Инкапсуляция при помощи свойств

Свойства (properties) объединяют функции полей и методов. Они используются для чтения и записи **вспомогательного поля (backing field)**. Именно так называется поле, заданное свойством.

Закрытое поле numberOfCows станет **вспомогательным полем** свойства NumberOfCows.

Свойства часто объединяются с обычным объявлением полей. Это объявление для NumberOfCows.

Метод чтения вызывается каждый раз, когда свойство NumberOfCows нужно **прочитать**. В данном случае он возвращает значение закрытого свойства numberOfCows.

Метод записи вызывается при каждой **записи** в свойство NumberOfCows. Он имеет параметр **value**, содержащий значение, записываемое в поле.

```
private int numberOfCows;

public int NumberOfCows
{
    get
    {
        return numberOfCows;
    }
    set
    {
        numberOfCows = value;
        BagsOfFeed = numberOfCows * FeedMultiplier;
    }
}
```

Методы чтения и записи используют так же, как поля. Вот код для кнопки, которая задает количество коров, а в ответ получает количество мешков с кормом:

В этой строчке метод записи задает значение закрытого поля numberOfCows и тем самым обновляет открытое поле BagsOfFeed.

Так как метод записи NumberOfCows обновил поле BagsOfFeed, вы можете получить его значение.

```
private void button1_Click(object sender, EventArgs e) {
    Farmer myFarmer = new Farmer();
    myFarmer.NumberOfCows = 10;

    int howManyBags = myFarmer.BagsOfFeed;

    myFarmer.NumberOfCows = 20;
    howManyBags = myFarmer.BagsOfFeed;
}
```

Поле NumberOfCows запускает метод записи, передавая значение 20. Запрос к полю BagsOfFeed запускает метод чтения, возвращающий значение 20*30=600.

Приложение для проверки класса Farmer

Создайте новое приложение Windows Forms для проверки класса **`Farmer`** и его свойств. Для вывода результатов будет использован метод **`Console.WriteLine()`**.

Упражнение!

Будьте осторожны!

Консольный вывод в окне Output.

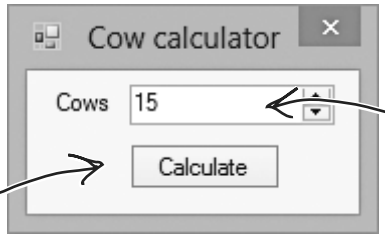
Когда в приложении Windows Forms результат выводится при помощи метода `Console.WriteLine()`, появляется окно Output. Обычно в приложениях WinForms консольный вывод не применяется, но мы воспользовались им как обучающим инструментом.

1 Добавьте к проекту класс `Farmer`:

```
class Farmer {
    public int BagsOfFeed;
    public const int FeedMultiplier = 30;

    private int numberOfCows;
    public int NumberOfCows {
        // (добавьте методы чтения и записи
        // с предыдущей страницы)
    }
}
```

2 Создайте форму:



Кнопка называется «вычислить» и использует открытые данные класса Farmer для вывода результата.

Присвойте параметрам Value, Minimum и Maximum элемента NumericUpDown значения 15, 5 и 300 соответственно.

3 Код формы использует метод `Console.WriteLine()` для отправки итоговых данных в **окно Output** (это окно вызывается также командой Output из меню Debug >> Windows). Методу `WriteLine()` можно передать несколько параметров, и первый — это выводимая строка. Включив в эту строку «`{0}`», вы выведете первый параметр, «`{1}`» — второй параметр, «`{2}`» — третий параметр и т. д.

```
public partial class Form1 : Form {
    Farmer farmer;
    public Form1() {
        InitializeComponent();
        farmer = new Farmer() { NumberOfCows = 15 };
    }
    private void numericUpDown1_ValueChanged(object sender, EventArgs e) {
        farmer.NumberOfCows = (int)numericUpDown1.Value;
    }
    private void calculate_Click(object sender, EventArgs e) {
        Console.WriteLine("I need {0} bags of feed for {1} cows",
            farmer.BagsOfFeed, farmer.NumberOfCows);
    }
}
```

Метод Console.WriteLine() отправляет строчку с текстом в окно Output.

Метод WriteLine () замещает "{0}" значением первого параметра, а "{1}" — значением второго параметра.

Не забудьте, что элементы управления следует «привязать» к обработчикам событий! Дважды щелкните на Button и NumericUpDown в конструкторе, чтобы IDE создала заглушки их методов-обработчиков события.

Автоматические свойства

Кажется, наш счетчик коров работает корректно. Запустите программу и щелкните на кнопке для проверки. Сделайте количество коров равным 30 и снова щелкните на кнопке. Повторите эту операцию для 5 коров, а потом для 20 коров. Вот что должно появиться в окне Output:

Если в IDE отсутсвует окно Output, откройте его командой меню View.

Вы понимаете, почему это стало причиной ошибки?

Но есть небольшая проблема. Добавьте к форме кнопку, которая выполняет оператор:

```
farmer.BagsOfFeed = 5;
```

Запустите программу. Все работает до нажатия новой кнопки. Попробуйте после этого нажать кнопку Calculate. Окажется, что 5 мешков корма требуется для любого количества коров! После редактирования параметра `NumericUpDown` кнопка Calculate снова начнет работать корректно.

Полностью инкапсулируем класс Farmer

Напечатав prop и дважды нажав tab, вы добавите к коду автоматическое свойство.

Проблема в том, что класс **не полностью инкапсулирован**. С помощью свойств мы инкапсулировали переменную `NumberOfCows`, но переменная `BagsOfFeed` до сих пор общедоступна. Это крайне распространенная проблема. Настолько распространенная, что в C# существует автоматическая процедура ее решения. Просто замените поле общего доступа `BagsOfFeed` автоматическим свойством:

1. Удалите поле `BagsOfFeed` из класса `Farmer`. Вместо него введите **`prop`** и дважды нажмите tab. Появится следующая строка кода:

```
public int MyProperty { get; set; }
```

2. Снова нажмите Tab, чтобы выделить поле `MyProperty`. Введите имя `BagsOfFeed`:

```
public int BagsOfFeed { get; set; }
```

Теперь у вас свойство вместо поля. Компилятор обрабатывает эту информацию как вспомогательное поле.

3. Впрочем, проблема еще не решена. Для ее решения сделайте свойство **доступным только для чтения**:

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

При попытке построить код вы получите сообщение об ошибке в строчке, задающей свойство `BagsOfFeed`: **метод записи недоступен**, — ведь вы не можете редактировать свойство `BagsOfFeed` вне класса `Farmer`. Удалите строчку кода, соответствующую второй кнопке. Теперь класс `Farmer` хорошо инкапсулирован!

Редактируем множитель feed

При построении счетчика коров множитель, указывающий количество корма на одну особь, мы определили как константу. Но представим, что нам требуется его изменить. Вы уже видели, как доступ к полям одного класса со стороны других классов может стать причиной ошибки. Именно поэтому **общий доступ к полям и методам имеет смысл оставлять только там, где это необходимо**. Так как программа никогда не обновляет `FeedMultiplier`, нам не требуется запись в это поле из других классов. Поэтому сделаем его свойством доступным только для чтения, которое использует вспомогательное поле.

1. Удалите строчку

```
public const int FeedMultiplier = 30;
```

Воспользуйтесь комбинацией prop-tab-tab, чтобы добавить свойство, доступное только для чтения. Но вместо автоматического свойства добавьте вспомогательное поле:

```
private int feedMultiplier;
public int FeedMultiplier { get { return feedMultiplier; } }
```

Так как вместо константы общего доступа у нас закрытое поле типа int, его имя теперь начинается со строчной буквы f.

Свойство возвращает вспомогательное поле feedMultiplier. Метод записи отсутствует, то есть оно доступно только для чтения. Метод чтения при этом открыт, то есть значение поля FeedMultiplier можно прочитать из любого другого класса.

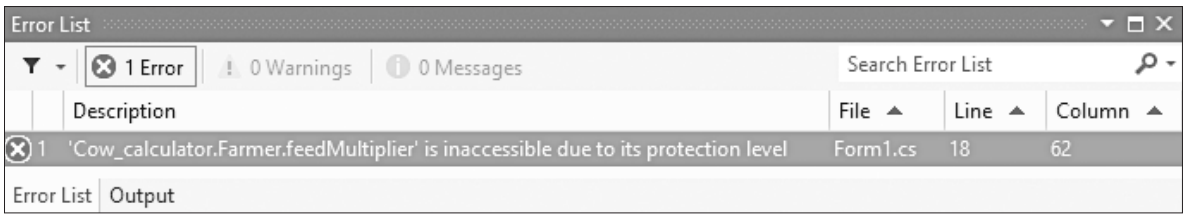
2. Запуск кода после внесения в него изменений покажет абсурдный результат. Свойство `BagsOfFeed` **всегда возвращает 0 мешков**.

Дело в том, что переменная `FeedMultiplier` не была инициализирована. Поэтому она по умолчанию имеет значение ноль. Добавим инициализатор объекта:

```
public Form1() {
    InitializeComponent();
    farmer = new Farmer() { NumberOfCows = 15, feedMultiplier = 30 };
```

Теперь **программа не компилируется**! Вот как выглядит сообщение об ошибке:

Проверьте окно Error List. В нем можно увидеть предупреждения, например, о том, что вы пытаетесь использовать переменную, которую забыли инициализировать.



Дело в том, что инициализатор объекта работает только с открытыми полями и свойствами. Что же делать, если требуется инициализировать закрытые поля?

Конструктор

Итак, вы уже убедились, что с закрытыми полями инициализатор объектов не работает. К счастью, существует особый метод, называемый **конструктором (constructor)**. Это **самый первый метод, который выполняется** при создании класса оператором `new`. Передавая конструктору параметры, вы указываете значения, которые требуется инициализировать. Но этот метод **не имеет возвращаемого значения**, так как напрямую не вызывается. Параметр передается оператору `new`. А как вы уже знаете, этот оператор возвращает объект, поэтому конструктору возвращать уже ничего не нужно.

Чтобы снабдить класс конструктором, добавьте метод, имеющий имя класса и не имеющий возвращаемого значения.

1 Добавление конструктора к классу Farmer

Требуется добавить всего две строчки кода, но как много они значат. Как вы помните, в классе должны присутствовать данные о количестве коров и мешков корма на одну корову. Добавим эту информацию к конструктору в качестве параметров. Для переменной `feedMultiplier` требуется начальное значение, так как она более не является константой.

Ключевое слово this в конструкции this.feedMultiplier указывает, что вы имеете в виду поле, а не одноименный параметр.

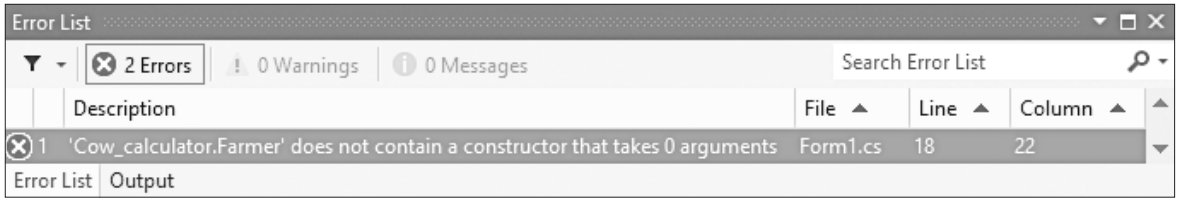
Отсутствие после public ключевых слов void или int либо других объявлений типа связано с тем, что конструктор не возвращает значения.

```
public Farmer(int numberOfCows, int feedMultiplier) {
    this.feedMultiplier = feedMultiplier;
    NumberOfCows = numberOfCows;
}
```

Перед вызовом метода записи NumberOfCows нужно задать параметр feedMultiplier.

Запись в закрытое поле numberOfCows исключает дальнейший вызов метода записи NumberOfCows. Данная же строка гарантирует его вызов.

Это сообщение об ошибке означает, что оператор new не имеет параметров.



2 Настройки формы, необходимые для работы с конструктором

Теперь нужно сделать так, чтобы оператор `new`, создающий объект `Farmer`, использовал конструктор вместо инициализатора объекта. После редактирования оператора `new` сообщения об ошибках исчезнут и код начнет компилироваться!

```
public Form1() {
    InitializeComponent();
    farmer = new Farmer(15, 30);
}
```

Так как форма — это тоже объект, для нее определен конструктор с именем Form1. Обратите внимание, что он не возвращает значение.

*Метод new, вызывающий конструктор, **отличается** наличием передаваемых конструктору параметров.*