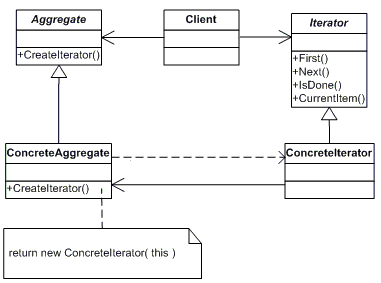
**Section20 Iterator Design Pattern**

**Notes: -**

**1-Iterator : Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.**



**Example:-**

**using IteratorDesignPro.Iterators;**

**namespace IteratorDesignPro.Aggregators{**

**abstract class Aggregate{**

**public abstract Iterator CreateIterator();}}**

**using IteratorDesignPro.Iterators;**

**using System.Collections;**

**namespace IteratorDesignPro.Aggregators{**

**class ConcreteAggregate : Aggregate{**

**private ArrayList \_items = new ArrayList();**

**//create new instance of the CreateIterator and link between ConcreteAggregate and ConcreteIterator**

**public override Iterator CreateIterator(){return new ConcreteIterator(this);}**

**// Gets item count**

**public int Count{get { return \_items.Count; }}**

**//to set value and get value from list through indexer**

**public object this[int index]{get { return \_items[index]; }set { \_items.Insert(index, value); }}}}**

**namespace IteratorDesignPro.Iterators{**

**abstract class Iterator{**

**public abstract object First();**

**public abstract object Next();**

**public abstract bool IsDone();**

**public abstract object CurrentItem();}}**

**using IteratorDesignPro.Aggregators;**

**namespace IteratorDesignPro.Iterators{**

**class ConcreteIterator : Iterator{**

**private ConcreteAggregate \_aggregate;**

**private int \_current = 0;**

**//we pass the ConcreteAggregator to accesss the list and apply operations and assign ConcreteAggregate with this \_aggregate**

**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;}}}**

**//on Main app we initialize the ConcreteAggregate() and set multiple items and create iterator() and //with the dual assignment on both class the iterator can access to the same instance of the //Concrete Aggregate()**

**static void Main(string[] args){**

**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**

**Iterator i = a.CreateIterator();**

**Console.WriteLine("Iterating over collection:");**

**//so the iterator is now accessable to the aggregator**

**object item = i.First();**

**while (item != null){**

**Console.WriteLine(item);**

**item = i.Next();}**

**Console.ReadKey();}**

**Example: -**

**using DoFactoryIterator2Pro.Aggregators;**

**namespace DoFactoryIterator2Pro.Iterators{**

**interface IAbstractIterator{**

**Item First();**

**Item Next();**

**bool IsDone { get; }**

**Item CurrentItem { get; }}}**

**using DoFactoryIterator2Pro.Aggregators;**

**namespace DoFactoryIterator2Pro.Iterators{**

**class Iterator : IAbstractIterator{**

**private Collection \_collection;**

**private int \_current = 0;**

**private int \_step = 1;**

**public Iterator(Collection collection){this.\_collection = collection;}**

**// Gets first item**

**public Item First(){**

**\_current = 0;**

**return \_collection[\_current] as Item;}**

**// Gets next item**

**public Item Next(){**

**\_current += \_step;**

**if (!IsDone)**

**return \_collection[\_current] as Item;**

**else**

**return null;}**

**// Gets or sets stepsize**

**public int Step{get { return \_step; }set { \_step = value; }}**

**// Gets current iterator item**

**public Item CurrentItem{get { return \_collection[\_current] as Item; }}**

**// Gets whether iteration is complete**

**public bool IsDone{get { return \_current >= \_collection.Count; }}}}**

**using DoFactoryIterator2Pro.Iterators;**

**namespace DoFactoryIterator2Pro.Aggregators{**

**interface IAbstractCollection{Iterator CreateIterator();}}**

**using DoFactoryIterator2Pro.Iterators;**

**using System.Collections;**

**namespace DoFactoryIterator2Pro.Aggregators{**

**class Collection : IAbstractCollection{**

**private ArrayList \_items = new ArrayList();**

**//to create iterator based**

**public Iterator CreateIterator(){return new Iterator(this);}**

**// Gets item count**

**public int Count{get { return \_items.Count; }}**

**// Indexer**

**public object this[int index]{get { return \_items[index]; }set { \_items.Add(value); }}}}**

**namespace DoFactoryIterator2Pro.Aggregators{**

**class Item{**

**private string \_name;**

**// Constructor**

**public Item(string name){this.\_name = name;}**

**// Gets name**

**public string Name{get { return \_name; }}}**

**using DoFactoryIterator2Pro.Aggregators;**

**using DoFactoryIterator2Pro.Iterators;**

**using System;**

**namespace DoFactoryIterator2Pro{**

**class Program{**

**static void Main(string[] args){**

**// Build a collection**

**Collection collection = new Collection();**

**collection[0] = new Item("Item 0");**

**collection[1] = new Item("Item 1");**

**collection[2] = new Item("Item 2");**

**collection[3] = new Item("Item 3");**

**collection[4] = new Item("Item 4");**

**collection[5] = new Item("Item 5");**

**collection[6] = new Item("Item 6");**

**collection[7] = new Item("Item 7");**

**collection[8] = new Item("Item 8");**

**// Create iterator**

**Iterator iterator = collection.CreateIterator();**

**// Skip every other item**

**iterator.Step = 2;**

**Console.WriteLine("Iterating over collection:");**

**for (Item item = iterator.First();**

**!iterator.IsDone; item = iterator.Next()){Console.WriteLine(item.Name);}**

**// Wait for user**

**Console.ReadKey();}}}**