CSCI 2120:
Software Design & Development II

UNIT 2: Collections Framework & Generics

Iterate ArrayList

Overview

1. Introduction
2. Iterate ArrayList using for loop
3. Iterating ArrayList using Enhanced for loop
4. How Enhanced for loop works?
    a. Example 1: For-Loop & Enhanced For-Loop
5. Iterating ArrayList using While loop
    a. Example 2: While-Loop
6. How to iterate ArrayList using Iterator
    a. Example 3: Removing Element Using Iteration
    b. Example 4: Adding Element Using Iteration
7. Iterate ArrayList using ListIterator
    a. Example 5: ArrayList Using ListIterator
    b. Example 6: Add Remove Test
    c. Example 7: Specific Element Test

Introduction

In this lecture, we will learn **how to iterate ArrayList in Java**. Basically, there are five ways by which we can iterate over elements of an ArrayList.

That is, Java Collections framework provides five ways to retrieve elements from a Collection object. They are as follows:

1. Using for loop
2. Using Enhanced for loop or Advanced for loop
3. Using while Loop
4. By using Iterator
5. By ListIterator

Iterate ArrayList using for loop

The general syntax for simple for loop is as follows:

```
for (initialization; condition; step) {
   //loop body
}
```

Where initialization is a variable declaration, condition is an expression of type boolean, step is an increment/decrement, and body is a statement.

The loop is executed as follows:

1. The initialization is executed.

2. The condition is evaluated if it is true, the body is executed. If it is false, the loop terminates.

Iterate ArrayList using for loop

For example:

```
// Get size of the ArrayList.
int size = al.size();

// Iterate list of objects.
System.out.println("for Loop");

for(int i = 0; i < size; i++) {
    // Call get() method to return or get the elements on the specified index after iterating.
     String getElement = al.get(i);
     System.out.println(getElement);
}
```

Iterating ArrayList using Enhanced for loop

This Enhanced for loop is introduced in Java 1.5 version. This Enhanced for loop is mostly used in the industry. We can easily use for-each loop to retrieve elements of a collection object.

There are three benefits of using for-each loop to iterate list of objects. They are as:

» In Enhanced for loop, we don’t need to increment or decrement.

» No size calculation.

» We must use the generic concept to iterate elements of a list.

The general syntax for Enhanced for loop is as follows:

```
for(data_type element : Object reference variable ) {
   // loop body
}
```

How Enhanced for loop works?

Enhance for loop works in two steps. They are as follows:

1. It iterates over each element in the collection or array.
2. It stores each element in a variable and executes the body.

Let's take some examples based on it.

```
1. ArrayList<String> al = new ArrayList<String>();
   for(String element: al) {
     System.out.println(element);
   }

2. ArrayList<Employee> al = new ArrayList<Employee>();
   for(Employee emp: al ) {
     System.out.println(emp.name);
     System.out.println(emp.age);
   }
```

Let's take an example program where we will implement the concepts of simple for loop and enhanced for loop to understand better.

Example 1: For-Loop & Enhanced For-Loop

```
import java.util.ArrayList;
public class IterateArrayList1 {
   public static void main(String[] args) {
       ArrayList<String> al = new ArrayList<String>();     // Create ArrayList of type String. In the list, we can add only String type of elements.

       al.add("A");    // Adding element at index 0.
       al.add("B");    // Adding element at index 1.
       al.add("C");    // Adding element at index 2.
       al.add("D");    // Adding element at index 3.
       al.add("E");    // Adding element at index 4.
       System.out.println(al);                           // Displaying original elements of the ArrayList.

       System.out.println("Using for loop");              // Iterating ArrayList using for loop and call size() method to get the size of elements.
       int elementSize = al.size();
       System.out.println("Size: " +elementSize);

       for(int i = 0; i < al.size(); i++) {
           String getElement = al.get(i);                 // Call get() method to return elements on specified index after iterating.
           System.out.println(getElement);
       }
       al.set(2, "G");                                    // It will replace current element at position 2 with element G.
       al.set(3, null);                                   // adding null element at position 3.

       System.out.println("Using Enhance for loop");      // Iterating ArrayList using Enhance for loop.
       for(String element:al) {
           System.out.println(element);
       }
   }
}
```

Example 1: For-Loop & Enhanced For-Loop

Output:

```
[A, B, C, D, E]
Using for loop
Size: 5
A
B
C
D
E
Using Enhance for loop
A
B
G
null
E
```

Key point:

Iteration over elements in the ArrayList using Enhanced for loop is fail-fast. That means that we can not add or remove an element in the ArrayList during Iteration otherwise, it will throw **ConcurrentModificationException**.

Iterating ArrayList using While loop

The general syntax for while loop is as follows:

```
Initialization;
while(condition) {
    statement1;
    statement2;
    increment/decrement;
}
```

Let’s take an example based on it.

```
System.out.println("Iteration of ArrayList using while loop");
int i = 0; // initialization.
while( ar.size() > i) {
    System.out.println(i);
     i++; // Increment.
}
```

Let’s create a program where we will iterate elements of ArrayList using while loop concept.

Example 2: While-Loop

```
import java.util.ArrayList;
public class IterateArrayList2 {
   public static void main(String[] args) {
       ArrayList<Integer> al = new ArrayList<Integer>();
       al.add(20);
       al.add(25);
       al.add(null);
       al.add(30);
       al.add(25);
       System.out.println(al);

       // Iteration of ArrayList using while loop.
       System.out.println("Iteration using while loop");
       int i = 0;
       while(al.size() > i) {
           Integer itr = al.get(i);
           System.out.println(itr);
           i++;
       }
   }
}
```

Example 2: While-Loop

Output:

```
[20, 25, null, 30, 25]
Iteration using while loop
20
25
null
30
25
```

How to iterate ArrayList using Iterator

The iterator() method is used to iterate over elements of ArrayList in Java. It is useful when we want to remove the last element during iteration. The Iterator interface defines three methods. They are as follows:

1. **hasNext():** This method returns true if the iteration has more elements in the forward direction.

2. **next():** It returns the next element in the iteration. If the iteration has no more elements then it will throw “NoSuchElementException”.

3. **remove():** The remove() method removes the last element returned by the iterator. This method must be called after calling next() method otherwise, it will throw “IllegalStateException”.

For more detail with diagram, go to this lecture: Iterator in Java

Let’s make an example program where we iterate elements of ArrayList by using iterator. We will also remove the last element returned by the iterator.

Example 3: Removing Element Using Iteration

```
import java.util.ArrayList;
import java.util.Iterator;

public class IterateArrayList3 {

   public static void main(String[] args) {
       ArrayList<String> al = new ArrayList<String>();
       al.add("Apple");
       al.add("Mango");
       al.add("Banana");
       al.add("Guava");
       al.add("Pineapple");
       System.out.println(al); // It will print all elements at a time.

       System.out.println("Iteration using iterator concept.");
       // Create an object of Iterator by calling iterator() method using reference variable al.
       // At the beginning, itr(cursor) will point to index just before the first element in al.
       Iterator<String> itr = al.iterator();

       // Checking the next element availability using reference variable itr.
       while(itr.hasNext()) {
           // Moving cursor to next element using reference variable itr.
           String str = itr.next();
           System.out.println(str);

           // Removing the pineapple element.
           if(str.equals("Pineapple")) {
               itr.remove();
               System.out.println("After removing pineapple element");
               System.out.println(al);
           }
       }
   }
}
```

Example 3: Removing Element Using Iteration

Output:

```
[Apple, Mango, Banana, Guava, Pineapple]
Iteration using iterator concept.
Apple
Mango
Banana
Guava
Pineapple
After removing pineapple element
[Apple, Mango, Banana, Guava]
```

Key point:

`iterator` returned by `ArrayList` is a fail-fast. That means that if we add an element or remove an element in the ArrayList during the Iteration then it will throw "`ConcurrentModificationException`".

This is because the loop internally creates a fail-fast iterator which throws an exception whenever any structural modification in the underlying data structure.

Example 4: Adding Element Using Iteration

Let's make a program where we will understand the concept of fail-fast during the iteration of elements.

```
import java.util.ArrayList;
import java.util.Iterator;
public class IterateArrayList4 {

   public static void main(String[] args) {
       ArrayList<String> al = new ArrayList<>();
       al.add("Lion");
       al.add("Tiger");
       al.add("Elephant");
       al.add("Bear");
       Iterator<String> itr = al.iterator();

       while(itr.hasNext()) {
           System.out.println(itr.next());
           // Adding element during iteration. Since the return type of add() method is boolean.
           //Therefore, we will store it using variable b with data type boolean.
           boolean b = al.add("Leopard"); // Compile time error. It will throw ConcurrentModificationException.
           System.out.println(b);
       }
   }
}
```

Example 4: Adding Element Using Iteration

Output:

```
Lion
true
Exception in thread "main" java.util.ConcurrentModificationException Create breakpoint
    at java.base/java.util.ArrayList$Itr.checkForComodification(ArrayList.java:1013)
    at java.base/java.util.ArrayList$Itr.next(ArrayList.java:967)
    at IterateArrayList4.main(IterateArrayList4.java:12)
```

Iterate ArrayList using ListIterator

In Java programming, ListIterator is used to iterate the elements of array list in both forward as well as backward directions. Using ListIterator, we can also start the iteration from a specific element in the ArrayList.

We can perform different kinds of operations such as read, remove, replacement (current object), and the addition of the new objects.

ListIterator interface defines 9 methods in Java. Since ListIterator is the child interface of Iterator. Therefore, all the methods of Iterator are available by default to ListIterator. ListIterator interface has total of 9 methods.

For more detail with diagram, go to ListIterator tutorial: ListIterator in Java

Let’s make a example program where we will iterate elements of ArrayList in both forward and backward directions using ListIterator.

Example 5: ArrayList Using ListIterator

```
import java.util.ArrayList;
import java.util.ListIterator;
public class IterateArrayList5 {
   public static void main(String[] args) {
       ArrayList al = new ArrayList();
       al.add("First");
       al.add("Second");
       al.add("Third");
       al.add("Fourth");
       al.add("Fifth");
       System.out.println(al);

       // Iterating using ListIterator.
       // Call listIterator() method to create object of ListIterator using reference variable al.
       ListIterator litr = al.listIterator(); // Here, we are not using generic. Therefore, Typecasting is required.

       // Checking the next element availability in the forward direction using reference variable litr.
       System.out.println("Iteration in the forward direction");
       while(litr.hasNext()) {
           // Moving cursor to next element in the forward direction using reference variable litr.
           Object o = litr.next();
           String str = (String)o; // Typecasting is required because the return type of next() method is an Object.
           System.out.println(str);
       }
       // Checking the previous element in the backward direction using reference variable litr1.
       System.out.println("Iteration in the backward direction.");
       while(litr.hasPrevious()) {
           // Moving cursor to the previous element in the backward direction.
           Object o = litr.previous();
           String str1 = (String)o; // Typecasting.
           System.out.println(str1);
       }
   }
}
```

Example 5: ArrayList Using ListIterator

Output:

```
[First, Second, Third, Fourth, Fifth]
Iteration in the forward direction
First
Second
Third
Fourth
Fifth
Iteration in the backward direction.
Fifth
Fourth
Third
Second
First
```

Example 6: Add Remove Test

Let's make a program, we will perform add and remove operations using ListIterator. We will use generic in this program so that we do not require Typecasting.

```
import java.util.ArrayList;
import java.util.ListIterator;
public class IterateArrayList6 {
   public static void main(String[] args) {
       ArrayList<String> al = new ArrayList<String>();
       al.add("One");
       al.add("Two");
       al.add("Three");
       al.add("Nine");
       al.add("Five");
       al.add("Seven");
       System.out.println(al);

       ListIterator<String> litr = al.listIterator();
       while(litr.hasNext()) {
           String str = litr.next();
           if(str.equals("Nine")) {
               litr.remove();
               litr.add("Four");
               System.out.println(al);
           }
           else if(str.equals("Seven")) {
               litr.set("Six");
               System.out.println(al);
           }
       }
   }
}
```

Example 6: Add Remove Test

Output:

```
[One, Two, Three, Nine, Five, Seven]
[One, Two, Three, Four, Five, Seven]
[One, Two, Three, Four, Five, Six]
```

Example 7: Specific Element Test

Let's create one more program where we will add elements from 0 to 9 numbers in the list by using "for loop". But we will iterate from a specific element '4' in the list.

```
import java.util.ArrayList;
import java.util.ListIterator;
public class IterateArrayList7 {
   public static void main(String[] args) {
       ArrayList<Integer> al = new ArrayList<Integer>();
       for(int i = 0 ; i < = 9 ; i++) {
           al.add(i);
       }
       System.out.println(al);
       ListIterator<Integer> litr = al.listIterator(4); // Iterating through a specific element '4'.
       while(litr.hasNext()) {
           Integer it = litr.next();
           System.out.println(it);
       }
       while(litr.hasPrevious()) {
           al.add(20); // throws ConcurrentModificationException because we can not add element in the ArrayList during Iteration.
           Integer it1 = litr.next();
           System.out.println(it1);
       }
   }
}
```

Example 7: Specific Element Test

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
4
5
6
7
8
9
Exception in thread "main" java.util.ConcurrentModificationException Create breakpoint
    at java.base/java.util.ArrayList$Itr.checkForComodification(ArrayList.java:1013)
    at java.base/java.util.ArrayList$Itr.next(ArrayList.java:967)
    at IterateArrayList7.main(IterateArrayList7.java:17)
```

Key point:

ListIterator returned by ArrayList is also fail-fast. It will throw ConcurrentModificationException. That is, we cannot add or remove an element in the ArrayList during Iteration.

END