**Iterator Design Pattern**

[](https://cdn.journaldev.com/wp-content/uploads/2013/07/iterator-design-pattern.jpg)

According to GoF, iterator design pattern intent is:

Provides a way to access the elements of an aggregate object without exposing its underlying represenation.

Iterator pattern is not only about traversing through a collection, we can provide different kind of iterators based on our requirements.

Iterator design pattern hides the actual implementation of traversal through the collection and client programs just use iterator methods.

**Iterator Pattern Example**

Let’s understand iterator pattern with a simple example. Suppose we have a list of Radio channels and the client program want to traverse through them one by one or based on the type of channel. For example some client programs are only interested in English channels and want to process only them, they don’t want to process other types of channels.

So we can provide a collection of channels to the client and let them write the logic to traverse through the channels and decide whether to process them. But this solution has lots of issues such as client has to come up with the logic for traversal. We can’t make sure that client logic is correct. Furthermore if the number of client grows then it will become very hard to maintain.

Here we can use Iterator pattern and provide iteration based on type of channel. We should make sure that client program can access the list of channels only through the iterator.

The first part of implementation is to define the contract for our collection and iterator interfaces.

ChannelTypeEnum.java

package com.journaldev.design.iterator;

public enum ChannelTypeEnum {

ENGLISH, HINDI, FRENCH, ALL;

}

ChannelTypeEnum is [**java enum**](https://www.journaldev.com/716/java-enum) that defines all the different types of channels.

Channel.java

package com.journaldev.design.iterator;

public class Channel {

private double frequency;

private ChannelTypeEnum TYPE;

public Channel(double freq, ChannelTypeEnum type){

this.frequency=freq;

this.TYPE=type;

}

public double getFrequency() {

return frequency;

}

public ChannelTypeEnum getTYPE() {

return TYPE;

}

@Override

public String toString(){

return "Frequency="+this.frequency+", Type="+this.TYPE;

}

}

Channel is a simple POJO class that has attributes frequency and channel type.

ChannelCollection.java

package com.journaldev.design.iterator;

public interface ChannelCollection {

public void addChannel(Channel c);

public void removeChannel(Channel c);

public ChannelIterator iterator(ChannelTypeEnum type);

}

ChannelCollection interface defines the contract for our collection class implementation. Notice that there are methods to add and remove a channel but there is no method that returns the list of channels. ChannelCollection has a method that returns the iterator for traversal. ChannelIterator interface defines following methods;

ChannelIterator.java

package com.journaldev.design.iterator;

public interface ChannelIterator {

public boolean hasNext();

public Channel next();

}

Now our base interface and core classes are ready, let’s proceed with the implementation of collection class and iterator.

ChannelCollectionImpl.java

package com.journaldev.design.iterator;

import java.util.ArrayList;

import java.util.List;

public class ChannelCollectionImpl implements ChannelCollection {

private List<Channel> channelsList;

public ChannelCollectionImpl() {

channelsList = new ArrayList<>();

}

public void addChannel(Channel c) {

this.channelsList.add(c);

}

public void removeChannel(Channel c) {

this.channelsList.remove(c);

}

@Override

public ChannelIterator iterator(ChannelTypeEnum type) {

return new ChannelIteratorImpl(type, this.channelsList);

}

private class ChannelIteratorImpl implements ChannelIterator {

private ChannelTypeEnum type;

private List<Channel> channels;

private int position;

public ChannelIteratorImpl(ChannelTypeEnum ty,

List<Channel> channelsList) {

this.type = ty;

this.channels = channelsList;

}

@Override

public boolean hasNext() {

while (position < channels.size()) {

Channel c = channels.get(position);

if (c.getTYPE().equals(type) || type.equals(ChannelTypeEnum.ALL)) {

return true;

} else

position++;

}

return false;

}

@Override

public Channel next() {

Channel c = channels.get(position);

position++;

return c;

}

}

}

Notice the [**inner class**](https://www.journaldev.com/996/java-inner-class) implementation of iterator interface so that the implementation can’t be used by any other collection. Same approach is followed by collection classes also and all of them have inner class implementation of Iterator interface.

Let’s write a simple iterator pattern test program to use our collection and iterator to traverse through the collection of channels.

IteratorPatternTest.java

package com.journaldev.design.iterator;

public class IteratorPatternTest {

public static void main(String[] args) {

ChannelCollection channels = populateChannels();

ChannelIterator baseIterator = channels.iterator(ChannelTypeEnum.ALL);

while (baseIterator.hasNext()) {

Channel c = baseIterator.next();

System.out.println(c.toString());

}

System.out.println("\*\*\*\*\*\*");

// Channel Type Iterator

ChannelIterator englishIterator = channels.iterator(ChannelTypeEnum.ENGLISH);

while (englishIterator.hasNext()) {

Channel c = englishIterator.next();

System.out.println(c.toString());

}

}

private static ChannelCollection populateChannels() {

ChannelCollection channels = new ChannelCollectionImpl();

channels.addChannel(new Channel(98.5, ChannelTypeEnum.ENGLISH));

channels.addChannel(new Channel(99.5, ChannelTypeEnum.HINDI));

channels.addChannel(new Channel(100.5, ChannelTypeEnum.FRENCH));

channels.addChannel(new Channel(101.5, ChannelTypeEnum.ENGLISH));

channels.addChannel(new Channel(102.5, ChannelTypeEnum.HINDI));

channels.addChannel(new Channel(103.5, ChannelTypeEnum.FRENCH));

channels.addChannel(new Channel(104.5, ChannelTypeEnum.ENGLISH));

channels.addChannel(new Channel(105.5, ChannelTypeEnum.HINDI));

channels.addChannel(new Channel(106.5, ChannelTypeEnum.FRENCH));

return channels;

}

}

When I run above program, it produces following output;

Frequency=98.5, Type=ENGLISH

Frequency=99.5, Type=HINDI

Frequency=100.5, Type=FRENCH

Frequency=101.5, Type=ENGLISH

Frequency=102.5, Type=HINDI

Frequency=103.5, Type=FRENCH

Frequency=104.5, Type=ENGLISH

Frequency=105.5, Type=HINDI

Frequency=106.5, Type=FRENCH

\*\*\*\*\*\*

Frequency=98.5, Type=ENGLISH

Frequency=101.5, Type=ENGLISH

Frequency=104.5, Type=ENGLISH

**Iterator Design Pattern Important Points**

* Iterator pattern is useful when you want to provide a standard way to iterate over a collection and hide the implementation logic from client program.
* The logic for iteration is embedded in the collection itself and it helps client program to iterate over them easily.

**Iterator Design Pattern in JDK**

We all know that Collection framework Iterator is the best example of iterator pattern implementation but do you know that java.util.Scanner class also Implements Iterator interface. Read this post to learn about [Java Scanner Class](https://www.journaldev.com/872/scanner-class-in-java).

That’s all for iterator design pattern, I hope it’s helpful and easy to understand