Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

Методические к лабораторным работам по дисциплине “Разработка приложений на языке C#”

2022 г.

СОДЕРЖАНИЕ

Лабораторная работа №1. Основы языка C#

Теоретическая часть:

Microsoft® .NET – это платформа нового поколения для создания распределенных бизнес-приложений. Основа .Net – это Microsoft .Net Framework – своеобразный верстак, набор средств и технологий по разработке и выполнению таких приложений. .NET имеет в себе общеязыковую среду времени выполнения(CLR, common language runtime). Любой программный код, написанный под новую платформу называется управляемым (managed code) и компилируется в бинарный вид, понятный .NET runtime. Этот формат называется Microsoft Intermediate Language (MSIL, IL). В связи с этим появляется возможность легкой интеграции кодов, написанных на разных языках программирования, т.к. платформа более низкого уровня у них одна.

В среду включен автоматический подсчет ссылок и сборщик мусора. Среда предоставляет расширенные возможности по управлению безопасностью кода, имеет стандартные возможности по работе с различными версиями одних и тех же компонент (versioning).

В платформу .Net входят много языков, которые без особого труда интегрируются друг с другом. В данных ЛР будет рассматриваться язык C#, так как именно этот язык был создан специально для данной платформы, прочие языки были лишь адаптированы для данной платформы. Язык C# схож с языками C++ и Java, но, безусловно, имеет и отличия.

Пример программы на C#:

```
class TestClass
{
    static void Main(string[] args)
    {
        // Display the number of command line arguments.
        Console.WriteLine(args.Length);
    }
}
```

Метод Main — это точка входа приложения C#. (Библиотекам и службам точка входа в виде метода Main не требуется.) Когда приложение запускается, первым вызывается именно метод Main.

В программе на C# может существовать только одна точка входа. Если у вас есть несколько классов с методом Main, программу нужно компилировать с параметром компилятора **StartupObject**, чтобы указать, какой из методов Main будет использоваться в качестве точки входа.

Если вы включаете директивы using, они должны быть первыми в файле.

Система типов C#:

C# является строго типизированным языком. Каждая переменная и константа имеет тип, как и каждое выражение, результатом вычисления которого является значение. Каждое объявление метода задает имя, тип и вид (значение, ссылка или вывод) для каждого входного параметра и для возвращаемого значения.

В библиотеке классов .NET определены встроенные числовые типы и комплексные типы, представляющие разнообразные конструкции. К ним относятся файловая система, сетевые подключения, коллекции и массивы объектов, а также даты. Обычная программа на C# использует типы из этой библиотеки классов и пользовательские типы, которые моделируют уникальные концепции конкретной сферы применения.

Когда вы объявляете в программе переменную или константу, для нее нужно задать тип либо использовать ключевое слово *var*, чтобы компилятор определил тип самостоятельно. В следующем примере показаны некоторые объявления переменных, использующие встроенные числовые типы и сложные пользовательские типы:

```
// Declaration only:
float temperature;
string name;
MyClass myClass;

// Declaration with initializers (four examples):
char firstLetter = 'C';
var limit = 3;
int[] source = { 0, 1, 2, 3, 4, 5 };
var query = from item in source
            where item <= limit
            select item;
```

C# предоставляет стандартный набор встроенных типов. Они используются для представления целых чисел, значений с плавающей запятой, логических выражений, текстовых символов, десятичных значений и других типов данных. Также существуют встроенные типы string и object.

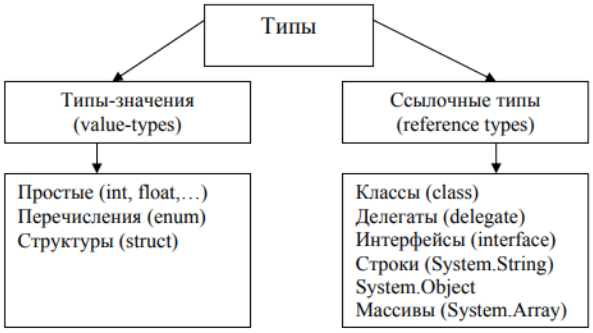

Рисунок 1 – Типы в C#

Действия программы выражаются с помощью *операторов*. C# поддерживает несколько типов операторов, некоторые из которых определяются как внедренные операторы.

- С помощью *блоков* можно использовать несколько операторов в таких контекстах, где ожидается только один оператор. Блок состоит из списка инструкций, заключенных между разделителями { и }.
- *Операторы объявления* используются для объявления локальных переменных и констант.
- *Операторы выражений* позволяют вычислять выражения. В качестве оператора можно использовать такие выражения, как вызовы методов, выделение объектов с помощью оператора new, назначения с помощью = и составных операторов присваивания, операторы ++ и -- для приращения и уменьшения, а также выражения await.
- *Операторы выбора* используются для выбора одного оператора из нескольких возможных вариантов в зависимости от значения какого-либо выражения. Эта группа содержит операторы if и switch.
- *Операторы итерации* используются для многократного выполнения внедренного оператора. Эта группа содержит операторы while, do, for и foreach.
- *Операторы перехода* используются для передачи управления. Эта группа содержит операторы break, continue, goto, throw, return и yield.
- Операторы try...catch позволяют перехватывать исключения, создаваемые при выполнении блока кода, а оператор try...finally используется для указания кода завершения, который выполняется всегда, независимо от появления исключений.
- Операторы checked и unchecked операторы позволяют управлять контекстом проверки переполнения для целочисленных арифметических операций и преобразований.
- Оператор lock позволяет создать взаимоисключающую блокировку заданного объекта перед выполнением определенных операторов, а затем снять блокировку.
- Оператор using используется для получения ресурса перед определенным оператором, и для удаления ресурса после его завершения.

Задание:

Создать консольное приложение – калькулятор. Необходимо обеспечить защиту от неправильного ввода, выбор операций, а также защиту от деления на ноль.

Лабораторная работа №2. Массивы

Теоретическая часть:

Массив — это структура данных, содержащая ряд переменных, к которым осуществляется доступ с помощью вычисляемых индексов. Все содержащиеся в массиве переменные, также называемые *элементами массива*, относятся к одному типу. Этот тип называется *типом элемента* массива.

Сами массивы имеют ссылочный тип, и объявление переменной массива только выделяет память для ссылки на экземпляр массива. Фактические экземпляры массива создаются динамически во время выполнения с помощью оператора new.

Операция new задает *длину* нового экземпляра массива, который затем фиксируется на время существования экземпляра. Элементы массива имеют индексы в диапазоне от 0 до Length - 1. Оператор new автоматически инициализирует все элементы массива значением по умолчанию. Например, для всех числовых типов устанавливается нулевое значение, а для всех ссылочных типов — значение null.

Следующий пример кода создает массив из int элементов, затем инициализирует этот массив и выводит содержимое массива.

```
int[] a = new int[10];
for (int i = 0; i < a.Length; i++)
{
    a[i] = i * i;
}
for (int i = 0; i < a.Length; i++)
{
    Console.WriteLine($"a[{i}] = {a[i]}");
}
```

Кроме этого, C# поддерживает *многомерные массивы*. Число измерений типа массива, также известное как *ранг* типа массива, равно одному плюс числу запятых между квадратными скобками типа массива. Следующий пример кода поочередно создает одномерный, двухмерный и трехмерный массивы.

```
int[] a1 = new int[10];
int[,] a2 = new int[10, 5];
int[,,] a3 = new int[10, 5, 2];
```

Элементы массива могут иметь любой тип, в том числе тип массива. Следующий пример создает массив массивов int.

```
int[][] a = new int[3][];
a[0] = new int[10];
a[1] = new int[5];
a[2] = new int[20];
```

Оператор new разрешает указывать начальные значения элементов массива с помощью *инициализатора массива*, который представляет собой список выражений, написанных между разделителями { и }.

Задание 1: даны два массива a и b размерностью n и m соответственно, сформировать массив c таким образом, что первая часть — отсортированный по возрастанию массив a, а вторая часть — отсортированный по убыванию массив b.

Задание 2: создать двумерный массив, размерность задается пользователем, заполнить его случайными числами в диапазоне от 0 до 9. Отсортировать элементы массива по возрастанию вначале по строкам, а затем по столбцам. Вывести на экран исходный массив, массив отсортированный по-строчно, массив отсортированный по столбцам.

Лабораторная работа №3. Классы

Теоретическая часть:

Классы. Классы в C# имеют тот же смысл, что и в языках C++ и Java.

Формально класс описывается следующим образом:

```
модификатор-доступа class Имя-класса {
    … описание данных и методов …
}
```

Следующий код является простым примером объявления класса с именем Point:

```
public class Point
{
    public int X { get; }
    public int Y { get; }

    public Point(int x, int y) => (X, Y) = (x, y);
}
```

Экземпляры классов создаются с помощью оператора new, который выделяет память для нового экземпляра, вызывает конструктор для инициализации этого экземпляра и возвращает ссылку на экземпляр.

Некоторые методы и свойства специально предназначены для того, чтобы их вызов или доступ к ним осуществлялся из *клиентского кода*, то есть из кода за пределами этого класса или структуры. Другие методы и свойства могут использоваться только в самом классе или структуре.

Важно ограничить доступность кода так, чтобы только нужные элементы клиентского кода получали к нему доступ. Уровень доступности для типов и их элементов вы можете задать с помощью следующих модификаторов доступа.

- public
- protected
- internal
- protected internal
- private
- private protected.

По умолчанию используется режим доступа private.

Задание:

Создать консольное приложение. Программа представляет собой автоматизированную систему учета банковских сведений.

На каждого клиента банка хранятся следующие сведения:

− Ф.И.О.;

− Возраст;

− Место работы;

− Номера счетов.

На каждом счете хранится информация о текущем балансе и история прихода, расхода. Для каждого клиента может быть создано неограниченное количество счетов. С каждым счетом можно производить следующие действия: открытие, закрытие, вклад денег, снятие денег, просмотр баланса, просмотр истории.

Вся информация должна хранится в массивах. Рекомендуется объекты клиента и счета реализовать в виде классов. Баланс счета организовать в виде свойства только для чтения.

Лабораторная работа №4. Windows Forms

Теоретическая часть:

Windows Forms — это платформа пользовательского интерфейса для создания классических приложений Windows. Она обеспечивает один из самых эффективных способов создания классических приложений с помощью визуального конструктора в Visual Studio. Такие функции, как размещение визуальных элементов управления путем перетаскивания, упрощают создание классических приложений.

В Windows Forms можно разрабатывать графически сложные приложения, которые просто развертывать, обновлять, и с которыми удобно работать как в автономном режиме, так и в сети. Приложения Windows Forms могут получать доступ к локальному оборудованию и файловой системе компьютера, на котором работает приложение.

На приведенном ниже рисунке показан шаблон проекта Windows Forms с C#:

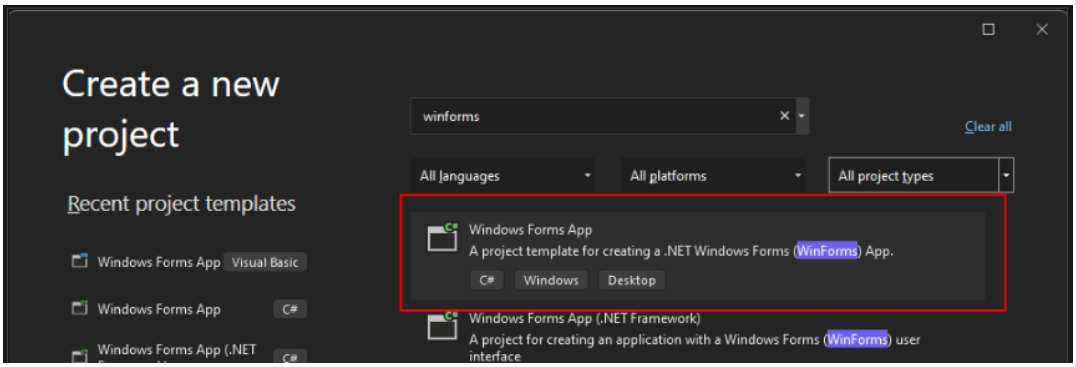

Рисунок 2 – Шаблон проекта

Поддержка Windows Forms в Visual Studio состоит из четырех важных компонентов, с которыми вы будете взаимодействовать при создании приложения.

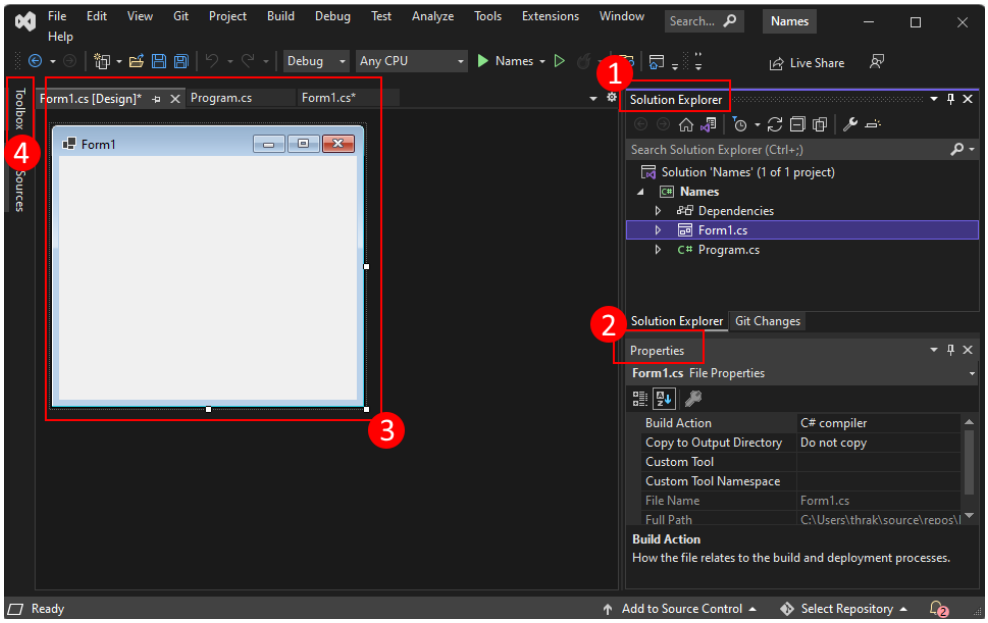

Рисунок 3 – Компоненты Windows Forms

1. Обозреватель решений

   Все файлы проекта, код, формы и ресурсы отображаются в этой области.
2. Properties (Свойства)

На этой панели отображаются параметры свойств, которые можно настроить в зависимости от выбранного элемента. Например, если выбрать элемент в **Обозревателе решений**, отобразятся параметры свойств, связанные с файлом. Если выбрать объект в **конструкторе**, отобразятся параметры элемента управления или формы.

3. Конструктор форм

Это конструктор для формы. Он является интерактивным, и на него можно перетаскивать объекты из **панели элементов**. Выбирая и перемещая элементы в конструкторе, можно визуально создавать пользовательский интерфейс для приложения.

4. Панель элементов

Панель элементов содержит все элементы управления, которые можно добавить на форму. Чтобы добавить элемент управления на текущую форму, дважды щелкните элемент управления или перетащите его.

Задание:

Создать интерфейс к программе, созданной в лабораторной работе 3. Ввод сведений о новых клиентах и счетах реализовать через дополнительные формы.

Лабораторная работа №5. Файлы

Задание:

Разработать программу, реализующую работу с файлами.

1. Программа должна быть разработана в виде приложения Windows Forms на языке C#. По желанию вместо Windows Forms возможно использование WPF (Windows Presentation Foundation).

2. Добавить кнопку, реализующую функцию чтения текстового файла в список слов List<string>.

3. Для выбора имени файла используется класс OpenFileDialog, который открывает диалоговое окно с выбором файла. Ограничить выбор только файлами с расширением «.txt».

4. Для чтения из файла рекомендуется использовать статический метод ReadAllText() класса File (пространство имен System.IO). Содержимое файла считывается методом ReadAllText() в виде одной строки, далее делится на слова с использованием метода Split() класса string. Слова сохраняются в список List<string>.

5. При сохранении слов в список List<string> дубликаты слов не записываются. Для проверки наличия слова в списке используется метод Contains().

6. Вычислить время загрузки и сохранения в список с использованием класса Stopwatch (пространство имен System.Diagnostics). Вычисленное время вывести на форму в поле ввода (TextBox) или надпись (Label).

7. Добавить на форму поле ввода для поиска слова и кнопку поиска. При нажатии на кнопку поиска осуществлять поиск введенного слова в списке Слово считается найденным, если оно входит в элемент списка как подстрока (метод Contains() класса string).

8. Добавить на форму список (ListBox). Найденные слова выводить в список с использованием метода «название_списка.Items.Add()». Вызовы метода «название_списка.Items.Add()» должны находится между вызовами методов «название_списка.BeginUpdate()» и «название_списка. EndUpdate()».

9. Вычислить время поиска с использованием класса Stopwatch. Вычисленное время вывести на форму в поле ввода (TextBox) или надпись (Label).

Лабораторная работа №6. Делегаты

Теоретическая часть:

Делегат — это тип, который представляет ссылки на методы с определенным списком параметров и типом возвращаемого значения. При создании экземпляра делегата этот экземпляр можно связать с любым методом с совместимой сигнатурой и типом возвращаемого значения. Метод можно вызвать (активировать) с помощью экземпляра делегата.

Делегаты используются для передачи методов в качестве аргументов к другим методам. Обработчики событий — это ничто иное, как методы, вызываемые с помощью делегатов. При создании пользовательского метода класс (например, элемент управления Windows) может вызывать этот метод при появлении определенного события. В следующем примере показано объявление делегата:

```
public delegate int PerformCalculation(int x, int y);
```

Делегату можно назначить любой метод из любого доступного класса или структуры, соответствующей типу делегата. Этот метод должен быть статическим методом или методом экземпляра. Такая гибкость позволяет программно изменять вызовы метода, а также включать новый код в существующие классы.

Делегаты имеют следующие свойства.

- Делегаты подобны указателям на функции в C++, но являются полностью объектно-ориентированными и, в отличие от указателей C++ на функции-члены, инкапсулируют экземпляр объекта вместе с методом.
- Делегаты допускают передачу методов в качестве параметров.
- Делегаты можно использовать для определения методов обратного вызова.
- Делегаты можно связывать друг с другом; например, при появлении одного события можно вызывать несколько методов.
- Точное соответствие методов типу делегата не требуется. Для краткой записи встроенных блоков кода введены лямбда-выражения. В результате компиляции лямбда-выражений (в определенном контексте) получаются типы делегатов.

Задание:

Разработать программу, использующую делегаты.

1. Программа должна быть разработана в виде консольного приложения на языке C#.

2. Определите делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа.

3. Напишите метод, соответствующий данному делегату.

4. Напишите метод, принимающий разработанный Вами делегат, в качестве одного из входных параметров. Осуществите вызов метода, передавая в качестве параметра-делегата:

· метод, разработанный в пункте 3;

· лямбда-выражение.

5. Повторите пункт 4, используя вместо разработанного Вами делегата, обобщенный делегат Func< > или Action< >, соответствующий сигнатуре разработанного Вами делегата.

Лабораторная работа №6. Рефлексия

Теоретическая часть:

Механизм отражения позволяет получать объекты (типа Type), которые описывают сборки, модули и типы. Отражение можно использовать для динамического создания экземпляра типа, привязки типа к существующему объекту, а также получения типа из существующего объекта и вызова его методов или доступа к его полям и свойствам. Если в коде используются атрибуты, отражение обеспечивает доступ к ним.

Вот простой пример отражения, в котором для получения типа переменной используется метод GetType(), наследуемый всеми типами от базового класса Object:

```
// Using GetType to obtain type information:
int i = 42;
Type type = i.GetType();
Console.WriteLine(type);
```

Выходные данные: *System.Int32*.

В этом примере отражение используется для получения полного имени загруженной сборки.

```
// Using Reflection to get information of an Assembly:
Assembly info = typeof(int).Assembly;
Console.WriteLine(info);
```

Выходные данные: *System.Private.CoreLib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=7cec85d7bea7798e.*

Отражение удобно использовать в следующих ситуациях:

- При необходимости доступа к атрибутам в метаданных программы.
- Для проверки и создания экземпляров типов в сборке.
- Для создания типов во время выполнения.
- Для выполнения позднего связывания, которое обеспечивает доступ к методам в типах, созданных во время выполнения.

Задание:

Разработать программу, реализующую работу с рефлексией.

1. Программа должна быть разработана в виде консольного приложения на языке C#.

2. Создайте класс, содержащий конструкторы, свойства, методы.

3. С использованием рефлексии выведите информацию о конструкторах, свойствах, методах.

4. Создайте класс атрибута (унаследован от класса System.Attribute).

5. Назначьте атрибут некоторым свойствам классам. Выведите только те свойства, которым назначен атрибут.

6. Вызовите один из методов класса с использованием рефлексии

Лабораторная работа №8. Исключения

Теоретическая часть:

Функции обработки исключений в языке C# помогают вам справиться с непредвиденными или исключительными проблемами, которые возникают при выполнении программы. При обработке исключений используются ключевые слова try, catch и finally для действий, которые могут оказаться неудачными. Это позволяет обрабатывать ошибки так, как кажется разумным, а также правильно высвобождать ресурсы. Исключения могут создаваться средой выполнения (CLR), платформой .NET , библиотеками сторонних поставщиков или кодом самого приложения. Чтобы создать исключение, используйте ключевое слово throw.

Во многих случаях исключение может создаваться не тем методом, который вызывается в вашем коде, а одним из последующих методов в стеке вызовов. Если создается такое исключение, среда CLR разворачивает стек, находит метод с блоком catch для исключений соответствующего типа и выполняет первый такой обнаруженный блок catch. Если подходящий блок catch не будет обнаружен во всем стеке вызовов, среда CLR завершает процесс и выводит сообщение для пользователя.

В этом примере метод выполняет проверку деления на нуль и перехватывает ошибку. Если не использовать обработку исключений, такая программа завершит работу с ошибкой **DivideByZeroException was unhandled** (Исключение DivideByZero не обработано).

```
public class ExceptionTest
{
    static double SafeDivision(double x, double y)
    {
        if (y == 0)
            throw new DivideByZeroException();
        return x / y;
    }

    public static void Main()
    {
        // Input for test purposes. Change the values to see
        // exception handling behavior.
        double a = 98, b = 0;
        double result;

        try
        {
            result = SafeDivision(a, b);
            Console.WriteLine("{0} divided by {1} = {2}", a, b, result);
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine("Attempted divide by zero.");
        }
    }
}
```

Исключения имеют следующие свойства.

- Исключения представляют собой типы, производные в конечном счете от System.Exception.
- Используйте блок try для выполнения таких инструкций, которые могут создавать исключения.
- Когда внутри такого блока try возникает исключение, поток управления переходит к первому подходящему обработчику исключений в стеке вызовов. В C# ключевое слово catch обозначает обработчик исключений.
- Если для созданного исключения не существует обработчиков, выполнение программы прекращается с сообщением об ошибке.
- Не перехватывайте исключение, если вы не намерены его обрабатывать с сохранением известного состояния приложения. Если вы перехватываете System.Exception, создайте его заново в конце блока catch, используя ключевое слово throw.
- Если блок catch определяет переменную исключения, ее можно использовать для получения дополнительных сведений о типе созданного исключения.
- Программа может явным образом создавать исключения с помощью ключевого слова throw.
- Объекты исключения содержат подробные сведения об ошибке, например состояние стека вызовов и текстовое описание ошибки.
- Код в блоке finally выполняется, даже если создано исключение. Используйте блок finally, чтобы высвободить ресурсы, например закрыть потоки и файлы, которые были открыты внутри блока try.
- Управляемые исключения реализованы в платформе .NET на основе структурированного механизма обработки исключений Win32.

Задание:

Составить программу деления вещественных чисел. программа должна выполнять обработку исключений с использованием конструкции try … catch, и выдавать следующие сообщения о характере ошибки:

1. не введено число (с помощью оператора условия);
2. введено слишком длинное число (с помощью оператора условия);
3. деление на ноль;
4. ошибка преобразования.

Лабораторная работа №9. Сериализация

Теоретическая часть:

Сериализация представляет собой процесс преобразования состояния объекта в форму, пригодную для сохранения или передачи. Дополнением к сериализации служит десериализация, при которой осуществляется преобразование потока в объект. Вместе эти процессы обеспечивают хранение и передачу данных.

В .NET доступны следующие технологии сериализации:

- При двоичной сериализации сохраняется правильность типов, что полезно для сохранения состояния объекта между разными вызовами приложения. Например, можно обеспечить совместный доступ к объекту для разных приложений, сериализовав его в буфер обмена. Объект можно сериализовать в поток, на диск, в память, передать по сети и т. д. При удаленном управлении сериализация используется для передачи объектов "по значению" с одного компьютера или домена приложения на другой.
- При сериализации XML и SOAP сериализуются только открытые свойства и поля, а правильность типов не сохраняется. Этот метод полезен для предоставления или использования данных без ограничений работающего с ними приложения. Будучи открытым стандартом, XML привлекателен для совместного использования данных в Интернете. Аналогичным образом и SOAP представляет собой открытый стандарт, использование которого эффективно и удобно.
- При сериализации JSON сериализуются только открытые свойства, а правильность типов не сохраняется. Будучи открытым стандартом, JSON привлекателен для совместного использования данных в Интернете.

Задание:

Требуется разработать программу, ведущую учёт заказов в магазине. Классы:

- (Покупатель) с тремя атрибутами: имя (string), адрес (string), скидка (double)
- (Товар). Поля, соответствующие названию (string) и цене (decimal)
- (База данных товаров), хранящий ассоциативный массив («словарь») с информацией о товарах
- OrderLine с полями количество (int) и продукт (Product)
- Order с полями номер заказа (int), клиент (Customer), скидка (decimal), общая стоимость (decimal) и строки заказа (List<OrderLIne>).

Реализовать следующую логику основной программы:

1. Создаётся и заполняется база данных товаров (ассоциативный массив).
2. В консоли вводятся данные по конкретному покупателю, создаётся соответствующий объект.
3. Создаётся заказ для введённого ранее покупателя. Устанавливается скидка на заказ в соответствии со скидкой покупателя.
4. В цикле формируются необходимое количество строк заказа: вводятся коды товаров и количества их единиц.
5. Полная информация о заказе сохраняется в файле с заданным именем.

Создать методы, которые осуществляют сериализацию/десериализацию объекта типа База данных товаров. Формат выбрать самостоятельно.

Лабораторная работа №10. Интерфейсы, сортировка с компаратором

Задание:

Дан текстовый файл, содержащий данные о продуктах на складе и их описания, например:

```
3кг Апельсины

10л Квас

100л Вода

3780г Шоколад

10т Бананы

13кг Мангал
```

1. Определите класс с тремя закрытыми полями:
    - Количество в кг (вещественное число);
    - Исходное представление количества (строка);
    - Название (строка).

2. Реализуйте конструктор, принимающий на вход два *строковых* значения: количество и название товара. Конструктор должен генерировать исключение, если количество является некорректным (меньше нуля). Добавьте свойства для преобразования количества в кг. В основной программе загрузите все температурные данные из исходного файла в список `List< >`.

3. Попытайтесь вызвать метод *Sort* для загруженного ранее списка температур. Возникающее при этом исключение свидетельствует о невозможности выполнять сравнение объектов произвольного класса. Чтобы это стало возможным, необходимо, например, *реализовать в классе интерфейс* `IComparable<T>`. Для этого:
    - измените заголовок класса на следующий

```
class ‘Название’: IComparable<’Название’>
```

    - Необходимости реализовать метод сравнения. Метод сравнения должен возвращать отрицательное число, если объект, для которого вызывается метод, *меньше* объекта, переданного в качестве параметра, `0` — если оба объекта *равны*, и положительное число — если исходный объект *больше* — реализуйте этот метод;
4. Убедитесь, что метод Sort работает и сортирует список.

Лабораторная работа №11. Интерфейсы, сортировка с компаратором

Задание:

1. Создать класс Point - точка на плоскости с вещественными координатами x, y. Создать конструктор, ToString() и свойства для доступа к координатам точки.
2. Создайте метод, который генерирует набор (LIst<Point>) случайно расположенных точек в квадрате [0,1]x[0,1].
3. Используя интерфейс IComparer, выведите все точки, упорядочивая их следующими способами:
    - по удалению от начала координат (сначала выводится ближайшая к началу координат, порядок равноудалённых точек не важен);
    - по удалению от оси абсцисс (сначала выводится ближайшая к оси абсцисс, порядок равноудалённых точек не важен);
    - по удалению от оси ординат (сначала выводится ближайшая к оси ординат, порядок равноудалённых точек не важен);
    - по удалению от диагонали первой и третьей четвертей (прямая y=x, порядок равноудалённых точек не важен).

Лабораторная работа №12. Творческий проект

Задание: Разработать программу для игры в крестики-нолики.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация C# - URL: https://docs.microsoft.com/ru-ru/dotnet/csharp/