

# **Модуль 8: Создание и разрушение объектов**

# Жизненный цикл объекта

## ■ Создание объектов

- Вы выделяете память, используя ключевое слово **new**
- Вы инициализируете объект, используя конструктор

## ■ Использование объектов

- Вы вызываете методы

## ■ Удаление объектов

- Объект преобразуется обратно в память
- Память возвращается куче

# Обзор

- **Использование конструкторов**
- **Инициализация данных**
- **Объекты и память**
- **Управление ресурсами**

# ◆ Использование конструкторов

- Создание объектов
- Конструкторы в структурах
- Использование конструкторов по умолчанию
- Замена конструкторов по умолчанию
- Перегрузка конструкторов

# Создание объектов

## ■ Шаг 1: Выделение памяти

- Для выделения памяти из кучи используйте ключевое слово `new`

## ■ Шаг 2: Инициализация объекта через вызов конструктора

- Имя класса с круглыми скобками

```
Demo a = new Demo ();
```

```
Demo a;  
a = new Demo ();
```

# Конструктор

- **Конструктором** называется множество операторов кода, которому передается управление при создании объекта:
  - конструктор инициализирует объект при его создании.
- **Конструкторы:**
  - конструкторы класса (для статических классов)
  - конструкторы экземпляра класса (всех остальных классов).

# Конструкторы экземпляра

- Имя конструктора совпадает с именем класса.
- Синтаксис объявления конструктора аналогичен объявлению метода.

## Особенности конструктора:

- конструктор не имеет никакого возвращаемого спецификатора, даже `void`;
- в классе и в структуре можно объявлять множество вариантов конструкторов. Они должны отличаться списками параметров
- в структуре невозможно объявить конструктор с пустым списком параметров;
- если программист не указал ни одного конструктора или какие-то поля не были инициализированы, полям значимых типов присваивается нуль, полям ссылочных типов - значение `null`;
- не существует выражения вызова для конструктора, управление в конструктор передается посредством выполнения специальной операции `new`.

# Конструкторы в структурах

## ■ Компилятор

- Всегда создает конструктор по умолчанию, который инициализирует параметры структуры **0**, **false** или **null**.

## ■ Разработчик

- Может создавать конструкторы с одним или несколькими параметрами. При этом необходимо проинициализировать все поля структуры.
- Не может переопределить конструктор по умолчанию
- Не может объявить конструктор с модификатором доступа `protected`



# Использование конструкторов по умолчанию

## ■ Характерные черты конструктора по умолчанию:

- Модификатор доступа `public`
- Имя совпадает с именем класса
- Нет типа возвращаемого значения (даже не **`void`**)
- Не использует параметры
- Инициализирует все поля **`0`**, **`false`** или **`null`**

## ■ Синтаксис конструктора

```
class Date { public Date( ) { ... } }
```

# Замена конструкторов по умолчанию

- Конструкторы по умолчанию не всегда удовлетворяют требованиям программистов
  - В этом случае можно заменять их собственными

```
class Date
{
    public Date( )
    {
        ccyu = 1970;
        mm = 1;
        dd = 1;
    }
    private int ccyu, mm, dd;
}
```

# Перегрузка конструкторов

- **Конструкторы – это методы, значит их можно перегружать**
  - Принимают различные параметры
  - Возможность инициализации объекта несколькими способами
- **Если в классе определен хотя бы один конструктор, то конструктор по умолчанию автоматически не создается**

```
class Date
{
    public Date( ) { ... }
    public Date(int year, int month, int day) { ... }
    ...
}
```

# ◆ Инициализация данных

- Списки инициализации
- Объявление констант и переменных для чтения
- Инициализация readonly-переменных
- Закрытые конструкторы
- Статические конструкторы

# Использование ключевого слова **this** – модуль 6

- Ключевое слово **this** ссылается на объект, для которого вызывается метод
- Используется для избежания конфликтов между именами

```
class BankAccount
{
    ...
    public void SetName(string name)
    {
        this.name = name;
    }
    private string name;
}
```

# Списки инициализации

- Перегруженные конструкторы могут содержать повторяющийся код
  - Этого можно избежать, вызывая из одного конструктора другой конструктор этого же класса
  - В списках инициализации используется ключевое слово `this`

```
class Date
{
    ...
    public Date( ) : this(1970, 1, 1) { }
    public Date(int year, int month, int day) { ... }
}
```

# Объявление констант и переменных для чтения



**compile time**

- Значение константы должно быть известно на этапе компиляции

- Readonly-переменные инициализируются на этапе выполнения



# Инициализация readonly-переменных

- Три способа инициализации readonly-переменных
  - Использовать значения по умолчанию (0, false, null)
  - Непосредственно при объявлении
  - В теле конструктора

```
class SourceFile
{
    private readonly ArrayList lines;
}
```



# Закрытые конструкторы

## ■ Закрытый конструктор предотвращает создание ненужных объектов

- Нельзя вызывать методы экземпляра объекта
- Можно вызывать статические методы

```
public class Math
{
    public static double Cos(double x) { ... }
    public static double Sin(double x) { ... }
    private Math( ) { }
}
```

# Статические конструкторы

## ■ Назначение

- Вызывается загрузчиком класса на этапе выполнения
- Может использоваться для инициализации статических переменных
- Гарантированно будет вызван до создания хотя бы одного объекта класса

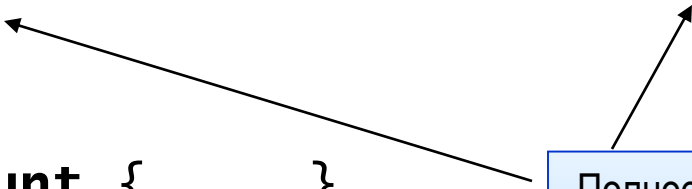
## ■ Ограничения

- Нельзя вызвать явно
- Нельзя объявлять с модификатором доступа
- Не может использовать параметры

# Создание вложенных классов – модуль 6

- Классы можно вкладывать в другие классы

```
class Program
{
    static void Main( )
    {
        Bank.Account yours = new Bank.Account( );
    }
}
class Bank
{
    ... class Account { ... }
}
```



Полное имя вложенного  
класса включает в себя  
имя внешнего класса

# Доступ к вложенным классам – модуль 6

- Вложенные классы также могут объявляться как **public** или **private**

```
class Bank
{
    public class Account { ... }
    private class AccountNumberGenerator { ... }
}
class Program
{
    static void Main( )
    {
        Bank.Account                accessible; ✓
        Bank.AccountNumberGenerator inaccessible; ✗
    }
}
```

# Лабораторная работа 8.1: Создание объектов

