList<String> list = new LinkedList<>();

        list.add("Java");       // Adding elements to the LinkedList

        list.add("Python");

        System.out.println("Original LinkedList: " + list);       // Displaying the LinkedList

        list.add(2, "C#");       // Adding an element at a specific index

        System.out.println("After adding C# at index 2: " + list);

        list.remove("Python"); // Remove by element

        list.remove(1);       // Remove by index

        System.out.println("After removals: " + list);

        String firstElement = list.getFirst();       // Accessing elements in the LinkedList

        String lastElement = list.getLast();
```

Collections Methods

- sort(),
- reverse(),
- max(),
- min(),
- frequency(),
- swap()
- shuffle()

Collections vs Collection

Feature	Collection (Interface)	Collections (Class)
Definition	A root interface in the Java Collection Framework.	A utility class with static methods.
Package	`java.util`	`java.util`
Type	Interface	Final Class
Purpose	Provides a framework for data storage structures.	Provides utility methods to manipulate collections.
Implementation	Needs to be implemented by concrete classes like `ArrayList`, `HashSet`, etc.	No implementation needed, just use static methods.
Inheritance	Extended by `List`, `Set`, `Queue`.	No inheritance, provides helper functions.
Example Usage	`Collection<String> list = new ArrayList<>();`	`Collections.sort(list);`
Modifiability	Allows adding, removing, and updating elements.	Provides operations like sorting, searching, and shuffling.
Common Methods	`add()`, `remove()`, `size()`, `isEmpty()`, `iterator()`.	`sort()`, `reverse()`, `shuffle()`, `max()`, `min()`.
Supports Generics	Yes	Yes
Can Store Elements?	Yes	No (only provides operations)

Sorting using Comparable Interface

- Comparable Interface is from java.lang package which has only one method compareTo(Object).

  public int compareTo(<T> object)

- String class and Wrapper class implements Comparable Interface by default.
- This method returns an integer and if it returns -
  - zero, it means the current object is equal to the passed object
  - positive integer, it means the current object is greater than the passed object
  - a negative integer means the current object is lesser than the passed object.

Sorting using Comparator Interface

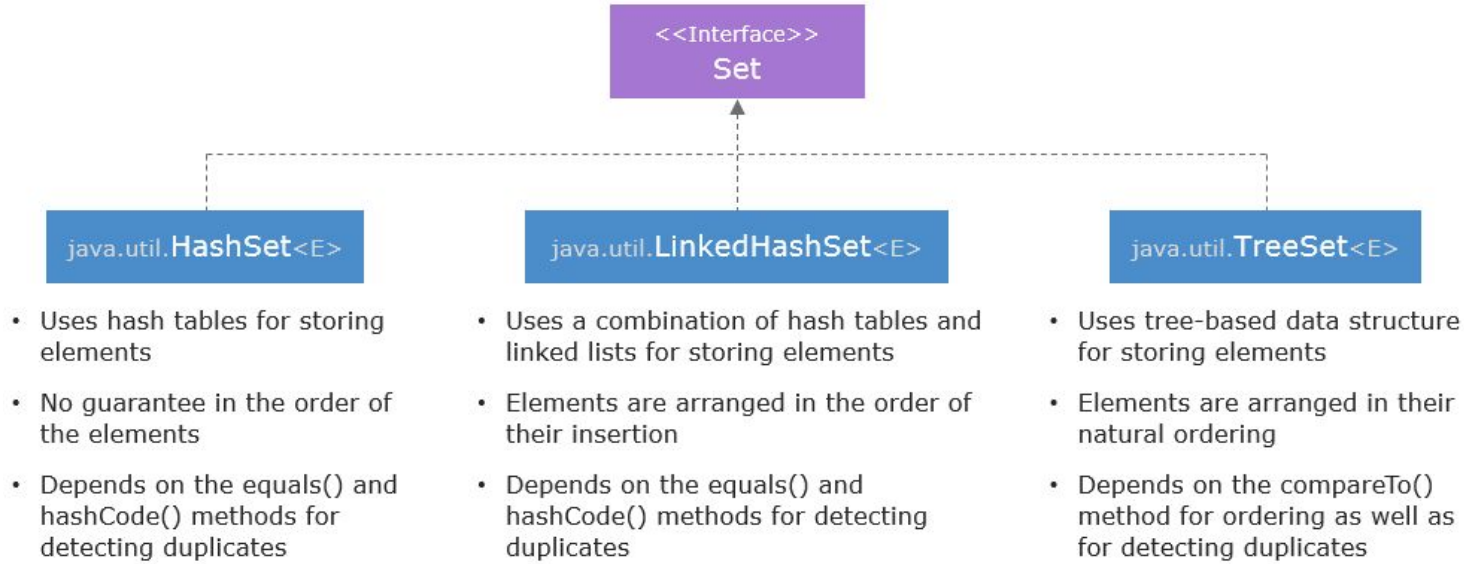
- Comparator Interface is from java.util package which has only one method compareTo(Object).

  int compare(T o1, T o2);

- This method returns an integer and if it returns -
  - zero, it means the current object is equal to the passed object
  - positive integer, it means the current object is greater than the passed object
  - a negative integer means the current object is lesser than the passed object.

Introduction of **Set**

- Set Interface represents an unordered collection with unique elements. It is present in the **java.util package**. Just like List, Set also has several implementations.It implementations allow a single null element.



Assignments

1. Write a Java program to Store a list of 20 city names in an ArrayList Search for a specific city. Display whether the city is found or not.
2. Write a Java program to Add 100 random numbers to an ArrayList. Iterate through the list using:
   - for loop
   - for-each loop
   - Iterator //dont try as of now
   - ListIterator. //dont try as of now

   Print the elements using each approach.
3. Write a Java program to Add a list of 15 random integers to a LinkedList. Sort the elements in ascending order and display the result.
4. Write a Java program Add numbers 1 to 15 in a LinkedList. Find and print the middle element of the list without using the size method.
5. Create a Student class with attributes: id, name, and marks. Implement the Comparable interface to sort students based on their marks in ascending order. Display the sorted student records.

Assignments

6. Write a Java program that defines an Employee class with fields: id, name, and salary. Create two comparators:

- NameComparator - to sort employees alphabetically by name.
- SalaryComparator - to sort employees in descending order of salary.

Add employee objects to an ArrayList and sort using both comparators. Display the results.

Thanks