# Лабораторная работа № 20

**Тема:** Разработка программ на языке С# с использованием коллекций и методов работы с ними

1. Формирование навыков разработки программ на языке С# с использованием коллекций.
2. Изучение методов работы с коллекциями на языке С#.

. **Оборудование:**

1. ПК
2. Программное обеспечение: ОС Windows, среда Visual Studio.Net

**Контрольные вопросы:**

1. Коллекции в C# представляют собой контейнеры для хранения и управления группами объектов или значений. Они используются для различных целей, включая управление, хранение, поиск, сортировку и обработку данных.

2. Отличия между коллекциями и массивами включают:

- Размер: Массив имеет фиксированный размер, тогда как коллекции могут динамически расти.

- Гибкость: Коллекции могут включать различные типы данных, в то время как массивы ограничены одним типом данных.

- Удобство использования: Коллекции обычно предоставляют более удобные методы и функции для работы с элементами.

- Управление памятью: Массивы в C# хранят свои элементы в управляемой куче памяти, в то время как коллекции могут использовать разные методы для управления памятью.

3. В C# имеется большой набор классов коллекций, включающий в себя, например, List<T>, LinkedList<T>, Queue<T>, Stack<T>, Dictionary<TKey, TValue> и другие.

4. В С# есть несколько классов коллекций общего назначения, таких как List<T>, ArrayList, LinkedList<T>, Queue<T> и Stack<T>. Эти классы предоставляют гибкие и эффективные способы работы с данными.5. Класс Stack<T> представляет собой коллекцию, которая работает по принципу "последним вошел - первым вышел" (LIFO). Он используется для хранения и управления элементами в стеке. Некоторые методы этой коллекции включают Push, Pop, Peek и Clear, которые позволяют добавлять элементы в стек, извлекать их, получать верхний элемент и очищать стек соответственно.

6. Класс List<T> представляет собой динамический массив, который может хранить объекты определенного типа T. Он позволяет добавлять, удалять, вставлять и изменять элементы. List<T> может автоматически увеличивать свой размер при добавлении элементов, и он предоставляет удобные методы для выполнения различных операций с элементами, таких как Add, Remove, Insert, Clear, Sort и другие.

7. Класс LinkedList<T> представляет собой двусвязный список, который состоит из узлов, содержащих элементы типа T. В отличие от List<T>, LinkedList<T> позволяет эффективно вставлять или удалять элементы в середине списка. Класс LinkedList<T> предоставляет методы для добавления, удаления, вставки, поиска элементов, а также для перебора элементов списка.

8. Класс Queue<T> представляет собой коллекцию, которая работает по принципу "первым вошел - первым вышел" (FIFO). Он используется для хранения и управления элементами в очереди. Некоторые методы этой коллекции включают Enqueue, Dequeue, Peek и Clear, которые позволяют добавлять элементы в очередь, извлекать их, получать первый элемент и очищать очередь соответственно.

**Ход работы:**

1. В соответствии с вариантом задания для лабораторной работы №16-17 из объектов Вашей иерархии классов создать коллекцию ArrayList, список, очередь или стек (по вариантам: 1 - 5 варианты ArrayList; 6 - 10 варианты список; 11 – 15 варианты очередь; 16 – 20 варианты стек; 21-25 варианты словари) и продемонстрировать методы работы с этой коллекцией (вставка элементов, удаление, поиск и т.д.). Разработать алгоритм задачи и представить его в виде  программы на С#
2. Используя средства Visual Studio.Net создать файл с программой , выполнить тестирование и отладку программу.
3. Результаты представить в виде отчета
4. Сделать вывод о проделанной работе

**Варианты заданий для лабораторной работы №20**

Двигатель, дизель, двигатель внутреннего сгорания, реактивный двигатель.

**Листинг с исходным кодом**

using System;

using System.Collections;

using System.Collections.Generic;

abstract class Двигатель

{

public abstract void запуск();

public virtual void остановка()

{

Console.WriteLine("Двигатель остановлен");

}

public virtual void информация()

{

Console.WriteLine("Информация о двигателе");

}

}

class Дизель : Двигатель

{

private int мощность;

public Дизель(int \_мощность)

{

мощность = \_мощность;

}

public override void запуск()

{

Console.WriteLine("Дизель запущен");

}

public override void остановка()

{

Console.WriteLine("Дизель остановлен");

}

public override void информация()

{

Console.WriteLine("Информация о дизеле. Мощность: " + мощность);

}

}

class ДвигательВнутреннегоСгорания : Двигатель

{

private int обороты;

public ДвигательВнутреннегоСгорания(int \_обороты)

{

обороты = \_обороты;

}

public override void запуск()

{

Console.WriteLine("Двигатель внутреннего сгорания запущен");

}

public override void остановка()

{

Console.WriteLine("Двигатель внутреннего сгорания остановлен");

}

public override void информация()

{

Console.WriteLine("Информация о двигателе внутреннего сгорания. Обороты: " + обороты);

}

}

class РеактивныйДвигатель : Двигатель

{

private int тяга;

public РеактивныйДвигатель(int \_тяга)

{

тяга = \_тяга;

}

public override void запуск()

{

Console.WriteLine("Реактивный двигатель запущен");

}

public override void остановка()

{

Console.WriteLine("Реактивный двигатель остановлен");

}

public override void информация()

{

Console.WriteLine("Информация о реактивном двигателе. Тяга: " + тяга);

}

}

class Program

{

static void Main(string[] args)

{

Двигатель[] двигатели = new Двигатель[3];

двигатели[0] = new Дизель(100);

двигатели[1] = new ДвигательВнутреннегоСгорания(2000);

двигатели[2] = new РеактивныйДвигатель(5000);

Queue<Двигатель> очередь = new Queue<Двигатель>(); // Создаем очередь типа Двигатель

очередь.Enqueue(new Дизель(200)); // Добавление элемента 200 в конец очереди

очередь.Enqueue(new ДвигательВнутреннегоСгорания(3000));// Добавление элемента 3000 в конец очереди

очередь.Enqueue(new РеактивныйДвигатель(6000));// Добавление элемента 6000 в конец очереди

while (очередь.Count > 0)//это условие, которое проверяет, что количество элементов в очереди (Count) больше нуля.

{

Двигатель двигатель = очередь.Dequeue();//Метод Dequeue() извлекает и возвращает первый элемент очереди, одновременно удаляя его из очереди.

двигатель.запуск();

двигатель.информация();

двигатель.остановка();

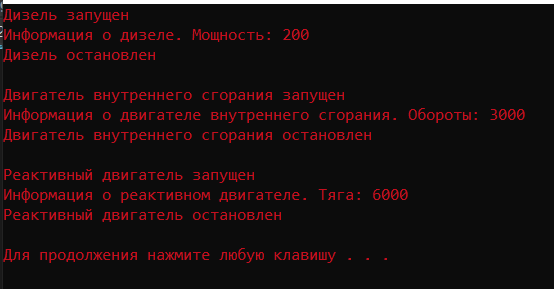
Console.WriteLine();

}

}

}

**Результаты тестирования**

****

**Вывод по работе.**

Программа работает корректно