Основы программирования

Основы ООП

Члены класса
Конструкторы

Конструкторы класса

Конструктор экземпляра является членом, реализующим действия, необходимые для инициализации экземпляра класса.

(Спецификация языка C#)

Конструктор класса

Он должен правильно и корректно задать начальные значения полей объекта.

Конструктор вызывается тогда, когда экземпляр класса создается с помощью операции `new`.

Конструктор может задавать разные значения полей для разных экземпляров.

Синтаксис

```
public имя_класса ([аргументы])
{
  // тело конструктора
}
```

Особенности конструктора класса

- имя конструктора совпадает с именем класса;
- у конструктора нет возвращаемого значения (даже `void`);
- в классе может быть несколько конструкторов с разными параметрами для разных инициализаций (перегруженный конструктор), все конструкторы должны иметь разные сигнатуры;
- если в конструкторе каким-либо полям не задаются значения, то они инициализируются значениями по умолчанию – `false` для булевских полей, `0` – для числовых полей, `null` – для ссылочных полей;

Особенности конструктора класса

- если в классе не описан ни один конструктор, то автоматически в состав класса добавляется конструктор по умолчанию, который задает полям значения по умолчанию;
- пользователь может описать обычный конструктор без параметров (также называемый конструктором по умолчанию) и задать в нем собственную инициализацию полей;
- конструктор должен иметь спецификатор доступа шире, чем `private`.

...

- Если в классе нет ни конструктора по умолчанию, ни конструктора без параметров, значит, в классе пользователем описан как минимум один конструктор с параметрами.
- Конструктор может вызвать на исполнение другой конструктор текущего класса. Такой вызов удобен, если один конструктор инициализирует те поля, которые еще не были заданы в другом.

Пример 1

```
class MyClass {
   int a, b;
   double c;

   public MyClass()   {
      a = 1;
      b = 1;
      c = 1.1;
}
```

Пример 2

```
public MyClass(int a)
{     this.a = a;              }
/*public MyClass(int b)
{     this.b = b;              }*/

public MyClass(double c)
{     c = c;                        }
public MyClass(int a, double c)
{     this.a = a; this.c = c;   }
```

Пример 3

```
public MyClass(int a, int b)
{    this.a = a;    this.b = b;    }

public MyClass(int a, int b, double c)
{    this.a = a; this.b = b; this.c = c;   }
}
```

Все приведенные конструкторы имеют разные сигнатуры. Для сигнатуры важны не имена параметров, а их тип и порядок следования.

Вызов конструктора из другого конструктора

перепишем предыдущий пример:

Пример

```
class MyClass
{
  int a, b;
  double c;

public MyClass(int a)   {
  this.a = a;           }

public MyClass(int a, int b): this(a)      {
  this.b = b;                              }

public MyClass(int a, int b, double c): this(a, b)     {
      this.c = c;                                      }
}
```

Инициализаторы

Конструкция после «:» вызывается на исполнение до начала исполнения тела конструктора и называется *инициализатором*.

Таким образом, поля, значения которых задаются в инициализаторе, нет необходимости инициализировать еще раз в самом конструкторе.

Циклические взаимные вызовы конструкторов отслеживаются на этапе компиляции.

Количество конструкторов

Количество конструкторов в классе в общем случае не ограничивается, однако все же стоит руководствоваться здравым смыслом и необходимостью описания того или иного конструктора.

Порядок инициализации полей

Иногда при описании полей явным образом задается их начальная инициализация, в то время как в конструкторе может быть задана другая инициализация тем же полям.

Порядок инициализации полей в такой ситуации:

- при вызове конструктора сначала инициализируются все поля, для которых начальное значение было задано явно при описании, затем выполняется тело конструктора, то есть задаются значения полей, указанные в конструкторе.

Конструктор «по умолчанию» и проинициализированные поля

Конструктор, называемый конструктором «по умолчанию» (добавляемый системой в случае полного отсутствия конструкторов в классе) не изменяет значения заранее явно проинициализированных полей.

То есть не сбрасывает значения заданных пользователем полей их в значения «по умолчанию».

Такая тактичность не присуща конструкторам «по умолчанию», написанным самим пользователем, то есть конструкторам без параметров.

Пример

```
1  class Room {
2      public double length; //длина комнаты
3      public double height; //высота комнаты
4      public double width; //ширина комнаты
5      public bool windows; //есть ли окна?
6  }
```

```
1  Room myRoom = new Room();
```

```
1  myRoom.length = 7;
2  myRoom.height = 2.5;
3  myRoom.width = 4;
4  myRoom.windows = true;
```

Пример. Конструктор без параметров

```
1  public Room() {
2      length = 7;
3      height = 2.5;
4      width = 4;
5      windows = false;
6  }
```

```
1  Room newRoom = new Room();
2  Console.WriteLine(newRoom.length);//выведет 7
```

Пример. Конструктор с параметрами

```
1  public Room(double a, double b, double c, boolean d) {
2      length = a;
3      height = b;
4      width = c;
5      windows = d;
6  }
```

```
1  Room newRoom = new Room(10,3,4,true);
2  Console.WriteLine(newRoom.length); //выведет 10
```

Источники

1. METANIT.COM. C#/.Net: Полное руководство по языку программирования C# 8.0 и платформе .NET Core 3. https://metanit.com/sharp/tutorial/
2. METANIT.COM. C#/.Net: Алгоритмы и структуры данных в C#. https://metanit.com/sharp/algoritm/