Iterator :

Iterator is an interface which is made for Collection objects like List, Set. It comes inside java.util package and it was introduced in java 1.2 as public interface Iterator. To generate successive elements from a Collection, we can use java iterator. It contains three methods: those are,

* boolean hasNext(): this method returns true if this Iterator has more element to iterate.
* remove(): method remove the last element return by the iterator this method only calls once per call to next().
* Object next() : Returns the next element in the iteration.

Every collection classes provides an iterator() method that returns an iterator to the beginning of the collection. By using this iterator object, we can access each element in the collection, one element at a time. In general, to use an iterator to traverse through the contents of a collection follow these steps:

* Obtain Iterator object by calling Collections’s iterator() method
* Make a loop to decide the next element availability by calling hasNext() method.
* Get each element from the collection by using next() method.

Example to use iterator:

Here is an example demonstrating Iterator. It uses an ArrayList object, You can apply to any type of collection.

/\*\*

\*

\*/

package com.pamu.test;

/\*\*

\* @author Pamu

\*

\*/

import java.util.ArrayList;

import java.util.Iterator;

public class IteratorExample {

public static void main(String args[]) {

// create an array list. But, you can use any collection

ArrayList<String> aList = new ArrayList<String>();

// add elements to the array list

aList.add("Raja");

aList.add("Rani");

aList.add("Shankar");

aList.add("Uday");

aList.add("Philips");

aList.add("Sachin");

// use iterator to display contents of aList

System.out.print("Original contents of aList: ");

Iterator<String> itr = aList.iterator();

while (itr.hasNext()) {

Object obj = itr.next();

System.out.print(obj + "\n");

}

}

}

Iterator is the best choice if you have to iterate the objects from a Collection. But, Iterator having some limitations.

Those are,

1. Iterator is not index based
2. If you need to update the collection object being iterated, normally you are not allowed to because of the way the iterator stores its position. And it will throw ConcurrentModificationException.
3. It does not allow to traverse to bi directions.

To overcome these on other side, we have For-each loop. It was introduced in Java 1.5

Advantages of For each loop :

1. Index Based
2. Internally the for-each loop creates an Iterator to iterate through the collection.
3. If you want to replace items in your collection objects, use for-each loop

Consider the following code snippet, to understand the difference between how to use Iterator and for-each loop.

Iterator<String> itr = aList.iterator();

while (itr.hasNext()) {

Object obj = itr.next();

System.out.print(obj + "\n");

}

Is same as,

For (String str : aList){

System.out.println(str);

}

Same code has been rewritten using enhanced for-loops which looks more readable. But enhanced for-loops can’t be used automatically in Collection objects that we implement. Then what do we do? This is where Iterable interface comes in to picture. Only, if our Collection implemented Iterable interface, we can iterate using enhanced for-loops. The advantage of implementing Iterable interface So, we should implement Java Iterable interface in order to make our code more elegant.

Here I am going to write a best Practice to develop a custom iterator. Following is the Syntax.

public class MyCollection<E> implements Iterable<E>{

public Iterator<E> iterator() {

return new MyIterator<E>();

}

public class MyIterator <T> implements Iterator<T> {

public boolean hasNext() {

//implement...

}

public T next() {

//implement...;

}

public void remove() {

//implement... if supported.

}

public static void main(String[] args) {

MyCollection<String> stringCollection = new MyCollection<String>();

for(String string : stringCollection){

}

}

}

}

I have given you complete example.

**package** com.pamu.test;

**public** **class** Book {

String name;

String author;

**long** isbn;

**float** price;

**public** Book(String name, String author, **long** isbn, **float** price) {

**this**.name = name;

**this**.author = author;

**this**.isbn = isbn;

**this**.price = price;

}

**public** String toString() {

**return** name + "\t" + author + "\t" + isbn + "\t" + ": Rs" + price;

}

}

This is a Book class containing, Book information name, author,isbn and price

Program 2 :

**package** com.pamu.test;

**import** java.util.ArrayList;

**import** java.util.Iterator;

**import** java.util.List;

**import** com.pamu.test.Book;

**import** java.lang.Iterable;

**public** **class** BookShop **implements** Iterable<Book> {

List<Book> avail\_books;

**public** BookShop() {

avail\_books = **new** ArrayList<Book>();

}

**public** **void** addBook(Book book) {

avail\_books.add(book);

}

**public** Iterator<Book> iterator() {

**return** (Iterator<Book>) **new** BookShopIterator();

}

**class** BookShopIterator **implements** Iterator<Book> {

**int** currentIndex = 0;

@Override

**public** **boolean** hasNext() {

**if** (currentIndex >= avail\_books.size()) {

**return** **false**;

} **else** {

**return** **true**;

}

}

@Override

**public** Book next() {

**return** avail\_books.get(currentIndex++);

}

@Override

**public** **void** remove() {

avail\_books.remove(--currentIndex);

}

}

//main

**public** **static** **void** main(String[] args) {

Book book1 = **new** Book("Java", "JamesGosling", 123456, 1000.0f);

Book book2 = **new** Book("Spring", "RodJohnson", 789123, 1500.0f);

Book book3 = **new** Book("Struts", "Apache", 456789, 800.0f);

BookShop avail\_books = **new** BookShop();

avail\_books.addBook(book1);

avail\_books.addBook(book2);

avail\_books.addBook(book3);

System.***out***.println("Displaying Books:");

**for**(Book total\_books : avail\_books){

System.***out***.println(total\_books);

}

}

}

Here, I made a BookShop class contains books. This class able to use for-each loop only if it implements Iterable interface.Here, you have to provide the implementation for iterator method. I define my BookShopIterator as inner class. Inner classes will provide more security, So that your class is able to access with in the Outer class and this will achieve data encapsulation.

The best reusable option is to implement the interface Iterable and override the method iterator().The main advantage of Iterable is, when you implement Iterable then those object gets support for for:each loop syntax.

An eclipse example :

