**Лабораторная работа №8. Конкретизация.**

**Важно! Говорить что мы делаем – конкретизируем класс или метод.**

A

IA

Public

Private

Protected

Main

Swap <T>

Public

int

U<T> where T:class

Main

Private

Protected

A

MainSwap IdAccount IdAccount()

**T T**

**Текст программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab8

{

public interface IA

{

void F();

}

class A : IA

{

public void F() { Console.WriteLine("Привет"); }

}

class U<T> where T:class// с помощью where T: class мы можем ограничить тип Т. в данном случае здесь по классу, т.е. в Т можем конкретизировать только классы.

// здесь задача не используя наследования, с помощью этого класса обратиться и вызвать метод другого класса

{ // при использовании категории использование, он будет выдавать ошибку.

//поэтому я определил интерфейс и сделал преобразования типа к интерфейсу для универсального параметра

public void F1() { Console.WriteLine("Как дела?"); }

public U(T t) { this.t = t; }

public T t { get; set; }

//IA ia;// = new A();

public void F()

{

Console.WriteLine(t is IA);

Console.WriteLine(t);// вывод что это вообще за параметр

if (t is IA)

{

IA ia = (IA)t;// делаем преобразование типа к интерфейсу

ia.F();// вызов функции в классе А F()

}

}

}

class IdAccount<T>

{

public IdAccount(T name, int Sum) { this.\_Sum = Sum; this.namet = name; }

public T namet;

public int \_Sum;

}

class Swap<T>// здесь Т универсальный параметр, он позволяет указать, какой тип здесь будет использоваться

{

public void FSwap(ref T x, ref T y)

{

t = x;

x = y;

y = t;

}

public T t;

}

class Program

{

static void Main(string[] args)

{

Swap<int> swap = new Swap<int>();// здесь мы используем объект типа int

int x = 5;

int y = 6;

Console.WriteLine("x = {0} y = {1}", x, y);

swap.FSwap(ref x, ref y);// передача параметров по ссылке. при передачи по значению, в метод передается только копия, а здесь передается ссылка на саму переменную в память

Console.WriteLine("x = {0} y = {1}", x, y);

IdAccount<string> idAccount1 = new IdAccount<string>("Ваня", 5000);

IdAccount<int> idAccount2 = new IdAccount<int>(15954, 70000);

Console.WriteLine("{0} {1}", idAccount1.namet, idAccount1.\_Sum);

Console.WriteLine("{0} {1}", idAccount2.namet, idAccount2.\_Sum);

A a = new A();

//Console.WriteLine(a is IA);

U<A> u = new U<A>(a);// передача по ссылке объекта в конструктор. здесь мы передали класс А в параметр типа.

//U<A> u = new U<A>();

//u.t = a;// связываем объекты двух классов u и a.

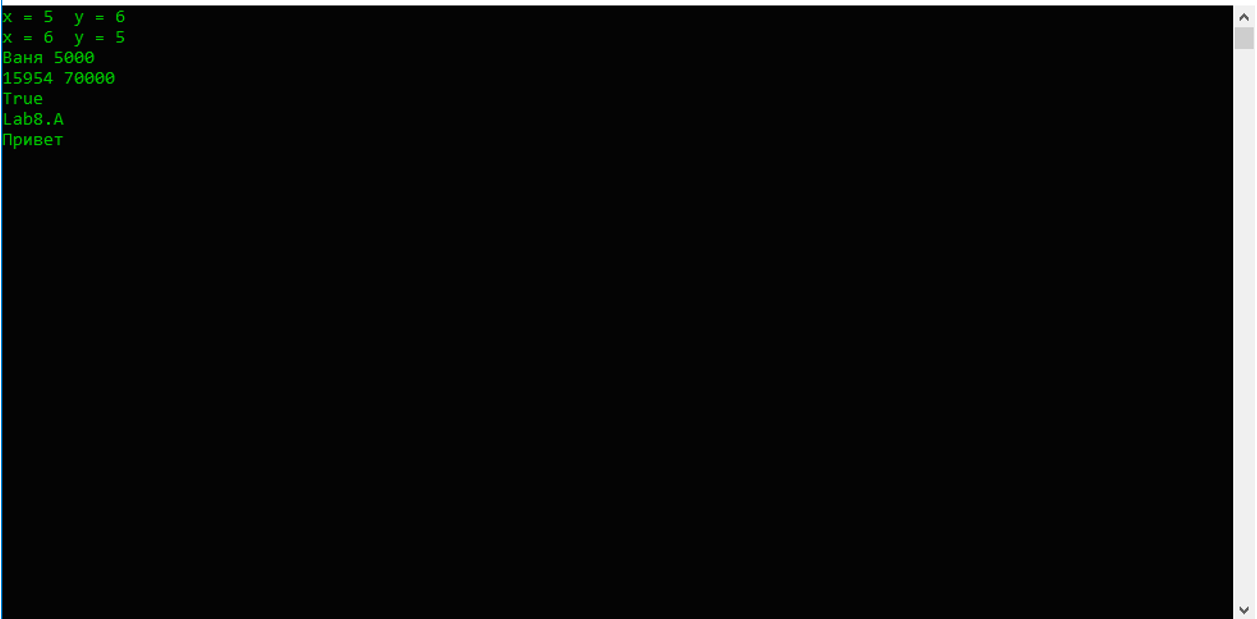
u.F();

Console.ReadKey();

}

}

}



Вывод: Мы конкретизируем классы и функции для работы с объектами, типы которых заранее не определены. С каким(и) именно типам(и) придется работать определяется в момент вызова – имя типа настраивает функцию\класс для работы. Это нужно для повторного использования кода (повышает гранулярность программного кода).