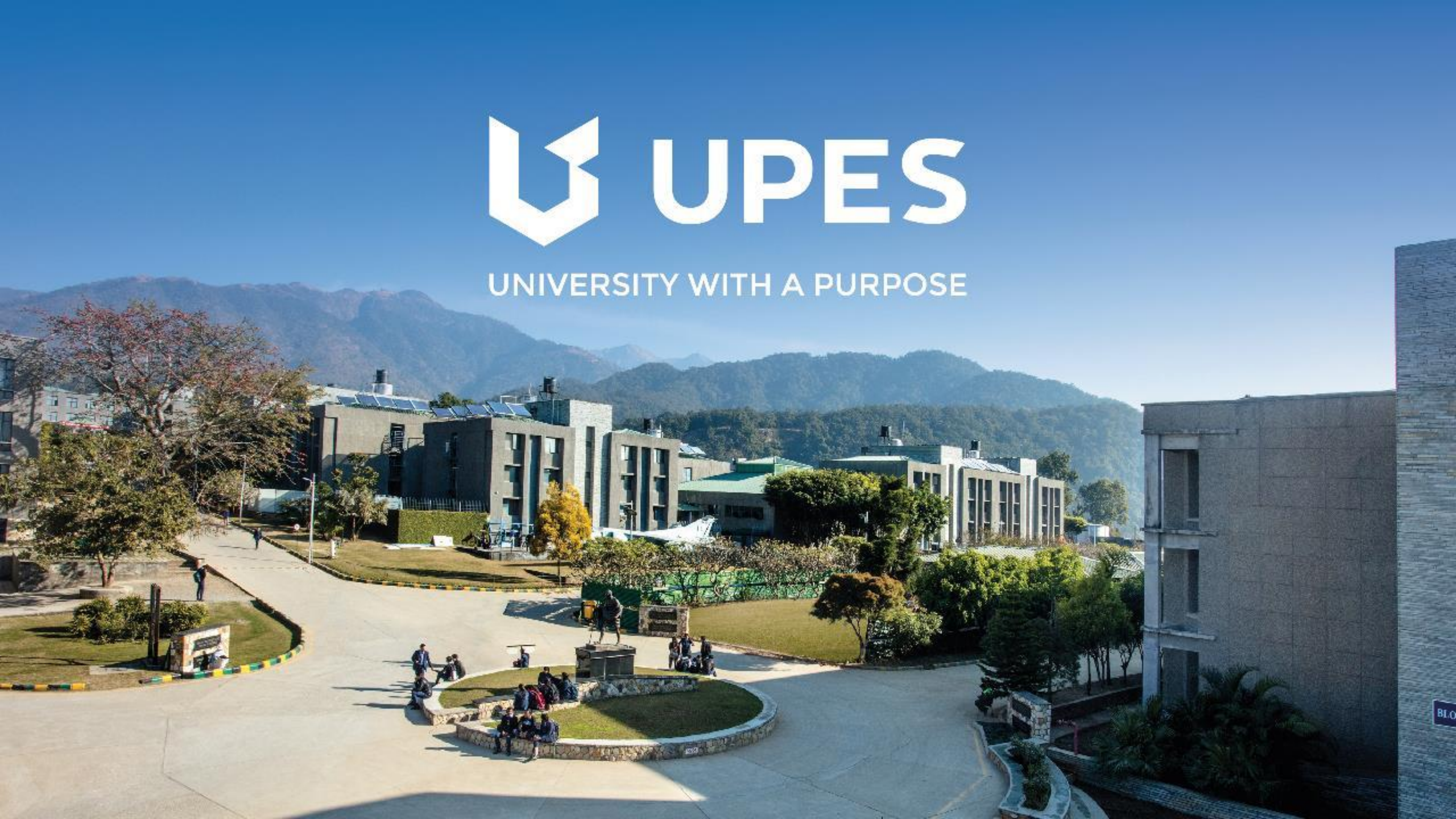

UPES

UNIVERSITY WITH A PURPOSE

Collection Framework-II

Collection Framework

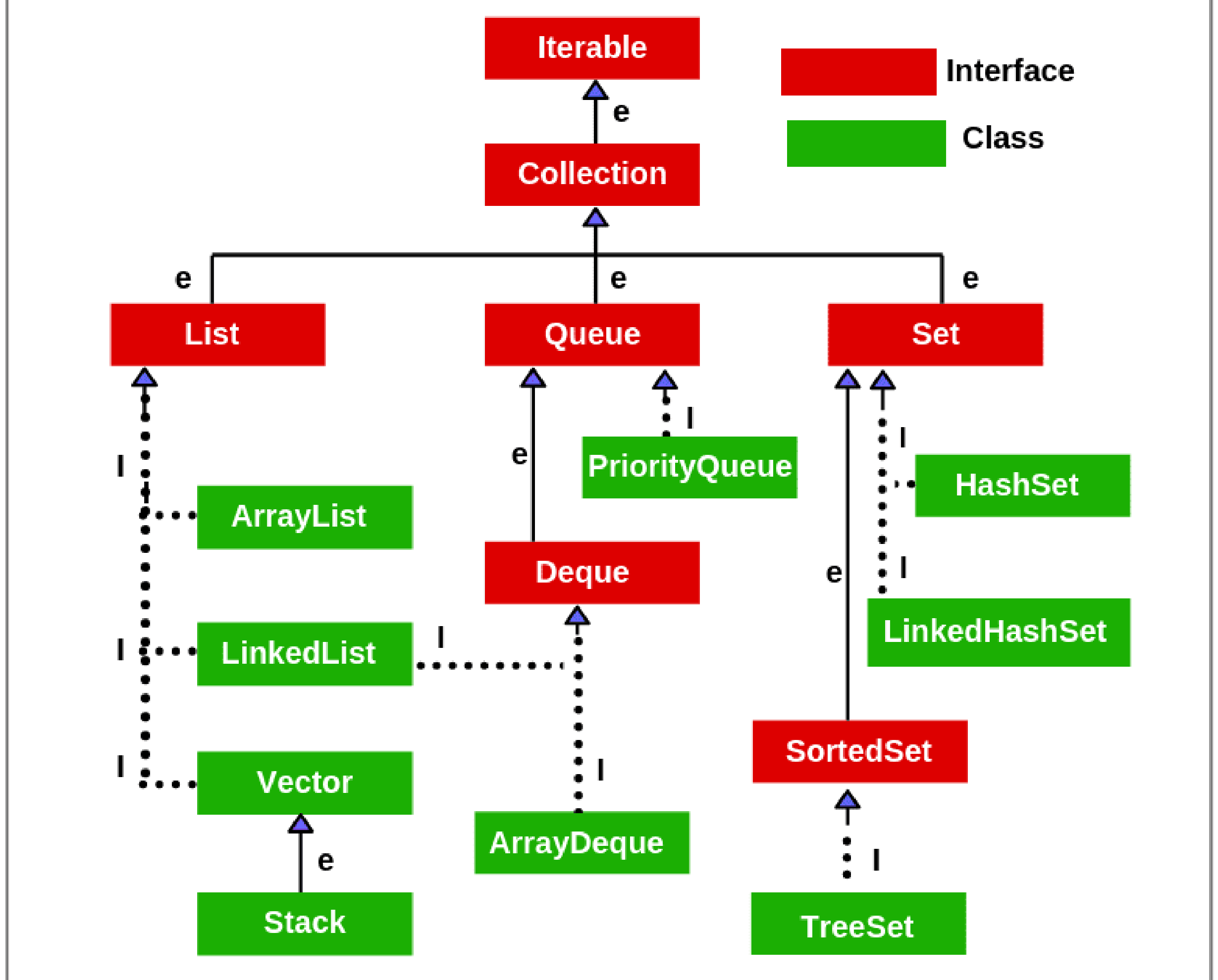

Fig: Collection Hierarchy in Java

Set Interface

- A collection that contains no duplicate elements. More formally, sets contain no pair of elements e1 and e2 such that e1.equals(e2), and at most one null element.
- As implied by its name, this interface models the mathematical *set* abstraction.

Set Interface

Unmodifiable Sets: The `Set.of` and `Set.copyOf` static methods provide a convenient way to create unmodifiable sets. The Set instances created by these methods have the following characteristics:

1. They are unmodifiable. Elements cannot be added or removed. Calling any mutator method on the Set will always cause `UnsupportedOperationException` to be thrown.
2. They disallow null elements. Attempts to create them with null elements result in `NullPointerException`.
3. They reject duplicate elements at creation time. Duplicate elements passed to a static factory method result in `IllegalArgumentException`.
4. The iteration order of set elements is unspecified and is subject to change.

Set Interface Methods

Modifier and Type	Method	Description
`boolean`	`add(E e)`	Adds the specified element to this set if it is not already present (optional operation).
`boolean`	`addAll(Collection c)`	Adds all of the elements in the specified collection to this set if they're not already present (optional operation).
`void`	`clear()`	Removes all of the elements from this set (optional operation).
`boolean`	`contains(Object o)`	Returns true if this set contains the specified element.
`boolean`	`containsAll(Collection c)`	Returns true if this set contains all of the elements of the specified collection.
`static`	`copyOf(Collection c)`	Returns an unmodifiable Set containing the elements of the given Collection.
`boolean`	`equals(Object o)`	Compares the specified object with this set for equality.
`int`	`hashCode()`	Returns the hash code value for this set.
`boolean`	`isEmpty()`	Returns true if this set contains no elements.

Set Interface Methods (cont..)

Modifier and Type	Method	Description
Iterator<E>	iterator()	Returns an iterator over the elements in this set.
static <E> Set<E>	of()	Returns an unmodifiable set containing zero elements.
static <E> Set<E>	of(E e1)	Returns an unmodifiable set containing one element.
static <E> Set<E>	of(E... elements)	Returns an unmodifiable set containing an arbitrary number of elements.
static <E> Set<E>	of(E e1, E e2)	Returns an unmodifiable set containing two elements.
static <E> Set<E>	of(E e1, E e2, E e3)	Returns an unmodifiable set containing three elements.
static <E> Set<E>	of(E e1, E e2, E e3, E e4)	Returns an unmodifiable set containing four elements.
static <E> Set<E>	of(E e1, E e2, E e3, E e4, E e5)	Returns an unmodifiable set containing five elements.
static <E> Set<E>	of(E e1, E e2, E e3, E e4, E e5, E e6)	Returns an unmodifiable set containing six elements.
static <E> Set<E>	of(E e1, E e2, E e3, E e4, E e5, E e6, E e7)	Returns an unmodifiable set containing seven elements.
static <E> Set<E>	of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8)	Returns an unmodifiable set containing eight elements.
static <E> Set<E>	of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8, E e9)	Returns an unmodifiable set containing nine elements.
static <E> Set<E>	of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8, E e9, E e10)	Returns an unmodifiable set containing ten elements.

Set Interface Methods (cont..)

Modifier and Type	Method	Description
`boolean`	`remove(Object o)`	Removes the specified element from this set if it is present (optional operation).
`boolean`	`removeAll(Collection c)`	Removes from this set all of its elements that are contained in the specified collection (optional operation).
`boolean`	`retainAll(Collection c)`	Retains only the elements in this set that are contained in the specified collection (optional operation).
`int`	`size()`	Returns the number of elements in this set (its cardinality).
`Object[]`	`toArray()`	Returns an array containing all of the elements in this set.

Java Set vs. List

- Same element cannot occur more than once in a Java Set. This is different from a Java List where each element can occur more than once.
- Elements in a Set has no guaranteed internal order. The elements in a List has an internal order, and the elements can be iterated in that order.

Java Set Example:

```
import java.util.Set;
import java.util.HashSet;
public class SetExample {
    public static void main(String[] args) {
        Set setA = new HashSet();
        setA.add("A");
        System.out.println( setA.contains("A") );
    }
}
```

Set Implementations

- Being a Collection subtype all methods in the Collection interface are also available in the Set interface.
- Since Set is an interface you need to instantiate a concrete implementation of the interface in order to use it. You can choose between the following Set implementations in the Java Collections API:

1. `java.util.EnumSet`
2. `java.util.HashSet`
3. `java.util.LinkedHashSet`
4. `java.util.TreeSet`

Set Implementations (contd..)

Create a Set:

```
import java.util.HashSet;
import java.util.LinkedHashSet;
import java.util.Set;
import java.util.TreeSet;

public class SetExample {

    public static void main(String[] args) {

        Set setA = new HashSet();
        Set setB = new LinkedHashSet();
        Set setC = new TreeSet();

    }
}
```

Set Implementations (contd..)

Add Element to Set: To add elements to a Set you call its add() method. This method is inherited from the Collection interface. Here are a few examples:

```
Set setA = new HashSet();
setA.add("element");
```

Iterate Set Elements: There are two ways to iterate the elements of a Java Set:

1. Using an Iterator obtained from the Set.
2. Using the for-each loop.

```
//Using Iterator
Set setA = new HashSet();
setA.add("element 1");
setA.add("element 2");
setA.add("element 3");
Iterator iterator = SetA.iterator();
while(iterator.hasNext(){
  String element = iterator.next();
}
```

Set Implementations (contd..)

Iterate Set Using For-Each Loop:

```
Set set = new HashSet();
for(Object object : set) {
    String element = (String) object; }
```

Remove Elements From Set: You remove elements from a Java Set by calling the remove(Object o) method.

```
set.remove("object-to-remove");
```

Remove All Elements From Set

```
set.clear();
```

Add All Elements From Another Collection

```
Set<String> set = new HashSet<>();
set.add("one"); set.add("two"); set.add("three");
Set<String> set2 = new HashSet<>();
set2.add("four");
set2.addAll(set);
```

Remove All Elements From Another Collection

```
Set<String> set = new HashSet<>();
set.add("one"); set.add("two"); set.add("three");
Set set2 = new HashSet();
set2.add("three");  set.removeAll(set2);
```

Set Implementations (contd..)

Retain All Elements Present in Another Collection:

```
Set set = new HashSet ();
set.add("one");  set.add("two");  set.add("three");
Set set2 = new HashSet();
set2.add("three");   set2.add("four");
set.retainAll(set2);
```

Set Size:

```
Set set = new HashSet();
set.add("123"); set.add("456"); set.add("789");
int size = set.size();
```

Check if Set is Empty

```
Set<String> set = new HashSet<>();
boolean isEmpty = set.isEmpty();
```

- You can also check if a Set is empty by comparing the value returned by the size() method with 0:

```
Set set = new HashSet();
boolean isEmpty = (set.size() == 0);
```

Set Implementations (contd..)

Check if Set Contains Element

```
Set set = new HashSet ();
set.add("123"); set.add("456");
boolean contains123 = set.contains("123");
```

- Since it is possible to add null values to a Set, it is also possible to check if the Set contains a null value. Here is how you check if a Set contains a null value:

```
set.add(null);
containsElement = set.contains(null);
System.out.println(containsElement);
```

Convert Java Set to List: You can convert a Java Set to a Java List by creating a List and calling its addAll() method, passing the Set as parameter to the addAll() method.

```
Set set = new HashSet();

set.add("123"); set.add("456");

List list = new ArrayList ();

list.addAll(set);
```

SortedSet

- The Java SortedSet interface, java.util.SortedSet, is a subtype of the java.util.Set interface.
- The Java SortedSet interface behaves like a normal Set with the exception that the elements it contains are sorted internally.
- This means that when you iterate the elements of a SortedSet the elements are iterated in the sorted order.

The TreeSet SortedSet Implementation

The Java Collections API has one implementation of the Java SortedSet interface - the `java.util.TreeSet class.`

- All elements of a SortedSet must implement the Comparable interface (or be accepted by the specified Comparator) and all such elements must be mutually comparable (i.e. Mutually Comparable simply means that two objects accept each other as the argument to their compareTo method).

SortedSet (contd..)

Methods of SortedSet interface:

- **comparator() :** Returns the comparator used to order the elements in this set, or null if this set uses the natural ordering of its elements.
- **first() :** Returns the first (lowest) element currently in this set.
- **headSet(E toElement) :** Returns a view of the portion of this set whose elements are strictly less than toElement.
- **last() :** Returns the last (highest) element currently in this set.
- **subSet(E fromElement, E toElement) :** Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive.
- **tailSet(E fromElement) :** Returns a view of the portion of this set whose elements are greater than or equal to fromElement.

SortedSet (contd..)

```
// A Java program to demonstrate working of SortedSet
import java.util.SortedSet; import java.util.TreeSet;
public class SortedSetDemo {
    public static void main(String[] args)     {
        // Create a TreeSet and inserting elements
        SortedSet sites = new TreeSet();
        sites.add(“practice”);
        sites.add(“UPES”);
        sites.add(“quiz”);
        sites.add(“code”);
        System.out.println("Sorted Set: " + sites);
        System.out.println("First: " + sites.first());
        System.out.println("Last: " + sites.last());
        // Getting elements before quiz (Excluding) in a sortedSet
        SortedSet beforeQuiz = sites.headSet("quiz");
        System.out.println(beforeQuiz);
         // Getting elements between code (Including) and practice (Excluding)
        SortedSet betweenCodeAndQuiz = sites.subSet("code","practice");
        System.out.println(betweenCodeAndQuiz);
        // Getting elements after code (Including)
        SortedSet<String> afterCode = sites.tailSet("code");
        System.out.println(afterCode);
    }   }
```

Java Iterator interface

Java Iterator interface is used to iterate over the elements in a collection (list, set or map). It helps to retrieve the specified collection elements one by one and perform operations over each element.

Java Iterator interface: All Java collection classes provide iterator() method which return the instance of Iterator to walk over the elements in that collection. For example, ArrayList class iterator() method return an iterator over the elements in this list in proper sequence.

Iterator Example:

```
ArrayList<String> list = new ArrayList<>();
list.add("A");
list.add("B");
list.add("C");
list.add("D");

Iterator<String> iterator = list.iterator();
while(iterator.hasNext()) {
    System.out.println( iterator.next() );
}
```

Java Iterator Methods

1. **Iterator hasNext()**

- This method returns true if the iteration has more elements remaining in the collection.
- If iterator has gone over all elements then this method will return false.

2. **Iterator next()**

- This method returns the next element in the iteration.
- It throws `NoSuchElementException` if the iteration has no more elements.

3. **Iterator remove()**

- It removes from the underlying collection the last element returned by the iterator.
- This method can be called only once per call to next().
- If the underlying collection is modified while the iteration is in progress in any way other than by calling remove() method, iterator will throw a `ConcurrentModificationException`.
- Iterators that do this are known as fail-fast iterators, as they fail quickly and cleanly, rather that risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Java Iterator Methods (contd..)

4. Iterator forEachRemaining()

- This method performs the given action for each remaining element until all elements have been processed or the action throws an exception.
- Actions are performed in the order of iteration, if that order is specified.
- It throws `NullPointerException` if the specified action is null.

Java Iterator Methods (contd..)

Example to iterate over ArrayList elements:

```
ArrayList list = new ArrayList();
list.add("A");
list.add("B");
list.add("C");
list.add("D");
System.out.println(list);
//Get iterator
Iterator<String> iterator = list.iterator();
//Iterate over all elements
while(iterator.hasNext())
{
    //Get current element
    String value = iterator.next();
    System.out.println( value );
    //Remove element
    if(value.equals("B")) {
        iterator.remove();
    }
}
System.out.println(list);
```

```
Output:
[A, B, C, D]
A
B
C
D
[A, C, D]
```

Map Interface

Interface Map<K,V>

Type Parameters:

K - the type of keys maintained by this map

V - the type of mapped values

- The `java.util.Map` interface represents a mapping between a key and a value.
- The Map interface is not a subtype of the Collection interface. Therefore it behaves a bit different from the rest of the collection types.

Few characteristics of the Map Interface are:

1. A Map cannot contain duplicate keys and each key can map to at most one value. Some implementations allow null key and null value like the HashMap and `LinkedHashMap`, but some do not like the `TreeMap`.
2. The order of a map depends on specific implementations, e.g `TreeMap and LinkedHashMap` have predictable order, while HashMap does not.
3. There are two interfaces for implementing Map in java: Map and SortedMap, and three classes: `HashMap`, `TreeMap` and `LinkedHashMap`.

References

http://java.sun.com/docs/books/tutorial/collections/interfaces/collection.html.

THANK YOU

UPES

UNIVERSITY WITH A PURPOSE