

## ЛАБОРАТОРНА РОБОТА №3

### Тема: ВИКОРИСТАННЯ ВЛАСТИВОСТЕЙ. АВТОМАТИЧНІ ВЛАСТИВОСТІ. СКОРОЧЕНИЙ ЗАПИС ВЛАСТИВОСТЕЙ

**Мета:** Здійснити введення та виведення даних закритих полів класів, застосовуючі методи-властивості.

#### Постановка задачі:

1. Доповнити програмну реалізацію класів методами-властивостями класів, введення та виведення даних закритих полів класів
2. Протестувати програму, шляхом запуску програми, введенням та виведенням значень закритих полів.

### ХІД ВИКОНАННЯ РОБОТИ

- 1 У пункті меню «Тема, мета та постановка задачі ЛР №4 » звітного HTML-документу розмістити :
  - 1.1 Тему лабораторної роботи №4
  - 1.2 Мету лабораторної роботи №4
  - 1.3 Постановку задачі лабораторної роботи №4
  - 1.4 Вказати класи та методи-властивості, які реалізуються у цих класах.
- 2 У пункті меню «Місце розташування проекту.Середовище розробки» вказатим **місце розташування проекту** (шлях до папки з проектом). Це може бути GitHub або Google Disk. Також написати **середовище розробки програми**.
- 3 У пункті меню «Класи, з описом властивостей для закритих полів» вказати перелік класів, у яких використовуються властивості для закритих полів. Написати призначення цих полів
- 4 У пункті «Проектування структури ПЗ» розмістит Діаграму класів, попередньо добавивши методи-властивості у UML-діаграму класів.
- 5 У пункті «Кодування ПЗ (програмування)» показати файлову структуру проекту (скрін шот Solution explorer).. Надати програмні коди файлів проекту та програмний код класу Program та виведення результатів виконання
- 6 У пункті меню «Впровадження та супроводження ПЗ» розмістити посилання на папку з проектом, усіма файлами проекту, виконувальним файлом.exe, з можливістю відкрити проекту зі звіту.
- 7 У пункті меню «ВИСНОВКИ »написати висновки по лабораторній роботі №3. Зазначити переваги використання методів-властивостей

8 Для оформлення звітнього HTML-документу приведено орієнтовний макет.

**ЗВІТИ З ЛАБОРАТОРНИХ РОБІТ**  
**з ДИСЦИПЛІНИ «Технології .NET розробки веб-додатків»**  
Студент групи \_\_\_\_\_ Іванов Іван Іванович

Орієнтовний макет

Лабораторна робота №1	Лабораторна робота №2	Лабораторна робота №3	Лабораторна робота №4	Лабораторна робота №5	Лабораторна робота	Лабораторна робота №7	Лабораторна робота №8	Лабораторна робота №1
-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	--------------------	-----------------------	-----------------------	-----------------------

Тема, мета та постановка задачі ЛР№3

Місце розташування проекту (GitHub/GoogleDisk)  
Середовище розробки

Класи, з описом властивостей для закритих полів

Проектування структури ПЗ

UML- діаграма класів

Кодування ПЗ (програмування)

Файлова структура проекту

Програмний код всіх файлів  
Результат виконання програми

Впровадження та супроводження ПЗ

Папка з проектом

Запуск .exe файлу

**В И С Н О В К И**

Місце виведення інформації

## ПРИМІТКА:

Звіт з лабораторних робіт слід підготувати у вигляді **гіпертекстового документа у форматі html**. Документ має містити меню, яке включає команди, що подані нижче. Слід реалізувати запуск програм на виконання з гіпертекстового документа. Звіт та проекти лабораторних робіт слід записати на Гугл-Диск а. На диску має бути файл readme.txt, який містить відомості про автора звіту та проектів.

### Титульна html- сторінка.

Назва роботи

Автор (ПІБ, група, курс, № заліковки)

Фото

Рік навчання

## ТЕОРЕТИЧНІ ВІДОМОСТІ

Крім звичайних методів в мові C # передбачені спеціальні методи доступу, які називають **властивості** . Вони забезпечують простий доступ до полів класів та структур, дізнатися їх значення або виконати їх установку.

Стандартне опис властивості має наступний синтаксис:

```
1  [модификатор_доступа] возвращаемый_тип произвольное_название
2  {
3      // код свойства
4  }
```

наприклад:

```
1  class Person
2  {
3      private string name;
4
5      public string Name
6      {
7          get
8          {
9              return name;
10         }
11
12         set
13         {
14             name = value;
15         }
16     }
17 }
```

Тут у нас є закрите поле `name` і є загальнодоступне властивість `Name`. Хоча вони мають практично однакову назву за винятком регістра, але це не більше ніж стиль, назви у них можуть бути довільні і не обов'язково повинні збігатися.

Через це властивість ми можемо управляти доступом до змінної `name`. Стандартне визначення властивості містить блоки **get** і **set** . У блоці `get` ми повертаємо значення поля, а в блоці `set` встановлюємо. Параметр `value` представляє передане значення.

Ми можемо використовувати дане властивість наступним чином:

```
1  Person p = new Person();
2
3  // Установлюємо властивість - срабатывает блок Set
4  // значение "Tom" и есть передаваемое в свойство value
5  p.Name = "Tom";
6
7  // Получаем значение свойства и присваиваем его переменной - срабатывает блок Get
```

```
8 string personName = p.Name;
```

Можливо, може виникнути питання, навіщо потрібні властивості, якщо ми можемо в даній ситуації обходитися звичайними полями класу? Але властивості дозволяють вкласти додаткову логіку, яка може бути необхідна, наприклад, при присвоєнні змінної класу будь-якого значення. Наприклад, нам треба встановити перевірку за віком:

```
1 class Person
2 {
3     private int age;
4
5     public int Age
6     {
7         set
8         {
9             if (value < 18)
10            {
11                Console.WriteLine("Возраст должен быть больше 17");
12            }
13            else
14            {
15                age = value;
16            }
17        }
18        get { return age; }
19    }
20 }
```

Якби змінна `age` була б публічною, то ми могли б передати їй ззовні будь-яке значення, в тому числі негативне. Властивість же дозволяє приховати дані об'єкти та опосередовувати до них доступ.

Блоки `set` і `get` не обов'язково одночасно повинні бути присутніми в властивості. Якщо властивість визначають тільки блок `get`, то така властивість доступно тільки для читання - ми можемо отримати його значення, але не встановити. І, навпаки, якщо властивість має тільки блок `set`, тоді це властивість доступно тільки для запису - можна тільки встановити значення, але не можна отримати:

```
1 class Person
2 {
3     private string name;
4     // свойство только для чтения
5     public string Name
6     {
7         get
```

```

8      {
9          return name;
10     }
11 }
12
13 private int age;
14 // свойство только для записи
15 public int Age
16 {
17     set
18     {
19         age = value;
20     }
21 }
22 }

```

Хоча в прикладах вище властивості визначалися в класі, але точно також ми можемо визначати і використовувати властивості в структурах.

### **модифікатори доступу**

Ми можемо застосовувати модифікатори доступу не тільки до всього властивості, але і до окремих блоках - або get, або set:

```

1  class Person
2  {
3      private string name;
4
5      public string Name
6      {
7          get
8          {
9              return name;
10         }
11
12         private set
13         {
14             name = value;
15         }
16     }
17     public Person(string name)
18     {
19         Name = name;
20     }
21 }

```

Тепер закритий блок `set` ми зможемо використовувати тільки в даному класі - в його методах, властивості, конструкторі, але ніяк не в іншому класі:

```
1  Person p = new Person("Tom");
2
3  // Ошибка - set объявлен с модификатором private
4  //p.Name = "John";
5
6  Console.WriteLine(p.Name);
```

При використанні модифікаторів у властивостях слід враховувати ряд обмежень:

- Модифікатор для блоку `set` або `get` можна встановити, якщо властивість має обидва блоки (і `set`, і `get`)
- Тільки один блок `set` або `get` може мати модифікатор доступу, але не обидва відразу
- Модифікатор доступу блоку `set` або `get` повинен бути більш обмежуючим, аніж модифікатор доступу властивості. Наприклад, якщо властивість має модифікатор `public`, то блок `set` / `get` може мати тільки модифікатори `protected internal`, `internal`, `protected`, `private`

## Автоматичні властивості

Властивості керують доступом до полів класу. Однак що, якщо у нас з десяток і більше полів, то визначати кожне поле і писати для нього однотипне властивість було б утомливо. Тому в фреймворк `.NET` були додані автоматичні властивості. Вони мають скорочене оголошення:

```
1  class Person
2  {
3      public string Name { get; set; }
4      public int Age { get; set; }
5
6      public Person(string name, int age)
7      {
8          Name = name;
9          Age = age;
10     }
11 }
```

Насправді тут також створюються поля для властивостей, тільки їх створює не програміст в коді, а компілятор автоматично генерує при компіляції.

У чому перевага автосвойств, якщо по суті вони просто звертаються до автоматично створюваної змінної, чому б прямо не звернутися до змінної без автосвойств? Справа в тому, що в будь-який момент часу при необхідності ми можемо розгорнути автосвойство в звичайне властивість, додати в нього якусь певну логіку.

Варто враховувати, що не можна створити автоматичне властивість тільки для запису, як у випадку зі стандартними властивостями.

Автосвойствам можна привласнити значення за замовчуванням (ініціалізація автосвойств):

```
1  class Person
2  {
3      public string Name { get; set; } = "Tom";
4      public int Age { get; set; } = 23;
5  }
6
7  class Program
8  {
9      static void Main(string[] args)
10     {
11         Person person = new Person();
12         Console.WriteLine(person.Name); // Tom
13         Console.WriteLine(person.Age); // 23
14
15         Console.Read();
16     }
17 }
```

І якщо ми не вкажемо для об'єкта Person значення властивостей Name і Age, то будуть діяти значення за замовчуванням.

Варто відзначити, що в структурах ми не можемо використовувати ініціалізацію автосвойств.

Автосвойства також можуть мати модифікатори доступу:

```
1  class Person
2  {
3      public string Name { private set; get; }
4      public Person(string n)
5      {
6          Name = n;
7      }
8  }
```

Ми можемо прибрати блок set і зробити автосвойство доступним тільки для читання. В цьому випадку для зберігання значення цієї властивості для нього неявно буде створюватися поле з модифікатором readonly, тому слід враховувати, що подібні get-властивості можна встановити або з конструктора класу, як в прикладі вище, або при ініціалізації властивості:

```
1  class Person
```

```
2 {  
3   public string Name { get; } = "Tom"  
4 }
```

### **Скорочена запис властивостей**

Як і методи, ми можемо скорочувати властивості. наприклад:

```
1 class Person  
2 {  
3   private string name;  
4  
5   // еквівалентно public string Name { get { return name; } }  
6   public string Name => name;  
7 }
```

Додаткові матеріали