Основы типов

Тип определяет концепцию для значения. В данном примере мы использовали два литерала типа `int` со значениями 12 и 30. Мы также объявили *переменную* типа `int` с именем `x`.

Переменная определяет место хранения, которое в разные моменты времени может содержать разные значения. В противоположность этому *константа* всегда представляет одно и то же значение (более подробно мы поговорим об этом позднее).

Все значения в языке C# являются *экземплярами* конкретного типа. Смысл значения и набор возможных значений, которые может принимать переменная, определяются ее типом.

Примеры предопределенных типов

Предопределенные типы (они также называются *встроенными*) — это типы, которые специально поддерживаются компилятором. Тип `int` — это предопределенный тип, описывающий множество целых чисел, которые можно записать в памяти с помощью 32 битов в диапазоне от -2^{31} до $2^{31}-1$. К экземплярам типа `int` можно применять функции, например арифметические.

```
int x = 12 * 30;
```

Другим предопределенным типом в языке C# является тип `string`. Этот тип `string` представляет последо-

вательность символов, например ".NET" или "*http://oreilly.com*". К строкам также можно применять функции.

```
string message = "Hello world";
string upperMessage = message.ToUpper();
Console.WriteLine (upperMessage);     // HELLO WORLD

int x = 2012;
message = message + x.ToString();
Console.WriteLine (message);          // Hello world2012
```

Предопределенный тип `bool` имеет только два возможных значения: `true` и `false`. Тип `bool` широко используется в условных конструкциях с оператором `if`. Рассмотрим пример.

```
bool simpleVar = false;
if (simpleVar)
  Console.WriteLine ("This will not print");

int x = 5000;
bool lessThanAMile = x < 5280;
if (lessThanAMile)
  Console.WriteLine ("This will print");
```

ЗАМЕЧАНИЕ

Пространство имен `System` на платформе .NET Framework содержит много важных типов, которые не являются предопределенными в языке C# (например, `DateTime`).

Примеры пользовательских типов

Точно так же, как сложные функции состоят из простых функций, можно создавать сложные типы из элементарных типов. В данном примере мы определим пользовательский тип `UnitConverter` — класс, который будет служить схемой для преобразования единиц измерений.

```
using System;

public class UnitConverter
{
  int ratio;                              // Поле

  public UnitConverter (int unitRatio)    // Конструктор
  {
    ratio = unitRatio;
  }

  public int Convert (int unit)           // Метод
  {
    return unit * ratio;
  }
}

class Test
{
  static void Main()
  {
    UnitConverter feetToInches = new UnitConverter(12);
    UnitConverter milesToFeet = new UnitConverter(5280);

    Console.Write (feetToInches.Convert(30));    // 360
    Console.Write (feetToInches.Convert(100));   // 1200
    Console.Write (feetToInches.Convert
                    (milesToFeet.Convert(1)));   // 63360
  }
}
```

Член типа

Тип содержит *данные-члены* и *функции-члены*. Данные-члены класса `UnitConverter` представляют собой *поле* с именем `ratio`. Функциями-членами класса `UnitConverter` являются метод `Convert` и *конструктор* класса `UnitConverter`.

Симметрия между элементарными и пользовательскими типами

Прекрасным свойством языка C# является то, что между предопределенными и пользовательскими типами есть мало различий. Предопределенный тип `int` является схемой для описаний целых чисел. Он содержит данные — 32 бита — и функции-члены, использующие эти данные, например `ToString`. Аналогично наш пользовательский тип `UnitConverter` представляет собой схему для преобразований единиц измерения. Он содержит данные — коэффициент — и функции-члены для работы с этими данными.

Конструкторы и создание объектов

Данные создаются путем *создания объекта данного типа*. Объекты предопределенных типов можно создавать просто в виде литералов, например `12` или `"Hello, world"`.

Оператор `new` создает объекты пользовательского типа. В начале метода `Main` создаются два экземпляра типа `UnitConverter`. Сразу после того, как оператор `new` создаст объект, вызывается *конструктор* объекта для его инициализации. Конструктор определяется как

обычный метод, за исключением того, что его имя и тип возвращаемого значения должны совпадать с именем содержащего его класса.

```
public UnitConverter (int unitRatio)  // Конструктор
{
  ratio = unitRatio;
}
```

Экземпляр и статические члены

Данные-члены и функции-члены, действующие на *экземпляр* типа, называются *членами экземпляра*. Метод `Convert` класса `UnitConverter` и метод `ToString` типа `int` — это примеры членов экземпляра. По умолчанию члены класса являются членами экземпляра.

Данные-члены и функции-члены, действующие не на экземпляр типа, а на сам тип, называются статическими `static`. Примерами статических методов являются методы `Test.Main` и `Console.WriteLine`. Класс `Console` на самом деле является *статическим классом*, т.е. *все* его члены являются статическими. Экземпляр класса `Console` никогда не создается — все приложение использует одну консоль.

Для контракта со статическими методами отметим поле экземпляра `Name`, принадлежащее экземпляру типа `Panda`, в то время как поле `Population` принадлежит всем экземплярам типа `Panda`.

```
public class Panda
{
  public string Name;           // Поле экземпляра
  public static int Population; // Статическое поле
```

```
  public Panda (string n)      // Конструктор
  {
    Name = n;                  // Присваиваем поле экземпляра
    Population = Population+1; // Увеличиваем статическое поле
  }
}
```

Следующий код создает два экземпляра класса Panda, печатает их имена, а затем распечатывает всю популяцию:

```
Panda p1 = new Panda ("Pan Dee");
Panda p2 = new Panda ("Pan Dah");

Console.WriteLine (p1.Name);           // Pan Dee
Console.WriteLine (p2.Name);           // Pan Dah
Console.WriteLine (Panda.Population);  // 2
```

Ключевое слово public

Ключевое слово public открывает доступ к членам для функций-членов других классов. Если бы в данном примере поле Name в классе Panda не было *открытым*, то класс Test не имел бы к нему доступа. Ключевое слово public описывает способ общения типа: “Здесь находится то, что можно видеть другим типам, за исключением закрытых деталей моей реализации”. В терминах объектно-ориентированного программирования говорят, что класс *инкапсулирует* свои закрытые члены.

Преобразования

Язык C# может преобразовывать друг в друга экземпляры сравнимых типов. Преобразование всегда создает новое значение на основе существующего. Преобразования могут быть *неявными* или *явными*: неявные пре-

образования выполняются автоматически, а явные требуют *приведения*. В следующем примере мы *неявно* преобразовываем объект типа `int` в объект типа `long` (который в два раза больше по количеству битов) и *явно* приводим объект типа `int` в объект типа `short` (который в два раза меньше по количеству битов).

```
int x = 12345;     // int - это 32-битовое целое число
long y = x;        // Неявное преобразование в 64-битовое целое
short z = (short)x; // Явное преобразование в 16-битовое целое
```

В общем, неявные преобразования допускаются, когда компилятор может гарантировать отсутствие потери информации. В противном случае вы должны выполнить явное приведение к сравнимому типу.

Типы значений и ссылочные типы

Типы в языке C# разделяются на *типы значений* и *ссылочные типы* . *Типы значений* включают в себя большинство предопределенных типов (в частности, все числовые типы, тип `char` и тип `bool`), а также пользовательские типы `struct` и `enum`. *К ссылочным типам* относятся все классы, массивы, делегаты и интерфейсные типы.

Главным отличием между типами значений и ссылочными типами является их способ хранения в памяти.

Типы значений

Содержанием переменной или константы, имеющей тип значений, является обычное значение. Например, содержанием переменной предопределенного типа значений `int` являются 32 бита.

Пользовательский тип значений можно определить с помощью ключевого слова `struct` (рис. 1).

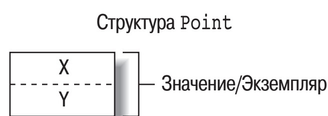

Рис. 1. Экземпляр типа значений в памяти

Присваивание экземпляра типа значений всегда означает *копирование* экземпляра. Рассмотрим пример.

```
Point p1 = new Point();
p1.X = 7;

Point p2 = p1;          // Присваивание вызывает копирование

Console.WriteLine (p1.X);   // 7
Console.WriteLine (p2.X);   // 7

p1.X = 9;                   // Изменяем p1.X
Console.WriteLine (p1.X);   // 9
Console.WriteLine (p2.X);   // 7
```

На рис. 2 показано, что экземпляры `p1` и `p2` хранятся независимо друг от друга.

Рис. 2. Присваивание копирует экземпляр типа значений

Ссылочные типы

Ссылочный тип сложнее, чем тип значений. Он состоит из двух частей: объекта и ссылки на этот объект. Содержанием переменной или константы ссылочного типа является ссылка на объект, содержащий значение. На рис. 3 тип `Point` из предыдущего примера переписан в виде класса.

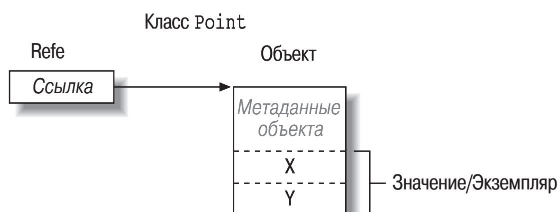

Рис. 3. Экземпляр ссылочного типа в памяти

Присваивание переменной ссылочного типа приводит к копированию ссылки, а не объекта. Это позволяет нескольким переменным ссылаться на один и тот же объект, что невозможно для типов значений. Если повторить предыдущий пример, сделав структуру `Point` классом, то операция над переменной `p1` повлияет на переменную `p2`.

```
Point p1 = new Point();
p1.X = 7;

Point p2 = p1;             // Копируем ссылку p1

Console.WriteLine (p1.X);  // 7
Console.WriteLine (p2.X);  // 7
```

```
p1.X = 9;                    // Изменяем p1.X
Console.WriteLine (p1.X);  // 9
Console.WriteLine (p2.X);  // 9
```

На рис. 4 показано, что переменные `p1` и `p2` являются ссылками, указывающими на один и тот же объект.

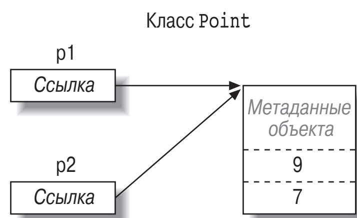

Рис. 4. Присваивание копирует ссылку

Литерал null

Ссылке можно присвоить литерал `null`. Это значит, что ссылка не ссылается ни на один объект. Предположим, что `Point` — это класс.

```
Point p = null;
Console.WriteLine (p == null); // True
```

Присвоение члену нулевой ссылки приводит к ошибке во время выполнения программы.

```
Console.WriteLine (p.X);       // NullReferenceException
```

И наоборот, переменная типа значений не может иметь значение `null`.

```
struct Point {...}
...
Point p = null; // Ошибка во время компиляции
int x = null;   // Ошибка во время компиляции
```

ЗАМЕЧАНИЕ

В языке C# для представления нулей с помощью типов значений есть специальная конструкция, которая называется *типами, допускающими значение null* (nullable types) (см. раздел "Нулевые типы").

Классификация предопределенных типов

Предопределенные типы в языке C# классифицируются следующим образом.

Типы значений

- Числовые
  - Целые со знаком (`sbyte`, `short`, `int`, `long`)
  - Целые без знака (`byte`, `ushort`, `uint`, `ulong`)
  - Действительное число (`float`, `double`, `decimal`)
- Логический (`bool`)
- Символьный (`char`)

Ссылочные типы

- Строка (`string`)
- Объект (`object`)

Предопределенные типы в языке C# являются синонимами типов Framework в пространстве имен `System`. Между следующими операторами существует только синтаксическая разница:

```
int i = 5;
System.Int32 i = 5;
```