**18-КБ-ПР2 Буниф Мажда**

**Отчёт по лабораторной работе №6**

1. **Наименование и цель работы**

**Наименование:** Модульное тестирование объектно-ориентированных программ

**Цель работы:** Изучить подход к автоматизации процесса модульного тестирования объектно-ориентированных программ.

1. **Задание на лабораторную работу**

1) Создать класс (в соответствии с вариантом задания из п.5), реализующий работу с АСД (абстрактная структура данных).

2) Построить диаграмму состояний класса.

3) Составить тестовые требования к методам класса не основе полученной диаграммы состояний.

4) Определить наборы тестов на основе полученных тестовых требований.

5) Создать проект для автоматизированного модульного тестирования на основе тестовых наборов.

6) Запустить тестирование и проверить результаты работы.

7) Составить отчет о результатах проведенного тестирования.

**3. Текст программы на алгоритмическом языке**

**Класс**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

public class Node<T>

{

public Node(T data)

{

Data = data;

}

public T Data { get; set; }

public Node<T> Next { get; set; }

}

public class QueueL<T> : IEnumerable<T>

{

Node<T> head; // the head of the 1st element

Node<T> tail; // the tail

int count;

// Adding in the queue

public void Enqueue(T data)

{

Node<T> node = new Node<T>(data);

Node<T> tempNode = tail;

tail = node;

if (count == 0)

head = tail;

else

tempNode.Next = tail;

count++;

}

// Deleting from the queue

public T Dequeue()

{

if (count == 0)

throw new InvalidOperationException();

T output = head.Data;

head = head.Next;

count--;

return output;

}

// Get THE 1ST ELEMENT

public T First

{

get

{

if (IsEmpty)

throw new InvalidOperationException();

return head.Data;

}

}

// Get the last element

public T Last

{

get

{

if (IsEmpty)

throw new InvalidOperationException();

return tail.Data;

}

}

public int Count { get { return count; } }

public bool IsEmpty { get { return count == 0; } }

public void Clear()

{

head = null;

tail = null;

count = 0;

}

public bool Contains(T data)

{

Node<T> current = head;

while (current != null)

{

if (current.Data.Equals(data))

return true;

current = current.Next;

}

return false;

}

IEnumerator IEnumerable.GetEnumerator()

{

return ((IEnumerable)this).GetEnumerator();

}

IEnumerator<T> IEnumerable<T>.GetEnumerator()

{

Node<T> current = head;

while (current != null)

{

yield return current.Data;

current = current.Next;

}

}

}

}

**Решение 2, тестовый класс**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ConsoleApp1;

using NUnit.Framework;

namespace ClassLibrary1

{

[TestFixture]

public class Class1

{

[Test()]

public void EnqueueTest()

{

QueueL<int> q = new QueueL<int>();

q.Enqueue(1);

q.Enqueue(2);

q.Enqueue(3);

string s = "";

foreach (var i in q)

s += i.ToString();

string expected = "123";

Assert.AreEqual(s, expected);

}

[Test()]

public void DequeueTest()

{

QueueL<int> q = new QueueL<int>();

q.Enqueue(1);

q.Enqueue(2);

q.Enqueue(3);

q.Dequeue();

string s = "";

foreach (var i in q)

s += i.ToString();

string expected = "23";

Assert.AreEqual(s, expected);

}

[Test()]

public void FirstPassTest()

{

QueueL<int> q = new QueueL<int>();

q.Enqueue(1);

q.Enqueue(2);

q.Enqueue(3);

int actual = q.First;

int expected = 1;

Assert.AreEqual(actual, expected);

}

[Test()]

public void FirstExcTest()

{

QueueL<int> q = new QueueL<int>();

int actual ;

Assert.Throws<InvalidOperationException>(

() => { actual = q.First(); });

}

[Test()]

public void LastPassTest()

{

QueueL<int> q = new QueueL<int>();

q.Enqueue(1);

q.Enqueue(2);

q.Enqueue(3);

int actual = q.Last;

int expected = 3;

Assert.AreEqual(actual, expected);

}

[Test()]

public void LastExcTest()

{

QueueL<int> q = new QueueL<int>();

int actual;

Assert.Throws<InvalidOperationException>(

() => { actual = q.Last(); });

}

[Test()]

public void CountTest()

{

QueueL<int> q = new QueueL<int>();

q.Enqueue(1);

q.Enqueue(2);

q.Enqueue(3);

int actual = q.Count;

int expected = 3;

Assert.AreEqual(actual, expected);

}

[Test()]

public void IsEmptyTest()

{

QueueL<int> q = new QueueL<int>();

q.Enqueue(1);

q.Enqueue(2);

q.Enqueue(3);

bool actual = q.IsEmpty;

bool expected = false;

Assert.AreEqual(actual, expected);

}

public void ClearTest()

{

QueueL<int> q = new QueueL<int>();

q.Enqueue(1);

q.Enqueue(2);

q.Enqueue(3);

q.Clear();

int actual = q.Count;

int expected = 0;

Assert.AreEqual(actual, expected);

}

[Test()]

public void ContainsTest()

{

QueueL<int> q = new QueueL<int>();

q.Enqueue(1);

q.Enqueue(2);

q.Enqueue(3);

bool actual = q.Contains(5);

bool expected = false;

Assert.AreEqual(actual, expected);

}

[Test()]

public void DequeueExcTest()

{

QueueL<int> q = new QueueL<int>();

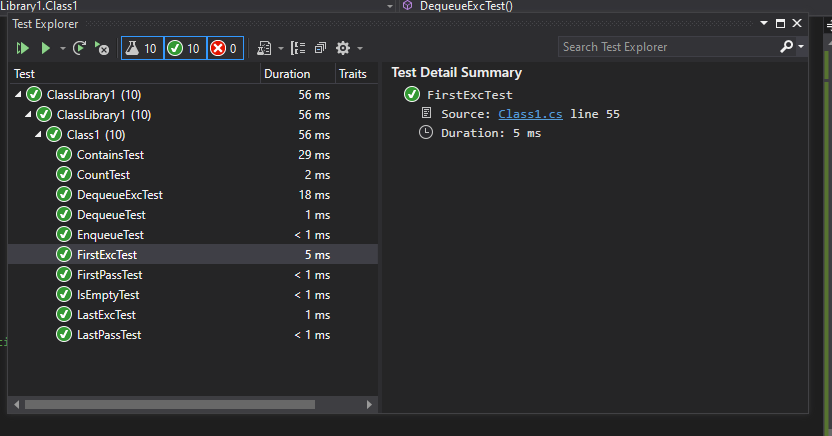
int actual;

Assert.Throws<InvalidOperationException>(

() => { actual = q.Dequeue(); });

}}}

1. **Результат тестов**

****