

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра информационной безопасности**

**ОТЧЕТ**  
**по лабораторной работе №7**  
**по дисциплине «Объектно-ориентированное программирование»**  
**Тема: Использование переменных ссылочного типа**

Студенты гр. 3363

Преподаватель

Гончаренко О. Д.  
Овсейчик Н. И.,  
Минко Д. А.

Новакова Н. Е.

Санкт-Петербург  
2024

## **Цель работы**

Изучить использование переменных ссылочного типа в объектно-ориентированном программировании на языке C#, научиться добавлять методы с параметрами в класс, работать с методами, использующими ссылочные параметры, преобразовывать символы в файлах, тестировать реализацию интерфейсов, а также применять оператор `as` для работы с объектами интерфейсов.

## ХОД РАБОТЫ

Упражнение 1 – Добавление методов с параметрами в класс

1. Открытие Visual Studio 2022

Запустим Visual Studio 2022 и выберем "New" в меню "File", затем "Project".

2. Создание консольного приложения

Создадим новый проект с именем "Bank".

3. Редактирование класса BankAccount

Откроем файл BankAccount.cs и добавим метод TransferFrom (рис. 1).

```
public void TransferFrom(BankAccount accForm, decimal amount)
{
    if (accForm.Withdraw(amount) == true)
    {
        Deposit(amount);
    }
}
```

Рисунок – Метод TransferFrom в классе BankAccount

4. Тестирование метода в классе Test

В методе Main создадим два объекта b1 и b2, инициализируем их баланс (рис. 2).

```
Ссылка 0
public static void Main()
{
    BankAccount b1 = new BankAccount();
    BankAccount b2 = new BankAccount();
    b1.Populate(100);
    b2.Populate(100);
}
```

Рисунок 2 – Создание и инициализация объектов b1 и b2

5. Вызов метода TransferFrom

Добавим вызов метода TransferFrom для перевода \$10 с одного счета на другой (рис. 3).

```
Console.WriteLine("b2 - Тип: {0}, Номер: {1}, Баланс: {2}", b2.Type(), b2.Number(), b2.Balance());  
b1.TransferFrom(b2, 10);  
Console.WriteLine("Счета после операций:");
```

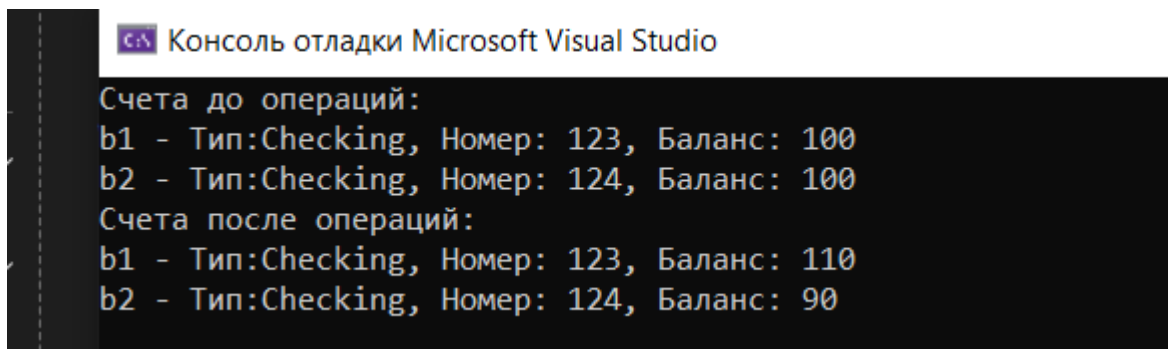
Рисунок 3 – Вызов метода TransferFrom для перевода денег

#### 6. Отображение результатов

Выведем в консоль информацию о балансах счетов до и после перевода (рис. 4, 5).

```
Console.WriteLine("Счета до операций:");  
Console.WriteLine("b1 - Тип:{0}, Номер: {1}, Баланс: {2}", b1.Type(), b1.Number(), b1.Balance());  
Console.WriteLine("b2 - Тип:{0}, Номер: {1}, Баланс: {2}", b2.Type(), b2.Number(), b2.Balance());  
b1.TransferFrom(b2, 10);  
Console.WriteLine("Счета после операций:");  
Console.WriteLine("b1 - Тип:{0}, Номер: {1}, Баланс: {2}", b1.Type(), b1.Number(), b1.Balance());  
Console.WriteLine("b2 - Тип:{0}, Номер: {1}, Баланс: {2}", b2.Type(), b2.Number(), b2.Balance());
```

Рисунок 4 – “Вывод информации о балансах счетов до и после перевода”



Консоль отладки Microsoft Visual Studio

Счета до операций:  
b1 - Тип:Checking, Номер: 123, Баланс: 100  
b2 - Тип:Checking, Номер: 124, Баланс: 100  
Счета после операций:  
b1 - Тип:Checking, Номер: 123, Баланс: 110  
b2 - Тип:Checking, Номер: 124, Баланс: 90

Рисунок 5 – Информации, выведенная в консоль, о балансах счетов до и после перевода

Реализована диаграмма класса для данного упражнения (Диаграмма 1).

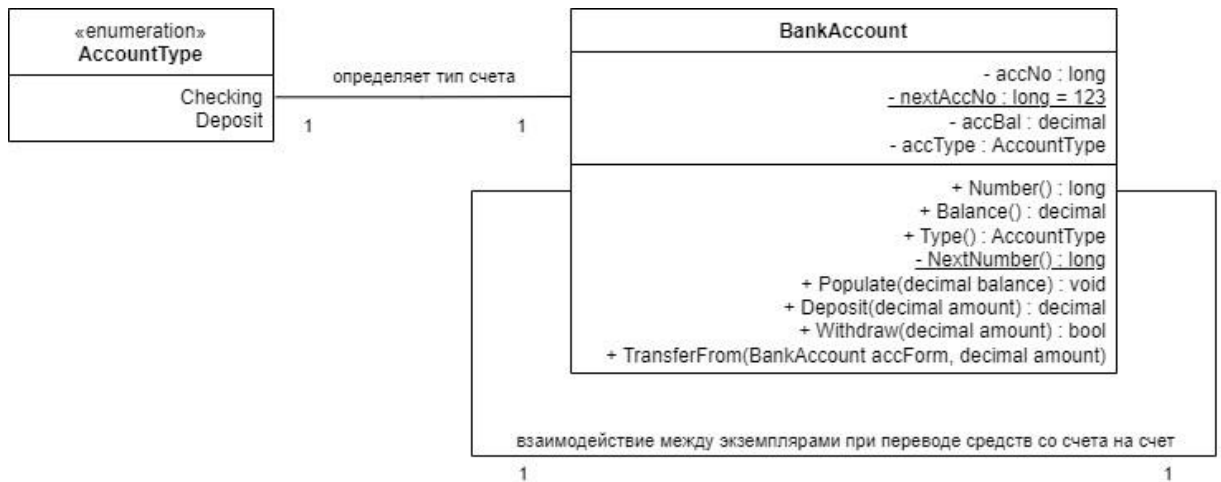


Диаграмма 1 – Диаграмма классов для упражнения 1

Упражнение 2 – Использование методов со ссылочными параметрами

1. Создание нового проекта:

Создадим проект с именем "Utils".

2. Добавление метода Reverse:

Добавим метод Reverse в класс Utils (рис.6).

```

public static void Reverse(ref string s)
{
    string sRev = "";
    for (int i = s.Length; i > 0; i--)
    {
        sRev += s[i - 1];
    }
    s = sRev;
}
    
```

Рисунок 6 – Метод Reverse в классе Utils

3. Тестирование метода в классе Test.

Создадим переменную **message**, считаем ее значение с консоли и передадим в метод **Reverse** (рис. 7, 8).

```

public class Test
{
    Ссылка: 0
    public static void Main()
    {
        string message;
        Console.WriteLine("До");
        message = Console.ReadLine();
        Console.WriteLine(message);
        Utils.Reverse(ref message);
        Console.WriteLine("После");
        Console.WriteLine(message);
    }
}

```

Рисунок 7 – Вывод тестирования метода Reverse

```

Консоль отладки Microsoft Visual
Utils
До
1:abcdef
1:abcdef
1:После
1:fedcba
1:
1:C:\Labs\Lab7\Utils\bin\Debug
1:Чтобы автоматически закрывать

```

Рисунок 8 – Тестирование метода Reverse

Реализована диаграмма класса для данного упражнения (Диаграмма 2).

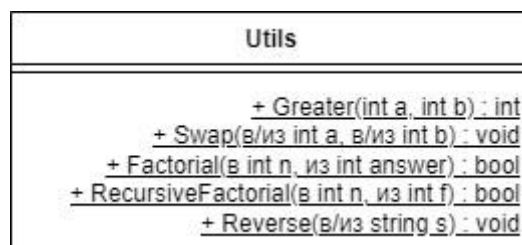


Диаграмма 2 – Диаграмма классов для упражнения 2

Упражнение 3 – Преобразование символов файла в верхний регистр

1. Создание нового проекта

Создадим проект с именем "CopyFileUpper".

## 2. Редактирование файла CopyFileUpper.cs

Добавим необходимые пространства имен и создадим блоки для чтения и записи файлов (рис. 9)

```
using System.IO;
```

Рисунок 9 – Добавление пространства имен System.IO

## 3. Добавление переменных sFrom и sTo, объявление переменных StreamReader и StreamWriter

В методе Main объявим строки, которые будут использоваться для хранения имен входного и выходного файлов. Для работы с входным и выходным потоками создадим переменные srFrom и swTo (рис.10):

```
string sFrom, sTo;  
StreamReader srFrom;  
StreamWriter swTo;
```

Рисунок 10 – Добавление переменных sFrom и sTo, объявление переменных StreamReader и StreamWriter

#### 4. Запрос имени входного/выходного файла у пользователя

Добавим запрос для ввода имени входного файла и считывание его в переменную `sFrom`, а для ввода имени выходного файла и считывание его в переменную `sTo` (рис. 11):

```
Console.WriteLine("Введите имя входного файла");  
sFrom = Console.ReadLine();  
Console.WriteLine("Введите имя выходного файла");  
sTo = Console.ReadLine();
```

Рисунок 11 – Запрос имени входного/выходного файла у пользователя

#### 5. Создание блока try-catch

Обернём основной код программы в блок `try-catch` для обработки исключений, связанных с файловыми операциями (рис.12):

```
try  
{  
    string sFrom, sTo;  
    StreamReader srFrom;  
    StreamWriter swTo;  
    Console.WriteLine("Введите имя входного файла");  
    sFrom = Console.ReadLine();  
    Console.WriteLine("Введите имя выходного файла");  
    sTo = Console.ReadLine();  
}  
catch (FileNotFoundException e)  
{  
    Console.WriteLine("Файл не существует");  
}  
catch (Exception e)  
{  
    Console.WriteLine("Произошла ошибка");  
}
```

Рисунок 12 – Создание блока try-catch



## 6. Открытие входного и выходного потоков

Внутри блока try создадим объекты StreamReader и StreamWriter для чтения из входного файла и записи в выходной файл (рис.13):

```
srFrom = new StreamReader(sFrom);  
swTo = new StreamWriter(sTo);
```

Рисунок 13 – Открытие входного и выходного потоков

## 7. Чтение данных и запись в верхнем регистре

Добавим цикл while, который работает до тех пор, пока метод Peek() из входного потока не возвратит значение -1, что указывает на конец файла. Внутри цикла используем методы ReadLine() для чтения строки и ToUpper() для преобразования строки в верхний регистр. После завершения работы цикла закроем оба потока StreamReader и StreamWriter, чтобы освободить используемые ресурсы (рис.14):

```
while (srFrom.Peek() != -1)  
{  
    string sBuffer = srFrom.ReadLine();  
    swTo.WriteLine(sBuffer.ToUpper());  
}  
srFrom.Close();  
swTo.Close();
```

Рисунок 14 – Чтение данных и запись в верхнем регистре и закрытие потоков

Проверим работу данной программы на входном файле text1.txt и выходном файле text2.txt: в text1.txt запишем текст, а в файле text2.txt он должен записаться в верхнем регистре (рис.15):

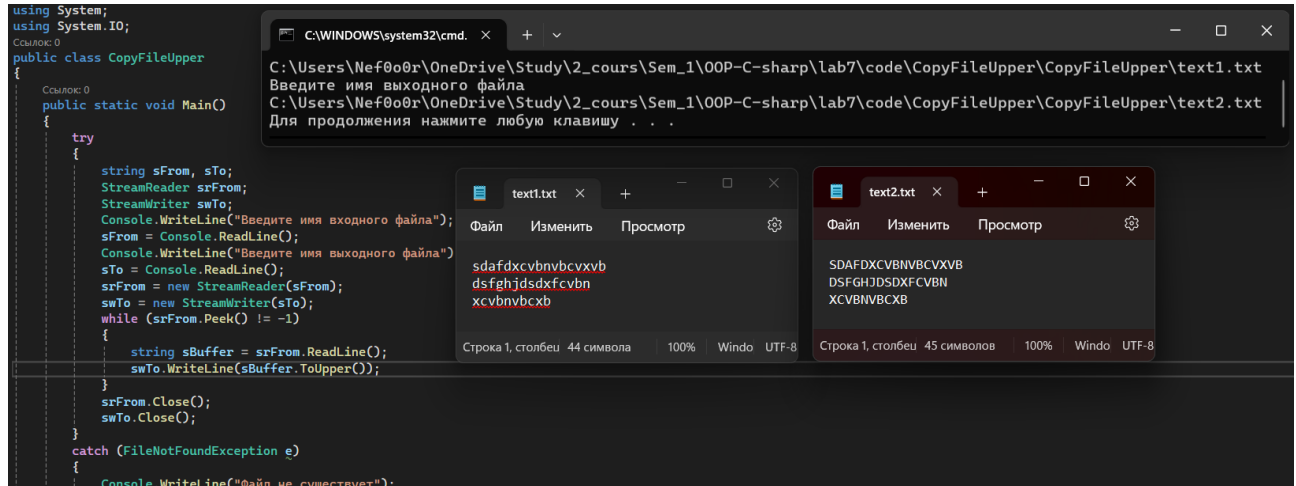


Рисунок 15 – Проверка работы программы CopyFileUpper

Теперь протестируем программу через командную строку: запустим программу через командную строку и в качестве входного файла напишем CopyFileUpper.cs, а в качестве выходного файла Test.cs (в нашем случае Program.cs) и получаем текст программы в верхнем регистре (рис.16)

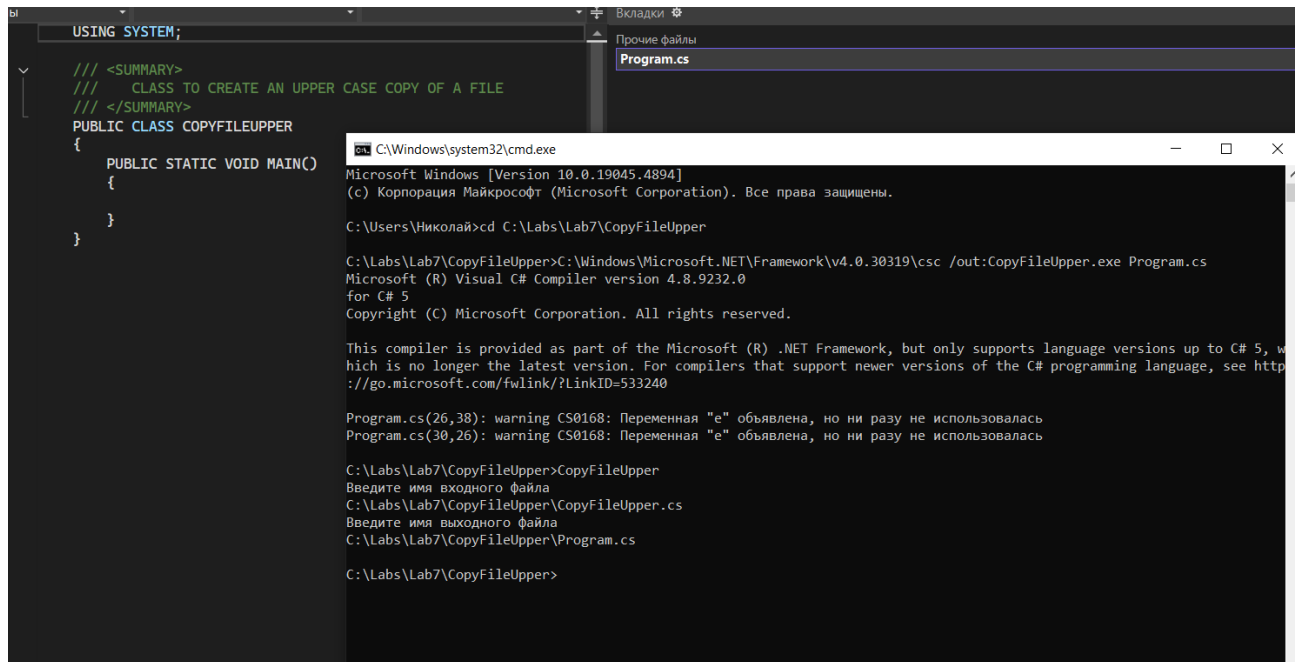


Рисунок 16 – Проверка работы программы CopyFileUpper через командную строку

Поскольку в данном упражнении все выполняется через класс с методом Main, диаграмма классов для упражнения не имеет смысла.

#### Упражнение 4 – Проверка реализации интерфейса

##### 1. Создание нового проекта

Создадим проект с именем "InterfaceTest".

##### 2. Добавление метода IsItFormattable

Добавим метод, который проверяет, поддерживает ли объект интерфейс IFormattable (рис. 17).

```

Ссылка: 3
public static bool IsItFormattable(object x)
{
    return x is IFormattable;
}

```

Рисунок 17 – Метод IsItFormattable в классе Utils

### 3. Тестирование метода в классе Test

Создадим три переменные и протестируем метод (рис. 18).

```

18 }
19
20 Ссылка: 3
21 public static bool IsItFormattable(object x)
22 {
23     return x is IFormattable;
24 }
25
26
27
28
29
30
31
32
33
34
35 Ссылка: 0
36 public class Test
37 {
38
39     Ссылка: 0
40     public static void Main()
41     {
42         int i = 0;
43         ulong ul = 0;
44         string s = "Test";
45         Console.WriteLine(Utils.IsItFormattable(i));
46         Console.WriteLine(Utils.IsItFormattable(ul));
47         Console.WriteLine(Utils.IsItFormattable(s));
48     }
49 }

```

Консоль отладки Microsoft Visual Studio

```

True
True
False

```

C:\Labs\Lab7\InterfaceTest\bin\Deb  
 Чтобы автоматически закрывать консоль при окончании отладки, нажмите любую клавишу, чтобы закрыть консоль.

Рисунок 18 – Тестирование метода IsItFormattable

Реализована диаграмма класса для данного упражнения (Диаграмма 3).

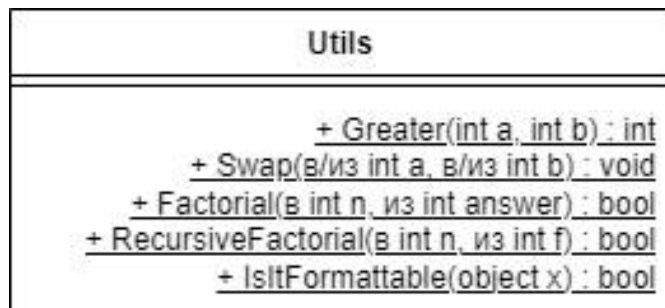


Диаграмма 3 – Диаграмма классов для упражнения 4

### Упражнение 5 – Работа с интерфейсами

### 1. Создание нового проекта

Создадим проект с именем "TestDisplay".

### 2. Добавление метода Display

Добавим метод, который выводит информацию об объекте (рис. 19).

```
public static void Display(object item)
{
    IPrintable ip;
    ip = item as IPrintable;
    if (ip != null)
    {
        ip.Print();
    }
    else
    {
        Console.WriteLine(item.ToString());
    }
}
```

Рисунок 19 – Метод Display в классе Utils

### 3. Тестирование метода в классе Test

Создадим переменные и передадим их в метод Display (рис. 20).

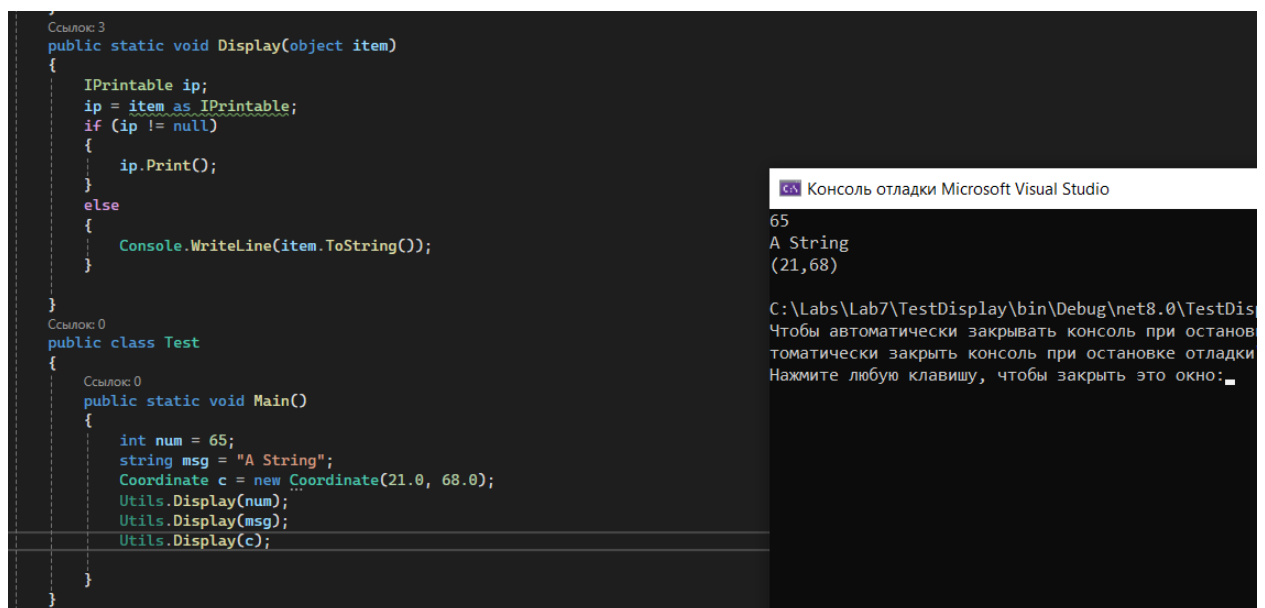


Рисунок 20 – Тестирование метода Display

Реализована диаграмма класса для данного упражнения (Диаграмма 4).

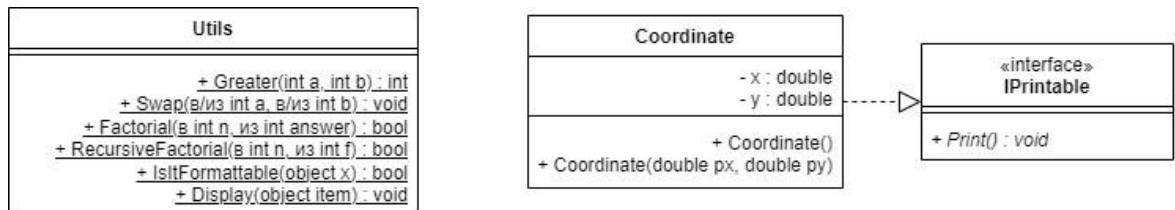


Диаграмма 4 – Диаграмма классов для упражнения 5

## Спецификация классов

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Выходные параметры	Назначение
Display	void	public	object item	-	Отображает информацию об объекте, поддерживающем интерфейс IPrintable, или его строковое представление.
Factorial	bool	public	int n, int answer	out int answer	Вычисляет факториал числа n, результат возвращается через параметр answer.
Greater	int	public	int a, int b	-	Возвращает большее из двух целых чисел.
IsItFormattable	bool	public	object x	-	Проверяет, реализует ли объект интерфейс IFormattable.
RecursiveFactori	bool	public	int n, int f	out int f	Рекурсивно вычисляет факториал числа n, результат возвращается через параметр f.

Swap	void	public	int a, int b	-	Обменивает значения двух целых чисел, переданных по ссылке.
------	------	--------	--------------	---	---

Таблица 1 – Описание методов класса Utils

Имя	Тип	Модификатор доступа	Назначение
x	double	private	Координата X объекта Coordinate.
y	double	private	Координата Y объекта Coordinate.

Таблица 2 – Описание полей класса Coordinate

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Выходные параметры	Назначение
Coordinate	-	public	double px, double py	-	Конструктор, инициализирующий координаты объекта значениями px и py.
Print	void	public	-	-	Выводит координаты объекта в консоль.

Таблица 3 – Описание методов класса Coordinate

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Выходные параметры	Назначение
-------	------------------	---------------------	-------------------	--------------------	------------



IPrintable	void	-	-	-	Определяет метод для вывода данных объекта.
------------	------	---	---	---	---

Таблица 4 – Описание методов класса Coordinate

## **Вывод**

В ходе выполнения лабораторной работы были изучены и применены на практике переменные ссылочного типа, методы с параметрами, а также работа с интерфейсами. Эти знания помогут в дальнейшем развитии навыков программирования и реализации более сложных задач.

## Исходный Код

### Упражнение 1:

```
using System;
using System.Collections.Specialized;
using System.Runtime.Intrinsics.X86;
enum AccountType
{
    Checking,
    Deposit
}
class BankAccount
{
    private long accNo;
    private decimal accBal;
    private AccountType accType;

    private static long nextNumber = 123;

    public void Populate(decimal balance)
    {
        accNo = NextNumber();
        accBal = balance;
        accType = AccountType.Checking;
    }

    public bool Withdraw(decimal amount)
    {
        bool sufficientFunds = accBal >= amount;
        if (sufficientFunds)
        {
            accBal -= amount;
        }
        return sufficientFunds;
    }

    public decimal Deposit(decimal amount)
    {
        accBal += amount;
        return accBal;
    }

    public void TransferFrom(BankAccount accForm, decimal amount)
    {
        if (accForm.Withdraw(amount) == true)
        {
            Deposit(amount);
        }
    }

    //b1.Transfer(b2, 100)
    public long Number()
    {
        return accNo;
    }

    public decimal Balance()
    {
        return accBal;
    }

    public string Type()
    {
        return accType.ToString();
    }

    private static long NextNumber()
```

```

    {
        return nextNumber++;
    }
    public class Test
    {
        public static void Main()
        {
            BankAccount b1 = new BankAccount();
            BankAccount b2 = new BankAccount();
            b1.Populate(100);
            b2.Populate(100);
            Console.WriteLine("Счета до операций:");
            Console.WriteLine("b1 - Тип:{0}, Номер: {1}, Баланс: {2}",
b1.Type(), b1.Number(), b1.Balance());
            Console.WriteLine("b2 - Тип:{0}, Номер: {1}, Баланс: {2}",
b2.Type(), b2.Number(), b2.Balance());
            b1.TransferFrom(b2, 10);
            Console.WriteLine("Счета после операций:");
            Console.WriteLine("b1 - Тип:{0}, Номер: {1}, Баланс: {2}",
b1.Type(), b1.Number(), b1.Balance());
            Console.WriteLine("b2 - Тип:{0}, Номер: {1}, Баланс: {2}",
b2.Type(), b2.Number(), b2.Balance());
        }
    }
}

```

## Упражнение 2:

```

namespace Utils
{
    using System;

    class Utils
    {
        //
        // Return the larger of two integer values
        //

        public static int Greater(int a, int b)
        {
            if (a > b)
                return a;
            else
                return b;

            // Alternative version - more terse
            // return (a>b) > (a) : (b);
        }

        //
        // Swap two integers, passed by reference
        //

        public static void Swap(ref int a, ref int b)
        {
            int temp;
            temp = a;
            a = b;
            b = temp;
        }
    }
}

```

```

//
// Calculate factorial
// and return the result as an out parameter
//

public static bool Factorial(int n, out int answer)
{
    int k;          // loop counter
    int f;          // working value
    bool ok = true; // true if ok, false if not

    // Check the input value

    if (n < 0)
        ok = false;

    // Calculate the factorial value as the
    // product of all the numbers from 2 to n

    try
    {
        checked
        {
            f = 1;
            for (k = 2; k <= n; ++k)
            {
                f = f * k;
            }

            // Here is a terse alternative
            // for (f=1,k=2;k<=n;++k)
            //     f*=k;
        }
    }
    catch (Exception)
    {
        // If something goes wrong in the calculation,
        // catch it here. All exceptions
        // are handled the same way: set the result to
        // to zero and return false.

        f = 0;
        ok = false;
    }

    // assign result value
    answer = f;

    // return to caller
    return ok;
}

//
// Another way to solve the factorial problem, this time
// as a recursive function
//

public static bool RecursiveFactorial(int n, out int f)
{
    bool ok = true;

    // Trap negative inputs
    if (n < 0)
    {

```

```

        f = 0;
        ok = false;
    }

    if (n <= 1)
        f = 1;
    else
    {
        try
        {
            int pf;
            checked
            {
                ok = RecursiveFactorial(n - 1, out pf);
                f = n * pf;
            }
        }
        catch (Exception)
        {
            // Something went wrong. Set error
            // flag and return zero.
            f = 0;
            ok = false;
        }
    }

    return ok;
}

public static void Reverse(ref string s)
{
    string sRev = "";
    for (int i = s.Length; i > 0; i--)
    {
        sRev += s[i - 1];
    }
    s = sRev;
}

public class Test
{
    public static void Main()
    {
        string message;
        Console.WriteLine("До");
        message = Console.ReadLine();
        Console.WriteLine(message);
        Utils.Reverse(ref message);
        Console.WriteLine("После");
        Console.WriteLine(message);
    }
}
}

```

### Упражнение 3:

```

using System;
using System.IO;
public class CopyFileUpper
{
    public static void Main()
    {
        try
        {
            string sFrom, sTo;

```

```

        StreamReader srFrom;
        StreamWriter swTo;
        Console.WriteLine("Введите имя входного файла");
        sFrom = Console.ReadLine();
        Console.WriteLine("Введите имя выходного файла");
        sTo = Console.ReadLine();
        srFrom = new StreamReader(sFrom);
        swTo = new StreamWriter(sTo);
        while (srFrom.Peek() != -1)
        {
            string sBuffer = srFrom.ReadLine();
            swTo.WriteLine(sBuffer.ToUpper());
        }
        srFrom.Close();
        swTo.Close();
    }
    catch (FileNotFoundException e)
    {
        Console.WriteLine("Файл не существует");
    }
    catch (Exception e)
    {
        Console.WriteLine("Произошла ошибка");
    }
}
}

```

#### Упражнение 4:

```

namespace Utils
{
    using System;

    class Utils
    {
        //
        // Return the larger of two integer values
        //

        public static int Greater(int a, int b)
        {
            if (a > b)
                return a;
            else
                return b;

            // Alternative version - more terse
            // return (a>b) ? a : b;
        }

        //
        // Swap two integers, passed by reference
        //

        public static void Swap(ref int a, ref int b)
        {
            int temp;
            temp = a;
            a = b;
            b = temp;
        }
    }
}

```

```

//
// Calculate factorial
// and return the result as an out parameter
//

public static bool Factorial(int n, out int answer)
{
    int k;          // loop counter
    int f;          // working value
    bool ok = true; // true if ok, false if not

    // Check the input value

    if (n < 0)
        ok = false;

    // Calculate the factorial value as the
    // product of all the numbers from 2 to n

    try
    {
        checked
        {
            f = 1;
            for (k = 2; k <= n; ++k)
            {
                f = f * k;
            }

            // Here is a terse alternative
            // for (f=1,k=2;k<=n;++k)
            //     f*=k;
        }
    }
    catch (Exception)
    {
        // If something goes wrong in the calculation,
        // catch it here. All exceptions
        // are handled the same way: set the result to
        // to zero and return false.

        f = 0;
        ok = false;
    }

    // assign result value
    answer = f;

    // return to caller
    return ok;
}

//
// Another way to solve the factorial problem, this time
// as a recursive function
//

public static bool RecursiveFactorial(int n, out int f)
{
    bool ok = true;

    // Trap negative inputs
    if (n < 0)
    {

```



```

        f = 0;
        ok = false;
    }

    if (n <= 1)
        f = 1;
    else
    {
        try
        {
            int pf;
            checked
            {
                ok = RecursiveFactorial(n - 1, out pf);
                f = n * pf;
            }
        }
        catch (Exception)
        {
            // Something went wrong. Set error
            // flag and return zero.
            f = 0;
            ok = false;
        }
    }

    return ok;
}

public static bool IsItFormattable(object x)
{
    return x is IFormattable;
}
}

public class Test
{
    public static void Main()
    {
        int i = 0;
        ulong ul = 0;
        string s = "Test";
        Console.WriteLine(Utils.IsItFormattable(i));
        Console.WriteLine(Utils.IsItFormattable(ul));
        Console.WriteLine(Utils.IsItFormattable(s));
    }
}
}

```

### Упражнение 5:

```

namespace Utils
{
    using System;

    class Utils
    {
        public static bool IsItFormattable(object x)
        {
            // Use is to test if the object has the
            // IFormattable interface

            if (x is IFormattable)
                return true;
            else
                return false;
        }
    }
}

```

```

}

//
// Return the larger of two integer values
//
public static int Greater(int a, int b)
{
    if (a > b)
        return a;
    else
        return b;

    // Alternative version - more terse
    // return (a>b) ? (a) : (b);
}

//
// Swap two integers, passed by reference
//

public static void Swap(ref int a, ref int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}

//
// Calculate factorial
// and return the result as an out parameter
//

public static bool Factorial(int n, out int answer)
{
    int k;          // loop counter
    int f;          // working value
    bool ok = true; // true if ok, false if not

    // Check the input value

    if (n < 0)
        ok = false;

    // Calculate the factorial value as the
    // product of all the numbers from 2 to n

    try
    {
        checked
        {
            f = 1;
            for (k = 2; k <= n; ++k)
            {
                f = f * k;
            }

            // Here is a terse alternative
            // for (f=1,k=2;k<=n;++k)
            //     f*=k;
        }
    }
    catch (Exception)
    {

```

```

        // If something goes wrong in the calculation,
        // catch it here. All exceptions
        // are handled the same way: set the result to
        // to zero and return false.

        f = 0;
        ok = false;
    }

    // assign result value
    answer = f;

    // return to caller
    return ok;
}

//
// Another way to solve the factorial problem, this time
// as a recursive function
//

public static bool RecursiveFactorial(int n, out int f)
{
    bool ok = true;

    // Trap negative inputs
    if (n < 0)
    {
        f = 0;
        ok = false;
    }

    if (n <= 1)
        f = 1;
    else
    {
        try
        {
            int pf;
            checked
            {
                ok = RecursiveFactorial(n - 1, out pf);
                f = n * pf;
            }
        }
        catch (Exception)
        {
            // Something went wrong. Set error
            // flag and return zero.
            f = 0;
            ok = false;
        }
    }

    return ok;
}

interface IPrintable
{
    void Print();
}

class Coordinate : IPrintable
{
    private double x;
    private double y;
}

```

```

    public Coordinate()
    {
        x = 0.0;
        y = 0.0;
    }

    public Coordinate(double px, double py)
    {
        x = px;
        y = py;
    }

    public void Print()
    {
        Console.WriteLine("{0},{1}", x, y);
    }
}

public static void Display(object item)
{
    IPrintable ip;
    ip = item as IPrintable;
    if (ip != null)
    {
        ip.Print();
    }
    else
    {
        Console.WriteLine(item.ToString());
    }
}

public class Test
{
    public static void Main()
    {
        int num = 65;
        string msg = "A String";
        Coordinate c = new Coordinate(21.0, 68.0);
        Utils.Display(num);
        Utils.Display(msg);
        Utils.Display(c);
    }
}
}
}

```