

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Саратовский государственный технический университет
имени Гагарина Ю.А.»

Филиал федерального государственного бюджетного образовательного
учреждения высшего образования
«Саратовский государственный технический университет
имени Гагарина Ю.А.» в г. Петровске




МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

по дисциплине

МДК. 01.01 «Разработка программных модулей»

направление подготовки

09.02.07 «Информационные системы и программирование»

Методические указания рассмотрены
на заседании предметной (цикловой)
комиссии общепрофессиональных
дисциплин, профессиональных модулей
специальностей технического профиля
«14» июня 2023 года, протокол №12
Председатель ПЦК /Лескина Т.А./

Петровск 2023

Пояснительная записка

Методические указания по выполнению лабораторных работ подготовлены на основе рабочей программы учебной дисциплины МДК. 01.01 «Разработка программных модулей», разработанной на основе ФГОС СПО по специальности 09.02.07 «Информационные системы и программирование» и соответствующих общих (ОК) и профессиональных (ПК) компетенций:

ОК 01. Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02. Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях.

ОК 04. Эффективно взаимодействовать и работать в коллективе и команде.

ОК 05. Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 06. Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей, в том числе с учетом гармонизации межнациональных и межрелигиозных отношений, применять стандарты антикоррупционного поведения

ОК 07. Содействовать сохранению окружающей среды, ресурсосбережению, применять знания об изменении климата, принципы бережливого производства, эффективно действовать в чрезвычайных ситуациях

ОК 08. Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности

ОК 09. Пользоваться профессиональной документацией на государственном и иностранном языках.

ОК 10. Использовать знания по финансовой грамотности, планировать предпринимательскую деятельность в профессиональной сфере

ПК 1.1. Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием

ПК 1.2. Разрабатывать программные модули в соответствии с техническим заданием.

При выполнении лабораторных работ студент должен *знать*:

- основные этапы разработки программного обеспечения;
- основные принципы технологии структурного и объектно-ориентированного программирования;
- способы оптимизации и приемы рефакторинга;
- основные принципы отладки и тестирования программных продуктов

При выполнении лабораторных работ студент должен *уметь*:

- осуществлять разработку кода программного модуля на языках низкого и высокого уровней;
- создавать программу по разработанному алгоритму как отдельный модуль;
- выполнять отладку и тестирование программы на уровне модуля; осуществлять разработку кода программного модуля на современных языках программирования;
- уметь выполнять оптимизацию и рефакторинг программного кода;
- оформлять документацию на программные средства

Содержание лабораторных занятий определено рабочей программой и тематическим планированием, соответствует теоретическому материалу изучаемых разделов учебной дисциплины.

Объем лабораторных занятий по дисциплине определяется учебным планом по данной специальности.

Продолжительность лабораторной работы – 2 академических часа. Перед проведением лабораторной работы преподавателем организуется инструктаж, а по ее окончании – обсуждение итогов.

Комплект методических указаний по выполнению лабораторных работ по дисциплине МДК. 01.01 «Разработка программных модулей» содержит 5 лабораторных работ.

**Перечень лабораторных работ
по дисциплине МДК. 01.01 «Разработка программных модулей»**

ЛАБОРАТОРНАЯ РАБОТА № 1

Тема: Перегрузка методов

ЛАБОРАТОРНАЯ РАБОТА № 2

Тема: Определение операций в классе

ЛАБОРАТОРНАЯ РАБОТА № 3

Тема: Создание наследованных классов

ЛАБОРАТОРНАЯ РАБОТА № 4

Тема: Создание хранимых процедур

ЛАБОРАТОРНАЯ РАБОТА № 5

Тема: Создание хранимых процедур

ЛАБОРАТОРНАЯ РАБОТА №1

Тема: Перегрузка методов

Цель работы: научиться работать с методами, написать программу с использованием методов.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

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 C#.

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 C#, 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.

Порядок выполнения работы:

Задание 1. Использование перегрузки методов

1. Запустите Visual Studio
2. Создайте консольное приложение
3. Запишите следующий код:

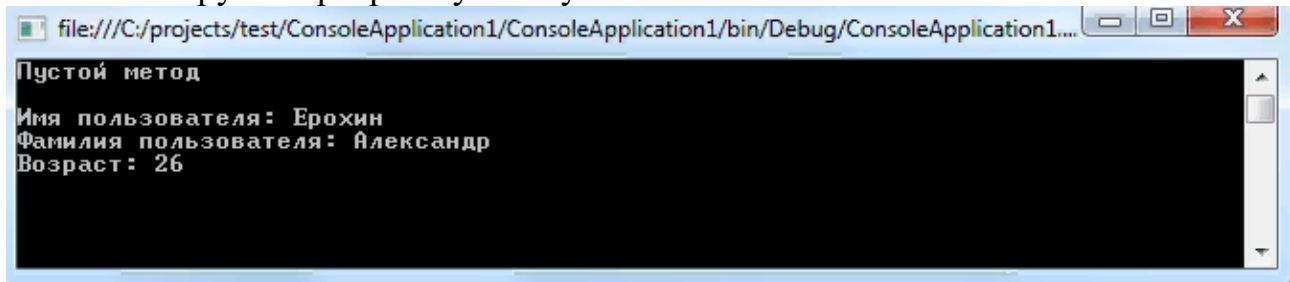
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class UserInfo
    {
        // Перегружаем метод ui
        public void ui()
        {
            Console.WriteLine("Пустой метод\n");
        }
        public void ui(string Name)
        {
            Console.WriteLine("Имя пользователя: {0}",Name);
        }
        public void ui(string Name, string Family)
        {
            Console.WriteLine("Имя пользователя: {0}\nФамилия пользователя: {1}",Name,Family);
        }
        public void ui(string Name, string Family, byte Age)
        {
```

```

        Console.WriteLine("Имя пользователя: {0}\nФамилия пользователя: {1}\nВозраст: {2}", Name, Family, Age);    }    }
    class Program    {
        static void Main(string[] args)    {
            UserInfo user1 = new UserInfo();
            // Разные реализации вызова перегружаемого метода
            user1.ui();
            user1.ui("Ерохин", "Александр", 26);
            Console.ReadLine();    }    }

```

4.Скомпилируйте программу и запустите ее на выполнение:



Как видите метод `ui` перегружается три раза. Модификаторы параметров `ref` и `out` также учитываются, когда принимается решение о перегрузке метода. Несмотря на то, что модификаторы параметров `ref` и `out` учитываются, когда принимается решение о перегрузке метода, отличие между ними не столь существенно.

5.Давайте добавим еще одну перегрузку в вышеуказанный пример:

```

// Используем модификатор параметров
public void ui(string Name, string Family, ref byte Age)
{
}

```

Задание 2. Создать перегрузку встроенного метода `IndexOf` класса `String` пространства имен `System`

```

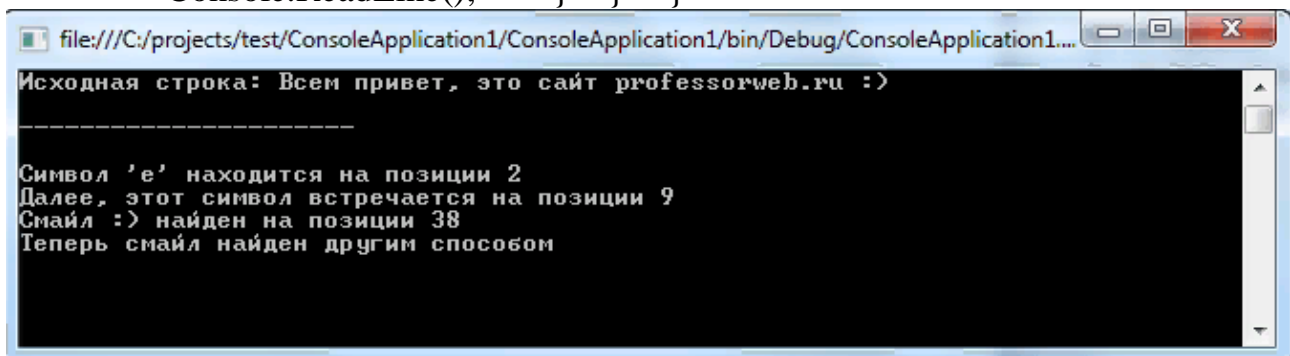
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1    {
    class Program    {
        static void Main(string[] args)    {
            string s = "Всем привет, это сайт professorweb.ru :)";
            char ch = 'e';
            string smile = ":";
            Console.WriteLine("Исходная строка: {0}\n\n-----\n",s);
            // Первая перегрузка
            if (s.IndexOf(ch) != -1)
                Console.WriteLine("Символ '{0}' находится на позиции {1}",ch,s.IndexOf(ch));

```

```

// Вторая перегрузка
if (s.IndexOf(ch, s.IndexOf(ch)+1) != -1)
    Console.WriteLine("Далее, этот символ встречается на позиции {0}",
s.IndexOf(ch, s.IndexOf(ch) + 1));
// Третья перегрузка
if (s.IndexOf(smile, 0, s.Length) != -1)
    Console.WriteLine("Смайл {0} найден на позиции {1}", smile,
s.IndexOf(smile, 0, s.Length));
// Четвертая перегрузка
if (s.IndexOf(smile, StringComparison.Ordinal) != -1)
    Console.WriteLine("Теперь смайл найден другим способом");
    Console.ReadLine();    }    }    }

```



В данном примере используется только часть доступных перегрузок метода `IndexOf`, если бы C# не поддерживал перегрузки, то пришлось бы присваивать каждому методу свое имя, что конечно же очень неудобно. В данном случае метод `IndexOf` реализует несколько перегрузок, для поиска символов и подстрок в исходной строке.

Содержание отчета о выполненной работе:

1. Название и цель работы.
2. Написание программ в Visual Studio.
3. Сохранить созданные проекты и показать преподавателю.

ЛАБОРАТОРНАЯ РАБОТА № 2

Тема: Определение операций в классе

Цель работы: формирование целостного представления о сущности и назначении классов, основных операциях классов, овладение практическими навыками работы с классами в C#.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

В языке C# имеется готовый набор лексем, используемых для выполнения базовых операций над встроенными типами. Например, известно, что операция + может применяться к двум целым, чтобы дать их сумму. Но может ли одна и та же операция + может применяться к большинству встроенных типов данных, например, для строк:

```
string s1 = "Иванов";  
string s2 = "Сергей";  
string s3 = s1 + s2;      // s3 теперь содержит "Иванов Сергей"
```

По сути, функциональность операции + уникальным образом базируются на представленных типах данных (строках). Когда операция + применяется к числовым типам, мы получаем арифметическую сумму операндов. Однако когда та же операция применяется к строковым типам, получается конкатенация строк.

Операция — конструкция языка, аналогичная по записи математическим операциям, то есть специальный способ записи некоторых действий. Путаница связана еще и с тем, что в C и C++ присваивание и инкремент/декремент являются и операторами, и операциями. Чаще всего, операция является частью оператора. Поэтому далее будем употреблять термин «перегрузка операции». Язык C# предоставляет возможность строить специальные классы, которые также уникально реагируют на один и тот же набор базовых лексем (вроде операции +). Необходимо знать, что абсолютно каждую встроенную операцию C# перегружать нельзя.

Порядок выполнения работы:

Задание 1. Написать программу бинарной операции

1. Запустить Visual Studio
2. Создать консольное приложение
3. Ввести код программы:

```
using System;  
namespace ПерегрузкаОпераций {  
    class Vector {  
        // Координаты точки в трехмерном пространстве  
        public int x, y, z;  
        // конструктор  
        public Vector(int x = 0, int y = 0, int z = 0) {  
            this.x = x;  
            this.y = y;  
            this.z = z;  
        }  
    }  
}
```

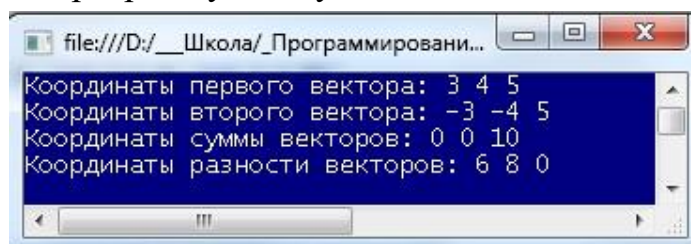


```

// Перегружаем бинарную операцию + (сложение векторов)
public static Vector operator + (Vector v1, Vector v2)      {
    Vector v = new Vector();
    v.x = v1.x + v2.x;
    v.y = v1.y + v2.y;
    v.z = v1.z + v2.z;
    return v;      }
// Перегружаем бинарную операцию - (разность векторов)
public static Vector operator - (Vector v1, Vector v2)      {
    Vector v = new Vector();
    v.x = v1.x - v2.x;
    v.y = v1.y - v2.y;
    v.z = v1.z - v2.z;
    return v;      }
public void printV(string s)      {
    Console.WriteLine(s + x + " " + y + " " + z);      } }
class Program  {
    static void Main(string[] args)      {
        Vector V1 = new Vector(3, 4, 5);
        Vector V2 = new Vector(-3, -4, 5);
        V1.printV("Координаты первого вектора: ");
        V2.printV("Координаты второго вектора: ");
        Vector V3 = V1 + V2;  // ПЕРЕГРУЗКА!
        V3.printV("Координаты суммы векторов: ");
        V3 = V1 - V2;        // ПЕРЕГРУЗКА!
        V3.printV("Координаты разности векторов: ");
        Console.ReadLine();      } } }

```

4.Скомпилировать программу и запустить на выполнение



Результат: Здесь выполняются операции сложения и разности двух векторов (выделено красным цветом).

Задание 2. Создать класс-массив, элементы которого должны находиться в диапазоне [0,100]. Кроме того, при доступе к элементу проверяется, не вышел ли индекс за допустимые границы

- 1.Запустить Visual Studio
- 2.Создать консольное приложение

3.Ввести код программы:

```

using System;
namespace ConsoleApplication1      {

```

```

class SafeArray
{ public SafeArray( int size ) // конструктор класса
{      a = new int [size];
length = size;      }
public int Length // свойство - размерность
{      get { return length; }      }
public int this[int i] //индексатор
{      get      {
if ( i >= 0 && i < length ) return a [ i ];
else { error = true; return 0; }      }
set      {
if ( i >= 0 && i < length &&
value >= 0 && value <= 100 ) a[i] = value;
else error = true      }      }
public bool error = false; // открытый признак ошибки
int[] a; // закрытый массив
int length; } // закрытая размерность
class Class1      {
static void MainO      {
int n = 100;
SafeArray sa = new SafeArray( n ); // создание объекта
for ( int i = 0; i < n; ++i ) {
sa[i] = i * 2; // 1 использование индексатора
Console.Write( sa[i] ); } // 2 использование индексатора
if ( sa.error ) Console.Write( "Были ошибки!" ); }      }      }
4.Скомпилировать программу и запустить на выполнение

```

Задание 3. Определение операции сложения для класса SafeArray из задания 2.

Содержание отчета о выполненной работе:

- 1.Название и цель работы.
- 2.Написание программ в Visual Studio.
- 3.Сохранить созданные проекты и показать преподавателю.

ЛАБОРАТОРНАЯ РАБОТА № 3

Тема: Создание наследованных классов

Цель работы: научиться создавать иерархии классов, реализовать наследование на примере классов.

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Порядок выполнения работы:

Задание 1. Создать базовый класс Student, который будет содержать информацию о студенте (фамилия, курс обучения, номер зачетной книги). С помощью механизма наследования реализовать класс Aspirant (аспирант – студент, который готовится к защите кандидатской диссертации). Класс Aspirant есть производным от класса Student. В классах Student и Aspirant необходимо реализовать следующие элементы:

- конструкторы классов с соответствующим количеством параметров. В классе Aspirant для доступа к методам класса Student нужно использовать ключевое слово base;
- свойства get/set для доступа к полям класса;
- метод Print(), который выводит информацию о содержимом полей класса.

1. Запустить Visual Studio

2. Запустить Console Application.

3. Вести текст программы.

```
using static System.Console;
```

```
namespace ConsoleApp1 {
```

```
// Базовый класс Student, содержит информацию о студенте
```

```
public class Student {
```

```
// 1. Поля класса - объявленные как protected - видимы в производных классах,
```

```
// и невидимы из экземпляра класса
```

```
protected string name; // Фамилия и имя студента
```

```
protected int course; // Курс обучения
```

```
protected string gradeBook; // Номер зачетной книги
```

```
// 2. Конструктор класса с 3 параметрами
```

```
public Student(string Name, int course, string gradeBook) {
```

```
    this.Name = Name;
```

```
    this.course = course;
```

```
    this.gradeBook = gradeBook; }
```

```
// 3. Свойства доступа к полям класса
```

```
public string Name {
```

```
    get { return name; }
```

```
    set { name = value; } }
```

```
public int Course {
```

```
    get { return course; }
```

```
    set { course = value; } }
```

```
public string GradeBook {
```

```
    get { return gradeBook; }
```

```

    set { gradeBook = value; } }
// 4. Метод Print() - вывести значения полей на экран
public void Print() {
    WriteLine("The values of fields are:");
    WriteLine($"Name = {name}");
    WriteLine($"Course = {course}");
    WriteLine($"GradeBook = {gradeBook}"); } }
// Класс Aspirant - наследует возможности класса Student
public class Aspirant: Student {
    // 1. Внутреннее поле класса
    protected string topic; // Тема кандидатской диссертации
    // 2. Конструктор класса Aspirant - с помощью ключевого слова base
    // обращается
    // к конструктору базового класса Student
    public Aspirant(string name, int course, string gradeBook, string topic) :
        base(name, course, gradeBook) {
        // Можно изменять protected-члены базового класса
        base.name = name; // доступ к полю name класса Student с помощью base
        this.course = course; // доступ к полю course класса Student с помощью this
        this.gradeBook = gradeBook;
        this.topic = topic; } // инициализация внутреннего поля класса Aspirant
    // 3. Свойство для доступа к полю topic
    public string Topic {
        get { return topic; }
        set { topic = value; } }
    // 4. Метод Print() - печать полей класса Aspirant
    // Имя данного метода перекрывает имя метода Student.Print(),
    // поэтому перед именем метода указывается new
    public new void Print() // new - переопределение метода базового класса
    {
        base.Print(); // вызвать метод Print() базового класса
        WriteLine($"Topic = {topic}"); } }
class Program {
    static void Main(string[] args) {
        // Демонстрация работы с классами Student и Aspirant
        // 1. Объявить экземпляр класса Student
        Student st1 = new Student("Ivanov I.I.", 2, "0519");
        // 2. Вывести поля класса Student
        WriteLine("The instance of st1:");
        st1.Print();
        // 3. Объявить экземпляр класса Aspirant
        // При объявлении используется свойство get экземпляра st1
        Aspirant asp1 = new Aspirant(st1.Name, st1.Course, st1.GradeBook, "Hello
world!");
        // 4. Вызвать метод Print() экземпляра asp1
        WriteLine("-----");

```

```
WriteLine("The instance of asp1:");  
asp1.Print(); } } }
```

4.Скомпилировать и запустить программу:

The instance of st1:

The values of fields are:

Name = Ivanov I.I.

Course = 2

GradeBook = 0519

The instance of asp1:

The values of fields are:

Name = Ivanov I.I.

Course = 2

GradeBook = 0519

Topic = Hello world!

Задание 2. Задан класс Book, который описывает книгу. Класс содержит следующие элементы:

- название книги;
- фамилия и имя автора;
- стоимость книги.

В классе Book нужно реализовать следующие методы:

- конструктор с 3 параметрами;
- свойства get/set для доступа к полям класса;
- метод Print(), который выводит информацию о книге.

Разработать класс BookGenre, который наследует возможности класса Book и добавляет поле жанра (genre). В классе BookGenre реализовать следующие элементы:

- конструктор с 4 параметрами – реализует вызов конструктора базового класса;
- свойство get/set доступа к внутреннему полю класса;
- метод Print(), который обращается к методу Print() базового класса Book для вывода информации о всех полях класса.

Разработать класс BookGenrePubl, который унаследован от класса BookGenre. Данный класс добавляет поле, которое содержит информацию об издателе. В классе BookGenrePubl реализовать следующие элементы:

- конструктор с 5 параметрами;
- свойство get/set для доступа к внутреннему полю класса;
- метод Print(), который вызывает одноименный метод базового класса и выводит на экран информацию об издателе.

Класс BookGenrePubl сделать таким, что не может быть унаследован.

1.Запустить Visual Studio

2.Запустить Console Application.

3.Вести текст программы.

```

using static System.Console;
namespace ConsoleApp1 {
    // Класс Book - базовый класс
    class Book {
        // 1. Внутренние поля класса
        string title; // название (заголовок) книги
        string author; // фамилия и имя автора
        double price; // стоимость книги
        // 2. Конструктор с 3 параметрами
        public Book(string _title, string _author, double _price) {
            title = _title;
            author = _author;
            // корректировка стоимости книги
            if (price < 0) price = 0.0;
            else price = _price; }
        // 3. Свойства типа get/set для доступа к полям класса
        public string Title {
            get { return title; }
            set { title = value; } }
        public string Author {
            get { return author; }
            set { author = value; } }
        public double Price {
            get { return price; }
            set {
                if (price < 0) price = 0.0;
                else price = value; } }
        // 4. Метод Print() - вывести поля класса
        public void Print() {
            WriteLine("title = {0}, author = {1}, price = {2:f2}", title, author, price); } }
    // Класс, который наследует класс Book - добавляет жанр к книге.
    class BookGenre: Book {
        // 1. Внутреннее поле - жанр, к которому относится книга
        string genre;
        // 2. Конструктор с 4 параметрами.
        // Вызывает конструктор базового класса с помощью base(...).
        public BookGenre(string _title, string _author, double _price, string _genre) :
            base(_title, _author, _price) {
            genre = _genre; }
        // 3. Свойство доступа к полю genre
        public string Genre {
            get { return genre; }
            set { genre = value; } }
        // 4. Метод Print() - вызывает метод базового класса
        // В заголовке метода используется ключевое слово new. Это

```

```

// есть рекомендация компилятора - подчеркнуть, что данный метод
// прячет одноименный метод базового класса.
public new void Print() {
    base.Print(); // вызвать метод Print() базового класса Book
    WriteLine("genre = {0}", genre); } }
// Класс, который наследует класс BookGenre - добавляет к иерархии классов
// поле издателя.
// Перед классом используется ключевое слово sealed - это означает,
// что данный класс не может быть унаследован другими классами.
sealed class BookGenrePubl : BookGenre {
    // 1. Внутреннее поле - информация об издателе
    private string publisher;
    // 2. Конструктор с 5 параметрами - Вызывает конструктор базового
    // класса BookGenre() с помощью ключевого слова base.
    public BookGenrePubl(string _title, string _author, double _price,
        string _genre, string _publisher) : base(_title, _author, _price, _genre) {
        publisher = _publisher; }
    // 3. Свойство доступа к полю publisher
    public string Publisher {
        get { return publisher; }
        set { publisher = value; } }
    // 4. Метод Print() - вызывает метод базового класса.
    // В объявлении метода рекомендуется использовать
    // ключевое слово new, так как данный метод "прячет" метод базового класса.
    public new void Print() {
        base.Print();
        WriteLine("publisher = {0}", publisher); } }
class Program {
    static void Main(string[] args) {
        // 1. Объявить экземпляр класса Book
        Book b1 = new Book("Title - 01", "Author - 01", 122.25);
        // 2. Вывести значения полей класса экземпляра b1
        b1.Print();
        // 3. Объявить экземпляр класса BookGenre
        BookGenre bg1 = new BookGenre("Title - BookGenre", "Author - BookGenre",
200.33, "Story");
        // 4. Вывести значения полей экземпляра bg1
        WriteLine("-----");
        bg1.Print();
        // 5. Объявить экземпляр класса BookGenrePubl
        BookGenrePubl bp1 = new BookGenrePubl("Title - BookGenrePubl", "Aurhor -
BookGenrePubl",
        300.55, "Story", "Pupkin Inc.");
        // 6. Вывести значения полей экземпляра bp1
        WriteLine("-----");
    }
}

```

```
bp1.Print(); } } }
```

4.Скомпилировать и запустить программу.

- 01, price = 122.25

title = Title - BookGenre, author = Author - BookGenre, price = 200.33
genre = Story

title = Title - BookGenrePubl, author = Aurhor - BookGenrePubl, price = 300.55
genre = Story
publisher = Pupkin Inc.

Содержание отчета о выполненной работе:

- 1.Название и цель работы.
- 2.Написание программ в Visual Studio.
- 3.Сохранить созданные проекты и показать преподавателю.

ЛАБОРАТОРНАЯ РАБОТА № 4

Тема: Создание хранимых процедур

Цель работы: научиться создавать хранимые процедуры для работы с базой данных в Visual Studio

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Справочный материал:

Хранимой процедурой называется одна или несколько SQL-конструкций, которые записаны в базе данных. Задача администрирования базы данных включает в себя в первую очередь распределение уровней доступа к БД. Разрешение выполнения обычных SQL-запросов большому числу пользователей может стать причиной неисправностей из-за неверного запроса или их группы. Чтобы этого избежать, разработчики базы данных могут создать ряд хранимых процедур для работы с данными и полностью запретить доступ для обычных запросов.

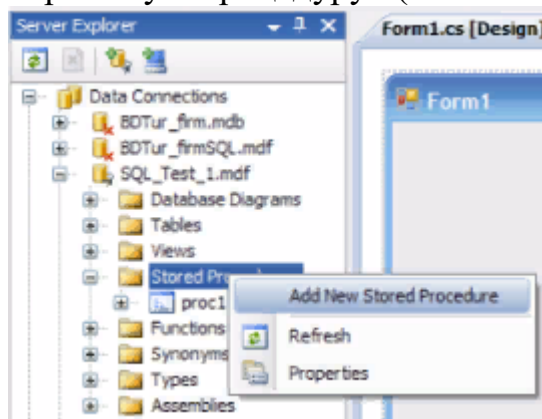
Такой подход при прочих равных условиях обеспечивает большую стабильность и надежность работы. Это одна из главных причин создания собственных хранимых процедур. Другие причины – быстрое выполнение, разбиение больших задач на малые модули, уменьшение нагрузки на сеть – значительно

облегчают процесс разработки и обслуживания архитектуры «клиент – сервер».

Содержание работы:

Задание 1. Создание хранимых процедур к базе данных из Практической работы № 45

Запустим Visual Studio (даже нет необходимости создавать какой-либо проект), перейдем на вкладку «Обозреватель баз данных» («Server Explorer»), раскрываем подключение к базе данных, затем на узле «Хранимые процедуры» («Stored Procedures») щелкаем правой кнопкой и выбираем пункт «Добавить новую хранимую процедуру» («New Stored Procedure»)



Появляется шаблон структуры, сгенерированный мастером:

```
CREATE PROCEDURE dbo.StoredProcedure1
```

```
/*(@parameter1 int = 5,
```

```
@parameter2 datatype OUTPUT)*/
```

```
AS
```

```
/* SET NOCOUNT ON */
```

RETURN

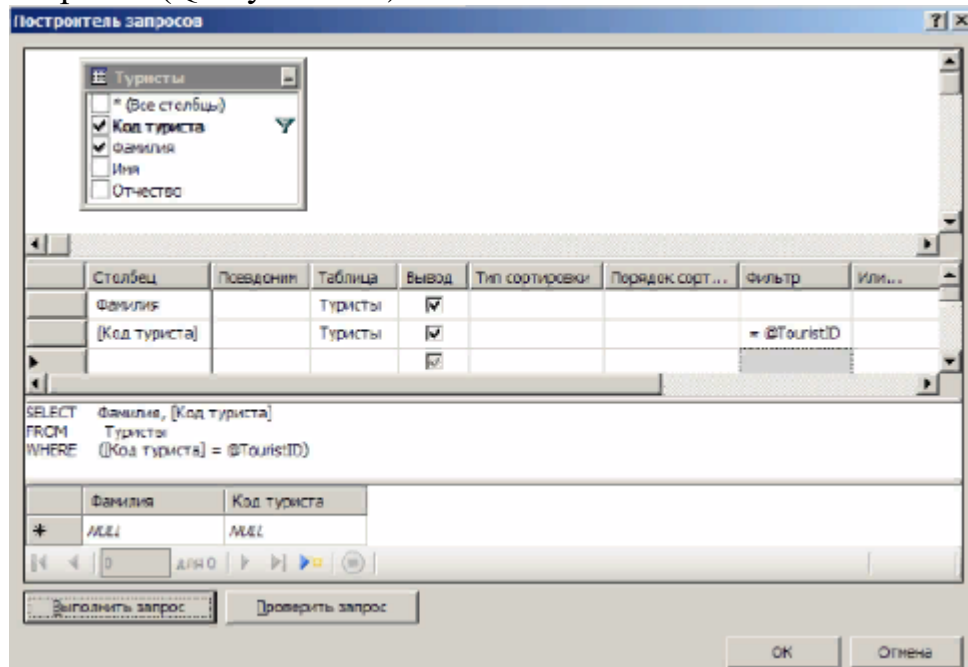
Для того чтобы приступить к редактированию, достаточно убрать знаки комментариев «/*», «*/».

Команда NOCOUNT со значением ON отключает выдачу сообщений о количестве строк таблицы, получающейся в качестве запроса. Необходимость данного оператора заключается в том, что при использовании более чем одного оператора (SELECT, INSERT, UPDATE или DELETE) в начале запроса надо поставить команду SET NOCOUNT ON, а перед последним оператором SELECT – команду SET NOCOUNT OFF.

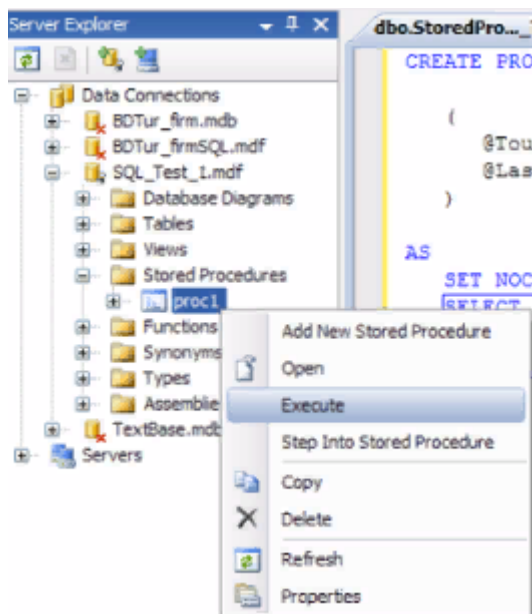
Например, хранимую процедуру proc_po1 (см. таблицу 15) можно переписать следующим образом:

```
CREATE PROCEDURE dbo.proc_vs1(  
@TouristID int,  
@LastName nvarchar(60) OUTPUT)  
AS  
SET NOCOUNT ON  
SELECT @LastName = Фамилия  
FROM Туристы WHERE ([Код туриста] = @TouristID)  
RETURN
```

После завершения редактирования SQL-конструкция будет обведена синей рамкой. Щелкнув правой кнопкой в этой области и выбрав пункт меню «Разработать блок SQL» («Design SQL Block»), можно перейти к построителю запросов (Query Builder).



Для выполнения процедуры необходимо в выпадающем меню процедуры, выбрать пункт Execute.



При попытке выполнить хранимую процедуру в окно сообщений будет выведена информация о том, что данная процедура требует значение параметра.

Для записи процедуры в базу данных достаточно ее сохранить. Происходит синхронизация с базой данных, и процедура "proc_vs1" появляется в списке. Двойной щелчок открывает ее для редактирования, причем заголовок имеет следующий вид:

ALTER PROCEDURE dbo.proc_vs1

Оператор ALTER позволяет производить действия (редактирование) с уже имеющимся объектом базы данных.

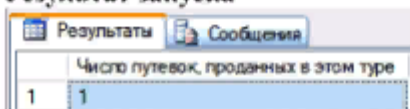
Для удаления хранимой процедуры используется оператор drop:

drop proc proc1

Здесь proc1 – название процедуры

Задание 2. Создать хранимые процедуры к БД «Туристы», используя следующие примеры

SQL-конструкция для создания	Команда для извлечения
<pre>create proc proc_po1 @TouristID int, @LastName nvarchar(60) output as select @LastName = Фамилия from Туристы where [Код туриста] = @TouristID</pre>	<pre>declare @LastName nvarchar(60) exec proc_po1 '4', @LastName output select @LastName as 'Фамилия туриста'</pre>
Описание Извлечение фамилии туриста по заданному коду	
Результат запуска 	

SQL-конструкция для создания	Команда для извлечения
<pre>create proc proc_po2 @CountCity int output as select @CountCity = count([Код туриста]) from [Информация о туристах] where Город like '%рг%'</pre>	<pre>declare @CountCity int exec proc_po2 @CountCity output select @CountCity as 'Количество туристов, прожи- вающих в городах %рг%'</pre>
Описание Подсчет количества туристов из городов, имеющих в своем названии сочетание букв «рг». Следует ожидать число три (Екатеринбург, Оренбург, Санкт-Петербург)	
SQL-конструкция для создания	Команда для извлечения
<pre>create proc proc_po5 @CodeTour int, @ChisloPutevok int output as select @ChisloPutevok = count(Путевки.[Код сезона]) from Путевки inner join Сезоны on Путевки.[Код сезона] = Сезоны.[Код сезона] inner join Туры on Туры.[Код тура] = Сезоны.[Код тура] where Сезоны.[Код тура] = @CodeTour</pre>	<pre>declare @ChisloPutevok int exec proc_po5 '1', @ChisloPutevok output select @ChisloPutevok AS 'Число путевок, проданных в этом туре'</pre>
Описание Подсчет количества путевок, проданных по заданному туру	
Результат запуска 	

Содержание отчета о выполненной работе:

- 1.Название и цель работы.
- 2.Написание хранимых процедур в Visual Studio.
- 3.Сохранить созданные проекты и показать преподавателю.

ЛАБОРАТОРНАЯ РАБОТА № 5

Тема: Создание хранимых процедур

Цель работы: научиться создавать хранимые процедуры для работы с базой данных в Visual Studio

Оборудование: ПК, программное обеспечение – Visual Studio, инструкции по выполнению работы.

Содержание работы:

Задание 1. Создать хранимые процедуры к БД по индивидуальным заданиям из Практической работы № 46.

Содержание отчета о выполненной работе:

- 1.Название и цель работы.
- 2.Написание хранимых процедур в Visual Studio.
- 3.Сохранить созданные проекты и показать преподавателю.

Информационное обеспечение обучения

Печатные издания:

Основные учебные издания:

1. Зыков, С. В. Введение в теорию программирования. Объектно-ориентированный подход: учебное пособие для СПО / С. В. Зыков. — Саратов: Профобразование, 2021. — 187 с. — ISBN 978-5-4488-0995-8. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/102188>
2. Кариев, Ч. А. Разработка Windows-приложений на основе Visual C#: учебное пособие / Ч. А. Кариев. — 3-е изд. — Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. — 978 с. — ISBN 978-5-4497-0909-7. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: <https://www.iprbookshop.ru/102057.html>
3. Лебедева, Т. Н. Технология программирования: учебное пособие для СПО / Т. Н. Лебедева, С. С. Юнусова. — Саратов: Профобразование, 2019. — 140 с. — ISBN 978-5-4488-0351-2. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/86081>

Дополнительные учебные издания:

4. Биллиг, В. А. Основы программирования на C#: учебное пособие / В. А. Биллиг. — 3-е изд. — Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. — 573 с. — ISBN 978-5-4497-0893-9. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/102033>
5. Зубкова, Т. М. Технология разработки программного обеспечения: учебное пособие для СПО / Т. М. Зубкова. — Саратов: Профобразование, 2019. — 468 с. — ISBN 978-5-4488-0354-3. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/86208>

Электронные издания (электронные ресурсы)

6. Учебники по программированию <http://programm.ws/index.php>

Электронно-библиотечная система:

7. ЭБС «IPRbooks», ООО «Ай Пи Ар Медиа»
8. ЭБС «Znanium»
9. ЭБС «PROФобразование»
10. ЭБС «Book.ru»