Structuring Data with Java

By Dr. Subrat Kumar Nayak

Associate Professor

Department of CSE

ITER, SOADU

Structuring Data with Java
(How Shall We Iterate Thee?)

Using Iterator object

- Iterator interface : Iterator is an **interface** that iterates the elements. It is used to traverse the list and modify the elements. Iterator interface has three methods which are mentioned below.

✓ **public boolean hasNext()** - This method returns true if the iterator has more elements.

✓ **public object next()** -It returns the element and moves the cursor pointer to the next element.

✓ **public void remove()** -This method removes the last elements returned by the iterator.

```
import java.util.Iterator;

public static void main(String[]arg)

{

     ArrayList<Integer> as=new ArrayList<Integer>();

     as.add(10);

     as.add(20);

     as.add(30);

     Iterator<Integer> it=as.iterator();

     while(it.hasNext())

     {

          System.out.println(it.next());

     }

}
```

```
void remove()
while(it.hasNext())
{
     if(it.next().equals(20))
        it.remove();
}
```

Structuring Data with Java

(How Shall We Iterate Thee?)

- **Using foreach loop**

```
import java.util.ArrayList;
public class TestArrayList {
public static void main(String[]arg)
{
    ArrayList<Integer> as=new ArrayList<Integer>();
    as.add(10);
    as.add(20);
    as.add(30);
    for(Integer i:as)
    {
        System.out.println(i);
    }
}}
```

- **Using three part for loop**
- **Using while loop**

Structuring Data with Java
(Eschewing Duplicates with a Set)

- Set is an interface which extends Collection. It is an unordered collection of objects in which **duplicate values cannot be stored**.
- Basically, Set is implemented by HashSet, LinkedHashSet or TreeSet (sorted representation).
- Set has various methods to add, remove clear, size, etc to enhance the usage of this interface.
- If you add the "same" item (as considered by its equals() method) twice or more, it will only be present once in the set. For this reason, the index-based methods such as add(int, Object) and get(int), are missing from the Set implementation.

Structuring Data with Java
(Eschewing Duplicates with a Set)

HashSet

- Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.
- The important points about Java HashSet class are:

- HashSet stores the elements by using a mechanism called **hashing.**

Hashing is an important Data Structure which is designed to use a special function called the Hash function which is used to map a given value with a particular key for faster access of elements.

- HashSet contains unique elements only.
- HashSet allows null value.
- HashSet class is non synchronized.
- HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
- HashSet is the best approach for search operations.

Structuring Data with Java
(Eschewing Duplicates with a Set)

Constructors

- `HashSet()`

It is used to construct a default HashSet

Example

```
Set<Integer> x=new HashSet<Integer>();
x.add(5);
x.add(10);
x.add(20);
x.add(10);
Iterator<Integer> it=x.iterator();
while(it.hasNext())
{
    if(it.next().equals(20))
    it.remove();
}
System.out.println(x);
System.out.println(x.remove(25));
System.out.println(x.size());
System.out.println(x.isEmpty());
System.out.println(x.contains(10));
x.clear();
System.out.println(x);
```

Output:

```
[1, 4, 10]
false
3
false
true
[]
```

Structuring Data with Java
(Using Iterators or Enumerations for Data- Independent Access)

```
Iterator it = l.iterator();
// Process the data structure using an iterator.
// This part of the code does not know or care
// if the data is an an array, a List, a Vector, or whatever.
while (it.hasNext()) {
    Object o = it.next();
    System.out.println("Element " + i++ + " = " + o);
}
```

Enumeration

- The Enumeration interface defines the methods by which you can enumerate (obtain one at a time) the elements in a collection of objects.
- This legacy interface has been superceded by Iterator. Although not deprecated, Enumeration is considered obsolete for new code. It is used by several methods defined by the legacy classes such as **Vector and Properties**.

Structuring Data with Java
(Structuring Data in a Linked List)

- Java LinkedList class uses a **doubly linked list** to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.
- The important points about Java LinkedList are:

- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.
- Java LinkedList class is non synchronized.
- In Java LinkedList class, manipulation is fast because no shifting needs to occur.
- Java LinkedList class can be used as a list, stack or queue.

Constructors:

- **LinkedList()**

It is used to construct an empty list.

Structuring Data with Java

(Structuring Data in a Linked List)

Methods

`add()`

- `Syntax: boolean add(E e)`

It is used to append the specified element to the end of a list

- `Syntax: void add(int index, E element)`

It is used to insert the specified element at the **specified position** index in a list.

`addAll()`

- `Syntax: boolean addAll(Collection<? extends E> c)`

It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.

- `Syntax: boolean addAll(int index, Collection<? extends E > c)`

It is used to append all the elements in the specified collection, starting at the specified position of the list

`addFirst()`

- `Syntax: void addFirst(E e)`

It is used to insert the given element at the beginning of a list.

Structuring Data with Java
(Structuring Data in a Linked List)

addLast()

- `Syntax: void addLast(E e)`

It is used to append the given element to the end of a list.

getFirst()

- `Syntax: E getFirst()`

It is used to return the first element in a list.

getLast()

- `Syntax: E getLast()`

It is used to return the last element in a list.

get()

- `Syntax: E get(int index)`

It is used to return the element at the specified position in a list.

indexOf()

- `Syntax: int indexOf(Object o)`

It is used to return the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element.

Structuring Data with Java
(Structuring Data in a Linked List)

peek()

- Syntax: `E peek()`

It retrieves the first element of a list

peekFirst()

- Syntax: `E peekFirst()`

It retrieves the first element of a list or returns null if a list is empty.

peekLast()

- Syntax: `E peekLast()`

It retrieves the last element of a list or returns null if a list is empty.

remove()

- Syntax: `E remove()`

It is used to retrieve and removes the first element of a list

- Syntax: `E remove(int index)`

It is used to remove the element at the specified position in a list.

- Syntax: `boolean remove(Object o)`

It is used to remove the first occurrence of the specified element in a list

Structuring Data with Java
(Structuring Data in a Linked List)

removeFirst()

- `Syntax: E removeFirst()`

It removes and returns the first element from a list.

removeLast()

- `Syntax: E removeLast()`

It removes and returns the last element from a list.

set()

- `Syntax: E set(int index, E element)`

It replaces the element at the specified position in a list with the specified element.

push()

- `Syntax: void push(E e)`

It pushes an element onto the stack represented by a list.

pop()

- `Syntax: E pop()`
- It pops an element from the stack represented by a list.

size()

- `Syntax: int size()`

It is used to return the number of elements in a list.

Structuring Data with Java
(Structuring Data in a Linked List)

`listIterator()`

- `Syntax: ListIterator<E> listIterator(int index)`

It is used to return a list-iterator of the elements in proper sequence, starting at the specified position in the list.

`descendingIterator()`

- `Syntax: Iterator<E> descendingIterator()`

It is used to return an iterator over the elements in a **deque** in reverse sequential order.

End of Session