Занятие 3, ч2. 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

Контракт

- Реализуется с помощью инкапсуляции
- Представляет собой интерфейс, видимый потребителю класса
- При этом возможно изменение внутренней реализации

Индексаторы

- Индексатор обеспечивает механизм инкапсуляции множества значений, так же, как свойство инкапсулирует одно значение
- Индексатор позволяет обеспечить индексированный доступ к объекту
- Главное назначение индексаторов – поддержать создание специализированных массивов, на которые налагаются одно или несколько ограничений
- int this[int index]
- {
- get { }
- set { }
- }

Задача

- Продемонстрировать возможности использования индексаторов для созданного ранее класса
- Использовать одномерный и многомерный индексатор

Особенности структуры

- Структура очень похожа на класс, кроме того, что она уменьшает накладные расходы из-за способа, которым CLR создает и управляет экземплярами структуры. Однако, структуры имеют некоторые ограничения. Как правило, структуры используются для моделирования элементов, которые содержат относительно небольшое количество данных
- Для создания экземпляра типа структура необязательно использовать оператор new, достаточно просто объявить переменную этого типа
- Для структуры нельзя определить конструктор по умолчанию (без параметров)
- Все конструкторы структуры должны явно инициализацировать каждое поле в структуре
- Конструктор в структуре не может вызывать другие методы до присваивания значений всем ее полям

Композиция классов

- Композиция (а также ассоциация и агрегирование) – механизм для создания нового класса путем объединения нескольких объектов существующих классов в единое целое
- При агрегировании между классами действует «отношение принадлежности»
  - У самолета есть крылья, хвост, шасси
  - У человека есть голова, ноги, руки
- Вложенные объекты обычно объявляются закрытыми (private) внутри класса агрегатора

Наследование

- Наследование – это механизм ООП, позволяющий описать новый класс на основе уже существующего
- При наследовании свойства и функциональность родительского (базового) класса наследуются новым классом
- Класс-наследник (потомок или дочерний) имеет доступ к публичным (public) и защищенным (protected) методам и полям родительского класса
- Класс-наследник может добавлять свои данные и методы, а также переопределять методы базового класса

Виды наследования

- Открытое (public)
- Закрытое (private)
- Защищенное (protected)

- По количеству базовых классов:
- Одиночное
- множественное

Полиморфизм

- Полиморфизм – это свойство, которое позволяет одно и то же имя использовать для решения нескольких схожих, но технически разных задач. Целью полиморфизма является использование одного имени для задания общих для класса действий
- Использование полиморфизма достигается присвоением указателю на базовый класс адреса производного класса с переопределенными методами

Вложенные классы

- Вложенные классы определяются внутри других классов. Имеют доступ ко всем членам содержащего класса (в том числе к приватным)

- Пример композиции и вложенных классов

Наследование

```
class BaseClass
{
    protected int value;
}

class DerivedClass : BaseClass
{
    public DerivedClass() {value = 10;}
}
```

Несколько конструкторов в классах

```
class BaseClass
{
    public BaseClass () {  }
    public BaseClass (int x, int y) {  }
}

class DerivedClass : BaseClass
{
    public DerivedClass() { }
    public DerivedClass (int x, int y, int z): base (x, y) { }
}
```

Вызов конструкторов и финализаторов

- Конструкторы производных классов вызывают конструкторы базовых классов, но выполняются после них.
- Если в конструкторе производного класса нет явного вызова конструктора базового класса, компилятор пытается вызвать конструктор по умолчанию. Если при этом в базовом классе определен хотя бы один конструктор (и он имеет параметры), возникнет ошибка времени компиляции.
- Финализаторы создаются в виде деструкторов. Уничтожение объекта производного класса приводит к вызову деструктора этого класса и всех базовых классов.

Задача

- Продемонстрировать порядок вызова конструкторов и методов (в том числе статических) для базового и наследуемого классов

Задача

- Продемонстрировать порядок вызова финализаторов для базового и наследуемого классов

base и this

- this – ссылка на текущий экземпляр класса
- base – ссылка на базовый класс

```
class Employee
{
    protected virtual void DoWork()
    {
        ...
    }
}

class Manager : Employee
{
    protected override void DoWork()
    {
        // Do processing specific to Managers
        ...
        // Call the DoWork method in the base class
        base.DoWork();
    }
    ...
}
```

Задача

- Продемонстрировать использование this и base в наследовании

Переопределение и сокрытие методов

- При переопределении (override) метода предоставляется реализация, имеющая тот же смысл, что и оригинальный метод, но являющая специфической для наследуемого класса
- Переопределить можно только методы, которые помечены в базовом классе как virtual, override или abstract.

- Сокрытие (использование new). В наследуемом классе можно определить методы, имеющие то же имя, что и методы базового класса, даже если они не помечены как virtual, abstract или override. Это означает, что не существует никакой связи между переопределенным и оригинальным методами, и новый метод скрывает оригинальный

Использование new

- При создании метода производного класса можно использовать new:
- public class BaseClass
- {
- public void SomeMethod() { }
- }
- public class DerrivedClass : BaseClass
- {
- public new void SomeMethod() { }
- }

Герметизированные классы

- Sealed – указывает, что класс герметизированный, и наследование от него не разрешается

Абстрактные классы

- Противоположны герметизированным классам. Иногда возникает необходимость спроектировать класс, который будет служить исключительно базовым классом.
- Используется abstract
- Создание объектов абстрактного класса не допускается

Задача

- Продемонстрировать возможности использования абстрактных и герметизированных классов

Виртуальные методы

- Используются для реализации механизма динамического полиморфизма
- virtual
- override

Абстрактные методы

- Абстрактные методы, в отличии от виртуальных, должны обязательно быть переопределены
- Классы, которые содержат абстрактные методы, должны быть тоже абстрактными

Методы sealed

- Используется для запрещения производным классам дальнейшего переопределения метода

Задача

- Реализовать базовый и производные классы с виртуальными, абстрактными и sealed методами

Полиморфизм

- Обеспечивается благодаря использованию виртуальных методов
- Виртуальные методы, определенные в классах, разделяющих иерархию наследования, позволяют вызывать различные версии одного и того же метода в зависимости от типа объекта, который определяется динамически во время выполнения

Приведение типов с использованием полиморфизма

- class Employee
- {...
- }
- class Manager : Employee
- {...
- }
- class ManualWorker : Employee
- {...
- }
- ...
- // Manager constructor expects a name and a grade
- Manager myManager = new Manager("Fred", "VP");
- //ManualWorker myWorker = myManager; // error – different types
- Employee myEmployee = myManager;
- Manager myManagerAgain = myEmployee as Manager;
- // OK - myEmployee is a Manager
- ...
- ManualWorker myWorker = new ManualWorker("Bert");
- myEmployee = myWorker; // myEmployee now refers to a ManualWorker
- ...
- myManagerAgain = myEmployee as Manager; // returns null

Задача

- Создать базовый класс и 3 производных класса с виртуальными методами. Переопределить в одном производном классе метод с помощью new. Продемонстрировать возможности полиморфизма при вызове виртуальных методов.
- Определить особенность переопределенного метода.

Тип object

- object – это первичный родительский тип, от которого наследуются все внутренние и пользовательские типы
- Ссылку на object можно применять для связи с объектом любого конкретного подтипа
- Тип object реализует множество базовых методов общего назначения, среди которых Equals(), GetHashCode(), GetType(), ToString()
- Пример

Упаковка и распаковка

- Упаковка – преобразование значимого типа в тип object или любой другой тип интерфейса, реализуемый этим типом значения.
- Распаковка извлекает тип значения из объекта
- int i = 32;
- object o = i;

- o = 111;
- i = (int) o;

Тип dynamic

- Тип является статическим типом, но объект типа dynamic обходит проверку статического типа. В большинстве случаев он функционирует, как тип object. Во время компиляции предполагается, что элементы с типом dynamic поддерживают любые операции
- Операции, которые содержат выражения типа dynamic, не разрешаются или выполняется проверка типа компилятором. Компилятор пакует сведения об операции, затем эти сведения используются для оценки операции во время выполнения. Как часть процесса, переменные типа dynamic компилируются в переменные типа object. Поэтому тип dynamic существует только во время компиляции, но не во время выполнения.

Тип dynamic

```
class ExampleClass
{
    public ExampleClass(){ }
    public ExampleClass(int v){ }
    public void exampleMethod1(int i){ }
    public void exampleMethod2(string str){ }
}

static void Main(string[] args)
{
    ExampleClass ec = new ExampleClass ();
    // The following line causes a compiler error if exampleMethod1
    // has only one parameter.
    //ec.exampleMethod1(10, 4);

    dynamic dynamic_ec = new ExampleClass ();
    // The following line is not identified as an error by the
    // compiler, but it causes a run-time exception.
    dynamic_ec.exampleMethod1(10, 4);

    // The following calls also do not cause compiler errors, whether
    // appropriate methods exist or not.
    dynamic_ec.someMethod("some argument", 7, null);
    dynamic_ec.nonexistentMethod();
}
```

Интерфейсы

- Интерфейс определяет контракт, специфицирующий методы, которые класс должен реализовывать. Интерфейс может также определять свойства. Однако детали реализации этих свойств являются ответственностью класса. Чтобы добавить свойства в интерфейс, можно использовать тот же синтаксис автоматических свойств, за исключением того, что нельзя указать модификатор доступа.
- Синтаксически интерфейс похож на класс за исключением того, что методы в нем только объявляются и не предлагается код, их реализующий.
- Для определения интерфейса используется ключевое слово interface.
- Внутри интерфейса, объявляются методы также, как в классе или структуре, для которых никогда не указывается модификатор доступа (public, private или protected), тело метода заменяется точкой с запятой.

Интерфейсы

- Для реализации интерфейса объявляются класс или структура, наследующие интерфейс, а также предоставляется код для каждого метода, который интерфейс определяет. При реализации интерфейса, следует убедиться, что каждый имплементируемый метод соответствует в точности соответствующему методу интерфейса, согласно следующими правилами:
  - Имена методов и типы возвращаемого значения в точности совпадают.
  - Любые параметры (в том числе модификаторы ref и out) в точности совпадают.
  - Все методы реализации интерфейса доступны. Однако, если используется явная реализация интерфейса, метод не должен иметь модификатор доступа.
- Если между определением интерфейса и его реализацией есть разница, класс не будет компилироваться. Хотя нельзя создать типы структур, которые наследуются от других типов структур или классов, но тип структура может реализовывать интерфейс.

Явная и неявная реализация интерфейса

```
interface IPerson
{
    string Name { get; set; }
    int Age { get; }
    DateTime DateOfBirth { set; }
    void PersonDataOutput();
}
class Person : IPerson
{
    public string Name
    {
        get
        {
            throw new NotImplementedException();
        }
        set
        {
            throw new NotImplementedException();
        }
    }
    int IPerson.Age
    {
        get { throw new NotImplementedException(); }
    }
    public DateTime DateOfBirth
    {
        set { throw new NotImplementedException(); }
    }
    public void PersonDataOutput()
    {
        Console.WriteLine("Name = {0}, Age = {1}, Date = {2},Name, Age, DateOfBirth)
    }
}
```

- **Используются при наследовании нескольких интерфейсов с одинаковыми именами методов**

Явная и неявная реализация интерфейса

```
interface IPerson
{
    string Name { get; set; }
    int Age { get; }
    void PersonDataOutput();
}
Interface IStudent
{
    int Course {get; }
     void PersonDataOutput();
}
class Person : Iperson, IStudent
{
    public string Name
    {   set { throw new NotImplementedException();   }
        set { throw new NotImplementedException();       }
    }
    int Age
      {  get { throw new NotImplementedException(); }
      }
   void IPerson.PersonDataOutput()
    {   Console.WriteLine("Name = {0}, Age = {1}, Date = {2},Name,
Age, DateOfBirth)
      }
    int Course
      {  get { throw new NotImplementedException(); }
      }
   void IStudent.PersonDataOutput()
      {   Console.WriteLine("Cource = {0}, Cource);
      }
}
Person per = new Person();
IPerson preson = per;
person.PersonDataOutput();
Istudent student = per;
student.PersonDataOurput();
```

Наследование интерфейсов

- В отличии от наследования классов, возможно множественная реализация интерфейсов
- interface ICalculator
- {
- double Add();
- double Subtract();
- double Multiply();
- double Divide();
- }
- class Calculator : ICalculator , IComparable
- {
- // The methods of the ICalculator interface return test data in this code.
- public double Add()
- {
- return 0;
- }
- public double Subtract()
- {
- return 0;
- }
- public double Multiply()
- {
- return 0;
- }
- public double Divide()
- {
- return 0;
- }
- public int CompareTo(Object obj)
- {
- //...
- }
- }

Задача

- Создать интерфейс студент со свойствами и методом для вывода данных. Реализовать интерфейс в классе, используя явную и неявную реализацию.
- Обратится к свойствам и вызвать методы в главной программе для объекта класса

Ссылки на объект через интерфейс

- Calculator myCalculator = new Calculator();
- ICalculator iMyCalculator = myCalculator;

Ссылки на объект через интерфейс

- Нельзя присвоить объект ICalculator переменной Calculator без приведения, предварительно не проверив, является ли он ссылкой на объект Calculator, а не на какой-нибудь другой класс, также реализующий интерфейс ICalculator.
- Чтобы убедиться, что объект реализует интерфейс, можно использовать ключевые слова is и as, или проверить, что объект, на который ссылается интерфейс, экземпляр определенного класса.

Ссылки на объект через интерфейс

- Ссылки на объект через интерфейс позволяют определить методы, которые могут принимать различные типы в качестве параметров, при условии, что типы реализуют указанный интерфейс.
- Метод PerformAnalysis может принимать любой аргумент, реализующий интерфейс ICalculator.
- int PerformAnalysis(ICalculator calculator)
- {
- //...
- }
- **При ссылке на объект через интерфейс, можно вызвать только методы, которые видны через интерфейс.**

Задача

- Создать 2 интерфейса IPerson и IStudent со свойствами, методами для вывода каждого свойства отдельно, и методами для вывода всей информации по каждому свойству для каждого интерфейса с одинаковыми именами.
- Реализовать интерфейсы в классе. При этом должны быть методы вывода всех данных, и отдельно реализации каждого интерфейса.
- Вызвать поочередно каждый метод базового и производных классов

Абстрактные классы и интерфейсы

Абстрактные классы	Интерфейсы
Не могут быть созданы напрямую, но могут содержать конструктор, который вызывается в классе-наследнике	Не могут содержать конструктор
Абстрактный класс может быть так дополнен элементами, что это не повлияет на его классы-наследники	Если в интерфейс помещаются дополнительные элементы, все классы, которые его реализуют, должны быть дополнены
Может хранить данные в полях	Не может хранить данные
Виртуальные элементы могут содержать базовую реализацию. Допустимы невиртуальные элементы	Все элементы являются виртуальными и не включают реализацию
Класс может наследоваться от единственного абстрактного класса	Класс может реализовывать несколько интерфейсов
Класс-наследник может переопределить только некоторые элементы абстрактного класса	Класс, который реализует интерфейс, должен реализовать все элементы интерфейса
Наследование поддерживается только для классов	Интерфейс может быть реализован структурой

Задача

- Создать 2 интерфейса IPerson и IStudent со свойствами, методами для вывода каждого свойства отдельно, и методами для вывода всей информации по каждому свойству для каждого интерфейса с одинаковыми именами.
- Реализовать интерфейсы в классе. При этом должны быть методы вывода всех данных, и отдельно реализации каждого интерфейса.
- Наследовать от него 2 класса: студент-спортсмен и студент-артист с собственными методами.
- Вызвать поочередно каждый метод базового и производных классов
- Изменить базовый класс на абстрактный
- Вызвать поочередно каждый метод базового и производных классов

Домашнее задание

- Создать интерфейс для работы с фигурой с вершинами (хранения данных, вычисление сторон, периметра, площади).
- Создать интерфейс для работы с фигурой без вершин (хранение данных, вычисление длины эллипса, площади)
- Для координат вершин фигуры создать отдельный класс.
- Создать класс фигура, в котором реализован механизм вычисления сторон, периметра и площади фигуры по координатам ее вершин, и который реализует созданные ранее интерфейсы. Класс должен быть максимально универсальным для вычисления периметра, площади, сторон (почти) любой фигуры (по аналогии с предыдущим домашним заданием). При необходимости методы могут быть абстрактными. Площадь и периметр хранить в соответствующих свойствах.
- Создать классы треугольник (равносторонний), квадрат, прямоугольник и круг, которые реализуют интерфейс (-ы) и/или наследуют класс фигура (или друг друга). Реализовать методы вычисления периметра фигуры, длину круга, а также их площади (при необходимости использовать переопределение и сокрытие методов). Классы, которые не нужно наследовать, реализовать герметизированными.
- Создать фигуры с использованием конструкторов по умолчанию и с параметрами. Вычислить площадь и периметр каждой фигуры (в том числе, используя универсальный класс), используя возможности полиморфизма и ссылок на объекты с помощью переменной, определенной, как интерфейс.