КЛАССЫ

Классы *(как и структуры)* предназначены для объявления **типов**.
Отличие классов от структур состоит в том, что классы между собой поддерживают отношения наследования (структуры поддерживают наследование только от интерфейсов).

Разновидностью класса являются **специализированные** типы

- интерфейсы;
- делегаты.

Ориентированы на расширение только **функциональности** разрабатываемого типа (*методы и свойства*), но при этом позволяют реализовать ***множественное*** наследование

На основе делегатов программируется **реакция** систем на разнообразные внешние воздействия или события

Объявление класса и создание его экземпляров

```
class A{}

A o1 = new A();
A o2 = new A();
A o3 = o2;
```

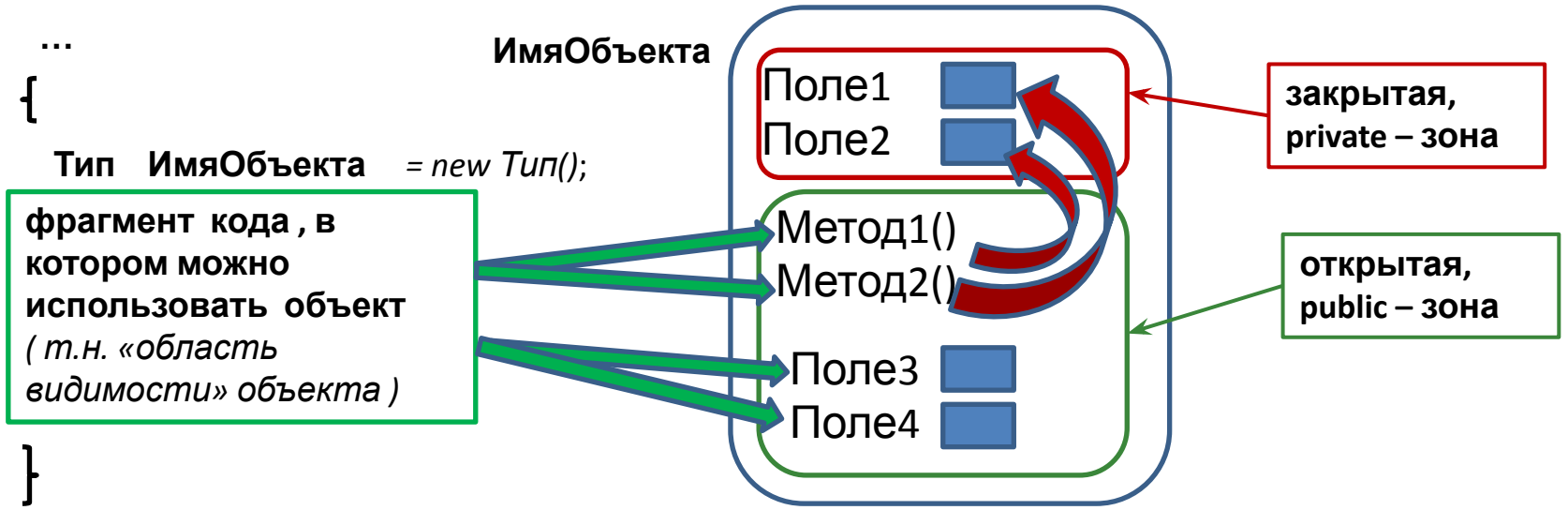
Элементы класса

Элементами класса могут быть ***поля, методы и события***.

Поля – пассивная, но при этом, ***основная*** часть класса *(предназначены* для ***размещения данных)***.

Методы – активная, но при этом ***вспомогательная*** часть класса: задача методов – ***обслуживание полей*** *(инициализация, модификация, представление и другие действия, связанные с обработкой информации)*.

События – это объекты *типа* ***делегат***, предназначенные для размещения ссылок ***на метод*** *или* ***методы*** *(с помощью событий обеспечивается т.н. обратный вызов соответствующих методов*)



Методы *(как элементы класса)*

делятся на:

- ***особые***
  - *Конструкт**оры** - может быть несколько,* предназначены для создания экземпляров класса*;*
  - *Деструкт**ор** - всегда один,* предназначен для уничтожения экземпляра класса*;*
- ***неособые***.

Особые методы могут быть в режиме ***по умолчанию:***

Конструктор класса ***по умолчанию*** выполняет инициализацию полей объекта предустановленными значениями *(значениями по умолчанию)*. Но уже **первый *явный*** конструктор фактом своего появления в классе **отменяет** конструктор ***по умолчанию***. Явный ***деструктор***, соответственно, отменяет деструктор ***по умолчанию***.

Объявление конструктора:

модификатор Имя (сигнатура) { тело }

- ***конструктор*** не может, *в принципе*, иметь **возвращаемого** значения, а, значит и не имеет **типа, ИМЯ** конструктора – это всегда **ИМЯ** класса,
- ***деструктор также** не имеет типа, дополнительно деструктор не имеет и входных аргументов. Имя **деструктора** – это **имя** класса с точностью до «тильда» (~).

И ***класс***, и ***структура***, могут иметь неограниченное количество

Режимы защиты элементов класса

устанавливаются **для каждого** элемента класса модификаторами доступа:

- **public** – **общедоступный** элемент класса (нет защиты):
  доступен в любом месте области видимости объекта класса;

- **protected** – **защищенный** элемент класса:
  доступен только элементам данного классса и производного класса.

*Примечание: спецификатора **protected** не используется в структурах,*
*так как структурные типы **не поддерживают** наследования;*

- **private** – частный элемент класса: доступен только элементам данного класса *(в режиме «по умолчанию»)* действует защита **private**

При отсутствии у элемента класса *(или структуры)* явного модификатора

Объявление и реализация структуры и класса, конструкторы по умолчанию, поля

```
using System;
   class Primer
   {
      struct A
      {
         public   string      s;
         public          bool        a;
         public          int         b;
      }
      class B
      {
         public   string      s;
         public   bool        a;
         public   int    b;
      }
      static void Main()
      {    A  obj ;                    //Конструктор по умолчанию создаёт экземпляр
       obj.a = true;                  //структурного типа:  поля конструктором не   obj.b =
10;                  //инициализируются!
              obj.s = "Объект типа A";
              Console.WriteLine("{0} a={1}   b={2}", obj.s, obj.a, obj.b);
          B   obj1 = new B(); //Выделение памяти и вызов конструктора по
умолчанию   obj1.s = "Объект типа B";
```

В консольном окне:

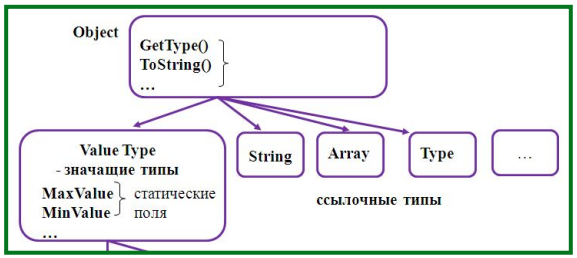
```
Объект типа A a=True b=10
Объект типа B a=False b=0
```

Пример *(объявление класса и создание трёх его экземпляров, знакомство с переопределением)*

```
class A
  {  int a;
     public A( int ia)
        {
           a = ia;
        }
      public override string ToString()  //реализация  переопределения   метода ToString
        {
           return String.Format ( "a={0} ", a++ ) ;
        }
       static void Main()
        {
           A    o1 = new A ( 1 ),
                o2 = new A ( 1 ),
                o3 = o2;
           Console.WriteLine ( o1 );       //Использование переопределения
           Console.WriteLine ( o2 );
           Console.WriteLine ( o3 );
        }
  }
```

```
a=1
a=1
a=2
```

Краткая справка по **object**



object – это псевдоним для *супербазового* класса ***System.Object***.

Класс ***object*** имеет один конструктор и семь методов.

Статические методы:

Equals – проверка равенства двух экземпляров ***object***,

ReferenceEquals – проверка на совпадение двух экземпляров ***object,***

Нестатические методы:

GetType – возвращает тип экземпляра,

MemberwiseClone – выполняет поверхностное копирование текущего объекта,

Finalize – освобождение ресурсов или иная зачистка перед утилизацией объекта (***virtual*** *- виртуальный*),

GetHashCode – создание числа (хэш-кода), соответствующего значению объекта (***virtual***),

ToString – возвращение значения объекта в виде строки (***virtual***).

Как следует из модификаторов, методы ***Finalize, GetHashCode и ToString*** в пользовательских типах целесообразно переопределять