CSE-2102
Object Oriented Programming

Dr. Muhammad Ibrahim
Assistant Professor
Dept. of Computer Science and Engineering
University of Dhaka

Topics

- Collections framework
  - Structure
    - Interfaces
    - Classes
  - Usage of classes
    - ArrayList
    - LinkedList
    - HashSet
    - TreeSet
    - HashMap

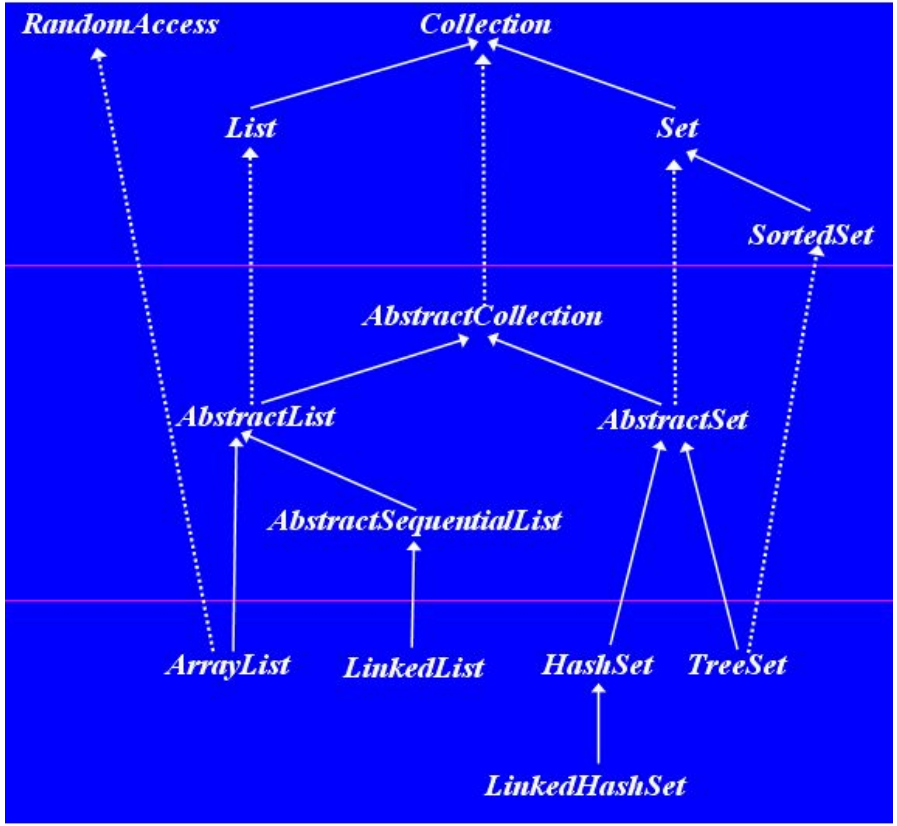
Collections Framework

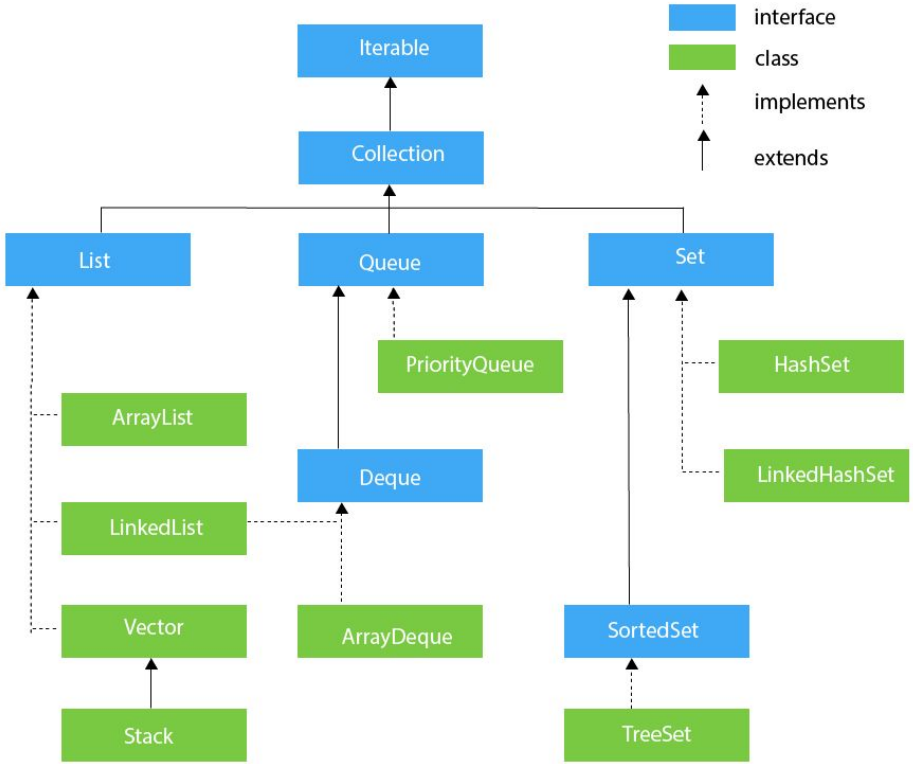
- Collections is a group of classes and interfaces that facilitates manipulation of groups of objects of different types.
- These classes and interfaces are contained in `java.util` package.
- Some important and commonly used classes:
  - HashMap
  - Random
  - ArrayList
  - Arrays
  - Scanner
  - HashSet
  - Collections

Collections Framework

- Interface `java.lang.Iterable`
  - Interface `Collection`
    - Interface `List`
      - Class `ArrayList`
      - Class `LinkedList` (also implements `Deque`)
      - Class `Vector`
        - Class `Stack` (legacy class, use `Deque`, which is more powerful)
    - Interface `Set`
      - Class `HashSet`
        - Class `LinkedHashSet`
      - Interface `SortedSet`
        - Interface `NavigableSet`
          - Class `TreeSet`
      - Class `EnumSet`
    - Interface `Queue`
      - Class `PriorityQueue`
      - Interface `Deque`
        - Class `LinkedList` (also implements `List`)
        - Class `ArrayDeque`
- Interface `Map`
  - Class `HashMap`
  - Interface `SortedMap`
    - Interface `NavigableMap`
      - Class `TreeMap`

https://en.wikiversity.org/wiki/Java_Collections_Overview





https://www.cs.cmu.edu/~pattis/15-1XX/15-200/lectures/collectionsiii/lecture.html

https://www.javatpoint.com/collections-in-java

Collections Framework

- Commonly used interfaces
  - Iterator
  - Comparator
  - Map
  - Set

A Motivating Example

Before delving into descriptive details of different members of collections framework, let us work out an example of ArrayList…

Collections Framework

- The collections are highly efficient.
  - If we deal with collections of objects manually, performance would be compromised.
  - Collections are carefully designed and coded to yield better performance.
- They allow different data types to follow the same algorithms.
- Extending the collections are easy.

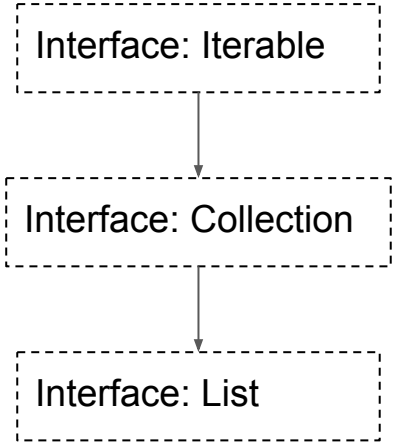
Structure of Collections: **Collection** Interface

- **Collection** is an interface which is at the top of the class hierarchy
- Declaration: `interface Collection<E>`
  - Here, E specifies the type of objects that the collection will hold.
- It extends **Iterable** interface
- Each class of collections framework must implement **Collection** interface
- The methods are: add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, parallelStream, remove, removeAll, removeIf, retainAll, size, splitIterator, stream, toArray.
- It is generic because any type of data can be manipulated using this interface.

Structure of Collections: **List** Interface

- **List** is an interface that extends **Collection**
- It is used to store a sequence of elements
- Declaration: `interface List<E>`
  - Here, E specifies the type of objects that the list will hold.

Diagram So Far



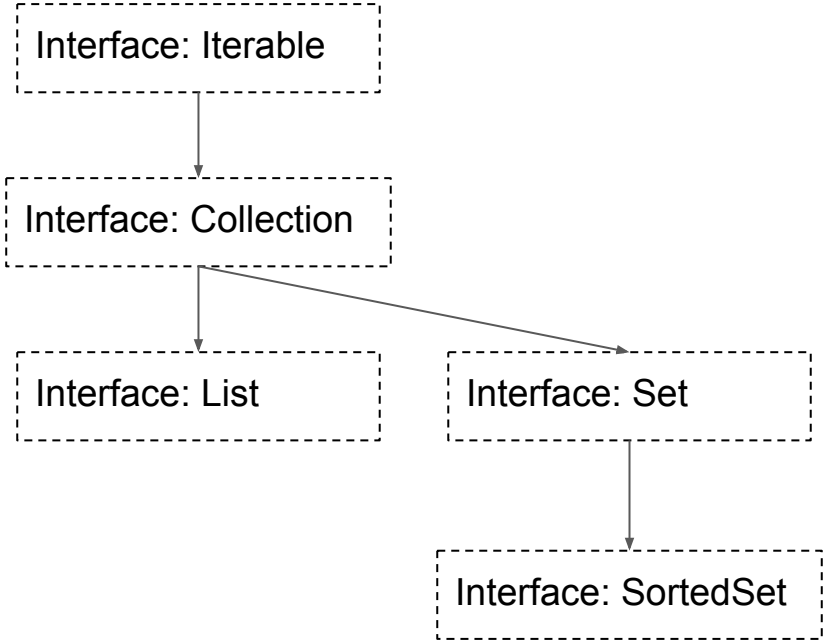
Structure of Collections: **Set** Interface

- **Set** is an interface that extends **Collection**
- It does not allow duplicate existence of elements
- Declaration: `interface Set<E>`
  - Here, E specifies the type of objects that the set will hold.

Structure of Collections: **SortedSet** Interface

- **SortedSet** is an interface that extends **Set**
- It keeps the elements sorted in ascending order
- Declaration: `interface SortedSet<E>`
  - Here, E specifies the type of objects that the set will hold.

Diagram So Far



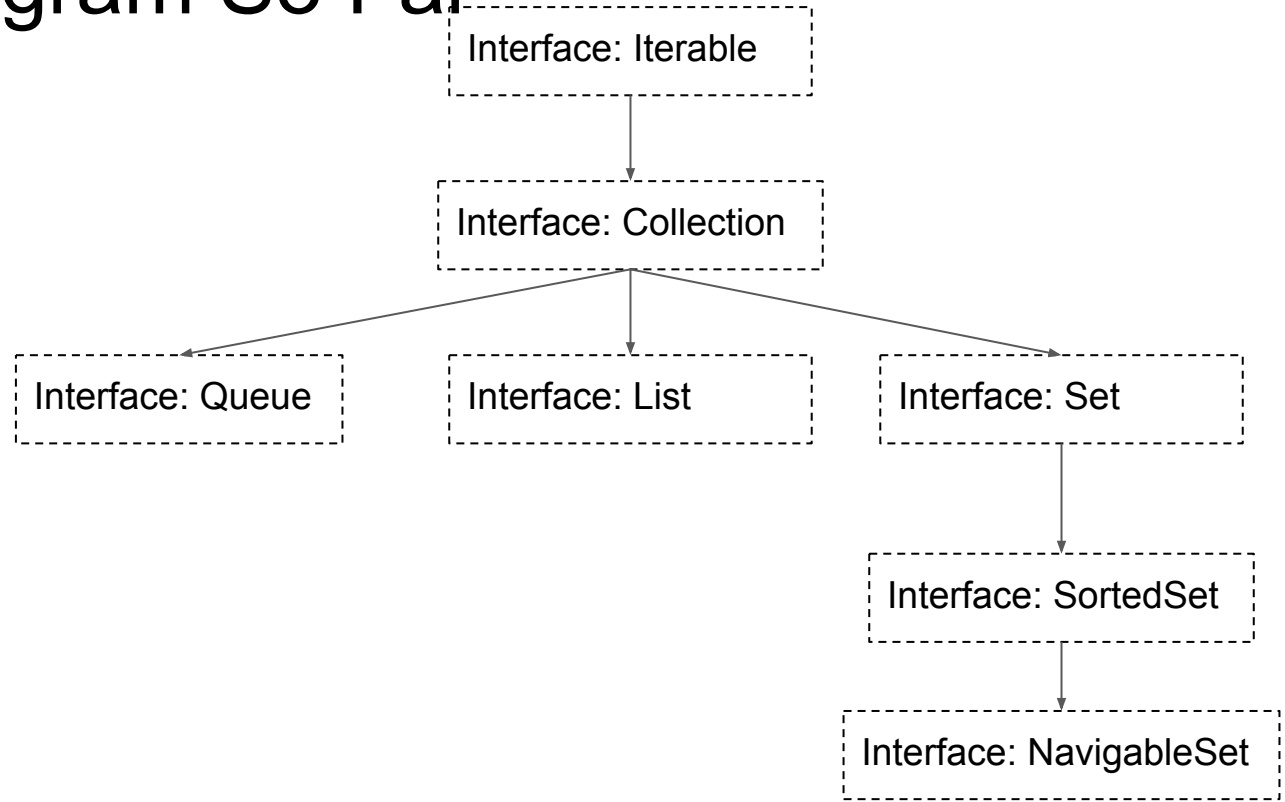
Structure of Collections: **NavigableSet** Interface

- **NavigableSet** is an interface that extends **SortedSet**
- It allows retrieval of elements based on closed matching of values
- Declaration: `interface NavigableSet<E>`
  - Here, E specifies the type of objects that the set will hold.

Structure of Collections: **Queue** Interface

- **Queue** is an interface that extends **Collection**
- It implements the first-in-first-out arrangement of elements
- `interface Queue<E>`
  - Here, E specifies the type of objects that the queue will hold.

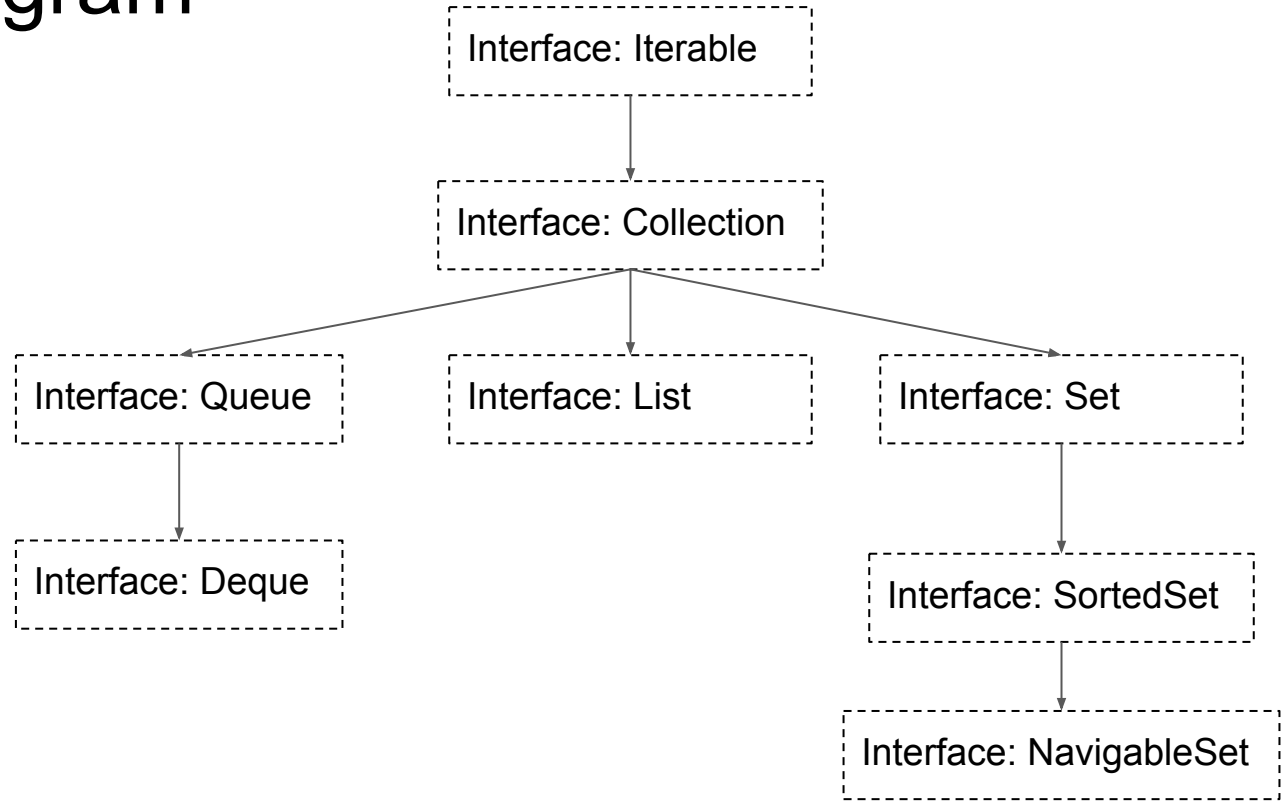
Diagram So Far



Structure of Collections: **Deque** Interface

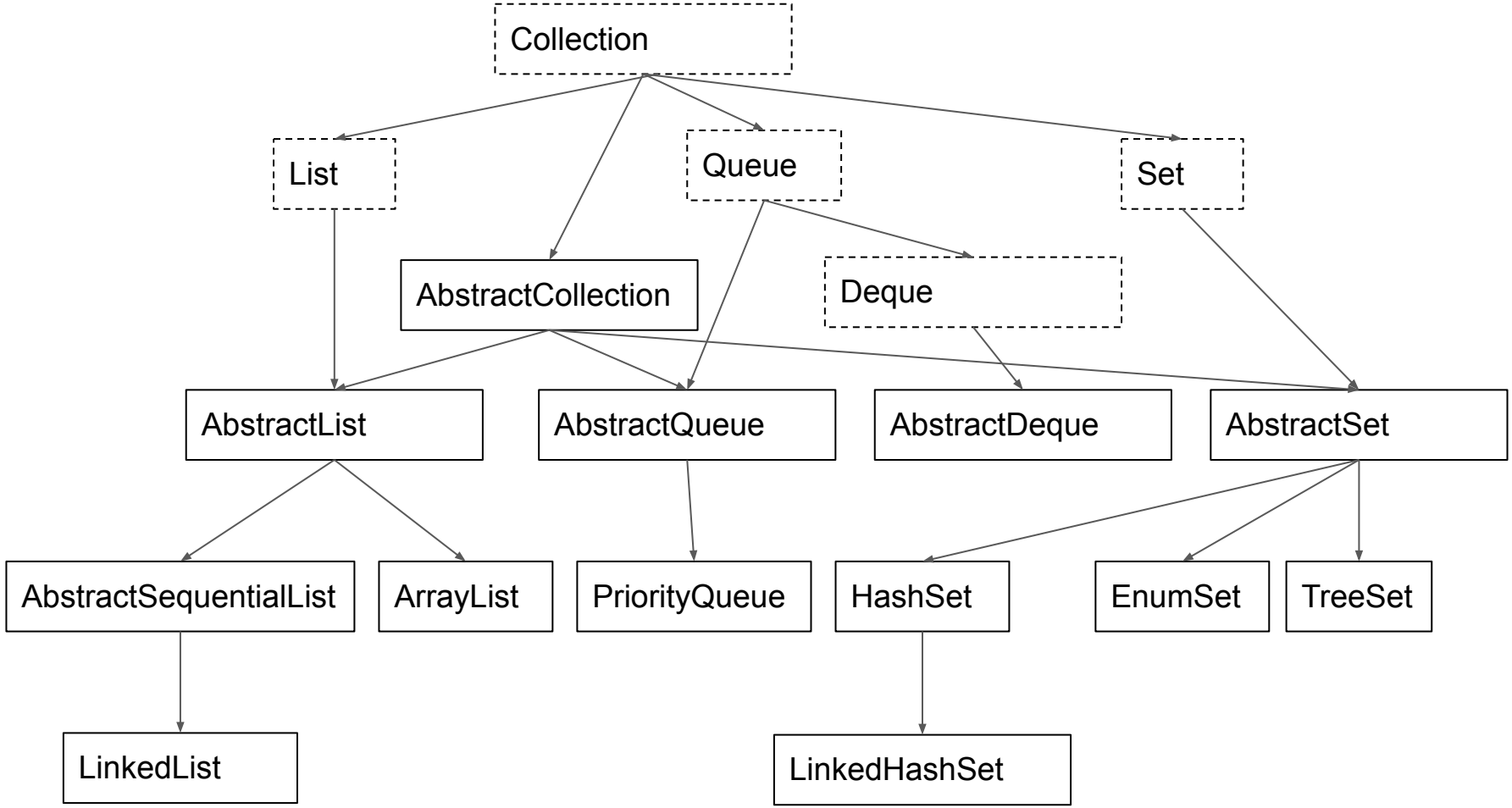
- **Deque** is an interface that extends **Queue**
- It implements a double-ended queue
- `interface Deque<E>`
  - Here, E specifies the type of objects that the queue will hold.

Diagram


Interface: Iterable
Interface: Collection
Interface: Queue
Interface: List
Interface: Set
Interface: Deque
Interface: SortedSet
Interface: NavigableSet

Structure of Collections: Classes

- Now that we've become familiarized with several interfaces of collections framework, it is time to move on to classes
- Some of the classes are fully implemented, while others are abstract


Collection
List
Queue
Set
AbstractCollection
Deque
AbstractList
AbstractQueue
AbstractDeque
AbstractSet
AbstractSequentialList
ArrayList
PriorityQueue
HashSet
EnumSet
TreeSet
LinkedList
LinkedHashSet

Collections Framework in Practice

Now let’s work out examples of each of the following:

- ArrayList
- LinkedList
- HashSet
- TreeSet
- HashMap
- TreeMap

ArrayList

- is a variable-length array of object references.
- Declaration: `class ArrayList<E>`
- Constructors
  - `ArrayList( )`
  - `ArrayList(Collection<? extends E> c)`
    - Note: <? extends E> will be explained in our class on Java Generics.
  - `ArrayList(int capacity)`
- Let us examine an example of ArrayList that demonstrates some of its commonly used methods that include size, add, remove, get, contains, toString, clear, equals, indexOf, set, toArray.

LinkedList

- Implements a linked list
- Declaration: `class LinkedList<E>`
- Constructors
  - `LinkedList( )`
  - `LinkedList(Collection<? extends E> c)`
- Key methods: add, addFirst, addLast, removeFirst, removeLast, set.

HashSet

- Hashing:
  - usually to find an element from an array we need to check the elements one by one, by incrementing the index. So it takes linear computational time
    - Thus, if the array is quite large, searching for an element takes a long time
  - In hashing method, each element is stored in an index that is computed based on the element itself using a (hash) function that is computed in constant time
    - Example: suppose we want to store strings, so a simple hash function could compute the index for a string as the summation of the ASCII values present in that string.
- Declaration: `class HashSet<E>`
- Constructors:
  - `HashSet( ), HashSet(Collection<? extends E> c), HashSet(int capacity), HashSet(int capacity, float fillRatio)`
- Key methods: add, remove, isEmpty, size, clear, Contains

TreeSet

- HashSet doesn’t necessarily put the elements in sorted order, but a TreeSet does
- Declaration: `class TreeSet<E>`
- Constructors
  - `TreeSet( )`
  - `TreeSet(Collection<? extends E> c)`
  - `TreeSet(Comparator<? super E> comp)`
  - `TreeSet(SortedSet<E> ss)`
- Key methods: subset

HashMap

- Declaration: `class HashMap<K, V>`
  - Here, K specifies the type of keys, and V specifies the type of values.
- Stores the mapping data as a hashtable
- Puts and retrieves element in constant time
- Constructors
  - `HashMap( )`
  - `HashMap(Map<? extends K, ? extends V> m)`
  - `HashMap(int capacity)`
  - `HashMap(int capacity, float fillRatio)`
- Key methods: put, get, containsKey, containsValue, entrySet

End of Lecture 15.