CSE3040 Java Language

Lecture 18: Collection Framework (3)

Dept. of Computer Engineering,
Sogang University

5. Iterator

- Java Collection Framework standardizes how to read data from a collection.
  - Use an Iterator

- Iterator is defined as an interface

```
public interface Iterator<E> {
    ...
    boolean hasNext();
    E next();
    void remove();
}
```

- Interface Collection has an instance method iterator() which returns an Iterator.
  - classes that implement interface Collection (List, Set) define implementations of method iterator().

```
public interface Collection<E> {
    ...
    public Iterator iterator();
    ...
}
```

Iterator: methods

- Methods defined in interface Iterator
  - Typically, hasNext() is called before calling next() to make sure there are more elements left to read.

Method	Description
boolean hasNext()	Returns true if the iteration has more elements.
E next()	Returns the next element in the iteration.
default void remove()	Removes from underlying collection the last element returned by this iterator.

- A typical use of Iterator

```
List<Integer> list = new ArrayList<Integer>();
Iterator<Integer> it = list.iterator();

while(it.hasNext()) {
    System.out.println(it.next());
}
```

  - Here we use type List<Integer> instead of ArrayList<Integer> for generality.
    - Since list is defined as type List<Integer>, only methods that are defined in List can be used. (Methods defined in ArrayList but not in List cannot be used.)
    - Later, if we want to use LinkedList instead of ArrayList, it is enough to change ArrayList to LinkedList.

Iterator: example 1

- The program prints elements in the list using an Iterator.

```
import java.util.*;

class Lecture {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<String>();
        list.add("1");
        list.add("2");
        list.add("3");
        list.add("4");
        list.add("5");

        Iterator<String> it = list.iterator();
        while(it.hasNext()) {
            String str = it.next();
            System.out.println(str);
        }
    }
}
```

ListIterator

- An interface that extends Iterator and has additional methods.
  - Can traverse a list in both forward and backward direction.
- Only works with classes that implements interface List.
  - ArrayList, LinkedList

- Methods of interface ListIterator<E>

Method	Description
void add(E e)	Inserts the specified element into the list.
boolean hasNext()	Returns true if this list iterator has more elements when traversing the list in the forward direction.
boolean hasPrevious()	Returns true if this list iterator has more elements when traversing the list in the reverse direction.
E next()	Returns the next element in the list and advances the cursor position.
int nextIndex()	Returns the index of the element that would be returned by a subsequent call to next().
E previous()	Returns the previous element in the list and moves the cursor position backwards.
int previousIndex()	Returns the index of the element that would be returned by a subsequent call to previous().
void remove()	Removes from the list the last element that was returned by next() or previous().
void set(E e)	Replaces the last element returned by next() or previous() with the specified element.

Iterator: example 2

- Using a ListIterator, the program prints the elements in normal and reverse order.

```
import java.util.*;

class Lecture {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<String>();
        list.add("1");
        list.add("2");
        list.add("3");
        list.add("4");
        list.add("5");

        ListIterator<String> it = list.listIterator();
        while(it.hasNext()) {
            System.out.print(it.next());
        }
        System.out.println();
        while(it.hasPrevious()) {
            System.out.print(it.previous());
        }
        System.out.println();
    }
}
```

6. Arrays

- class Arrays is a member of Java Collections Framework, which provides useful methods for manipulating arrays.

- copyOf(), copyOfRange()
  - copyOf() generates a new array by copying elements from an original array.
  - copyOfRange() is similar to copyOf() but is possible to specify range of elements that should be copied from the original array.

```
int[] arr = {0, 1, 2, 3, 4};
int[] arr2 = Arrays.copyOf(arr, arr.length);    // arr2 is [0, 1, 2, 3, 4]
int[] arr3 = Arrays.copyOf(arr, 3);             // arr3 is [0, 1, 2]
int[] arr4 = Arrays.copyOf(arr, 7);             // arr4 is [0, 1, 2, 3, 4, 0, 0]
int[] arr5 = Arrays.copyOfRange(arr, 2, 4);     // arr5 is [2, 3] (index 4 is excluded)
int[] arr6 = Arrays.copyOfRange(arr, 0, 7);     // arr6 is [0, 1, 2, 3, 4, 0, 0]
```

Arrays: methods

- fill(), setAll()
  - fill() sets elements of an array with a specified value.
  - setAll() sets elements of an array using the provided generator function given as an argument.

```
int[] arr = new int[5];
Arrays.fill(arr, 9);                                    // arr is [9, 9, 9, 9, 9]
Arrays.setAll(arr, () -> (int)(Math.random()*5)+1);     // arr is [1, 5, 2, 1, 1]
                                                        // (random numbers between 1 and 5)
```

  - () -> (int)(Math.random()*5+1) is a lambda expression which is basically a brief way of defining a function.

Arrays: methods

- sort(), binarySearch()
  - sort(): sorts the specified array into ascending order.
  - binarySearch(): searches the specified array for the specified value using the binary search algorithm
    - the array must be sorted in order to get a correct result from binarySearch()

```
int[] arr = {3, 2, 0, 1, 4};
int idx = Arrays.binarySearch(arr, 2);    // idx = -5 (wrong result)

Arrays.sort(arr);
System.out.println(Arrays.toString(arr));
int idx = Arrays.binarySearch(arr, 2);    // idx = 2 (correct result)
```

Arrays: methods

- toString()
  - puts all elements to a string.
  - for multi-dimensional arrays, deepToString() is used.

```
int[] arr = {0, 1, 2, 3, 4};
int[][] arr2D = {{11, 12}, {21, 22}};

System.out.println(Arrays.toString(arr));         // [0, 1, 2, 3, 4]
System.out.println(Arrays.deepToString(arr2D));   // [[11, 12], [21, 22]]
```

- equals()
  - compares two arrays and returns true if all elements are equal
  - deepEquals() used for multi-dimensional arrays

```
String[][] str2D = new String[][]{{"aaa","bbb"},{"AAA","BBB"}};
String[][] str2D2 = new String[][] {{"aaa","bbb"},{"AAA","BBB"}};

System.out.println(Arrays.equals(str2D, str2D2));        // false
System.out.println(Arrays.deepEquals(str2D, str2D2));    // true
```

Arrays: example

- Using class Arrays to manipulate arrays

```
public static void main(String[] args) {
    int[] arr = {0, 1, 2, 3, 4};
    int[][] arr2D = {{11, 12, 13}, {21, 22, 23}};
    System.out.println("arr="+Arrays.toString(arr));
    System.out.println("arr2D="+Arrays.deepToString(arr2D));

    int[] arr2 = Arrays.copyOf(arr, arr.length);
    int[] arr3 = Arrays.copyOf(arr, 3);
    int[] arr4 = Arrays.copyOf(arr, 7);
    int[] arr5 = Arrays.copyOfRange(arr, 2, 4);
    int[] arr6 = Arrays.copyOfRange(arr, 0, 7);

    System.out.println("arr2="+Arrays.toString(arr2));
    System.out.println("arr3="+Arrays.toString(arr3));
    System.out.println("arr4="+Arrays.toString(arr4));
    System.out.println("arr5="+Arrays.toString(arr5));
    System.out.println("arr6="+Arrays.toString(arr6));
```

Arrays: example

- Using class Arrays to manipulate arrays (cont.)

```
int[] arr7 = new int[5];
Arrays.fill(arr7, 9);
System.out.println("arr7="+Arrays.toString(arr7));

Arrays.setAll(arr7, i -> (int)(Math.random()*6)+1);
System.out.println("arr7="+Arrays.toString(arr7));

for(int i : arr7) {
    char[] graph = new char[i];
    Arrays.fill(graph, '*');
    System.out.println(new String(graph)+i);
}

String[][] str2D = new String[][]{{"aaa","bbb"},{"AAA","BBB"}};
String[][] str2D2 = new String[][]{{"aaa","bbb"},{"AAA","BBB"}};
System.out.println(Arrays.equals(str2D, str2D2));
System.out.println(Arrays.deepEquals(str2D, str2D2));
```

Arrays: example

- Using class Arrays to manipulate arrays (cont.)

```
    char[] chArr = {'A', 'D', 'C', 'B', 'E'};

    System.out.println("chArr="+Arrays.toString(chArr));
    System.out.println("index of B="+Arrays.binarySearch(chArr, 'B'));
    System.out.println("=== After sorting ===");
    Arrays.sort(chArr);
    System.out.println("chArr="+Arrays.toString(chArr));
    System.out.println("index of B="+Arrays.binarySearch(chArr, 'B'));
}
```

7. Comparable and Comparator

- Class Arrays has a method called sort.
- If the elements are of primitive types such as int and float, the sort method knows how to compare two elements so that the elements are sorted in an ascending order.
- However, if the elements are objects, how to compare two objects is not defined.
- Sorting is possible only when the elements are comparable.

- The method sort only works when the elements are of type which implements interface Comparable<T>.
  - For example, class Integer implements interface Comparable<T>.

Comparable: example

- class Integer
  - implements Comparable<Integer>
  - the class implements method compareTo defined in the interface Comparable.
  - when Arrays.sort() is called, elements are sorted by the order defined by method compareTo.

```
public final class Integer extends Number implements Comparable<Integer> {
    ...
    public int compareTo(Integer anotherInteger) {
        int thisVal = this.value;
        int anotherVal = anotherInteger.value;
        return (thisVal < anotherVal ? -1 : (thisVal == anotherVal ? 0 : 1));
    }
    ...
}
```

Comparable: example

- If we define a class that implements interface Comparable, we can use Collections.sort to sort the elements.

```
class Employee implements Comparable<Employee> {
    private String name;
    private int salary;
    public Employee(String name, int salary) { this.name = name; this.salary = salary; }
    public String getName() { return this.name; }
    public int getSalary() { return this.salary; }

    public int compareTo(Employee e) {
        if(this.salary > e.getSalary()) return -1;
        if(this.salary < e.getSalary()) return 1;
        return this.name.compareTo(e.getName());
    }

    public String toString() { return this.name + " " + this.salary; }
}
```

Comparable: example

- If we define a class that implements interface Comparable, we can use Collections.sort to sort the elements. (cont.)

```
public class Lecture {
    public static void main(String[] args) {
        ArrayList<Employee> emplist = new ArrayList<Employee>();
        emplist.add(new Employee("Peter", 50000));
        emplist.add(new Employee("John", 100000));
        emplist.add(new Employee("Robert", 100000));

        System.out.println(emplist);
        Collections.sort(emplist);
        System.out.println(emplist);
     }
}
```

Comparator

- If you want to sort elements in a different way, you can define a custom order using Comparator.
- For example, if you sort an array of Strings using Arrays.sort, the elements will be sorted in ascending order.

```
public class Lecture {
    public static void main(String[] args) {
        String[] strArr = {"lion", "DOG", "TIGER", "cat"};
        System.out.println("strArr = " + Arrays.toString(strArr));

        Arrays.sort(strArr);
        System.out.println("strArr = " + Arrays.toString(strArr));
    }
}
```

- What if you want to sort the elements in descending order?

Comparator

- You can define a class which implements interface Comparator.
- In the class, implement method compare, which returns results according to the order you like.

```
class Descending<T> implements Comparator<T> {
    public int compare(T o1, T o2) {
        if(o1 instanceof Comparable && o2 instanceof Comparable) {
            Comparable c1 = (Comparable)o1;
            Comparable c2 = (Comparable)o2;
            return c1.compareTo(c2) * -1;    // reverse order
        }
        return -1;    // undefined
    }
}
```

```
String[] strArr = {"lion", "DOG", "TIGER", "cat"};
Arrays.sort(strArr, new Descending());
System.out.println("strArr = " + Arrays.toString(strArr));
```

Comparator

- class String has a static final variable CASE_INSENSITIVE_ORDER, which is an instance of an inner class that implements Comparator<String>.

```
public static final Comparator<String> CASE_INSENSITIVE_ORDER = new CaseInsensitiveComparator();

private static class CaseInsensitiveComparator implements Comparator<String>, java.io.Serializable {
    private static final long serialVersionUID = 8575799808933029326L;
    public int compare(String s1, String s2) {
        byte v1[] = s1.value;
        byte v2[] = s2.value;
        if(s1.coder() == s2.coder()) {
            return s1.isLatin1() ? StringLatin1.compareToCI(v1, v2) : StringUTF16.compareToCI(v1, v2);
        }
        return s1.isLatin() ? StringLatin1.compareToCI_UTF16(v1, v2) : StringUTF16.compareToCI_Latin1(v1, v2);
    }
    private Object readResolve() { return CASE_INSENSITIVE_ORDER; }
}
```

```
String[] strArr = {"lion", "DOG", "TIGER", "cat"};
Arrays.sort(strArr, String.CASE_INSENSITIVE_ORDER);
System.out.println("strArr = " + Arrays.toString(strArr));
```

Programming Lab #18

18-01. Iterator and ListIterator

- Write the following code and understand the results.
- Modify the code to print the elements in the reverse order using ListIterator.

```
class Ex18_01 {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<String>();
        list.add("1");
        list.add("2");
        list.add("3");
        list.add("4");
        list.add("5");

        Iterator<String> it = list.iterator();
        while(it.hasNext()) {
            String str = it.next();
            System.out.println(str);
        }
    }
}
```

18-02. Using class Arrays

- Write the following code and understand the results.

```
public static void main(String[] args) {
    int[] arr = {0, 1, 2, 3, 4};
    int[][] arr2D = {{11, 12, 13}, {21, 22, 23}};
    System.out.println("arr="+Arrays.toString(arr));
    System.out.println("arr2D="+Arrays.deepToString(arr2D));

    int[] arr2 = Arrays.copyOf(arr, arr.length);
    int[] arr3 = Arrays.copyOf(arr, 3);
    int[] arr4 = Arrays.copyOf(arr, 7);
    int[] arr5 = Arrays.copyOfRange(arr, 2, 4);
    int[] arr6 = Arrays.copyOfRange(arr, 0, 7);

    System.out.println("arr2="+Arrays.toString(arr2));
    System.out.println("arr3="+Arrays.toString(arr3));
    System.out.println("arr4="+Arrays.toString(arr4));
    System.out.println("arr5="+Arrays.toString(arr5));
    System.out.println("arr6="+Arrays.toString(arr6));

    int[] arr7 = new int[5];
    Arrays.fill(arr7, 9);
    System.out.println("arr7="+Arrays.toString(arr7));

    Arrays.setAll(arr7, i -> (int)(Math.random()*6)+1);
    System.out.println("arr7="+Arrays.toString(arr7));
```

18-02. Using class Arrays

```
    for(int i : arr7) {
        char[] graph = new char[i];
        Arrays.fill(graph, '*');
        System.out.println(new String(graph)+i);
    }

    String[][] str2D = new String[][]{{"aaa","bbb"},{"AAA","BBB"}};
    String[][] str2D2 = new String[][]{{"aaa","bbb"},{"AAA","BBB"}};
    System.out.println(Arrays.equals(str2D, str2D2));
    System.out.println(Arrays.deepEquals(str2D, str2D2));

    char[] chArr = {'A', 'D', 'C', 'B', 'E'};

    System.out.println("chArr="+Arrays.toString(chArr));
    System.out.println("index of B="+Arrays.binarySearch(chArr, 'B'));
    System.out.println("=== After sorting ===");
    Arrays.sort(chArr);
    System.out.println("chArr="+Arrays.toString(chArr));
    System.out.println("index of B="+Arrays.binarySearch(chArr, 'B'));
}
```

18-03. Implementing interface Comparable

- Modify the code so that it works properly.
- Employees should be sorted by descending order of their salaries. If two employees have the same salary, they should be listed in the alphetical order of their names.

```
class Employee {
    private String name;
    private int salary;
    public Employee(String name, int salary) { this.name = name; this.salary = salary; }
    public String getName() { return this.name; }
    public int getSalary() { return this.salary; }

    public String toString() { return this.name + " " + this.salary; }
}
public class Ex18_03 {
    public static void main(String[] args) {
        ArrayList<Employee> emplist = new ArrayList<Employee>();
        emplist.add(new Employee("Peter", 50000));
        emplist.add(new Employee("John", 100000));
        emplist.add(new Employee("Robert", 100000));

        System.out.println(emplist);
        Collections.sort(emplist);
        System.out.println(emplist);
     }
}
```

18-04. Comparator

- Write the following code and understand the results.
- Try modifying the code so that the warning disappears.

```
class Descending<T> implements Comparator<T> {
    public int compare(T o1, T o2) {
        if(o1 instanceof Comparable && o2 instanceof Comparable) {
            Comparable c1 = (Comparable)o1;
            Comparable c2 = (Comparable)o2;
            return c1.compareTo(c2) * -1;    // reverse order
        }
        return -1;    // undefined
    }
}

public class Ex18_04 {
    public static void main(String[] args) {
        String[] strArr = {"lion", "DOG", "TIGER", "cat"};
        System.out.println("strArr = " + Arrays.toString(strArr));
        Arrays.sort(strArr);
        System.out.println("strArr = " + Arrays.toString(strArr));
        Arrays.sort(strArr, new Descending());
        System.out.println("strArr = " + Arrays.toString(strArr));
    }
}
```

End of Class

Instructor office: AS818A

Email: jso1@sogang.ac.kr