Design Principles/Patterns Foundation Assignment

Iterator Design Pattern

Iterator pattern is very commonly used design pattern in Java programming environment. This pattern is used to get a way to access the elements of a collection object in sequential manner without any need to know its underlying representation. Iterator Pattern is used to access the elements of an aggregate object sequentially without exposing its underlying implementation". The Iterator pattern is also known as Cursor.

In collection framework, Iterator design pattern is preferred over Enumeration.

Intent

- Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
- The C++ and Java standard library abstraction that makes it possible to decouple collection classes and algorithms.
- Promote to "full object status" the traversal of a collection.
- Polymorphism traversal.

Advantages of Iterator Design Pattern

- It supports variations in the traversal of a collection.
- It simplifies the interface to the collection.

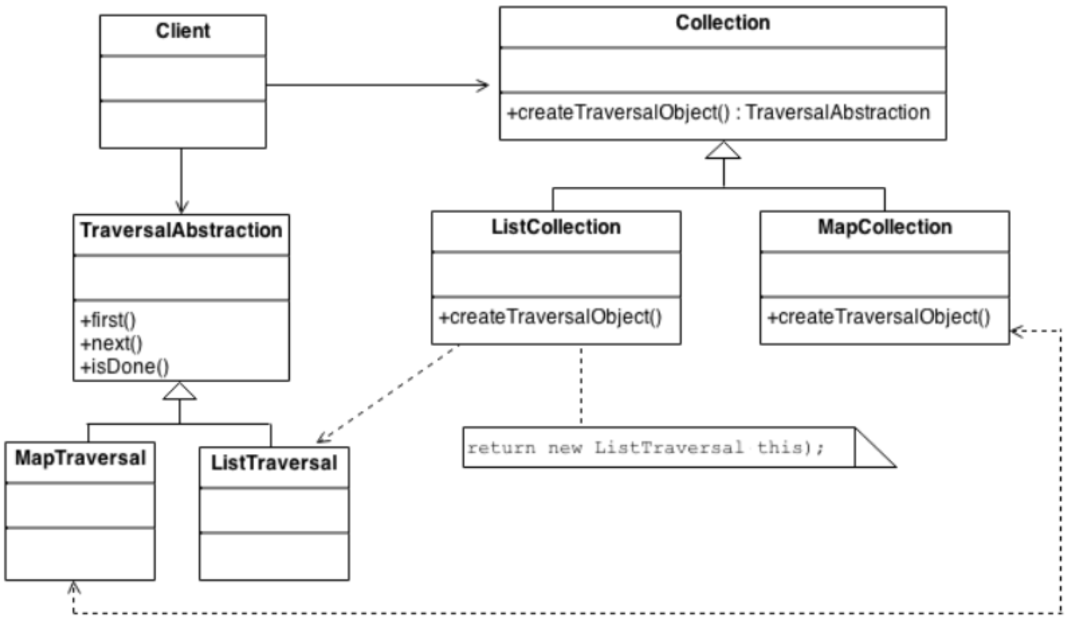
Usage of Iterator Pattern:

- It is used when we want to access a collection of objects without exposing its internal representation.
- It is used when there are multiple traversals of objects need to be supported in the collection.

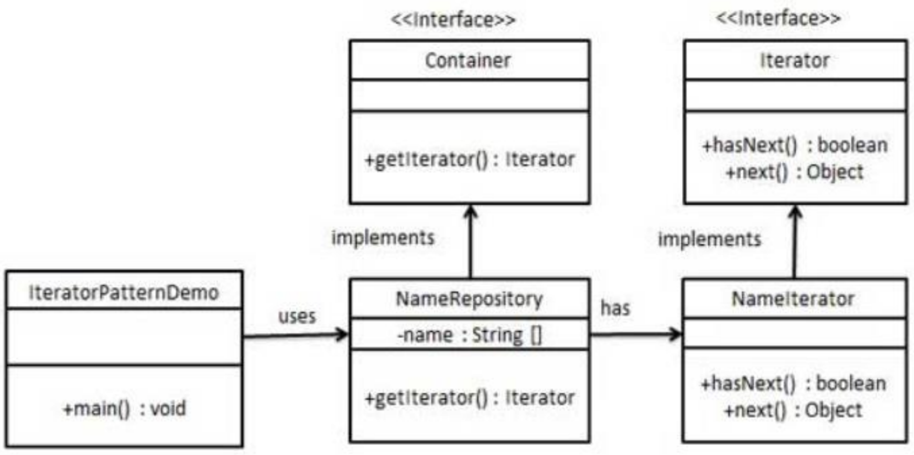
Structure

The Client uses the Collection class public interface directly. But access to the Collection's elements is encapsulated behind the additional level of abstraction called Iterator. Each

Collection derived class knows which Iterator derived class to create and return. After that, the Client relies on the interface defined in the Iterator base class.



Implementation:



We're going to create a Iterator interface which narrates navigation method and a Container interface which returns the iterator. Concrete classes implementing the Container interface will be responsible to implement Iterator interface and use it.

IteratorPatternDemo, is demo class will use NamesRepository, a concrete class implementation to print a Names stored as a collection in NamesRepository.

Step 1

Create interfaces.

Iterator.java

```
public interface Iterator {
   public boolean hasNext();
   public Object next();
}
```

Container.java

```
public interface Container {
   public Iterator getIterator();
}
```

Step 2

Create concrete class implementing the Container Interface. This class has inner class NameIterator implementing the Iterator interface.

NameRepository.java

```
public class NameRepository implements Container {
   public String names[] = {"Robert" , "John" ,"Julie" , "Lora"};

   @Override
   public Iterator getIterator() {
      return new NameIterator();
   }

   private class NameIterator implements Iterator {

      int index;

      @Override
      public boolean hasNext() {

         if(index < names.length){
            return true;
         }
         return false;
      }

      @Override
      public Object next() {

         if(this.hasNext()){
            return names[index++];
         }
         return null;
      }
   }
}
```

Step 3

Use the NameRepository to get iterator and print names.

IteratorPatternDemo.java

```
public class IteratorPatternDemo {

   public static void main(String[] args) {
      NameRepository namesRepository = new NameRepository();

      for(Iterator iter = namesRepository.getIterator();
iter.hasNext();){
         String name = (String)iter.next();
         System.out.println("Name : " + name);
      }
   }
}
```

Step 4

Verify the output

```
Name : Robert
Name : John
Name : Julie
Name : Lora
```