Java Iterator with examples

BY CHAITANYA SINGH | FILED UNDER: JAVA COLLECTIONS

Iterator is used for iterating (looping) various collection classes such as HashMap, ArrayList, LinkedList etc. In this tutorial, we will learn what is iterator, how to use it and what are the issues that can come up while using it. Iterator took place of Enumeration, which was used to iterate legacy classes such as Vector. We will also see the differences between Iterator and Enumeration in this tutorial.

Iterator without Generics Example

Generics got introduced in Java 5. Before that there were no concept of Generics.

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

public class IteratorDemo1 {

  public static void main(String args[]){
    ArrayList names = new ArrayList();
    names.add("Chaitanya");
    names.add("Steve");
    names.add("Jack");

    Iterator it = names.iterator();

    while(it.hasNext()) {
      String obj = (String)it.next();
      System.out.println(obj);
    }
  }

}
```

Output:

```
Chaitanya
Steve
Jack
```

In the above example we have iterated ArrayList without using Generics. Program ran fine without any issues, however there may be a possibility of `ClassCastException` if you don't use Generics (we will see this in next section).

Read them too:
How to iterate HashMap
How to iterate LinkedList

Iterator with Generics Example

In the above section we discussed about ClassCastException. Lets see what is it and why it occurs when we don’t use Generics.

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

public class IteratorDemo2 {

  public static void main(String args[]){
    ArrayList names = new ArrayList();
    names.add("Chaitanya");
    names.add("Steve");
    names.add("Jack");

    //Adding Integer value to String ArrayList
    names.add(new Integer(10));

    Iterator it = names.iterator();

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

Output:

```
ChaitanyaException in thread "main"
Steve
Jack
java.lang.ClassCastException: java.lang.Integer cannot be cast to java.lang.String
        at beginnersbook.com.Details.main(Details.java:18)
```

In the above program we tried to add Integer value to the ArrayList of String but we didn’t get any compile time error because we didn’t use Generics. However since we type casted the integer value to String in the while loop, we got ClassCastException.

Use Generics:
Here we are using Generics so we didn’t type caste the output. If you try to add a integer value to ArrayList in the below program, you would get compile time error. This way we can avoid ClassCastException.

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

public class IteratorDemo3 {
  public static void main(String args[]){
    ArrayList<String> names = new ArrayList<String>();
    names.add("Chaitanya");
    names.add("Steve");
    names.add("Jack");

    Iterator<String> it = names.iterator();

    while(it.hasNext()) {
      String obj = it.next();
      System.out.println(obj);
    }
 }
}
```
Note: We did not type cast iterator returned value[it.next()] as it is not required when using Generics.

Difference between Iterator and Enumeration

An iterator over a collection. Iterator takes the place of Enumeration in the Java Collections Framework. Iterators differ from enumerations in two ways:
1) Iterators allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
2) Method names have been improved. hashNext() method of iterator replaced hasMoreElements() method of enumeration, similarly next() replaced nextElement().

ConcurrentModificationException while using Iterator

```
import java.util.ArrayList;
public class ExceptionDemo {
  public static void main(String args[]){
     ArrayList<String> books = new ArrayList<String>();
     books.add("C");
     books.add("Java");
     books.add("Cobol");

     for(String obj : books) {
        System.out.println(obj);
        //We are adding element while iterating list
        books.add("C++");
     }
  }
}
```
Output:

```
C
Exception in thread "main" java.util.ConcurrentModificationException
        at java.util.ArrayList$Itr.checkForComodification(Unknown Source)
        at java.util.ArrayList$Itr.next(Unknown Source)
        at beginnersbook.com.Details.main(Details.java:12)
```
We cannot add or remove elements to the collection while using iterator over it.

Explanation From Javadoc:
This exception may be thrown by methods that have detected concurrent modification of an object when such modification is not permissible.
For example, it is not generally permissible for one thread to modify a Collection while another thread is iterating over it. In general, the results of the iteration are undefined under these circumstances. Some Iterator implementations (including those of all the general purpose collection implementations provided by the JRE) may choose to throw this exception if this behavior is detected. Iterators that do this are known as fail-fast iterators, as they fail quickly and cleanly, rather that risking arbitrary, non-deterministic behavior at an undetermined time in the future.