Ниже перечислены операции, задействованные в примере программы:

```
. () * =
```

Точкой обозначается членство (или десятичная точка в числовых литералах). Круглые скобки применяются при объявлении или вызове метода; пустые круглые скобки используются, когда метод не принимает аргументов. Знак "равно" выполняет *присваивание*. (Двойной знак "равно", т.е. ==, производит сравнение эквивалентности.)

Комментарии

В C# поддерживаются два разных стиля документирования исходного кода: *однострочные комментарии* и *многострочные комментарии*. Однострочный комментарий начинается с двойной косой черты и продолжается до конца строки. Например:

```
int x = 3;  // Комментарий относительно присваивания
            // переменной x значения 3
```

Многострочный комментарий начинается с символов `/*` и заканчивается символами `*/`. Например:

```
int x = 3;  /* Это комментарий, который
               занимает две строки */
```

В комментарии могут быть встроены XML-дескрипторы документации (см. раздел "XML-документация" на стр. 209).

Основы типов

Тип определяет шаблон для значения. В рассматриваемом примере мы применяем два литерала типа `int` со значениями `12` и `30`. Мы также объявляем *переменную* типа `int` по имени `x`.

Переменная обозначает ячейку в памяти, которая с течением времени может содержать различные значения. В противоположность этому *константа* всегда представляет одно и то же значение (подробнее об этом — позже).

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

Примеры предопределенных типов

Предопределенные типы (также называемые встроенными типами) — это типы, которые специально поддерживаются компилятором. Тип `int` является предопределенным типом для представления набора целых чисел, которые умещаются в 32 бита памяти, от -2^{31} до $2^{31}-1$. С экземплярами типа `int` можно выполнять функции, например, арифметические:

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

Еще одним предопределенным типом в C# является `string`. Тип `string` представляет последовательность символов, такую как `".NET"` или `"http://oreilly.com"`. Со строками можно работать, вызывая для них функции следующим образом:

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

int x = 2015;
message = message + x.ToString();
Console.WriteLine (message);            // Hello
world2015
```

Предопределенный тип `bool` поддерживает в точности два возможных значения: `true` и `false`. Тип `bool` обычно используется для условного разветвления потока выполнения с помощью оператора `if`. Например:

```
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");
```

НА ЗАМЕТКУ!

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

Примеры специальных типов

Точно так же, как из простых функций можно строить сложные функции, из элементарных типов допускается создавать сложные типы. В следующем примере мы определим специальный тип по имени `UnitConverter` — класс, который служит шаблоном для преобразования единиц:

```
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
 }
}
```

Члены типа

Тип содержит *данные-члены* и *функции-члены*. Данными-членами типа `UnitConverter` является *поле* по имени `ratio`. Функции-члены типа `UnitConverter` — это метод `Convert` и *конструктор* `UnitConverter`.

Симметричность предопределенных и специальных типов

Привлекательный аспект языка C# заключается в том, что между предопределенными и специальными типами имеется

лишь несколько отличий. Предопределенный тип `int` служит шаблоном для целых чисел. Он содержит данные — 32 бита — и предоставляет функции-члены, использующие эти данные, такие как `ToString`. Аналогичным образом наш специальный тип `UnitConverter` действует в качестве шаблона для преобразований единиц. Он хранит данные — коэффициент (`ratio`) — и предоставляет функции-члены для работы с этими данными.

Конструкторы и создание экземпляров

Данные создаются путем *создания экземпляров* типа. Создавать экземпляры предопределенных типов можно просто за счет применения литерала наподобие `12` или `"Hello, world"`.

Операция `new` создает экземпляры специального типа. Наш метод `Main` начинается с создания двух экземпляров типа `UnitConverter`. Немедленно после создания объекта операцией `new` вызывается *конструктор* объекта для выполнения инициализации. Конструктор определяется подобно методу за исключением того, что вместо имени метода и возвращаемого типа указывается имя типа, которому конструктор принадлежит:

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

Члены экземпляра и статические члены

Данные-члены и функции-члены, которые оперируют на *экземпляре* типа, называются членами экземпляра. Примерами членов экземпляра могут служить метод `Convert` типа `UnitConverter` и метод `ToString` типа `int`. По умолчанию члены являются членами экземпляра.

Данные-члены и функции-члены, которые не оперируют на экземпляре типа, а вместо этого имеют дело с самим типом, должны помечаться как `static`. Примерами статических методов являются `Test.Main` и `Console.WriteLine`. Класс `Console` в действительности представляет собой *статический класс*, а это значит, что *все* его члены являются статическими. Создавать экземпляры класса `Console` никогда не придется — одна консоль совместно используется всем приложением.

Чтобы понять отличия между членами экземпляра и статическими членами, рассмотрим следующий код, в котором поле

экземпляра Name относится к конкретному экземпляру Panda, тогда как поле Population принадлежит набору всех экземпляров класса Panda:

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

  public Panda (string n)       // Конструктор
  {
    Name = n;     // Присвоить значение полю экземпляра
    Population = Population+1;
        // Инкрементировать значение статического поля
  }
}
```

В показанном ниже коде создаются два экземпляра Panda, выводятся их имена (поле Name) и затем общее количество (поле Population):

```
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 со стороны других классов. Если бы в рассматриваемом примере поле Name класса Panda не было помечено как public, то класс Test не смог бы получить к нему доступ. Маркировка члена как открытого (public) означает, что тип позволяет его видеть другим типам, а все остальное будет относиться к закрытым деталям реализации. Согласно объектно-ориентированной терминологии, говорят, что открытые члены *инкапсулируют* закрытые члены класса.

Преобразования

В C# возможны преобразования между экземплярами совместимых типов. Преобразование всегда приводит к созданию нового значения из существующего. Преобразования могут быть либо *неявными*, либо *явными*: неявные преобразования происходят автоматически, в то время как явные преобразования требуют *приведения*. В следующем примере мы *неявно* преобразуем int

в тип `long` (который имеет в два раза больше битов, чем `int`) и *явно* приводим `int` к типу `short` (который имеет вполовину меньше битов, чем `int`):

```
int x = 12345;       // int - это 32-битное целое
long y = x; // Неявное преобразование в 64-битное целое
short z = (short)x; // Явное приведение к 16-битному
целому
```

В общем случае неявные преобразования разрешены, когда компилятор может гарантировать, что они всегда будут проходить успешно без потери информации. В других обстоятельствах для преобразования между совместимыми типами должно выполняться явное приведение.

Типы значений и ссылочные типы

Типы в C# можно разделить на *типы значений* и *ссылочные типы*.

Типы значений включают большинство встроенных типов (а именно — все числовые типы, тип `char` и тип `bool`), а также специальные типы `struct` и `enum`. *Ссылочные типы* включают все классы, массивы, делегаты и интерфейсы.

Фундаментальное отличие между типами значений и ссылочными типами связано с тем, как они поддерживаются в памяти.

Типы значений

Содержимым переменной или константы, относящейся к *типу значения*, является просто значение. Например, содержимое встроенного типа значения `int` — это 32 бита данных.

С помощью ключевого слова `struct` можно определить специальный тип значения (рис. 1):

```
public struct Point { public int X, Y; }
```

Рис. 1. *Экземпляр типа значения в памяти*

Присваивание экземпляра типа значения всегда приводит к *копированию* этого экземпляра. Например:

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

Point p2 = p1;   // Присваивание приводит к копированию

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

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

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

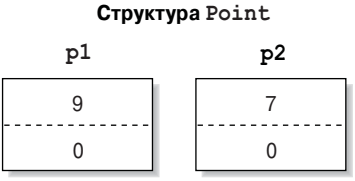

Рис. 2. *Присваивание копирует экземпляр типа значения*

Ссылочные типы

Ссылочный тип сложнее типа значения из-за наличия двух частей: *объекта* и *ссылки* на этот объект. Содержимым переменной или константы ссылочного типа является ссылка на объект, который содержит значение. Ниже приведен тип `Point` из предыдущего примера, переписанный в виде класса (рис. 3):

```
public class Point { public int X, Y; }
```

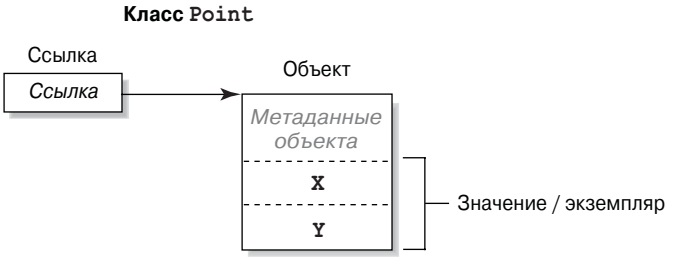

Рис. 3. *Экземпляр ссылочного типа в памяти*

Присваивание переменной ссылочного типа вызывает копирование ссылки, но не экземпляра объекта. Это позволяет множеству переменных ссылаться на один и тот же объект — то, что обычно невозможно с типами значений. Если повторить предыдущий пример при условии, что `Point` теперь является классом, операция над `p1` будет воздействовать на `p2`:

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

Point p2 = p1;                  // Копирует ссылку на p1

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

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

На рис. 4 видно, что `p1` и `p2` — это две ссылки, которые указывают на один и тот же объект.

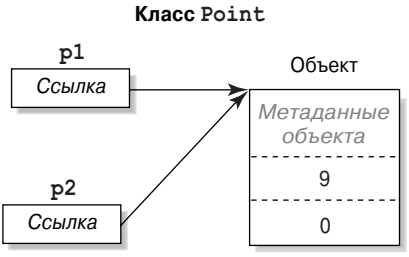

Рис. 2.4. *Присваивание копирует ссылку*

Значение `null`

Ссылке может быть присвоен литерал `null`, который отражает тот факт, что ссылка не указывает на какой-либо объект. Предположим, что `Point` является классом:

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

Обращение к члену ссылки `null` приводит к возникновению ошибки времени выполнения:

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

В противоположность этому тип значения обычно не может иметь значение null:

```
struct Point {...}
...
Point p = null;  // Ошибка на этапе компиляции
int x = null;    // Ошибка на этапе компиляции
```

НА ЗАМЕТКУ!

В C# имеется специальная конструкция под названием *типы, допускающие значение null*, которая предназначена для представления null в типах значений (см. раздел "Типы, допускающие значение null" на стр. 144).

Классификация предопределенных типов

Предопределенные типы в C# классифицируются следующим образом.

Типы значений

- Числовой
  - Целочисленный со знаком (sbyte, short, int, long)
  - Целочисленный без знака (byte, ushort, uint, ulong)
  - Вещественный (float, double, decimal)
- Логический (bool)
- Символьный (char)

Ссылочные типы

- Строка (string)
- Объект (object)

Предопределенные типы C# являются псевдонимами типов .NET Framework в пространстве имен System. Показанные ниже два оператора отличаются только синтаксисом:

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

Набор предопределенных типов *значений*, исключая decimal, в общеязыковой исполняющей среде (Common Language

Runtime — CLR) известен как *примитивные типы*. Примитивные типы называются так потому, что они поддерживаются непосредственно через инструкции в скомпилированном коде, которые обычно транслируются в прямую поддержку внутри имеющегося процессора.

Числовые типы

Ниже показаны предопределенные числовые типы в C#.

Тип C#	Системный тип	Суффикс	Размер в битах	Диапазон
Целочисленный со знаком				
`sbyte`	`SByte`		8	-2^7 — 2^7-1
`short`	`Int16`		16	-2^{15} — $2^{15}-1$
`int`	`Int32`		32	-2^{31} — $2^{31}-1$
`long`	`Int64`	`L`	64	-2^{63} — $2^{63}-1$
Целочисленный без знака				
`byte`	`Byte`		8	0 — 2^8-1
`ushort`	`UInt16`		16	0 — $2^{16}-1$
`uint`	`UInt32`	`U`	32	0 — $2^{32}-1$
`ulong`	`UInt64`	`UL`	64	0 — $2^{64}-1$
Вещественный				
`float`	`Single`	`F`	32	$\pm(\sim10^{-45}$ — $10^{38})$
`double`	`Double`	`D`	64	$\pm(\sim10^{-324}$ — $10^{308})$
`decimal`	`Decimal`	`M`	128	$\pm(\sim10^{-28}$ — $10^{28})$

Из всех *целочисленных* типов `int` и `long` являются первоклассными типами, которым обеспечивается поддержка как в языке C#, так и в исполняющей среде. Другие целочисленные типы обычно применяются для реализации взаимодействия или когда главная задача связана с эффективностью хранения.

В рамках вещественных числовых типов `float` и `double` называются *типами с плавающей точкой* и обычно используются в научных и графических вычислениях. Тип `decimal`, как пра-

вило, применяется в финансовых вычислениях, при которых требуется десятичная арифметика и высокая точность. (Формально `decimal` также является типом с плавающей точкой, хотя обычно на него так не ссылаются.)

Числовые литералы

Целочисленные литералы могут использовать десятичную или шестнадцатеричную форму записи; шестнадцатеричная форма записи предусматривает применение префикса `0x` (например, `0x7f` эквивалентно `127`). *Вещественные литералы* могут использовать десятичную и/или экспоненциальную форму записи, такую как `1E06`.

Выведение типа числового литерала

По умолчанию компилятор *выводит* тип числового литерала, относя его либо к `double`, либо к какому-то целочисленному типу.

- Если литерал содержит десятичную точку или символ экспоненты (`E`), то он получает тип `double`.
- В противном случае типом литерала будет первый тип, в который может уместиться значение литерала, из следующего списка: `int`, `uint`, `long` и `ulong`.

Например:

```
Console.Write(      1.0.GetType()); //Double (double)
Console.Write(     1E06.GetType()); //Double (double)
Console.Write(        1.GetType()); //Int32  (int)
Console.Write(0xF0000000.GetType()); //UInt32 (uint)
Console.Write(0x100000000.GetType()); // Int64  (long)
```

Числовые суффиксы

Числовые суффиксы, указанные в предыдущей таблице, явно определяют тип литерала:

```
decimal d = 3.5M; //M = decimal (не чувствителен к регистру)
```

Необходимость в суффиксах `U` и `L` возникает редко, поскольку типы `uint`, `long` и `ulong` могут почти всегда либо *выводиться*, либо *неявно преобразовываться* из `int`:

```
long i = 5;    // Неявное преобразование из int в long
```