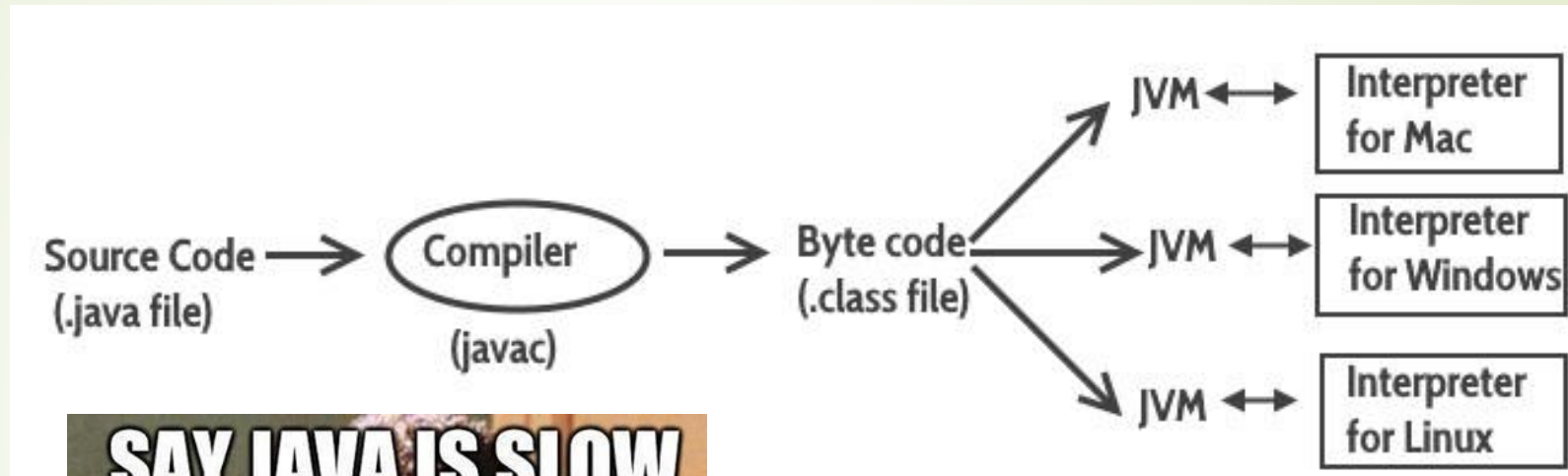




# Лекция 2

## .Net Framework

# JVM. JIT-КОМПЛЯЦИЯ

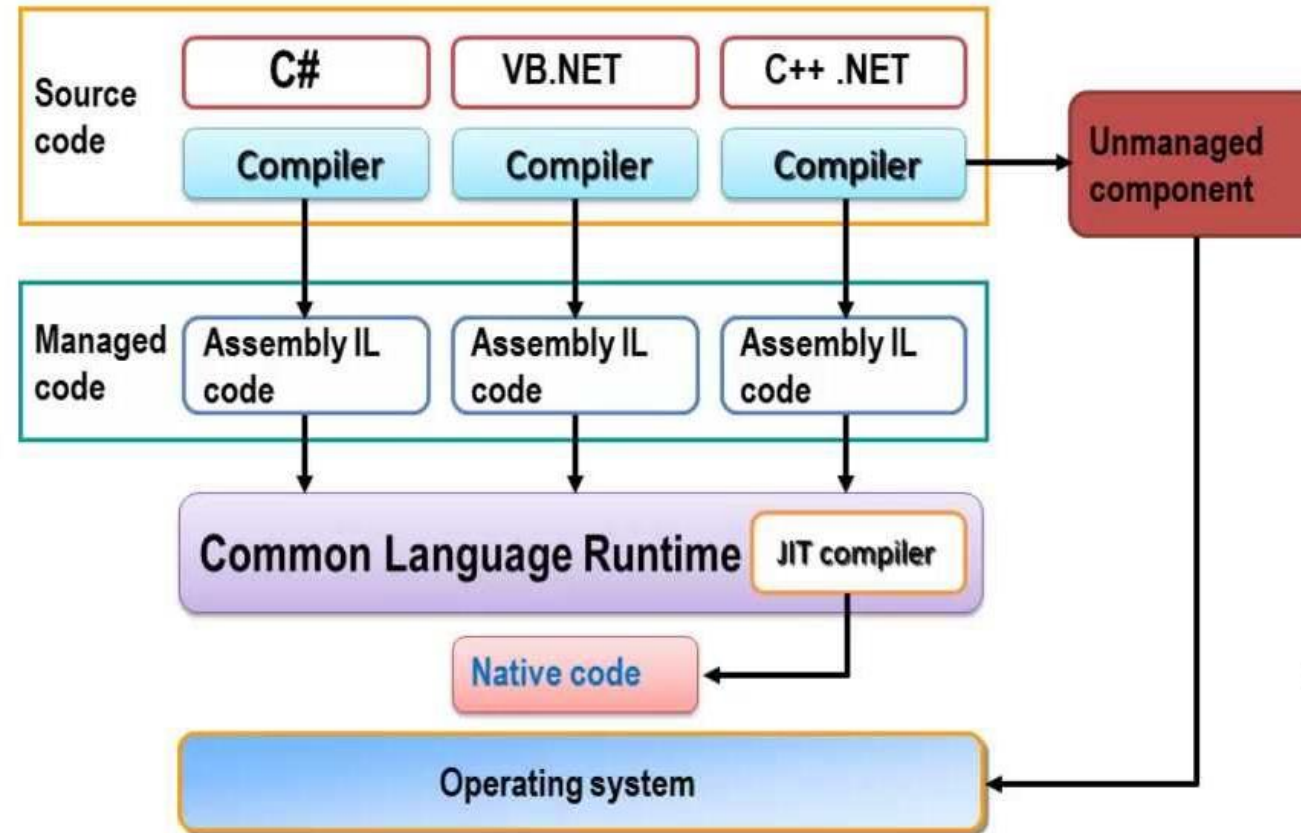


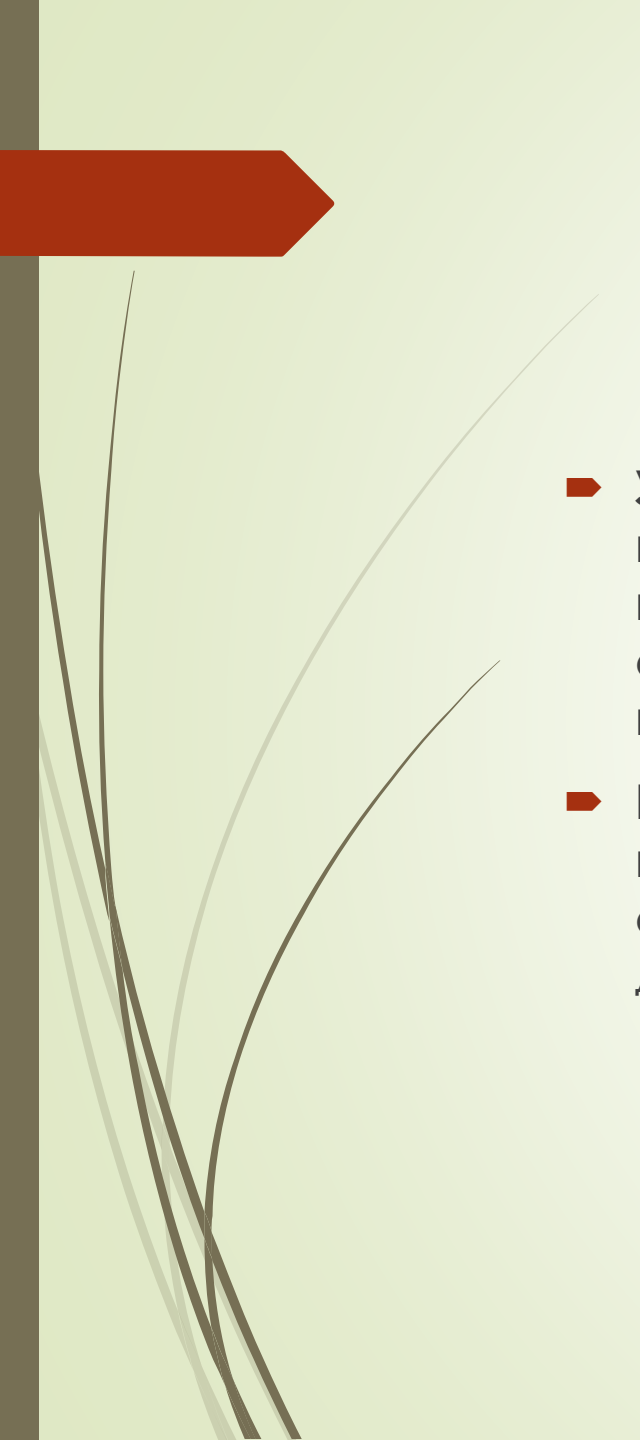


# .Net Framework

- CLR (Common language runtime) — общезыковая исполняющая среда
- Framework Class Library (FCL) - библиотека классов
- Компиляторы C# и VB.NET
- Система сборки MSBuild
- Набор специфических утилит командной строки

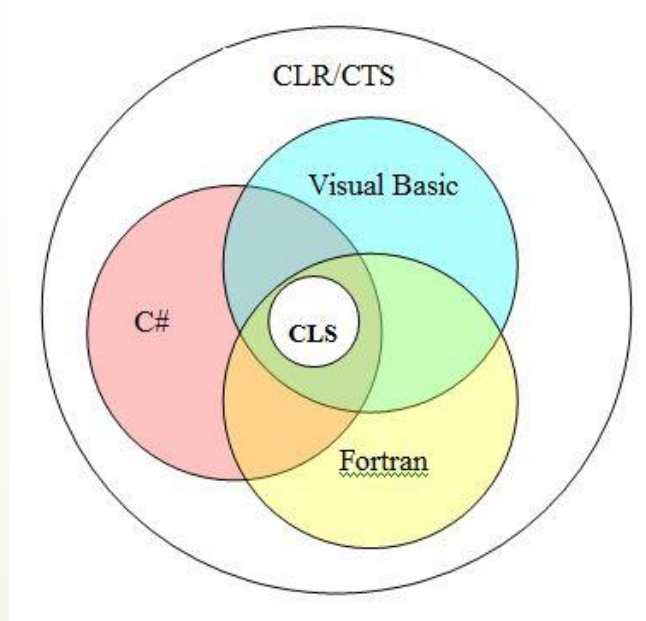
# The CLR Execution Model



- 
- **Управляемый код** (managed) компилируется не в машинный код, а в промежуточный язык, который интерпретируется и выполняется службой на физической машине и поэтому работает в рамках безопасной структуры, которая обрабатывает опасные вещи, такие как память и потоки.
  - **Неуправляемый код** (unmanaged) компилируется в машинный код и поэтому выполняется непосредственно ОС. Поэтому он обладает способностью делать разрушительные/мощные вещи, которых не делает управляемый код. Пример: вызов функций из dll.



- **CTS (Common Type System)** — общая система типов в CLR. Это — стандарт, признанный ECMA который описывает определение типов и их поведение. Также определяет правила наследования, виртуальных методов, времени жизни объектов.
- **CLS (Common Language Specification)** — спецификации выпущенная Microsoft. Она описывает минимальный набор возможностей, которые должны реализовать производители компиляторов, чтобы их продукты работали в CLR.



# Типы данных



+ Пользовательские структуры и перечисления



# Struct

- Структуры имеют свои члены: методы, поля, индексаторы, свойства, операторные методы и события. В структурах допускается также определять конструкторы (но не деструкторы!). В то же время для структуры нельзя определить конструктор, используемый по умолчанию (т.е. конструктор без параметров). Помимо того, структуры не поддерживают наследование.
- Начиная с C# 8.0, можно также использовать модификатор `readonly`, чтобы объявить, что член экземпляра не изменяет состояние структуры.



# Передача параметров по ссылке

- Модификатор ref

```
public void SampleMethod(ref int i) { }
```

- Модификатор in (нельзя изменять в теле подпрограммы)

```
public void SampleMethod(in int i) { }
```

- Модификатор out (не требуется предварительная инициализация)

```
public void SampleMethod(out int i) { }
```

- Перегружать методы, отличающиеся **ТОЛЬКО** данными модификаторами **нельзя!**



# Enum

- Тип перечисления — это тип значения, определенный набором именованных констант применяемого целочисленного типа.

```
enum Season  
{  
    Spring,  
    Summer,  
    Autumn,  
    Winter  
}
```

```
enum ErrorCode : ushort  
{  
    None = 0,  
    ConnectionLost = 100,  
}
```

# Типы перечислений как битовые флаги

[Flags]

```
public enum Days
```

```
{
```

```
    None    = 0b_0000_0000, // 0
```

```
    Monday  = 0b_0000_0001, // 1
```

```
    Tuesday = 0b_0000_0010, // 2
```

```
    Wednesday = 0b_0000_0100, // 4
```

```
    Thursday = 0b_0000_1000, // 8
```

```
    Friday   = 0b_0001_0000, // 16
```

```
    Saturday = 0b_0010_0000, // 32
```

```
    Sunday   = 0b_0100_0000, // 64
```

```
    Weekend  = Saturday | Sunday
```

```
}
```

```
....
```

```
Days meetingDays = Days.Monday | Days.Wednesday | Days.Friday;
```



# Класс Object

- Equals — поддерживает сравнения между объектами
- Finalize — выполняет операции очистки до автоматического освобождения объекта
- GetHashCode — создает число, соответствующее значению объекта для поддержки использования хэш-таблицы
- ToString — Создает понятную для человека текстовую строку, описывающую экземпляр класса
- GetType - точный тип текущего экземпляра в среде выполнения

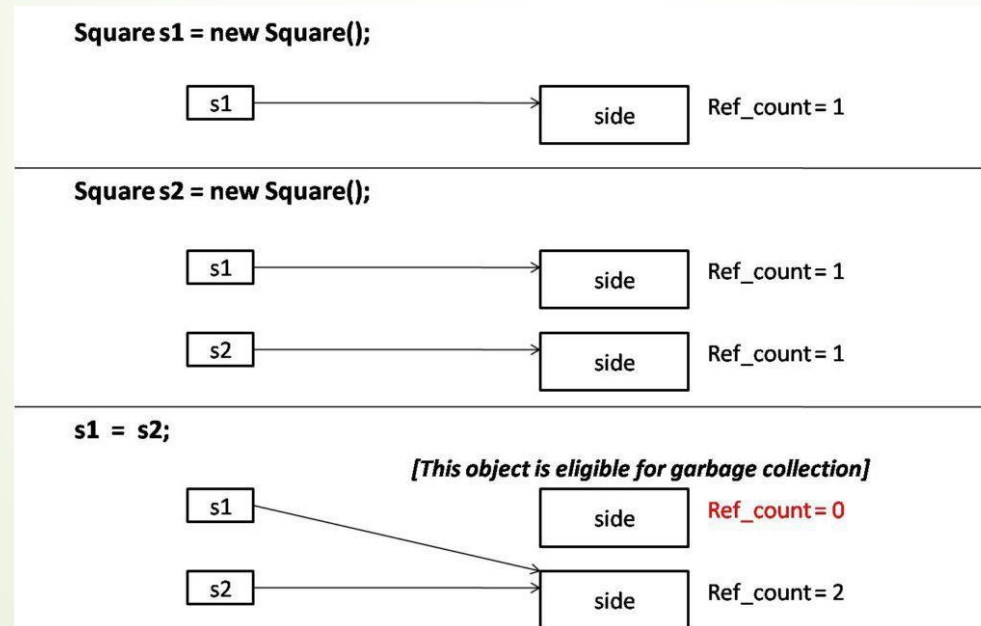


# Namespaces

- **Пространство имен** определяет область объявлений, в которой допускается хранить одно множество имен отдельно от другого. По существу, имена, объявленные в одном пространстве имен, не будут вступать в конфликт с аналогичными именами, объявленными в другой области.
- Примеры пространств имен FCL
  - ❖ System
  - ❖ System.Collections
  - ❖ System.Data
  - ❖ System.Drawing
  - ❖ System.IO
  - ❖ System.Runtime
  - ❖ System.Text
  - ❖ System.Threading

# Garbage Collector (GC)

- Сборка мусора – одна из форм автоматического управления памятью





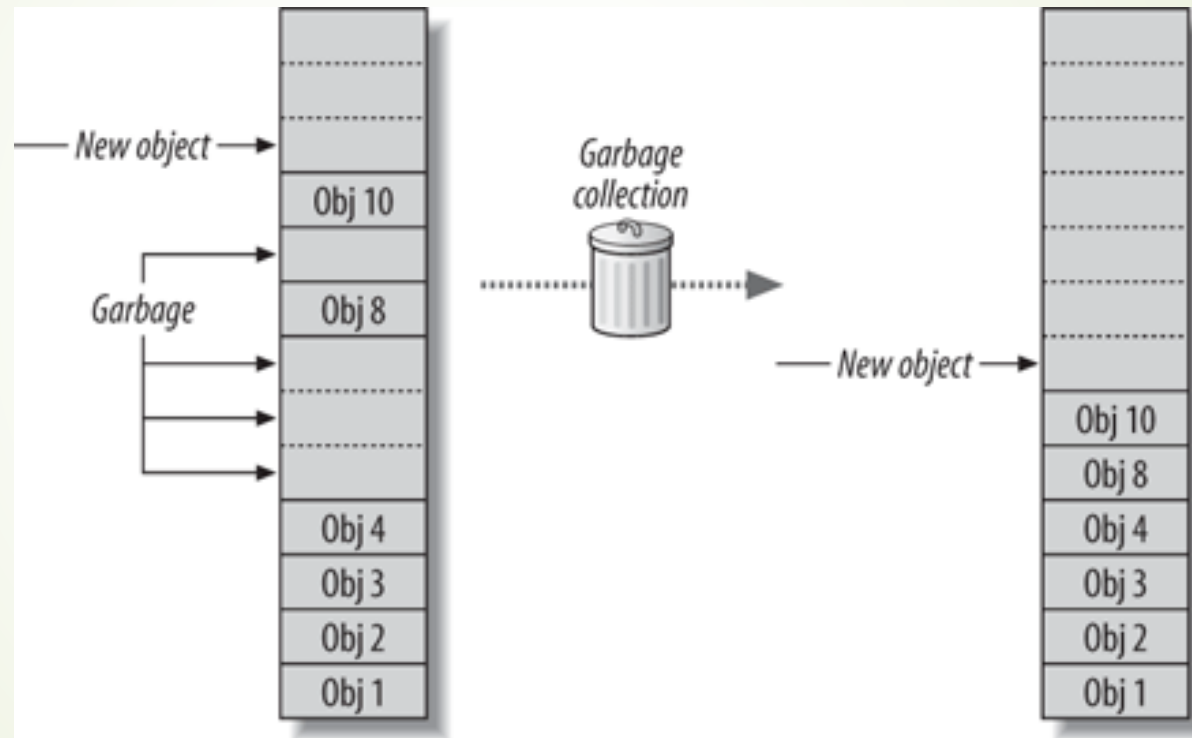


Все объекты делятся на 3 категории:

- Поколение 0. Это самое молодое поколение содержит короткоживущие объекты
- Поколение 1. Это поколение содержит коротко живущие объекты и служит буфером между короткоживущими и долгоживущими объектами
- Поколение 2. Это поколение содержит долгоживущие объекты.

Сборка мусора состоит из следующих этапов:


- Этап маркировки, выполняющий поиск всех используемых объектов и составляющий их перечень.
- Этап перемещения, обновляющий ссылки на сжимаемые объекты.
- Этап сжатия, освобождающий пространство, занятое неиспользуемыми объектами и сжимающий выжившие объекты. На этапе сжатия объекты, пережившие сборку мусора, перемещаются к более старому концу сегмента.



# Финализация ресурсов

```
class Car
{
    // destructor
    ~Car()
    {
        // очистка
        // ...
    }
}

protected override void Finalize()
{
    try
    {
        // вызов деструктора
    }
    finally
    {
        base.Finalize();
    }
}
```



```
class MyFile : IDisposable
{
    public virtual void Dispose()
    {
        ....
    }
}
...
var file = new MyFile("D://example.txt");
...
file.Dispose();
```

```
using (var variableName = new MyFile())
{
    ...
}
```

Интерфейс **IDisposable** может быть реализован и в классах и в структурах  
Даже многократный вызов **Dispose** должен происходить без ошибок!

# Generics (универсальные шаблоны)

Универсальными шаблонами являются классы, структуры, интерфейсы и методы, которые имеют прототипы (параметры типов) для одного или нескольких типов, которые они хранят или используют.

```
public class GenericList<T>
{
    public void Add(T input) { }
}

class TestGenericList
{
    private class ExampleClass { }
    static void Main()
    {
        var list1 = new GenericList<Int32>();
        list1.Add(1);
        var list2 = new GenericList<String>();
        list2.Add("");
    }
}
```

# List<T> (ArrayList)

```
var list = new List<char>();  
list.Add('h');  
list.Add('e');  
list.Add('l');  
list.Add('l');  
list.Add('o');  
list.Add(' ');  
list.Add('w');  
list.Add('o');
```

// [] Count: 0  
// Capacity: 0

// [h] Count: 1  
// Capacity: 4

// [h e] Count: 2  
// Capacity: 4

// [h e l] Count: 3  
// Capacity: 4

// [h e l l] Count: 4  
// Capacity: 4

// [h e l l o] Count: 5  
// Capacity: 8

// [h e l l o ] Count: 6  
// Capacity: 8

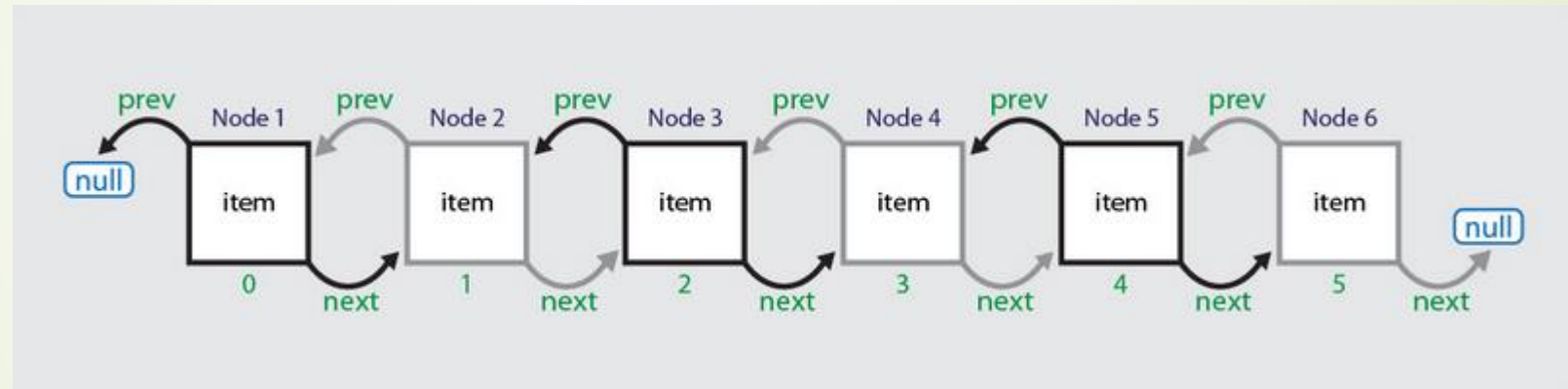
// [h e l l o w] Count: 7  
// Capacity: 8

// [h e l l o w o] Count: 8  
// Capacity: 8



# LinkedList<T>

- LinkedList<T> - двунаправленный список





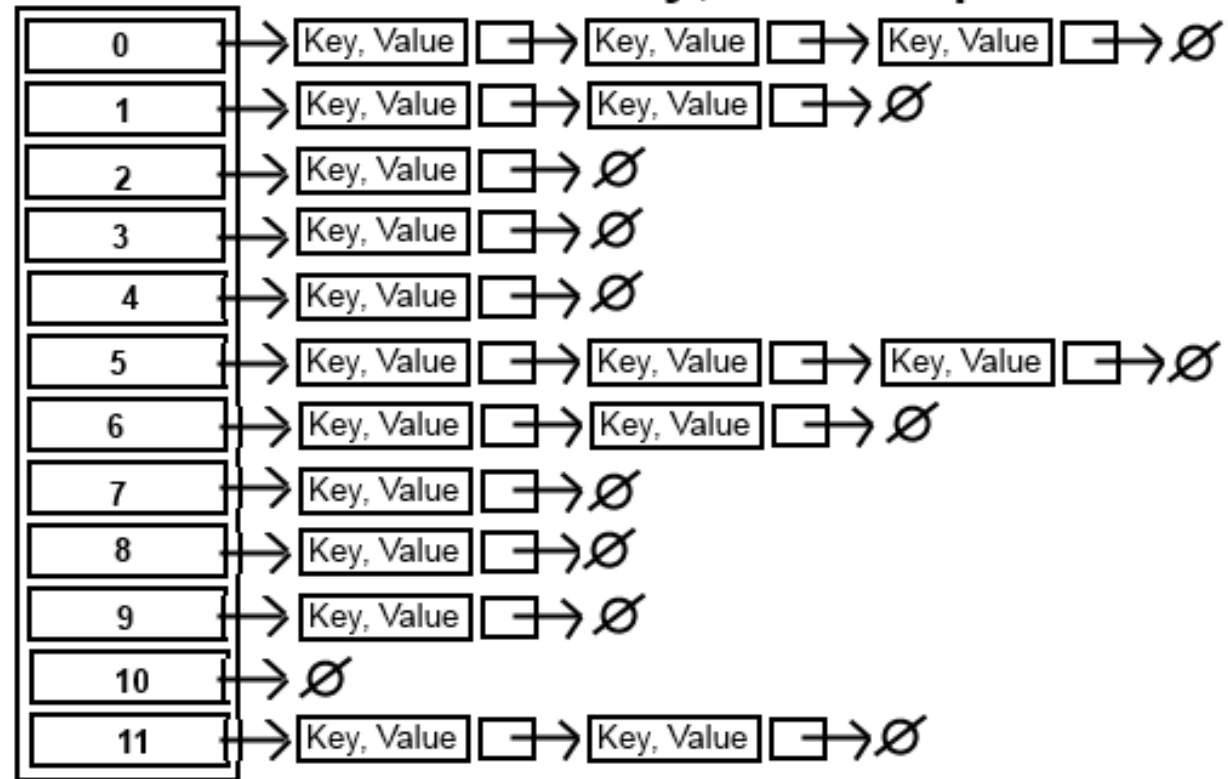
# KeyValuePair<TKey,TValue> (Структура)

- Определяет пару "ключ-значение", которая может быть задана или получена
  - TKey - тип ключа
  - TValue - тип значения
- Используется в ассоциативных массивах (словарях)

# Dictionary<TKey,TValue> (Hashtable)

- обеспечивает сопоставление набора ключей с набором значений. Каждый элемент, добавляемый в словарь, состоит из значения и связанного с ним ключа. Получение значения с помощью его ключа выполняется очень быстро, (в идеальном случае  $O(1)$ ), поскольку Dictionary<TKey,TValue> класс реализуется как хэш-таблица
- В целях перечисления каждый элемент словаря рассматривается как KeyValuePair<TKey,TValue> структура, представляющая значение и его ключ. Порядок, в котором возвращаются элементы, не определен.

## Index      Lists of Key, Value pairs





# Универсальные множества

- `HashSet<T>` - коллекция, которая не содержит повторяющихся элементов, элементы которой не имеют определенного порядка
- `SortedSet<T>` - поддерживает отсортированный порядок, не влияя на производительность по мере вставки и удаления элементов. Дублирующиеся элементы не допускаются. Изменение значений сортировки существующих элементов не поддерживается и может привести к непредвиденному поведению.