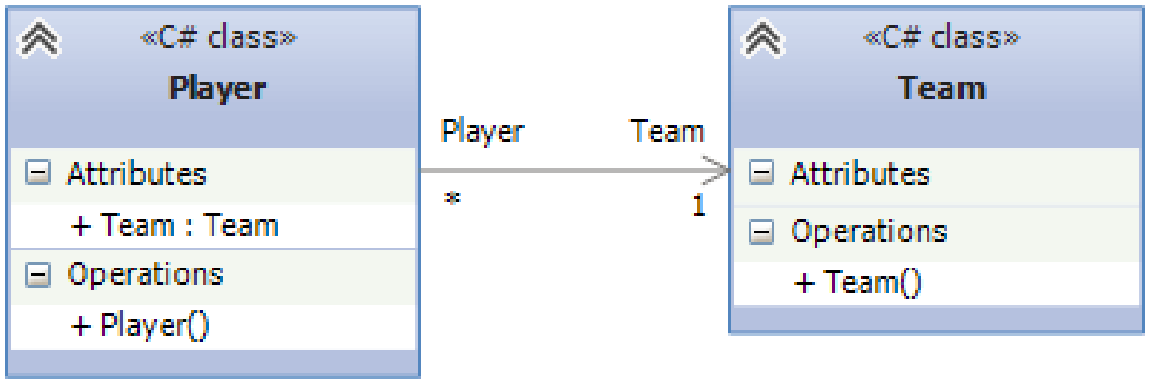
В.В. Подбельский, О.В. Максименкова

Модуль 2, практическое занятие 8

Отношения между классами: агрегация, композиция, наследование

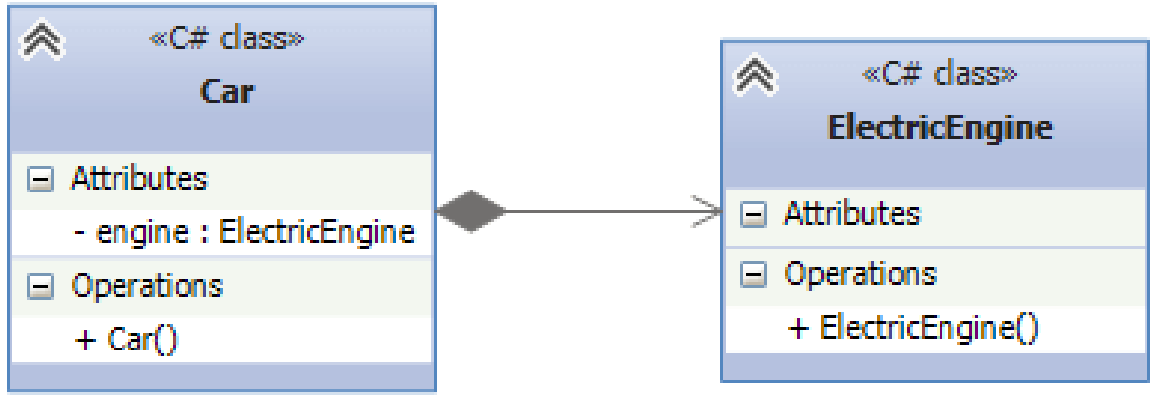
Ассоциация

```
class Team
{

}
class Player
{
    public Team Team { get; set; }
}
```



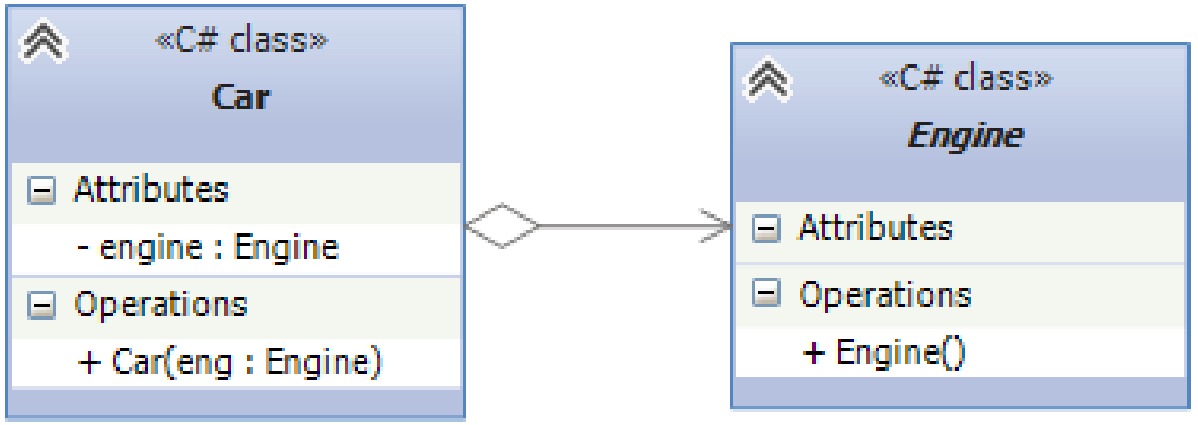
Композиция

```
public class ElectricEngine
{ }

public class Car
{
    ElectricEngine engine;
    public Car()
    {
        engine = new ElectricEngine();
    }
}
```

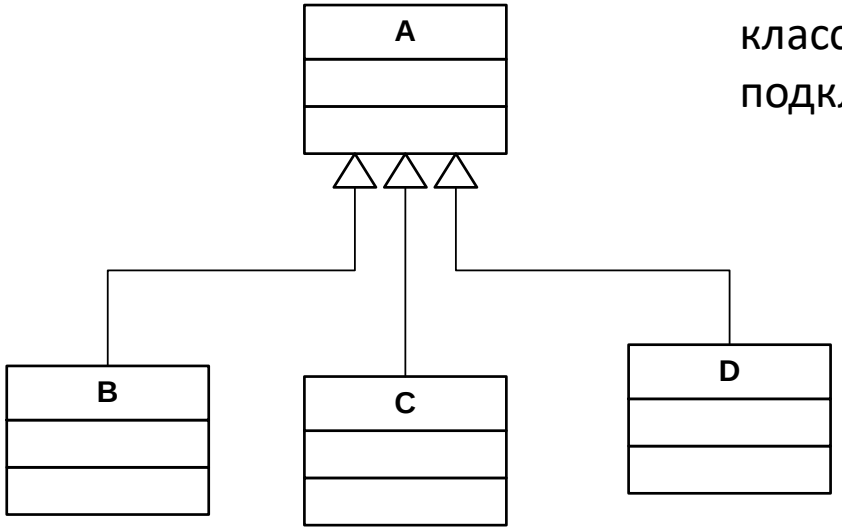


Агрегация

```
public abstract class Engine
{ }

public class Car
{
    Engine engine;
    public Car(Engine eng)
    {
        engine = eng;
    }
}
```



Наследование



Необходимое условие наследования
Если класс X не является разновидностью класса Y, то класс X не может быть подклассом класса Y.

```
class A { }
    class B : A { }
    class C : A { }
    class D : A { }
```

Задача 1. Наследование без полиморфизма

Класс **Validator** используется для проверки валидности (правильности) строк. Поле класса length содержит максимально-допустимую длину строки.

Унаследовать от **Validator** класс **EmailValidator**. Конструктор **EmailValidator** () содержит один параметр типа **int** – максимально-допустимую длину строки с электронной почтой. Поместить в класс **EmailValidator** метод **EmailValidate**() со строковым параметром **S**. Метод возвращает **true**, если в строке содержится адрес электронной почты (воспользуйтесь регулярными выражениями).

Унаследовать от **Validator** класс **IPValidator**. Конструктор **IPValidator**() без параметров присваивает полю length значение 15, конструктор с параметрами проверяет параметр длины строки. Поместить в класс **IPValidator** метод **IPValidate** () со строковым параметром **S**. Метод возвращает **true**, если переданное значение соответствует формату XXX.XXX.XXX.XXX и **false** в противном случае.

В консольном приложении создать массив валидаторов и проверить тестовые строки на соответствие почтам и IP-адресам

Ниже предложена на слайдах приведена частичная реализация, которая использует полиморфную ссылку для доступа к валидаторам-наследникам, но не использует полиморфизм полноценно, что приводит к стене условных операторов при вызове…. Старайтесь избегать таких решений!

Полный код можно получить по ссылке: https://replit.com/@olgamaksimenkova/InheritanceWithoutPolim

Оператор is (справочник по C#) (https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/is)

Задача 1. Наследование без полиморфизма

```
/// <summary>
    /// Класс Validator используется для проверки валидности (правильности)
    /// строк. В поле класса содержится максильно-допустимая длина строки
    /// </summary>
    public class Validator
    {
        protected int length;
        protected Validator() { }
        protected Validator(int x)
        {
            if (x > 0)
            {
                Length = x;
            }

        }
        public int Length {
            get { return length; }
            set
            {
                if (value > 0)
                {
                    Length = value;
                }
                else
                {
                    throw new ArgumentOutOfRangeException("Length must be positive...");
                }
            }
        }
     }
```

Задача 1. Наследование без полиморфизма

```
/// <summary>
    /// Класс валидатор почт, наследник валидатора Validator
    /// </summary>
    public class EmailValidator : Validator
    {
        public EmailValidator() {  }
        public EmailValidator(int n) : base(n) { }
        public bool EmailValidate(string testValue)
        {
            Regex regex = new Regex(@"^([\w\.\-]+)@([\w\-]+)((\.(\w){2,3})+)$");
            if (regex.IsMatch(testValue))
            {
                return true;
            }
            return false;
        }
    }
```

Задача 1. Наследование без полиморфизма

```
/// <summary>
    /// Класс валидатор IP-адресов, наследник валидатора Validator
    /// </summary>
    public class IPValidator : Validator
    {
        public IPValidator () { }
        public IPValidator (int n) : base(n) { }
        public bool IPValidate(string testValue)
        {
            Regex regex = new Regex(@"\b(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-
9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\b");
            if (regex.IsMatch(testValue))
            {
                return true;
            }
            return false;
        }
    }
```

Задача 1. Наследование без полиморфизма

```
public class Program     {
   public static void Main(string[] args)       {
      string[] testValues = { "myMail@hse.ru", "error@in.mail@mail.ru", "mailhost.ru", "191.167.1.0",
"localhost.ru", "102.168.255.255"};
      Validator[] validatorChain = { new EmailValidator(), new IPValidator() };
      foreach(string testString in testValues)    {
         foreach(Validator cur in validatorChain)  {
            if (cur is EmailValidator) // если текущий объект валидатор почт {
               if (((EmailValidator)cur).EmailValidate(testString))
               // вызываем метод-валидатор
               {
                   Console.WriteLine(testString + " EMAIL");
                   break;
               }
            }
            else   {
               if (cur is IPValidator) // если текущий объект валидатор IP адресов  {
                  if (((IPValidator)cur).IPValidate(testString))
                  // вызываем метод-валидатор
                  {
                      Console.WriteLine(testString + " IP");
                      break;
                  }
               }
               Console.WriteLine(testString + " UNKNOWN");
            }
         }
      }
   }
}
```

Отсутствие полиморфной версии метода, приводит к дублированию кода и лишним проверкам типа с использованием IS

Задания к задаче 1

1. Дополните коды конструкторов в классах наследниках проверками, указанными в задании
2. Переопределите метод ToString() для всех типов в иерархии. В классе почте возвращайте строку “EMAIL”, в классе IP-адресов строку “IP”
3. * Получите тестовые данные с почтами, IP-адресами и не валидными строками из текстового файла, файл расположите рядом с запускаемым файлом программы

Задача 2. Наследование с полиморфизмом

Модифицируем код предыдущий задачи. Введём в базовый тип Validator виртуальный метод Validate() и переопределим (**override**) его в наследниках.

- Virtual (Справочник по C#) (https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/virtual)
- Override (справочник по C#) (https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/override)

Изменим код консольного приложения:

Старая версия без виртуального метода

```
foreach(string testString in testValues)    {
   foreach(Validator cur in validatorChain)  {
      if (cur is EmailValidator) // если текущий объект валидатор почт {
         if (((EmailValidator)cur).EmailValidate(testString))
        // вызываем метод-валидатор
         {
             Console.WriteLine(testString + " EMAIL");
             break;
         }
      }
      else   {
         if (cur is IPValidator) // если текущий объект валидатор IP адресов  {
            if (((IPValidator)cur).IPValidate(testString))
           // вызываем метод-валидатор
            {
                Console.WriteLine(testString + " IP");
                break;
            }
         }
         Console.WriteLine(testString + " UNKNOWN");
      }
   }
}
```

Новая версия с виртуальным методом

```
foreach (string testString in testValues)
            {
                foreach (Validator cur in validatorChain)
                {
                    if (cur.Validate(testString)) {
                        Console.WriteLine(testString + " " + cur.ToString());
                        break;
                    }
                }
            }
```

https://replit.com/@olgamaksimenkova/InheritanceWithPolim

Задача 2. Наследование с полиморфизмом

```
/// <summary>
    /// Класс Validator используется для проверки валидности (правильности)
    /// строк. В поле класса содержится максильно-допустимая длина строки
    /// </summary>
    public class Validator
    {
        protected int length;
        protected Validator() { }
        protected Validator(int x)
        {
            if (x > 0)
            {
                Length = x;
            }

        }
        public int Length
        {
            get { return length; }
            set
            {
                if (value > 0)
                {
                    Length = value;
                }
                else
                {
                    throw new ArgumentOutOfRangeException("Length must be positive...");
                }
            }
        }
        public virtual bool Validate(string testValue)
        {
            if (testValue.Length < this.Length)
            {
                return true;
            }
            return false;
        }
    }
```

Добавляем виртуальный метод Validate(), чтобы при вызове по ссылке базового типа, среда исполнения ориентировалась на тип объекта при override

Задача 2. Наследование с полиморфизмом

```
/// <summary>
    /// Класс валидатор почт, наследник валидатора Validator
    /// </summary>
    public class EmailValidator : Validator
    {
        public EmailValidator() { }
        public EmailValidator(int n) : base(n) { }
        public override bool Validate(string testValue)
        {
            Regex regex = new Regex(@"^([\w\.\-]+)@([\w\-]+)((\.(\w){2,3})+)$");
            if (regex.IsMatch(testValue))
            {
                return true;
            }
            return false;
        }
        public override string ToString()
        {
            return "EMAIL";
        }
    }
```

Добавляем переопределение метода Validate()
и переопределение метода ToString()

Задача 2. Наследование с полиморфизмом

```
/// <summary>
    /// Класс валидатор IP-адресов, наследник валидатора Validator
    /// </summary>
    public class IPValidator : Validator
    {
        public IPValidator() { }
        public IPValidator(int n) : base(n) { }
        public override bool Validate(string testValue)
        {
            Regex regex = new Regex(@"\b(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-
9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\b");
            if (regex.IsMatch(testValue))
            {
                return true;
            }
            return false;
        }
        public override string ToString()
        {
            return "IP";
        }
    }
```

Добавляем переопределение метода Validate()
и переопределение метода ToString()

Задача 2. Наследование с полиморфизмом

```
public class Program
    {
        public static void Main(string[] args)
        {
            string[] testValues = { "myMail@hse.ru", "error@in.mail@mail.ru",
"mailhost.ru", "191.167.1.0", "localhost.ru", "102.168.255.255" };
            Validator[] validatorChain = { new EmailValidator(), new IPValidator() };
            foreach (string testString in testValues)
            {
                foreach (Validator cur in validatorChain)
                {
                    if (cur.Validate(testString)) {
                        Console.WriteLine(testString + " " + cur.ToString());
                        break;
                    }
                }
            }
        }
    }
```

В цикле теперь полагаемся на определение типа объекта по полиморфной ссылке, метод будет вызываться по типу связанного объекта

Задание к задаче 2

1. Перенесите исправления для классов и источника данных из задания 1 в код задачи 2
2. Сейчас нераспознанные строки не выводятся на экран с меткой UNKNOWN, модифицируйте код консольного приложения, чтобы эта информация отображалась на экране

Задача 3. Наследование vs Агрегация

Часто вместо наследования может быть реализована агрегация. Нужно не забывать, что в таком случае доступ к объекту-агрегату не может осуществляться по полиморфной ссылке базового типа, т.к. при наследовании реализуется отношение **is a**, а при агрегации **has a**

В данной задаче реализован тип Point2D, представляющий точку на вещественной плоскости. На основе этого класса построены два других:

1. Класс InheritateCircle наследует из Point2D, т.е. конкретизирует его до окружности
2. Класс AggregateCircle агрегирует объект Point2D, то есть точка является частью окружности

Перенесите к свой проект код, доступный по ссылке: https://replit.com/@olgamaksimenkova/InherintanceVSAggregation

Задача 4

1. Владея знаниями, полученными в предыдущей задаче, реализуйте самостоятельно на основе кода задачи 2 о валидаторах новый класс MultiValidator, в который включен массив объектов с типом Validator. В массив должны добавляться валидаторы почт и IP-адресов (предусмотрите это в коде класса MultiValidator) – предусмотрите такую возможность в интерфейсе типа MultiValidator
2. В тестовом консольном приложении получите из файла массив тестовых строк, файл располагается рядом с исполнимым файлом. Создайте объект MultiValidator и свяжите его с одним валидатором почт и одним валидатором IP-адресов. На экран выведите результат прохождения валидации
3. * Разработайте дополнительный валидатор URLValidator, тоже наследник Validator, который проверят корректность URL-адресов (можете воспользоваться готовой регуляркой из интернета). Добавьте возможность связать новый валидатор с MultiValidator, допишите тестовый код в консольное приложение

Агрегация

Ассоциация -это двустороннее семантическое отношение классов.

Агрегация - это наиболее сильная форма ассоциации, показывающая связь между целым и его частью.

```
public class Unit {
    string name = "Unit";
    Part part;
    public Unit(Part p) { // конструктор целого
        part = p; // ссылка
    }
}
public class Part {
    string name="Part";
}
```

МЕМО: При агрегации «целое» не управляет временем жизни своих «частей»

Композиция

Композиция (**композитная агрегация**) – это самая сильная форма ассоциации, где часть неотъемлема от единого целого.

Композитная агрегация означает временну**ю** зависимость – создание целого, создание частей; удаление целого, удаление частей.

```
public class Unit {
        string name = "Unit";
        Part part;
        public Unit() { // конструктор целого
            part = new Part(); // создание части
        }
}
public class Part {
        string name="Part";
}
```

МЕМО: При композиции «целое» «управляет» временем жизни своих «частей»

Задача 5. Корзина покупок

Класс *Item,* моделирует одну покупку. У покупок есть название, цена и количество. Класс *ShoppingCart* реализует корзину покупок в виде массива элементов.

Дополните класс, реализующий корзину покупок в виде массива элементов.

Дополните *ShoppingCart* :

- Объявите переменную *_cart* и инициализируйте её массивом размера *capacity* в конструкторе
- Дополните метод *IncreaseSize ()* кодом. Размер массива должен увеличиваться на 3 элемента.
- Дополните метод *AddToCart*() кодом. Этот метод должен добавлять элемент в корзину и обновлять переменную *_totalPrice*.

Напишите программу, имитирующую покупки. Программа должна содержать цикл, который повторяется до тех пор, пока пользователь хочет что-нибудь купить. Каждую итерацию цикла считывайте название, цену и количество вещей, которые хочет приобрести пользователь и добавляйте их в корзину. После добавления элемента в корзину выводите содержимое корзины. После выхода из цикла напишите “Пожалуйста, заплатите …”, подставив вместо троеточия сумму покупок.

Доп. задание:

Дополните *ShoppingCart* индексатором для доступа к массиву.

https://repl.it/@Maksimenkova/ClassesInheritance01

Задача 5. Корзина покупок

```
using System;

/*
 * Item.cs
 *
 * Представляет предмет в корзине покупок.
 */

public class Item {
    /// <summary>
    /// Название предмета
    /// </summary>
    public string Name { get; }

    /// <summary>
    /// Цена предмета
    /// </summary>
    public double Price { get; }

    /// <summary>
    /// Количество предметов
    /// </summary>
    public int Quantity { get; }

    /// <summary>
    /// Создаёт новый предмет на основе переданных свойств
    /// </summary>
    /// <param name="itemName">Название предмета</param>
    /// <param name="itemPrice">Цена предмета</param>
    /// <param name="numPurchased">Количество предметов</param>
    public Item(string itemName, double itemPrice, int numPurchased)
    {
        Name = itemName;
        Price = itemPrice;
        Quantity = numPurchased;
    }

    /// <summary>
    /// Возвращает строку, представляющую текущий объект
    /// </summary>
    /// <returns></returns>
    public override string ToString()
    {
        return $"{Name}\t\t{Price:C}\t\t{Quantity}\t\t{Price * Quantity:C}";
    }
}
```

Задача 5. Корзина покупок

```
/*
 * ShoppingCart.cs
 *
 * Представляет корзину покупок
 */

public class ShoppingCart {
    private int _itemCount; // количество предметов в корзине
    private double _totalPrice; // цена всех предметов в корзине
    private int _capacity; // текущая вместимость корзины

    /// <summary>
    /// Создаёт новый экземпляр корзины с вместимостью в 5 элементов
    /// </summary>
    public ShoppingCart()
    {
        _capacity = 5;
        _itemCount = 0;
        _totalPrice = 0.0;
    }

    /// <summary>
    /// Добавляет предмет в корзину
    /// </summary>
    /// <param name="itemName">Название предмета</param>
    /// <param name="price">Цена предмета</param>
    /// <param name="quantity">Количество предметов</param>
    public void AddToCart(string itemName, double price, int quantity) { }

    /// <summary>
    /// Увеличивает вместимость корзины на 3
    /// </summary>
    private void IncreaseSize() { }

    /// <summary>
    /// Возвращает предметы в корзине с дополнительной информацией
    /// </summary>
    public override string ToString()
    {
        string contents = "\nShopping Cart\n";

        contents += "\nItem\t\tUnit Price\tQuantity\tTotal\n";

        for (int i = 0; i < _itemCount; i++)
            contents += _cart[i] + "\n";

        contents += $"\nTotal Price: {_totalPrice:C}\n";

        return contents;
    }
}
```

Решите самостоятельно (вспомним файлы)

Описать класс **Vector** – вектор на плоскости. Вектор задаётся двумя точками (**Point**). Классы **Point** и **Vector** находятся в отношении агрегации. Описать свойства доступа к концам вектора.
Связь вектора с точками задаётся в конструкторе с параметрами – точками.
1. В классе **Vector** определить перегруженный метод, вычисляющий скалярное произведение векторов **ScalarMult**():
    1. С параметром типа **Vector**
    2. С двумя параметрами типа **Point** (концы второго вектора)
2. В классе **Vector** определить перегруженный метод – векторное произведение векторов **VectorMult**():
    1. С параметром типа **Vector**
    2. С двумя параметрами типа **Point** (концы второго вектора)

В текстовом файле **dots.txt** разместить пары точек – координат концов векторов в формате **(x1;y1):(x2,y2)**. Дробная часть координаты отделяется от целой символом «точка». В случае, если формат исходных данных нарушен, программа должна оповещать об этом пользователя. (Файл разместить в папке проекта.)

В основной программе создать массив A объектов типа **Vector**, элементы которого - ссылки на объекты, построенные из данных файла **dots.txt**.

Массив A упорядочить по возрастанию величины их скалярных произведений с вектором **(0;0):(0;1)**. Для сравнения использовать лямбда-выражение. Упорядоченный массив вывести на экран.

Получить по массиву **A** массив **B** – векторов, полученных в результате попарных векторных произведений элементов массива **A** самих на себя. Вектора массива **B** сохранить в файле **B.txt** в таком же формате, что и в **dots.txt**.

Решите самостоятельно

Реализовать класс, представляющий сведения о человеке **Person**. Реализовать свойства: Ф.И.О.(string FullName), дата рождения (DateTime BirthDate), пол (bool IsMale). Реализовать метод для вывода информации о человеке void ShowInfo().

Реализовать класс, представляющий сведения о студенте **Student** (наследуется от **Person**). Реализовать свойства: название ВУЗа (string Institute), специальность (string Speciality).

Реализовать класс, представляющий сведения о сотруднике фирмы **Employee** (наследуется от **Person**). Реализовать свойства: название компании (string CompanyName), должность (string Post), график (string Schedule), оклад (decimal Salary).

В основной программе решить задачи:

- Создать объекты всех трех типов и вызвать ShowInfo(), чтобы показать всю доступную информацию.

- Создать массив **Person**[] arr и присвоить его членам объекты всех трех типов. Продемонстрировать работу метода ShowInfo() на массиве.

Решите самостоятельно

Попробуйте самостоятельно подумать об архитектуре типов, входящих в иерархии, в следующих задачах:

1. Класс **Administrator** наследует из класса **User** и предоставляет некоторые инструменты для управления администратором.
    - Создайте проект консольного приложения, добавьте в него код, указанный ниже и выполнить действия, указанные в комментариях TODO.
    - Код для работы получите по ссылке https://replit.com/@olgamaksimenkova/UserAdminInherit
2. В программе организовать работу с документами общего вида и заявлениями. Для этого
    - Описать класс Документ (**Doc**)
        - Поля класса: содержание документа, дата составления, дата изменения, поле состояние «подписан ли документ». Свойства доступа. Конструкторы. Методы: вывод информации о документе, метод отправки документа по почте.
    - Из класса Документ унаследовать класс Заявление (**App**)
        - Поля класса: автор заявления, кому адресовано заявление. Свойства доступа. Конструкторы Методы: метод изменения состояния заявления
        - В программе использовать любые дополнительные поля, свойства и методы
    - Реализовать в консольном приложении тестовый код, позволяющий получить документы разного типа и вывести информацию о них на экран, не используя операцию IS