



## Лекция № 11



**IT Education  
Academy**

[WWW.ITEA.UA](http://WWW.ITEA.UA)

# C# Base

Урок 11

Структуры. Перечисления

## План урока

- Понятие структур и необходимость в них
- Создание структур и работа с ними
- Отличия структур от классов
- Понятие перечислений
- Область применения перечислений
- Упаковка-Распаковка (Boxing/UnBoxing)



# Понятие структур и необходимость в них

- Наряду с классами структуры представляют еще один способ создания собственных типов данных в C#. Более того многие примитивные типы, например, `int`, `double` и т.д., по сути являются **структурами**.
- Как и классы, **структуры** — это сущности для хранения данных, которые могут методы, поля, индексаторы, свойства. Но в отличие от классов, структуры являются типами значений и для них не выделяется память из кучи.
- Переменная типа **структура** напрямую хранит все свои данные, а переменная типа класс хранит ссылку на динамически выделяемый объект.

# Создание структур и работа с ними

- Главное отличие состоит в том, что при объявлении структуры используется ключевое слово **struct** вместо class. Ниже приведена общая форма объявления структуры:

```
struct имя : интерфейсы  
{  
    // объявления членов  
}
```

- Структуры особенно удобны для небольших структур данных.
- Хорошим примером структуры можно считать точку в системе координат.

```
struct Point  
{  
    public int x;  
    public int y;  
}
```

# Конструкторы

**Конструкторы структур**, как и конструкторы класса, вызываются с помощью оператора `new`. Но вместо того, чтобы динамически выделять объект в управляемой куче и возвращать ссылку на него, конструктор структуры возвращает само значение структуры (обычно сохраняя его во временном расположении в стеке), и затем это значение копируется туда, где оно нужно.

```
struct Point
{
    public int x;
    public int y;
    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}
```

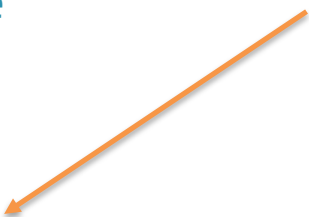
- При наличии пользовательского конструктора, все неинициализированные поля должны быть в нем инициализированы.
- Структурам запрещается иметь конструктор по умолчанию.

# Наследование

- Типы структуры не поддерживают определяемое пользователем наследование. Все типы структуры неявно наследуются от типа `ValueType`, который, в свою очередь, неявно наследуется от `object`.
- Структуры могут реализовывать интерфейсы.
- От структур наследоваться запрещено.
- Структура не может быть базовой для класса.

```
struct Point: IInterface
{
    public int x;
    public int y;
    public void Method()
    {
        Console.WriteLine();
    }
}

interface IInterface
{
    void Method();
}
```



# Вложенные структуры и классы

```
public struct MyStruct
{
    struct Nested
    {
        public void Method() {}
    }
}
```

## Nested struct

Структуры могут содержать **вложенные структуры**.

```
public struct MyStruct
{
    class Nested
    {
        public void Method()
        {}
    }
}
```

## Nested class

Структуры могут содержать **вложенные классы**.



# Поведение структур

- Структуры размещаются в стеке .
- При копировании структур, создаётся отдельная копия объекта, которая живёт «своей жизнью».
- От структур нельзя наследоваться.
- Структуры могут наследоваться только от интерфейсов.
- Не разрешается дополнение своими свойствами;
- При передаче структуры как параметра передается вся структура.
- В структуре нельзя создать конструктор по умолчанию.
- Структура уничтожается при выходе за пределы видимости.

# Перечисление

- Перечисление – определяемый пользователем целочисленный тип, который позволяет задавать набор допустимых значений, и назначить каждому понятное имя. Для объявления перечисления используется ключевое слово `enum` (от англ. enumeration).
- Перечисления наследуются от `Enum`, который наследуется от `ValueType` по этому они относятся к категории структурных типов.

```
enum Status
{
    Error = 10,
    InProgress,
    Done = 77
}
```

# Перечисление

- Перечислимый тип определяется как набор идентификаторов, с точки зрения языка играющих ту же роль, что и обычные именованные константы, но связанные с этим типом.
- По умолчанию в качестве базового для перечислений выбирается тип `int`, тем не менее перечисление может быть создано любого целочисленного типа, за исключением `char`. Для того чтобы указать другой тип, кроме `int`, достаточно поместить этот тип после имени перечисления, отделив его двоеточием.

```
enum BinaryDigit: byte
{
    Zero,
    One
}
```

Явно указан тип перечисления `byte`

# Преимущества перечислений

Обычно enum используется для того, чтобы ограничить диапазон входных значений, это в свою очередь избавляет от дополнительных проверок, и помогает избежать ошибок времени выполнения программы. К примеру светофор может иметь три состояния которые удобно описать с использованием перечисления:

```
enum TrafficLight
{
    Red,
    Yellow,
    Green
}
```

Также перечисления удобно использовать в операторе switch.

# Преимущества перечислений

- Перечисления облегчают сопровождение кода, гарантируя, что переменным будут присваиваться только легитимные, ожидаемые значения.
- Перечисления делают код яснее, позволяя обращаться к целым значениям, называя их осмысленными именами вместо малопонятных "магических" чисел.
- Перечисления облегчают ввод исходного кода. Когда вы собираетесь присвоить значение экземпляру перечислимого типа, то интегрированная среда Visual Studio с помощью средства IntelliSense отображает всплывающий список с допустимыми значениями, что позволяет сэкономить несколько нажатий клавиш и напомнить о возможном выборе значений.

## Упаковка-Распаковка (Boxing/UnBoxing)

- Когда любой значимый тип присваивается к ссылочному типу данных, значение перемещается из области стека в кучу. Эта операция называется упаковкой.
- Когда любой ссылочный тип присваивается к значимому типу данных, значение перемещается из области кучи в стек. Это называется распаковкой.
- Упаковка является неявной; распаковка является явной.
- Значимые типы легче ссылочных.

# Упаковка (Boxing)

**Упаковка** это преобразование value(значимого) типа в Object тип, преобразование является неявным.

```
static void Main()  
{
```

```
    //Значимый тип
```

```
    int item = 10;
```

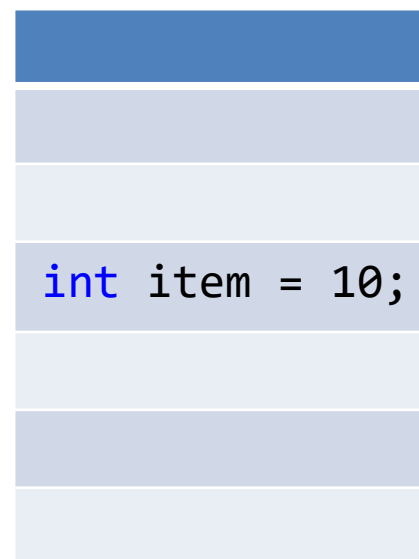
→ стек

```
    //Ссылочный тип
```

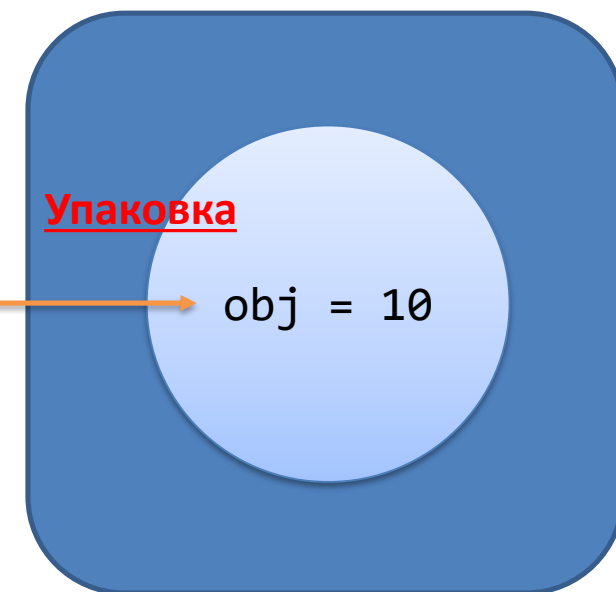
```
    object obj = item;
```

→ куча

стек



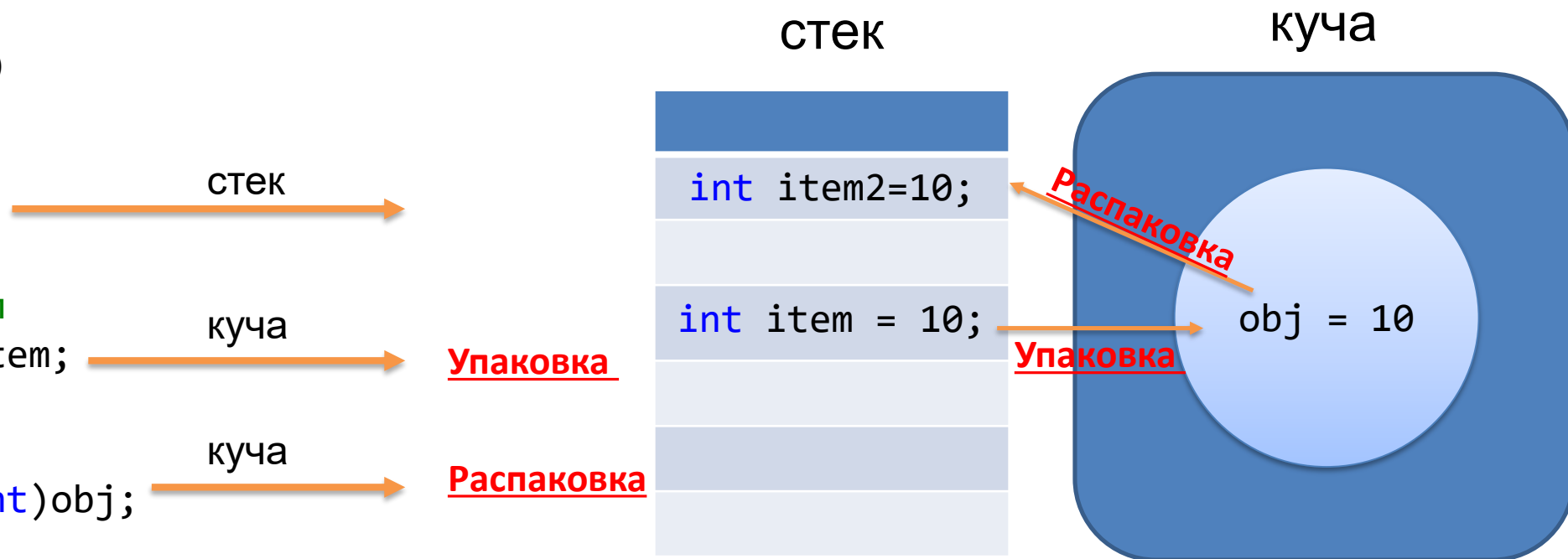
куча



# Распаковка (UnBoxing)

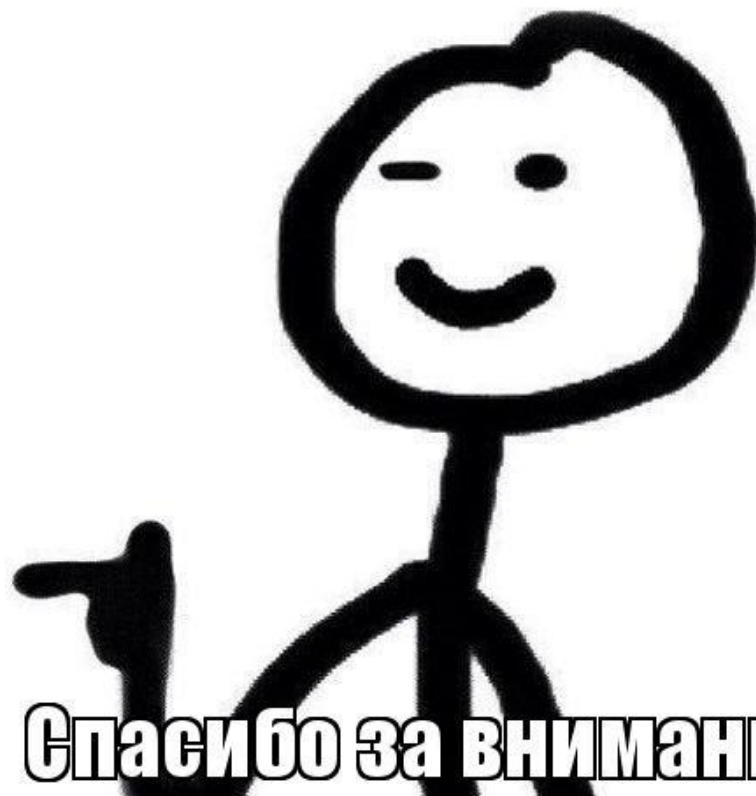
**Распаковка** это преобразование Object типа в value(значимый) тип.

```
static void Main()  
{  
    //Значимый тип  
    int item = 10;  
  
    //Ссылочный тип  
    object obj = item;  
  
    //Значимый тип  
    int item2 = (int)obj;  
}
```

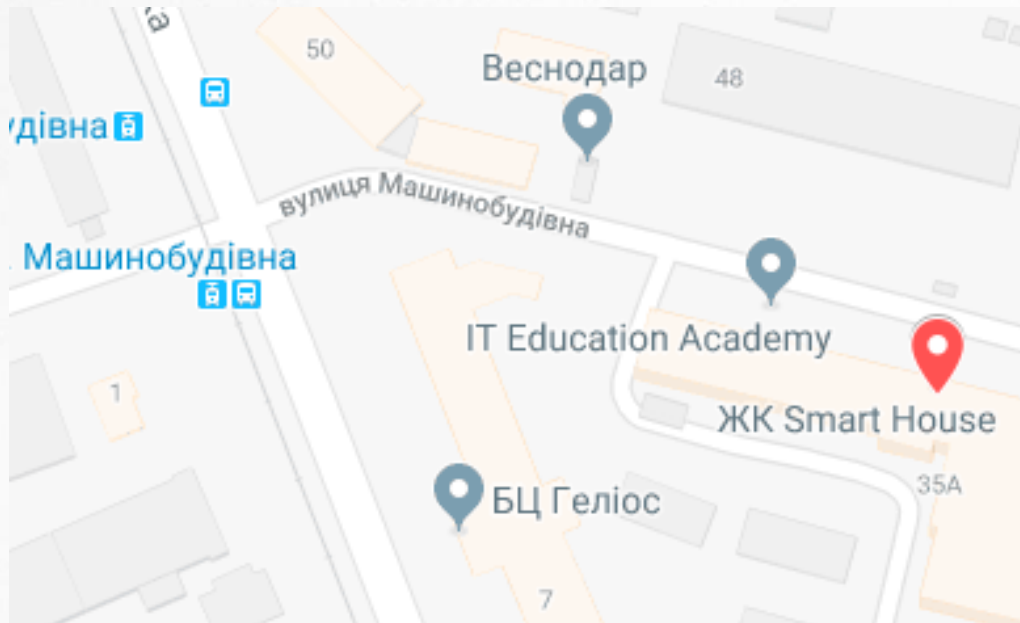




**Презентация окончена.**



**Спасибо за внимание!**



# КОНТАКТНЫЕ ДАННЫЕ

## ITEA

ЖК “Smart House”, ул.  
 Машиностроительная, 41  
 (м.Берестейская)

ЖК «Корона» улица  
 Срибнокильская, 1  
 м. Позняки

+38 (044) 599-01-79

facebook.com/Itea

info@itea.ua

itea.ua

