ITERATOR

Design Pattern

INTERNATIONAL ISLAMIC UNIVERSITY ISLAMABAD

Report

Name: Tamoor Ahmad

Registration No: 4125-FBAS/BSSE/F20

Course Code: SE322

Course Title: Software Design and Architecture

Submitted to: Dr. Muhammad Nasir

Suppose we are a notification bar in our application that displays all the notifications which creating are held in a notification collection. NotificationCollection provides an iterator to iterate over its elements without exposing how it has implemented the collection (array in this case) to the Client (NotificationBar).

Class Diagram:

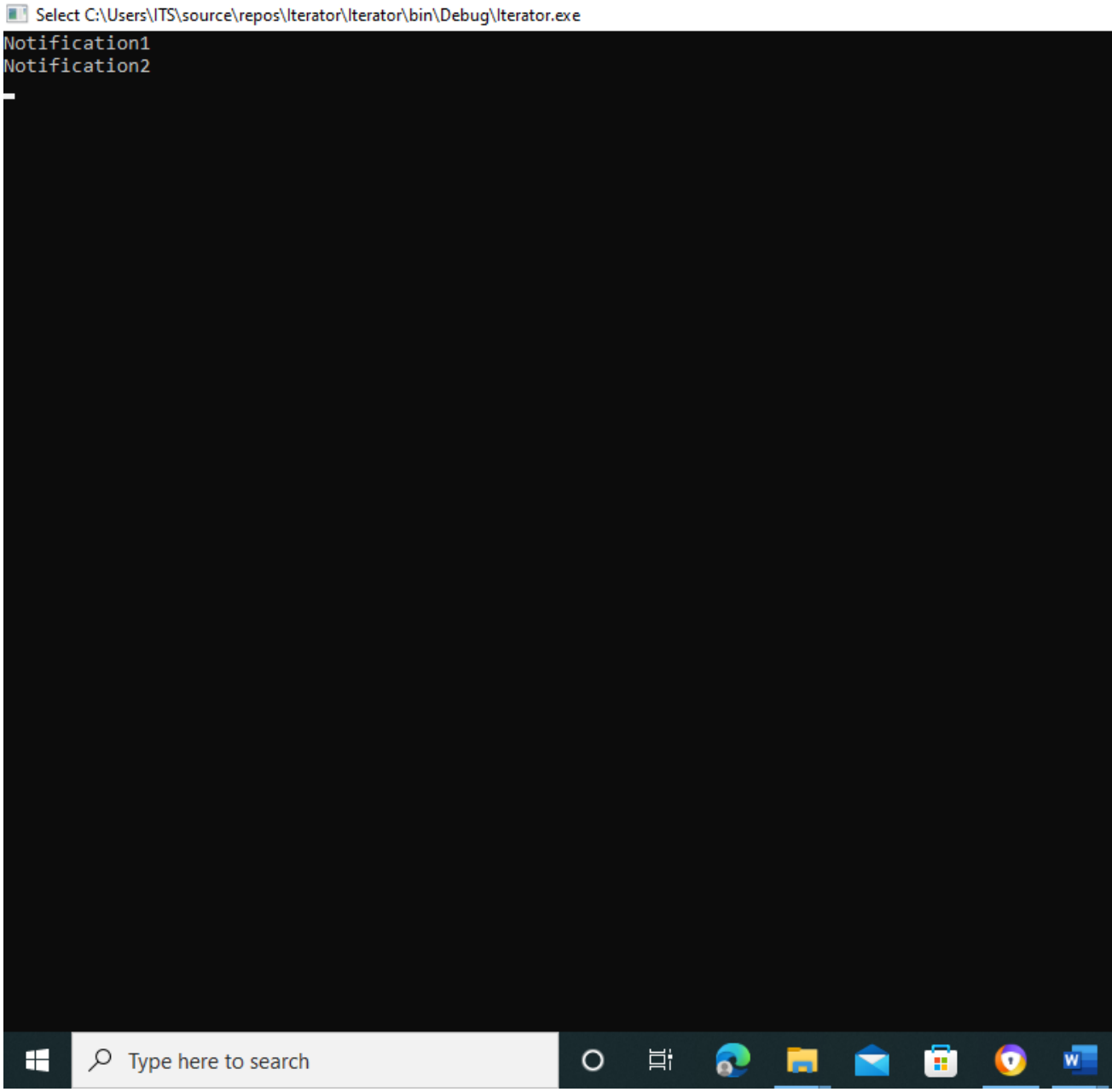
Source Code:

```
using System;
using System.Collections.Generic;
namespace Iterator.Structural
{

    public class MainProgram
    {
        public static void Main(string[] args)
        {
            NotificationCollect a = new NotificationCollect();
            a[0] = "Notification1";
            a[1] = "Notification2";

            Iterator i = a.CreateIterator();
            object item = i.First();
            while (item != null)
            {
```

```
            Console.WriteLine(item);
            item = i.Next();
        }

        Console.ReadKey();
    }
}

public abstract class Collection
{
    public abstract Iterator CreateIterator();
}

public class NotificationCollect : Collection
{
    List<object> items = new List<object>();
    public override Iterator CreateIterator()
    {
        return new NotificationIterator(this);
    }
    public int Count
    {
        get { return items.Count; }
    }

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

public abstract class Iterator
{
    public abstract object First();
    public abstract object Next();
    public abstract bool IsDone();
    public abstract object CurrentItem();
}
/// <summary>
/uì// The 'ConcreteIterator' class
/// </summary>
public class NotificationIterator : Iterator
{
    NotificationCollect Collection;
    int current = 0;

    public NotificationIterator(NotificationCollect collection)
    {
        this.Collection = collection;
    }

    public override object First()
    {
        return Collection[0];
    }

    public override object Next()
    {
        object ret = null;
        if (current < Collection.Count - 1)
        {
```

```
                ret = Collection[++current];
            }
            return ret;
        }

        public override object CurrentItem()
        {
            return Collection[current];
        }

        public override bool IsDone()
        {
            return current >= Collection.Count;
        }
    }
}
```

Output:


Select C:\Users\ITS\source\repos\Iterator\Iterator\bin\Debug\Iterator.exe
Notification1
Notification2
Type here to search