**Лабораторная работа 2. Основы языка C#**

Для изучения основ языка программирования вы будете использовать созданный ранее контроллер, метод действия которого будет возвращать результат в браузер.

1. Добавьте в папку **Models** класс **StudyCsharp**, который будет реализовывать все последующие действия по изучению языка.

***Упражнение 1. Создание и использование размерных типов данных***

В этом упражнении вы создадите перечисление для представления различных типов банковских счетов (например, Checking и Deposit). Вы создадите структуру, которую можно использовать для моделирования банковских счетов: переменные для хранения номеров счетов (тип данных int), балансов счетов (тип данных decimal) и типов счетов (перечисление).

**Создание перечисления**

Перечисляемый тип представляет собой тип значений, содержащий конечное число именованных констант. Ниже объявляется и используется перечисляемый тип AccountType, содержащий два постоянных значения Checking и Deposit.

1. Откройте файл StudyCsharp.cs и перед описанием класса добавьте перечисление AccountType со значениями Checking и Deposit:

namespace WebMVCR1.Models

{

public enum AccountType

{

Checking,

Deposit

}

public class StudyCsharp

{

}

}

2. В котроллере HomeController объявите новый метод ExeEnum() и объявите две переменные типа AccountType:

public string ExeEnum()

{

AccountType goldAccount;

AccountType platinumAccount;

3. Далее присвойте первой переменной значение Checking, а второй – Deposit:

goldAccount = AccountType.Checking;

platinumAccount = AccountType.Deposit;

4. Объявите две строковые переменные, с помощью метода Format() сформируйте строки результата и верните их в браузер:

string res1 = String.Format("Тип банковского счета {0}", goldAccount);

string res2 = String.Format("Тип банковского счета {0}", platinumAccount);

string res = res1 + "<p>"+ res2;

return res;

}

**Тестирование перечисления**

1. В методе действия Index() объявите строковую переменную res и присвойте ей вызов метода ExeEnum(), замените в выражении return переменную Greeting на новую переменную res:

string res = ExeEnum();

return res;

2. Запустите проект, браузер должен отобразить результат метода действия Index() – две строки с указанием типа банковских счетов.

// Результат:

Тип банковского счета Checking

Тип банковского счета Deposit

Метод действия ExeEnum() можно было также вызвать указав в строке браузера

localhost:port/Home/ExeEnum

// localhost:44360/Home/ExeEnum – моя строка

Почему-то не получилось

**Создание и использование структуры**

Структуры представляют собой структуры данных, содержащие члены-данные и члены-функции (методы структуры). Структуры имеют тип значений и не требуют выделения памяти в куче. Переменная типа структуры непосредственно хранит данные структуры.

1. В файле StudyCsharp.cs после перечисления добавьте **public** (общедоступную) структуру **BankAccount**, содержащую следующие поля:

|  |  |  |
| --- | --- | --- |
| **Модификатор доступа** | **Тип** | **Имя переменной** |
| public | long | *accNo* |
| public | decimal | *accBal* |
| public | AccountType | *accType* |

В итоге структура должна выглядеть следующим образом:

public struct BankAccount

{

public long accNo;

public decimal accBal;

public AccountType accType;

}

2. В котроллере HomeController объявите новый метод ExeStruct() и в нем объявите переменную типа **BankAccount**:

public string ExeStruct()

{

BankAccount goldBankAccount;

3. Далее присвойте значения полям *accNo, accBal* и *accType* переменной goldAccount*:*

goldBankAccount.accType = AccountType.Checking;

goldBankAccount.accBal = (decimal)3200.00;

goldBankAccount.accNo = 123;

4. Сформируйте строку информации о счете и верните ее в браузер:

string res = String.Format("Номер счета {0}, баланс {1}, тип {2}", goldBankAccount.accNo, goldBankAccount.accBal, goldBankAccount.accType);

return res;

**Тестирование структуры**

1. В методе действия Index()закомментируйте строковую переменную res, которая вызывала метод перечисления, объявите новую переменную res и присвойте ей вызов метода ExeStruct():

string res = ExeStruct();

return res;

2. Запустите проект, браузер должен отобразить результат метода действия Index() – строку с указанием информации о банковском счете.

// Результат:

Номер счета 123, баланс 3200, тип Checking

**Переопределение метода ToString()**

Любой тип языка С# является производным типом от типа **Object**, в котором определен метод ToString(), возвращающий строковое представление объекта. Для производных типов данный метод рекомендуется переопределить, указав правило представления объекта в виде строки.

1. В структуру **BankAccount** добавьте переопределение (ключевое слово override) метода ToString(), в котором укажите способ преобразования:

public struct BankAccount

{

public long accNo;

public decimal accBal;

public AccountType accType;

**public override string ToString()**

**{**

**string res = String.Format("Номер счета {0}, баланс {1}, тип {2}", accNo, accBal, accType);**

**return res;**

**}**

}

2. В методе ExeStruct() котроллера HomeController замените выражение вычисления строки res на следующее выражение, использующее преобразование объекта в строку:

string res = String.Format("Информация о банковском счете: {0}", goldBankAccount);

3. Запустите проект, браузер должен отобразить результат метода действия Index() – строку с указанием информации о банковском счете согласно логике метода ToString().

// Результат:

Информация о банковском счете: Номер счета 123, баланс 3200, тип Checking

**Использование C# версии 6.0 для формирование строки**

Начиная с версии 6.0 для формирования строки допускается функция интерполяции строк, позволяющая внедрить выражения в строку формата – строка начинается с символа $, а затем в кавычках указываются данные, например:

string str = $"Номер счета {accNo}, баланс {accBal}, тип {accType}";

Примените новый синтаксис к указанному выше коду и протестируйте его работу. В дальнейшем вы можете использовать любой вариант синтаксиса на ваше усмотрение.

// Полезное написание кода. Одно и то же можно сделать по разному

Результат:

Информация о банковском счете: Номер счета 123, баланс 3200, тип Checking

***Упражнение 2. Создание и использование методов***

В этом упражнении вы реализуете различные алгоритмы с помощью методов. Обратите внимание на реализуемые в этих методах управляющие конструкции (ветвление, циклы).

1. Откройте файл StudyCsharp.cs.

**Реализация выбора**

Оператор **if** выбирает оператор для выполнения на основании значения логического выражения.

2. В классе StudyCsharp объявите статический метод, принимающий целочисленный параметр и рассчитывающий в зависимости от значения этого параметра статус разработчика:

public static string SetStatus(int age)

{

string status = "junior developer";

if ((age > 2) && (age < 7)) status = "middle developer";

else if ((age >= 7) && (age < 15)) status = "senior developer";

else if ((age >= 15)) status = "sensei";

return status;

}

3. В методе Index() котроллера HomeController укажите выражение вычисления строки res, использующее вызов метода SetStatus() (ранее вычисляемую переменную res закомментируйте):

string res = StudyCsharp.SetStatus(3);

4. Запустите проект, браузер должен отобразить результат метода действия Index() – строку с указанием информации о вычисленном статусе.

// Результат:

middle developer

Оператор **switch** выбирает для выполнения список операторов, метка которого соответствует значению **switch**-выражения.

1. В классе StudyCsharp объявите статический метод, принимающий строковый параметр (статус) и определяющий в зависимости от значения этого параметра возможные действия:

public static string ExeSwitch(string status)

{

string res;

switch (status)

{

case "junior developer":

res = "Набирайся знаний"; break;

case "middle developer":

res = "Набирайся опыта"; break;

case "senior developer":

res = "Руководи другими"; break;

case "sensei":

res = "Учи других"; break;

default:

res = "Не знаю, что делать";

break;

}

return res;

}

2. В методе Index() котроллера HomeController укажите выражение вычисления строки res, использующее вызов метода ExeSwitch(), принимающий в качестве параметра метод SetStatus() (ранее вычисляемую переменную res закомментируйте):

string res = StudyCsharp.ExeSwitch(Stud yCsharp.SetStatus(3));

3. Запустите проект, браузер должен отобразить результат метода действия Index() – строку с указанием действий в зависимости от статуса.

// Результат:

Набирайся опыта

**Реализация цикла – расчет значения функции на интервале**

4. В классе StudyCsharp объявите статический метод, принимающий два вещественных параметра (границы интервала) и возвращающий значения функции в виде строки:

public static string GetFunction(double x1, double x2)

{

StringBuilder str = new StringBuilder();

double x = x1;

do

{

str.AppendFormat("x = {0:0.##} : y = {1:0.##}; <br>", x, Math.Pow(x,3));

x = x + 0.5;

}

while (x <= x2);

return str.ToString(); ;

}

Обратите внимание на использование для построения результата объект класса **StringBuilder**, применяемого для изменяемых строк и использующий метод AppendFormat().

5. В методе Index() котроллера HomeController укажите выражение вычисления строки res, использующее вызов метода GetFunction() (ранее вычисляемую переменную res закомментируйте):

string res = StudyCsharp.GetFunction(0, 9);

6. Запустите проект, браузер должен отобразить результат метода действия Index() – строку с расчетными значениями.

// Результат

x = 0 : y = 0;  
x = 0,5 : y = 0,13;  
x = 1 : y = 1;  
x = 1,5 : y = 3,38;  
x = 2 : y = 8;  
x = 2,5 : y = 15,63;  
x = 3 : y = 27;  
x = 3,5 : y = 42,88;  
x = 4 : y = 64;  
x = 4,5 : y = 91,13;  
x = 5 : y = 125;  
x = 5,5 : y = 166,38;  
x = 6 : y = 216;  
x = 6,5 : y = 274,63;  
x = 7 : y = 343;  
x = 7,5 : y = 421,88;  
x = 8 : y = 512;  
x = 8,5 : y = 614,13;  
x = 9 : y = 729;

**Использование возвращаемых параметров в методах**

**Реализация класса – факториал**

В этом задании вы создадите метод **Factorial()**, принимающий целочисленную переменную и рассчитывающий ее факториал по итерационному алгоритму.

1. В классе StudyCsharp объявите статический метод:

a. принимающий два параметра (первый параметр типа **int** передается по значению (это число, для которого рассчитывается факториал), второй параметр типа **out int** используется для возвращения результата)

b. возвращающий значения типа **bool**, отражающее успешность выполнения метода, так как может произойти переполнение и выброс исключения:

public static bool Factorial(int n, out int answer)

{

}

2. Внутри метода напишите код расчета факториала для передаваемого на вход значения:

a. Объявите три переменных: целочисленные k (будет использоваться в цикле в качестве счетчика) и f (будет использоваться внутри цикла, присвойте начальное значение 1), а также булевого типа ok (будет использоваться для отслеживания ошибок):

int k;

int f = 1;

bool ok = true;

b. Создайте цикл **for**. Начальное значение k = 2, итерации продолжаются до тех пор, пока не будет достигнуто значение параметра n. На каждом шаге значение k увеличивается на единицу. В теле цикла f умножается на k и сохраняется результат в f. Значение факториала растет достаточно быстро, поэтому реализуйте проверку на арифметическое переполнение в блоке **checked**:

checked

{

for (k = 2; k <= n; ++k)

{

f = f \* k;

}

}

c. Реализуйте перехват возможных исключений, все исключения обрабатываются одинаково: обнуляется результат и переменной *ok* присваивается **false**:

try

{

checked

{

for (k = 2; k <= n; ++k)

{

f = f \* k;

}

}

}

catch (Exception)

{

f = 0;

ok = false;

}

d. Итоговое значение переменной f присвойте возвращаемому параметру answer:

answer = f;

return ok;

Если метод отработал успешно, он возвращает значение **true**, если произошло арифметическое переполнение (выброс исключения), то возвращается значение **false**.

**Тестирование метода расчета факториала**

1. В котроллере HomeController объявите новый метод, принимающий целочисленную переменную x для расчета факториала:

public string ExeFactorial(int x)

{

2. В этом методе объявите две переменные:

a. переменную f типа **int** для хранения факториала числа,

b. переменную ok типа **bool** и присвойте ей результат вызова метода **Factorial(),** передав число x и переменную f в качестве параметров:

int f;

bool ok = StudyCsharp.Factorial(x, out f);

3. Реализуйте возврат из метода значения переменных x и f в случае, если переменная ok принимает значение **true**. В противном случае передайте сообщение об ошибке:

if (ok)

return String.Format("Факториал числа {0} равен {1} ", x, f);

else

return "Невозможно вычислить факториал";

4. В методе Index() котроллера HomeController укажите выражение вычисления строки res, использующее вызов метода для расчета факториала числа 5 – ExeFactorial(5) (переменную res ранее вычисляемую функцию GetFunction закомментируйте):

string res = ExeFactorial(5);

5. Запустите проект, браузер должен отобразить результат метода действия Index() – строку с указанием информации о вычисленном значении факториала.

// Результат

Факториал числа 5 равен 14400

***Упражнение 3. Реализация класса***

**Реализация класса – треугольник**

В этом упражнении вы создадите класс – треугольник, описываемый сторонами и имеющий периметр и площадь.

1. В пространстве имен namespace WebMVCR1.Models в любом месте (например, после класса StudyCsharp) объявите класс Triangle:

public class Triangle

{

}

2. В классе Triangle объявите автоматические свойства, определяющие стороны треугольника:

public class Triangle

{

public double Sta { get; set; }

public double Stb { get; set; }

public double Stc { get; set; }

3. Далее реализуйте еще три свойства – имя, периметр и площадь. Все эти свойства должны быть определены только для чтения поскольку зависят от значений сторон:

public string Name

{

get { return String.Format("\"Треугольник со сторонами {0}, {1} и {2}\"", Sta, Stb, Stc); }

}

public double Perimeter

{

get

{

double p = Sta + Stb + Stc;

return p;

}

}

public double Area

{

get

{

double sq = Math.Sqrt(Perimeter / 2 \* (Perimeter / 2 - Sta) \* (Perimeter / 2 - Stb) \* (Perimeter / 2 - Stc));

return sq;

}

}

**Использование C# версии 6.0 для формирование свойств только для чтения**

Начиная с версии 6.0 сократить синтаксис, написав член класса, представляющий выражение с применением оператора “=>”. Например, для свойства периметра и имени новый синтаксис будет выглядит следующим образом:

public double Perimeter => Math.Round(Sta + Stb + Stc);

public string Name => $"\"Треугольник со сторонами {Sta},{Stb} и {Stc}\"";

Примените новый синтаксис к указанному выше коду и протестируйте его работу. В дальнейшем вы можете использовать любой вариант синтаксиса на ваше усмотрение (это работает для методов и свойств, доступных только для чтения).

4. Реализуйте в классе конструктор с тремя параметрами – сторонами треугольника:

public Triangle(double a, double b, double c)

{

Sta = a;

Stb = b;

Stc = c;

}

**Тестирование класса**

1. В котроллере HomeController объявите новый метод ExeTriangle() и в нем создайте объект – треугольник с заданными сторонами – 3, 5, 6:

public string ExeTriangle()

{

Triangle tr1 = new Triangle(3, 5, 6);

2. Далее в этом методе выведите на экран значения имени объекта и его площади:

string sq1 = String.Format("Площадь фигуры {0} равна: {1:0.##}", tr1.Name, tr1.Area);

return sq1;

}

3. В методе действия Index() закомментируйте строковую переменную res, которая вызывала предыдущий метод, объявите новую переменную res и присвойте ей вызов метода ExeTriangle():

string res = ExeTriangle();

4. Запустите проект, браузер должен отобразить результат метода действия Index() – строку с указанием информации о треугольнике.

// Результат:

Площадь фигуры "Треугольник со сторонами 3, 5 и 6" равна: 7,48

**Реализация класса – Окружность**

1. В пространстве имен namespace WebMVCR1.Models в любом месте (например, после класса Triangle) объявите класс Circle, реализующий окружность, определяемую радиусом.

2. В классе объявите автоматическое свойство, определяющее радиус.

3. Далее реализуйте еще три свойства – имя, длина окружности и площадь. Все эти свойства должны быть определены только для чтения поскольку зависят от значения радиуса.

4. В классе реализуйте конструктор с одним параметром – радиусом окружности.

5. Реализация класса в итоге может быть следующей:

public class Circle

{

public double St{ get; set; }

public string Name

{

get { return String.Format("\"Окружность с радиусом {0}\"", St; }

}

public Circle(double a)

{

St = a;

}

public double Dlina

{

get

{

double p = 2 \* Math.PI \* St;

return p;

}

}

public double Area

{

get

{

double sq = Math.PI \* St\*St;

return sq;

}

}

}

6. Постройте приложение.

**Тестирование класса**

1. В котроллере HomeController объявите новый метод ExeCircle() и в нем создайте объект – окружность с заданным радиусом – 3.

2. Далее в этом методе выведите на экран значения имени объекта и его площади.

3. Реализация метода в итоге может быть следующей:

public string ExeCircle()

{

Circle cir1 = new Circle(3);

string sq = String.Format("Площадь фигуры {0} равна: {1:0.##}", cir1.Name, cir1.Area);

return sq;

}

4. В методе действия Index() закомментируйте строковую переменную res, которая вызывала предыдущий метод, объявите новую переменную res и присвойте ей вызов метода ExeCircle():

string res = ExeCircle();

5. Запустите проект, браузер должен отобразить результат метода действия Index() – строку с указанием информации об окружности.

// Результат

Площадь фигуры "Окружность с радиусом 3" равна: 28,27

**// По хорошему, для каждой фигуры нужно сделать свой отдельный класс – лист. А я сделал в одном…**

***Упражнение 4. Наследование и реализация полиморфизма***

**Реализация класса – Фигура**

В этом упражнении вы создадите на основе классов предыдущего упражнения иерархию – наследование, создадите базовый класс, в котором объявите функциональность, наследуемую производными классами.

1. В пространстве имен namespace WebMVCR1.Models в любом месте (например, после класса Circle) объявите класс Shape, реализующий фигуру.

2. В классе объявите автоматическое свойство, определяющее сторону:

public class Shape

{

public double St { get; set; }.

3. Далее объявите свойство – имя, которое укажите виртуальным (виртуальные свойства и методы в производных классах могут быть переопределены):

public class Shape

{

public double St { get; set; }

virtual public string Name

{

get { return String.Format("\"Фигура\""); }

}

4. Внесите изменения в класс окружности:

// нужно зайти в класс (файл) окружности и менять его

a. Укажите, что класс Circle наследуется от класса Shape:

public class Circle : Shape

{

b. Удалите объявление свойства стороны – радиуса. Теперь сторона St будет наследоваться у базового класса Shape.

c. Укажите, что версия свойства Name класса окружности переопределяет базовую версию, добавив ключевое слово override:

override public string Name

{

get { return String.Format("\"Окружность с радиусом {0}\"", St); }

}

5. Внесите изменения в класс треугольника:

// нужно зайти в класс (файл) треугольника и менять его

a. Укажите, что класс Triangle наследуется от класса Shape:

public class Triangle : Shape

{

b. Удалите объявление свойства стороны – Sta. Теперь эта сторона будет наследоваться у базового класса Shape.

c. Переименуйте имя переменной Sta на St во всем коде класса.

d. Укажите, что версия свойства Name класса треугольника переопределяет базовую версию, добавив ключевое слово override:

override public string Name

{

get { return String.Format("\"Треугольник со сторонами {0}, {1} и {2}\"", St, Stb, Stc); }

}

**Тестирование полиморфизма**

1. В котроллере HomeController объявите новый метод ExePolim().

// public string ExePolim()

2. Создайте объект класса изменяемых строк:

StringBuilder str = new StringBuilder();

3. Далее в этом методе объявите массив типа Shape, но инициализируйте его объектами производных классов, например:

Shape[] sh = {

new Triangle(1,2,3),

new Circle(5),

new Triangle(5,6,8)

};

4. В цикле foreach реализуйте обращение к массиву и вызовите для каждого элемента массива свойство Name (так как оно в базовом классе объявлено виртуальным и переопределено в производных классах, то вызов будет полиморфным):

foreach (Shape item in sh)

{

str.AppendFormat("Это фигура {0}", item.Name + "<p>");

}

5. В выражении return укажите сформированную строку:

return str.ToString();

// Пока не написал return, ExePolim был подчеркнут и выдавал ошибку

6. В методе действия Index() закомментируйте строковую переменную res, которая вызывала предыдущий метод, объявите новую переменную res и присвойте ей вызов метода ExePolim():

string res = ExePolim();

7. Запустите проект, браузер должен отобразить результат метода действия Index() – три строки с указанием информации о фигурах.

// Результат:

Это фигура "Треугольник со сторонами 1, 2 и 3"

Это фигура "Окружность с радиусом 5"

Это фигура "Треугольник со сторонами 5, 6 и 8"

***Упражнение 5. Применение коллекций для группировки объектов***

В этом упражнении вы создадите коллекцию для организации объектов. В отличие от массивов, группа объектов в коллекции может динамически возрастать и сокращаться в соответствии с потребностями приложения. // Очень полезная функция

Коллекция является классом, поэтому необходимо объявить новую коллекцию перед добавлением в неё элементов.

Если коллекция содержит элементы только одного типа данных, можно использовать один из классов в пространстве имен System.Collections.Generic (типизированные классы коллекций). Такая коллекция обеспечивает безопасность типов, так что другие типы данных не могут быть в нее добавлены. При извлечении элемента из этой коллекции нет необходимости определять или преобразовывать его тип данных.

В созданной в этом упражнении коллекции можно будет сортировать объекты. Для задания свойства объекта по которому будет выполняться сортировка требуется реализовать специальный интерфейс. В библиотеке классов определено множество стандартных интерфейсов, задающих желаемую функциональность объектов. В частности, интерфейс **IComparable** задает метод сравнения объектов по принципу больше и меньше, что заставляет переопределить соответствующие операции для объектов класса.

1. Перейдите в файл StudyCsharp.cs и для класса Circle добавьте реализацию интерфейса IComparable<Circle>:

// У меня для удобства окружность в отдельном файле (классе) Circle – идём туда

public class Circle : Shape, IComparable<Circle>

{ ...

// На этом этапе подчеркивает IComparable<Circle>

Интерфейс **IComparable** и содержит единственный метод **CompareTo**, возвращающий результат сравнения двух объектов – текущего и переданного ему в качестве параметра. Реализация данного метода должна возвращать 0 – если текущий объект и параметр равны; отрицательное число, если текущий объект меньше параметра; положительное число, если текущий объект больше параметра. Этот метод является только определением, и для его использования необходимо реализовать его в определенном классе. Смысл, вкладываемый в понятия "меньше, чем", "равно" и "больше, чем", зависит от конкретной реализации.

2. Добавьте к класс Circle реализацию метода CompareTo()для сравнения объектов по площади:

public int CompareTo(Circle other)

{

if (this.Area == other.Area) return 0;

else if (this.Area > other.Area) return 1;

else return -1;

}

3. В котроллере HomeController объявите новый метод ExeCollection().

// public string ExeCollection()

4. В этом методе создайте коллекцию – список однотипных объектов – окружностей, используя класс **List<T>,** заполненный некоторым количеством окружностей:

List<Circle> cirs = new List<Circle>

{

new Circle(12),

new Circle(5),

new Circle(15),

new Circle(6)

};

5. С помощью метода **Add(T item)** добавьте в список новую окружность:

cirs.Add(new Circle(7));

6. С помощью метода **Sort()** отсортируйте коллекцию:

cirs.Sort();

7. Реализуйте вывод в браузер содержимое отсортированной коллекции:

StringBuilder str = new StringBuilder();

foreach (Shape item in cirs)

{

str.AppendFormat("Это фигура {0}", item.Name + "<p>");

}

return str.ToString();

8. В методе действия Index() закомментируйте строковую переменную res, которая вызывала предыдущий метод, объявите новую переменную res и присвойте ей вызов метода ExeCollection():

string res = ExeCollection();

9. Запустите проект, браузер должен отобразить результат метода действия Index() – строки с указанием информации об окружностях, отсортированные по площади .

// результат

Это фигура "Окружность с радиусом 5"

Это фигура "Окружность с радиусом 6"

Это фигура "Окружность с радиусом 7"

Это фигура "Окружность с радиусом 12"

Это фигура "Окружность с радиусом 15"

10. Аналогичным образом реализуйте коллекцию треугольников с возможностью сортировки по их периметру.

11. Протестируйте внесенные изменения.

Circle на Triangle

Area на Perimeter