Лекция 2: Использование размерных типов данных

Обзор

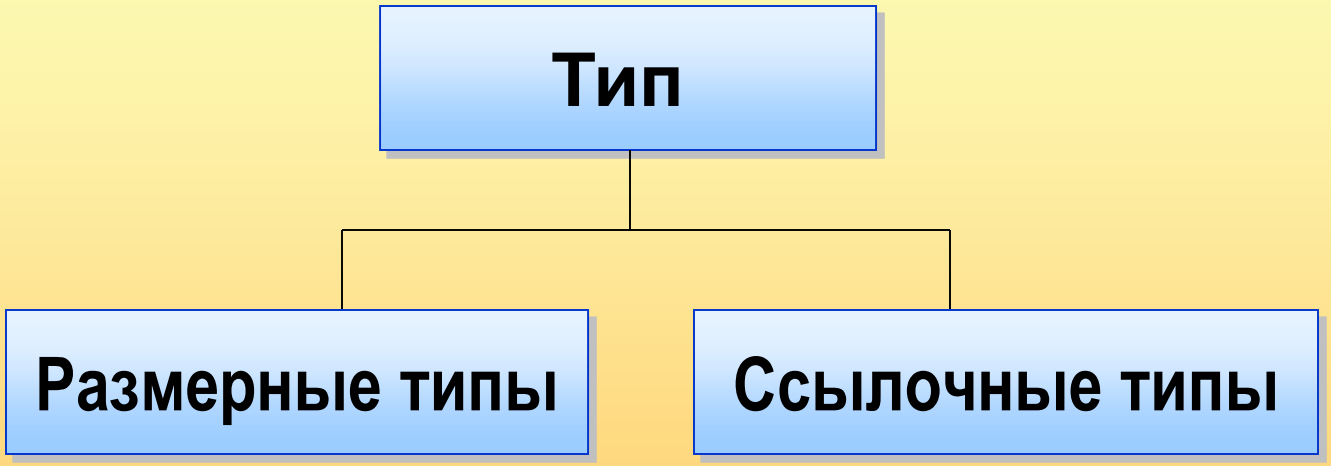
- Common Type System (унифицированная система типов)
- Объявление переменных
- Использование встроенных типов данных
- Создание пользовательских типов данных
- Преобразование типов

Common Type System

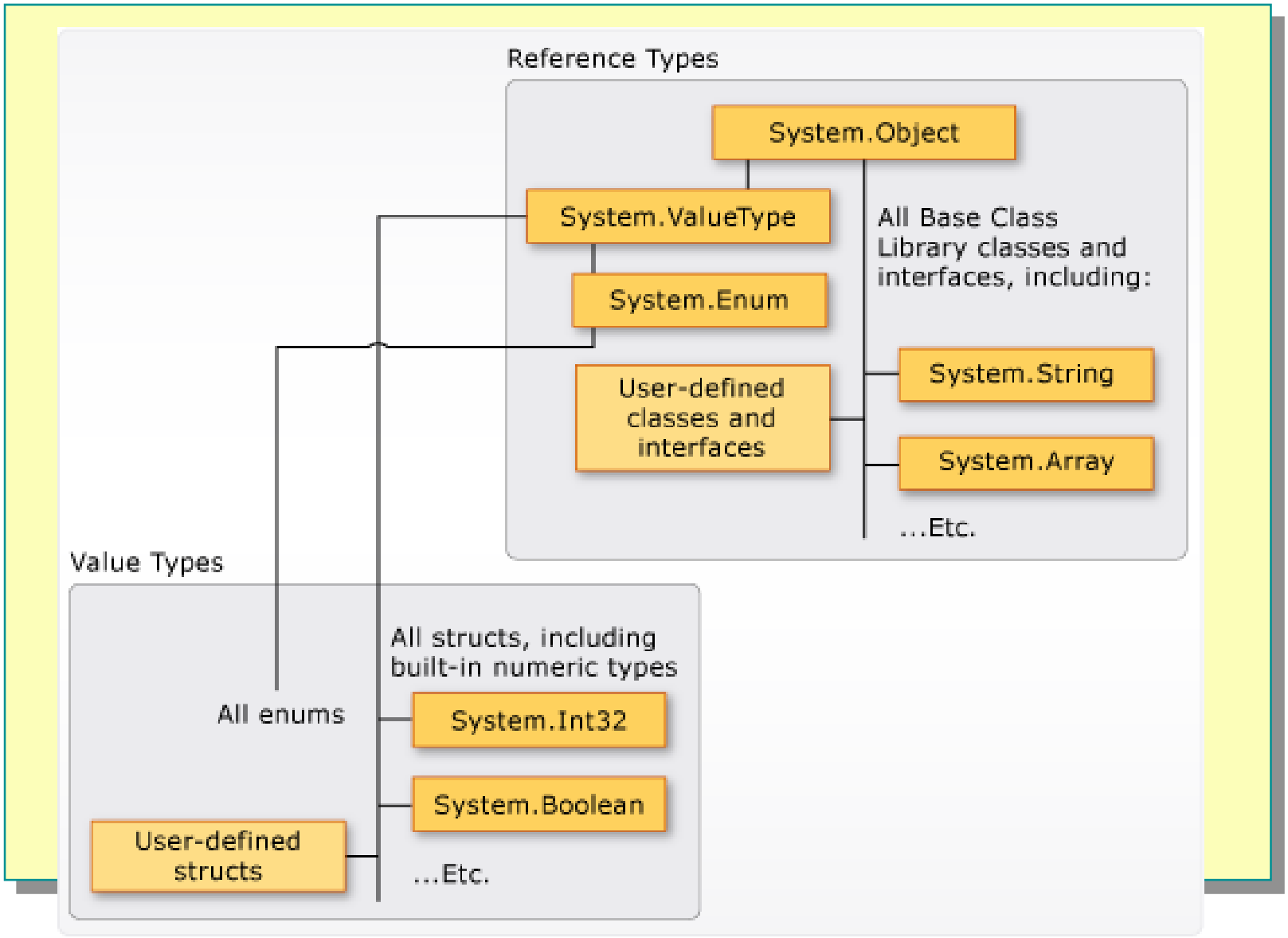
- Обзор CTS
- Сравнение размерных и ссылочных типов данных
- Сравнение встроенных типов данных с пользовательскими
- Простые типы

Обзор CTS

- CTS поддерживает, как размерные, так и ссылочные типы данных



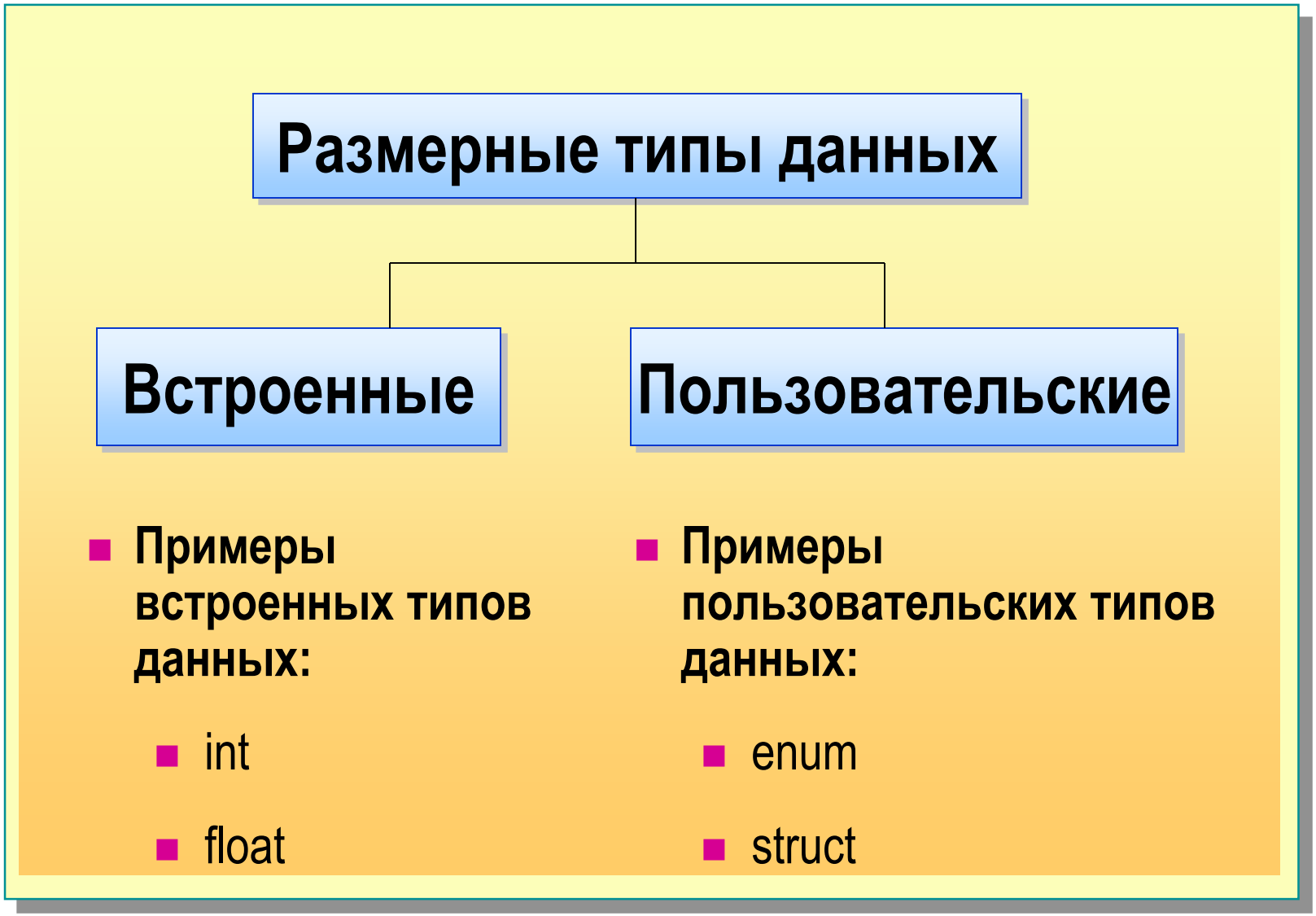
Система общих типов CTS



Сравнение размерных и ссылочных типов данных

- **Размерные типы:**
  - Непосредственно хранят данные
  - В каждой хранится своя копия данных
  - Операции над одной переменной не могут повлиять на другую
- **Ссылочные типы:**
  - Хранят ссылки на данные (объекты)
  - Две ссылочные переменные могут указывать на один объект
  - Операции над одной переменной могут повлиять на другую

Сравнение встроенных типов данных с пользовательскими



Встроенные типы данных

- Объявление локальных переменных
- Инициализация переменных
- Составная операция присваивания
- Основные операторы
- Инкремент и декремент
- Приоритет операторов

Простые типы

- **Идентифицируются через зарезервированные ключевые слова**
- **int /* Reserved keyword - или - */ System.Int32**

Тип	Область значений	Размер	Пояснение
bool	true или false	1 байт	Логический, или булев, представляет значения ИСТИНА/ЛОЖЬ
byte	0 - 255	1 байт	8-разрядный целочисленный без знака
char	от U+0000 до U+ffff	2 байта	16-разрядный символ Юникода
decimal	от $\pm 1{,}0 \times 10^{-28}$ до $\pm 7{,}9 \times 10^{28}$	16 байт	Числовой тип для финансовых вычислений
double	от $\pm 5{,}0 \times 10^{-324}$ до $\pm 1{,}7 \times 10^{308}$	8 байт	С плавающей точкой двойной точности
float	от $\pm 1{,}5 \times 10^{-45}$ до $\pm 3{,}4 \times 10^{38}$	4 байта	С плавающей точкой
int	от -2 147 483 648 до 2 147 483 647	4 байта	32-разрядное целое число со знаком
long	от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807	8 байт	64-разрядное целое число со знаком
sbyte	от -128 до 127	1 байт	8-разрядное целое число со знаком
short	от -32 768 до 32 767	2 байта	16-разрядное целое число со знаком
uint	от 0 до 4 294 967 295	4 байта	32-разрядное целое число без знака
ulong	от 0 до 18 446 744 073 709 551 615	8 байт	64-разрядное целое число без знака
ushort	от 0 до 65 535	2 байта	16-разрядное целое число без знака

Назначение имен переменных

- **Общая структура объявления**

```
[<атрибуты>] [<модификаторы>] <тип> <объявители>;
```

- **Правила и рекомендации по назначению имен переменных**
- **Ключевые слова в C#**

Правила и рекомендации по назначению имен

- **Правила**
  - Использовать буквы, цифры или символ подчеркивания
- **Рекомендации**
  - Не использовать только прописные буквы
  - Не использовать первым символ подчеркивания
  - Не пользоваться аббревиатурами
  - Для имен, состоящих из нескольких слов использовать технику «Паскаль»

Ключевые слова C#

- **Ключевые слова – предопределенные идентификаторы, имеющие определенное значение для компилятора**

```
abstract, base, bool, default, if, finally
```

- **Не используйте ключевые слова для именования переменных**
  - Иначе ошибка на этапе компиляции
- **Избегайте использования ключевых слов через смену регистра**

```
int INT;  // Poor style
```

Объявление локальных переменных

- Объявляется тип данных и имя переменной:

```
int itemCount;
```

```
int itemCount = new int();
```

- Допускается объявление нескольких переменных в одной строке:

```
int itemCount, employeeNumber;
```

--или--

```
int itemCount,
    employeeNumber;
```

Инициализация переменной

- **Присвоение значений уже объявленным переменным :**

```
int employeeNumber;

employeeNumber = 23;
```

- **Инициализация переменной при объявлении:**

```
int employeeNumber = 23;
```

- **Также можно инициализировать символьные переменные:**

```
char middleInitial = 'J';
```

Типы, допускающие неопределенные значения

- **Язык C# позволяет из любого значимого типа данных построить новый тип, отличающийся тем, что множество возможных значений дополнено специальным значением null.**

```
System.Nullable<T>

эквивалентная, более простая форма записи:

T?
```

```
  int x = 3, y = 7;
  int? x1 = null, y1, z1;
  y1 = x + y;
  z1 = x1 ?? y1;
Console.WriteLine("x1 = {0}, y1 = {1}, z1 = {2}",
      x1, y1, z1);
```

Null, NaN и Infinity

Правила, приводящие к появлению особых значений:

- **Если при выполнении операций умножения или деления результат по модулю превосходит максимально допустимое число, то значением является *бесконечность* или *отрицательная бесконечность* в зависимости от знака результата.**
- **При делении вещественного числа на бесконечность результат равен нулю.**
- **Если один из операндов вычисляемого выражения есть null, а остальные - обычные вещественные числа или бесконечность, то результат выражения будет иметь значение null - не определен.**
- **Если бесконечность делится на бесконечность или ноль умножается на бесконечность, то результат будет NaN.**
- **Если NaN участвует в операциях, то результатом будет NaN.**

Время жизни и область видимости переменных

- **Роль переменных, - они задают свойства структур, интерфейсов, классов.**
  - В языке C# такие переменные называются полями
  - Время существования и область видимости полей определяется объектом, которому они принадлежат
- **В пространствах имен, проектах, решениях - *нельзя* объявлять переменные**
- **Переменную можно объявлять в любой точке процедурного блока.**
  - Область ее видимости распространяется от точки объявления до конца процедурного блока

Неявно типизированные переменные

- Локальным переменным вместо явного типа может быть задан определенный "тип" var.
- Ключевое слово var сообщает компилятору необходимости определения типа переменной из выражения, находящегося с правой стороны оператора инициализации.

```
var i = 5; // i is compiled as an int

var s = "Hello";// s is compiled as a string

var a = new[] { 0, 1, 2 };// a is compiled as
int[]
```

readonly

- **Если объявление поля содержит модификатор readonly, присвоение значений таким полям может происходить только как часть объявления или в конструкторе в том же классе**

```
public readonly int y = 5;
```

```
class Age   {
  readonly int _year;
  Age(int year)
        { _year = year; }
  void ChangeYear()
    {   //_year = 1967; // Compile error   }
    }
```

Константы

- **Могут появляться, как обычно, в виде**
  - Литералов

```
y = 7.7f;
```

  - и именованных констант

```
const int x = 0;
public const double gravitationalConstant = 6.6e-11;
private const string productName = "Visual C#";
public const double x = 1.0, y = 2.0, z = 3.0;
public const int c1 = 5;
public const int c2 = c1 + 100;
```

Составная операция присваивания

- Часто бывает необходимо добавить значение к уже существующей переменной:

```
itemCount = itemCount + 40;
```

- Существует удобная краткая запись:

```
itemCount += 40;
```

- Эта краткая запись применима ко всем арифметическим операциям:

```
itemCount -= 24;
```

Основные операторы

Основные операторы	Пример
• Операторы равенства	== !=
• Операторы отношения	< > <= >= is
• Условные операторы	&& || ?:
• Инкремент	++
• Декремент	--
• Арифметические операторы	+ - * / %
• Операторы присваивания	= *= /= %= += -= <<= >>= &= ^= |=

Инкремент и декремент

- Часто необходимо изменить значение на единицу:

```
itemCount += 1;
itemCount -= 1;
```

- Существует удобная краткая запись:

```
itemCount++;
itemCount--;
```

- Эта краткая запись может использоваться двумя способами:

```
++itemCount;
--itemCount;
```

Пример

```
static void Main()
{
 int i = 3, j = 4;
 Console.WriteLine("{0} {1}", i, j);
 Console.WriteLine("{0} {1}", ++i, --j);
 Console.WriteLine("{0} {1}", i++, j--);
 Console.WriteLine("{0} {1}", i, j);
}
```

Результат работы программы:

- 3 4
- 4 3
- 4 3
- 5 2

Приоритет операторов

- **Приоритет операторов и ассоциативность**
  - Все бинарные операторы, кроме оператора присваивания, левоассоциативны
  - Операторы присваивания и условные операторы - правоассоциативны

Создание пользовательских типов данных

- Перечисления
- Структуры

Перечисления

- **Синтаксис определения перечисления**

```
enum <имя> [ : базовый тип]
{список-перечисления констант(через запятую)};
```

- **Создание перечисления**

```
enum Color { Red, Green, Blue }
```

- **Использование перечисления**

```
Color colorPalette = Color.Red;
```

- **Отображение перечисления**

```
Console.WriteLine("{0}", colorPalette); // Displays Red
```

Структуры

- **Создание структуры**

```
public struct Employee
{
    public string firstName;
    public int age;
}
```

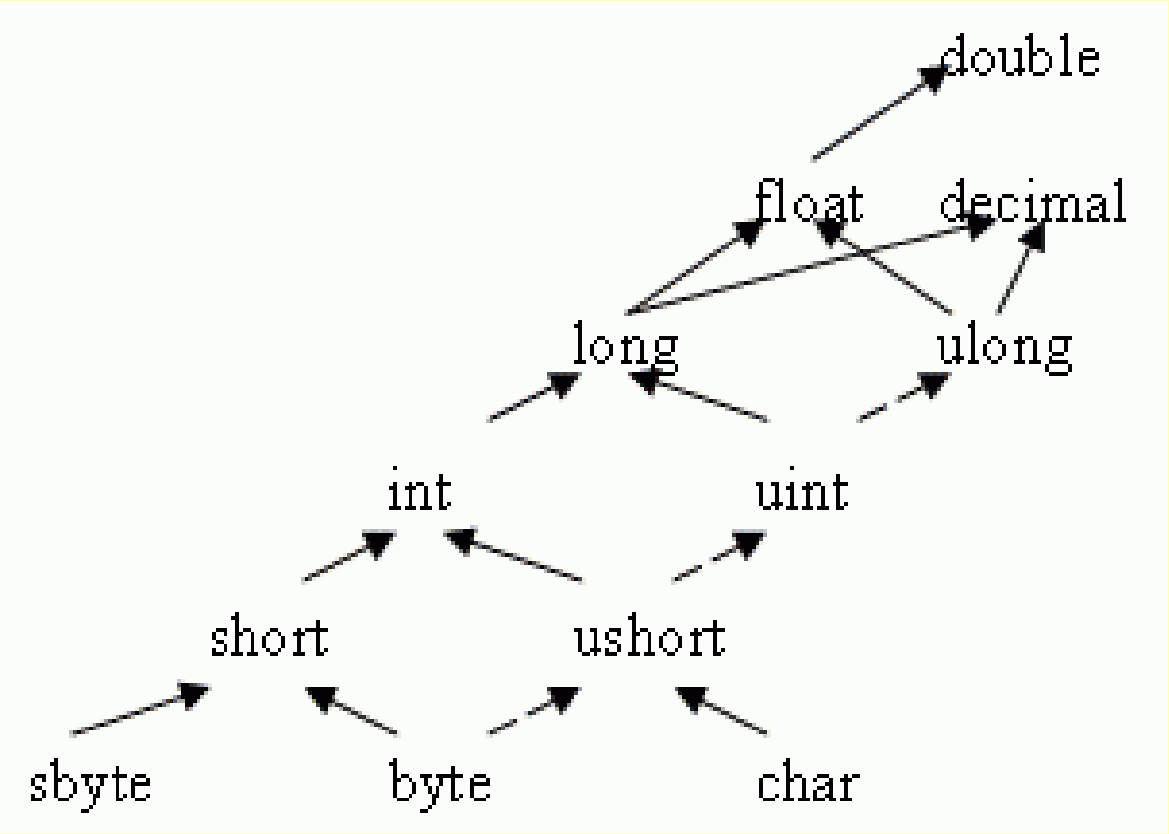
- **Использование структуры**

```
Employee companyEmployee;
companyEmployee.firstName = "Joe";
companyEmployee.age = 23;
```

Преобразование типов

- Неявное преобразование типов
- Явное преобразование типов

Иерархия типов при неявном преобразовании



Неявное (автоматическое) преобразование типов

- **Преобразование int в long:**

```
using System;
class Test
{
    static void Main( )
    {
        int intValue = 123;
        long longValue = intValue;
        Console.WriteLine("(long) {0} = {1}", intValue,
    ➥longValue);
    }
}
```

- **Неявные преобразования всегда проходят успешно**
  - Можно потерять точность, но не значение

Явное преобразование типов

- **При явном преобразовании используется приведение типов:**

```
static void Main()
{
 int i = 24;
 byte j = 250;
 int a = (int)j;  //преобразование без потери точности
 byte b = (byte)i; //преобразование с потерей точности
 Console.WriteLine("{0} {1}", a, b);
}
```

Результат работы программы:
250 24

Лабораторная работа 2: Создание и использование размерных типов