CSE3040 Java Language

Lecture 16: Collection Framework (1)

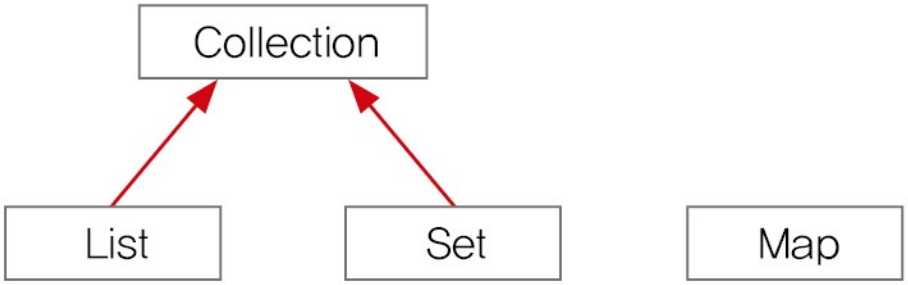
Dept. of Computer Engineering,
Sogang University

Collection Framework

- A framework for defining and using classes that deal with group of objects.
- Java provides three major types of collections.
  - Implemented as interfaces.
- List
  - A group of data that has an ordering. Allows duplicate data.
  - e.g.) customers waiting to get a seat in a restaurant
  - Classes: ArrayList, LinkedList, Stack, Vector
- Set
  - A group of data that does not have an ordering. Does not allow duplicate data.
  - e.g.) a set of positive integers, a set of prime numbers
  - Classes: HashSet, TreeSet
- Map
  - A group of data which consist of key-value pairs. Has no ordering. Duplicate keys not allowed. Duplicate values are allowed.
  - e.g.) area codes, zip codes
  - Classes: HashMap, TreeMap, Hashtable, Properties

1. Interfaces defined in the Collection Framework

- Hierarchy of interfaces
  - Interface List and Interface Set are subtypes of Interface Collection
  - Interface Map is a separate interface



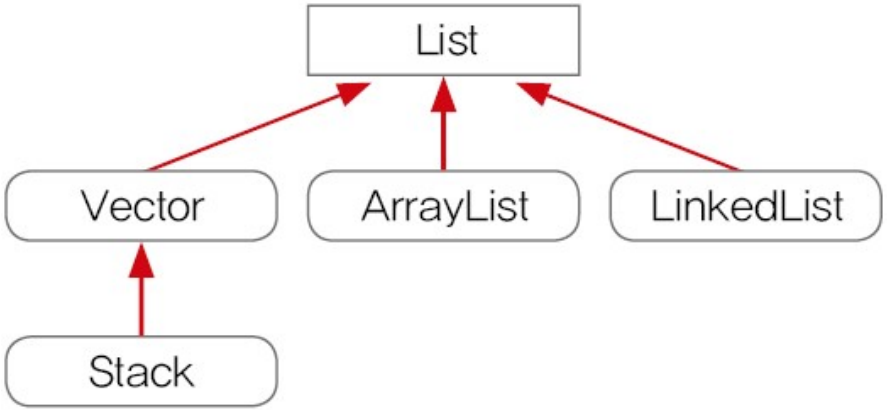
Interface Collection<E>

- Methods defined in interface Collection<E>.

Method	Description
boolean add(E e) boolean addAll(Collection<? extends E> c)	Adds an object to the collection. Adds objects in a collection to the collection.
void clear()	Deletes all objects in the collection.
boolean contains(Object o) boolean containsAll(Collection<?> c)	Checks whether an object is included in the collection. Checks whether objects in a collection is included in the collection.
boolean equals(Object o)	Checks whether the collection is equal to another collection.
int hashCode()	Returns the hash code of the collection.
boolean isEmpty()	Checks whether the collection is empty.
Iterator iterator()	Returns the Iterator of the collection.
boolean remove(Object o)	Removes an object from the collection.
boolean removeAll(Collection<?> c)	Removes objects in a collection from the collection.
boolean retainAll(Collection<?> c)	Maintains objects in a collection and removes all else from the collection.
int size()	Returns the number of objects in the collection.
Object[] toArray()	Converts objects stored in the collection to an object array.
<T> T[] toArray(T[] a)	Sets the given object array with objects in the collection.

Lists

- An ordered collection
- Allows duplicate elements



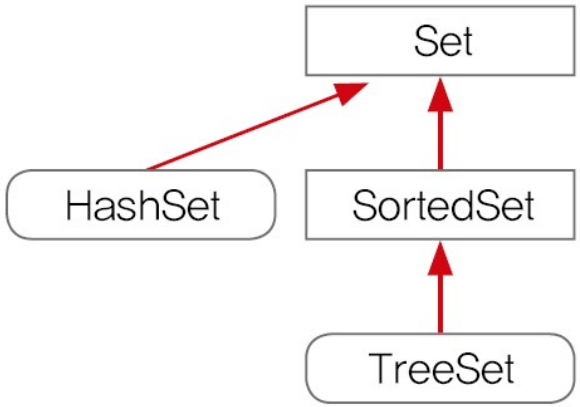
Interface List<E>

- Methods defined in interface List<E>.

Method	Description
void add(int index, E element) boolean addAll(int index, Collection<? extends E> c)	Inserts the specified element at the specified position in the list. Inserts all of the elements in the specified collection into the list at the specified position.
E get(int index)	Returns the element at the specified position in the list.
int indexOf(Object o)	Returns the index of the first occurrences of the specified element in the list, or -1 if the list does not contain the element.
int lastIndexOf(Object o)	Returns the index of the last occurrences of the specified element in the list, or -1 if the list does not contain the element.
ListIterator<E> listIterator() ListIterator<E> listIterator(int index)	Returns a list iterator over the elements in the list. Returns a list iterator over the elements in the list, starting at the specified position in the list.
E remove(int index)	Removes the element at the specified position in the list.
E set(int index, E element)	Replaces the element at the specified position in this list with the specified element.
void sort(Comparator<? super E> c)	Sorts this list according to the order induced by the specified **Comparator**.
List<E> subList(int fromIndex, int toIndex)	Returns a view of the portion of this list between the specified **fromIndex**, inclusive, and **toIndex**, exclusive.

Sets

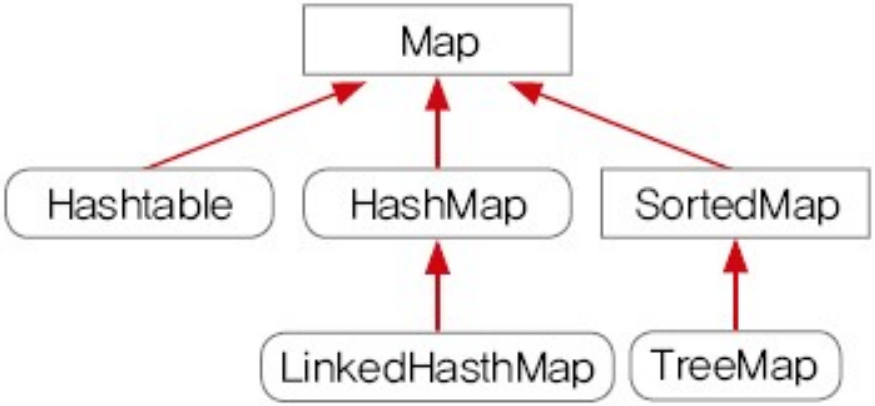
- A collection that contains no duplicate elements
  - No pair of elements e1 and e2 such that e1.equals(e2)
  - At most one null element
  - No ordering of elements



- interface Set<E>
  - The list of methods is the same as interface Collection<E>.

Maps

- An object that maps keys to values.
- A map cannot contain duplicate keys.
- Each key can map to at most one value.



Interface Map<K,V>

- Methods defined in interface Map<K,V>.

Method	Description
void clear()	Removes all of the mapping from this map.
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.
Set<Map.entry<K,V>> entrySet()	Returns a Set view of the mappings contained in this map.
boolean equals(Object o)	Compares the specified object with this map for equality.
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.
int hashCode()	Returns the hash code value for this map.
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 mapping from the specified map to this map.
V remove(Object key)	Removes the mapping for a key from this map if it is 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.

Map.Entry

- Map.Entry is an inner interface that is defined inside interface Map.

```
public interface Map<K,V> {
    ...
    interface Entry<K,V> {
        K getKey();
        V getValue();
        V setValue(V value);
        boolean equals(Object o);
        int hashCode();
        ...
    }
}
```

- Methods defined in Map.Entry<K,V>

Method	Description
boolean equals(Object o)	Compares the specified object with this entry for equality.
K getKey()	Returns the key corresponding to this entry.
V getValue()	Returns the value corresponding to this entry.
int hashCode()	Returns the hash code value for this map entry.
V setValue(V value)	Replaces the value corresponding to this entry with the specified value.

2. ArrayLists

- ArrayList
  - A frequently used collection class.
  - Implements interface List.
    - Elements have order.
    - Duplicates are allowed.

```
public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable
```

```
public class AbstractList<E> extends AbstractCollection<E> implements List<E>
```

```
public class AbstractCollection<E> extends Object implements Collection<E>
```

  - Internally contains an instance variable which is an Object array.
    - The size of array is automatically allocated according to the number of elements.

```
public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable {
    ...
    transient Object[] elementData;
    ...
}
```

ArrayLists: Example 1

- main method (1/2)

```
import java.util.ArrayList;
import java.util.Collections;
public class Lecture {
    public static void main(String[] args) {
        ArrayList<Integer> list1 = new ArrayList<Integer>(10);
        list1.add(Integer.valueOf(5));
        list1.add(Integer.valueOf(4));
        list1.add(Integer.valueOf(2));
        list1.add(Integer.valueOf(0));
        list1.add(Integer.valueOf(1));
        list1.add(Integer.valueOf(3));

        ArrayList<Integer> list2 = new ArrayList<Integer>(list1.subList(1,4));
        print(list1, list2);

        Collections.sort(list1);
        Collections.sort(list2);
        print(list1, list2);

        System.out.println("list1.containsAll(list2):"+list1.containsAll(list2));
        list2.add(Integer.valueOf(11));
        list2.add(Integer.valueOf(12));
        list2.add(Integer.valueOf(13));
        print(list1, list2);
```

ArrayLists: Example 1

- main method (2/2), print method

```
        list2.set(3, Integer.valueOf(21));
        print(list1, list2);

        System.out.println("list1.retainAll(list2):"+list1.retainAll(list2));
        print(list1, list2);

        for(int i=list2.size()-1; i>=0; i--) {
            if(list1.contains(list2.get(i)))
                list2.remove(i);
        }
        print(list1, list2);
    }

    static void print(ArrayList<Integer> list1, ArrayList<Integer> list2) {
        System.out.println("list1:"+list1);
        System.out.println("list2:"+list2);
        System.out.println();
    }
}
```

ArrayLists: Example 1

- What happens if we change the order of iteration?

```
        list2.set(3, Integer.valueOf(21));
        print(list1, list2);

        System.out.println("list1.retainAll(list2):"+list1.retainAll(list2));
        print(list1, list2);

        for(int i=0; i<=list2.size()-1; i++) {
            if(list1.contains(list2.get(i)))
                list2.remove(i);
        }
        print(list1, list2);
    }

    static void print(ArrayList<Integer> list1, ArrayList<Integer> list2) {
        System.out.println("list1:"+list1);
        System.out.println("list2:"+list2);
        System.out.println();
    }
}
```

ArrayLists: Example 2

- When creating an ArrayList, it is common to make the list larger than the expected number of elements
  - If number of elements exceeds the size, the size is automatically increased but it takes time.

```
import java.util.ArrayList;
import java.util.List;

public class Lecture {
    public static void main(String[] args) {
        final int LIMIT = 10;
        String source = "0123456789abcdefghijABCDEFGHIJ!@#$%^&*()ZZZ";
        int length = source.length();

        List<String> list = new ArrayList<String>(length/LIMIT + 10);

        for(int i=0; i<length; i+=LIMIT) {
            if(i+LIMIT < length) list.add(source.substring(i, i+LIMIT));
            else list.add(source.substring(i));
        }

        for(int i=0; i<list.size(); i++) {
            System.out.println(list.get(i));
        }
    }
}
```

ArrayLists: Example 3

- A Vector is similar to an ArrayList, but is synchronized.
  - Only one thread can access a vector at a time.

```
import java.util.*;
public class Lecture {
    public static void main(String[] args) {
        Vector<String> v = new Vector<>(5);
        v.add("1");
        v.add("2");
        v.add("3");
        print(v);
        v.trimToSize();
        System.out.println("=== After trimToSize() ===");
        print(v);
        v.ensureCapacity(6);
        System.out.println("=== After ensureCapacity(6) ===");
        print(v);
        v.setSize(7);
        System.out.println("=== After setSize(7) ===");
        print(v);
        v.clear();
        System.out.println("=== After clear() === ");
        print(v);
    }
```

ArrayLists: Example 3

- size: number of elements in the vector
- capacity: the total space reserved in the vector

```
    public static void print(Vector<?> v) {
        System.out.println(v);
        System.out.println("size: " + v.size());
        System.out.println("capacity: " + v.capacity());
    }
}
```

ArrayLists: Example 3

- Create a String Vector of capacity 5.

```
Vector<String> v = new Vector<>(5);
```

- Insert three elements into the Vector.

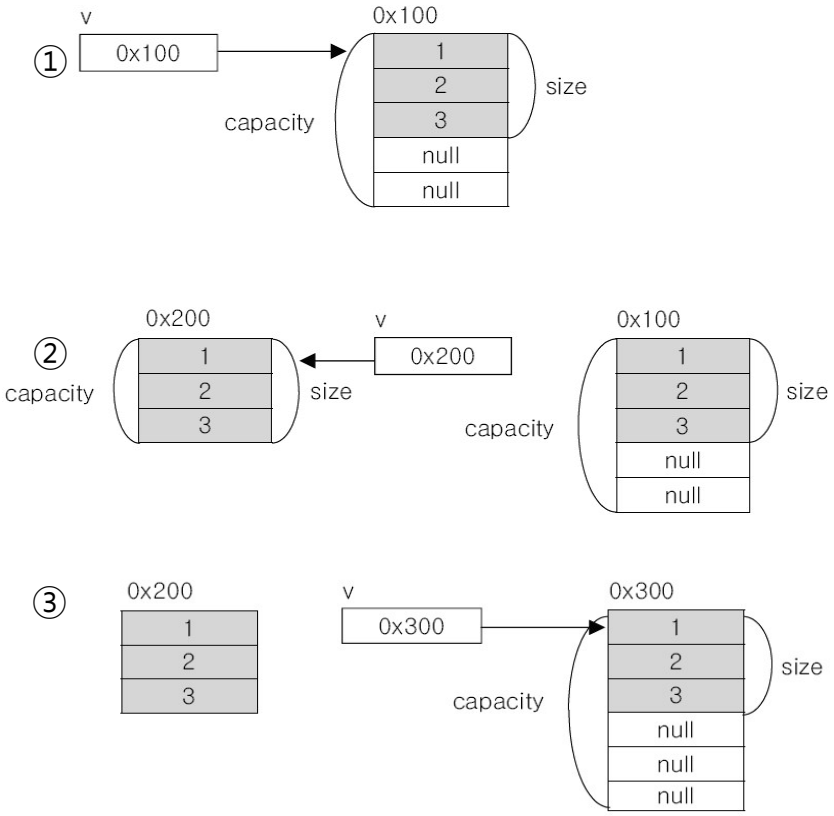
```
v.add("1");
v.add("2");
v.add("3");
```

- Remove empty space
  - creates a new instance of Vector.

```
v.trimToSize();
```

- Increase capacity to 6
  - If capacity is larger than or equal to 6, nothing happens.
  - Otherwise, creates a new instance of Vector.

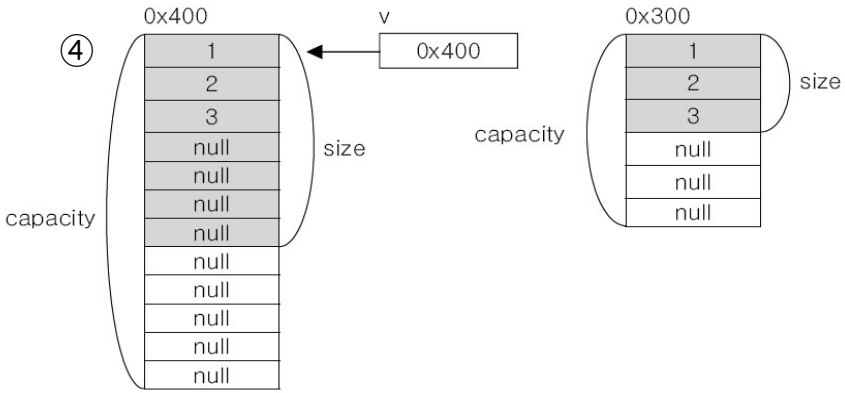
```
v.ensureCapacity(6);
```



ArrayLists: Example 3

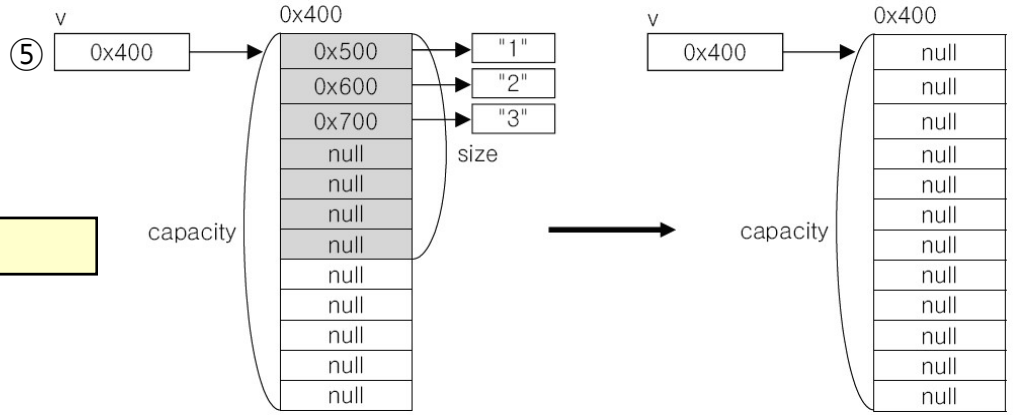
- Make the size 7.
  - If capacity is larger than or equal to 7, no new instance is created.
  - If capacity is smaller than 7, a new instance is created.

```
v.setSize(7);
```



- Remove all elements in the list.
  - size becomes zero.
  - capacity stays the same.

```
v.clear();
```



Programming Lab #16

16-01. Using ArrayList

- Write the following code and understand the results.

```
public class Ex16_01 {
     public static void main(String[] args) {
           ArrayList<Integer> list1 = new ArrayList<Integer>(10);
           list1.add(Integer.valueOf(5));
           list1.add(Integer.valueOf(4));
           list1.add(Integer.valueOf(2));
           list1.add(Integer.valueOf(0));
           list1.add(Integer.valueOf(1));
           list1.add(Integer.valueOf(3));

           ArrayList<Integer> list2 = new ArrayList<Integer>(list1.subList(1,4));
           print(list1, list2);

           Collections.sort(list1);
           Collections.sort(list2);
           print(list1, list2);

           System.out.println("list1.containsAll(list2): " + list1.containsAll(list2));
           list2.add(Integer.valueOf(11));
           list2.add(Integer.valueOf(12));
           list2.add(Integer.valueOf(13));
           print(list1, list2);
```

16-01. Using ArrayList

- Try changing the for loop so that *i* starts at 0 and ends at list2.size()-1.
  - What is the problem when you do this?

```
            list2.set(3, Integer.valueOf(21));
            print(list1, list2);

            System.out.println("list1.retainAll(list2): " + list1.retainAll(list2));
            print(list1, list2);

            for(int i=list2.size()-1; i>=0; i--) {
                if(list1.contains(list2.get(i)))
                    list2.remove(i);
            }
            print(list1, list2);
    }

    static void print(ArrayList<Integer> list1, ArrayList<Integer> list2) {
            System.out.println("list1: " + list1);
            System.out.println("list2: " + list2);
            System.out.println();
    }
}
```

16-02. Capacity of an ArrayList

- Write the following code and understand the results.

```
import java.util.ArrayList;
import java.util.List;

public class Ex16_02 {
     public static void main(String[] args) {
           final int LIMIT = 10;
           String source = "0123456789abcdefghijABCDEFGHIJ!@#$%^&*()ZZZ";
           int length = source.length();

           List<String> list = new ArrayList<String>(length/LIMIT + 10);

           for(int i=0; i<length; i+=LIMIT) {
                      if(i+LIMIT < length) list.add(source.substring(i, i+LIMIT));
                      else list.add(source.substring(i));
           }

           for(int i=0; i<list.size(); i++) {
                      System.out.println(list.get(i));
           }
     }
}
```

16-03. Size and Capacity of a Vector

- Write the following code and understand the results.

```
public class Ex16_03 {
	public static void main(String[] args) {
		Vector<String> v = new Vector<>(5);
		v.add("1");
		v.add("2");
		v.add("3");
		print(v);
		v.trimToSize();
		System.out.println("=== After trimToSize() ===");
		print(v);
		v.ensureCapacity(6);
		System.out.println("=== After ensureCapacity(6) ===");
		print(v);
		v.setSize(7);
		System.out.println("=== After setSize(7) ===");
		print(v);
		v.clear();
		System.out.println("=== After clear() === ");
		print(v);
	}
	public static void print(Vector<?> v) {
		System.out.println(v);
		System.out.println("size: " + v.size());
		System.out.println("capacity: " + v.capacity());
	}
}
```

End of Class

Instructor office: AS818A

Email: jso1@sogang.ac.kr