В обектно ориентираното програмиране итератор патърна е такъв патърн, в който се се използва итератор за траверсиране на контейнер и достъпване на елементите на контейнера. Итератора разделя алгоритми от контейнери. В някои случаи алгоритмите са строго зависещи от контейнера и не могат да се разделят. Например хипотетичния алгоритъм за ТърсенеНаЕлемент може да бъде имплементиран основно използвайки точно определен тип итератор вместо да се имплементира като алгоритъм зависещ от някакъв контейнер. Това позволява ТърсенеНаЕлемент да се използва за всеки контейнер, който използва точно определения тип итератор.

<http://www.dofactory.com/Patterns/Diagrams/iterator.gif>

|  |
| --- |
| // Iterator pattern -- Structural example |
| using System;  using System.Collections;    namespace DoFactory.GangOfFour.Iterator.Structural  {      /// <summary>      /// MainApp startup class for Structural      /// Iterator Design Pattern.      /// </summary>      class MainApp      {          /// <summary>          /// Entry point into console application.          /// </summary>          static void Main()          {              ConcreteAggregate a = new ConcreteAggregate();              a[0] = "Item A";              a[1] = "Item B";              a[2] = "Item C";              a[3] = "Item D";                // Create Iterator and provide aggregate              ConcreteIterator i = new ConcreteIterator(a);                Console.WriteLine("Iterating over collection:");                object item = i.First();              while (item != null)              {                  Console.WriteLine(item);                  item = i.Next();              }                // Wait for user              Console.ReadKey();          }      }        /// <summary>      /// The 'Aggregate' abstract class      /// </summary>      abstract class Aggregate      {          public abstract Iterator CreateIterator();      }        /// <summary>      /// The 'ConcreteAggregate' class      /// </summary>      class ConcreteAggregate : Aggregate      {          private ArrayList \_items = new ArrayList();            public override Iterator CreateIterator()          {              return new ConcreteIterator(this);          }            // Gets item count          public int Count          {              get { return \_items.Count; }          }            // Indexer          public object this[int index]          {              get { return \_items[index]; }              set { \_items.Insert(index, value); }          }      }        /// <summary>      /// The 'Iterator' abstract class      /// </summary>      abstract class Iterator      {          public abstract object First();          public abstract object Next();          public abstract bool IsDone();          public abstract object CurrentItem();      }        /// <summary>      /// The 'ConcreteIterator' class      /// </summary>      class ConcreteIterator : Iterator      {          private ConcreteAggregate \_aggregate;          private int \_current = 0;            // Constructor          public ConcreteIterator(ConcreteAggregate aggregate)          {              this.\_aggregate = aggregate;          }            // Gets first iteration item          public override object First()          {              return \_aggregate[0];          }            // Gets next iteration item          public override object Next()          {              object ret = null;              if (\_current < \_aggregate.Count - 1)              {                  ret = \_aggregate[++\_current];              }                return ret;          }            // Gets current iteration item          public override object CurrentItem()          {              return \_aggregate[\_current];          }            // Gets whether iterations are complete          public override bool IsDone()          {              return \_current >= \_aggregate.Count;          }      }  } |
| Output  Iterating over collection: Item A Item B Item C Item D |