Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГТУ»)**

**Институт компьютерных систем и информационной безопасности**

**Кафедра информационных систем и программирования**

**ЛАБОРАТОРНАЯ РАБОТА №7**

**Дисциплина: тестирование и отладка программного обеспечения**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Я.С Давков

Направление подготовки: 09.03.04 Программная инженерия

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Г. Волик

**1 Цель работы**

Изучить подход к автоматизации процесса модульного тестирования объектно-ориентированных программ.

**2 Краткая теория**

Модульное тестирование – это тестирование программы на уровне отдельно взятых модулей, функций или классов. Цель модульного тестирования состоит в выявлении локализованных в модуле ошибок в реализации алгоритмов, а также в определении степени готовности системы к переходу на следующий уровень разработки и тестирования. Модульное тестирование проводится по принципу "белого ящика", то есть основывается на знании внутренней структуры программы, и часто включает те или иные методы анализа покрытия кода. Модульное тестирование обычно подразумевает создание вокруг каждого модуля определенной среды, включающей заглушки для всех интерфейсов тестируемого модуля. Некоторые из них могут использоваться для подачи входных значений, другие для анализа результатов, присутствие третьих может быть продиктовано требованиями, накладываемыми компилятором и сборщиком. На уровне модульного тестирования проще всего обнаружить дефекты, связанные с алгоритмическими ошибками и ошибками кодирования алгоритмов, типа работы с условиями и счетчиками циклов, а также с использованием локальных переменных и ресурсов. Ошибки, связанные с неверной трактовкой данных, некорректной реализацией интерфейсов, совместимостью, производительностью и т.п. обычно пропускаются на уровне модульного тестирования и выявляются на более поздних стадиях тестирования. Unit testing (юнит тестирование или модульное тестирование) — заключается в изолированной проверке каждого отдельного элемента путем запуска тестов в искусственной среде. Для этого необходимо использовать драйверы и заглушки. Поэлементное тестирование — первейшая возможность реализовать исходный код. Оценивая каждый элемент изолированно, и подтверждая корректность его работы, точно установить проблему значительно проще чем, если бы элемент был частью системы.

**3 Задание**

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

**4 Выполнение работы**

Мой вариант – 4, логическая структура – список, физическое представление – массив.

Листинг класса:

namespace ConsoleApp1;  
  
public class MyArrayList<T>  
{  
 private T[] \_items;  
 private int \_count;  
 private const int *DefaultCapacity* = 4;  
  
 public int Count => \_count;  
 public int Capacity => \_items.Length;  
  
 public MyArrayList()  
 {  
 \_items = new T[*DefaultCapacity*];  
 \_count = 0;  
 }  
  
 public MyArrayList(int capacity)  
 {  
 if (capacity < 0)  
 throw new ArgumentOutOfRangeException(nameof(capacity));  
 \_items = new T[capacity];  
 \_count = 0;  
 }  
  
 public void Add(T item)  
 {  
 if (\_count == \_items.Length)  
 Resize();  
 \_items[\_count++] = item;  
 }  
  
 public void Insert(int index, T item)  
 {  
 if (index < 0 || index > \_count)  
 throw new ArgumentOutOfRangeException(nameof(index));  
  
 if (\_count == \_items.Length)  
 Resize();  
  
 Array.Copy(\_items, index, \_items, index + 1, \_count - index);  
 \_items[index] = item;  
 \_count++;  
 }  
  
 public bool Remove(T item)  
 {  
 int index = IndexOf(item);  
 if (index >= 0)  
 {  
 RemoveAt(index);  
 return true;  
 }  
 return false;  
 }  
  
 public void RemoveAt(int index)  
 {  
 if (index < 0 || index >= \_count)  
 throw new ArgumentOutOfRangeException(nameof(index));  
  
 \_count--;  
 if (index < \_count)  
 Array.Copy(\_items, index + 1, \_items, index, \_count - index);  
 \_items[\_count] = default(T);  
 }  
  
 public T Get(int index)  
 {  
 if (index < 0 || index >= \_count)  
 throw new ArgumentOutOfRangeException(nameof(index));  
 return \_items[index];  
 }  
  
 public void Set(int index, T item)  
 {  
 if (index < 0 || index >= \_count)  
 throw new ArgumentOutOfRangeException(nameof(index));  
 \_items[index] = item;  
 }  
  
 public int IndexOf(T item)  
 {  
 return Array.IndexOf(\_items, item, 0, \_count);  
 }  
  
 public bool Contains(T item)  
 {  
 return IndexOf(item) >= 0;  
 }  
  
 public void Clear()  
 {  
 Array.Clear(\_items, 0, \_count);  
 \_count = 0;  
 }  
  
 private void Resize()  
 {  
 int newCapacity = \_items.Length \* 2;  
 T[] newArray = new T[newCapacity];  
 Array.Copy(\_items, newArray, \_count);  
 \_items = newArray;  
 }  
}

Сделаем диаграмму состояния классов

Изображение выглядит как линия, диаграмма, снимок экрана, График

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 1 – Диаграмма состояния классов

Составим тестовые требования:

1. Создание пустого списка (конструктор по умолчанию)
2. Создание списка с заданной емкостью
3. Добавление элемента в пустой список
4. Добавление элемента в частично заполненный список
5. Добавление элемента с автоматическим расширением
6. Вставка элемента в начало списка
7. Вставка элемента в середину списка
8. Вставка элемента в конец списка
9. Удаление существующего элемента
10. Удаление несуществующего элемента
11. Удаление по индексу (валидному)
12. Удаление по индексу (невалидному - исключение)
13. Получение элемента по валидному индексу
14. Получение элемента по невалидному индексу (исключение)
15. Установка значения по валидному индексу
16. Поиск существующего элемента
17. Поиск несуществующего элемента
18. Очистка непустого списка
19. Проверка Contains для существующего элемента
20. Проверка Contains для несуществующего элемента

Составим наборы тестов:

[TestFixture]  
public class ArrayListTests  
{  
 [Test]  
 public void Constructor\_Default\_CreatesEmptyList()  
 {  
 var list = new MyArrayList<int>();  
 Assert.That(list.Count, Is.EqualTo(0));  
 Assert.That(list.Capacity, Is.GreaterThanOrEqualTo(0));  
 }  
  
 [Test]  
 public void Constructor\_WithCapacity\_CreatesListWithSpecifiedCapacity()  
 {  
 var list = new MyArrayList<int>(10);  
 Assert.That(list.Count, Is.EqualTo(0));  
 Assert.That(list.Capacity, Is.EqualTo(10));  
 }  
  
 [Test]  
 public void Constructor\_NegativeCapacity\_ThrowsException()  
 {  
 Assert.Throws<ArgumentOutOfRangeException>(() => new MyArrayList<int>(-1));  
 }  
  
 [Test]  
 public void Add\_ToEmptyList\_IncreasesCount()  
 {  
 var list = new MyArrayList<int>();  
 list.Add(5);  
 Assert.That(list.Count, Is.EqualTo(1));  
 Assert.That(list.Get(0), Is.EqualTo(5));  
 }  
  
 [Test]  
 public void Add\_MultipleElements\_MaintainsOrder()  
 {  
 var list = new MyArrayList<int>();  
 list.Add(1);  
 list.Add(2);  
 list.Add(3);  
  
 Assert.That(list.Count, Is.EqualTo(3));  
 Assert.That(list.Get(0), Is.EqualTo(1));  
 Assert.That(list.Get(1), Is.EqualTo(2));  
 Assert.That(list.Get(2), Is.EqualTo(3));  
 }  
  
 [Test]  
 public void Add\_ExceedingCapacity\_AutomaticallyResizes()  
 {  
 var list = new MyArrayList<int>(2);  
 list.Add(1);  
 list.Add(2);  
 int oldCapacity = list.Capacity;  
 list.Add(3);  
  
 Assert.That(list.Capacity, Is.GreaterThan(oldCapacity));  
 Assert.That(list.Count, Is.EqualTo(3));  
 }  
  
 [Test]  
 public void Insert\_AtBeginning\_ShiftsElements()  
 {  
 var list = new MyArrayList<int>();  
 list.Add(2);  
 list.Add(3);  
 list.Insert(0, 1);  
  
 Assert.That(list.Get(0), Is.EqualTo(1));  
 Assert.That(list.Get(1), Is.EqualTo(2));  
 Assert.That(list.Count, Is.EqualTo(3));  
 }  
  
 [Test]  
 public void Insert\_AtMiddle\_ShiftsElements()  
 {  
 var list = new MyArrayList<int>();  
 list.Add(1);  
 list.Add(3);  
 list.Insert(1, 2);  
  
 Assert.That(list.Get(1), Is.EqualTo(2));  
 Assert.That(list.Get(2), Is.EqualTo(3));  
 }  
  
 [Test]  
 public void Insert\_AtEnd\_AppendsElement()  
 {  
 var list = new MyArrayList<int>();  
 list.Add(1);  
 list.Insert(1, 2);  
  
 Assert.That(list.Get(1), Is.EqualTo(2));  
 Assert.That(list.Count, Is.EqualTo(2));  
 }  
  
 [Test]  
 public void Insert\_InvalidIndex\_ThrowsException()  
 {  
 var list = new MyArrayList<int>();  
 Assert.Throws<ArgumentOutOfRangeException>(() => list.Insert(-1, 1));  
 Assert.Throws<ArgumentOutOfRangeException>(() => list.Insert(5, 1));  
 }  
  
 [Test]  
 public void Remove\_ExistingElement\_ReturnsTrue()  
 {  
 var list = new MyArrayList<int>();  
 list.Add(1);  
 list.Add(2);  
  
 bool result = list.Remove(1);  
  
 Assert.That(result, Is.True);  
 Assert.That(list.Count, Is.EqualTo(1));  
 Assert.That(list.Get(0), Is.EqualTo(2));  
 }  
  
 [Test]  
 public void Remove\_NonExistingElement\_ReturnsFalse()  
 {  
 var list = new MyArrayList<int>();  
 list.Add(1);  
  
 bool result = list.Remove(5);  
  
 Assert.That(result, Is.False);  
 Assert.That(list.Count, Is.EqualTo(1));  
 }  
  
 [Test]  
 public void RemoveAt\_ValidIndex\_RemovesElement()  
 {  
 var list = new MyArrayList<int>();  
 list.Add(1);  
 list.Add(2);  
 list.Add(3);  
  
 list.RemoveAt(1);  
  
 Assert.That(list.Count, Is.EqualTo(2));  
 Assert.That(list.Get(1), Is.EqualTo(3));  
 }  
  
 [Test]  
 public void RemoveAt\_InvalidIndex\_ThrowsException()  
 {  
 var list = new MyArrayList<int>();  
 list.Add(1);  
  
 Assert.Throws<ArgumentOutOfRangeException>(() => list.RemoveAt(-1));  
 Assert.Throws<ArgumentOutOfRangeException>(() => list.RemoveAt(5));  
 }  
  
 [Test]  
 public void Get\_ValidIndex\_ReturnsElement()  
 {  
 var list = new MyArrayList<int>();  
 list.Add(42);  
  
 Assert.That(list.Get(0), Is.EqualTo(42));  
 }  
  
 [Test]  
 public void Get\_InvalidIndex\_ThrowsException()  
 {  
 var list = new MyArrayList<int>();  
 Assert.Throws<ArgumentOutOfRangeException>(() => list.Get(0));  
 }  
  
 [Test]  
 public void Set\_ValidIndex\_UpdatesElement()  
 {  
 var list = new MyArrayList<int>();  
 list.Add(1);  
 list.Set(0, 10);  
  
 Assert.That(list.Get(0), Is.EqualTo(10));  
 }  
  
 [Test]  
 public void Set\_InvalidIndex\_ThrowsException()  
 {  
 var list = new MyArrayList<int>();  
 Assert.Throws<ArgumentOutOfRangeException>(() => list.Set(0, 1));  
 }  
  
 [Test]  
 public void IndexOf\_ExistingElement\_ReturnsIndex()  
 {  
 var list = new MyArrayList<int>();  
 list.Add(1);  
 list.Add(2);  
  
 Assert.That(list.IndexOf(2), Is.EqualTo(1));  
 }  
  
 [Test]  
 public void IndexOf\_NonExistingElement\_ReturnsMinusOne()  
 {  
 var list = new MyArrayList<int>();  
 list.Add(1);  
  
 Assert.That(list.IndexOf(5), Is.EqualTo(-1));  
 }  
  
 [Test]  
 public void Contains\_ExistingElement\_ReturnsTrue()  
 {  
 var list = new MyArrayList<int>();  
 list.Add(1);  
  
 Assert.That(list.Contains(1), Is.True);  
 }  
  
 [Test]  
 public void Contains\_NonExistingElement\_ReturnsFalse()  
 {  
 var list = new MyArrayList<int>();  
 list.Add(1);  
  
 Assert.That(list.Contains(5), Is.False);  
 }  
  
 [Test]  
 public void Clear\_NonEmptyList\_EmptiesList()  
 {  
 var list = new MyArrayList<int>();  
 list.Add(1);  
 list.Add(2);  
  
 list.Clear();  
  
 Assert.That(list.Count, Is.EqualTo(0));  
 }  
  
 [Test]  
 public void Clear\_EmptyList\_RemainsEmpty()  
 {  
 var list = new MyArrayList<int>();  
 list.Clear();  
  
 Assert.That(list.Count, Is.EqualTo(0));  
 }  
  
 [TestCase(0)]  
 [TestCase(1)]  
 [TestCase(10)]  
 [TestCase(100)]  
 public void Add\_DifferentAmounts\_WorksCorrectly(int count)  
 {  
 var list = new MyArrayList<int>();  
 for (int i = 0; i < count; i++)  
 {  
 list.Add(i);  
 }  
  
 Assert.That(list.Count, Is.EqualTo(count));  
 }  
  
 [TestCase(0, 5)]  
 [TestCase(1, 5)]  
 [TestCase(2, 5)]  
 public void Insert\_AtDifferentPositions\_WorksCorrectly(int index, int value)  
 {  
 var list = new MyArrayList<int>();  
 list.Add(1);  
 list.Add(2);  
 list.Add(3);  
  
 list.Insert(index, value);  
  
 Assert.That(list.Get(index), Is.EqualTo(value));  
 }  
}

Запустим и проверим что все тесты выполняются.

Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 2 – результат запуска тестов