Iterator Design Pattern in Ja

APRIL 4, 2018 BY PANKAJ — 14 COMMENTS

Iterator design pattern in one of the behavioral pattern. Iterator pattern is used to provide a standard way to traverse through a group of Objects. Iterator pattern is widely used in Java Collection Framework. Iterator interface provides methods for traversing through a collection.

Table of Contents [hide]

Iterator Design Pattern

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** that defines all the different types of channels.

`Channel.java`

```
    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(){
        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

```
		@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** 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());
		}
	}
```

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.

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

« **PREVIOUS**
Interpreter Design Pattern in Java

NEXT »
Mediator Design Pattern in Java

About Pankaj

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: DESIGN PATTERNS

Comments

yash says
DECEMBER 3, 2017 AT 4:54 AM
sir aapne isme structure of iterator bhi mention karna tha usse aur acchi se samaj aata tha well explanation is nice
Reply

Indukumar Bhayani says
NOVEMBER 17, 2017 AT 11:21 PM
Very lucid explanations; Congrats for such a tutorial with great clarity!
One thing, I would like to point out on the code for the Channel POJO class: Don't we need to override equals and hashCode methods for removal of Channels from the Collection implementation class?
Reply

Pankaj says
NOVEMBER 18, 2017 AT 9:12 AM
If you look at the ArrayList remove method, it uses equals() method. Since we haven't implemented it here, it will use Object class implementation where == is used to check if two objects are equal or not.
In ideal world we should implement our own equals() method. For example, what if there are other fields in the Channel class that we don't care about. So we can skip them while comparing if two Channels are equal or not in equals() method implementation.
Reply

PriyankaNeeli says
MAY 10, 2017 AT 12:24 AM
I see int position is not initialized. Doesnt it supposed to be ?
Reply

Pramod Sapare says
JULY 31, 2017 AT 6:40 PM
It is an instance variable and hence defaults to zero
Reply

Monika says

APRIL 27, 2017 AT 9:36 AM

Very good example..learnt completely about how iterator works..Thanks for writing !!

Reply

Deepak Chourasia says

MAY 4, 2017 AT 11:18 AM

remember iterator might have different algorithms for traversal of the elements so just be aware. That is why we have different kind of Iterator implementations for different types of collections.

Reply

sa says

JULY 25, 2016 AT 4:13 PM

Can you clarify how the populateChannels() is intantiated?

Reply

Monika says

APRIL 27, 2017 AT 9:38 AM

inside populateChannels() we are careating an object of ChannelCollectionImpl which calling its defaut constructor. In that default constructor we are initializing an array list. After that we are adding channels to that array list,

Reply

asu says

OCTOBER 16, 2015 AT 8:53 AM

HasNext() and Next() if called one after other, your implementation will not work. you need to keep track if HasNext called before and return the Channel c in Next() before calling get(position) again. Otherwise, hasNext() had moved you to next and calling Next() after , will move you one more place.

Reply

Suruchi says

DECEMBER 26, 2014 AT 3:05 AM

How do I change the destination path for the written excel file?

Reply

suraj says

FEBRUARY 3, 2015 AT 8:28 PM

thanks

Reply

Vikash Saini says

SEPTEMBER 15, 2014 AT 4:52 AM

I dnt understand .how position variable is working. From where it would get value ?

Can somebody please answer .

Thanks!

Reply

santhosh says

SEPTEMBER 29, 2014 AT 9:37 AM

Position is an instance variable. In java, all instance variables default value is zero.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

GET IT ON Google Play

DESIGN PATTERNS TUTORIAL

- Composite
- Proxy
- Flyweight
- Facade
- Bridge
- Decorator

Behavioral Design Patterns

- Template Method
- Mediator
- Chain of Responsibility
- Observer
- Strategy
- Command
- State
- Visitor
- Interpreter
- Iterator
- Memento

Miscellaneous Design Patterns

- Dependency Injection
- Thread Safety in Java Singleton

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial