CS205 Object Oriented Programming in Java

Module 4 - **Advanced features of Java** (Part 5)

Prepared by

Renetha J.B.

AP

Dept.of CSE,

Lourdes Matha College of Science and Technology

Topics

- ☑ Java Library
  - ❑ **Collections framework**
    - ❑ Accessing Collections via an Iterator

Accessing Collections via an Iterator

- To **cycle through the elements in a collection** (e.g. display each element, sum of elements etc.), we can use *iterator, which is an object* that **implements** either
  - **Iterator** or
  - **ListIterator**

Accessing Collections via an Iterator(contd.)

- **Iterator** enables you to
  - cycle through a collection
  - obtaining or removing elements.
- **ListIterator extends Iterator** to allow
  - **bidirectional traversal** of a list,
  - the **modification** of elements

Accessing Collections via an Iterator(contd.)

- **Iterator** and **ListIterator** are **generic interfaces** which are declared as :

```
interface Iterator<E>
```

```
interface ListIterator<E>
```

  - Here, **E** specifies the type of objects being iterated

Accessing Collections via an Iterator(contd.)

The Methods Defined by **Iterator**

boolean **hasNext( )**

- Returns **true** if there are **more elements**. Otherwise, returns false.

E **next( )**

- Returns the **next element**.
- Throws **NoSuchElementException** if there is not a next element.

void **remove( )**

- Removes the **current element**.
- **Throws IllegalStateException** if an attempt is made to call remove( ) that is not preceded by a call to next( ).

Accessing Collections via an Iterator(contd.)

Method Defined by **ListIterator**	
void add(E obj)	• **Inserts obj into the list in front** of the element that will be returned by the next call to next( ).
boolean hasNext( )	• Returns **true** if there is a **next element**. Otherwise, returns false.
boolean hasPrevious( )	• Returns **true** if there is **a previous element**. Otherwise, returns false.
E next( )	• **Returns the next element.** NoSuchElementException is thrown if there is not a next element.
int nextIndex( )	• Returns the **index of the next element**. If there is not a next element, returns the size of the list.
E previous( )	• Returns the **previous element**. **NoSuchElementException** is thrown if there is not a previous element.
int previousIndex( )	• Returns the **index of the previous element**. If there is not a previous element, returns −1.
void remove( )	• Removes the **current element** from the list. An **IllegalStateException** is thrown if remove( ) is called before next( ) or previous( ) is invoked.
void set(E obj)	• **Assigns obj to the current element.** This is the element last returned by a call to either next( ) or previous( ).

Exceptions in methods

- Exceptions in the Methods Defined by **Iterator**
  - **NoSuchElementException**
  - **IllegalStateException**
- Exceptions in the Methods Defined by **ListIterator**
  - **NoSuchElementException**
  - **IllegalStateException**
  - **UnsupportedOperationException**

Using an Iterator

- Each of the **collection classes** provides an **iterator( ) method that returns an iterator to the start** of the collection.
  - By using this iterator object, we can access each element in the collection, one element at a time.
- To use an iterator to cycle through the contents of a collection,
  - 1. **Obtain an iterator** to the start of the collection by calling the collection's **iterator( )** method.
  - 2. Set up a loop that makes a call to **hasNext( ).**
    - Have the loop iterate as long as hasNext( ) returns true.
  - 3. Within the loop**, obtain each element** by calling **next( ).**

```
import java.util.*;
class IteratorDemo {
public static void main(String args[]) {
ArrayList<String> al = new ArrayList<String>();
al.add("C");
al.add("A");
al.add("E");
al.add("B");
al.add("D");
al.add("F");
System.out.print("Original contents of al: ");
Iterator<String> itr = al.iterator();
while(itr.hasNext())
{
String element = itr.next();
System.out.print(element + " ");
}
System.out.println();
ListIterator<String> litr = al.listIterator();
while(litr.hasNext())
 { String element = litr.next();
litr.set(element + "+");  }
System.out.print("Modified contents of al: ");
itr = al.iterator();
while(itr.hasNext()) {
String element = itr.next();
System.out.print(element + " "); }
System.out.println();
System.out.print("Modified list backwards: ");
while(litr.hasPrevious()) {
String element = litr.previous();
System.out.print(element + " ");
} System.out.println();  }   }
```

```
Original contents of al: C A E B D F
Modified contents of al: C+ A+ E+ B+ D+ F+
Modified list backwards: F+ D+ B+ E+ A+ C+
```

The For-Each Alternative to Iterators

- The **for** loop is substantially **shorter** and **simpler** to use than the iterator based approach.
- Bur **for** loop can only be used to cycle through a collection in the forward direction, and we **can't modify** the contents of the collection.
- If we don't want to modify the contents of a collection or obtaining elements in reverse order, then the **for-each version** of the for loop is often a more **convenient alternative** to cycling through a collection than is using an iterator.

for each loop used to find sum of elements in collection

```
import java.util.*;
class ForEachDemo {
public static void main(String args[]) {
ArrayList<Integer> vals = new ArrayList<Integer>();
vals.add(1);
vals.add(2);
vals.add(3);
vals.add(4);
vals.add(5);
System.out.print("Original contents of vals: ");
for(int v : vals)
      System.out.print(v + " ");
System.out.println();

int sum = 0;
for(int v : vals)
      sum += v;
System.out.println("Sum of values: " + sum);
}
}
```

Reference

- Herbert Schildt, Java: **The Complete Reference, 8/e, Tata McGraw Hill, 2011**.