# CSE3040 Java Language

## Lecture 20: Collection Framework (5)

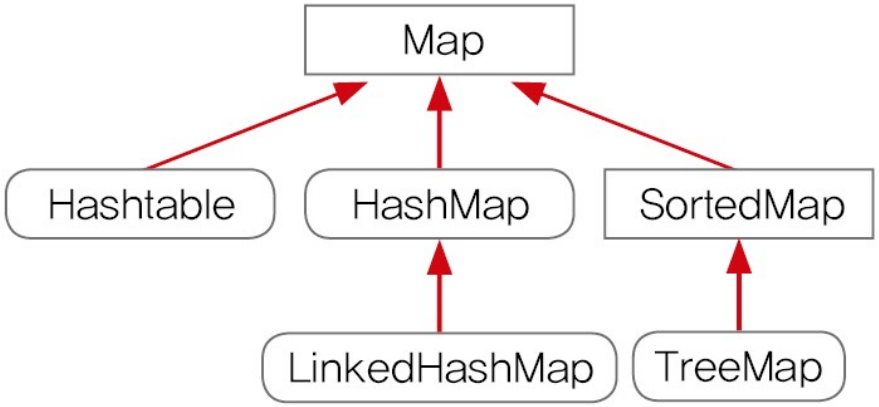
(HashMap and TreeMap)

Dept. of Computer Engineering,

Sogang University

# 10. HashMap

- HashMap is a class that implements interface Map
  - class Hashtable is similar to HashMap, but Hashtable is a synchronized class whereas HashMap is asynchronous.
  - HashMap stores data using hashing. Shows good performance in searching.



# HashMap

- class definition (partial)
  - defines an inner class Node<K,V> that implements Map.Entry<K,V>
  - The Node class has key and value as variables.
    - Keys must be unique inside a HashMap.
    - Duplicate values for different keys are allowed.
  - class HashMap has an array of Nodes as a variable named table.

```
public class HashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, Cloneable, Serializable {

    transient Node<K,V>[] table;
    ...
    static class Node<K,V> implements Map.Entry<K,V> {
        final K key;
        V value;
        ...
    }
}
```

# HashMap: Methods

- Constructors defined in HashMap<K,V>

| Method | Description |
|---|---|
| HashMap() | Constructs an empty HashMap with the default initial capacity (16) and the default load factor (0.75). |
| HashMap(int initialCapacity) | Constructs an empty HashMap with the specified initial capacity and the default load factor (0.75). |
| HashMap(int initialCapacity, float loadFactor) | Constructs an empty HashMap with the specified initial capacity and load factor. |
| HashMap(Map<? extends K, ? extends V> m | Constructs a new HashMap with the same mappings as the specified Map. |

# HashMap: Methods

- Methods defined in HashMap<K,V>

| Method | Description |
|---|---|
| void clear() | Removes all of the mappings from this map. |
| Object clone() | Returns a shallow copy of this HashMap instance: the keys and values them selves are not cloned. |
| boolean containsKey(Objet key) | Returns true if this map contains a mapping for the specified key. |
| boolean containsValue(Object value) | Returns true if this map maps one or more keys to the specified value. |
| Set<Map.Entry<K,V>> entrySet() | Returns a Set view of the mapping contained in this map. |
| V get(Object key) | Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key. |
| boolean isEmpty() | Returns true if this map contains no key-value mappings. |
| Set<K> keySet() | Returns a Set view of the keys contained in this map. |
| V put(K key, V value) | Associates the specified value with the specified key in this map. |
| void putAll(Map<? extends K, ? extends V> m | Copies all of the mappings from the specified map to this map. |
| V remove(Object key) | Removes the mapping for the specified key from this map if present. |
| int size() | Returns the number of key-value mappings in this map. |
| Collection<V> values() | Returns a Collection view of the values contained in this map. |

# HashMap: Example 1

- Authentication using HashMap

```
HashMap<String,String> map = new HashMap<>();
map.put("myId", "1234");
map.put("asdf", "1111");
map.put("asdf", "1234");
Scanner s = new Scanner(System.in);
while(true) {
    System.out.println("Please enter id and password.");
    System.out.print("id: ");
    String id = s.nextLine().trim();
    System.out.print("password: ");
    String password = s.nextLine().trim();
    System.out.println();
    if(!map.containsKey(id)) {
        System.out.println("id does not exist.");
        continue;
    } else {
        if(!(map.get(id)).equals(password)) {
            System.out.println("password does not match.");
        } else {
            System.out.println("welcome.");
            break;
        }
    }
}
s.close();
```

# HashMap: Example 2

- Reading and writing data using HashMap

```
HashMap<String,Integer> map = new HashMap<>();
map.put("Kim Java", Integer.valueOf(90));
map.put("Kim Java", Integer.valueOf(100));
map.put("Lee Java", Integer.valueOf(100));
map.put("Kang Java", Integer.valueOf(80));
map.put("Ahn Java", Integer.valueOf(90));

Set<Map.Entry<String,Integer>> set = map.entrySet();
Iterator<Map.Entry<String,Integer>> it = set.iterator();
while(it.hasNext()) {
    Map.Entry<String,Integer> e = (Map.Entry<String,Integer>)it.next();
    System.out.println("name: " + e.getKey() + ", score: " + e.getValue());
}

Set<String> keys = map.keySet();
System.out.println("participants: " + keys);
Collection<Integer> values = map.values();
Iterator<Integer> it2 = values.iterator();
int total = 0;

while(it.hasNext()) {
    Integer i = (Integer)it2.next();
    total += i.intValue();
}
```

# HashMap: Example 2

- Reading data using HashMap (cont.)

```
System.out.println("total: " + total);
System.out.println("average: " + (float)total/set.size());
System.out.println("max: " + Collections.max(values));
System.out.println("min: " + Collections.min(values));
```

# HashMap: Example 3

- HashMap can use HashMap as its parameterized type.

```
public class Lecture {
    static HashMap<String,HashMap<String,String>> phoneBook = new HashMap<>();

    public static void main(String[] args) {
        addPhoneNo("friend", "Lee Java", "010-111-1111");
        addPhoneNo("friend", "Kim Java", "010-222-2222");
        addPhoneNo("friend", "Kim Java", "010-333-3333");
        addPhoneNo("work", "Kim Daeri", "010-444-4444");
        addPhoneNo("work", "Kim Daeri", "010-555-5555");
        addPhoneNo("work", "Park Daeri", "010-666-6666");
        addPhoneNo("work", "Lee Guajang", "010-777-7777");
        addPhoneNo("laundary", "010-888-8888");
        printList();
    }

    static void addPhoneNo(String groupName, String name, String tel) {
        addGroup(groupName);
        HashMap<String,String> group = phoneBook.get(groupName);
        group.put(tel, name);  // use phone number as key because names can have duplicates.
    }

    static void addGroup(String groupName) {
        if(!phoneBook.containsKey(groupName)) phoneBook.put(groupName, new HashMap<String,String>());
    }
```

# HashMap: Example 3

```
    static void addPhoneNo(String name, String tel) {
        addPhoneNo("others", name, tel);
    }

    static void printList() {
        Set<Map.Entry<String,HashMap<String,String>>> set = phoneBook.entrySet();
        Iterator<Map.Entry<String,HashMap<String,String>>> it = set.iterator();

        while(it.hasNext()) {
            Map.Entry<String,HashMap<String,String>> e = (Map.Entry<String,HashMap<String,String>>)it.next();
            Set<Map.Entry<String,String>> subset = e.getValue().entrySet();
            Iterator<Map.Entry<String,String>> subIt = subset.iterator();
            System.out.println(" * " + e.getKey() + "[" + subset.size() + "]");
            while(subIt.hasNext()) {
                Map.Entry<String,String> subE = (Map.Entry<String,String>)subIt.next();
                String telNo = subE.getKey();
                String name = subE.getValue();
                System.out.println(name + " " + telNo);
            }
            System.out.println();
        }
    }
}
```

# HashMap: Example 4

- Counting number of characters using HashMap

```
public class Lecture {
    public static void main(String[] args) {

        String[] data = {"A", "K", "A", "K", "D", "K", "A", "K", "K", "K", "Z", "D"};

        HashMap<String,Integer> map = new HashMap<>();
        for(int i=0; i<data.length; i++) {
            if(map.containsKey(data[i])) {
                Integer value = map.get(data[i]);
                map.put(data[i], Integer.valueOf(value.intValue() + 1));
            } else {
                map.put(data[i], Integer.valueOf(1));
            }
        }

        Iterator<Map.Entry<String,Integer>> it = map.entrySet().iterator();
        while(it.hasNext()) {
            Map.Entry<String,Integer> entry = it.next();
            int value = entry.getValue().intValue();
            System.out.println(entry.getKey() + ": " + printBar('#', value) + " " + value);
        }
    }
```

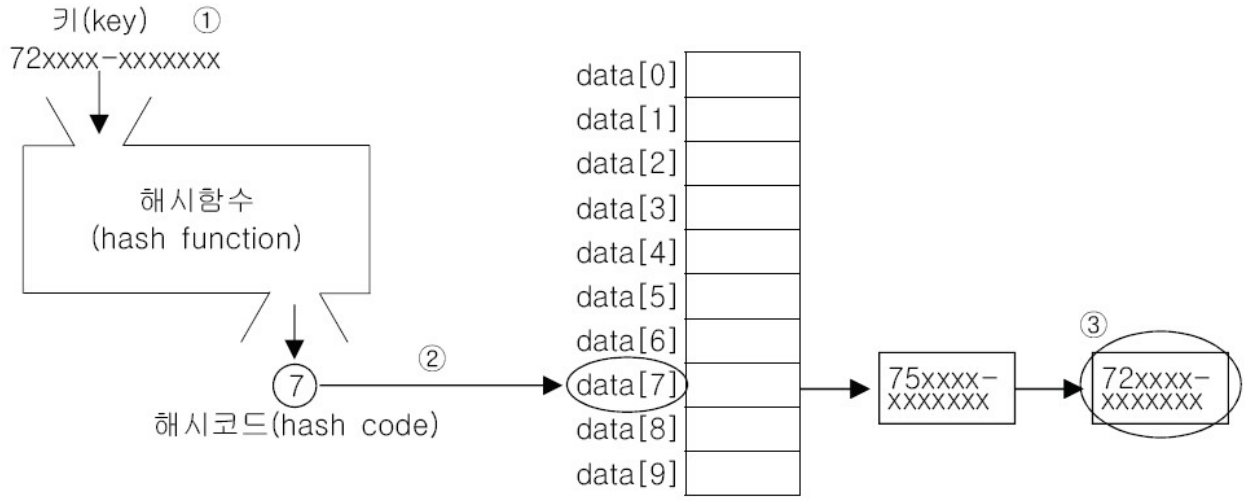
# HashMap: Example 4

- Counting number of characters using HashMap (cont.)

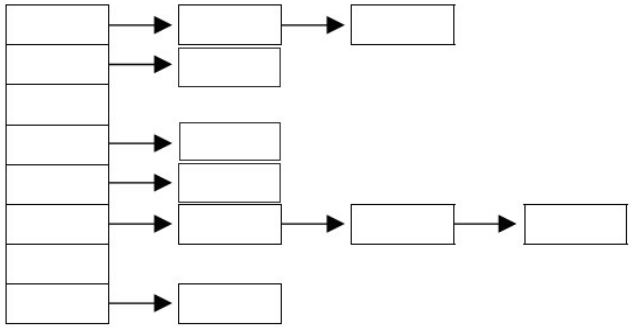
```
    public static String printBar(char ch, int value) {
        char[] bar = new char[value];
        for(int i=0; i<bar.length; i++) bar[i] = ch;
        return new String(bar);
    }
}
```

# HashMap: Hashing

- Class HashMap uses hashing to store data and also search for data.
  - Data entry is found by hashing the key.




  - For colliding entries, a linked list structure is used to store data



# HashMap: Hashing

- On hashing performance
  - Calculating hash value using a hash function is very fast.
  - If multiple data are stored under a "slot" (or a "bucket") using a linked list, searching data through a linked list is slow.
  - For search performance, it is better to have only one data entry per slot.

- Collection classes that use hashing such as HashMap uses method hashCode as the hash function to compute the slot.
  - The hashCode() defined in class Object uses an algorithm which takes object reference as input – the result of hashCode() is different for different objects.
  - If a class overrides hashCode() and the class is used as the type of keys in a HashMap, then this hashCode() will be used to compute slots in the HashMap.
    - If a class overrides equals(), it should also override hashCode() to match the definition of equality.
    - If the class overrides equals() but does not override hashCode(), then classes based on Hashing may regard two objects as different objects even if equals() returns true.

# 11. TreeMap

- A Map which uses a tree structure (Red-Black tree) to store data.
- When new elements are added to a TreeMap, TreeMap sorts the elements according to the order of keys.
- If the class that is used for keys is not comparable, an exception is thrown when a new element is added to the TreeMap.
  - Same thing for TreeSet: keys must be comparable, or an exception occurs.
- In general, HashMap finds the entry faster than TreeMap.
- TreeMap is useful when we need to sort the elements by the key.

- Constructors for TreeMap<K,V>

| Method | Description |
|---|---|
| TreeMap() | Constructs a new, empty tree map, using the natural ordering of its keys. |
| TreeMap(Comparator<? super K> comparator) | Constructs a new, empty tree map, ordered according to the given comparator. |
| TreeMap(Map<? extends K, ? extends V> m | Constructs a new tree map containing the same mappings as the given map, ordered according to the natural ordering of its keys. |
| TreeMap(SortMap<K, ? extends V> m | Constructs a new tree map containing the same mapping and using the same ordering as the specified sorted map. |

# TreeMap: Methods

- Methods defined in TreeMap<K,V>

| Method | Description |
|---|---|
| Map.Entry<K,V> ceilingEntry(K key) | Returns a key-value mapping associated with the least key greater than or equal to the given key, or null if there is no such key. |
| K ceilingKey(K key) | Returns the least key greater than or equal to the given key, or null if there is no such key. |
| void clear() | Removes all of the mappings from this map. |
| Object clone() | Returns a shallow copy of this TreeMap instance. |
| boolean containsKey(Object key) | Returns true if this map contains a mapping for the specified key. |
| boolean containsValue(Object value) | Returns true if this map maps one or more keys to the specified value. |
| NavigableSet<K> descendingKeySet() | Returns a reverse order NavigableSet view of the keys contained in this map. |
| NavigableMap<K,V> descendingMap() | Returns a reverse order view of the mappings contained in this map. |
| Set<Map.Entry<K,V>> entrySet() | Returns a Set view of the mappings contained in this map. |
| Map.Entry<K,V> firstEntry() | Returns a key-value mapping associated with the least key in this map, or null if the map is empty. |
| K firstKey() | Returns the first (lowest) key currently in this map. |
| Map.Entry<K,V> floorEntry(K key) | Returns a key-value mapping associated with the greatest key less than or equal to the given key, or null if there is no such key. |
| K floorKey(K key) | Returns the greatest key less than or equal to the given key, or null if there is no such key. |

# TreeMap: Methods

- Methods defined in TreeMap<K,V> (cont.)

| Method | Description |
|---|---|
| V get(Object key) | Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key. |
| SortedMap<K,V> headMap(K toKey) | Returns a view of the portion of this map whose keys are strictly less than toKey. |
| NavigableMap<K,V> headMap(K toKey, boolean inclusive) | Returns a view of the portion of this map whose keys are less than (or equal to, if inclusive is true) toKey. |
| Map.Entry<K,V> higherEntry(K key) | Returns a key-value mapping associated with the least key strictly greater than the given key, or null if there is no such key. |
| K higherKey(K key) | Returns the least key strictly greater than the given key, or null if there is no such key. |
| Set<K> keySet() | Returns a Set view of the keys contained in this map. |
| Map.Entry<K,V> lastEntry() | Returns a key-value mapping associated with the greatest key in this map, or null if the map is empty. |
| K lastKey() | Returns the last (highest) key currently in this map. |
| Map.Entry<K,V> lowerEntry(K key) | Returns a key-value mapping associated with the greatest key strictly less than the given key, or null if there is no such key. |
| K lowerKey(K key) | Returns the greatest key strictly less than the given key, or null if there is no such key. |
| NavigableSet<K> navigableKeySet() | Returns a NavigableSet view of the keys contained in this map. |

# TreeMap: Methods

- Methods defined in TreeMap<K,V> (cont.)

| Method | Description |
|---|---|
| Map.Entry<K,V> pollFirstEntry() | Removes and returns a key-value mapping associated with the least key in this map, or null if the map is empty. |
| Map.Entry<K,V> pollLastEntry() | Removes and returns a key-value mapping associated with the greatest key in this map, or null if the map is empty. |
| V put(K key, V value) | Associates the specified value with the specified key in this map. |
| void putAll(Map<? extends K, ? extends V> map) | Copies all of the mappings from the specified map to this map. |
| V remove(Object key) | Removes the mapping for this key from this TreeMap if present. |
| int size() | Returns the number of key-value mappings in this map. |
| NavigableMap<K,V> submap(K fromKey, boolean fromInclusive, K toKey, boolean toInclusive) | Returns a view of the portion of this map whose keys range from fromKey to toKey. |
| SortedMap<K,V> subMap(K fromKey, K toKey) | Returns a view of the portion of this map whose keys range from fromKey, inclusive, to toKey, exclusive. |
| SortedMap<K,V> tailMap(K fromKey) | Returns a view of the portion of this map whose keys are greater than or equal to fromKey. |
| NavigableMap<K,V> tailMap(K fromKey, boolean inclusive) | Returns a view of the portion of this map whose keys are greater than (or equal to, if inclusive is true) fromKey. |
| Collection<V> values() | Returns a Collection view of the values contained in this map. |

# TreeMap: Example

- A program that sorts characters by number of appearances.

```
public class Lecture {
    public static void main(String[] args) {
        String[] data = {"A","K","A","K","D","K","A","K","K","K","Z","D"};
        TreeMap<String,Integer> map = new TreeMap<>();
        for(int i=0; i<data.length; i++) {
            if(map.containsKey(data[i])) {
                Integer value = map.get(data[i]);
                map.put(data[i], Integer.valueOf(value.intValue() + 1));
            } else {
                map.put(data[i], Integer.valueOf(1));
            }
        }

        Iterator<Map.Entry<String,Integer>> it = map.entrySet().iterator();
        System.out.println("=== basic sort ===");
        while(it.hasNext()) {
            Map.Entry<String,Integer> entry = it.next();
            int value = entry.getValue().intValue();
            System.out.println(entry.getKey() + ": " + printBar('#', value) + " " + value);
        }
        System.out.println();
```

# TreeMap: Example

- A program that sorts characters by number of appearances. (cont.)

```
        Set<Map.Entry<String,Integer>> set = map.entrySet();
        List<Map.Entry<String,Integer>> list = new ArrayList<>(set);
        Collections.sort(list, new ValueComparator<Map.Entry<String,Integer>>());

        it = list.iterator();
        System.out.println("=== sort by the value ===");
        while(it.hasNext()) {
            Map.Entry<String,Integer> entry = it.next();
            int value = entry.getValue().intValue();
            System.out.println(entry.getKey() + ": " + printBar('#', value) + " " + value);
        }
    }
```

# TreeMap: Example

- A program that sorts characters by number of appearances. (cont.)

```
    // inner class
    @SuppressWarnings("unchecked")
    static class ValueComparator<T> implements Comparator<T> {
        public int compare(Object o1, Object o2) {
            if(o1 instanceof Map.Entry && o2 instanceof Map.Entry) {
                Map.Entry<String,Integer> e1 = (Map.Entry<String, Integer>)o1;
                Map.Entry<String,Integer> e2 = (Map.Entry<String, Integer>)o2;
                int v1 = e1.getValue().intValue();
                int v2 = e2.getValue().intValue();
                return v2 - v1;
            }
            return -1;
        }
    }

    public static String printBar(char ch, int value) {
        char[] bar = new char[value];
        for(int i=0; i<bar.length; i++) bar[i] = ch;
        return new String(bar);
    }
}
```

# Programming Lab #20

# 20-01. Authentication using HashMap

- Try entering id: asdf, password: 1111. Why doesn't it work?

```
public class Ex20_01 {
	public static void main(String[] args) {
		HashMap<String,String> map = new HashMap<>();
		map.put("myId", "1234");
		map.put("asdf", "1111");
		map.put("asdf", "1234");
		Scanner s = new Scanner(System.in);
		while(true) {
		    System.out.println("Please enter id and password.");
		    System.out.print("id: ");
		    String id = s.nextLine().trim();
		    System.out.print("password: ");
		    String password = s.nextLine().trim();
		    System.out.println();
		    if(!map.containsKey(id)) {
		        System.out.println("id does not exist.");
		        continue;
		    } else {
		        if(!(map.get(id)).equals(password)) {
		            System.out.println("password does not match.");
		        } else {
		            System.out.println("welcome.");
		            break;
		        }
		    }
		}
		s.close();
	}
}
```

# 20-02. Reading and Writing Data using HashMap

- Write the following code and understand the results.

```
public class Ex20_02 {
	public static void main(String[] args) {
		HashMap<String,Integer> map = new HashMap<>();
		map.put("Kim Java", Integer.valueOf(90));
		map.put("Kim Java", Integer.valueOf(100));
		map.put("Lee Java", Integer.valueOf(100));
		map.put("Kang Java", Integer.valueOf(80));
		map.put("Ahn Java", Integer.valueOf(90));

		Set<Map.Entry<String,Integer>> set = map.entrySet();
		Iterator<Map.Entry<String,Integer>> it = set.iterator();
		while(it.hasNext()) {
		    Map.Entry<String,Integer> e = (Map.Entry<String,Integer>)it.next();
		    System.out.println("name: " + e.getKey() + ", score: " + e.getValue());
		}

		Set<String> keys = map.keySet();
		System.out.println("participants: " + keys);
		Collection<Integer> values = map.values();
		Iterator<Integer> it2 = values.iterator();
		int total = 0;
```

# 20-02. Reading and Writing Data using HashMap

- Write the following code and understand the results. (cont.)

```
      while(it2.hasNext()) {
                Integer i = (Integer)it2.next();
                total += i.intValue();
            }

            System.out.println("total: " + total);
            System.out.println("average: " + (float)total/set.size());
            System.out.println("max: " + Collections.max(values));
            System.out.println("min: " + Collections.min(values));
      }
}
```

# 20-03. HashMap inside HashMap

- Write the following code and understand the results.

```
public class Ex20_03 {
    static HashMap<String,HashMap<String,String>> phoneBook = new HashMap<>();

    public static void main(String[] args) {
        addPhoneNo("friend", "Lee Java", "010-111-1111");
        addPhoneNo("friend", "Kim Java", "010-222-2222");
        addPhoneNo("friend", "Kim Java", "010-333-3333");
        addPhoneNo("work", "Kim Daeri", "010-444-4444");
        addPhoneNo("work", "Kim Daeri", "010-555-5555");
        addPhoneNo("work", "Park Daeri", "010-666-6666");
        addPhoneNo("work", "Lee Guajang", "010-777-7777");
        addPhoneNo("laundary", "010-888-8888");
        printList();
    }

    static void addPhoneNo(String groupName, String name, String tel) {
        addGroup(groupName);
        HashMap<String,String> group = phoneBook.get(groupName);
        group.put(tel, name);  // use phone number as key because names can have duplicates.
    }

    static void addGroup(String groupName) {
        if(!phoneBook.containsKey(groupName)) phoneBook.put(groupName, new HashMap<String,String>());
    }

    static void addPhoneNo(String name, String tel) {
        addPhoneNo("others", name, tel);
    }
```

# 20-03. HashMap inside HashMap

- Write the following code and understand the results. (cont.)

```
    static void printList() {
        Set<Map.Entry<String,HashMap<String,String>>> set = phoneBook.entrySet();
        Iterator<Map.Entry<String,HashMap<String,String>>> it = set.iterator();

        while(it.hasNext()) {
            Map.Entry<String,HashMap<String,String>> e = (Map.Entry<String,HashMap<String,String>>)it.next();
            Set<Map.Entry<String,String>> subset = e.getValue().entrySet();
            Iterator<Map.Entry<String,String>> subIt = subset.iterator();
            System.out.println(" * " + e.getKey() + "[" + subset.size() + "]");
            while(subIt.hasNext()) {
                Map.Entry<String,String> subE = (Map.Entry<String,String>)subIt.next();
                String telNo = subE.getKey();
                String name = subE.getValue();
                System.out.println(name + " " + telNo);
            }
            System.out.println();
        }
    }
}
```

# 20-04. HashMap vs. TreeMap

- Write the following code and understand the results.
- Try using a TreeMap instead of HashMap. How are the results different?

```
public class Ex20_04 {
     public static void main(String[] args) {

        String[] data = {"A", "K", "A", "K", "D", "K", "A", "K", "K", "K", "Z", "D"};

        TreeMap<String,Integer> map = new TreeMap<>();
        for(int i=0; i<data.length; i++) {
            if(map.containsKey(data[i])) {
                Integer value = map.get(data[i]);
                map.put(data[i], Integer.valueOf(value.intValue() + 1));
            } else {
                map.put(data[i], Integer.valueOf(1));
            }
        }

        Iterator<Map.Entry<String,Integer>> it = map.entrySet().iterator();
        System.out.println("=== basic sort ===");
        while(it.hasNext()) {
            Map.Entry<String,Integer> entry = it.next();
            int value = entry.getValue().intValue();
            System.out.println(entry.getKey() + ": " + printBar('#', value) + " " + value);
        }

        Set<Map.Entry<String,Integer>> set = map.entrySet();
        List<Map.Entry<String,Integer>> list = new ArrayList<>(set);
        Collections.sort(list, new ValueComparator<Map.Entry<String,Integer>>());
```

# 20-04. HashMap vs. TreeMap

- Write the following code and understand the results. (cont.)

```
        it = list.iterator();
        System.out.println("=== sort by the value ===");
        while(it.hasNext()) {
            Map.Entry<String,Integer> entry = it.next();
            int value = entry.getValue().intValue();
            System.out.println(entry.getKey() + ": " + printBar('#', value) + " " + value);
        }
    }

    @SuppressWarnings("unchecked")
    static class ValueComparator<T> implements Comparator<T> {
        public int compare(Object o1, Object o2) {
            if(o1 instanceof Map.Entry && o2 instanceof Map.Entry) {
                Map.Entry<String,Integer> e1 = (Map.Entry<String, Integer>)o1;
                Map.Entry<String,Integer> e2 = (Map.Entry<String, Integer>)o2;
                int v1 = e1.getValue().intValue();
                int v2 = e2.getValue().intValue();
                return v2 - v1;
            }
            return -1;
        }
    }

    public static String printBar(char ch, int value) {
        char[] bar = new char[value];
        for(int i=0; i<bar.length; i++) bar[i] = ch;
        return new String(bar);
    }
}
```

# End of Class


Instructor office: AS818A

Email: jso1@sogang.ac.kr