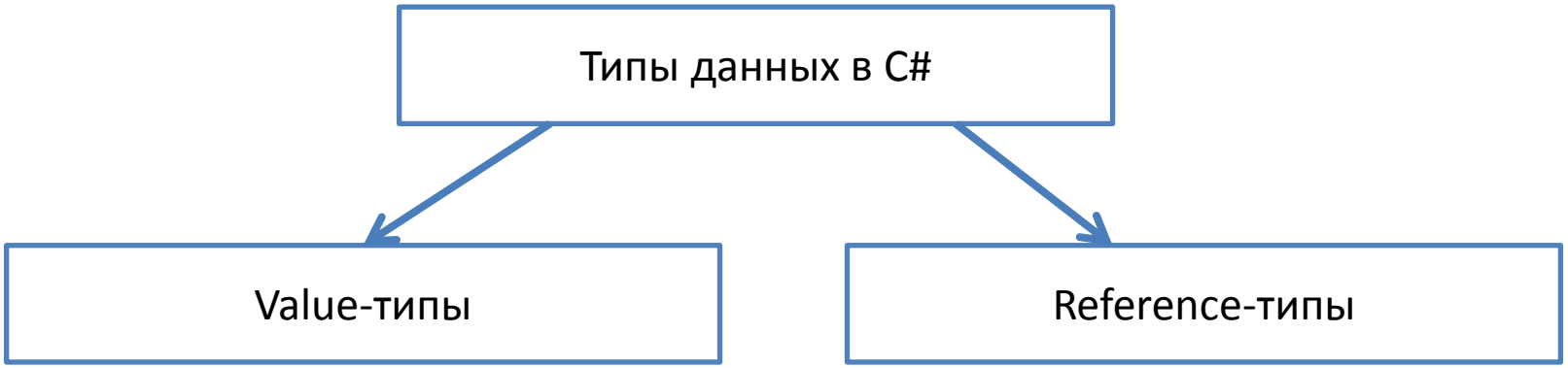
# Лекция 8. Value-типы. Reference-типы. Символьный тип

# Типы данных в C#

- Все типы в C# можно разделить на две категории: **value-типы (типы значений)** и **reference-типы (ссылочные типы)**
- Типы из данных категорий ведут себя по-разному
- К value-типам относятся, например числа, а к reference-типам относятся строки

# Типы данных в C#




**Структуры (объявлены как struct):**

- Все числовые типы
- bool, char
- DateTime, TimeSpan
- Nullable-типы

**Енумы (enum)**

**Классы (объявлены как class):**

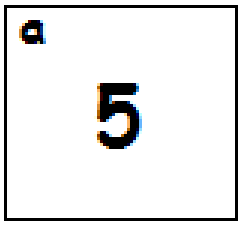
- string
- И другие типы

# Value-типы

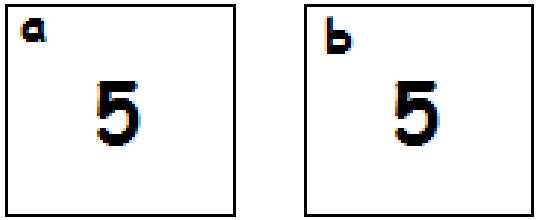
- Переменные value-типов хранят само значение типа
- При присваивании происходит копирование значения
- При передаче аргументов в функции, происходит копирование аргумента

# Как работают value-типы

- int a = 5;



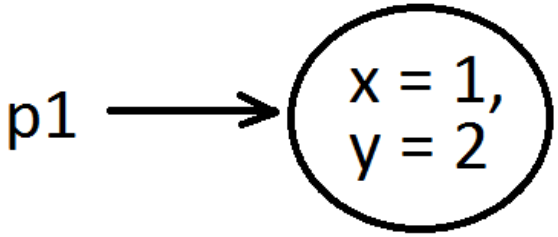

- int b = a;



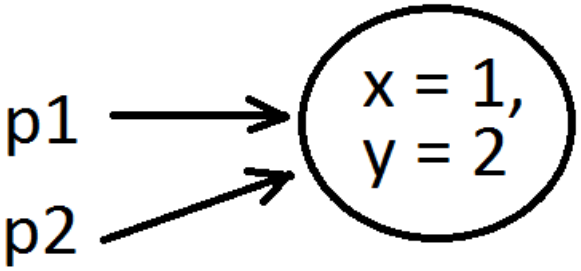

- Если изменить a или b, то это не повлияет на другую переменную

# Reference-типы

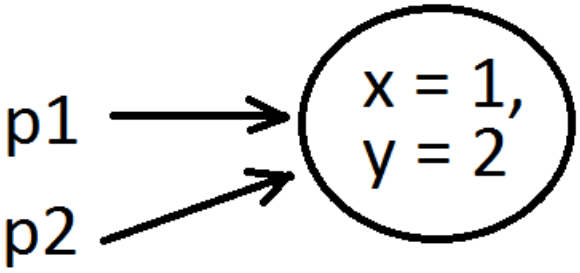
- Переменные хранят не само значение, а **ссылку** на него (по сути – адрес в памяти)
- Point p1 = new Point(1, 2);




- При присваивании происходит копирование ссылки:
- Point p2 = p1;



# Reference-типы




- Если изменим объект,
  то все ссылки будут
  указывать на измененный объект
- p2.SetX(3);
  Console.WriteLine(p1.GetX()); // 3
- При передаче объекта в функцию, происходит копирование ссылки на него
- Зачем нужны ссылки? Чтобы более эффективно работать с памятью. Объекты часто являются большими, и копировать их очень затратно по времени и памяти

# Проверка объектов на равенство

- Для объектов нельзя использовать проверку через == и !=, если эти операторы не переопределены
- Для объектов оператор == проверяет, что ссылки указывают на один и тот же объект в памяти или нет
- Аналогично != проверяет, что ссылки указывают на разные объекты
- Чтобы сравнить содержимое объектов, нужно использовать метод Equals
- bool x = o1.Equals(o2)

# Значение null

- Переменные ссылочных типов могут принимать специальное значение null
- Пример: string s = null;
- Оно означает пустую ссылку, то есть адрес, который никуда не указывает
- Если вызвать функцию для переменной, которая имеет значение null, то произойдет ошибка NullReferenceException

# Для чего полезен null?

- Значение null может быть полезно, если мы хотим показать, что функция отработала, но получить результат не удалось
- Например, мы написали функцию, которая ищет строку нужной длины среди некоторого набора строк
- Но такой строки не оказалось
- В этом случае функция может вернуть null, а вызывающий код проверить, что результат равен null и, например, напечатать сообщение, что ничего не найдено

# Для чего полезен null?

```
public static string FindString(int length)
{
    // код, который делает return, если нашел строку

    // в конце делается return null если
    // ничего не найдено
    return null;
}
public static void Main()
{
    if (FindString(4) == null)
    {
        Console.WriteLine("Ничего не найдено");
    }
}
```

# Символьный тип char

- Кроме строкового типа, в C# есть символьный тип char
- Это value-тип
- Размер переменной – 2 байта
- Его переменные могут хранить один символ
- Литералы заключены в одинарные кавычки: ‘a’, ‘5’, ‘\\’, ‘\n’
- char lineSeparator = ‘\n’;

# Символьный тип char

- У строк можно брать символ по порядковому номеру (отсчитывается от 0)
- `string s = "ABCDE";`
- `char secondSymbol = s[1]; // B`
- `char lastSymbol = s[5];`
  `// ошибка при исполнении программы –`
  `// выход за границы строки`
- Правильно:
  `char lastSymbol = s[s.Length – 1];`

# Функции для работы с символами

- Статические методы типа char:
  - bool IsDigit(char c) – проверка что цифра
  - bool IsLetter(char c) – проверка что буква
  - bool IsLetterOrDigit(char c) – что буква или цифра
  - bool IsLower(char c) – что буква в нижнем регистре
  - bool IsUpper(char c) – что буква в верхнем регистре

- Пример:

```
bool isDigit = char.IsDigit('4');   // true
```

# Пробельные символы

- bool char.IsWhiteSpace(char c) – проверка, что это пробельный символ
- **Пробельными символами** считаются пробел, табуляция и перевод строки

# Функции работы с символами

- Статические методы типа char:
  - char ToUpper(char c) – перевод в верхний регистр
  - char ToLower(char c) – перевод в нижний регистр

- Если символ уже в этом регистре, или не буква, то выдается сам символ
- **Пример:**
- ```
  char lowerCaseChar1 = char.ToLower('A');   // a
  char lowerCaseChar2 = char.ToLower('a');   // a
  char upperCaseChar1 = char.ToUpper('A');   // A
  char upperCaseChar2 = char.ToUpper('a');   // A
  ```
  ```

# Пример работы со строками

```
string name = Console.ReadLine();
if (char.IsLower(name[0]))
{
    Console.WriteLine(
      “Имя должно начинаться с заглавной буквы!”);
    return;
}
```

# Проход по всем символам строки

- ```
  string s = Console.ReadLine();

  for (int i = 0; i < s.Length; ++i)
  {
      char c = s[i];
      // работаем с текущим символом c
  }
  ```
  ```

# Задача «Подсчет символов»

- Прочитать с консоли строку
- Вывести число букв в этой строке
- Вывести число цифр в этой строке
- Вывести число пробелов в этой строке
- Вывести число остальных символов в строке

# Задача на курс «Макс. подстрока»

- Написать функцию, которая ищет в строке подстроку максимальной длины, состоящую из одного и того же символа, и выдает эту максимальную длину
- Например, есть строка "аааббдеггггв", должно выдаться число 4, потому что есть 4 подряд символа «г», и это максимальная подстрока, где подряд идет один и тот же символ
- Функция должна работать без учета регистра

# Задача на курс «Палиндром»

- Объявить некоторую строковую переменную в программе
- Проверить, что данная строка является палиндромом – то есть читается одинаково слева направо и справа налево.
- При проверке не учитывать регистр символов, учитывать только буквы
- Пример палиндрома: «Аргентина манит негра»
- **Требование:** сделать без создания новой строки и без удаления символов из строки