javapapers

Iterator Design Pattern

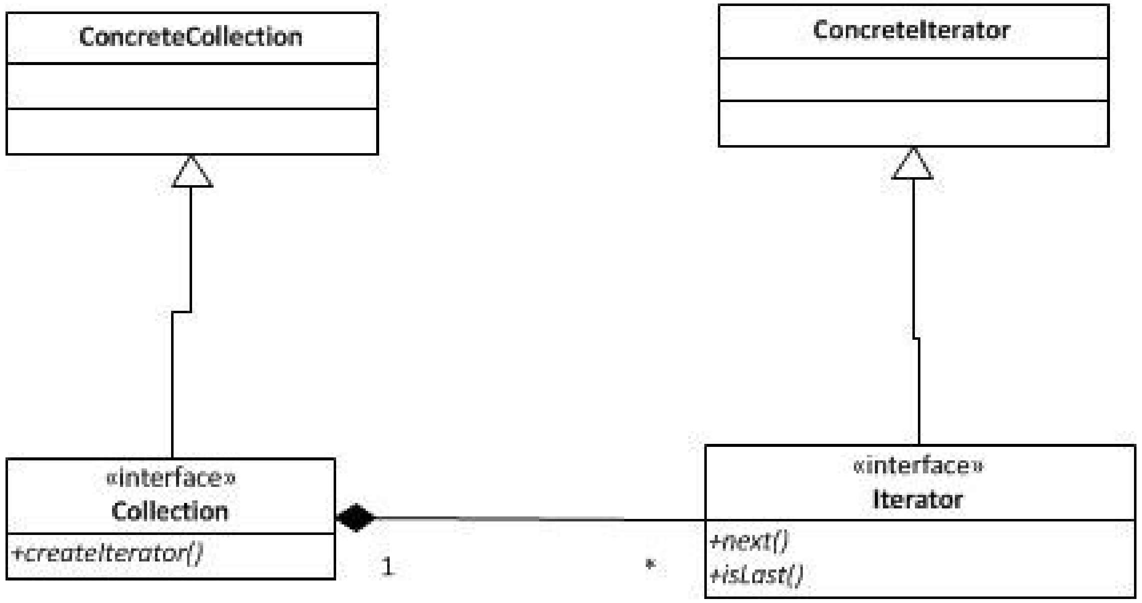
In this design pattern tutorial series we will discuss about iterator design pattern, which allows to traverse a collection without exposing its internals. Ah we all know about the Java Iterator. But it is not all about it, lets dive deeper and do a comprehensive study of iterator pattern. By the end of this tutorial we will know about internal iterator, external iterator, polymorphic iterator, robust iterator, null iterator and etc.

Intent of an iterator is to provide a way to traverse through a collection (otherwise called an aggregate) in a sequential way without exposing the underlying collection. Initially Java had Enumeration to iterate through the collections and from Java version 1.2 Iterator took its place. Java Iterator is a nice implementation of iterator design pattern allowing us to traverse through a collection in a sequential way. On call, the collection will create an iterator instance and return it, which we can use to iterate.

In the Iterator design pattern, iterator is an interface which provides a simple interaction mechanism. More importantly the user of the iterator interface need not know about the working mechanisms or complexity of the related collection. All, the client needs to know is about the simple interface provided by the iterator.

Iterator can do more than traversal

Iterators are not limited to traversal alone. It is entirely left to the purpose and implementation. There can be functional logic involved in an iterator. We can have a FilteringIterator, which can filter out certain required values and provide for traversal. For example in a list containing wild and domestic animals, we can have two different iterators as WildAnimalIterator and DomesticAnimalIterator.



A factory method available in the collection will return the instance of the iterator.

Polymorphic Iterator

If an iterator has common contract across multiple collections, the client code which interacts with the iterator need not be changed for different collections. This is called polymorphic iterator.

External vs Internal Iterator

When the client controls the iteration sequence and index position then it is called as the external iterator. Otherwise if the iterator controls the traversal then it is called internal iterator. On external iterator, the design is that the invoking client code should explicitly invoke the method to move the pointer to next element. For internal iterator, when an element is accessed, after access the pointer will be automatically moved to next index by the iterator. In general internal iterators are easier to use than the external iterators.

Robust Iterator

This is an important topic. Imagine an iterator instance is at work and what would happen if the contents of the underlying collection is modified like a new element is inserted or an existing element is deleted? It will behave inconsistently resulting in erroneous output. A simplistic iterator will create a copy of the collection when it is instantiated and use that for traversal. So that during the iteration process, even if the collection is modified nothing would happen.

A robust iterator is the one that works intact even if the associated collection is modified. It even does the implementation without making a copy of the association. That is the effects of modification of the collection will also be absorbed into the iterator. For example, it can be implemented using an event listener notifying the iterator of the changes and it gets updated for the changes.

Null Iterator

Imagine we are traversing a tree data structure. During the tree traversal, when we ask for the nextElement, we will get a concrete iterator which will help us to traverse through the tree. If we ask for the nextElement in the leaf node, we will get a null iterator returned by the collection signifying the leaf node. This behavior will allow us to design the tree traversal pattern.

Key Points for Iterator Design Pattern

- Iterators can have complex traversal algorithms and provide variety of mechanisms.
- At one instance multiple iterator instances can be live and being pointing to different index.

Example Iterator Design Pattern

Animal.java

```java
package com.javapapers.designpattern.iterator;

public class Animal {
        private String animalName;
        private String animalType;

        public Animal(String animalName, String animalType) {
                this.animalName = animalName;
                this.animalType = animalType;
        }

        public String getAnimalName() {
                return animalName;
        }

        public void setAnimalName(String animalName) {
                this.animalName = animalName;
        }

        public String getAnimalType() {
                return animalType;
        }

        public void setAnimalType(String animalType) {
                this.animalType = animalType;
        }
}
```

IZoo.java

```
package com.javapapers.designpattern.iterator;

import java.util.List;

public interface IZoo {

        public List getAnimals();

        public void addAnimal(Animal animal);

        public void removeAnimal(Animal animal);

        public Iterator createIterator(String iteratorType);
}
```

ZooImpl.java

```
package com.javapapers.designpattern.iterator;

import java.util.ArrayList;
import java.util.List;

public class ZooImpl implements IZoo {

        List animalList;
        public ZooImpl() {
                animalList = new ArrayList();
        }

        @Override
        public List getAnimals() {

                return animalList;
        }

        @Override
        public void addAnimal(Animal animal) {
                animalList.add(animal);

        }

        @Override
        public void removeAnimal(Animal animal) {
                animalList.remove(animal);

        }
```

Iterator.java

```
package com.javapapers.designpattern.iterator;
```

```
public interface Iterator {
        public Animal nextAnimal();

        public boolean isLastAnimal();

        public Animal currentAnimal();
}
```

WildIterator.java

```
package com.javapapers.designpattern.iterator;

import java.util.List;

public class WildIterator implements Iterator {

        public List animalList;
        private int position;

        public WildIterator(List animalList) {
                this.animalList = animalList;
        }

        @Override
        public Animal nextAnimal() {
                Animal animal = null;
                for (; position < animalList.size(); position++) {
```

```java
                        if ("Wild".equals((animalList.get(position)).getAnim
alType())) {
                                animal = animalList.get(position);
                                position++;
                                break;
                        }
                }
                return animal;
        }

        @Override
        public boolean isLastAnimal() {
                for (int i = position; i < animalList.size(); i++) {
```

DomesticIterator.java

```java
package com.javapapers.designpattern.iterator;

import java.util.List;

public class DomesticIterator implements Iterator {

        List animalList;
        private int position;

        public DomesticIterator(List animalList) {
                this.animalList = animalList;
        }

        @Override
        public Animal nextAnimal() {
                Animal animal = null;
                for (; position < animalList.size(); position++) {
                        if ("Domestic".equals((animalList.get(position)).get
AnimalType())) {
                                animal = animalList.get(position);
                                position++;
                                break;
                        }
                }
                return animal;
        }

        @Override
        public boolean isLastAnimal() {
                for (int i = position; i < animalList.size(); i++) {
```

IteratorSample.java

```
package com.javapapers.designpattern.iterator;

public class IteratorSample {
        public static void main(String args[]) {
                ZooImpl zoo = new ZooImpl();
```

```
                    zoo.addAnimal(new Animal("Tiger", "Wild"));
                    zoo.addAnimal(new Animal("Lion", "Wild"));
                    zoo.addAnimal(new Animal("Tom Cat", "Domestic"));
                    zoo.addAnimal(new Animal("Raging Bull", "Wild"));
                    zoo.addAnimal(new Animal("Scooby Doo", "Domestic"));

                    Iterator wildIterator = zoo.createIterator("Wild");
                    while (!wildIterator.isLastAnimal()) {
                            System.out.println("Wild Animal: "
                                            + wildIterator.nextAnimal().getAnima
lName());
                    }

                    Iterator domesticIterator = zoo.createIterator("Domestic");
                    while (!domesticIterator.isLastAnimal()) {
                            System.out.println("Domestic Animal: "
                                            + domesticIterator.nextAnimal().getA
nimalName());
                    }
            }
}
```

Sample Source Code Output:

```
Wild Animal: Tiger
Wild Animal: Lion
Wild Animal: Raging Bull
```

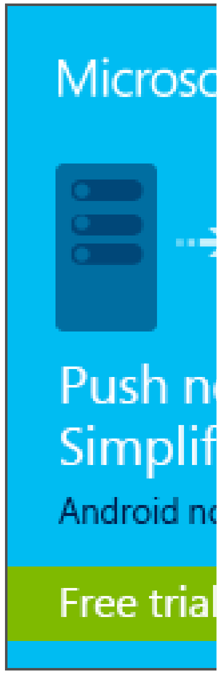
Domestic Animal: Tom Cat

Domestic Animal: Scooby Doo

Iterator Design Pattern Source Code Download

Iterator Design Pattern Source Code

This Behavioral Design Pattern tutorial was added on 30/06/2013.



« hashCode And equals Methods Override

Comments on “Iterator Design Pattern”

Omprakash *01/07/2013 at 2:31 pm*

Very nice post!!

Reply

---

manni *01/07/2013 at 6:04 pm*

well done!

Reply

---

sandy *02/07/2013 at 4:31 pm*

the content is definitely worth it… but what is that police wala doing in tech stuff? :D

Reply

Joe *02/07/2013 at 10:39 pm*

Ah! he is the traffic Iterator, sending one by one in sequence. Fun isn’t it :D

Reply

Srinu D *05/07/2013 at 5:34 pm*

Very useful this post..thanks

Reply

grails cookbook *20/07/2013 at 9:18 pm*

We usually don’t use iterators for the company I work for. Most of our managers says it has slow performance.

Reply

Ramesh Vankayala *21/07/2013 at 4:57 pm*

I am appreciating your hard work, You are doing very good job, I am bringing my observation to your notice

```
public class ZooImpl implements IZoo {

List animalList;

@Override
public List getAnimals() {

return animalList;
}
```

In the ZooImpl, animalList is our collection, which stores all our animals, While calling getAnimals(), you are returning the animalList storage collection to client.So that cient may modify collection like animalList.add(animal) or animalList.remove(animal).

As the iterator pattern, “provides a way to access the elements of an aggregate Object sequentially without exposing its underlying representation”

In this example, you are exposing the underlying representation of animalList.

Reply

Cookies *03/08/2014 at 8:49 am*

It seems that `position` is not initialized in WildIterator&DemosticIterator

Reply

Jitendra *22/07/2013 at 11:32 am*

Well explained :)

Reply

Rahul *01/08/2013 at 10:12 am*

As always, neat and good explanation !

Reply

Dineshkumar *05/09/2013 at 12:48 pm*

Nice explanation.

Reply

Anonymous *14/10/2013 at 2:51 pm*

pretty code

Reply

Satish K *31/12/2013 at 4:55 pm*

I appreciate your hard-work. It is really very good and step by step explanation with example.

Reply

---

sanyhn *14/03/2014 at 11:43 pm*

Hi.. Useful explained in a simple n clear way..

Keep going .. :)

Reply

Your Comment

Name	
Email	

Post Comment

TUTORIAL MENU

Java

Android

Design Patterns

Creational Design Patterns

Structural Design Patterns

Behavioral Design Patterns

Chain Of Responsibility Design Pattern

Command Design Pattern

Interpreter Design Pattern

Iterator Design Pattern

Mediator Design Pattern

Memento Design Pattern

Observer Design Pattern

State Design Pattern

Template Method Design Pattern

Spring

Web Services

Servlet

Java Android Design Patterns Spring Web Services

Servlet

Site Map