Частное учреждение образования

«Колледж бизнеса и права»

|  |
| --- |
| УТВЕРЖДАЮ  Заведующий  методическим кабинетом  \_\_\_\_\_\_\_\_\_\_\_\_\_Е.В.Фалей  « \_\_\_\_ » \_\_\_\_\_\_\_\_\_ 2016 |

|  |  |
| --- | --- |
| Специальность: 2-40 01 01 «Программное обеспечение информационных технологий» | Учебная практика по программированию |
| Составлена в соответствии с учебной программой, утвержденной директором Колледжа бизнеса и права 31.05.2016 | |

**Практическая работа №4**

**Инструкционно-технологическая карта**

Тема:Объектно-ориентированное программирование. Классы, объекты, методы.

* Принципы ООП. Классы как типы и объекты этих типов. Создание классов. Управление доступом к членам класса. Конструкторы. Деструктор (2 часа).
* Кодирование полей и свойств. Индексаторы (2 часа).
* Написание методов. Объявление параметров и передача аргументов. Возвращение данных и использование значений, возвращаемых методами (2 часа).
* Перегрузка методов и операторов (2 часа).

Цель: Развить умения по использованию принципов ООП. Научиться создавать классы и управлять доступом к членам класса. Закрепить умение работать с элементами класса. Развить умение объявлять и использовать методы. Развить умения по использованию перегрузки методов и операторов

Время выполнения: 8 часов

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Ознакомиться с теоретическими сведениями.
2. Получить вариант индивидуального задания и выполнить его.
3. Оформить решенные задания в отчет.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Основными понятиями ООП являются:

**Абстракция** – подразумевает разделение и независимое выполнение интерфейса и реализации (внутренние свойства объекта). Интерфейс – средство взаимодействия с другими объектами.

**Модульность** – предполагает скрытие деталей реализации.

Принцип абстракции обязывает использовать механизм запрета доступа или сокрытие компонент, который предотвращает умышленное или случайное изменение внутри реализации.

**Абстрактный тип данных (АТД)** - совокупность данных вместе со множеством операций над ними.

**Объектно-ориентированное программирование (ООП)** – метод построения программ в виде множества взаимодействующих объектов, структура и поведение которых описаны соответствующими классами.

Принципами ООП являются инкапсуляция, наследование, полиморфизм.

**Инкапсуляция** — это механизм программирования, объединяющий вместе код и данные, которыми он манипулирует, исключая как вмешательство извне, так и неправильное использование данных. В объектно-ориентированном языке данные и код могут быть объединены в совершенно автономный черный ящик. Внутри такого ящика находятся все необходимые данные и код. Когда код и данные связываются вместе подобным образом, создается объект. Иными словами, **объект** — это элемент, поддерживающий инкапсуляцию.

**Полиморфизм** (polymorphism) (от гр. polymorphos) – это свойство, которое позволяет одно и то же имя использовать для решения двух или более схожих, но технически разных задач. Целью полиморфизма, применительно к объектно-ориентированному программированию, является использование одного имени для задания общих для класса действий. Выполнение каждого конкретного действия будет определяться типом данных.

**Наследование** представляет процесс, в ходе которого один объект приобретает свойства другого объекта. Наследование является одной из наиболее важных особенностей ООП. Любой класс может наследоваться от другого класса, что означает, что он будет обладать всеми членами того класса, от которого он наследуется. В терминологии ООП класс, от которого происходит наследование, называется базовым классом, класс, который наследуется, называется производным классом.

**Классы как типы и объекты этих типов. Создание классов**.

*Класс* является **типом данных**, определяемым пользователем. Он должен представлять собой одну логическую сущность, например, является моделью реального объекта или процесса. *Элементами* класса являются *данные* и *функции*, предназначенные для их обработки.

Объекты (переменные типа класса, экземпляры класса) создаются явным или неявным образом, то есть либо программистом, либо системой.

Класс относится к **ссылочным типам данных,** память под которые выделяется в хипе. Таким образом, экземпляры классов хранят не сами объекты, а **ссылки** на объекты, то есть их адреса.

*Описание класса* содержит ключевое слово class, за которым следует его *имя*, а далее в фигурных скобках – *тело* класса, то есть список его элементов. Кроме того, для класса можно задать его базовые классы (предки) и ряд необязательных атрибутов и спецификаторов, определяющих различные характеристики класса. Синтаксис описания класса:

[ атрибуты ] [ спецификаторы ] class имя\_класса [ : предки ]

{

тело\_класса

}

Обязательными являются только ключевое слово class, а также имя и тело класса. *Имя класса* задается программистом по общим правилам C#. Тело класса – это список описаний его элементов, заключенный в фигурные скобки. Список может быть пустым, если класс не содержит ни одного элемента.

*Спецификаторы* определяют свойство класса, а также доступность класса для других элементов программы. Класс можно описывать непосредственно внутри пространства имен или внутри другого класса. В последнем случае называется **вложенным**.

Если классы описываются в пространстве имен непосредственно *(то есть не вложенные классы)*, тодля таких классов допускаются только два спецификатора**: public и internal.** *По умолчанию,* то есть если ни один спецификатор доступа не указан, подразумевается спецификатор *internal.*

**Управление доступом к членам класса**

При описании элементов класса можно так же указывать атрибуты и спецификаторы полей и констант, перечисленные ниже

* new – Новое описание поля, скрывающее указанный элемент класса;
* public – Доступ к элементу не ограничен;
* protected – Доступ только из данного и производных классов;
* internal – Доступ только из данной сборки;
* private – Доступ только из данного класса;
* static – Одно поле для всех экземпляров класса;
* readonly – Поле доступно только для чтения;
* volatile – Поле может изменяться другим процессом или системой.

Данные и методы являются основными элементами класса. Кроме того, в классе можно задавать целую гамму других элементов: свойства, события, индексаторы, операции, конструкторы, деструкторы, а также типы данных, внутренние по отношению к классу.

**Конструкторы.Деструктор**

*Конструктор* предназначен для инициализации объекта. Он вызывается автоматически при создании объекта класса с помощью операции new. Имя конструктора совпадает с именем класса.

**Особенности использования конструктора**

1. Имеет то же самое имя, что и класс
2. Не может иметь тип возвращаемого значения, даже void*.*
3. Вызывается конструктор автоматически в тот момент, когда создается экземпляр класса.
4. Нельзя вызвать конструктор в явном виде.
5. Конструктор, как и другие функции, может иметь параметры. Такой конструктор (с параметрами) называется конструктором инициализации.
6. Класс может иметь несколько конструкторов. Например, конструктор инициализации и конструктор по умолчанию (без параметров). Такой случай называется перегрузкой конструкторов.
7. Если в классе не объявлен ни один конструктор, компилятор сам создает функцию - конструктор класса.
8. Если программист не указал ни одного конструктора или какие-то поля не были инициализированы, полям значимых типов присваивается нуль, полям ссылочных типов – значение null.

Если при создании объектов требуется присваивать полю разные значения, это следует делать в конструкторе.

Деструктор – еще одна специальная функция класса. Во многих случаях требуется, чтобы были произведены какие-либо действия при окончании работы объекта. Это может быть освобождение памяти, восстановление экрана, закрытие файлов и т.д. В С# такой функцией является деструктор. При создании объекта, при выделении памяти под объект вызывается конструктор, а при освобождении памяти из-под объекта, т.е. при прекращении действия объекта, выполняется деструктор.

**Особенности использования конструктора**

1. Имеет то же самое имя, что и класс, но перед ним ставится знак тильды (~).
2. Не может иметь тип возвращаемого значения*.*
3. В отличие от конструктора не может иметь параметров.
4. Вызывается деструктор автоматически в тот момент, когда завершается работа с объектом класса.
5. Нельзя вызвать деструктор в явном виде.

Продемонстрируем работу деструктора

namespace DestructorCA

{

class Destruct

{

public int x;

// конструктор с параметром

public Destruct(int i) {

x = i;

}

//Вызывается при утилизации объекта

~Destruct() {

Console.WriteLine("Уничтожить "+x);

}

//Создает объект и тут же уничтожает его

public void Generator(int i) {

Destruct o = new Destruct(i);

}

}

class Program

{

static void Main(string[] args)

{

int count;

Destruct ob=new Destruct(0);

/\*А теперь создать большое число объектов. В какой-то

момент произойдет "сборка мусора"\*/

for (count = 1; count < 1000000000; count++)

ob.Generator(count);

Console.WriteLine("Готово!");

Console.ReadKey();

}

}

}

Эта программа работает следующим образом. Конструктор инициализирует пере­менную х известным значением. В данном примере переменная х служит в качестве идентификатора объекта. А деструктор выводит значение переменной х, когда объект утилизируется. Особый интерес вызывает метод **Generator** (), который создает и тут же уничтожает объект типа **Destruct.** Сначала в классе **DestructDemo** создается ис­ходный объект **ob** типа **Destruct,** а затем осуществляется поочередное создание и уни­чтожение 100 тыс. объектов. В разные моменты этого процесса происходит "сборка му­сора". Насколько часто она происходит — зависит от нескольких факторов, в том числе от первоначального объема свободной памяти, типа используемой операционной си­стемы и т.д. Тем не менее в какой-то момент начинают появляться сообщения, форми­руемые деструктором. Если же они не появятся до окончания программы, т.е. до того момента, когда будет выдано сообщение "Готово!", попробуйте увеличить число созда­ваемых объектов, повысив предельное количество подсчитываемых шагов в цикле **for.**

И еще одно важное замечание: метод **WriteLine** () вызывается в деструкторе **-Destruct** () исключительно ради наглядности данного примера его использования. Как правило, деструктор должен воздействовать только на переменные экземпляра, определенные в его классе.

В силу того, что порядок вызова деструкторов не определен точно, их не следует применять для выполнения действий, которые должны происходить в определенный момент выполнения программы. В то же время имеется возможность запрашивать "сборку мусора". Тем не менее инициализация «сборки мусора» вручную в большин­стве случаев не рекомендуется, поскольку это может привести к снижению эффектив­ности программы. Кроме того, у системы «сборки мусора» имеются свои особенно­сти — даже если запросить «сборку мусора» явным образом, все равно нельзя заранее знать, когда именно будет утилизирован конкретный объект.

**Кодирование полей и свойств. Индексаторы**

Свойства служат для организации доступа к полям класса. Как правило, свойство связано с закрытым полем класса и определяет методы его получения и установки. Синтаксис свойства:

[ атрибуты ] [спецификаторы] тип имя\_свойства

{

[get код\_доступа]

[set код\_доступа]

}

Значения спецификаторов для свойств и методов аналогичны. Чаще всего свойства объявляются как открытые (со спецификатором public), поскольку они входят в интерфейс объекта.

Код доступа представляет собой блоки операторов, которые выполняются при получении (get) или установки (set) свойства. Может отсутствовать либо часть get, либо set, но не обе одновременно. Если отсутствует часть set, свойство доступно тольно для чтения (read-only), если отсутствует часть get, свойство доступно только для записи (write-only).

Метод записи обычно содержит действия по проверке допустимости устанавливаемого значении, метод чтения может содержать, например, поддержку счетчика обращений к полю.

При обращении к свойству автоматически вызываются указанные в нем методы чтения и установки.

Синтаксически чтение и запись свойства выглядят почти как методы. Метод get должен содержать оператор return, возвращающий выражение, для типа которого должно существовать неявное преобразование к типу свойства. В методе set используется параметр со стандартным именем value, который содержит устанавливаемое значение.

Вообще говоря, свойство может и не связываться с полем. Фактически, оно описывает один или два метода, которые осуществляют некоторые действия над данными того же типа, что и свойства. В отличие от открытых полей, свойства обеспечивают разделение между внутренним состоянием объекта и его интерфейсом и, таким образом, упрощает внесение изменения в класс.

Индексатор — это особый вид свойства, которое можно добавлять в класс для обеспечения способа доступа, аналогичного доступу к массивам.

Индексатор (indexer)в C# является конструкцией, позволяющей программисту пользоваться привычным синтаксисом с квадратными скобками для обращения к объектам классов коллекций, являющихся членами класса.

В действительности индексаторы позволяют реализовать еще более сложные способы доступа, поскольку при необходимости разрешают определять и применять сложные типы параметров с использованием квадратных скобок; несмотря на это, применение простого численного индекса для доступа к элементам является наиболее распространенным.

Общий синтаксис индексатора:

спецификаторы тип\_возвращ\_значения this [список\_параметров]

{

get код\_доступа

set код\_доступа

}

Код доступа представляет собой блоки операторов, которые выполняются при получении (get) или установки (set) значения элемента массива. Если отсутствует часть set, индексатор доступен только для чтения (read only), если отсутствует часть get, то индексатор доступен только для записи (write only).

Список параметров содержит описание индексов, по которым выполняется доступ к элементу. Чаще всего это целочисленный индекс.

**Написание методов. Объявление параметров и передача аргументов. Возвращение данных и использование значений, возвращаемых методами**

Метод – это элемент (член) класса, который реализует любые действия, выполняемые классом. Методы определяют *поведение* класса. Метод – это законченный фрагмент кода, к которому можно обратиться по имени. Он описывается один раз, а вызываться столько, сколько это необходимо. Один и тот же метод может обрабатывать различные данные, переданные ему в качестве аргументов.

Синтаксис **заголовка** метода:

[спецификаторы] тип возвращаемого значения имя\_метода ([параметры])

{

[тело метода]

}

В качестве **спецификаторов** могут выступать спецификаторы void, virtual, extern, public,static и множество других, рассмотренных в следующих лекциях.

**Тело метода** задает последовательность действий (последовательность операторов), выполняемых методом. **Тип возвращаемого значения** определяет – значение какого типа будет вычислено с помощью данного метода.

После выполнения метода происходит возврат в точку вызова метода, передача туда значения, записанного в операторе return. Если метод не возвращает никакого значения, в его заголовке задается тип void, а оператор return отсутствует.

**Параметры метода** представляют собой локальные переменные, которые при вызове метода принимают значение соответствующего аргумента. Область действия параметра – весь метод.

*Пример простейшего метода*

public double Func1(int x, double y) //заголовок метода

{ //тело метода

double z = x \* y;

return z;

} //конец тела метода

Параметры, описываемые в заголовке метода, определяют множество значений аргументов, которые можно передавать в метод. Список аргументов при вызове метода должен соответствовать списку параметров по количеству, порядку следования и типу. Для каждого параметра должны задаваться его тип и имя.

При вызове метода выполняются следующие действия:

1. Вычисляются выражения, стоящие на месте аргументов
2. Выделяется память под параметры метода в соответствии с их типом
3. Каждому из параметров сопоставляется соответствующий аргумент
4. Выполняется тело метода
5. Если метод возвращает значение, оно передается в точку вызова. Если метод имеет тип void управление передается на оператор, следующий после вызова.

Количество аргументов должно соотстветствовать количеству параметров. На имена никаких ограничений не накладывается - могут совпадать или не совпадать.

Существует два способа передачи параметров:

1. по значению;
2. по ссылке.

При передаче *по значению* метод получает копии аргументов, и операторы внутри метода работают с этими копиями. Доступа к исходным значениям аргументов у метода нет, и, следовательно, нет возможности их изменять.

При передаче *по ссылке (по адресу)* метод получает адреса аргументов, осуществляет доступ к ячейкам памяти по этимадресам, следовательно может изменять значения аргументов, модифицируя параметры.

В языке C# для обмена данными между вызывающей и вызываемой функциями есть четыре типа параметров: параметры-значения, параметры-ссылки, выходные параметры, параметры-массивы.

Параметры-значения задаются в заголовке метода обычным образом, например:

void func(int x)

В данном примере параметр х – это параметр-значение, который предсталяет собой локальную переменную, которая получает свое значение при вызове метода func. В метод передается **копия** значения аргумента для х.

Во многих методах все величины, которые метод должен получить в качестве исходных данных, описываются в списке перемтров, а величина, которую вычислет метод как результат своей работы, возвращается в вызывающий код с помощью оператора return. Если метод должен возвращать более,чем одно значение, такой способ не годится.

Также, если в методе требуется изменить значение передаваемых в него величин используются параметры –ссылки.

Параметры-ссылки задаются в заголовке метода с использованием ключевого слова ref, например:

void func(**ref** int x)

При вызове такого метода параметр копирует не значение аргумента для х, а его адрес, поэтому внутри метода значение переданного параметра изменяется.

Часто возникает необходимость в методах, которые формируют несколько величин. Параметру, имеющему спецификатор out, должно быть присвоено значение внутри метода. Тем самым, можно из метода получить несколько значений-результатов работы.

void func(**out** int x)

**Перегрузка методов и операторов**

В C# допускается совместное использование одного и того же имени двумя или бо­лее методами одного и того же класса, при условии, что их параметры объявляются по-разному. В этом случае говорят, что методы *перегружаются,* а сам процесс называ­ется *перегрузкой методов.* Перегрузка методов относится к одному из способов реализа­ции полиморфизма в С#.

В общем, для перегрузки метода достаточно объявить разные его варианты, а об остальном позаботится компилятор. Но при этом необходимо соблюсти следующее важное условие: тип или число параметров у каждого метода должны быть разными. Совершенно недостаточно, чтобы два метода отличались только типами возвращае­мых значений. Они должны также отличаться типами или числом своих параметров. (Во всяком случае, типы возвращаемых значений дают недостаточно сведений ком­пилятору С#, чтобы решить, какой именно метод следует использовать.) Разумеется, перегружаемые методы могут отличаться и типами возвращаемых значений. Когда вызывается перегружаемый метод, то выполняется тот его вариант, параметры кото­рого соответствуют (по типу и числу) передаваемым аргументам.

В C# определено понятие сигнатуры, обозначающее имя метода и список его па­раметров; Применительно к перегрузке это понятие означает, что в одном классе не должно существовать двух методов с одной и той же сигнатурой. Следует подчеркнуть, что в сигнатуру не входит тип возвращаемого значения, поскольку он не учитывается, когда компилятор C# принимает решение о перегрузке метода. В сигнатуру не входит также модификатор params.

ВАРИАНТЫ ИНДИВИДУАЛЬНЫХ ЗАДАНИЙ

**Задание 1**. Каждый разрабатываемый класс должен, как правило, содержать следующие элементы: скрытые поля, конструкторы с параметрами и без параметров, методы, свойства. Методы и свойства должны обеспечивать непротиворечивый, полный минимальный и удобный интерфейс класса.

1. Разработать собственный класс, содержащий методы для вычисления суммы  и произведения , где m и n заданные числа.
2. Разработать собственный класс, содержащий метод для вычисления суммы , где n заданное число и 
3. Разработать собственный класс, содержащий метод для вычисления произведения , где n=10 и f(x) =
4. Разработать собственный класс, содержащий метод для вычисления приближенного значения cos(x) по формуле у=1-x2/2!+x4/4!-….+x2n/(2n)! для заданного числа n
5. Разработать собственный класс, содержащий методы для вычисления минимума и максимума из двух чисел, а также метод для вычисления значения выражения:

.

1. Разработать собственный класс, содержащий методы для вычисления минимума от двух чисел и максимума из трех чисел, а также метод для вычисления значения выражения:

.

1. Разработать собственный класс, содержащий методы для вычисления минимума и максимума из двух чисел, а также метод для вычисления значения выражения:

.

1. Разработать собственный класс, содержащий методы для вычисления минимума и максимума из двух чисел, а также метод для вычисления значения выражения:

.

1. Разработать собственный класс, содержащий методы для вычисления минимума и максимума из двух чисел, а также метод для вычисления значения выражения:

.

1. Разработать собственный класс, содержащий метод для определения значения функции Y. Для вычисления значения аргумента х реализовать свой пользовательский метод:

, где 

1. Разработать собственный класс, содержащий метод для нахождения суммы положительных элементов двумерного массива А(3,5).
2. Разработать собственный класс, содержащий методы для нахождения количества отрицательных элементов целочисленного массива В (100) и суммы элементов, стоящих на четных местах в массиве.
3. Разработать собственный класс, содержащий метод для нахождения произведения отрицательных элементов двумерного массива А(3,5).
4. Разработать собственный класс, содержащий метод для вычисления суммы диагональных элементов действительной матрицы размера [m, n].
5. Разработать собственный класс, содержащий методы для нахождения количества ненулевых элементов и вычисления суммы отрицательных элементов массива целых чисел А[20].
6. Разработать собственный класс, содержащий метод для нахождения количества элементов равных нулю в двумерном массиве А(3,5).
7. Разработать собственный класс, содержащий метод для формирования одномерного массива из диагональных элементов целочисленной матрицы размера [m, n].
8. Разработать собственный класс, содержащий методы для вычисления количества и суммы элементов, кратных 5 для одномерного массива целых чисел.
9. Разработать собственный класс, содержащий методы для вычисления произведения нечетных элементов одномерного массива целых чисел и нахождения количества элементов, не превышающих заданного числа n.
10. Разработать собственный класс, содержащий методы длянахождения количества отрицательных элементов одномерного массива целых чисел и позиции максимального элемента.
11. Разработать собственный класс, содержащий метод для нахождения номера строки и столбца, на пересечении которых находится минимальный элемент матрицы размерностью [m, n].
12. Разработать собственный класс, содержащий метод для нахождения количества отрицательных элементов двумерного массива А(3,5).
13. Разработать собственный класс, содержащий методы для нахождения значений z1 и z2



Результаты должны совпадать, если решение выполнено правильно.

1. Разработать собственный класс, содержащий метод для нахождения значения функции  где 

Для проверки значения z использовать тернарную операцию.

1. Разработать собственный класс, содержащий метод для нахождения значения функции



**Задание 2.** Каждый разрабатываемый класс должен, как правило, содержать следующие элементы: скрытые поля, конструкторы с параметрами и без параметров, методы; свойства, индексаторы; перегруженные операции. Функциональные элементы класса должны обеспечивать непротиворечивый, полный, минимальный и удобный интерфейс класса. При возникновении ошибок должны выбрасываться исключения. В программе должна выполняться проверка всех разработанных элементов класса.

Вариант 1

Описать класс для работы с одномерным массивом целых чисел (вектором). Обеспечить следующие возможности: задание произвольных целых границ индексов при создании объекта; обращение к отдельному элементу массива с контролем выхода за пределы массива; выполнение операций поэлементного сложения и вычитания массивов с одинаковыми границами индексов; выполнение операций умножения и деления всех элементов массива на скаляр; вывод на экран элемента массива по заданному индексу и всего массива. Написать программу, демонстрирующую. все разработанные элементы класса.

**Вариант 2**

Описать класс для работы с одномерным массивом строк фиксированной длины. Обеспечить следующие возможности: задание произвольных целых границ индексов при создании объекта; обращение к отдельной строке массива по индексу с контролем выхода за пределы массива; выполнение операций поэлементного сцепления двух массивов с образованием нового массива; выполнение операций слияния двух массивов с исключением повторяющихся элементов; вывод на экран элемента массива по заданному индексу и всего массива. Написать программу, демонстрирующую все разработанные элементы класса.

Вариант 3

Описать класс многочленов от одной переменной, задаваемых степенью многочлена и массивом коэффициентов. Обеспечить следующие возможности: вычисление значения многочлена для заданного аргумента; операции сложения, вычитания и умножения многочленов с получением но­вого объекта-многочлена; получение коэффициента, заданного по индексу; вывод на экран описания многочлена. Написать программу, демонстрирующую все разработанные элементы класса.

Вариант 4

Описать класс, обеспечивающий представление матрицы произвольного размера с возможностью изменения числа строк и столбцов, вывода на экран подматрицы любого размера и всей матрицы, доступа по индексам к элементу матрицы. Написать программу, демонстрирующую все разработанные элементы класса.

Вариант 5

Описать класс для работы с восьмеричным числом, хранящимся в виде строки символов. Реализовать конструкторы, свойства, методы и следующие операции: операции присваивания, реализующие значимую семантику; операции сравнения; преобразование в десятичное число; форматный вывод; доступ к заданной цифре числа по индексу. Написать программу, демонстрирующую все разработанные элементы класса.

**Вариант 6**

Описать класс «Домашняя библиотека». Предусмотреть возможность работы с произвольным числом книг, поиска книг по какому-либо признаку (по автору, году издания или категории), добавления книг в библиотеку, удаления книг из нее, доступа к книге по номеру. Написать программу, демонстрирующую все разработанные элементы класса.

**Вариант 7**

Описать класс «студенческая группа». Предусмотреть возможность работы с переменным числом студентов, поиска студента по какому-либо признаку (например, по фамилии, имени, дате рождения), добавления и удаления записей, сорти­ровки по разным полям, доступа к записи по номеру. Написать программу, демонстрирующую все разработанные элементы класса.

**Вариант 8**

Описать класс «записная книжка». Предусмотреть возможность работы с произвольным числом записей, поиска записи по какому-либо признаку (например, по фамилии, дате рождения или номеру телефона), добавления и удаления записей, сортировки по фамилии и доступа к записи по номеру. Написать программу, демонстрирующую все разработанные элементы класса.

**Вариант 9**

Описать класс, реализующий тип данных «вещественная матрица» и работу с ними. Класс должен реализовывать следующие операции над матрицами: сложение, вычитание (как с другой матрицей, так и с числом); комбинированные операции присваивания (+=, -=); операции сравнения на равенство/неравенство; операции вычисления обратной и транспонированной матрицы; доступ к элементу по индексам. Написать программу, демонстрирующую все разработанные элементы класса.

**Вариант 10**

Описать класс, реализующий тип данных «вещественная матрица» и работу с ними. Класс должен реализовывать следующие операции над матрицами: умножение, деление (как на другую матрицу, так и на число); комбинированные операции присваивания (\*=, /=); операцию возведения в степень; методы вычисления детерминанта и нормы; доступ к элементу по индексам. Написать программу, демонстрирующую все разработанные элементы класса.

**Вариант 11**

Описать класс, реализующий тип данных «вещественная матрица» и работу с ними. Класс должен реализовывать следующие операции над матрицами: методы, реализующие проверку типа матрицы (квадратная, диагональная, нулевая, единичная, симметричная, верхняя треугольная, нижняя треугольная); операции сравнения на равенство/неравенство; доступ к элементу по индексам. Написать программу, демонстрирующую все разработанные элементы класса.

Вариант 12

Описать класс «множество», позволяющий выполнять основные операции: добавление и удаление элемента, пересечение, объединение и разность множеств. Написать программу, демонстрирующую все разработанные элементы класса.

Вариант 13

Описать класс «предметный указатель». Каждый компонент указателя содержит слово и номера страниц, на которых это слово встречается. Количество номеров страниц, относящихся к одному слову, от одного до десяти. Предусмотреть воз­можность формирования указателя с клавиатуры и из файла, вывода указателя, вывода номеров страниц для заданного слова, удаления элемента из указателя. Написать программу, демонстрирующую все разработанные элементы класса.

Вариант 14

Описать класс «автостоянка» для хранения сведений об автомобилях. Для каж­дого автомобиля записываются госномер, цвет, фамилия владельца и признак присутствия на стоянке. Обеспечить возможность поиска автомобиля по разным критериям, вывода списка присутствующих и отсутствующих на стоянке авто­мобилей, доступа к имеющимся сведениям по номеру места. Написать программу, демонстрирующую все разработанные элементы класса.

Вариант 15

Описать класс «колода карт», включающий закрытый массив элементов класса «карта». В карте хранятся масть и номер. Обеспечить возможность вывода кар­ты по номеру, вывода всех карт, перемешивания колоды и выдачи всех карт из колоды поодиночке и по 6 штук, в случайном порядке. Написать программу, демонстрирующую все разработанные элементы классов.

Вариант 16

Описать класс «поезд», содержащий следующие закрытые поля: название пункта назначения; номер поезда (может содержать буквы и цифры); время отправления. Предусмотреть свойства для получения состояния объекта. Описать класс «вокзал», содержащий закрытый массив поездов. Обеспечить сле­дующие возможности: вывод информации о поезде по номеру с помощью индекса; вывод информации о поездах, отправляющихся после введенного с клавиату­ры времени; перегруженную операцию сравнения, выполняющую сравнение времени от­правления двух поездов; вывод информации о поездах, отправляющихся в заданный пункт назначения. Информация должна быть отсортирована по времени отправления. Написать программу, демонстрирующую все разработанные элементы классов.

Вариант 17

Описать класс «товар», содержащий следующие закрытые поля: название товара; название магазина, в котором продается товар; стоимость товара в рублях. Предусмотреть свойства для получения состояния объекта. Описать класс «склад», содержащий закрытый массив товаров. Обеспечить сле­дующие возможности: вывод информации о товаре по номеру с помощью индекса; вывод на экран информации о товаре, название которого введено с клавиату­ры; если таких товаров нет, выдать соответствующее сообщение; сортировку товаров по названию магазина, по наименованию и по цене; перегруженную операцию сложения товаров, выполняющую сложение их цен. Написать программу, демонстрирующую все разработанные элементы классов.

Вариант 18

Описать класс «самолет», содержащий следующие закрытые поля: название пункта назначения; шестизначный номер рейса; время отправления. Предусмотреть свойства для получения состояния объекта. Описать класс «аэропорт», содержащий закрытый массив самолетов. Обеспечить следующие возможности: вывод информации о самолете по номеру рейса с помощью индекса; вывод информации о самолетах, отправляющихся в течение часа после вве­денного с клавиатуры времени; вывод информации о самолетах, отправляющихся в заданный пункт назначения; перегруженную операцию сравнения, выполняющую сравнение времени отправления двух самолетов. Информация должна быть отсортирована по времени отправления. Написать программу, демонстрирующую все разработанные элементы классов.

**Вариант 19**

Описать класс «запись», содержащий следующие закрытые поля: фамилия, имя; номер телефона; дата рождения (массив из трех чисел). Предусмотреть свойства для получения состояния объекта. Описать класс «записная книжка», содержащий закрытый массив записей. Обес­печить следующие возможности: вывод на экран информации о человеке, номер телефона которого введен с клавиатуры; если такого нет, выдать на дисплей соответствующее сообщение; поиск людей, день рождения которых сегодня или в заданный день; поиск людей, день рождения которых будет на следующей неделе; поиск людей, номер телефона которых начинается на три заданных цифры. Написать программу, демонстрирующую все разработанные элементы классов.

Вариант 20

Описать класс «англо-русский словарь», обеспечивающий возможность хранения нескольких вариантов перевода для каждого слова. Реализовать доступ по строковому индексу — английскому слову. Обеспечить возможность вывода всех значений слов по заданному префиксу.

ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ

1. Албахари, Дж. C#5.0. Справочник. Полное описание языка / Дж. Албахари, Б. Албахари. – 5-е изд. – М.: ООО «И.Д. Вильямс», 2014. – 1008 с.: ил.
2. Дейтел, П. Как программировать на Visual C# 2012 / П. Дейтел, Х. Дейтел. – 5-е изд. – СПб.: Питер, 2014. – 864 с.: ил.
3. Майо, Дж. Самоучитель Microsoft Visual Studio 2010 / Дж. Майо. – СПб.: БХВ-Петербург, 2011. – 464 с.: ил.
4. Павловская Т.А. C#. Программирование на языке высокого уровня / Т.А. Павловская. – Учебник для вузов. – СПб.: Питер, 2014. – 432 с.: ил.
5. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Дж. Рихтер. – 4-е изд. – СПб.: Питер, 2013. – 896 с.: ил.
6. Стиллмен, Э. Изучаем C# / Э. Стиллмен, Дж. Грин. – 3-е изд. – СПб.: Питер, 2014. – 816 с.: ил.
7. Троелсен, Э. Язык программирования C# 5.0 и платформа .NET 4.5 / Э. Троелсен. – 6-е изд. – М.: ООО «И.Д. Вильямс», 2013. – 1312 с.: ил.
8. Шилдт, Г. C#4.0: Полное руководство / Г. Шилдт. – М.: ООО «И.Д. Вильямс», 2011. – 1056 с.: ил.
9. Шарп Джон. Microsoft Visual C#. Подробное руководство / Джон Шарп. – 8-е изд. – СПб.: Питер, 2017. – 848 с.
10. Полное руководство по языку программирования С# 6.0 и платформе .NET 4.6 [Электронный ресурс] / Сайт о программировании. – metanit.com, 2012-2016. – Режим доступа: [http://metanit.com/sharp/ tutorial](http://metanit.com/sharp/%20tutorial). – Дата доступа: 20.08.2016.

Преподаватель Е.В. Багласова

|  |
| --- |
| Рассмотрено на заседании цикловой  комиссии программного обеспечения информационных технологий № 10  Протокол №\_\_\_\_\_от «\_\_\_»\_\_\_\_2016  Председатель ЦК *( )* Т.Г.Багласова |