Высокоуровневые методы информатики и программирования

Лекция 3

План лекции

1. Переменные программы
2. Операции
3. Выражения
4. Операторы

Основные понятия

- Программа это множество типов.
- Типы - встроенные базовые типы и пользовательские типы (описанные программистами): классы, интерфейсы, … .
- Классы состоят из данных (константы, переменные) и методов (свойства, функции).
- Методы это данные + набор операторов (алгоритм).
- Операторы это элементарные высказывания языка программирования. Операторы включают: ключевые слова, переменные и выражения.
- Выражения это переменные (константы) объединенные знаками операций.

Логика рассмотрения языка C#

1. Переменные и константы.

```
int n;
float val;
float r;
```

2. Типы переменных и констант.

3. Операции над переменными.

```
(n + val) * 2
```

4. Операторы языка.

```
r = (n + val) * 2;
```

```
float calc(int m)
{
  int n;
  f loat val;
  float r;
  r = (n + val) * 2;
 return r;
}
```

5. Методы.

6. Классы.

```
class Circle
{
  int r;
  float calc(int m)
  {
    …
  }
}
```

1. Переменные и типы

Переменные программы

- *Переменные* – это именованные участки памяти, которые могут хранить:
  - значения некоторого типа (для значащих типов, в стеке),
  - ссылки на экземпляры некоторых классов (для ссылочных типов, ссылки на объекты, расположенные в "куче").
- В C# выделяют два типа переменных:
  - *поля классов* (объявляются в описаниях классов и создаются для каждого объекта) и
  - *локальные переменные методов* (создаются при каждом вызове метода класса).

Виды переменных по области видимости

- Уровня класса (статические переменные класса)
  - Доступ с помощью имени класса
  - Время жизни – время работы программы
  - Доступ из всех классов (если public)
- Уровня объекта класса (поля)
  - Доступ с помощью ссылки на объект
  - Время жизни – от new до удаления ссылок
  - Доступ в методах класса (если public то из других классов)
- Уровня метода (локальные переменные)
  - Доступ по имени
  - Время жизни – выполнения метода
  - Доступны только в методе после объявления

Объявление переменных

- Прежде, чем переменная может быть использована, она должна быть объявлена. Объявление переменных можно делать в любом месте программы.
- При объявлении переменных задается:
  - имя (идентификаторы)
    - Должно начинаться с буквы или подчерка (_).
    - Буква может быть из любого алфавита (unicode)
    - Количество символов не ограничено.
  - тип (встроенный или пользовательский)
  - могут быть заданы модификаторы
    - режим доступа,
    - возможность изменения,
    - сохранность значений.
- Формат объявление переменных:

  <тип> <имя>;
  <тип> <имя> [= <значение>]
  [<модификаторы>] <тип> <имя> [= <значение>];
  где [<модификаторы>] = {<режим доступа>, static, const}.

- Например:

```
public int x = 5;
public static const int n=10;
```

Константы

- В C# *константы* могут задаваться в виде
  - литералов (набора символов) или
  - именованных *констант*.
- Например:

  y = 7.7;
- В этом примере значение *константы* "7.7" является одновременно ее именем, она имеет и тип. Константы с дробной частью по умолчанию имеют тип double.
- Для точного указания некоторых типов можно задавать символ, стоящий после литерала (в верхнем или нижнем регистре). Такими символами могут быть: f – тип float; d – тип double; m – тип decimal.
- Также можно объявить именованную *константу*. Для этого в объявление переменной добавляется модификатор const, *инициализация констант* обязательна и не может быть отложена.
- Например:

  const float c = 0.1f;

Строковые константы

- Под строковыми константами понимается последовательность символов заключенных в двойные кавычки.
  - Например: “Петров С.А.”
- В C# существуют два вида строковых констант:
  - обычные константы, которые представляют строку символов, заключенную в двойные кавычки – "ssss";
  - *@-константы*, заданные обычной константой c предшествующим знаком @.
- В обычных константах некоторые символы интерпретируются особым образом. Это требуется, для задания в строке специальных управляющих символов, в виде escape-последовательностей.
- Например:
  - "\n" - символ перехода на новую строку;
  - "\t" - символ табуляции (отступ на заданное количество символов);
  - "\\" - символ обратной косой черты;
  - "\"" - символ кавычки, вставляемый в строку, но не сигнализирующий о ее окончании.

Строковые константы

- Часто при задании констант, определяющих путь к файлу, приходится каждый раз удваивать символ обратной косой черты: “C:\\test.txt”, что не совсем удобно.
- В этом случае и используются *@-константы*, в которых все символы понимаются в полном соответствии с их изображением.
- Например, две следующие строки будут аналогичными:

```
s1 = "c:\\c#book\\ch5\\chapter5.doc";
s2 = @"c:\c#book\ch5\chapter5.doc";
```

Время жизни переменных

- Переменные появляются (рождаются)
  - не статические
    - переменные методов появляются в результате их объявления.
    - переменные классов (поля) появляются в результате создания объекта класса.
  - статические переменные создаются при запуске программы.
- Переменные исчезают (умирают)
  - не статические
    - Переменные методов после закрытия блока в котором они объявлены (}).
    - Переменные класса после уничтожения объекта
  - статические уничтожаются после завершения программы.

Области видимости переменных

- Область видимости переменной (variable scope) это участок программы, в котором переменную можно использовать.
- В общем случае областью видимости локальной переменной является участок программы, от строки, в которой она объявляется, до первой фигурной скобки, завершающей блок или метод, в котором переменная объявлена.
- Областью видимости локальных переменных, которые объявляются в операторах цикла (например, for или while), является содержание (тело) данного цикла.
- Например:

```
public void ScopeTest() {
   int n = 0;
   for (int i = 0; i < 10; i++) {
     Console.WriteLine(i);
   } // i выходит из области видимости и удаляется
   // можно объявить другу переменную с именем i
   {  // начало блока
     var i = ”другой цикл”; // строка
     Console.WriteLine(i);
   } // i опять выходит из области видимости
} // переменная n тоже выходит из области видимости
```

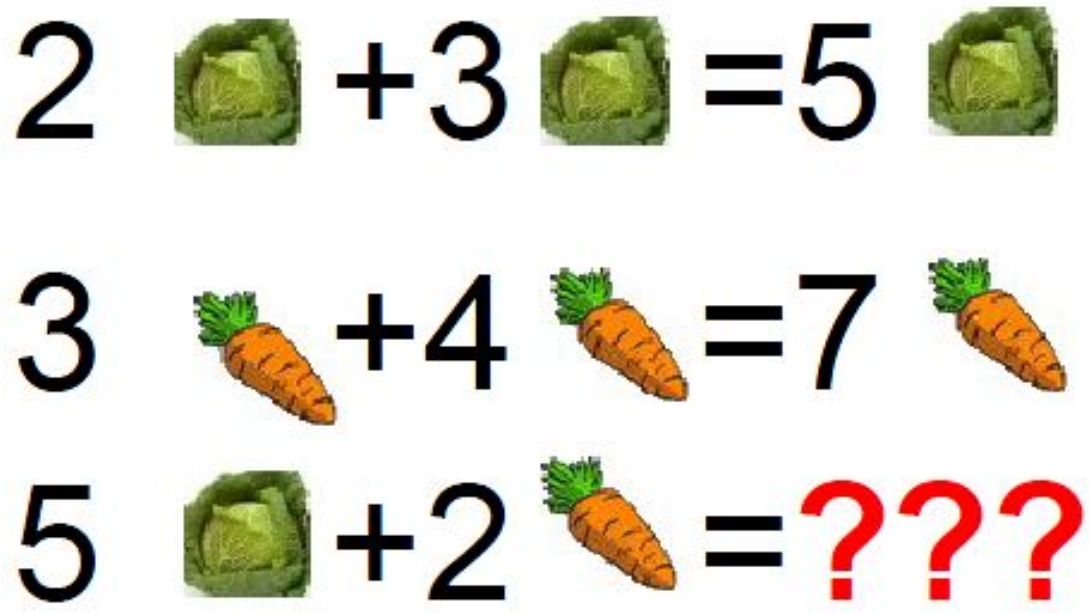
Тип данных

- Язык C# является строго типизированным языком. Это означает, что все данные (константы и переменные) программы имеют явно или неявно заданный ***тип***.

- Тип данных определяет:
  - количество используемой памяти (в байтах);
  - набор операции, в которых может участвовать данные такого типа;
  - способы явного и неявного преобразования в другие типы.

Зачем нужны типы данных?

Чтобы гарантировать осмысленность выполняемых операций:



Основные понятия

- Программа это набор типов

$P = \{T1, T2, \ldots, Tn\}$

- Тип задает:
  - Количество памяти
  - Состояние (данные)
  - Поведение (методы)
  - Операции в которых может участвовать
  - Преобразования к другим типам.

Основные сведения о типах

- Все элементы программы имеют тип *(переменные, константы, выражения, методы, параметры методов, и т.п.)*
- Для всех переменных требуется объявлять тип.
- Результат вычисления выражения имеет определенный тип.
- Переменные и выражения должны иметь один и тот же тип при присвоении.
- Если типы разные, выполняется их преобразование:
  - неявное (без прямого указания программиста)
  - явное (в результате заданного преобразования, кастинг)

Система типов данных на языке C#

- Все типы языка C# можно разделить на две большие группы:
  - встроенные типы и
  - типы, описываемые разработчиками.
- ***Встроенные (или фундаментальные) типы*** изначально принадлежат базисной *системе типов*, которая поддерживается средой CLR, но которые могут иметь специальные имена в конкретном языке. В соответствии со стандартом типов (Common Type Standard – CTS) в .Net имеется 15 встроенных типов (см. табл. 3.1).
- ***Типы, описываемые разработчиками.*** Кроме встроенных типов, которые предоставляются в языке C#, программист может описывать и использовать свои собственные (пользовательские) типы. Имеются следующие пользовательские типы:
  - Классы (class)
  - Структуры (struct)
  - Перечисления (enum)
  - Интерфейсы (interface)
  - Делегаты (delegate)

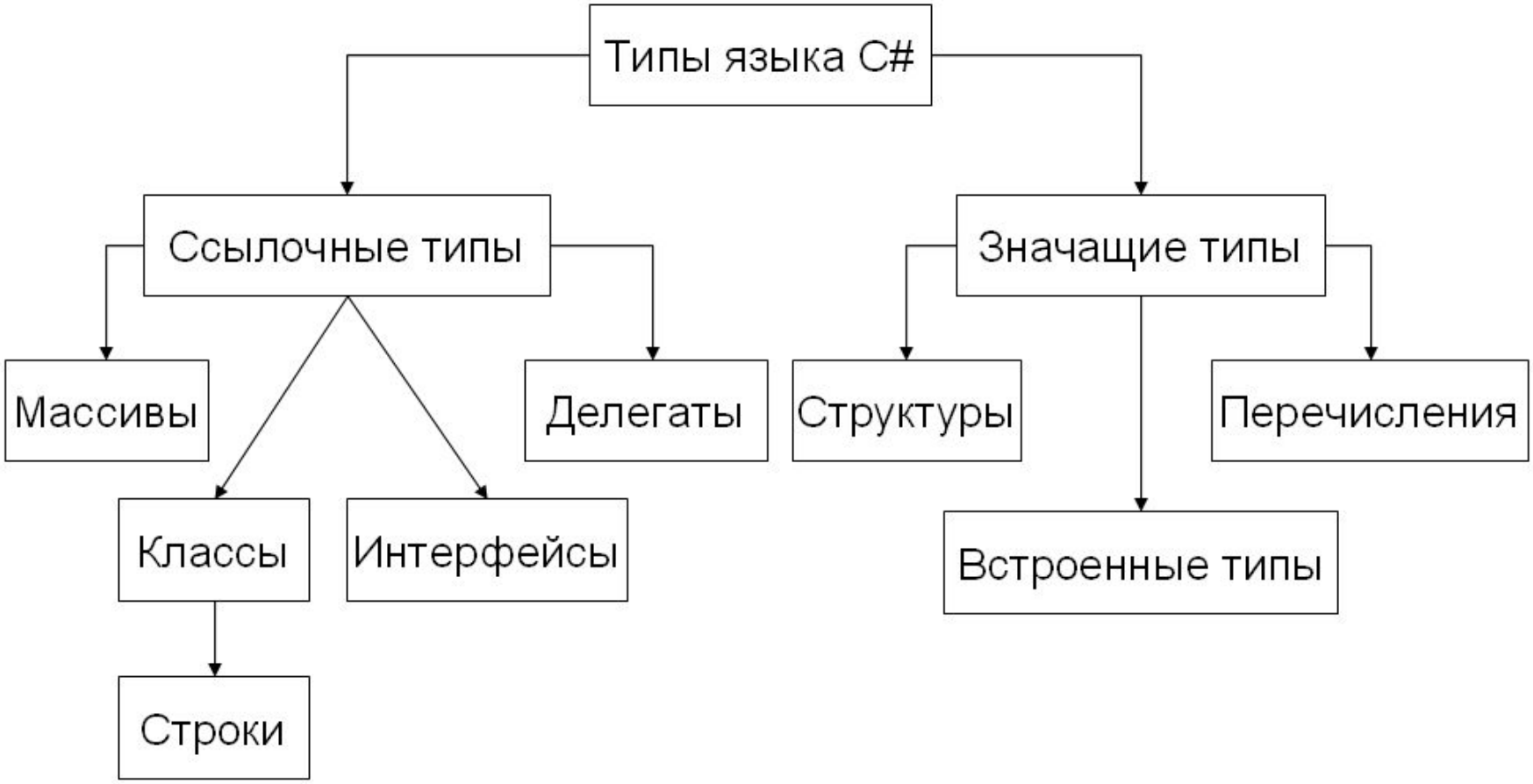
Пользовательские типы

- Пользовательские типы создаются с помощью объявлений типов, которые включают следующую информацию:
  - вид создаваемого типа (один из выше перечисленных);
  - имя нового типа;
  - объявление (имя и тип) каждого элемента данного типа.
- После того, как тип объявлен, можно создавать и использовать объекты данного типа, точно так же, как если бы они были встроенными типами.

Основные виды типов

- ***Значащие типы***
  - создаются в стеке
  - автоматически уничтожаются
- ***Ссылочные типы***
  - создаются в 2 шага
    - сперва объявляются – выделяется память в стеке для хранения адреса
    - затем выделяется память в куче с помощью оператора new

Структура типов языка C#



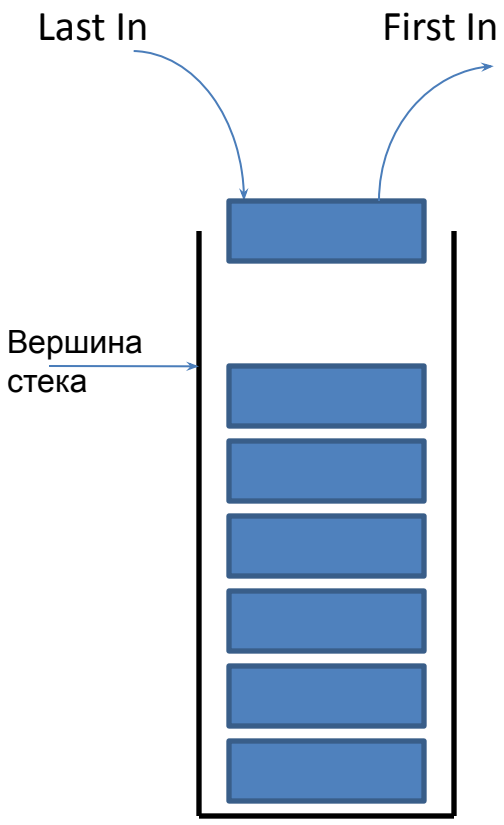
Хранение данных программы

Данные используемые программой (переменные, константы) могут храниться в в двух типах оперативной памяти:

- Стек (линейная память)
- Куча (динамическая память)

Стек (stack)

- Стек – это линейный участок памяти (массив), который действует как структура данных типа «Последним пришел – первым ушел» (last-in, first-out – LIFO).
- Основной особенностью стека являются то, что данные могут добавляться только к вершине стека и удаляться из вершены.
  - Добавление и удаление данных из произвольного места стека невозможно.
- Операции по добавлению и удаление элементов из стека выполняются очень быстро.
  - Однако размеры стека, как правило, ограничены, и время хранения данных зависит от времени жизни переменной.
- Для всех локальных переменных методов и передаваемых методам параметров память выделяется в вершине стека.
- После того, как методы заканчивают работу вся выделенная память в стеке для их переменных автоматически освобождается.

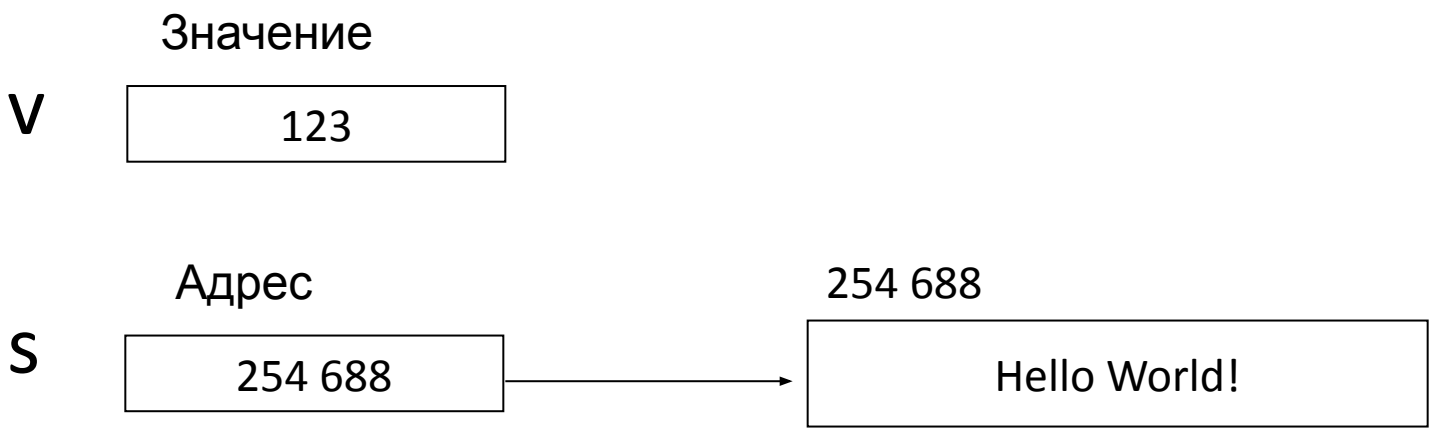


Куча (heap)

- **Куча (heap)** – это область оперативной памяти, в разных частях которой по запросу программы, операционная система может выделять участки требуемого размера для хранения объектов классов.
- Память в куче выделяется с помощью операции new.
- В отличие от стека, участки памяти в "куче" могут выделяться и освобождаться в любом порядке.
- Хотя программа может хранить элементы данных в "куче", она не может явно удалять их из нее.
- Вместо этого компонент среды CLR, называемый Garbage Collector (GC), автоматически удаляет не используемые участки "кучи", когда он определит, что код программы уже не имеет доступа к ним (не хранит их адреса).
- Автоматическая сборка мусора освобождает программиста от необходимости освобождать не используемую память вручную, что часто приводит к ошибкам работы программы.

Различие между значащими и ссылочными типами

```
int v = 123;
string s;
s = “Hello World!”;
```

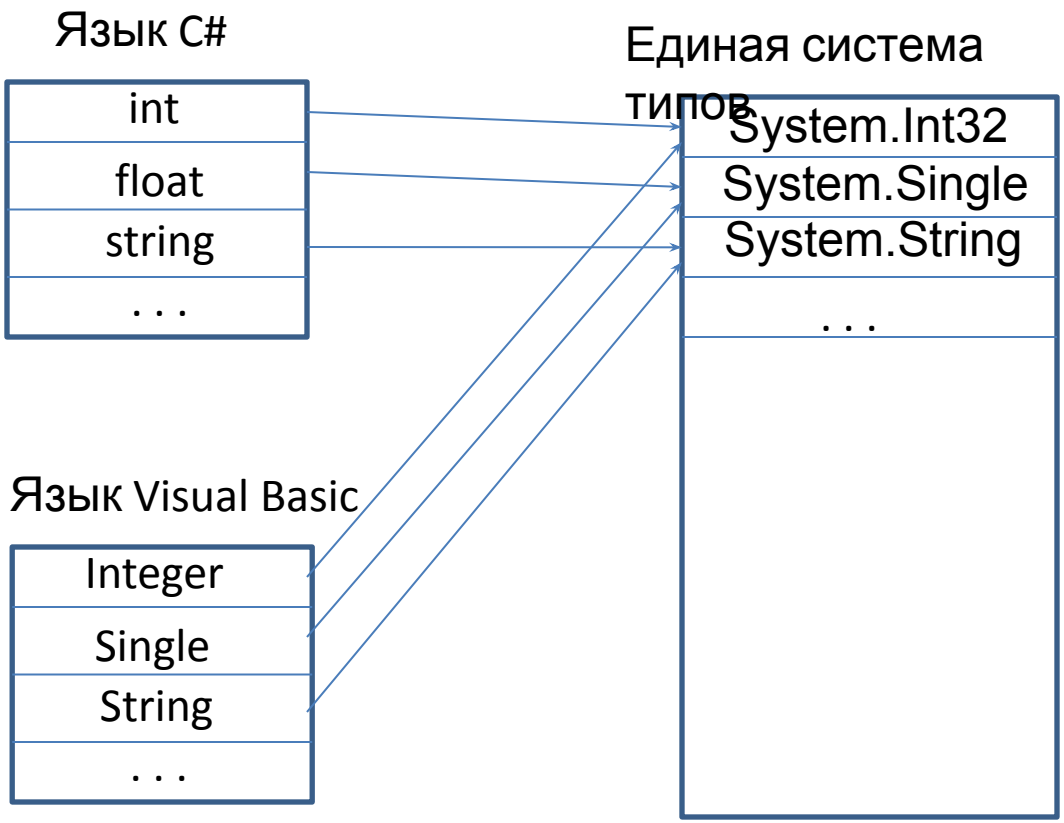


Память программы - стек и куча

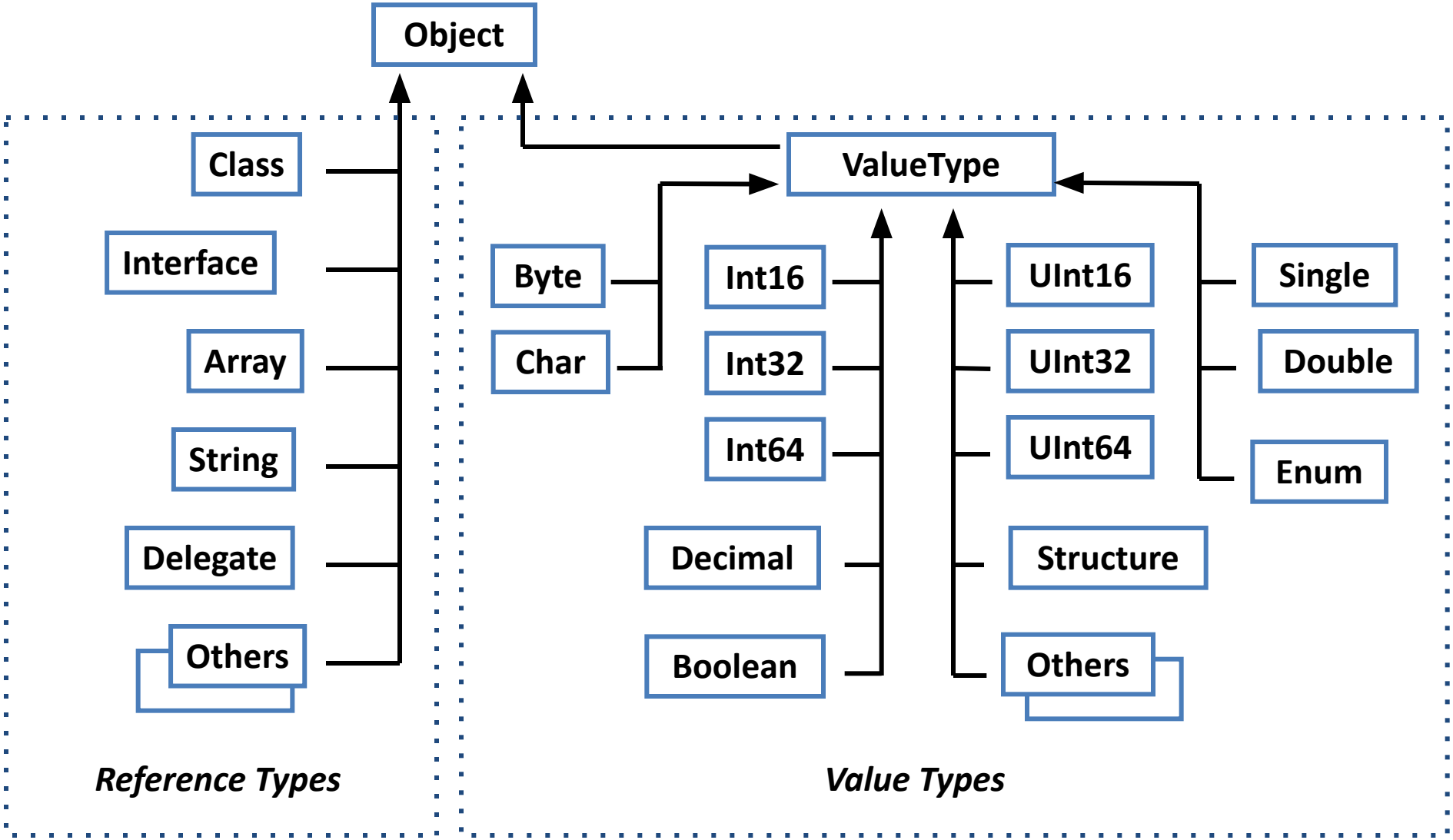
Системные типы данных CLR

- В .Net Framework есть общие для всех языков, системные встроенные типы.
- *Общая системы типов* Common Type System (CTS) для всех языков.
- Описание этих типов выполнено специалистами компании Microsoft.
  - Например:
    - System.Int32
    - System.Single
    - System.String
    - ….

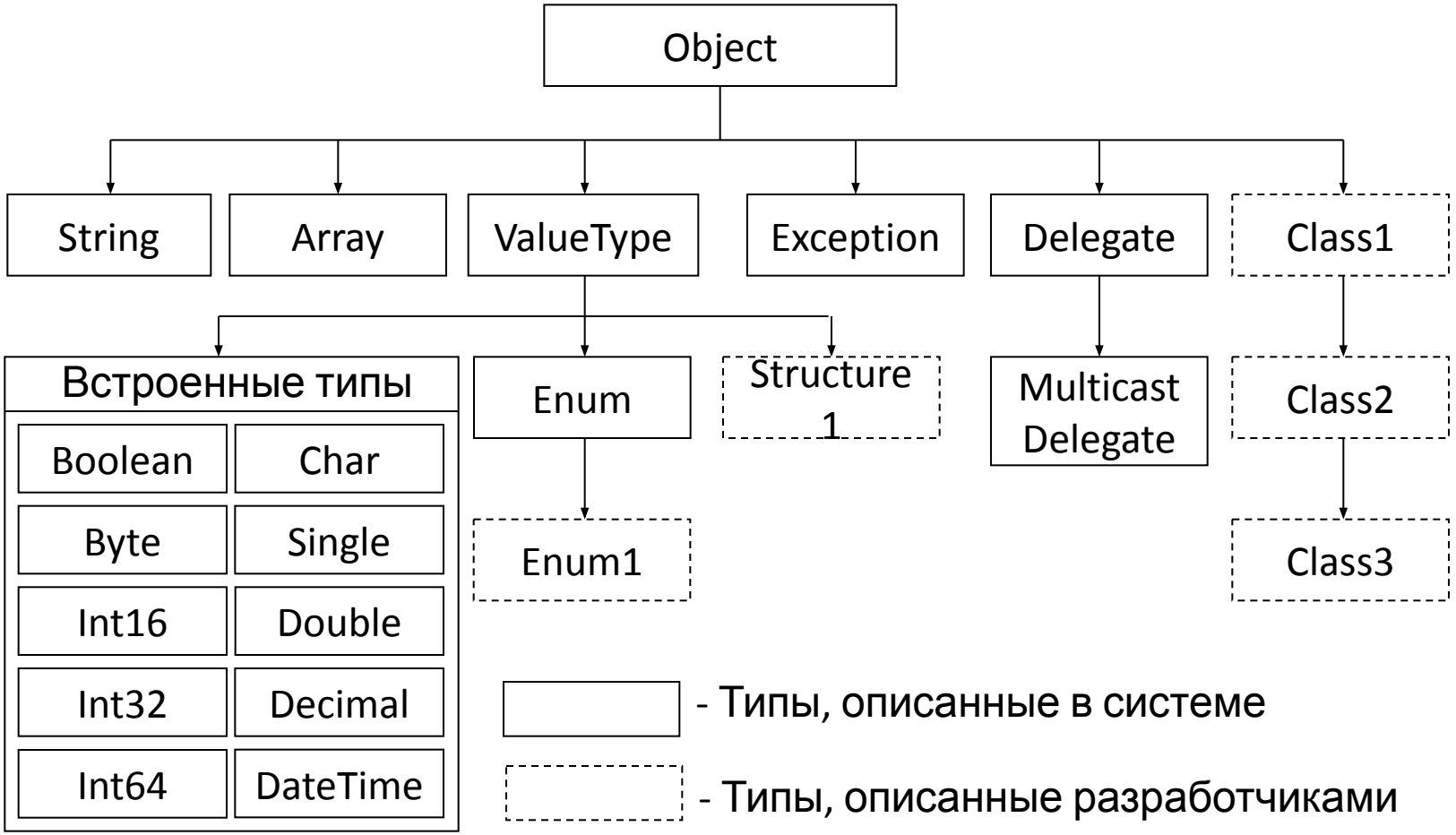
Соответствие встроенных типов и системных типов



Типы определенные в CLR



Наследование типов в CLR



Методы класса System.Object

- Equals() - виртуальный метод, возвращающий true если значения объектов совпадают (по умолчанию, если два объекта расположены в одном месте памяти).
- GetHashCode() - виртуальный метод, возвращает некоторое целое число (***хэш-код***), однозначно идентифицирующее экземпляр класса.
- GetType() - возвращает объект типа Type, описывающий соответствующий тип.
- ToString() - виртуальный метод, возвращающий символьное представление значения переменной (по умолчанию возвращает строку, представляющую полное имя типа объекта).

- Классы-потомки при создании наследует все свойства и методы родительского класса Object. Естественно, что все *встроенные типы* нужным образом переопределяют *методы* родителя и добавляют собственные *поля*, *свойства* и *методы*.
- Учитывая, что и типы, создаваемые пользователем, также являются *потомками класса* Object, то для них необходимо переопределить *методы* родителя, если предполагается использование этих *методов*; реализация родителя, предоставляемая по умолчанию, не обеспечивает нужного эффекта.
- Пример объявления переменных и присваивания им значений в языке C# показан ниже:

```
int x=11;
int v = new Int32();
v = 007;
string s1 = "Agent";
s1 = s1 + v.ToString() + x.ToString();
```

Встроенные типы

Имя типа	Системный тип CLR	Значения - диапазон	Размер – точность
Логический тип			
`bool`	`System.Boolean`	`true, false`	8 бит
Арифметические целочисленные типы			
`sbyte`	`System.SByte`	-128 ÷ +127	Знаковое, 8 бит
`byte`	`System.Byte`	0 ÷ 255	Беззнаковое, 8 бит
`short`	`System.Short`	-32768 ÷ +32767	Знаковое, 16 бит
`ushort`	`System.UShort`	0 ÷ 65535	Беззнаковое, 16 бит
`int`	`System.Int32`	-2,147,483,648 ÷ +2,147,483,647	Знаковое, 32 бит
`uint`	`System.UInt32`	$\approx(0 \div 4*10^{9})$	Беззнаковое, 32 бит
`long`	`System.Int64`	$\approx(-9*10^{18} \div 9*10^{18})$	Знаковое, 64 бит
`ulong`	`System.UInt64`	$\approx(0 \div 18*10^{18})$	Беззнаковое, 64 бит

Встроенные типы (продолжение)

Имя типа	Системный тип CLR	Значения - диапазон	Размер - точность
Арифметический тип с плавающей точкой			
`float`	`System.Single`	$\pm(1.5*10^{-45} \div 3.4*10^{+38})$	32 бита (точность 7 цифр)
`double`	`System.Double`	$\pm(5.0*10^{-324} \div 1.7*10^{+308})$	64 бита (точность 15–16 цифр)
Арифметический тип с фиксированной точкой			
`decimal`	`System.Decimal`	$\pm (1.0*10^{-28} \div 7.9*10^{+28})$	28–29 значащих цифр
Символьные типы			
`char`	`System.Char`	U+0000 ÷ U+ffff	16 бит Unicode символ
`string`	`System.String`	Строка из символов Unicode	
Объектный тип			
Имя типа	Системный тип	Примечание	
`object`	`System.Object`	Базовый тип всех *встроенных* и пользовательских типов	
`void`		Отсутствие какого-либо значения	
`var`		Отложенное определение типа	

Тип данных bool

- Соответствует системному типу System.Boolean
- Может хранить только значения констант true и false (булевые константы)
- Можно назначать только булевые значения или константы, а также значений логического выражения:

```
bool bc = (c > 64 && c < 123);
```

Тип данных decimal

- 128-битный тип данных;
- имеет б**о**льшую точность и меньший диапазон значений чем типы с плавающей точкой (floating-point);
  - Диапазон $\pm 1.0 \times 10^{-28} \div \pm 7.9 \times 10^{28}$
  - Точность 28 значащих цифр
- более подходит для финансовых и денежных вычислений;
- Чтобы константа обрабатывалась как decimal нужно добавить суффикс m или M:

```
decimal myMoney = 300.5m;
```

Не определенный тип - var

- Для переменной можно задать неопределенный тип (var) и присвоит некоторое значение. В этом случае компилятор автоматически определит тип присваиваемого значения и назначит его переменной.
- Например, объявление переменной:

  ```
  var name = "Петров А.В.";
  ```

  – аналогично следующему объявлению:

  ```
  string name = "Петров А.В.";
  ```

- В этом случае обязательно нужно инициализировать переменную при ее объявлении.
- Можно также использовать неявное задание типа массива. Например, следующие операторы объявляют массив типа Point:

  ```
  var points = new[] { new Point(1, 2), new Point(3, 4) };
  ```

Тип константам (литералам) для задания типа

- Тип целой константы определяется ее значением (количеством цифр).
- Константы с дробной частью имеют тип double.
- Для изменения типа констант используются приставки:
  - Float F: 0.23F
  - Double D: 2.7D
  - Decimal M: 12.34M

Nullable типы данных

- Nullable типы данных это такие значащие типы данных, которые кроме обычных значений могут хранить и значение null.
- Например, nullable System.Boolean может хранить значения из набора {true, false, null}.
- Это очень удобно при работе с базами данных, которые кроме значений колонок могут указывать на отсутствие значения.
- Для описания nullable типа переменной нужно добавит символ вопроса (?) в конец названия типа. Это можно делать только для значащих типов.
- Например:

```
static void LocalNullableVariables()
{
  // Описываем некоторые локальные значащие nullable типы.
  int? nullableInt = 10;
  double? nullableDouble = 3.14;
  bool? nullableBool = null;
  char? nullableChar = 'a';
  int?[] arrayOfNullableInts = new int?[10];
  // Error! String является ссылочным типом!
  // string? s = "oops";
}
```

- Nullable типы являются экземплярами обобщенного типа System.Nullable<T>.
- Нет неявного преобразования nullable типа в обычный тип.
- Описана операция ??: если присваиваемое nullable значение = null, то присваивается значение, стоящее после ??.

Свойства Nullable типов

- bool HasValue – есть ли значение у переменной.
- <тип> Value – значение переменной (если переменная используется в не допустипой операции, то формируется исключение System.InvalidOperationException)
- Пример 1:

```
int? n = null;
int m = 5 + n.Value; //формируется исключение
```

- Пример 2:

```
int? n = null;
// можно выполнить явное преобразование
int m = 5 + (int)n; //формируется исключение
```

- Пример 3:

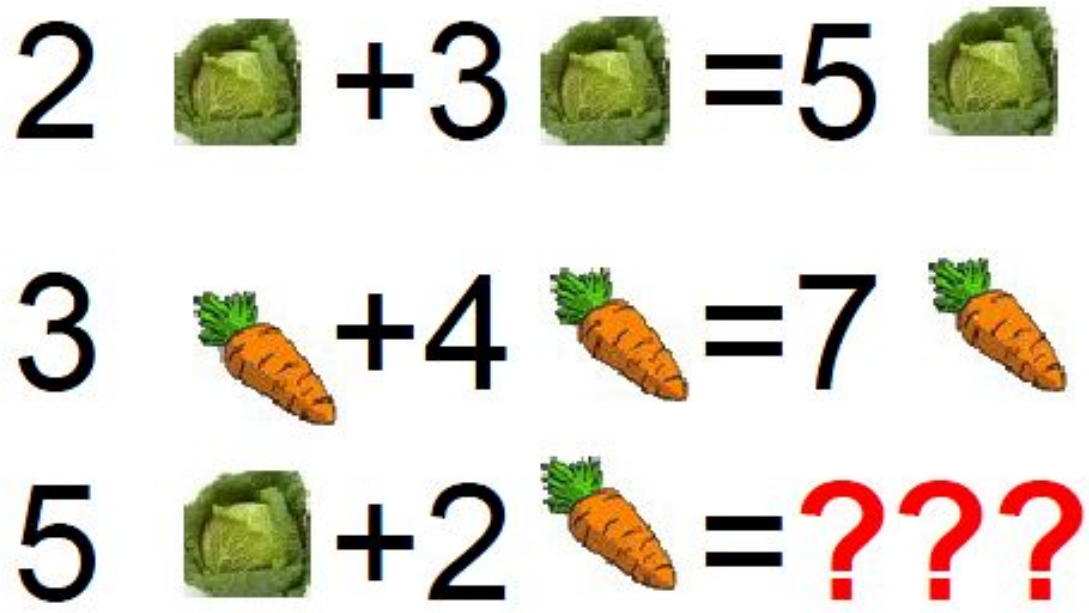
```
int? n = null;
int? m = 5 + n; // m = null
```

Использование nullable типов данных

```
static void Main(string[] args)
{
  Console.WriteLine("***** Работа с Nullable Data *****\n");
  DatabaseReader dr = new DatabaseReader();
  // Получаем int из “базы данных”.
  int? i = dr.GetIntFromDatabase();
  if (i.HasValue) // проверка на наличие значения переменной
     Console.WriteLine("Value of 'i' is: {0}", i.Value);
  else
     Console.WriteLine("Value of 'i' is undefined.");
  // Получаем bool из “базы данных”.
  bool? b = dr.GetBoolFromDatabase();
  if (b != null)
     Console.WriteLine("Значение 'b' равно: {0}", b.Value);
  else
     Console.WriteLine(" Значение 'b' не определено.");
  Console.ReadLine();
}
```

Зачем нужны типы данных?

Чтобы гарантировать осмысленность выполняемых операций:



Тип результата операции

- Тип результата операции зависит от типов участвующих в операции операндов.
- Типом арифметической операции является наиболее сложный тип операнда. Значение другого операнда преобразуется к более сложному типу.
- Наименее сложный тип byte, наиболее сложный decimal.

```
int a=5;
double d=2.6;
a * d // тип результата double
a / 2 // тип результата int
```

Тип результата операции (2)

- Типом результата операции присваивания является тип левого операнда (переменной, которой присваивается значение).

```
int n;
n = a * d // тип результата int
```

- Тип операций отношения является bool.

```
a > 5 // тип результата bool
```

- Тип логических операций является bool.

```
bool b = true, c = false;
b && c // тип результата bool
```

Преобразование типов

- **Неявное преобразование** (implicit conversion) – выполняется автоматически.

- **Явное преобразование** (explicit conversion) – выполняется по заданию программиста.

Неявное преобразование типов (implicit conversion)

- К ***неявным*** относятся преобразования, результат выполнения которых всегда успешен и не приводит к потере точности данных.
- *Неявные преобразования* выполняются автоматически.
  - Если на диаграмме есть переход из типа A в тип B то, выполняется неявное преобразование типов
  - Если нет неявного преобразования то выдается исключение

“Cannot implicitly convert type 'int?' to 'int'. An explicit conversion exists (are you missing a cast?)”

Явное преобразование типов (explicit conversion)

- К ***явным*** относятся разрешенные преобразования, выполнение которых не гарантируется или может привести к потере точности.
- Способы явного преобразования:
  - Использование операции приведения типов (cast)

```
// отсечение дробной части
int i = (int) f;
```

  - Использование стандартного класса Convert

```
// с округлением до ближайшего целого
int i = Convert.ToInt32(f);
```

  - Преобразование типов из строк с помощью метода Parse(string)

Неявное и явное преобразование

```
// Error: no conversion from int to short
int x=5, y=6;
short z = x + y;

int a = 5;
float b = 1.5F;
b = a;

// нужно явное преобразование (кастинг)
a = (int)b;
```

Неявное преобразование типов на языке Java

- char c='X';
- int code=c;
- System.out.println(code);

- Ответ: 88 (ASCII code of X)

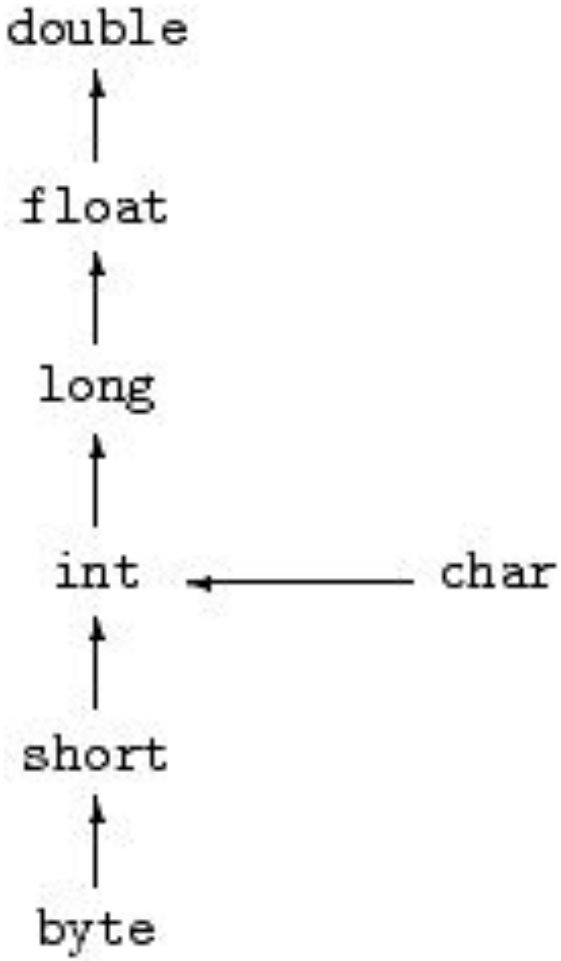


Схема неявного приведение встроенных типов

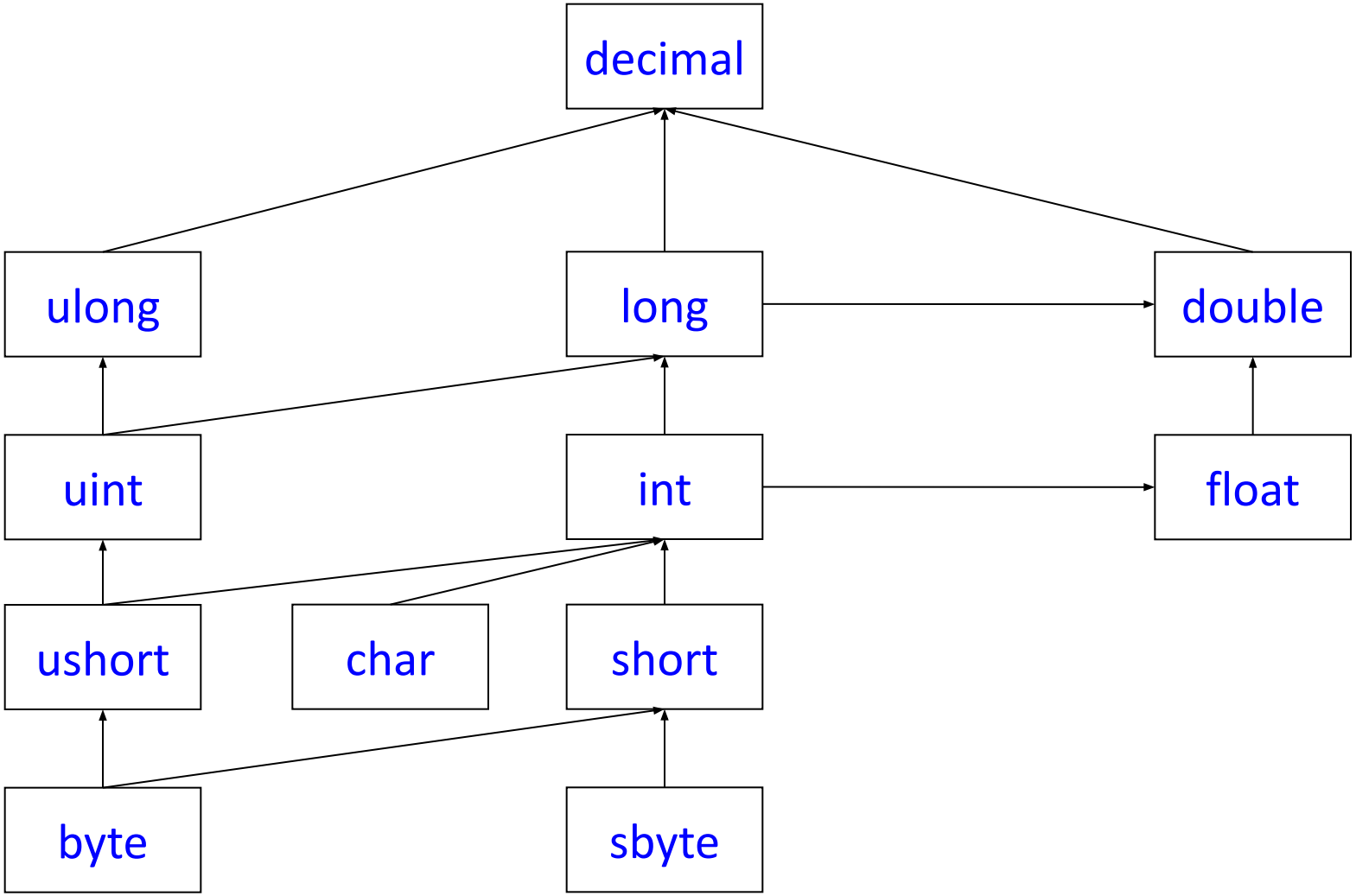


Схема неявного приведение встроенных типов (упрощенная)

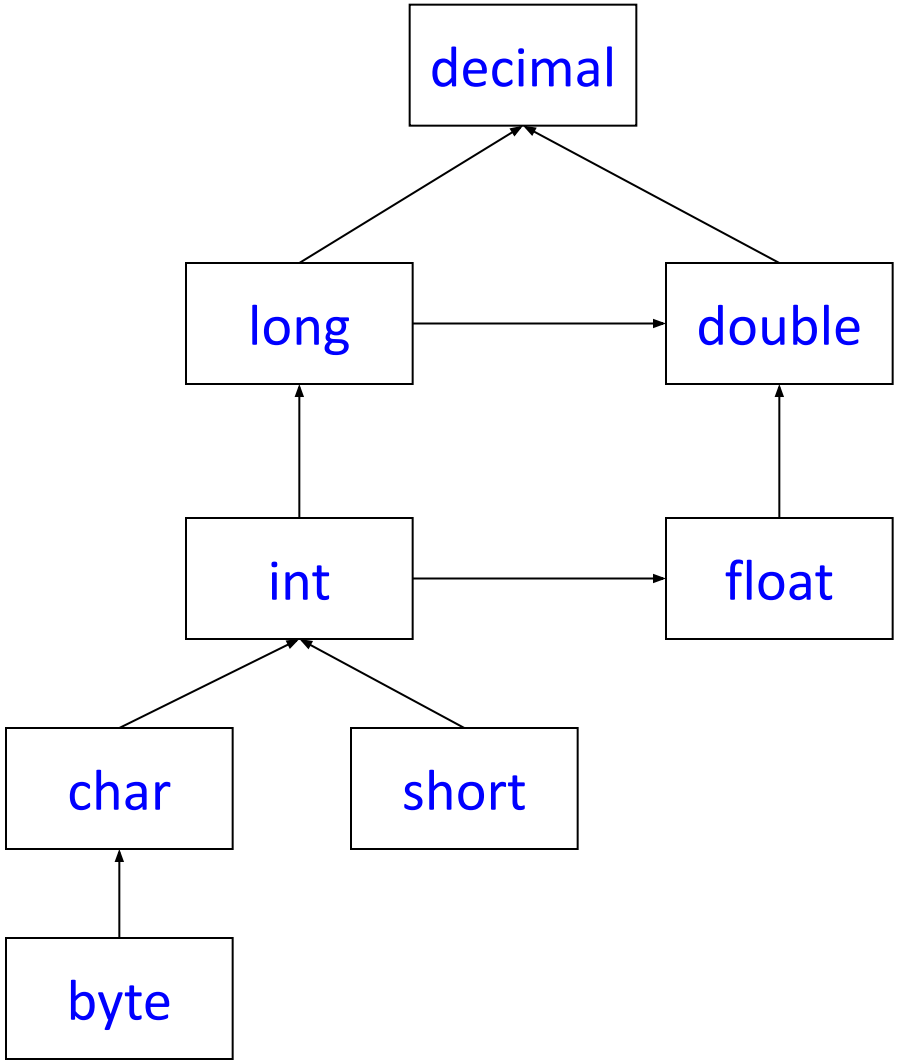
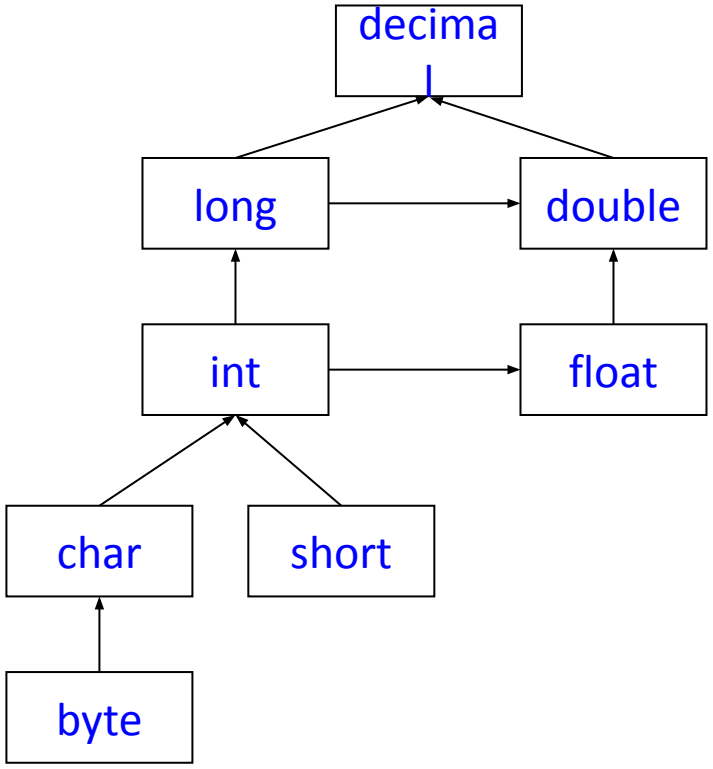


Схема неявного приведение встроенных типов (упрощенная)



byte -> char -> short -> int -> long -> float -> double -> decimal

наиболее простые типы

наиболее сложные типы

Применение диаграммы

- Если на диаграмме задан путь (стрелками) от типа A к типу B, то это означает, что возможно *неявно преобразовать* из типа A в тип B.
  - Например из short в float

- Все остальные преобразования между подтипами арифметического типа существуют, но являются *явными*.
  - Например из float в int

Пример приведения встроенных типов

```
bool c1 = true;
int d = c1;   // Error!  Cannot implicitly convert type 'bool' to 'int’
d = (int) c1; // Error! Cannot convert type 'bool' to 'int’

int a = 5;
float f = 1.9; // Error! Literal of type double cannot be implicitly
            // converted to type 'float'; use an 'F' //suffix to create a literal
                    // of this type

float b = 1.9;
a = (int)b;   // a = 1 – отбрасывается дробная часть
b = a;        // b = 1.0

decimal d = 2;
d = (decimal)b;
d = a;
```

Явное преобразование типа

- Для указания явного преобразования типов используется операция ***приведения к типу*** (***кастинг***), которая имеет высший *приоритет* и следующий вид:

  (type) <выражение>
- Она задает явное преобразование типа, определенного *выражением*, к типу, указанному в скобках.
  - Например: int i = (int) 2.99; // i = 2;
- Если описаны пользовательские типы T и P, и для типа T описано явное преобразование в тип P, то возможна следующая запись:

```
T y;
P x = new P();
y = (T) x;
```

Явное преобразование типа

- Существуют явные преобразования внутри арифметического типа,
- Не существует, например, явного преобразования арифметического типа в тип bool.
- Например:

```
double a = 5.0;
int p = (int)a;
//bool b = (bool)a; // ошибка!!!
```

  - явное преобразование из типа double в тип int выполняется,
  - явное преобразование double в тип bool приводит к ошибке, потому и закомментировано.

Преобразование типов с помощью класса Convert

- Преобразование типа с помощью методов класса System.Convert.
- *Класс Convert* содержит 15 статических *методов* вида To<Type> (ToBoolean(),...ToUInt64());

```
string s1 = Console.ReadLine();
int ux = Convert.ToUInt32(s1);
```

- Все *методы* To<Type> *класса Convert* перегружены и каждый из них имеет, как правило, более десятка реализаций с аргументами разного типа.

Пример явного преобразования типов с помощью класса Convert

- Преобразование вещественного к целому типу выполняется с округлением

```
float  b = 1.5;
a = Convert.ToInt32(b); // a=2
```

- Есть преобразование логического к целому типу

```
bool  b = true;
a = Convert.ToInt32(b); // a=1
```

Пример преобразования типов

```
System.Single f = 0.5F;
float b = f;
int a;
a = (int)f; // с обрезанием дробной части
a = Convert.ToInt32(f); // с округлением
string s = "123";
 // a = (int)s;
 a = Convert.ToInt32(s);
```

Преобразование типов из строк с помощью метода Parse()

- У всех типов есть статический метод Parse(), который выполняет преобразование строки текста в соответствующий формат.
- Для проверки возможности преобразования использовать метод bool TryParse(x), он возвращает true, если можно преобразовать иначе false

```
static void ParseFromStrings()
{
    Console.WriteLine("=> Data type parsing:");
    bool b = bool.Parse("True");
    Console.WriteLine("Value of b: {0}", b);
    double d = double.Parse("99.884");
    Console.WriteLine("Value of d: {0}", d);
    int i = int.Parse("8");
    Console.WriteLine("Value of i: {0}", i);
    char c = Char.Parse("w");
    Console.WriteLine("Value of c: {0}", c);
    Console.WriteLine();
}
```

Операция присваивания

- В C# присваивание является операцией, которая может использоваться в выражениях. В выражении, называемом множественным *присваиванием*, списку переменных присваивается одно и то же значение.
  - Например: `x = y = z = w =(u+v+w)/(u-v-w);`
- При присвоении переменных разного типа выполняется преобразование типа правого операнда к типу левого операнда.
- Т.е. компилятор пытается выполнить преобразование типа переменной стоящей справа в тип переменной, стоящей слева.

- Присваивание переменной стоящей слева (тип T) значения переменной или результата вычисления выражения (типа T1) возможно, если:
  1. типы T и T1 совпадают;
  2. тип T является базовым (родительским) типом для типа T1 (в соответствии с наследованием типов);
  3. в определении типа T1 описано явное или неявное преобразование в тип T.
- Так как все классы в языке C# – встроенные и определенные пользователем – по определению являются потомками класса Object, то отсюда и следует, что переменным класса Object можно присваивать выражения любого типа.

Специальные варианты присваивания

В языке C# для двух частных случаев *присваивания* предложен специальный синтаксис:

1. для *присваиваний* вида x=x+1; (переменная увеличивается или уменьшается на единицу), используются специальные ***префиксные*** и ***постфиксные*** операции "++" и "--".
2. для присваивания вида:

   X = X <operator> (выражение); например: x = x * 2;

   – Для таких присваиваний используется краткая форма записи: X <operator>= expression; например: x *= 2;

- Типы (встроенные, пользовательские – классы, структуры, интерфейсы).
- Операции
- Выражения
- Преобразование типов

2. Операции

Операции

- Переменные и константы могут участвовать (объединяться) с помощью операций.
- Операция – это термин или символ, получающий на вход одно или несколько операндов (переменных или констант) или выражений (переменных или констант, связанных между собой знаками операций), и возвращающий значение некоторого типа
- Например: a + b и или ++a * pi
  - Операции, получающие на вход один операнд, например операция приращения (++) или `new`, называются ***унарными операциями***.
  - Операции, получающие на вход два операнда, например, арифметические операции (+, –, *, /) называются ***бинарными операциями***.
  - Одна операция – условная (?:), получает на вход три операнда и является единственной ***третичной операцией*** в C#.

Базовые операции

Основные операции	
Выражение	*Описание*
`x.y`	доступ к элементам типа
`f(x)`	вызов метода и делегата
`a[x]`	доступ к массиву и индексатору
`x++`	постфиксное приращение
`x--`	постфиксное уменьшение
`new T(...)`	создание объекта класса или делегата
`new T(...) {...}`	создание объекта с инициализацией
`new T[...]`	создание массива (см. раздел 3.5).
`typeof(T)`	получение объекта System.Type для T
`delegate {}`	анонимная функция (анонимный метод)

Унарные операции

Унарные операции	
Выражение	*Описание*
`-x`	отрицательное значение
`!x`	логическое отрицание
`~x`	поразрядное отрицание
`++x`	префиксное приращение
`--x`	префиксное уменьшение
`(T)x`	явное преобразование x в тип T (кастинг)

Бинарные операции

Мультипликативные операции		**Аддитивные операции**	
Выражение	*Описание*	*Выражение*	*Описание*
`*`	умножение	`x + y`	сложение, объединение строк
`/`	деление	`x - y`	вычитание
`%`	остаток		
Операции сдвига		**Операции равенства**	
Выражение	*Описание*	*Выражение*	*Описание*
`x << y`	сдвиг влево	`x == y`	равно
`x >> y`	сдвиг вправо	`x != y`	не равно

Бинарные операции (продолжение)

Операции отношения и типа	
Выражение	*Описание*
`x < y`	меньше
`x > y`	больше
`x <= y`	меньше или равно
`x >= y`	больше или равно
`x is T`	возвращает значение true, если x относится к типу T, в противном случае возвращает значение false
`x as T`	возвращает x типа T или нулевое значение, если x не относится к типу T
Операции назначения и анонимные операции	
Выражение	*Описание*
`=`	присваивание
`x op= y`	составные операции присвоения: +=, -=, *=, /=, %=, &=, |=, !=, <<=, >>=

Логические и условные операции

Логические, условные операции и Null-операции		
Категория	*Выражение*	*Описание*
Логическое AND	`x & y`	целочисленное поразрядное AND, логическое AND
Логическое исключающее XOR	`x ^ y`	целочисленное поразрядное исключающее XOR, логическое исключающее XOR
Логическое OR	`x | y`	целочисленное поразрядное OR, логическое OR
Условное AND	`x && y`	вычисляет y только если x имеет значение true
Условное OR	`x || y`	вычисляет y только если x имеет значение false
Объединение нулей	`x ?? y`	равно y, если x = null, в противном случае равно x
Условное	`x ? y:z`	равно y, если x имеет значение true, z если x имеет значение false

Приоритеты операций языка C#

Приоритет	**Категория**	**Операции**	**Ассоциативность**
0	Первичные	`(expr) x.y f(x) a[x] x++ x-- new sizeof(t) typeof(t) checked(expr) unchecked(expr)`	Слева направо
1	Унарные	`+ - ! ~ ++x --x (T)x`	Слева направо
2	Мультипликативные (Умножение)	`* / %`	Слева направо
3	Аддитивные (Сложение)	`+ -`	Слева направо
4	Сдвиг	`<< >>`	Слева направо
5	Отношения, проверка типов	`< > <= >= is as`	Слева направо
6	Эквивалентность	`== !=`	Слева направо
7	Логическое	`&`	Слева направо
8	Логическое исключающее ИЛИ (XOR)	`^`	Слева направо
9	Логическое ИЛИ (OR)	`|`	Слева направо
10	Условное И	`&&`	Слева направо
11	Условное ИЛИ	`||`	Слева направо
12	Условное выражение	`? :`	Справа налево
13	Присваивание	`= *= /= %= += -= <<= >>= &= ^= |=`	Справа налево

Пояснение приоритета операций

- Вычисление *выражений* начинается с выполнения операций высшего *приоритета*.
- Например: первым делом вычисляются *выражения* в круглых скобках – (expr), определяются значения полей объекта – x.y, вычисляются функции – f(x), переменные с индексами – a[i].
- Можно заключать выражения в скобки для принудительного вычисления некоторых частей выражения раньше других. Например, выражение 2 + 3 * 2 в обычном случае будет иметь значение 8, поскольку операции умножения выполняются раньше операций сложения. А результатом вычисления выражения (2 + 3) * 2 будет число 10, поскольку компилятор C# получит данные о том, что операцию сложения (+) нужно вычислить до выполнения операции умножения (*).
- Если есть несколько операций с одинаковым приоритетом, то они вычисляются в соответствии с их ассоциативностью.
  - Операции с левой ассоциативностью вычисляются слева направо. Например, x * y / z вычисляется как (x * y)/z.
  - Операции с правой ассоциативностью вычисляются справа налево.
  - Операции присваивания и третичная операция (?:) имеют правую ассоциативность. Все другие двоичные операции имеют левую ассоциативность.
- Порядок выполнения операций с объектами пользовательских классов и структур можно изменить. Такой процесс называется перегрузкой операций.

Тип результата операции

- Тип результата операции зависит от типов участвующих в операции операндов.
- Типом арифметической операции является наиболее сложный тип операнда. Значение другого операнда преобразуется к более сложному типу.
- Наименее сложный тип byte, наиболее сложный decimal.

```
int a=5;
double d=2.6;
a * d // тип результата double
a / 2 // тип результата int
```

- Типом результата операции присваивания является тип левого операнда (переменной, которой присваивается значение).

```
int n;
n = a * d // тип результата int
```

- Тип операций отношения является bool.

```
a > 5 // тип результата bool
```

- Тип логических операций является bool.

```
bool b = true, c = false;
b && c // тип результата bool
```

Преобразование типов

- Неявное преобразование (implicit conversion)
  - К ***неявным*** относятся те преобразования, результат выполнения которых всегда успешен и не приводит к потере точности данных.
  - *Неявные преобразования* выполняются автоматически.
  - Если на диаграмме есть переход из типа A в тип B то, выполняется неявное преобразование типов
  - Если нет неявного преобразования то исключение
    - “Cannot implicitly convert type 'int?' to 'int'. An explicit conversion exists (are you missing a cast?)”
- Явное преобразование (explicit conversion)
  - К ***явным*** относятся разрешенные преобразования, успех выполнения которых не гарантируется или может приводить к потере точности
  - Использование приведения типов (cast)

    ```
    int i = (int) f; // с обрезанием дробной части
    ```
  - Использование стандартного класса Convert

    ```
    int i = Convert.ToInt32(f); // с округлением до ближайшего целого
    ```

Неявное и явное преобразование

```
// Error: no conversion from int to short
int x=5, y=6;
short z = x + y;

int a = 5;
float b = 1.5F;
b = a;

// нужно явное преобразование (кастинг)
a = (int)b;
```

Неявное преобразование типов на языке Java

- char c='X';
- int code=c;
- System.out.println(code);

- Ответ: 88 (ASCII code of X)

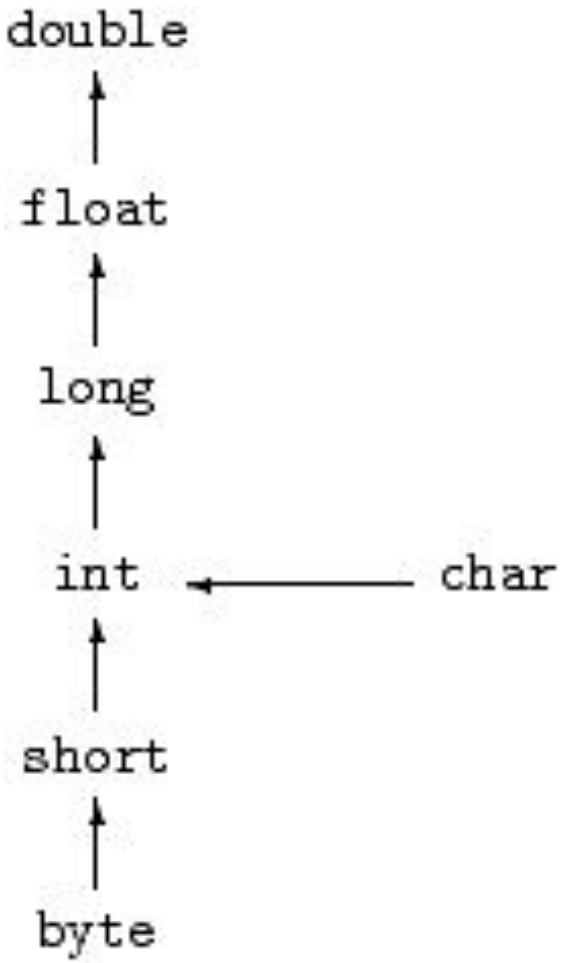


Схема неявного приведение встроенных типов

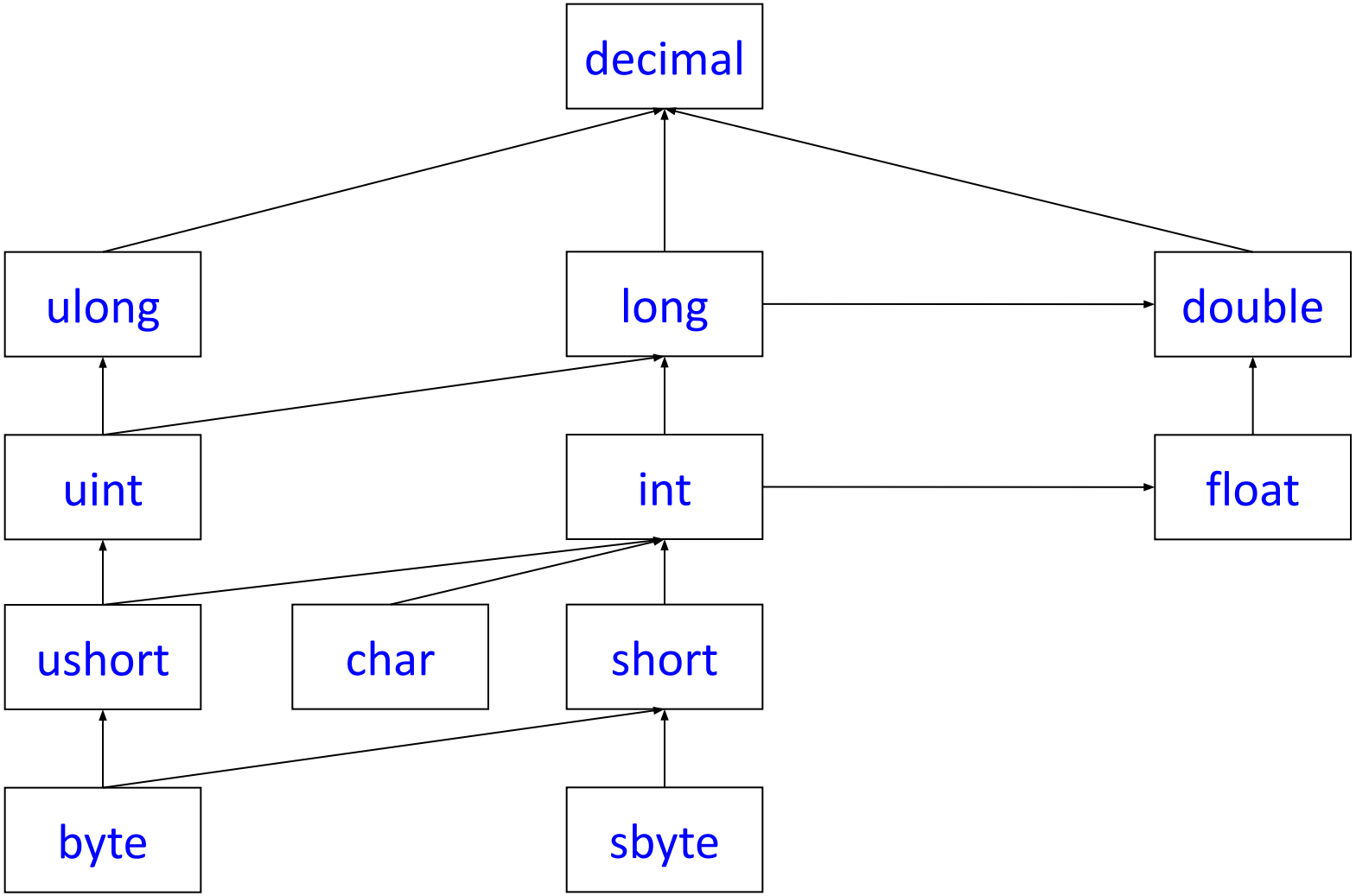
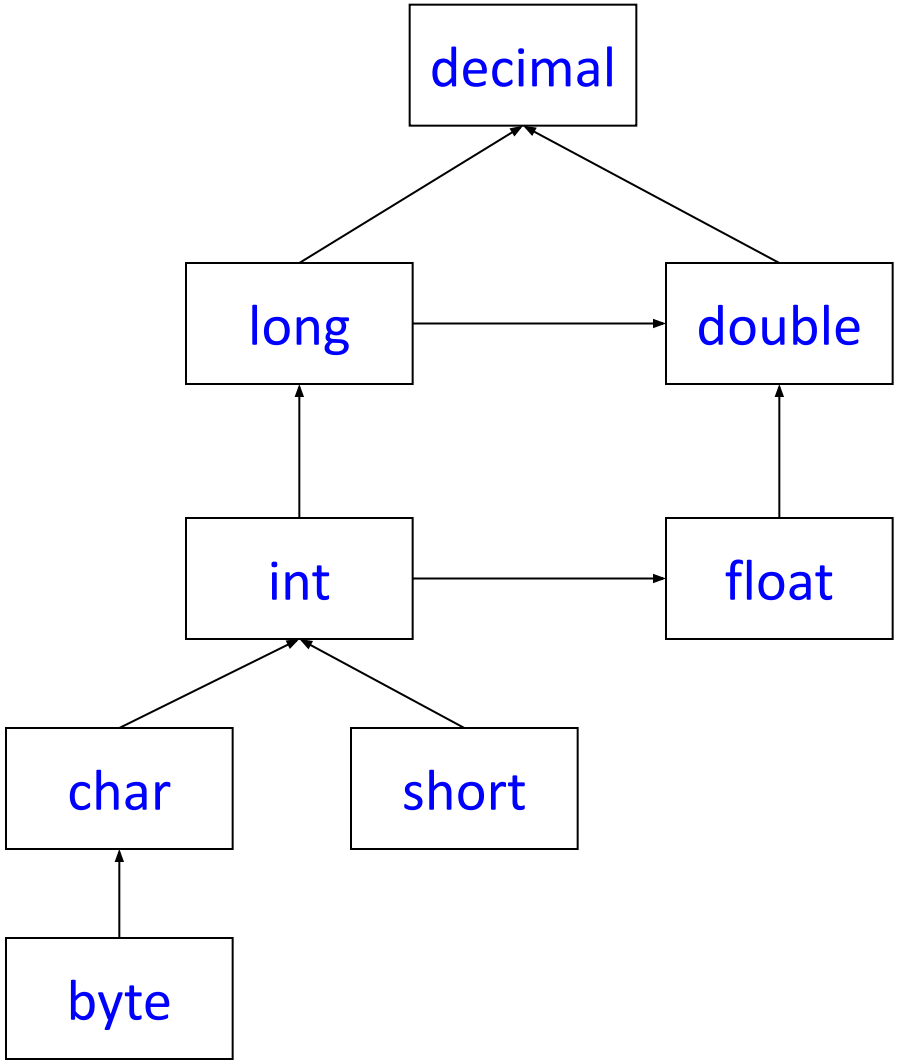


Схема неявного приведение встроенных типов (упрощенная)



Применение диаграммы

- Если на диаграмме задан путь (стрелками) от типа A к типу B, то это означает, что возможно *неявно преобразовать* из типа A в тип B.
  - Например из short в float
- Все остальные преобразования между подтипами арифметического типа существуют, но являются *явными*.
  - Например из float в int

Пример приведения встроенных типов

```
bool c1 = true;
int d = c1;   // Error!    Cannot implicitly convert type 'bool' to 'int’
d = (int) c1; //Error! Cannot convert type 'bool' to 'int’

int a = 5;
float f = 1.9; // Error! Literal of type double cannot be implicitly
          //converted to type 'float'; use an 'F' //suffix to create a literal
                // of this type

float b = 1.9;
a = (int)b; // a = 1 – отбрасывается дробная часть
b = a;       // b = 1.0

decimal d = 2;
d = (decimal)b;
d = a;
```

Явное преобразование типа

- Для указания явного преобразования типов используется операция ***приведения к типу*** (***кастинг***), которая имеет высший *приоритет* и следующий вид:

  (type) <выражение>
- Она задает явное преобразование типа, определенного *выражением*, к типу, указанному в скобках. Например: int i = (int) 2.99; // i = 2;
- Если описаны пользовательские типы T и P, и для типа T описано явное преобразование в тип P, то возможна следующая запись:

```
T y;
P x = new P();
y = (T) x;
```

- Следует отметить, что существуют явные преобразования внутри арифметического типа, но не существует, например, явного преобразования арифметического типа в тип bool. Например:

```
double a = 5.0;
int p = (int)a;
//bool b = (bool)a; // ошибка!!!
```

- В данном примере явное преобразование из типа double в тип int выполняется, а преобразование double в тип bool приводит к ошибке, потому и закомментировано.

Преобразование типов с помощью класса Convert

- Можно задать явным образом требуемое преобразование, используя специальные методы преобразования, определенные в классе System.Convert, которые обеспечивают преобразование значения одного типа к значению другого типа (в том числе значения строкового типа к значениям встроенных типов).
- *Класс Convert* содержит 15 статических *методов* вида To <Type> (ToBoolean(),...ToUInt64());

```
string s1 = Console.ReadLine();
int ux = Convert.ToUInt32(s1);
```

- Все *методы* To <Type> *класса Convert* перегружены и каждый из них имеет, как правило, более десятка реализаций с аргументами разного типа.
- Преобразование вещественного к целому типу выполняется с округлением

```
float  b = 1.5;
a = Convert.ToInt32(b); // a=2
```

- Есть преобразование логического к целому типу

```
bool  b = true;
a = Convert.ToInt32(b); // a=1
```

Пример преобразования типов

```
System.Single f = 0.5F;
float b = f;
int a;
a = (int)f; // с обрезанием дробной части
a = Convert.ToInt32(f); // с округлением
string s = "123";
 // a = (int)s;
 a = Convert.ToInt32(s);
```

Преобразование типов из строк с помощью метода Parse()

- У всех типов есть статический метод Parse(), который выполняет преобразование строки текста в соответствующий формат.
- Для проверки возможности преобразования использовать метод bool TryParse(x) // true, если можно преобразовать иначе false

```
static void ParseFromStrings()
{
    Console.WriteLine("=> Data type parsing:");
    bool b = bool.Parse("True");
    Console.WriteLine("Value of b: {0}", b);
    double d = double.Parse("99.884");
    Console.WriteLine("Value of d: {0}", d);
    int i = int.Parse("8");
    Console.WriteLine("Value of i: {0}", i);
    char c = Char.Parse("w");
    Console.WriteLine("Value of c: {0}", c);
    Console.WriteLine();
}
```

Операция присваивания

- В C# присваивание является операцией, которая может использоваться в выражениях. В выражении, называемом множественным *присваиванием*, списку переменных присваивается одно и то же значение.
- Например: x = y = z = w =(u+v+w)/(u-v-w);
- При присвоении переменных разного типа выполняется преобразование типа правого операнда к типу левого операнда. Т.е. компилятор пытается выполнить преобразование типа переменной стоящей справа в тип переменной, стоящей слева.
- Присваивание переменной стоящей слева (тип T) значения переменной или результата вычисления выражения (типа T1) возможно только в следующих случаях:
  - типы T и T1 совпадают;
  - тип T является базовым (родительским) типом для типа T1 (в соответствии с наследованием типов);
  - в определении типа T1 описано явное или неявное преобразование в тип T.
- Так как все классы в языке C# – встроенные и определенные пользователем – по определению являются потомками класса Object, то отсюда и следует, что переменным класса Object можно присваивать выражения любого типа.

Специальные варианты присваивания

- В языке C# для двух частных случаев *присваивания* предложен специальный синтаксис.
- Для *присваиваний* вида "x=x+1", в которых переменная увеличивается или уменьшается на единицу, используются специальные префиксные и постфиксные операции "++" и "--".
- Другой важный частный случай – это краткая запись для присваивания вида: X = X <operator> (expression); например: x = x * 2;
- Для таких присваиваний используется краткая форма записи: X <operator>= expression; например: x *= 2;
- В качестве операции разрешается использовать арифметические и логические (побитовые) операции языка C#. Например: x += u + v; y /= (u-v);

Арифметические операции

- В языке C# имеются обычные для всех языков *арифметические операции* – "+, -, *, /, %". Все они перегружены.
- Операции "+" и "-" могут быть унарными и бинарными.
- Операция деления "/" над целыми типами осуществляет деление нацело, для типов с плавающей и фиксированной точкой – обычное деление.
- Операция "%" определена над всеми арифметическими типами и возвращает остаток от деления нацело.

```
int a = 10;
int e = 4;
a %= e; // или a = a % e; - результат 2
```

- Тип результата зависит от типов операндов выражения: самый сложный тип задает тип результата выражения.

Пример вычислений с различными арифметическими типами

```
public void Sample()
{
  int n = 7, m =3, p, q;
  p= n/m; q= p*m + n%m;
  if (q == n) Console.WriteLine("q = n");
  else Console.WriteLine("q!=n");
  double x=7.2 , y =3, u,v,w;
  u = x/y;     // 2.4
  v= u*y;     //  7.1999999999999993
  w= x % y; // 1.2
  if (v == x) // false
    Console.WriteLine("v = x");
  else
    Console.WriteLine("v!=x");
  decimal d1=7M, d2 =3, d3,d4,d5;
  d3 = d1/d2; // 2.3333333333333333333333
  d4= d3*d2; // 6.9999999999999999999999
  d5= d1%d2; // 1.0
  if (d4 == d1) // false
    Console.WriteLine("d4 = d1");
  else
    Console.WriteLine("d4!=d1");
}
```

```
// небольшое изменение
decimal d1 = 7.2M, d2 = 3, d3, d4, d5;
d3 = d1 / d2;    // 2.4
d4 = d3 * d2;   // 7.2
d5 = d1 % d2;   // 1.2
 if (d4 == d1)     // true
    Console.WriteLine("d4=d1");
 else
    Console.WriteLine("d4!=d1");
```

Операции инкрементации и декрементации

- Операции *инкрементации* (увеличение на единицу) и *декрементации* (уменьшение на единицу) могут быть
  - префиксными (стоять перед переменной) и
  - постфиксными (стоять после переменной).
- К высшему *приоритету* относятся постфиксные операции x++ и x--. Префиксные операции имеют на единицу меньший *приоритет*.
- Результатом выполнения, как префиксных, так и постфиксных операций, является увеличение (++) или уменьшение (--) значения переменной на единицу.
- Для префиксных (++x, --x) операций результатом их выполнения является измененное значение x, постфиксные операции возвращают в качестве результата значение x до изменения. Например:

```
int n1, n2, n = 5;
n1 = n++; // n1 = 5; n = 6;
n2 = ++n; // n2 = 7; n = 7;
```

Операции отношения

- *Операции отношения* используются для сравнения значений переменных и констант.
- Всего имеется 6 операций отношения: ==, !=, <, >, <=, >=.
- Следует обратить внимание на запись операции "равно" – ‘==’ (два знака присвоить =) и "не равно" – ‘!=’.
- При сравнении ссылочных переменных сравниваются не сами объекты, а ссылки на объекты, если операция сравнения не переопределена.

Логические операции

- В языке C# *логические операции* делятся на две категории:
  - над логическими значениями операндов,
  - над битами операндов.
- По этой причине в C# существуют две унарные операции отрицания
  - логическое отрицание, заданное операцией "!", определена над операндом типа bool,
  - побитовое отрицание, заданное операцией "~", определена над операндом целочисленного типа, начиная с типа int и выше (int, uint, long, ulong).
- Результатом операции "~" является операнд, в котором каждый бит заменен его дополнением (0 на 1 и 1 на 0).

Пример логических операций

- Рассмотрим пример:

```
//операции отрицания ~,!
bool b1,b2;
b1 = 2*2==4; b2 =!b1; // b1 = true; b2 = false;
//b2= ~b1; // ошибка !
uint j1 =7, j2;
j2= ~j1; // j2 = 4294967288
//j2 = !j1; // ошибка !
int j4 = 7, j5;
j5 = ~j4;  // j5 = -8
```

- В этом фрагменте закомментированы операторы, приводящие к ошибкам.
- В первом случае была сделана попытка применения операции побитового отрицания к *выражению* типа bool, во втором – логическое отрицание применялось к целочисленным данным.
- Обратите внимание на разную интерпретацию побитового отрицания для беззнаковых и знаковых целочисленных типов. Для переменных j5 и j2 строка битов, задающая значение – одна и та же, но интерпретируется по-разному.

Бинарные логические операции

- Операции && и || определены только над данными типа bool:
  - && – условное **И** (результат true, если оба операнда имеют значение true);
  - || – условное **ИЛИ** (результат true, если хотя бы один операнд имеет значение true);
- Операции && и || называются условными (или краткими), поскольку, будет ли вычисляться второй операнд, зависит от уже вычисленного значения первого операнда.
  - в операции &&, если первый операнд равен значению false, то второй операнд не вычисляется и результат операции равен false.
  - в операции ||, если первый операнд равен значению true, то второй операнд не вычисляется и результат операции равен true.
- Такое свойство *логических операций* позволяет вычислить *логическое выражение*, имеющее смысл, но в котором второй операнд не определен.

Пример логических операций

- Например, рассмотрим задачу поиска элемента массива. Заданный элемент в массиве может быть, а может и не быть.

```
//Условное And – &&
int[] ar= {1,2,3};
int search = 7, i=0;
// search – заданное значение
while ((i < ar.Length) && (ar[i]!= search)) i++;
if(i < ar.Length)
   Console.WriteLine("Значение найдено");
else
   Console.WriteLine("Значение не найдено");
```

- Второй операнд не определен в последней проверке, поскольку индекс элемента массива выходит за допустимые пределы (в C# индексация элементов начинается с нуля). Отметим, что "нормальная" конъюнкция требует вычисления обеих операндов, поэтому ее применение в данной программе приводило бы к формированию исключения в случае, когда образца нет в массиве.

Побитовые операции

- Три бинарные побитовые операции:
  - & – AND (если значения двух бит = 1, то и результирующий бит =1);
  - | – OR (если значения хотя бы одного бита = 1, то и результирующий бит =1);
  - ^ – XOR (исключа́ющее ИЛИ) могут использоваться как с целыми типами выше int, так и с булевыми типами. В первом случае они используются как побитовые операции, во втором – как обычные *логические операции*.

$a = 01100101_2$

$b = 00101001_2$

тогда

$a \wedge b = 01001100_2$

- Иногда необходимо, чтобы оба операнда вычислялись в любом случае, тогда без этих операций не обойтись. Вот пример первого их использования:

```
//Логические побитовые операции And, Or, XOR (&,|,^)
int k2 = 7, k3 = 5, k4, k5, k6;
k4 = k2 & k3; k5 = k2 | k3; k6 = k2^k3;
```

Таблицы истинности

- a и b типа bool:

a	b	a && b
false	false	false
false	true	false
true	false	false
true	true	true

a	b	a || b
false	false	false
false	true	true
true	false	true
true	true	true

a	b	a ^ b
false	false	false
false	true	true
true	false	true
true	true	false

- a и b типа int:

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Условная операция

- В C# имеется *условный операция*, которые начинаются с условия, заключенного в круглые скобки, после которого следует знак вопроса и пара *выражений*, разделенных двоеточием ‘:’.
- Условием является *выражение* типа bool. Если оно истинно, то из пары *выражений* выбирается первое, в противном случае результатом является значение второго *выражения*.
- Например:

```
int a = 7, b = 9, max;
max = (a>b) ? a:b; // max получит значение 9.
```