Advanced Programming

Week 2

Java Collections

- Set Interface
- Map Interface
- Collection Algorithms
- Benefits of Java Collections Framework

Tilahun Melak(PhD)

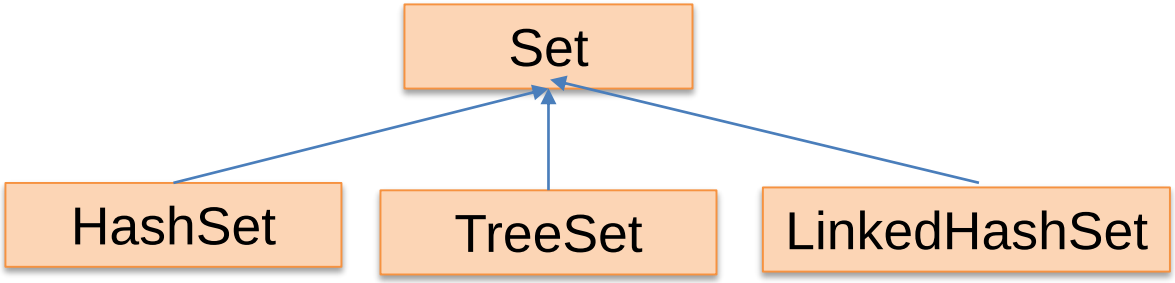
September, 2025

Objectives

At the end of this lecture students will be able to :

- Implement the Set Interface
- Implement the Map Interface
- Explain the main Collection Algorithms
- Discuss the Benefits of Java Collection Framework

Set Interface



- The Set interface is a collection that does not allow duplicate elements.
- Its common implementations include HashSet ,TreeSet and LinkedHashSet .
- Same methods as Collection
  - different contract - no duplicate entries

Deitel, H. M., & Deitel, P. J. (2006). Java™ How to Program (7th ed.). Prentice Hall.

Set Interface Cntd…

- In the Set interface the following fundamental methods are defined:
  - **boolean add(Object o)** - reject duplicates
  - **Iterator iterator()**
- Provides an Iterator to navigate through the elements in the Set
- No guaranteed order in the basic Set interface
- There is a SortedSet interface that extends Set

Deitel, H. M., & Deitel, P. J. (2006). Java™ How to Program (7th ed.). Prentice Hall.

HashSet

- Find and add elements very quickly
  - uses hashing implementation in HashMap
- Hashing uses an array of linked lists
  - The **hashCode()** is used to index into the array
  - Then **equals()** is used to determine if element is in the (short) list of elements at that index

Deitel, H. M., & Deitel, P. J. (2006). Java™ How to Program (7th ed.). Prentice Hall.

HashSet Cntd…

- No order imposed on elements
- HashSet stores its elements in a hash table
- The **hashCode()** method and the **equals()** method must be compatible
  - if two objects are equal, they must have the same **hashCode()** value

Deitel, H. M., & Deitel, P. J. (2006). Java™ How to Program (7th ed.). Prentice Hall.

TreeSet

- Elements can be added in any sequence.
- TreeSet automatically arranges them in sorted order.
- Iterating over the TreeSet always follows this sorted order.

Deitel, H. M., & Deitel, P. J. (2006). Java™ How to Program (7th ed.). Prentice Hall.

TreeSet Cntd…

- Default order is defined by natural order
  - objects implement the Comparable interface
  - TreeSet uses **compareTo(Object o)** to sort elements
- Can use a different Comparator
  - provide Comparator to the TreeSet constructor

Deitel, H. M., & Deitel, P. J. (2006). Java™ How to Program (7th ed.). Prentice Hall.

LinkedHashSet

- **Extends:** HashSet (no new methods)
- **Order:** Maintains insertion order via a linked list
- **Storage:** Uses hash codes internally for fast access

Deitel, H. M., & Deitel, P. J. (2006). Java™ How to Program (7th ed.). Prentice Hall.

Linked HashSet Cntd…

- The LinkedHashSet class supports four constructors.
- **Default constructor**
  - **LinkedHashSet()** → Creates an empty LinkedHashSet with default settings.
- **From Collection**
  - **LinkedHashSet(Collection c)** → Builds a set containing all elements of the specified collection.
- **With Capacity**
  - **LinkedHashSet(int capacity)** → Creates a set with the given initial capacity (auto-expands as needed).
- **With Capacity & Load Factor**
  - **LinkedHashSet(int capacity, float fillRatio)** → Creates a set with specified capacity and load factor (fill ratio).

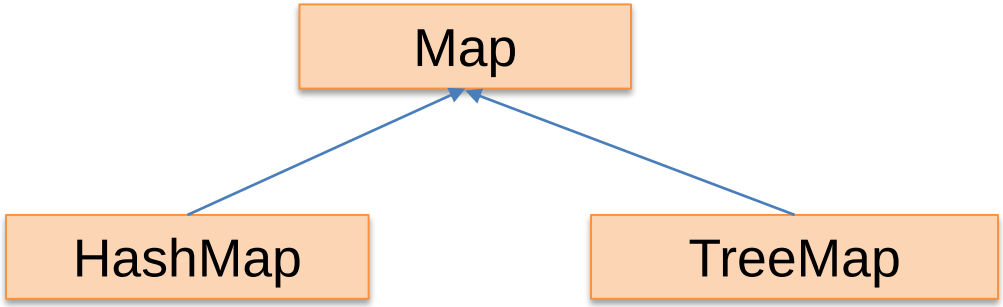
Comparison Among Set Implementations

	HashSet	**TreeSet**	**Linked HashSet**
Storage Type	Hash Table	Red-Black Tree	Hash Table with a Linked List
Performance	Best performance	Slower than HashSet	Little costly than HashSet
Order of Iteration	No guarantee of order of iteration	Order based	Orders elements based on insertion

Example: HashSet and Iterator

```
import java.util.*;
public class HS {
  public static void main(String args[])
  {HashSet <String> hs = new HashSet<String>();
   hs.add("Ethiopia");
   hs.add ("America");
   hs.add ("Japan");
   hs.add ("China");
   hs.add ("America");
   hs.add("S. Korea");
System.out.println ("HashSet = " + hs);
Iterator it = hs.iterator ();
System.out.println ("Elements Using Iterator: ");
   while (it.hasNext() )
   { String s = (String) it.next ();
     System.out.println(s);  }}}
```

Map Interface



- Stores key/value pairs
- Maps from the key to the value
- Keys are unique
  - a single key only appears once in the Map
  - a key can map to only one value
- Values do not have to be unique

Deitel, H. M., & Deitel, P. J. (2006). Java™ How to Program (7th ed.). Prentice Hall.

Map Interface Cntd…

- HashMap
  - The keys are a set - unique, unordered
  - Fast
- TreeMap
  - The keys are a set - unique, ordered
  - Same options for ordering as a TreeSet

    Natural order (Comparable, compareTo(Object))

    Special order (Comparator, compare(Object, Object))

Map Methods

- Map provides several useful methods:
  - Object put(Object key, Object value)
  - Object get(Object key)
  - Object remove(Object key)
  - boolean containsKey(Object key)
  - boolean containsValue(Object value)
  - int size()
  - boolean isEmpty()

Map Views

- **Iterating over a Map**
- **Set keySet()** → Returns a *Set* of all keys in the Map
- **Collection values()** → Returns a *Collection* of values (not a Set, since different keys may map to the same value)
- **Set entrySet()** → Returns a *Set* of key–value pairs
  - Uses the nested **Map.Entry** interface for each element

Oracle. (2015). Java Platform, Standard Edition 8, API Specification: java.util.Map.

Example: HashSet and Iterator

```
import java.util.*;
class HashMapDemo
{ public static void main(String args[])
  {
    HashMap<Integer, String> hm = new HashMap<Integer, String> ();
   hm.put (101,"Alemu");
   hm.put (102,"Kuma");
   hm.put (103,"Hanna");
   hm.put (104,"Mahesh");
   hm.put (105,"Kidst");
   Set<Integer> set = new HashSet<Integer>();
   set = hm.keySet();
   System.out.println (hm.get(101));}}
```

Bulk Operations

- In addition to the basic operations, a Collection provide “bulk” operations;
  - **boolean containsAll(Collection c);**
  - **boolean addAll(Collection c); // Optional**
  - **boolean removeAll(Collection c); // Optional**
  - **boolean retainAll(Collection c); // Optional**
  - **void clear(); // Optional**
  - **Object[] toArray();**
  - **Object[] toArray(Object a[]);**

Collection Algorithms

- The Java Collections Framework provides several algorithms applicable to collections and maps.
- These algorithms are implemented as static methods in the Collections class.
- Key method:
  - **static int binarySearch(List list, Object value, Comparator c)**
    - Searches for value in a list ordered by comparator c.
    - Returns the index of the value if found, otherwise -1.
  - **static int binarySearch(List list, Object value)**
    - Searches for value in list.
    - The list must be sorted.
    - Returns the position of value in list, or -1 if value is not found.

Deitel, H. M., & Deitel, P. J. (2006). Java™ How to Program (7th ed.). Prentice Hall.

Collection Algorithms Cntd…

- **static void copy(List list1, List list2)**
  - Copies the elements of list2 to list1.
- **static Enumeration enumeration(Collection c)**
  - Returns an enumeration over c.
- **static void fill(List list, Object obj)**
  - Assigns obj to each element of list.
- **static int indexOfSubList(List list, List subList)**
  - Searches list for the first occurrence of subList.
  - Returns the index of the first match, or .1 if no match is found.
- **static int lastIndexOfSubList(List list, List subList)**
  - Searches list for the last occurrence of subList.
  - Returns the index of the last match, or .1 if no match is found.

Deitel, H. M., & Deitel, P. J. (2006). Java™ How to Program (7th ed.). Prentice Hall.

Collection Algorithms Cntd…

- **static ArrayList list(Enumeration enum)**
  - Returns an ArrayList that contains the elements of enum.
- **static Object max(Collection c, Comparator comp)**
  - Returns the maximum element in c as determined by comp.
- **static Object max(Collection c)**
  - Returns the maximum element in c as determined by natural ordering.
  - The collection need not be sorted.
- **static Object min(Collection c, Comparator comp)**
  - Returns the minimum element in c as determined by comp.
  - The collection need not be sorted.
- **static Object min(Collection c)**
  - Returns the minimum element in c as determined by natural ordering.

Deitel, H. M., & Deitel, P. J. (2006). Java™ How to Program (7th ed.). Prentice Hall.

Collection Algorithms Cntd…

- **static List nCopies(int num, Object obj)**
  - Returns num copies of obj contained in an immutable list.
  - num must be greater than or equal to zero.
- **static boolean replaceAll(List list, Object old, Object new)**
  - Replaces all occurrences of old with new in list.
  - Returns true if at least one replacement occurred. Returns false, otherwise.
- **static void reverse(List list)**
  - Reverses the sequence in list.
- **static Comparator reverseOrder( )**
  - Returns a reverse comparator.

Deitel, H. M., & Deitel, P. J. (2006). Java™ How to Program (7th ed.). Prentice Hall.

Collection Algorithms Cntd…

- **static void rotate(List list, int n)**
  - Rotates list by n places to the right.
  - To rotate left, use a negative value for n.
- **static void shuffle(List list, Random r)**
  - Shuffles (i.e., randomizes) the elements in list by using r as a source of random numbers.
- **static void shuffle(List list)**
  - Shuffles (i.e., randomizes) the elements in list.
- **static Set singleton(Object obj)**
  - Returns obj as an immutable set. This is an easy way to convert a single object into a set.
- **static List singletonList(Object obj)**
  - Returns obj as an immutable list. This is an easy way to convert a single object into a list.

Deitel, H. M., & Deitel, P. J. (2006). Java™ How to Program (7th ed.). Prentice Hall.

Collection Algorithms Cntd…

- **static Map singletonMap(Object k, Object v)**
  - Returns the key/value pair k/v as an immutable map.
  - This is an easy way to convert a single key/value pair into a map.
- **static void sort(List list, Comparator comp)**
  - Sorts the elements of list as determined by comp.
- **static void sort(List list)**
  - Sorts the elements of list as determined by their natural ordering.
- **static void swap(List list, int idx1, int idx2)**
  - Exchanges the elements in list at the indices specified by idx1 and idx2.
- **static Collection synchronizedCollection(Collection c)**
  - Returns a thread-safe collection backed by c.

Deitel, H. M., & Deitel, P. J. (2006). Java™ How to Program (7th ed.). Prentice Hall.

Collection Algorithms Cntd…

- **static List synchronizedList(List list)**
  - Returns a thread-safe list backed by list.
- **static Map synchronizedMap(Map m)**
  - Returns a thread-safe map backed by m.
- **static Set synchronizedSet(Set s)**
  - Returns a thread-safe set backed by s.
- **static SortedMap synchronizedSortedMap(SortedMap sm)**
  - Returns a thread-safe sorted set backed by sm.
- **static SortedSet synchronizedSortedSet(SortedSet ss)**
  - Returns a thread-safe set backed by ss.

Deitel, H. M., & Deitel, P. J. (2006). Java™ How to Program (7th ed.). Prentice Hall.

Collection Algorithms Cntd…

- **static Collection unmodifiableCollection(Collection c)**
  - Returns an unmodifiable collection backed by c.
- **static List unmodifiableList(List list)**
  - Returns an unmodifiable list backed by list.
- **static Map unmodifiableMap(Map m)**
  - Returns an unmodifiable map backed by m.
- **static Set unmodifiableSet(Set s)**
  - Returns an unmodifiable set backed by s.
- **static SortedMap unmodifiableSortedMap(SortedMap sm)**
  - Returns an unmodifiable sorted map backed by sm.
- **static SortedSet unmodifiableSortedSet(SortedSet ss)**
  - Returns an unmodifiable sorted set backed by ss.

Deitel, H. M., & Deitel, P. J. (2006). Java™ How to Program (7th ed.). Prentice Hall.

Example: Algorithms

```
import java.util.*;
public class AlgorithmsDemo
{  public static void main(String args[])
   {LinkedList ll = new LinkedList();
      ll.add(new Integer(-8));
      ll.add(new Integer(20));
      ll.add(new Integer(-20));
      ll.add(new Integer(8));
     Comparator r =
    Collections.reverseOrder();
Collections.sort(ll, r);
      // Get iterator
      Iterator li = ll.iterator();
      System.out.print("List sorted in reverse:
    ");
       while(li.hasNext()){
          System.out.print(li.next() + " ");}
      System.out.println();
      Collections.shuffle(ll);
    // display randomized list
    li = ll.iterator();
    System.out.print("List shuffled: ");
    while(li.hasNext()){
      System.out.print(li.next() + " ");
    }
    System.out.println();
    System.out.println("Minimum: " +
Collections.min(ll));
    System.out.println("Maximum: " +
Collections.max(ll));}}
```

Benefits of Java Collections Framework

- **Benefits of the Java Collections Framework;**
  - **Reduce programming Effort:**
    - Provides ready-to-use data structures and algorithms, allowing you to focus on the core logic of your program instead of building low-level data handling from scratch.
  - **Improves performance and quality:**
    - Offers efficient, well-tested implementations that can be easily swapped to fine-tune your program's speed and behavior.
    - You can spend more time enhancing your application rather than reinventing common structures.
  - **Supports API interoperability:**
    - Standardized collection interfaces make it easy for different libraries and APIs to work together, even if they were developed independently, ensuring smooth data exchange across components.

Oracle. (n.d.). *Introduction to collections*. The Java™ Tutorials.

Benefits of Java Collections Framework Cntd…

- **Reduces effort to learn and to use new APIs:**
  - Many APIs naturally take collections on input and furnish them as output.
  - In the past, each such API had a small sub-API devoted to manipulating its collections. There was little consistency among these ad hoc collections sub-APIs, so you had to learn each one from scratch, and it was easy to make mistakes when using them. With the advent of standard collection interfaces, the problem went away.
- **Reduces effort to design new APIs:**
  - This is the flip side of the previous advantage.
  - Designers and implementers don't have to reinvent the wheel each time they create an API that relies on collections; instead, they can use standard collection interfaces.
- **Fosters software reuse:**
  - New data structures that conform to the standard collection interfaces are by nature reusable. The same goes for new algorithms that operate on objects that implement these interfaces.

Oracle. (n.d.). *Introduction to collections*. The Java™ Tutorials.

Exercise

1) Discuss the difference between List, Set and Queue?
2) Write a program to create three list objects l1,l2, and l3. and copy the content of l1 and l2 in to l3
3) Create a Map object of minimum 20 employees as m1<empname,salary> and
    1) find total, minimum, and maximum salary.
    2) display name of employee with minimum, maximum and Average salary.
    3) Sort and display employee detail by name
4) Explain why and when we use List,Set and Map with concrete example

Summary

- In today's lecture we have discussed about;
  - Set Interface
  - Map Interface
  - Collection Algorithms
  - Benefits of Java Collections Framework
  - Examples
  - Exercises

References

- Oracle. (n.d.). Collections overview. In Java Platform, Standard Edition 8 Documentation. Oracle.
- Deitel, H. M., & Deitel, P. J. (2006). *Java™ How to Program* (7th ed.). Prentice Hall.
- Oracle. (2015). Java Platform, Standard Edition 8, API Specification: java.util.Map.
- Oracle. (n.d.). *The Java™ tutorials: Collections framework*. Oracle.