Буниф Мажда

18-КБ-ПР2

**Отчет КПО от Лаб 2**

1. Создать интерфейс (в соответствии с вариантом задания из п.5) для АТД.
2. Создать класс (в соответствии с вариантом задания из п.5), реализующий указанный интерфейс и обеспечивающий работу с АТД.

3) Составить тесты к методам класса, проверяющие корректность их работы.

4) Запустить тестирование и проверить результаты работы.

5) В программе, демонстрирующей работу класса, реализовать доступ к АТД через интерфейс.

6) Создать класс для АТД (в соответствии с вариантом задания из предыдущей работы), реализующий стандартные интерфейсы IEnumerable и ICloneable.

7) Создать параметризованный класс (в соответствии с вариантом задания из предыдущих работ), реализующий указанный интерфейс и обеспечивающий работу с АТД.

8) Создать параметризованный метод, реализующий работу с АТД.

9) Добавить в разработанный ранее класс (в соответствии с вариантом задания из предыдущих работ), обеспечивающий работу с АТД возможность сериализации

**Листинг :**

**Interface.cs // Stack Interface**

interface IStack<T> : IEnumerable<T>// Stack interface

{

void Push(T value);

T Peek();

T Pop();

void Clear();

}

**ImplementInterface.cs //**

public class StackImplementation<T> : IStack<T>

{

LinkedList<T> ll = new LinkedList<T>();

public StackImplementation() { }

public StackImplementation(LinkedList<T> ll)

{

this.ll = ll;

}

public void Push(T data)=> ll.AddFirst(data);

public T Pop()

{

T value = ll.First.Value;

ll.RemoveFirst();

return value;

}

public T Peek()=> ll.First.Value;

public void Clear()=> ll.Clear();

public bool Contains(T data)

{

int counter = 0;

foreach (T x in ll)

if (x.Equals(data))

counter++;

if (counter > 0)

return true;

else return false;

}

public int Count() => ll.Count;

**ImplementDataStructure.cs // Class which implements Stack using a linked list**

public class implDataStru<T> : IStack<T>, ICloneable, ISerializable

{

LinkedList<T> linkedlist = new LinkedList<T>();

//public implDataStru(implDataStru<int> linkedlist) { }

public implDataStru() { }

public implDataStru(LinkedList<T> linkedlist)

{

this.linkedlist = linkedlist;

}

public void Push(T value)

{

linkedlist.AddFirst(value);

}

public T Pop()

{

if (linkedlist.Count == 0)

{

throw new InvalidOperationException("The Stack is empty");

}

T value = linkedlist.First.Value;

linkedlist.RemoveFirst();

return value;

}

public T Peek()

{

if (linkedlist.Count == 0)

{

throw new InvalidOperationException("The Stack is empty");

}

return linkedlist.First.Value;

}

public void Clear()

{

linkedlist.Clear();

}

public int Count() => linkedlist.Count;

public bool Contains(T data)

{

int counter = 0;

foreach (T x in linkedlist)

if (x.Equals(data))

counter++;

if (counter > 0)

return true;

else return false;

}

public IEnumerator<T> GetEnumerator() => linkedlist.GetEnumerator();

IEnumerator IEnumerable.GetEnumerator()

{

T Value = linkedlist.First.Value;

while (Value != null)

{

yield return Value;

linkedlist.RemoveFirst();

Value = linkedlist.First.Value;

}

}

public object Clone() => new implDataStru<T>(linkedlist);

public void GetObjectData(SerializationInfo info, StreamingContext context) => info.AddValue("props", linkedlist, typeof(string)); //to save data

public implDataStru(SerializationInfo info, StreamingContext context)//constructer to deserialize values

{

linkedlist = (LinkedList<T>)info.GetValue("props", typeof(string));

}

}}

**Program.cs**

public class Program

{

/\* [Serializable]

public class Player

{

public string Name;

public int Score;

public Player() { }

public Player(string name, int score)

{

Name = name;

Score = score;

}\*/

class DataSerializer

{

public void BinarySerialize(object data, string file)

{

FileStream fileStream;

BinaryFormatter bf = new BinaryFormatter();

if (File.Exists(file)) File.Delete(file);

fileStream = File.Create(file);

bf.Serialize(fileStream, data);

fileStream.Close();

}

public void BinaryDeSerialize(string file)

{

object obj = null;

FileStream fileStream;

BinaryFormatter bf = new BinaryFormatter();

if (File.Exists(file))

{

fileStream = File.OpenRead(file);

obj = bf.Deserialize(fileStream);

fileStream.Close();

}

}

}

static void Main(string[] args)

{

implDataStru<int> s = new implDataStru<int>();

s.Push(3);

s.Push(2);

s.Push(6);

implDataStru<int> s1 = (implDataStru<int>)s.Clone();

s1.Push(4);

foreach (int x in s) //1

Console.Write(x + " ");

Console.WriteLine("\n");

foreach (int x in s1)//2

Console.Write(x + " ");// by comparing 1 && 2 we realise that the list is cloned

}

}

}

**Tects.cs**

[TestFixture]

public class Tests

{

[Test()]

public void PeekTest()

{

implDataStru<int> s = new implDataStru<int>();

s.Push(1);

s.Push(2);

s.Push(3);

Assert.AreEqual(s.Peek(), 3);

}

[Test()]

public void PushTest()

{

implDataStru<int> s = new implDataStru<int>();

s.Push(1);

s.Push(2);

s.Push(3);

string actual = "";

string expected = "321";

foreach (var x in s)

actual += x.ToString();

Assert.AreEqual(actual, expected);

}

[Test()]

public void PopTest()

{

implDataStru<int> s = new implDataStru<int>();

s.Push(1);

s.Push(2);

s.Push(3);

Assert.AreEqual(s.Pop(), 3);

}

[Test()]

public void ClearTest()

{

implDataStru<int> s = new implDataStru<int>();

s.Push(1);

s.Push(2);

s.Push(3);

s.Clear();

int actual = s.Count();

int expected = 0;

Assert.AreEqual(actual, expected);

}

[Test()]

public void PushElementTest()

{

implDataStru<int> s = new implDataStru<int>();

s.Push(1);

s.Push(2);

s.Push(3);

s.Clear();

int actual = s.Count();

int expected = 0;

Assert.AreEqual(actual, expected);

}

[Test()]

public void PeekEmptyStackTest()

{

implDataStru<int> s = new implDataStru<int>();

Assert.Throws<InvalidOperationException>(() => { int actual = s.Peek(); });

}

[Test()]

public void LIFOTest()

{

implDataStru<int> s = new implDataStru<int>();

s.Push(1);

s.Push(2);

s.Push(3);

Assert.AreEqual(3, s.Pop());

Assert.AreEqual(2, s.Pop());

Assert.AreEqual(1, s.Pop());

Assert.Throws<InvalidOperationException>(() => { int actual = s.Pop(); }); // impossible to pop an element bcs the stack is already empty

}

[Test()]

public void CloneTest()

{

implDataStru<int> s = new implDataStru<int>();

s.Push(1);

s.Push(2);

s.Push(3);

implDataStru<int> s1 = (implDataStru<int>)s.Clone();

s1.Push(4);

Assert.AreNotEqual(s, s1);

}

[Test()]

public void ContainsElementTest()

{

implDataStru<int> s = new implDataStru<int>();

s.Push(1);

s.Push(2);

s.Push(3);

Assert.AreEqual(s.Contains(3), true);

}

[Test()]

public void DoesNotContainsElementTest()

{

implDataStru<int> s = new implDataStru<int>();

s.Push(1);

s.Push(2);

s.Push(3);

Assert.AreEqual(s.Contains(7), false);

}

[Test()]

public void SerializableToJsonTest()

{

implDataStru<int> s = new implDataStru<int>();

s.Push(1);

s.Push(2);

s.Push(3);

string actual = "";

string expected = "321";

foreach (var x in s)

actual += JsonConvert.SerializeObject(x).ToString();

Assert.AreEqual(actual, expected);

}

}

}

**Результат Работы :**

