Объектно-Ориентированное Программирование

Лекция 3

А.М. Задорожный

2016

Содержание

1. Повторное использование кода и Наследование
2. Статические методы и свойства
3. Определение операций

Контрольные вопросы

Что означают следующие термины:

1. Класс, объект?
2. Конструктор?
3. Поле данных?
4. Метод?
5. Квалификатор доступа?
6. Свойство?
7. Вычисляемое свойство?
8. Эластичность кода программы?
9. Инкапсуляция?

Повторное использование кода

Повторное использование кода – возможность использовать код написанный 1 раз во всех других местах, где он понадобится, а не включать его копию или аналог.

Например, если однажды был написан алгоритм сортировки целочисленного массива, то было бы очень желательно использовать именно этот код и для сортировки массива типа Double.

Повторное использование кода – один из механизмов повышения эффективности разработки программ!

Если в последующем потребуется улучшить алгоритм сортировки, то изменить придется только одну часть кода (которая используется в разных случаях), а не две или более, если для каждого типа массивов будут использоваться отдельные реализации алгоритма.

Наследование

В ООП имеется специальный механизм, расширяющий повторное использование кода – наследование.

Наследование в ООП позволяет создавать новые класса на основе уже существующих классов (расширять разработанные ранее классы).

Пусть кто-то разработал класс (Histogram). В другом проекте другим разработчикам понадобилось добавить в него свойство, например, для нахождения шага (ширины канала) гистограммы.

Наследование

Задача. Добавить в Histogram вычисляемое свойство, для шага гистограммы.

Текста класса Histogram у нас нет!

Вариант 1:

Самим разработать аналогичный класс и добавить в него нужное свойство.

Вариант 2:

Позвонить разработчику Histogram, попросить его прислать код класса и доработать его, добавив нужное свойство.

Наследование

ООП предлагает более эффективный вариант – разработать свой собственный класс на основе Histogram (унаследовав все возможности Histogram).

```
class ExtendedHist: Histogram
{
    ExtendedHist (string t, double a, double b, int N): base(t,a,b,N)
        {
        }
}
```

Это объявление можно (и нужно) понимать так:

Объекты класса ExtendedHist являются и объектами класса Histogram, но с некоторыми расширениями.

Применяем класс-потомок

```
{
    Random r = new Random();

    ExtendedHist h = new ExtendedHist("Вес ТС", 0, 10, 10);
    for(int i = 0; i < 1000; i++)
        h.Hist(10*r.NextDouble());
    h.Write();
    Console.WriteLine(h.MeanValue());
}
```

Расширение класса-предка

```
class ExtendedHist: Histogram
{
    ExtendedHist (string t, double a, double b, int N): base(t,a,b,N){
        }

        public double ChannelWidth
            {get {return (RightEdge - LeftEdge)/NChannels;}}
}
```

Нужное свойство добавлено!

Наш класс полностью использует код класса-предка, расширяя его лишь небольшой вставкой!

Наследование. Терминология

Класс **B**, производный от класса **A** называется классом-наследником, классом-потомком, дочерним классом или производным классом.

Говорят, что класс **B** наследует от класса **A**.

Класс **A** называется базовым классом, классом-предком или родительским классом для класса **B**.

По существу же, класс **B** является и классом **A**, но с некоторыми расширениями.

Контрольные вопросы

1. Что означает “Повторное использование кода”?
2. Что означает “Наследование” в ООП?
3. Как между собою связаны Наследование и Повторное использование кода?
4. Как в C# объявить класс **B**, который является потомком для класса **A**?

За сценой синтаксиса ООП

Когда мы пишем h.Write(), то метод Write выведет содержимое гистограммы h на консоль.

Конечно же, он каким-то “таинственным” образом получил данные объекта h.

На самом деле метод Write имеет параметр! Этот параметр имеет тип класса Histogram и имя **this**.

Write(Histogram **this**);

В качестве этого параметра и передается объект, указанный слева от метода (h).

h.Write() эквивалентно Write(h)!

Это относится и ко всем остальным методам. Их первым (скрытым) параметром является параметр с именем **this** и имеющим тип класса, метод которого рассматривается!

Как выполняется программа?

Выполняемая программа (exe-файл) – это набор данных и команд для компьютера. У каждой программы имеется стандартный заголовок, в котором указано, для какой системы написана программа, некоторая другая важная информация, в том числе адрес точки начала исполнения программы (адрес точки входа).

В C# компилятор в качестве такой точки начала программы (точки входа) указывает адрес метода **Main**. Этот метод автоматически создается в классе Program, когда создается новый проект.

Следствие. Метод **Main** должен быть в каждой программе и он должен быть один!

Операционная система:

- загружает выполняемую программу в память,
- читает заголовок программы и
- передает управление на точку входа.

С этого момента начинает работать наша программа.

C# полностью ООП язык

В C# все типы данных являются объектами.

int, double, … - это классы наследники от класса Object.

Если при объявлении класса не указывается класс-предок, то по умолчанию базовым классом будет класс Object.

Такая структура языка программирования приводит к еще одной загадке.

Что бы вызвать метод некоторого класса, нужно:

- Создать объект этого класса
- И вызвать метод для данного объекта.

Но, создавать объекты можно ТОЛЬКО внутри методов!

Как начать выполнение C# программы?

Напоминание. Что бы вызвать метод некоторого класса, нужно:

- Создать объект этого класса
- И вызвать метод для данного объекта.

Создавать объекты можно ТОЛЬКО внутри методов!

Как же можно вызвать метод **Main** класса **Program**? Ведь сначала нужно вызвать какой-то другой метод, который создаст объект класса **Program** и для него вызвать **Main**. Но что бы вызвать этот метод опять нужно создать объект…

Замкнутый круг?

Статические методы

Правило:

Что бы вызвать метод некоторого класса, нужно:

- *Создать объект этого класса*
- *И вызвать метод для данного объекта.*

не совсем точно!

Некоторые методы можно вызывать без создания объекта!

Console.WriteLine(), Math.Sin(x), …

Здесь слева стоит имя класса, а не имя объекта. Такие методы не имеют скрытого параметра **this**. (Соответственно, они не получают данных какого либо объекта)

Такие методы называются **статическими**.

Статические методы

Что бы объявить **статический** метод перед ним нужно указать слово **static**.

static void Main(string[] args)

Вот как объявлен метод **Main** в классе **Program**.

Так что **Main** можно вызывать не создавая объекта и парадокса нет!

Статическими объявляют методы, которым не нужно никаких других данных, кроме тех, что поступают через параметры.

Math.Sin(x) – чтобы вычислить синус нужно знать только значение x.

Статические методы

Иногда статические методы называют методами класса (они принадлежат классу).

Статические методы не имеют доступа к полям данных объекта.

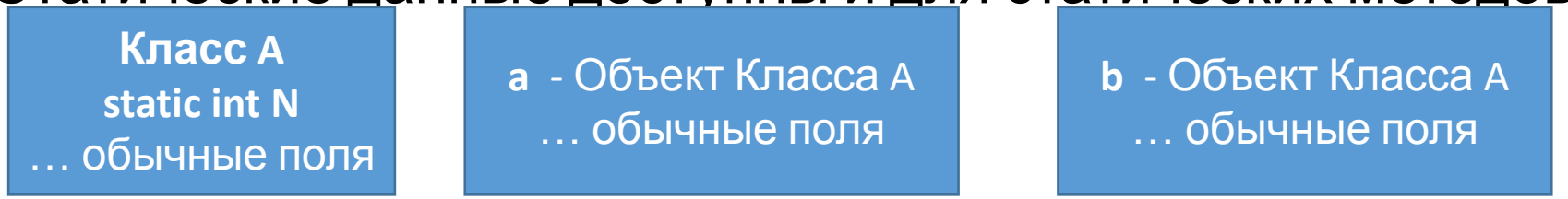
Обычные методов, в этом случае называют методами объектов.

Статические данные

Кроме статических методов можно объявлять и статические поля данных или свойства. Просто указать квалификатор **static**.

Такие данные являются общими для всех объектов. Например, можно объявить статический счетчик объектов и, тогда, если в конструкторе увеличивать этот счетчик, то в нем будет храниться количество созданных объектов (все объекты будут увеличивать один и тот же счетчик).

Статические данные доступны и для статических методов.



Контрольные вопросы

1. Каким образом методы получают данные объекта, для которого они вызываются?
2. Почему метод **Main** можно вызывать без создания объекта?
3. Как объявить статический метод?
4. Как вызвать статический метод?
5. В каких случаях имеет смысл делать метод статическим?
6. Что такое “статические данные”?

Объекты и операции

Что бы сделать использование разработанных классов удобным, в C# существует еще одна полезная возможность – переопределять операции над объектами.

Допустим мы разработали класс матриц, который реализует много полезных методов работы с матрицами. Но как сделать, чтобы сложение матриц было таким, как мы привыкли c=a+b?

```
class Matrix
{
        …
        static public Matrix operator+ (Matrix a, Matrix b)
        {
                Matrix c = new Matrix(…)
                for(int i …
                        for(int j …
                                c[i, j]= a[i, j]+b[i,j];
                return c;
        }
}
```

Объекты и операции

Теперь в программах для объектов класса Matrix можно использовать операцию сложения

```
Matrix c = a + b; // a и b – объекты класса Matrix
```

Аналогично можно определить и операции умножения, деления матриц и пр.
Тогда программы, работающие с матрицами будут выглядеть привычно. Работать с ними будет проще!

Переопределять можно:

!, ~, ++, --

+, -, *, /, %, &, |, ^, <<, >>

==, !=, <, >, <=, >=

&&, ||

[]

+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=

…

Контрольные вопросы

1. Зачем в C# предоставляется возможность переопределять операции над объектами?

2. Как переопределить ту или иную операцию в C#?