CSE3040 Java Language

Lecture 19: Collection Framework (4)

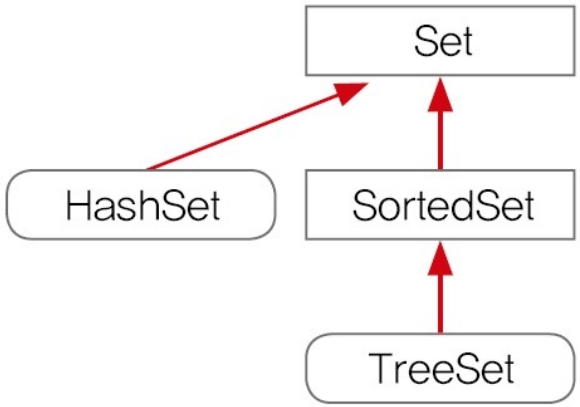
(HashSet and TreeSet)

Dept. of Computer Engineering,

Sogang University

8. HashSet

- HashSet is a class that implements interface Set.
  - No pair of elements e1 and e2 such that e1.equals(e2).
  - At most one null element
  - No ordering of elements



HashSet: Methods

- Constructors
  - load factor: if load factor is 0.75, it means that when 75% of the capacity is filled with elements, the capacity is automatically doubled.

Method	Description
HashSet()	Constructs a new, empty set; the backing HashMap instance has default initial capacity (16) and load factor (0.75).
HashSet(int initialCapacity)	Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and default load factor (0.75).
HashSet(int initialCapacity, float loadFactor)	Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and the specified load factor.
HashSet(Collection<? extends E> c)	Constructs a new set containing the elements in the specified collection.

HashSet: Methods

- Methods

Method	Description
boolean add(E e)	Adds the specified element to this set if it is not already present.
void clear()	Removes all of the elements from this set.
Object clone()	Returns a shallow copy of this HashSet instance: the elements themselves are not cloned.
boolean contains(Object o)	Returns true if this set contains the specified element.
boolean isEmpty()	Returns true if this set contains no elements.
Iterator<E> iterator()	Returns an iterator over the elements in this set.
boolean remove(Object o)	Removes the specified element from this set if it is present.
int size()	Returns the number of elements in this set (its cardinality).

- Methods declared in class AbstractCollection
  - addAll, containsAll, retainAll, toArray(), toArray(T[] a), toString

- Methods declared in class AbstractSet
  - equals, hashCode, removeAll

HashSet: Example 1

- A set does not allow duplicate elements
- A set does not maintain the order of elements

```
class Lecture {
    public static void main(String[] args) {
        String[] strArr = {"1", "1", "2", "2", "3", "3", "4", "4", "4"};
        Set<String> set = new HashSet<>();

        for(int i=0; i<strArr.length; i++) {
            set.add(strArr[i]);
        }

        System.out.println(set);
    }
}
```

HashSet: Example 2

- A program that draws lotto numbers using a HashSet.
  - Why do we use a set and then convert it to a list?

```
class Lecture {
    public static void main(String[] args) {
        Set<Integer> set = new HashSet<>();
        for( ; set.size() < 6; ) {
            int num = (int)(Math.random()*45) + 1;
            set.add(Integer.valueOf(num));
        }
        List<Integer> list = new LinkedList<>(set);
        Collections.sort(list);
        System.out.println(list);
    }
}
```

HashSet: Example 3

- A program that creates a bingo board.
  - When we use HashSet, we can observe that certain numbers often appear at certain places.
    - It is because the ordering of elements is not defined, and thus Java applies its own way of ordering elements.
    - If we use LinkedHashSet instead of HashSet, the order of elements is maintained.

```
class Lecture {
    public static void main(String[] args) {
        Set<String> set = new HashSet<>();
        int[][] board = new int[5][5];
        for( ; set.size() < 25; ) {
            set.add((int)(Math.random()*50)+1+"");
        }
        Iterator<String> it = set.iterator();
        for(int i=0; i<board.length; i++) {
            for(int j=0; j<board[i].length; j++) {
                board[i][j] = Integer.parseInt(it.next());
                System.out.print((board[i][j] < 10 ? "  " : " ") + board[i][j]);
            }
            System.out.println();
        }
    }
}
```

HashSet: Example 4

- The two Person objects have the same name and age, but they are not treated as duplicates. Why?
  - Duplicates are determined based on equals() and hashCode().
    - If two objects are duplicate, equals() should return true and hashCode() should return the same value for the two objects.

```
class Person {
    String name;
    int age;
    Person(String name, int age) { this.name = name; this.age = age; }
    public String toString() { return name + ":" + age; }
}

public class Lecture {
    public static void main(String[] args) {
        HashSet<Person> set = new HashSet<>();
        set.add(new Person("David", 10));
        set.add(new Person("David", 10));
        System.out.println(set);
    }
}
```

HashSet: Example 5

- Methods equals() and hashCode() implemented in class Person.
  - We can also use Objects.hash(name, age); to create hash code.

```
class Person {
    String name;
    int age;
    Person(String name, int age) { this.name = name; this.age = age; }
    public boolean equals(Object obj) {
        if(obj instanceof Person) {
            Person tmp = (Person)obj;
            return name.equals(tmp.name) && age==tmp.age;
        }
        return false;
    }
    public int hashCode() { return (name+age).hashCode(); }
    public String toString() { return name + ":" + age; }
}

public class Lecture {
    public static void main(String[] args) {
        HashSet<Person> set = new HashSet<>();
        set.add(new Person("David", 10));
        set.add(new Person("David", 10));
        System.out.println(set);
    }
}
```

HashSet: hashCode

- You may need to override hashCode() to implement your own hash function. In this case, your hashCode() method should obey the following rules.
  - In the same execution of an application, hashCode() should return the same value for the same object. For different executions of an application, hashCode() does not need to return the same value for the same object.
  - If equals() returns true for two objects (o1.equals(o2) returns true), then their hashCode() must be the same.
  - If equals() returns false for two objects (o1.equals(o2) returns false), it is better that their hashCode() are different. It is logically ok if the hashCode() of the two objects are the same, but that will lead to performance degradation when searching for elements in the hash set (or hash map).

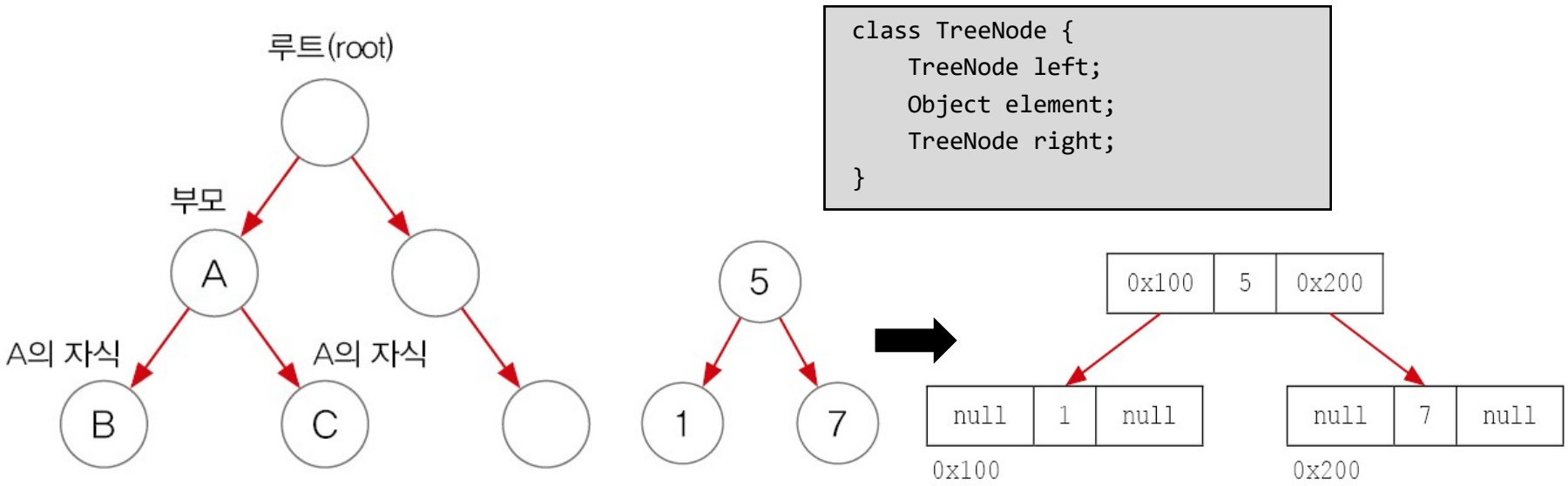
HashSet: Example 6

- A code that computes intersection, union, and difference of two sets.

```
HashSet<String> setA = new HashSet<>();
HashSet<String> setB = new HashSet<>();
HashSet<String> setHab = new HashSet<>();
HashSet<String> setKyo = new HashSet<>();
HashSet<String> setCha = new HashSet<>();
setA.add("1"); setA.add("2"); setA.add("3"); setA.add("4"); setA.add("5"); System.out.println("A = " + setA);
setB.add("4"); setB.add("5"); setB.add("6"); setB.add("7"); setB.add("8"); System.out.println("B = " + setB);

Iterator<String> it = setB.iterator();
while(it.hasNext()) { String tmp = it.next(); if(setA.contains(tmp)) setKyo.add(tmp); }

it = setA.iterator();
while(it.hasNext()) { String tmp = it.next(); if(!setB.contains(tmp)) setCha.add(tmp); }

it = setA.iterator();
while(it.hasNext()) setHab.add(it.next());
it = setB.iterator();
while(it.hasNext()) setHab.add(it.next());

System.out.println("A ∩ B = " + setKyo);
System.out.println("A ∪ B = " + setHab);
System.out.println("A – B = " + setCha);
```

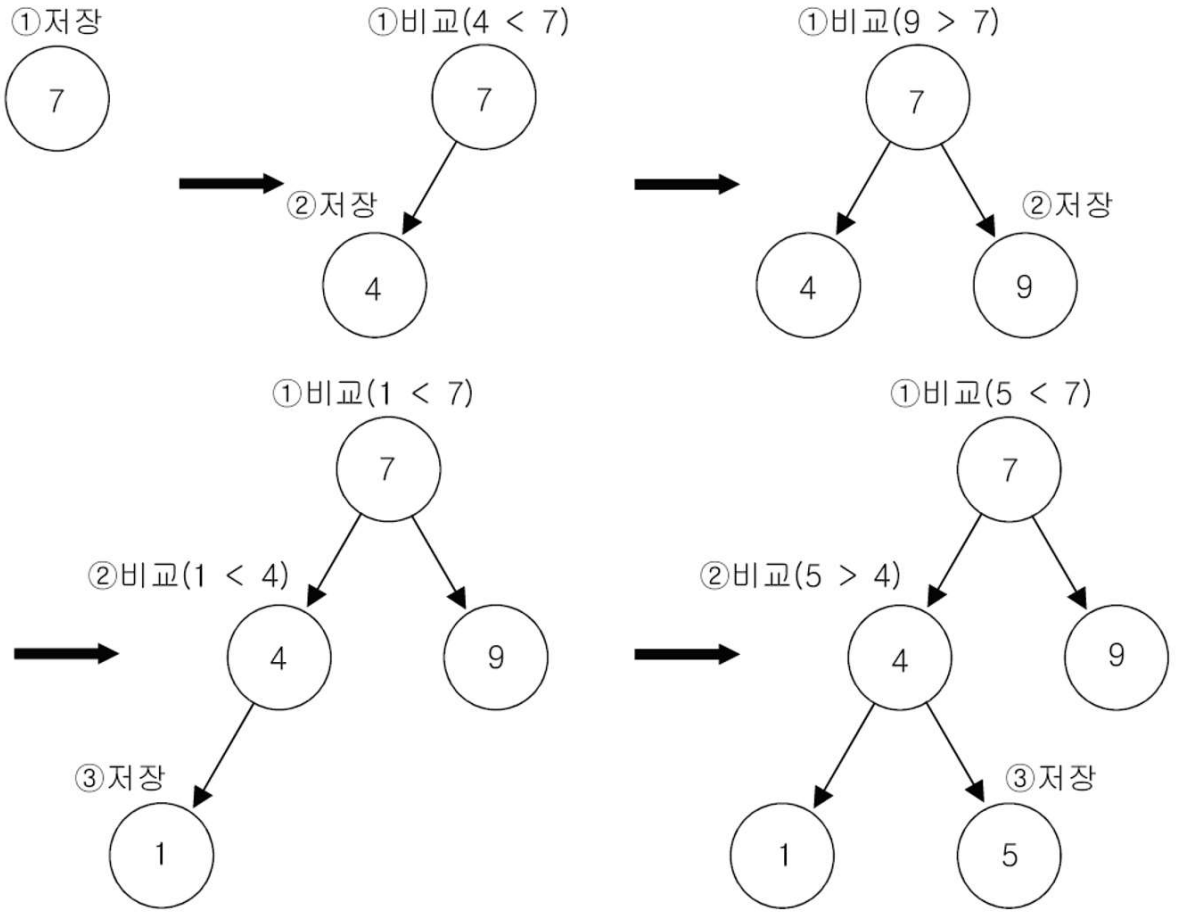
9. TreeSet

- TreeSet uses binary search tree to store data.
  - Each node has at most two child nodes.
  - Value of left child is smaller than the parent; value of right child is larger than the parent.
- Does not allow duplicates, and does not maintain order of input because the elements are sorted.
  - Nodes are placed according to the rules of a binary search tree.
- Good performance in sorting, searching, and range search.



TreeSet: Behavior

- Behavior of a tree set when 7, 4, 9, 1, 5 are added.



TreeSet: Methods

- Constructors

Method	Description
TreeSet()	Constructs a new, empty tree set, sorted according to the natural ordering of its elements.
TreeSet(Collection<? extends E> c)	Constructs a new tree set containing the elements in the specified collection, sorted according to the natural ordering of its elements.
TreeSet(Comparator<? super E> comparator	Constructs a new, empty tree set, sorted according to the specified comparator.
TreeSet(SortedSet<E> s)	Constructs a new tree set containing the same elements and using the same ordering as the specified sorted set.

TreeSet: Methods

- Methods defined in class TreeSet<E>

Method	Description
boolean add(E e)	Adds the speicifed element to this set if it is not already present.
boolean addAll(Collection<? extends E> c	Adds all of the elements in the specified collection to this set.
E ceiling(E e)	Returns the least element in this set greater than or equal to the given element, or null if there is no such element.
void clear()	Removes all of the elements from this set.
Object clone()	Returns a shallow copy of this TreeSet instance.
boolean contains(Object o)	Returns true if this set contains the specified element.
Iterator<E> descendingIterator()	Returns an iterator over the elements in this set in descending order.
NavigableSet<E> descendingSet()	Returns a reverse order view of the elements contained in this set.
E first()	Returns the first (lowest) element currently in this set.
E floor(E e)	Returns the greatest element in this set less than or equal to the given element, or null if there is no such element.
SortedSet<E> headSet(E toElement)	Returns a view of the portion of this set whose elements are strictly less than toElement.
NavigableSet<E> headSet(E toElement, boolean inclusive)	Returns a view of the portion of this set whose elements are less than (or equal to, if inclusive is true) toElement.
E higher(E e)	Returns the least element in this set strictly greater than the given element, or null if there is no such element.
boolean isEmpty()	Returns true if this set contains no elements.

TreeSet: Methods

- Methods defined in class TreeSet<E> (cont.)

Method	Description
Iterator<E> iterator()	Returns an iterator over the elements in this set in ascending order.
E last()	Returns the last (highest) element currently in this set.
E lower(E e)	Returns the greatest element in this set strictly less than the given element, or null if there is no such element.
E pollFirst()	Retries and removes the first (lowest) element, or returns null if this set is empty.
E pollLast()	Retries and removes the last (highest) element, or returns null if this set is empty.
boolean remove(Object o)	Removes the specified element from this set if it is present.
int size()	Returns the number of elements in this set (its cardinality).
Spliterator<E> spliterator()	Creates a late-binding and fail-fast Spliterator over the elements in this set.
NavigableSet<E> subset(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive)	Returns a vew of the portion of this set whose elements range from fromElement to toElement.
SortedSet<E> subset(E fromElement, E toElement)	Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive.
SortedSet<E> tailSet(E fromElement)	Returns a view of the portion of this set whose elements are greater than or equal to fromElement.
NavigableSet<E> tailSet(E fromElement, boolean inclusive)	Returns a vew of the portion of this set whose elements are greater than (or equal to, if inclusive is true) fromElement.

TreeSet: Example 1

- Lotto drawing using TreeSet.
  - Does not need to sort the elements.

```
public class Lecture {
    public static void main(String[] args) {
        Set<Integer> set = new TreeSet<>();

        for( ; set.size() < 6; ) {
            int num = (int)(Math.random() * 45) + 1;
            set.add(Integer.valueOf(num));
        }

        System.out.println(set);
    }
}
```

TreeSet: Example 2

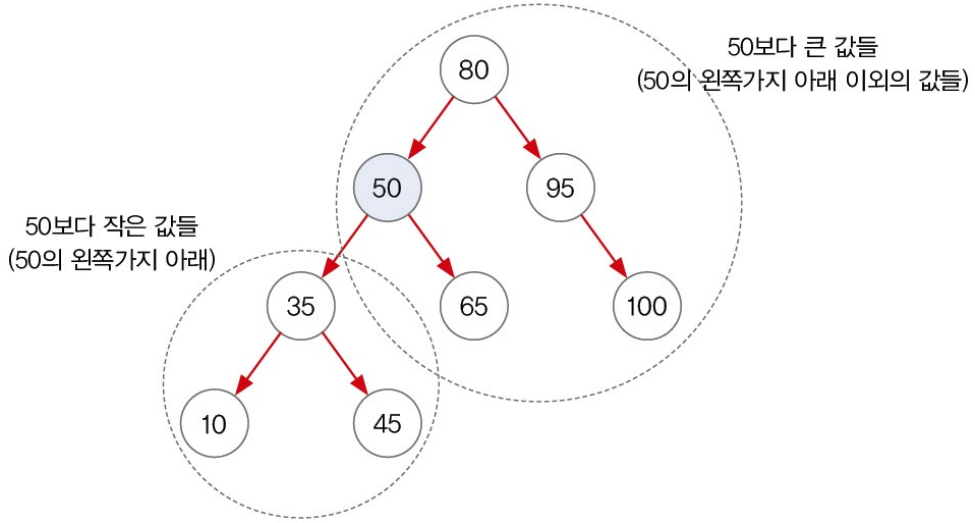
- An example of range search using a tree set.

```
public class Lecture {
    public static void main(String[] args) {
        TreeSet<String> set = new TreeSet<>();
        String from = "b";
        String to = "d";
        set.add("abc"); set.add("alien"); set.add("bat"); set.add("car"); set.add("Car");
        set.add("disc"); set.add("dance"); set.add("dZZZZ"); set.add("dzzzz"); set.add("elephant");
        set.add("elevator"); set.add("fan"); set.add("flower");
        System.out.println(set);
        System.out.println("range search: from " + from + " to " + to);
        System.out.println("result1: " + set.subSet(from, to));
        System.out.println("result2: " + set.subSet(from, to + "zzz"));
    }
}
```

TreeSet: Example 3

- An example of range search using a tree set.

```
public class Lecture {
    public static void main(String[] args) {
        TreeSet<Integer> set = new TreeSet<Integer>();
        int[] score = {80, 95, 50, 35, 45, 65, 10, 100};

        for(int i=0; i<score.length; i++) set.add(Integer.valueOf(score[i]));

        System.out.println("less than 50: " + set.headSet(Integer.valueOf(50)));
        System.out.println("greater than or equal to 50: " + set.tailSet(Integer.valueOf(50)));
    }
}
```



Programming Lab #19

19-01. Characteristics of a Set

- Write the following code and understand the results.
- Sets do not allow duplicate objects.
- Try using a list instead of a set.

```
class Ex19_01 {
    public static void main(String[] args) {
        String[] strArr = {"1", "1", "2", "2", "3", "3", "4", "4", "4"};
        Set<String> set = new HashSet<>();

        for(int i=0; i<strArr.length; i++) {
            set.add(strArr[i]);
        }

        System.out.println(set);
    }
}
```

19-02. HashSet vs. TreeSet

- Write the following code and understand the results.
- Try removing Collections.sort(list); for the case of HashSet and TreeSet.

```
class Ex19_02 {
    public static void main(String[] args) {
        Set<Integer> set = new HashSet<>();
        for( ; set.size() < 6; ) {
            int num = (int)(Math.random()*45) + 1;
            set.add(Integer.valueOf(num));
        }
        List<Integer> list = new LinkedList<>(set);
        Collections.sort(list);
        System.out.println(list);
    }
}
```

19-03. HashSet vs. LinkedHashSet

- Write the following code and understand the results.
- For a HashSet, elements are maintained according to Java's own way.
- For a LinkedHashSet, elements are maintained in the order of insertion.

```
class Ex19_03 {
    public static void main(String[] args) {
        Set<String> set = new LinkedHashSet<>();
        int[][] board = new int[5][5];
        for( ; set.size() < 25; ) {
            set.add((int)(Math.random()*50)+1+"");
        }
        Iterator<String> it = set.iterator();
        for(int i=0; i<board.length; i++) {
            for(int j=0; j<board[i].length; j++) {
                board[i][j] = Integer.parseInt(it.next());
                System.out.print((board[i][j] < 10 ? "  " : " ") + board[i][j]);
            }
            System.out.println();
        }
    }
}
```

19-04. Duplicate Objects in HashSet

- Write the following code and understand the results.
- Override methods equals() and hashCode() so that only one object is maintained in the set.
- In order for two objects to be determined as duplicate, they should be "equal".
  - equals() must return true.
  - hashCode() must return the same value.

```
class Person {
    String name;
    int age;
    Person(String name, int age) { this.name = name; this.age = age; }
    public String toString() { return name + ":" + age; }
}

public class Ex19_04 {
     public static void main(String[] args) {
        HashSet<Person> set = new HashSet<>();
        set.add(new Person("David", 10));
        set.add(new Person("David", 10));
        System.out.println(set);
    }
}
```

19-05. Range Search using TreeSet

- Write the following code and understand the results.
- Range search is possible in TreeSet because the elements are maintained as a binary search tree.

```
public class Ex19_05 {
     public static void main(String[] args) {
        TreeSet<String> set = new TreeSet<>();
        String from = "b";
        String to = "d";
        set.add("abc"); set.add("alien"); set.add("bat"); set.add("car"); set.add("Car");
        set.add("disc"); set.add("dance"); set.add("dZZZZ"); set.add("dzzzz"); set.add("elephant");
        set.add("elevator"); set.add("fan"); set.add("flower");
        System.out.println(set);
        System.out.println("range search: from " + from + " to " + to);
        System.out.println("result1: " + set.subSet(from, to));
        System.out.println("result2: " + set.subSet(from, to + "zzz"));
    }
}
```

19-06. Range Search using TreeSet 2

- Write the following code and understand the results.
- Range search is possible in TreeSet because the elements are maintained as a binary search tree.

```
public class Ex19_06 {
     public static void main(String[] args) {
        TreeSet<Integer> set = new TreeSet<Integer>();
        int[] score = {80, 95, 50, 35, 45, 65, 10, 100};

        for(int i=0; i<score.length; i++) set.add(Integer.valueOf(score[i]));

        System.out.println("less than 50: " + set.headSet(Integer.valueOf(50)));
        System.out.println("greater than or equal to 50: " + set.tailSet(Integer.valueOf(50)));
    }
}
```

End of Class

Instructor office: AS818A

Email: jso1@sogang.ac.kr