Object Oriented Programming with Java (Subject Code: BCS-403)

Unit 4

Lecture 28

Lecture 28

- Collection in Java
- Collection Framework in Java
- Hierarchy of Collection Framework

Collection in Java

- The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.
- Java Collections can achieve all the operations that we perform on a data such as searching, sorting, insertion, manipulation, and deletion.
- Java Collection means a single unit of objects.
- Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

Collections in Java

The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

In Java, collections are part of the Java Collections Framework (JCF), which is a unified architecture for representing and manipulating groups of objects. The JCF provides several interfaces and classes that implement different types of collections, such as lists, sets, queues, and maps.

Why use collections instead of arrays?

Arrays in Java have several limitations:

Fixed Size: Once an array is created, its size cannot be changed. Collections, however, can grow or shrink dynamically as needed.

Type Safety: Collections can enforce type safety using generics, reducing runtime errors related to type casting.

Utility Methods: Collections provide various utility methods for common operations like searching, sorting, and manipulating data.

Performance: Collections are optimized for performance with various implementations for different use cases (e.g., fast random access, quick insertion/deletion, etc.).

What is Collection in Java

A Collection represents a single unit of objects, i.e., a group.

In Java, A Collection is a fundamental concept and a root interface in the Java Collections Framework (JCF). It represents a group of objects (known as elements) in a single unit including Interfaces and its implementations, i.e., classes and Algorithm.

The Collection interface is part of the java.util package and serves as the foundation for various data structures and algorithms that operate on collections of objects.

Benefits of using JCF

Reusability: Provides reusable data structures and algorithms.

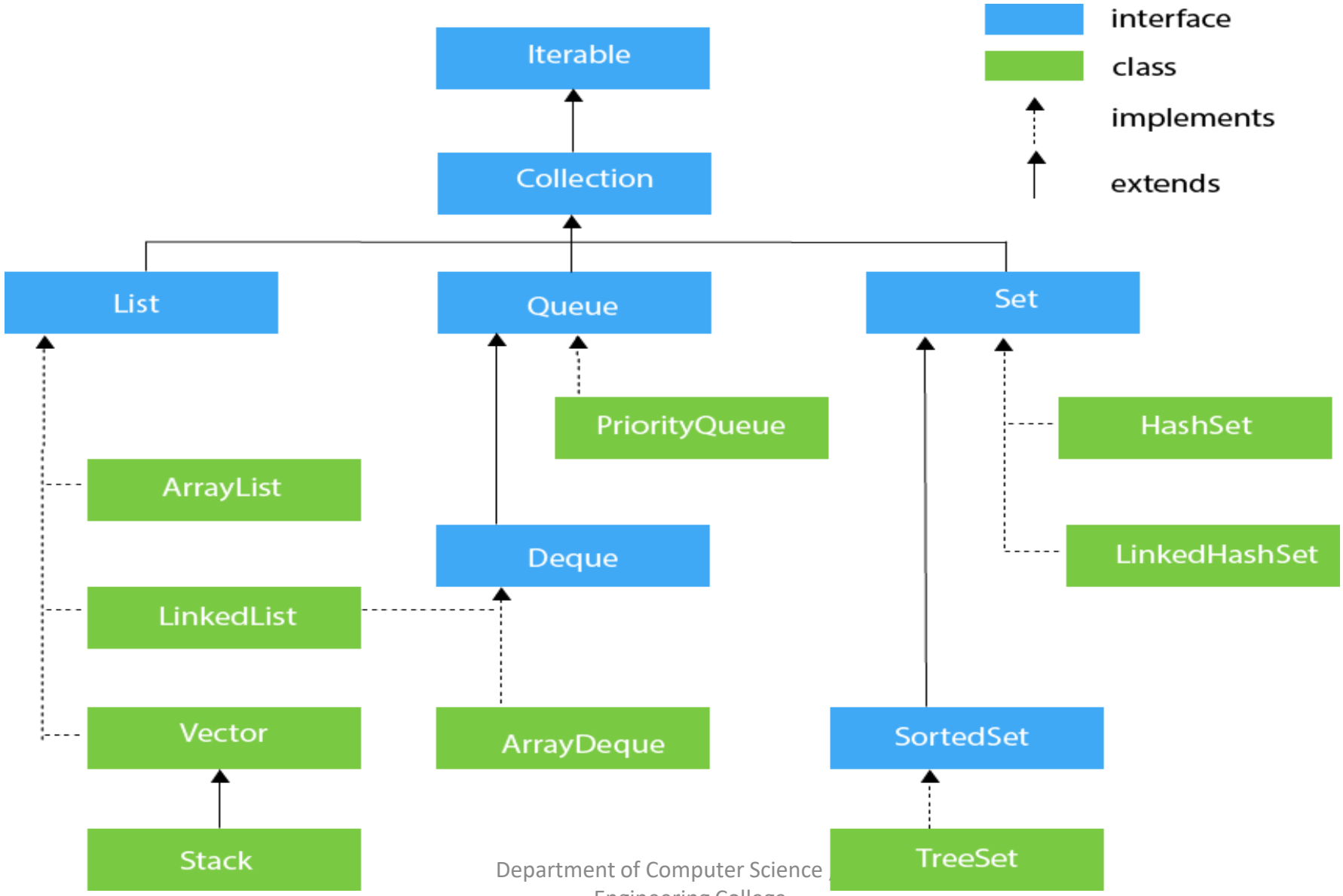
Interoperability: Allows different types of collections to work together seamlessly.

Performance: Offers various implementations optimized for different performance needs.

Flexibility: Supports dynamic resizing and allows collections to grow and shrink as needed.

Type Safety: Uses generics to ensure type safety at compile time.

Hierarchy of the Collection Framework in Java



Hierarchy of Collection Framework

- This image represents the hierarchy of interfaces and classes in the Java Collections Framework. It illustrates the relationships between different collection types and their implementations.
- At the top, we have the Iterable interface, which provides a way to iterate over elements in a collection. It is the root interface for all collection types.
- Moving down, we have the Collection interface, which is a child of Iterable and represents a group of objects. It defines the common methods and behaviors for all collection types.

Hierarchy of Collection Framework

Underneath Collection, we have three main interfaces:

List: An ordered collection that allows duplicate elements. Its implementations are ArrayList, LinkedList, Vector, and Stack.

Queue: A collection designed for holding elements prior to processing. Its implementations are PriorityQueue, Deque, and ArrayDeque.

Set: An unordered collection that does not allow duplicate elements. Its implementations are HashSet, LinkedHashSet, SortedSet, and TreeSet.

Hierarchy of Collection Framework

Each of these interfaces has its own concrete implementations, which are represented by the colored boxes in the image.

ArrayList, LinkedList, Vector, and Stack implement the List interface.

PriorityQueue, Deque, and ArrayDeque implement the Queue interface.

HashSet, LinkedHashSet, SortedSet, and TreeSet implement the Set interface.

These implementations provide different characteristics and performance trade-offs depending on the use case and requirements of the application.

Hierarchy of the Collection Framework in Java

- The utility package, (java.util) contains all the classes and interfaces that are required by the collection framework.
- The collection framework contains an interface named an iterable interface which provides the iterator to iterate through all the collections.
- This interface is extended by the main collection interface which acts as a root for the collection framework.
- All the collections extend this collection interface thereby extending the properties of the iterator and the methods of this interface.

Iterator interface

Iterator interface provides the facility of iterating the elements in a forward direction only.

Methods of Iterator interface

No.	Method	Description
1	public boolean hasNext()	It returns true if the iterator has more elements otherwise it returns false.
2	public Object next()	It returns the element and moves the cursor pointer to the next element.
3	public void remove()	It removes the last elements returned by the iterator. It is less used.

Iterable Interface

The Iterable interface is the root interface for all the collection classes.

The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.

It contains only one abstract method. i.e.,

Iterator<T> iterator()

It returns the iterator over the elements of type T.

Collection Interface

- The Collection interface is the interface which is implemented by all the classes in the collection framework.
- It declares the methods that every collection will have.

Methods of Collection interface

No.	Method	Description
1	public boolean add(E e)	It is used to insert an element in this collection.
2	public boolean addAll(Collection<? extends E> c)	It is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	It is used to delete an element from the collection.
4	public boolean removeAll(Collection<?> c)	It is used to delete all the elements of the specified collection from the invoking collection.

No.	Method	Description
5	public int size()	It returns the total number of elements in the collection.
6	public void clear()	It removes the total number of elements from the collection.
7	public boolean contains(Object element)	It is used to search an element.
8	public boolean containsAll(Collection<?> c)	It is used to search the specified collection in the collection.

List Interface

- This is a child interface of the collection interface.
- This interface is dedicated to the data of the list type in which we can store all the ordered collections of the objects.
- This also allows duplicate data to be present in it.
- This list interface is implemented by various classes like ArrayList, Vector, Stack, etc.

- List <data-type> list1= **new** ArrayList();
- List <data-type> list2 = **new** LinkedList();
- List <data-type> list3 = **new** Vector();
- List <data-type> list4 = **new** Stack();

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.
- There are various classes like PriorityQueue, Deque, and ArrayDeque which implements the Queue interface.

Queue interface can be instantiated as:

```
Queue<String> q1 = new PriorityQueue();
Queue<String> q2 = new ArrayDeque();
```

Set Interface

- Set Interface in Java is present in java.util package. It extends the Collection interface.
- It 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.

Set can be instantiated as:

```
Set<data-type> s1 = new HashSet<data-type>();
Set<data-type> s2 = new LinkedHashSet<data-type>();
Set<data-type> s3 = new TreeSet<data-type>();
```

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.
- The SortedSet provides the additional methods that inhibit the natural ordering of the elements.

The SortedSet can be instantiated as:

SortedSet<data-type> set = new TreeSet();