МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

 «БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра ИИТ

ЛАБОРАТОРНАЯ РАБОТА №5

Выполнил:

Студент ФЭИС

3-го курса, группы ПО-5

Белко В.А.

Проверил:

Крощенко А. А.

Брест 2021

**Цель работы**: приобрести практические навыки в области объектно-ориентированного проектирования.

**Задание 1**

Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов:

2) interface Abiturient ← abstract class Student ← class Student Of Faculty.

**Код программы:**

using System;

namespace task1

{

class Program

{

static void Main(string[] args)

{

StudentOfFaculty student = new StudentOfFaculty(3, 19);

Console.WriteLine("Years: " + student.GetYears());

Console.WriteLine("Experience: " + student.GetExperience());

}

}

public interface IAbiturient

{

public int GetYears();

}

public abstract class Student : IAbiturient

{

public Student(int years)

{

\_years = years;

}

int \_years;

public int GetYears()

{

return \_years;

}

}

public class StudentOfFaculty : Student

{

int \_experience;

public StudentOfFaculty(int years, int experience) : base(years)

{

\_experience = experience;

}

public int GetExperience()

{

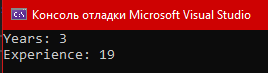
return \_experience;

}

}

}

**Вывод программы:**



**Задание 2**

В следующих заданиях требуется создать суперкласс (абстрактный класс, интерфейс) и определить общие методы для данного класса. Создать подклассы, в которых добавить специфические свойства и методы. Часть методов переопределить. Создать массив объектов суперкласса и заполнить объектами подклассов. Объекты подклассов идентифицировать конструктором по имени или идентификационному номеру. Использовать объекты подклассов для моделирования реальных ситуаций и объектов.

2) Создать суперкласс Учащийся и подклассы Школьник и Студент. Создать массив объектов суперкласса и заполнить этот массив объектами. Показать отдельно студентов и школьников.

**Код программы:**

using System;

using System.Collections.Generic;

using System.Security.Cryptography;

namespace task2

{

class Program

{

static void Task(Student obj1, Student obj2)

{

if (obj1.GetKnoweledge() > obj2.GetKnoweledge()) Console.WriteLine("Student1 has taken the automatic offset " +

"on the subject from a friend Student2");

else if (obj1.GetKnoweledge() < obj2.GetKnoweledge()) Console.WriteLine("Student2 has taken the automatic offset " +

"on the subject from a friend Student1");

else Console.WriteLine("They both have gone to the army :(");

}

static void Money(Schoolboy obj1, Schoolboy obj2)

{

if (obj1.GetPower() > obj2.GetPower()) Console.WriteLine("Schoolboy1 has taken money of Schoolboy2");

else if (obj1.GetPower() < obj2.GetPower()) Console.WriteLine("Schoolboy2 has taken the money of Schoolboy1");

else Console.WriteLine("They both have gone to the prison");

}

static void Main(string[] args)

{

List<Learner> learners = new List<Learner>();

Student student1 = new Student(100, "Stas", "U151");

Student student2 = new Student(500, "Pasha", "H522");

Schoolboy schoolboy1 = new Schoolboy(1000, "Dima", "GrodnoSchool123");

Schoolboy schoolboy2 = new Schoolboy(500, "Vasia", "MinskSchool543");

Money(schoolboy1, schoolboy2);

Task(student1, student2);

Console.WriteLine(schoolboy1.GetDocument());

Console.WriteLine(student2.GetDocument());

learners.Add(student1);

learners.Add(student2);

learners.Add(schoolboy1);

learners.Add(schoolboy2);

foreach (var item in learners)

{

if (item.GetType() == typeof(Student))

{

Console.WriteLine(item.Name + " - Student");

}

if (item.GetType() == typeof(Schoolboy))

{

Console.WriteLine(item.Name + " - Schoolboy");

}

}

}

}

public abstract class Learner

{

public string Name { get; set; }

public int Years { get; set; }

public string Passport { get; set; }

public virtual string GetDocument()

{

return Passport;

}

}

public class Student : Learner

{

public Student(int \_knoweledge, string \_name, string recordbook)

{

knoweledge = \_knoweledge;

Name = \_name;

\_recordbook = recordbook;

}

public int knoweledge;

public string university;

string \_recordbook;

public override string GetDocument()

{

return \_recordbook;

}

public string GetUniversity()

{

return university;

}

public int GetKnoweledge()

{

return knoweledge;

}

}

public class Schoolboy : Learner

{

public Schoolboy(int \_power, string \_name, string journal)

{

power = \_power;

Name = \_name;

\_journal = journal;

}

public int power;

public string school;

string \_journal;

public override string GetDocument()

{

return \_journal;

}

public string GetSchool()

{

return school;

}

public int GetPower()

{

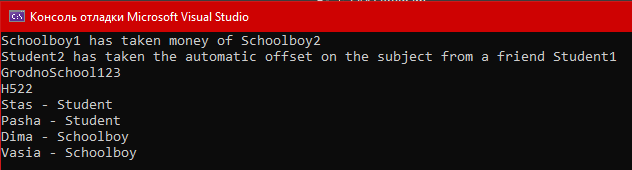
return power;

}

}

}

**Вывод программы:**

****

**Задание 3:**

В задании 3 ЛР №4, где возможно, заменить объявления суперклассов объявлениями абстрактных классов или интерфейсов.

**Код программы:**

using System;

using System.Collections.Generic;

namespace task3

{

class Program

{

static void Main(string[] args)

{

Payments.Client client1 = new Payments.Client();

Payments.Client client2 = new Payments.Client();

Good good1 = new Good

{

Sum = 200,

};

Payments.Administrator admin = new Payments.Administrator();

Console.WriteLine("Count client1: " + client1.GetCount());

client1.Pay(good1);

Console.WriteLine("Count client1: " + client1.GetCount());

Console.WriteLine("Count client2: " + client2.GetCount());

client1.PayTo(client2.GetAccount(), 10000);

Console.WriteLine("Count client1: " + client1.GetCount());

Console.WriteLine("Count client2: " + client2.GetCount());

Console.WriteLine("Close Account client2");

client2.CloseAccount();

Console.WriteLine("Close Card client2");

client2.CloseCard();

Console.WriteLine("Admin close Card client1");

admin.BlockClientCard(client1);

admin.ShowInfo();

client2.ShowInfo();

}

}

public class Good

{

public int Sum { get; set; }

}

public class Payments

{

static public List<Client> Clients = new List<Client>();

public abstract class User

{

public virtual void ShowInfo() { }

}

public class Client : User

{

Account account;

CCard card;

int \_code;

public override void ShowInfo()

{

Console.WriteLine(\_code + " root");

}

public Client()

{

Random random = new Random();

\_code = random.Next(100, 999);

account = new Account(5000);

card = new CCard(account);

Clients.Add(this);

}

public void Pay(Good good) // using Card

{

card.Pay(good);

}

public void PayTo(Account other, int sum) // using Card

{

card.PayTo(other, sum);

}

public void CloseCard() // using Card

{

card.Close();

}

public void CloseAccount() // using Account

{

account.CloseAccount();

}

public int GetCount()

{

return card.Count();

}

public Account GetAccount()

{

return account;

}

}

public class Administrator : User

{

public override void ShowInfo()

{

Console.WriteLine("Admin root");

}

public void BlockClientCard(Client obj)

{

if (obj.GetCount() < 0)

{

obj.CloseCard();

}

else Console.WriteLine("Card is not blocked. The count is correct.");

}

}

public class CCard

{

public Account Account;

public bool Closed;

public CCard(Account \_account) // any Card has Account

{

Closed = false;

Account = \_account;

}

public void Close()

{

Closed = true;

Console.WriteLine("The card was closed.");

}

public int Count() // return Count from Account

{

if (Closed)

{

Console.WriteLine("Card is locked");

return 0;

}

else return Account.Count;

}

public void Pay(Good obj) // taking Good and change our Count

{

if (Closed)

{

Console.WriteLine("Card is locked");

return;

}

else

{

Account.TakeSum(obj.Sum);

Console.WriteLine("The good was paid.");

}

}

public void PayTo(Account other, int sum)

{

if (Closed)

{

Console.WriteLine("Card is locked");

return;

}

else

{

Account.TakeSum(sum);

other.AddSum(sum);

Console.WriteLine("The sum was sent to the other client.");

}

}

}

public class Account

{

public int Number { get; private set; } // the private number of the Account

public int Count { get; set; } // the Count

public bool Validation { get; private set; } // private Validation

public void CloseAccount()

{

Validation = false;

Console.WriteLine("The account was closed.");

}

public Account(int \_count)

{

Random random = new Random();

Number = random.Next(1000, 9999); // the number is random value

Count = \_count; // open on our private Sum

Validation = true; // default - Account is valid

}

public void AddSum(int sum) // add some sum to Count

{

if (!Validation)

{

Console.WriteLine("Account is not valid");

return;

}

else Count += sum;

}

public void TakeSum(int sum) // take some sum from Count

{

if (!Validation)

{

Console.WriteLine("Account is not valid");

return;

}

else

{

Count -= sum;

}

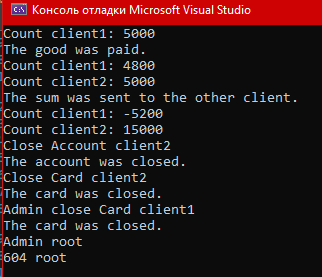
}

}

}

}

**Результат выполнения программы:**

****

**Вывод:** приобрёл практические навыки в области объектно-ориентированного проектирования.