# Java Programming

Lists & Maps

# Lists

# The List Interface

- The List interface extends **Collection** and declares the behavior of a collection that stores a sequence of elements.
- Elements can be inserted or accessed by their position in the list, using a zero-based index.
- A list may contain duplicate elements.
- Several of the list methods will throw an UnsupportedOperationException if the collection cannot be modified, and a ClassCastException is generated when one object is incompatible with another.
- The following are the most common and used:
  - ArrayList
  - LinkedList

# Methods Defined by List

- **void add(int index, Object obj)**
  - Inserts obj into the invoking list at the index passed in index. Any pre-existing elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten.
- **boolean addAll(int index, Collection c)**
  - Inserts all elements of c into the invoking list at the index passed in index. Any pre-existing elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten. Returns true if the invoking list changes and returns false otherwise.
- **Object get(int index)**
  - Returns the object stored at the specified index within the invoking collection.
- **int indexOf(Object obj)**
  - Returns the index of the first instance of obj in the invoking list. If obj is not an element of the list, .1 is returned.
- **int lastIndexOf(Object obj)**
  - Returns the index of the last instance of obj in the invoking list. If obj is not an element of the list, .1 is returned.
- **ListIterator listIterator( )**
  - Returns an iterator to the start of the invoking list.
- **ListIterator listIterator(int index)**
  - Returns an iterator to the invoking list that begins at the specified index.
- **Object remove(int index)**
  - Removes the element at position index from the invoking list and returns the deleted element. The resulting list is compacted. That is, the indexes of subsequent elements are decremented by one
- **Object set(int index, Object obj)**
  - Assigns obj to the location specified by index within the invoking list.
- **List subList(int start, int end)**
  - Returns a list that includes elements from start to end.1 in the invoking list. Elements in the returned list are also referenced by the invoking object.

# List Example

```
import java.util.*;

public class CollectionsDemo {
    public static void main(String[] args) {
        List a1 = new ArrayList();
        a1.add("Zara");
        a1.add("Mahnaz");
        a1.add("Ayan");
        System.out.println("ArrayList Elements");
        System.out.print("\t" + a1);

        List l1 = new LinkedList();
        l1.add("Zara");
        l1.add("Mahnaz");
        l1.add("Ayan");
        System.out.println();
        System.out.println("LinkedList Elements");
        System.out.print("\t" + l1);
    }
}
```

**SCREEN**

```
ArrayList Elements
        [Zara, Mahnaz, Ayan]

LinkedList Elements
        [Zara, Mahnaz, Ayan]
```

# Maps

# The Map Interface

- A map has the form Map <K, V> where:
  - ***K***: specifies the type of keys maintained in this map.
  - ***V***: defines the type of mapped values.
- Furthermore, the **Map** interface provides a set of methods that must be implemented.
- The most famous methods are the following:
  - **clear**: Removes all the elements from the map.
  - **containsKey**: Returns true if the map contains the requested key.
  - **containsValue**: Returns true if the map contains the requested value.
  - **equals**: Compares an Object with the map for equality.
  - **get**: Retrieve the value of the requested key.
  - **keySet**: Returns a Set that contains all keys of the map.
  - **put**: Adds the requested key-value pair in the map.
  - **remove**: Removes the requested key and its value from the map, if the key exists.
  - **size**: Returns the number of key-value pairs currently in the map.

# Examples of Map

- There are many classes that implement the Java Map interface.
- The following are the most common and used:
  - Hashtable
  - HashMap
  - LinkedHashMap
  - TreeMap
  - ConcurrentHashMap
  - etc.

# Hashtable

- This class implements a hash table, which maps keys to values. Any non-null object can be used as a key or as a value.
- To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode method and the equals method.

# Hashtable Example

```
import java.util.Hashtable;
import java.util.Map;

public class HashTableExample {
    public static void main(String[] args) {
        Map vehicles = new Hashtable();

        // Add some vehicles.
        vehicles.put("BMW", 5);
        vehicles.put("Mercedes", 3);
        vehicles.put("Audi", 4);
        vehicles.put("Ford", 10);
        System.out.println("Total vehicles: " + vehicles.size());

        // Iterate over all vehicles, using the keySet method.
        for(String key: vehicles.keySet())
            System.out.println(key + " - " + vehicles.get(key));
        System.out.println();

        String searchKey = "Audi";
        if(vehicles.containsKey(searchKey))
            System.out.println("Found total " + vehicles.get(searchKey) + " "
                    + searchKey + " cars!\n");

        // Clear all values.
        vehicles.clear();

        // Equals to zero.
        System.out.println("After clear operation, size: " + vehicles.size());

        // The next statements throw a NullPointerException, if uncommented.
        //vehicles.put("Nissan", null);
        //vehicles.put(null, 6);
    }
}
```

**SCREEN**

```
Total vehicles: 4
Audi - 4
Ford - 10
BMW - 5
Mercedes - 3

Found total 4 Audi cars!

After clear operation, size: 0
```

# HashMap

- Hash table based implementation of the Map interface.
- This implementation provides all of the optional map operations, and permits null values and the null key.
  - The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls.
- This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time.

# HashMap Example

```
import java.util.HashMap;
import java.util.Map;

public class HashMapExample {
    public static void main(String[] args) {
        Map vehicles = new HashMap();

        // Add some vehicles.
        vehicles.put("BMW", 5);
        vehicles.put("Mercedes", 3);
        vehicles.put("Audi", 4);
        vehicles.put("Ford", 10);

        System.out.println("Total vehicles: " + vehicles.size());

        // Iterate over all vehicles, using the keySet method.
        for(String key: vehicles.keySet())
            System.out.println(key + " - " + vehicles.get(key));
        System.out.println();

        String searchKey = "Audi";
        if(vehicles.containsKey(searchKey))
            System.out.println("Found total " + vehicles.get(searchKey) + " "
                    + searchKey + " cars!\n");

        // Clear all values.
        vehicles.clear();

        // Equals to zero.
        System.out.println("After clear operation, size: " + vehicles.size());
    }
}
```

**SCREEN**

```
Total vehicles: 4
Audi - 4
Ford - 10
BMW - 5
Mercedes - 3

Found total 4 Audi cars!

After clear operation, size: 0
```