



МИНИСТЕРСТВО ОБРАЗОВАНИЯ
РЕСПУБЛИКИ БЕЛАРУСЬ

Белорусский национальный
технический университет

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Лабораторный практикум

Минск
БНТУ
2014

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Белорусский национальный технический университет

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Лабораторный практикум

Минск
БНТУ
2014

УДК 004.42(076.5)

ББК 32.97-018я7

О-29

Составитель

А. В. Щербаков

Рецензент

О. И. Наранович

Объектно-ориентированное программирование : лабораторный
О-29 практикум / сост.: А. В. Щербаков. – Минск : БНТУ, 2014. – 38 с.

ISBN 978-985-550-391-1.

В практикуме приведены пять лабораторных работ по дисциплине «Объектно-ориентированное программирование». Каждая работа содержит краткие теоретические сведения, пример выполнения работы и индивидуальные задания. В качестве языка программирования используется C#.

УДК 004.42(076.5)

ББК 32.97-018я7

ISBN 978-985-550-391-1

© Белорусский национальный
технический университет, 2014

Содержание

Лабораторная работа № 1	
Класс, создание объекта класса. Понятие инкапсуляции.....	4
Лабораторная работа № 2	
Конструкторы. Статические члены класса.	
Шаблон проектирования Singleton.....	10
Лабораторная работа № 3	
Использование коллекций.....	18
Лабораторная работа № 4	
Наследование.....	28
Лабораторная работа № 5	
Полиморфизм.....	31

Лабораторная работа № 1

КЛАСС, СОЗДАНИЕ ОБЪЕКТА КЛАССА. ПОНЯТИЕ ИНКАПСУЛЯЦИИ

Цель работы: получить навыки проектирования простейших классов. Научиться создавать объекты класса. Освоить принцип инкапсуляции.

Краткие теоретические сведения

Класс – это тип, определяемый программистом, в котором объединяются структуры данных и функции их обработки. Переменные типа класс называются экземплярами класса и создаются при помощи оператора `new`.

Классы могут содержать переменные и константы, называемые полями. В классе могут быть объявлены функции, выполняющие действия над полями и именуемые методами.

Проектирование классов следует выполнять, придерживаясь стратегии минимальной связанности и зависимости между ними. Это достигается за счет использования принципа инкапсуляции. Инкапсуляция – это ограничение доступа к полям и методам при помощи модификаторов доступа. Основные модификаторы доступа в C#: `public` – доступ без ограничений; `private` – доступ разрешен только членам класса; `protected` – доступ разрешен как членам данного класса, так и производного. По умолчанию действует модификатор доступа `private`.

Пример объявления класса с закрытым полем и открытым методом:

```
class Employee
{
    private string name=«Петров»;
    public void PrintName()
    {
        Console.WriteLine («Name=«+name»);
    }
}
```

Кроме полей и методов в классе можно объявлять свойства. Свойства позволяют объявить два метода, один из которых вызывается при установке значения свойства (метод `set`), а второй при его чтении (метод `get`). Обычно код этих методов содержит обращение к полю, хранящему значение свойства. Возможно объявление свойства либо только для чтения, либо только для записи.

В методе записи `set` для доступа к записываемому значению используют ключевое слово `value`.

Пример класса `Employee` дополненного свойством для чтения и записи:

```
class Employee
{
    private string name=«Петров»;
    public string Name
    {
        get
        {
            return name;
        }
        set
        {
            name = value;
        }
    }
    public void PrintName()
    {
        Console.WriteLine («Name=«+name»);
    }
}
```

После создания объекта класса для доступа к открытым членам класса (полям, методам и свойствам) используют оператор точка. В одной программе можно создать несколько объектов одного класса.

Задание к работе

1. Выбрать предметную область согласно варианту индивидуального задания.

2. Спроектировать класс для выбранной предметной области.
3. Нарисовать диаграмму спроектированного класса.
4. Предусмотреть наличие у объекта полей, методов и свойств.
5. Предусмотреть наличие свойств только для записи.

Индивидуальные задания

1. Предметная область: **АТС**. В классе хранить информацию об адресе АТС, числе абонентов, абонентской плате (для всех абонентов одна). Реализовать метод для подсчета абонентской платы всех клиентов.

2. Предметная область: **Вокзал**. В классе хранить информацию о наименовании станции, стоимости билета (стоимость одинакова для всех направлений), числе мест, числе проданных билетов. Реализовать метод для подсчета общей стоимости всех непроданных билетов.

3. Предметная область: **ЖЭС**. В классе хранить информацию о районе, к которому принадлежит ЖЭС, номере ЖЭС, числе жильцов, оплате за месяц (для всех жильцов одна), числе оплативших. Реализовать метод для подсчета общей задолженности жильцов.

4. Предметная область: **Аэропорт**. В классе хранить информацию о названии аэропорта, стоимости билета (стоимость одинаковая), общем числе мест во всех самолетах, числе проданных билетов. Реализовать метод для подсчета общей стоимости всех проданных билетов.

5. Предметная область: **Банк**. В классе хранить информацию о наименовании банка, числе вкладов, размере вклада (все вклады одинаковые), размере процентной ставки. Реализовать метод для подсчета общей выплаты по процентам.

6. Предметная область: **Отдел кадров**. В классе хранить информацию о наименовании предприятия, числе работников, норме выработки часов в месяц (одна для всех работников), оплате за час, подоходном налоге. Реализовать метод для подсчета общей выплаты по подоходному налогу.

7. Предметная область: **Фирма грузоперевозок**. В классе хранить информацию об оплате за перевозку одной тонны грузов (не зависит от направления), о массе перевезенных грузов, наименовании фирмы. Реализовать метод для подсчета общей выручки фирмы.

8. Предметная область: **Гостиница**. В классе хранить информацию о названии гостиницы, числе заселенных мест, общем числе мест, оплате за день проживания (для всех жильцов одинаковая стоимость). Реализовать метод для подсчета общей выручки гостиницы.

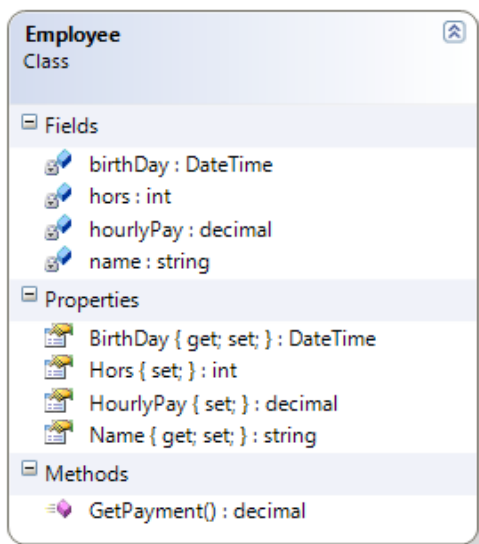
9. Предметная область: **Интернет-оператор**. В классе хранить информацию о стоимости тарифа (одна для всех пользователей), наименовании оператора, числе абонентов. Реализовать метод для подсчета общей выручки.

10. Предметная область: **Интернет-магазин** по продаже телевизоров. В классе хранить информацию о стоимости телевизора (одна для всех моделей), наименовании магазина, числе покупок. Реализовать метод для подсчета общей выручки.

Пример выполнения работы

Пусть задана предметная область: **Завод**. У работника завода хранить фамилию, год рождения, размер почасовой оплаты и количество отработанных часов. В классе реализовать метод для подсчета заработной платы работника, исходя из величины почасовой оплаты и отработанных часов.

Диаграмма спроектированного класса:



Текст программы:

```
using System;
// объявление пользовательского класса
class Employee
{
    string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
    //объявление закрытого поля
    DateTime birthDay;
    //объявление свойства для чтения и для записи
    public DateTime BirthDay
    {
        get { return birthDay; }
        set { birthDay = value; }
    }
    decimal hourlyPay;
    public decimal HourlyPay
    {
        set { hourlyPay = value; }
    }
    private int hors;
    public int Hors
    {
        set { hors = value; }
    }
    public decimal GetPayment()
    {
        return hors * hourlyPay;
    }
}
class Program
{
    static void Main()
```

```

{
//создание объекта класса
Employee ivanov = new Employee();
ivanov.Name = «Ivanov»;
ivanov.BirthDay = new DateTime(1977, 03,
18);
ivanov.Hors = 40;
ivanov.HourlyPay = 10.5M;
decimal p = ivanov.GetPayment();
Console.WriteLine(«Name: {0}
BirthDay:
{1}», ivanov.Name, ivanov.BirthDay);
Console.ForegroundColor = ConsoleCo-
lor.DarkRed;
Console.WriteLine(«Payment» + iva-
nov.GetPayment());
}
}

```

Результат работы:

Контрольные вопросы

1. В чем заключается принцип инкапсуляции?
2. При помощи какого ключевого слова создается объект класса?
3. Как объявить свойство только для чтения?
4. Чем поля класса отличаются от свойств?

Лабораторная работа № 2

КОНСТРУКТОРЫ. СТАТИЧЕСКИЕ ЧЛЕНЫ КЛАССА. ШАБЛОН ПРОЕКТИРОВАНИЯ SINGLETON

Цель работы: изучить работу и назначение конструкторов. Освоить возможности членов класса с модификатором static. Ознакомиться с шаблоном проектирования Singleton.

Краткие теоретические сведения

Конструктор – это метод, имеющий имя такое же, как и имя класса и не возвращающий параметров. Конструктор вызывается при создании объекта класса и служит для начальной инициализации полей и свойств. Как и любой метод класса, можно перегрузить конструктор, при этом вызов соответствующего конструктора будет определяться по списку параметров, который указывается при создании объекта класса.

Пример перегрузки конструктора:

```
class Item
{
    decimal price;

    public decimal Price
    {
        get { return price; }
        set { price = value; }
    }

    public Item()
    {
        price = 10;
    }

    public Item(decimal p)
    {
        price = p;
    }
}
```

Для вызова конструктора без параметров, именуемого конструктором по умолчанию, следует написать

```
Item i1=new Item();
```

Когда требуется вызвать конструктор с параметрами, передаваемые фактические значения указываются в круглых скобках, как показано ниже:

```
Item i1=new Item(50);
```

Если в одном конструкторе следует вызвать другой перегруженный конструктор, то после объявления первого следует поставить двоеточие и указать ключевое слово `this`. В круглых скобках после `this` через запятую указываются фактические значения, передаваемые конструктору.

Статический член класса объявляется с ключевым словом `static` и является независимым от всех объектов класса. Статическое поле или метод становится доступным до создания объекта класса. Для доступа к статическому члену за пределами класса достаточно указать имя этого класса и через оператор «точка» имя статического члена.

Шаблон проектирования Singleton – это порождающий шаблон, задачей которого является гарантия возможности создания только одного объекта класса и предоставление к этому объекту глобальной точки доступа.

Для применения шаблона проектирования Singleton к конкретному классу требуется: объявить в этом классе закрытый конструктор; объявить закрытую статическую ссылку на данный класс; добавить открытый статический метод, возвращающий ссылку на единственный созданный объект класса. Статический метод возвращает ссылку на единственный созданный объект класса и является глобальной точкой доступа к этому объекту.

Пример класса, написанного в соответствии с шаблоном Singleton:

```
class World
{
    private static World world;
```

```

private World()
{
}
public static World GetWorld()
{
    if (world==null) world = new World();
    return world;
}
}

```

Задание к работе

1. Спроектировать классы для выбранной предметной области.
2. Нарисовать диаграмму классов.
3. Применить к одному из классов шаблон проектирования Singleton.

Индивидуальные задания

Разработать два класса: класс-контейнер, управляющий контейнеризуемым классом, и контейнеризуемый класс. Для класса-контейнера применить шаблон проектирования Singleton.

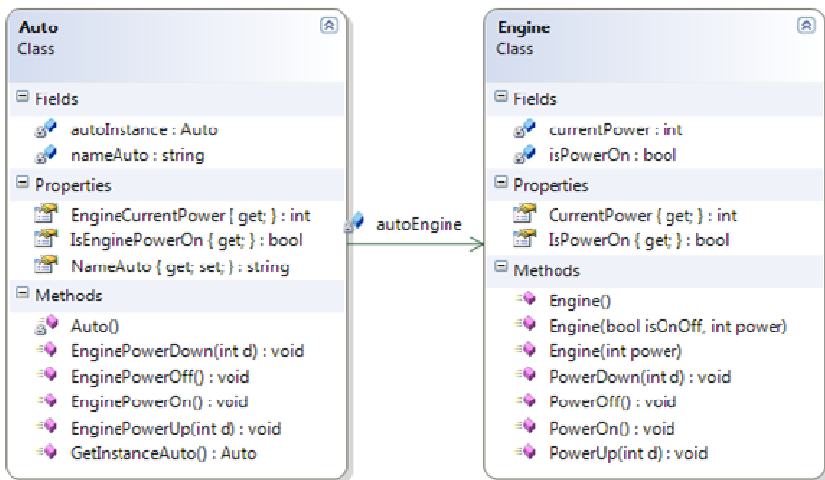
Описание предметной области:

1. Здание – Отопительная система.
2. Компьютер – Винчестер.
3. Больница – Приемное отделение.
4. Завод – Склад деталей.
5. Аэропорт – Взлетная полоса.
6. Вокзал – Богажное отделение.
7. Фирма – Отдел кадров.
8. Ресторан – Кухня.
9. Компьютер – Монитор.
10. Библиотека – Книгохранилище.

Пример выполнения работы

Пусть задана предметная область: Автомобиль – Двигатель автомобиля. Класс «автомобиль» является контейнерным классом, а класс «двигатель» – контейнеризуемым. В классе «двигатель» хранится информация о его состоянии (включен/выключен) и о текущей мощности.

Диаграмма классов:



Текст программы:

```
using System;
//класс двигатель
class Engine
{
    //конструктор
    public Engine()
    {
        isPowerOn = true;
        currentPower = 10;
    }
    //конструктор
    public Engine(int power)
    {
        isPowerOn = true;
        currentPower = power;
    }
    //конструктор
    public Engine(bool isOnOff, int power)
```

```

    {
        isPowerOn = isOnOff;
        if (!isPowerOn)
            currentPower = 0;
        else
            currentPower = power;
    }
// состояние двигателя
    private bool isPowerOn;

    public bool IsPowerOn
    {
        get { return isPowerOn; }
    }
//текущая мощность
    private int currentPower;
    public int CurrentPower
    {
        get { return currentPower; }
    }
// включить двигатель
    public void PowerOn()
    {
        isPowerOn = true;
    }
//выключить двигатель
    public void PowerOff()
    {
        isPowerOn = false;
        currentPower = 0;
    }
//поднять мощность
    public void PowerUp(int d)
    {
        if(isPowerOn)
            currentPower += d;
    }
//убавить мощность

```

```

        public void PowerDown(int d)
        {
            currentPower -= d;
        }
    }
    //класс автомобиль
    class Auto
    {
        //ссылка на объект класса Автомобиль
        private static Auto autoInstance;
        //ссылка на объект класса Двигатель
        private Engine autoEngine;
        //закрытый конструктор
        private Auto()
        {
            autoEngine = new Engine(false,0);
        }
        public static Auto GetInstanceAuto()
        {
            if (autoInstance == null)
            {
                autoInstance = new Auto();
            }
            return autoInstance;
        }
        //наименование автомобиля
        private string nameAuto;
        public string NameAuto
        {
            get { return nameAuto; }
            set { nameAuto = value; }
        }
        //состояние двигателя
        public bool IsEnginePowerOn
        {
            get
            {
                return autoEngine.IsPowerOn;
            }
        }
    }

```



```

    }

    public void EnginePowerOn()
    {
        autoEngine.PowerOn();
    }

    public void EnginePowerOff()
    {
        autoEngine.PowerOff();
    }
//текущая мощность автомобиля
    public int EngineCurrentPower
    {
        get
        {
            return autoEngine.CurrentPower;
        }
    }
//прибавить мощность
    public void EnginePowerUp(int d)
    {
        autoEngine.PowerUp(d);
    }
//убавить мощность
    public void EnginePowerDown(int d)
    {
        autoEngine.PowerDown(d);
    }
}
class Program
{
    static void Main(string[] args)
    {
        //создание объекта класса автомобиль

        Auto autol = Auto.GetInstanceAuto();
        autol.NameAuto = «BMW»;
        PrintInfoOfAuto(autol);
    }
}

```

```

        //включить двигатель
        autol.EnginePowerOn();
        //установить мощность
        autol.EnginePowerUp(20);
        PrintInfoOfAuto(autol);
        //выключить двигатель
        autol.EnginePowerOff();
        PrintInfoOfAuto(autol);
    }
    // вывод информации о параметрах
    автомобиля
    private static void PrintInfoOfAuto(Auto
    autol)
    {
        Console.WriteLine («{0}           двигатель
        включен
        {1}», autol.NameAuto, autol.IsEnginePowerOn);
        Console.WriteLine («Текущая           мощность
        (кВт) {0}»,
        autol.EngineCurrentPower);
    }
}

```

Контрольные вопросы

1. Чем конструктор отличается от обычного метода?
2. Какие преимущества дает перегрузка конструкторов?
3. Возможен ли доступ к статическому методу из экземплярного и наоборот?
4. Какое назначение шаблона проектирования Singleton?

Лабораторная работа № 3

ИСПОЛЬЗОВАНИЕ КОЛЛЕКЦИЙ

Цель работы: получить навыки проектирования приложения, состоящего из нескольких взаимосвязанных классов.

Краткие теоретические сведения

Чтобы хранить набор ссылок на объекты удобно использовать обобщенный класс `List`, объявленный в пространстве имен `System.Collections.Generic`. При создании объекта класса `List` в треугольных скобках указывается тип данных, который будет храниться в коллекции.

Пример объявления коллекции:

```
List<Employee> listEmployee = new List<Employee>();
```

Для обращения к элементам коллекции удобно использовать цикл `foreach`:

```
foreach (Employee emp in listEmployee  
{ //тело цикла }
```

Класс `List` унаследован от стандартных интерфейсов `ICollection`, `IEnumerable`, что позволяет выполнять основные действия над коллекцией: добавление элемента, удаление элемента, доступ через индекс.

В программе классы могут быть связаны отношением ассоциации, когда один класс содержит ссылки на другие классы. Различают кратность ассоциации «один к одному», когда один класс содержит ссылку на другой класс, и кратность «один ко многим», когда один класс содержит коллекцию ссылок на другой класс.

Ассоциации бывают однонаправленные и двунаправленные. Однонаправленные ассоциации предполагают навигацию от одного объекта к другому только в одном направлении. В случае двунаправленных ассоциаций ссылки на взаимно-ассоциированные объекты присутствуют как в первом, так и во втором классе.

Задание к работе

1. Для заданной предметной области спроектировать программную структуру, состоящую из 3–5 классов.
2. В соответствии с разработанной диаграммой классов выполнить программную реализацию.
3. Предусмотреть использование типа данных – перечисление.
4. Ввод/вывод должен быть реализован вне проектируемого класса.

Индивидуальные задания

1. Предметная область: **АТС**. На АТС хранится информация о всех клиентах станции. АТС имеет список тарифов на междугородние разговоры. Клиент АТС может совершать множество звонков в различные города.

Система должна:

- позволять вводить информацию о тарифах;
- вводить информацию о клиентах и регистрировать звонки;
- по введенной фамилии о клиенте определять стоимость всех сделанных им звонков в соответствии с действующими тарифами;
- вычислять общую стоимость всех выполненных на АТС звонков.

2. Предметная область: **Вокзал**. Касса вокзала имеет список тарифов на различные направления. При покупке билета регистрируются паспортные данные пассажира. Пассажир покупает билеты на различные направления.

Система должна:

- позволять вводить данные о тарифах;
- позволять вводить паспортные данные пассажира и регистрировать покупку билета;
- рассчитывать стоимость купленных пассажиром билетов;
- после ввода наименования направления выводить список всех пассажиров, купивших на него билет.

3. Предметная область: **ЖЭС**. В ЖЭС хранятся тарифы на коммунальные услуги. ЖЭС имеет информацию о всех жильцах. При потреблении жильцами коммунальных услуг информация регистрируется в системе.

Система должна позволять выполнять следующие задачи:

- ввод тарифов;
- ввод информации о жильцах и потребленных услугах;
- после ввода фамилии выводить сумму всех потребленных услуг;
- выводить стоимость всех оказанных услуг.

4. Предметная область: **Аэропорт**. Касса аэропорта имеет список тарифов на различные направления. При покупке билета регистрируются паспортные данные.

Система должна:

- позволять вводить данные о тарифах;
- позволять вводить паспортные данные пассажира и регистрировать покупку билета;
- рассчитывать стоимость купленных пассажиром билетов;
- рассчитывать стоимость всех проданных билетов.

5. Предметная область: **Банк**. Информационная система банка хранит описание процентов по различным вкладам. Система хранит информацию о вкладчиках и сделанных ими вкладах. Каждый клиент может поместить в банк только один вклад.

Система должна позволять выполнять следующие задачи:

- хранить информацию о процентах по вкладам;
- хранить информацию о клиентах;
- пополнять клиенту величину вклада;
- вычислять общую сумму выплат по процентам для всех вкладов.

6. Предметная область: **Отдел расчета зарплаты.** Информационная система отдела расчета зарплаты на предприятии хранит данные о величине оплаты за различные виды работ. Система хранит информацию о работниках предприятия.

Система должна позволять выполнять следующие задачи:

- вводить информацию о различных видах работ;
- вводить информацию о работниках и выполненных ими работах;
- после ввода фамилии выводить для работника зарплату;
- выводить сумму выплат всем работникам.

7. Предметная область: **Фирма грузоперевозок.** Фирма имеет список тарифов по перевозке грузов. Клиент регистрируется в системе, после чего может заказать перевозку определенного объема груза.

Система должна позволять выполнять следующие задачи:

- ввод тарифов;
- регистрация клиента и заказ на перевозку грузов;
- вывод суммы заказа для определенного клиента;
- подсчет суммарной стоимости всех заказов.

8. Предметная область: **Гостиница.** Информационная система гостиницы хранит информацию о всех номерах и их стоимости. Система регистрирует клиентов. Каждый клиент может заказать один номер. При попытке заказа номера, который занят, выводится предупреждение.

Система должна позволять выполнять следующие задачи:

- ввод информации о номерах и их стоимости;
- регистрация клиента и заказ номера;
- вывод списка не занятых номеров;
- после ввода фамилии клиента вывод стоимости проживания.

9. Предметная область: **Интернет-оператор.** Провайдер имеет различные тарифы доступа в Интернет за 1 Мбайт в зависимости от величины абонентской платы. Информационная система провайдера хранит данные о клиентах.

Система должна позволять выполнять следующие задачи:

- ввод тарифов;
- регистрация пользователя;
- ввод данных о потребленном трафике для конкретного пользователя;
- подсчет общей стоимости реализованного трафика;
- поиск клиента, заплатившего наибольшую стоимость за услуги.

10. Предметная область: **Интернет-магазин**. В информационной системе хранятся данные о товарах. Клиент звонит в магазин и оставляет заказ на товар.

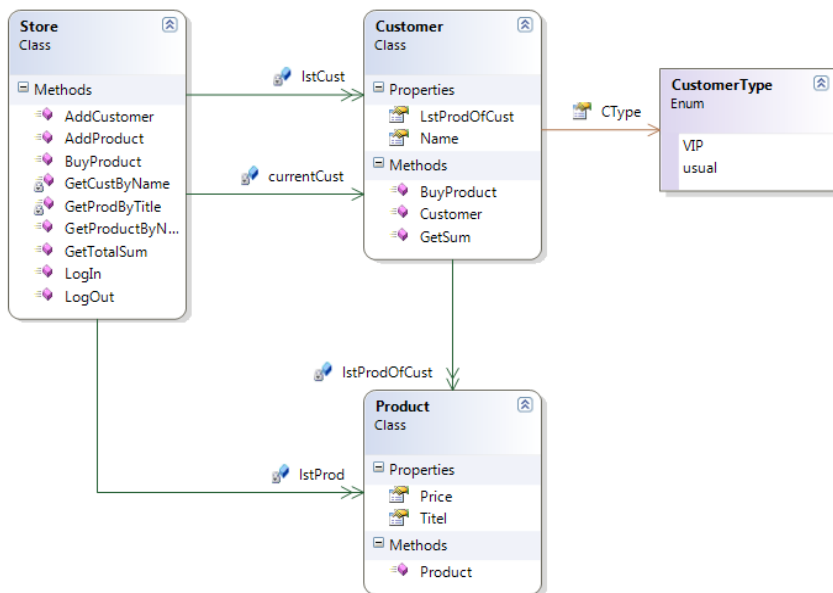
Система должна позволять выполнять следующие задачи:

- ввод информации о товарах;
- регистрация заказа клиента на покупку определенного товара;
- после ввода фамилии покупателя вывод списка заказанных им товаров;
- после ввода фамилии покупателя вывод суммы заказа.

Пример выполнения работы

Предметная область: **Торговая система**. В информационной системе хранятся данные о товарах и покупателях. Некоторые покупатели имеют статус «VIP». Для приобретения товара вводится фамилия, после чего регистрируются купленные товары. Система должна высчитывать общую сумму проданных товаров.

Диаграмма классов:



Текст программы:

```
using System;
using System.Collections.Generic;
class Product
{
    public string Titel { get; set; }
    public int Price { get; set; }
    public Product(string t, int p)
    {
        Titel = t;          Price = p;
    }
}

class Store
{

```



```

List<Product>          lstProd          =          new
List<Product>();
List<Customer>         lstCust          =          new
List<Customer>();
Customer currentCust;
public void AddProduct(string t, int p)
{
    lstProd.Add(new Product(t, p));
}
public void AddCustomer(string n, CustomerType ct)
{
    lstCust.Add(new Customer(n, ct));
}
Customer GetCustByName(string n)
{
    foreach (Customer item in lstCust)
    {
        if (item.Name == n) return item;
    }
    return null;
}

Product GetProdByTitle(string t)
{
    foreach (Product item in lstProd)
    {
        if (item.Titel == t) return item;
    }
    return null;
}

public void LogIn(string n)
{
    if (currentCust == null)
        currentCust = GetCustByName(n);
}

```

```

public void LogOut()
{
    currentCust = null;
}

public void BuyProduct(string titel)
{
    Product p = GetProdByTitle(titel);
    currentCust.BuyProduct(p);
}

public int GetTotalSum()
{
    int sum = 0;
    foreach (Customer c in lstCust)
    {
        foreach (Product p in
            c.LstProdOfCust)
        {
            sum += p.Price;
        }
    }
    return sum;
}

public string GetProductsByName(string
name)
{
    Customer c = GetCustByName(name);
    string s = «Customer:» + c.Name +
«\n»;
    foreach (Product p in c.LstProdOfCust)
    {
        s += p.Titel + « » + p.Price +
«\n»;
    }
    return s;
}

```

```

    }
}

enum CustomerType { VIP, usual };
class Customer
{
    List<Product>    lstProdOfCust    =    new
    List<Product>();
    public List<Product> LstProdOfCust
    {
        get { return lstProdOfCust; }
    }
    public string Name { get; set; }
    public CustomerType CType { get; set; }
    public Customer(string n, CustomerType ct)
    {
        Name = n;          CType = ct;
    }

    public void BuyProduct(Product p)
    {
        lstProdOfCust.Add(p);
    }

    public int GetSum()
    {
        int s = 0;
        foreach (Product item in lstProdOf-
Cust)
        {
            s += item.Price;
        }
        return s;
    }
}

class Program
{

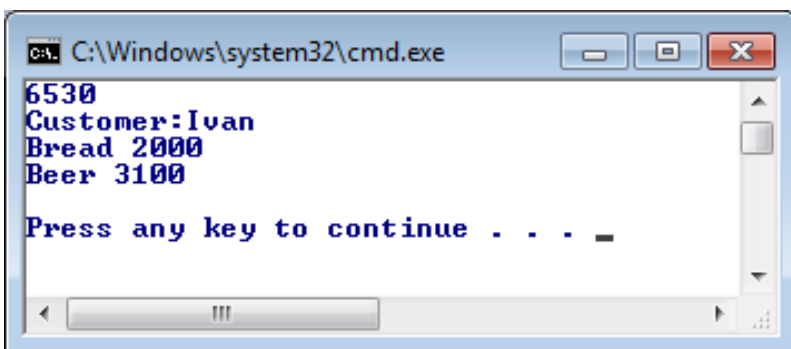
```

```

static void Main()
{
    Store riga = new Store();
    riga.AddProduct («Milk», 1430);
    riga.AddProduct («Bread», 2000);
    riga.AddProduct («Beer», 3100);
    riga.AddCustomer («Masha»,      Customer-
Type.VIP);
    riga.AddCustomer («Ivan»,      Customer-
Type.usual);
    riga.LogIn («Ivan»);
    riga.BuyProduct («Bread»);
    riga.BuyProduct («Beer»);
    riga.LogOut();
    riga.LogIn («Masha»);
    riga.BuyProduct («Milk»);
    riga.LogOut();
    Console.WriteLine (riga.GetTotalSum());
    Con-
sole.WriteLine (riga.GetProductsByName (
«Ivan»));
}
}

```

Результат работы программы:



```

C:\Windows\system32\cmd.exe
6530
Customer:Ivan
Bread 2000
Beer 3100

Press any key to continue . . . -

```

Контрольные вопросы

1. Как добавить элемент в коллекцию?
2. Как работает цикл `foreach`?
3. Для чего используют тип данных перечисление?

Лабораторная работа № 4

НАСЛЕДОВАНИЕ

Цель работы: освоить возможности использования наследования при проектировании классов объектно-ориентированного приложения.

Краткие теоретические сведения

Наследование служит для возможности повторного использования кода. Класс, от которого выполняется наследование, называется базовым. Производному классу становятся доступны члены базового класса с модификаторами `public` и `protected`. Для объявления производного класса после его имени ставится двоеточие и далее указывается имя базового.

Базовый класс представляет наиболее общие характеристики и поведение описываемого объекта. В производном классе определена более специфическая разновидность предмета описания базового класса. Примером иерархии наследования может служить базовый класс «Человек», с такими атрибутами как имя, возраст, адрес и производный класс «Работник», с атрибутами зарплата, должность.

Пример иерархии наследования:

```
class Man
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

```
class Employee:Man
```

```
{
    public decimal Payment { get; set; }
}
```

При проектировании классов, находящихся в отношении наследования, возможно объявление свойств и методов с одним и тем же именем как в базовом, так и в производном классе. При этом член производного класса будет «скрывать» доступ члену базового класса. В этом случае для обращения к одноименному члену базового класса нужно написать ключевое слово `base`.

Пример обращения к методу базового класса при его скрытии:

```
class Man
{
    public string Name { get; set; }
    public int Age { get; set; }
    public void Print()
    {
        Console.WriteLine («Name:»+Name+»
                           Age:»+Age);
    }
}

class Employee:Man
{
    public decimal Payment { get; set; }
    public void Print()
    {
        base.Print();
        Console.WriteLine («Payment:»      +
                           Payment);
    }
}
```

Иногда требуется наоборот запретить брать определенный класс в качестве базового, в этом случае следует перед объявлением класса указать ключевое слово `sealed`.

Задание к работе

1. Спроектировать иерархию, состоящую из 3–5 классов.
2. Запрограммировать классы в соответствии с новой диаграммой.
3. Изучить механизмы явного вызова конструкторов базовых классов и конструкторов этого же класса с использованием ключевого слова `this`.
4. Использовать в программе вызов методов базового класса из методов производного при его сокрытии.
5. Использовать модификатор доступа `protected`.
6. Создать класс, закрытый для наследования (`sealed`), обосновать его использование.
7. Использовать ключевое слово `struct`.
8. Проиллюстрировать использование модификаторов `ref` и `out`.

Индивидуальные задания

Спроектировать иерархию для заданной предметной области:

- 1) автотранспорт;
- 2) жилищно-коммунальная сфера;
- 3) здравоохранение;
- 4) бытовое обслуживание населения;
- 5) образование;
- 6) муниципальное управление;
- 7) железнодорожный транспорт;
- 8) авиаперевозки;
- 9) компьютерная техника;
- 10) энергетика.

Контрольные вопросы

1. В чем отличие оператора `is` от `as`?
2. В чем отличие модификатора `ref` от `out`?
3. Как можно использовать ключевое слово `base`?
4. Чем отличается значимый тип от ссылочного?

Лабораторная работа № 5

ПОЛИМОРФИЗМ

Цель работы: изучить механизмы реализации полиморфизма в C#. Ознакомиться с основными подходами при использовании интерфейсов. Изучить шаблон проектирования Strategy.

Краткие теоретические сведения

Полиморфизм позволяет классам с одинаковой спецификацией иметь различную реализацию, которая может быть изменена в процессе наследования.

Один из способов добавления полиморфного поведения в программу – это описание виртуальных методов. Виртуальным называется такой метод, который объявляется с ключевым словом `virtual` в базовом классе. Виртуальный метод отличается тем, что он может быть переопределен в одном или нескольких производных классах и у каждого производного класса может быть свой вариант реализации виртуального метода. При переопределении виртуального метода в производном классе указывается ключевое слово `override`.

В ряде случаев уровень абстрагирования, предоставляемый базовым классом, не предполагает какую-либо практическую реализацию некоторых методов класса. Такие методы объявляются с ключевым словом `abstract` и не содержат тело метода. Класс с абстрактными методами считается абстрактным, при этом становится запрещено создавать объекты этого класса. Как и в случае виртуального метода, абстрактный метод в производном классе переопределяется при помощи метода `override`.

Пример объявления абстрактного метода:

```
class Shape
{
    public abstract void Draw();
}
class Ellipse:Shape
{
```



```

public override void Draw()
{
    ///реализация
}

```

Развитием концепции абстрактных методов в C# является интерфейс. Для объявления интерфейса указывается ключевое слово `interface`, далее идет имя интерфейса и в фигурных скобках перечисляются имена методов без реализации. Класс может быть унаследован от одного или нескольких интерфейсов. Методы, реализующие интерфейс, должны быть объявлены как `public`. Имя интерфейса должно начинаться с буквы `I`.

Шаблон проектирования `Strategy` использует полиморфизм, для того чтобы определить семейство алгоритмов и сделать их взаимозаменяемыми. Для реализации шаблона `Strategy` можно объявить интерфейс, содержащий метод, предполагающий множество реализаций. Для каждого алгоритма реализации следует объявить класс, унаследованный от интерфейса и предоставляющий реализацию алгоритма.

Задание к работе

1. Составить диаграмму классов проектируемой системы.
2. Запрограммировать классы в соответствии с новой диаграммой.
3. Проиллюстрировать использование интерфейсов.
4. Показать вызов метода интерфейса через интерфейсную ссылку.
5. Применить в программе шаблон проектирования `Strategy`.

Индивидуальные задания

1. Предметная область: АТС. АТС имеет список тарифов на междугородние разговоры. Есть два типа тарифов: обычный и льготный. В классе «АТС» реализовать методы добавления обычного тарифа и добавления льготного тарифа. Класс «АТС» должен выполнять вычисление средней стоимости тарифов с учетом скидки.

2. Предметная область: Вокзал. Класс «вокзал» имеет список тарифов на различные направления. На некоторые тарифы может быть предоставлена скидка, заданная в процентах. В классе «вокзал» реализовать методы добавления нового тарифа со скидкой и без скидки, поиск направления с минимальной стоимостью.

3. Предметная область: ЖЭС. ЖЭС имеет информацию о всех жильцах. Имеются два типа жильцов со льготами и без льгот. В классе «ЖЭС» реализовать метод добавления нового жильца, имеющего и не имеющего льготы, а также метод подсчета стоимости всех оказанных услуг.

4. Предметная область: Аэропорт. Касса аэропорта имеет список тарифов на различные направления. Тариф содержит название направления и стоимость перевозки. На некоторые направления предоставляется фиксированная скидка. В классе «аэропорт» реализовать метод добавления нового тарифа и метод поиска направления с максимальной стоимостью.

5. Предметная область: Банк. Система хранит информацию о вкладчиках и сделанных ими вкладах. Класс «вкладчик» содержит имя вкладчика и величину вклада. Некоторым вкладчикам при создании вклада на счет может дополнительно перечисляться фиксированная сумма. В классе «банк» реализовать методы добавления нового вкладчика и метод вычисления общей суммы вкладов.

6. Предметная область: Отдел расчета зарплаты. Информационная система отдела расчета зарплаты на предприятии хранит данные о величине оплаты за различные виды работ. На некоторые виды работ предоставляется надбавка, заданная в процентах. В классе «отдел расчета зарплаты» реализовать методы добавления нового типа работ и метод вычисления средней величины оплаты.

7. Предметная область: Фирма грузоперевозок. Фирма имеет список тарифов по перевозке грузов. Класс «тариф» хранит наименование тарифа и цену. На некоторые тарифы предоставлена скидка, заданная в процентах. В классе фирма реализовать методы добавления нового тарифа и метод поиска тарифа с минимальной стоимостью.

8. Предметная область: Гостиница. Информационная система гостиницы хранит информацию о всех номерах и их стоимости. На проживание в некоторых номерах предоставляется скидка, заданная

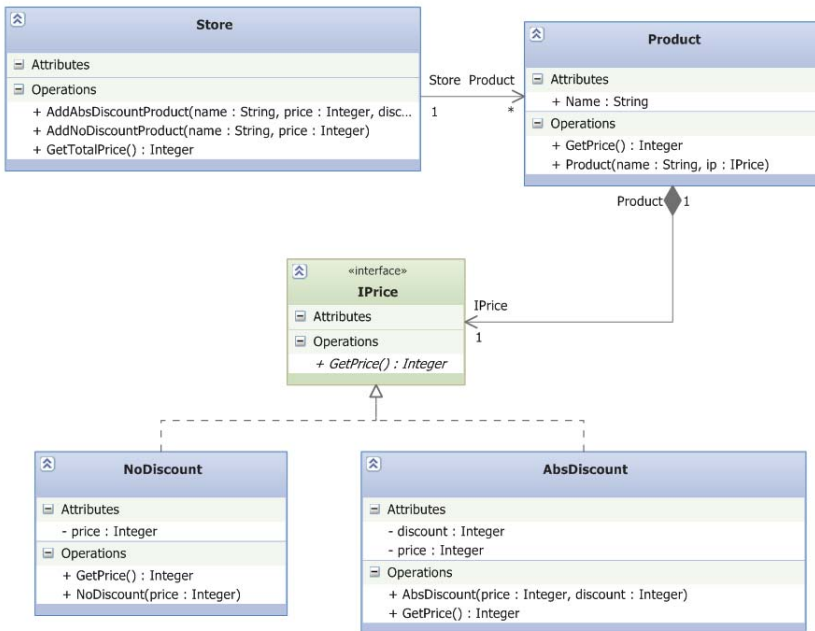
в процентах. В классе «гостиница» реализовать метод добавления информации о номере и метод вычисления средней стоимости.

9. Предметная область: Интернет-оператор. Информационная система провайдера хранит данные о клиентах. Некоторым клиентам предоставляется фиксированная скидка. В классе «оператор» реализовать метод добавления нового клиента и метод вычисления суммарной стоимости оказанных услуг.

10. Предметная область: Интернет-магазин. В информационной системе хранятся данные о товарах. Класс «товар» содержит стоимость товара и его наименование. На некоторые товары предоставляется скидка, заданная в процентах. В классе «магазин» реализовать метод добавления нового товара, имеющего скидку и не имеющего, также метод поиска товара с минимальной стоимостью.

Пример выполнения работы

UML диаграмма классов:



Текст программы:

```
using System;
using System.Collections.Generic;
interface IPrice
{
    int GetPrice();
}
class NoDiscount:IPrice
{
    int price;
    public NoDiscount(int p)
    {
        price = p;
    }
    public int GetPrice()
    {
        return price;
    }
}
class AbsDiscount:IPrice
{
    int price;
    int discount;
    public AbsDiscount(int p,int d)
    {
        price = p;
        discount = d;
    }
    public int GetPrice()
    {
        return price - discount;
    }
}

class Product
{
    public Product(string name,IPrice ip)
```

```

    {
        Name = name;
        this.ip = ip;
    }
    IPrice ip;
    public string Name { get; set; }
    public int GetPrice()
    {
        return ip.GetPrice();
    }
}
class Store
{
    List<Product>      lstProd      =      new
    List<Product>();
    public void AddNoDiscountProduct(string
name,int price)
    {
        Product p = new Product(name, new No-
Discount(price));
        lstProd.Add(p);
    }
    public void AddAbsDiscountProduct(string
name,int price,int discount)
    {
        Product p = new Product(name, new AbsDis-
count(price,discount));
        lstProd.Add(p);
    }
    public int GetTotalPrice()
    {
        int s = 0;
        foreach (Product p in lstProd)
        {
            s += p.GetPrice();
        }
        return s;
    }
}

```

```

}
class Program
{
    static void Main(string[] args)
    {
        Store riga = new Store();
        riga.AddAbsDiscountProduct (« bread»,
        3200, 200);
        riga.AddNoDiscountProduct («milk»,
        1200);
        Console.WriteLine (riga.GetTotalPrice());
    }
}

```

Контрольные вопросы

1. Каков механизм действия виртуальных функций?
2. Возможно ли множественное наследование интерфейсов?
3. Отличие интерфейса от абстрактного класса.
4. Для чего служат интерфейсы IEnumerable и IEnumerator?

Литература

1. Шилдт, Г. C# 4.0: полное руководство : пер. с англ. / Г. Шилдт. – М. : ООО «И. Д. Вильямс», 2011. – 1056 с.
2. Троелсен, Э. Язык программирования C# 2010 и платформа. Net 4.0 : пер. с англ. / Э. Троелсен. – 5-е изд. – М. : ООО «И. Д. Вильямс», 2011. – 1392 с.
3. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма [и др.]. – СПб. : Питер, 2009. – 366 с.
4. Пышкин, Е. В. Основные концепции и механизмы объектно-ориентированного программирования / Е. В. Пышкин. – СПб. : Питер, 2005. – 640 с.

Учебное издание

**ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ
ПРОГРАММИРОВАНИЕ**

Лабораторный практикум

Составитель

ЩЕРБАКОВ Александр Владимирович

Редактор *Л. Н. Шалаева*

Компьютерная верстка *А. Г. Занкевич*

Подписано в печать 23.12.2013. Формат 60×84 $\frac{1}{16}$. Бумага офсетная. Ризография.

Усл. печ. л. 2,21. Уч.-изд. л. 1,73. Тираж 100. Заказ 1134.

Издатель и полиграфическое исполнение: Белорусский национальный технический университет. ЛИ № 02330/0494349 от 16.03.2009. Пр. Независимости, 65. 220013, г. Минск.

