Занятие 3, ч1. ООП на C#

Тренер: Алексей Дышлевой

Основные вопросы

- Класи та об’єкти. Практичне застосування
- Поля. Методи. Параметри методів. Аргументи ref та out, масиви params. Властивості. Індексатори.
- Управління видимістю. Статичні класи та члени класу, перевантаження методів
- Герметизовані (sealed), абстрактні та вкладені класи
- Пакування та розпакування
- Створення об’єктів. Ініціалізація полів. Ключове слово new. Конструктори. Статичні конструктори та конструктори екземпляра.
- Наслідування. Порядок виклику конструкторів та фіналізаторів. Ключові слова base та this
- Поліморфізм. Приведення типів.
- Віртуальні та абстрактні методи.
- Методи new та override. Методи sealed
- Інтерфейси. Визначення інтерфейсів. Наслідування інтерфейсів. Реалізація інтерфейсів. Явна и неявна реалізація інтерфейсів. Перевизначення реалізації інтерфейсів у похідних класах.
- Контракти

Устройство класса

- Члены-данные (поля) хранят всю информацию об объекте, формируют его состояние, характеристики
- Метод – подпрограмма в классе. Реагирует на передачу методу сообщения. Определяет поведение объекта
- Свойство – составляющая часть объекта, доступ к которой осуществляется как к члену объекта, но таковым не является (реализуют принцип инкапсуляции для членов объекта)
- Кроме того, в классе C# могут быть: конструкторы, финализаторы, операции (operator), индексаторы

- Задание: создать класс со всеми возможными членами класса

Основные Принципы ООП

- Абстракция
- Инкапсуляция
- Наследование
- Полиморфизм

Абстракция

- Объекты предоставляют неполную информацию о реальных сущностях предметной области
- Абстракция позволяет оперировать с объектом на уровне, адекватном решаемой задаче
- Высокоуровневые обращения к объекту могут обрабатываться с помощью вызова функций и методов низкого уровня
- В основе реализации этого принципа лежит использование абстрактных классов и интерфейсов

Инкапсуляция

- Инкапсуляция – объединение в одной оболочке данных и методов их обработки, а также способность объекта скрывать внутреннее устройство своих полей и методов
- Согласно данному принципу, класс рассматривается, как «черный ящик». Внешний пользователь не знает детали реализации объекта и работает с ним только путем предоставленного объектом интерфейса

Определение классов

```
public class MyClass
    {
        public MyClass()
        { }
        protected int_var;
    }
```

Модификаторы доступа

public
Доступ к типу или члену возможен из любого другого кода в той же сборке или другой сборке, ссылающейся на него.
private
Доступ к типу или члену можно получить только из кода в том же классе или структуре.
protected
Доступ к типу или элементу можно получить только из кода в том же классе или структуре, либо в производном классе.
internal
Доступ к типу или члену возможен из любого кода в той же сборке, но не из другой сборки.
protected internal
Доступ к типу или элементу может осуществляться любым кодом в сборке, в которой он объявлен, или из наследованного класса другой сборки.
Доступ из другой сборки должен осуществляться в пределах объявления класса, производного от класса, в котором объявлен защищенный внутренний элемент, и должен происходить через экземпляр типа производного класса.

Задание

- Создать класс с использование следующих модификаторов доступа: public, private, protected. Показать различия в использовании.
- Продемонстрировать ограничения использования var в классе

Модификаторы доступа

- Static - Объявляет член, который относится к типу, а не к конкретному объекту
- Const – констатнта
- Readonly – поле, доступное только для чтения
- Volatile - Указывает на то, что поле может быть изменено в программе операционной системой, оборудованием, параллельным потоком и т. д.

Задание

- Добавить к классу, созданному в предыдущей задаче примеры использования static, const, readonly
- Показать разницу между counst и readonly полями

Свойства

```
public class MyClass
  {
      public MyClass()
      { }
      private int int_var;
      public int Int_Var
      {
          get
          {
              return Int_Var;
          }
          set
          {
              Int_Var = value;
          }
      }
//автореализуемое свойство
      public int MyVar { get; set; }
 }
```

Модификаторы доступа для свойств

```
public int Int_Var
        {
            get
            {
                return Int_Var;
            }
            private set
            {
                Int_Var = value;
            }
```

```
 public int MyVar2 { get; private set;
}
```

Задание

- Добавить в созданный ранее класс свойства и автосвойства (при необходимости модифицировать код)

Передача параметров методам

- По умолчанию параметры передаются по значению
- Использование ref – передача параметра по ссылке
- В отличии от out, параметр ref должен быть инициализирован до передачи в метод

```
public void SomeMethod(int a, out, b, ref c)
{
        a = 3;
        b = 4;
        c = 5;
}
```

Задание

- Продемонстрировать на примере созданного ранее класса стандартное поведение при передаче параметров в методах, а также с использованием модификаторов ref и out
- Исследовать разницу передачи параметров для ссылочных и значимых типов. Продемонстрировать возможности использования ссылочных типов (и разницу) с модификаторами ref и out

Именованные аргументы

- public string FullName(string FirstName, string LastName)
- {
- return FirstName + “ “ + LastName;
- }

- FullName (“Pavlo”, “Kravchenko”);
- FullName (LastName: “Kravchenko”, FirstName: “Pavlo”);

Необязательные аргументы

```
public string FullName(string LastName, string FirstName = “”)
{
        return FirstName + “ “ + LastName;
}

public string FullName(string FirstName = “”, string LastName) // ошибка
{
        return FirstName + “ “ + LastName;
}
```

Ключевое слово params

- Params позволяет определить параметр метода, принимающий переменное число аргументов
- class Program
- {
- public static void First_exampe(params int[] List) { }
- public static void Second_exampe(params object[] List) { }
- static void Main(string[] args)
- {
- First_exampe(1, 2);
- First_exampe(1, 2, 3);
- Second_exampe(1, 4.5, "param");
- Second_exampe();
- }
- }

Задание

- Продемонстрировать на примере созданного ранее класса и методов использование именованных и необязательных аргументов, а также передачу параметров с помощью params
- Исследовать возможность использования модификаторов ref и out в этих случаях
- Исследовать возможности передачи массивов в виде параметров и возвращения в методах
- Определить разницу в передачи параметров в виде массива и с помощью params

Перегрузка методов

```
class Display
{
    void DisplayString (string str) { }
    void DisplayInt (int i)  { }
}
```

Задание

- Перегрузить 2 метода из предыдущих примеров

Частичные классы

- Возможность определения классов в разных файлах.
- Перед классом использовать partial

Частичные методы

- Модификатор partial можно применять и к методам
- Ограничения использования частичных методов:
  - Возвращается void
  - Не могут принимать параметры out, но допускается ref
  - Не могут быть external
  - Не могут быть virtual или иметь модификатор доступа (они приватные)
  - Не могут быть static и unsafe
  - Не могут вызываться делегатами

Задание

- Продемонстрировать возможности использования partial для классов и методов

Статичные классы

- Используется модификатор static
- Создавать экземпляры статического класса не разрешено

Задача

- Продемонстрировать возможности использования статических классов и методов другими классами и методами (в том числе классом Program и методом Main)

Создание объектов

- MyClass my_obj = new MyClass();

- Инициализаторы объектов:
- new MyClass {Name = “Pavlo”, Age = 23};

Конструкторы

- Каждый раз когда создается класс или структура, вызывается конструктор
- Может быть несколько конструкторов
- Если не предоставить конструктор, создаться конструктор по умолчанию
- Статический конструктор используется для инициализации статических данных или для выполнения определенного действия разово

Явный и неявный вызов конструктора

- class MyClass
- {
- public int X {get; set;}
- public int Y {get; set;}
- public MyClass () { }
- public MyClass (int x, int y) { }
- }
- MyClass my_obj = new MyClass();
- MyClass my_obj = new MyClass(10, 20);
- MyClass my_obj = new MyClass() {X = 10, Y = 20};
- MyClass my_obj = new MyClass {X = 10, Y = 20};
- MyClass my_obj = new MyClass(0, 0) {X = 10, Y = 20};

Статические конструкторы

- Свойства:
  - Не принимают модификаторы доступа и не имеют параметров
  - Вызывается автоматически для инициализации класса перед созданием первого экземпляра или ссылкой на какие-либо статические члены
  - Нельзя вызвать напрямую
  - Пользователь не управляет тем, когда статический конструктор выполняется в программе
  - Если статический конструктор инициирует исключение, среда выполнения не вызывает его во второй раз, и тип остается не инициализированным на время существования приложения

Задача

- Создать класс с конструктором по умолчанию и с параметрами. Один конструктор должен быть статическим

Вызов одних конструкторов из других

```
class MyClass
{
 public MyClass (int i, string str) {…}
      public MyClass(int i) : this (i, “Some String”)
      {…}

}
```

Задача

- Дополнить предыдущий класс использованием конструктора с вызовом из других конструкторов

Домашнее задание

- Создать класс, в котором реализован механизм вычисления периметра фигуры. При этом координаты вершин фигуры сохранять в виде массива. Координаты могут быть целых и действительных типов. Размерность массива задавать по количеству координат для каждой отдельно созданной фигуры. Реализовать принцип сокрытия информации.
- В другом файле расширить функциональность созданного класса. Добавить к типу дополнительную функциональность в виде перегруженных методов вычисления площади для треугольника (равностороннего), квадрата, прямоугольника, ромба и шестиугольника (в этом классе). Реализовать 2 универсальных метода вычисления периметра фигуры с использованием параметров по умолчанию и передачей произвольного числа аргументов. Создать свойства для хранения площади и периметра фигуры. Один из них должен быть статическим (для вычисления периметра любой фигуры).
- Создать фигуры с использованием конструкторов по умолчанию и с параметрами. Вычислить площадь и периметр каждой фигуры. Вычислить периметр фигуры, для которой не реализован мет вычисления площади.