Особенности C#

Индексаторы, события, частичные методы, расширяющие методы, сборщик мусора

DraggonZ

Индексаторы

Индексатор позволяет индексировать элемент подобно массиву

Синтаксис

```
тип_элемента this[int индекс] {
// Аксессор для получения данных,
get {
// Возврат значения, которое определяет индекс.
}
// Аксессор для установки данных,
set {
// Установка значения, которое определяет индекс.
}
}
```

Частичные типы

Использует ключевое слово
partial

Применяется для разнесения одного класса, интерфейса, структуры по разным файлам

Расширяющие методы

Расширяющие методы позволяют существующим типам получать новую функциональность без необходимости непосредственного изменения расширяемого типа

Синтаксис

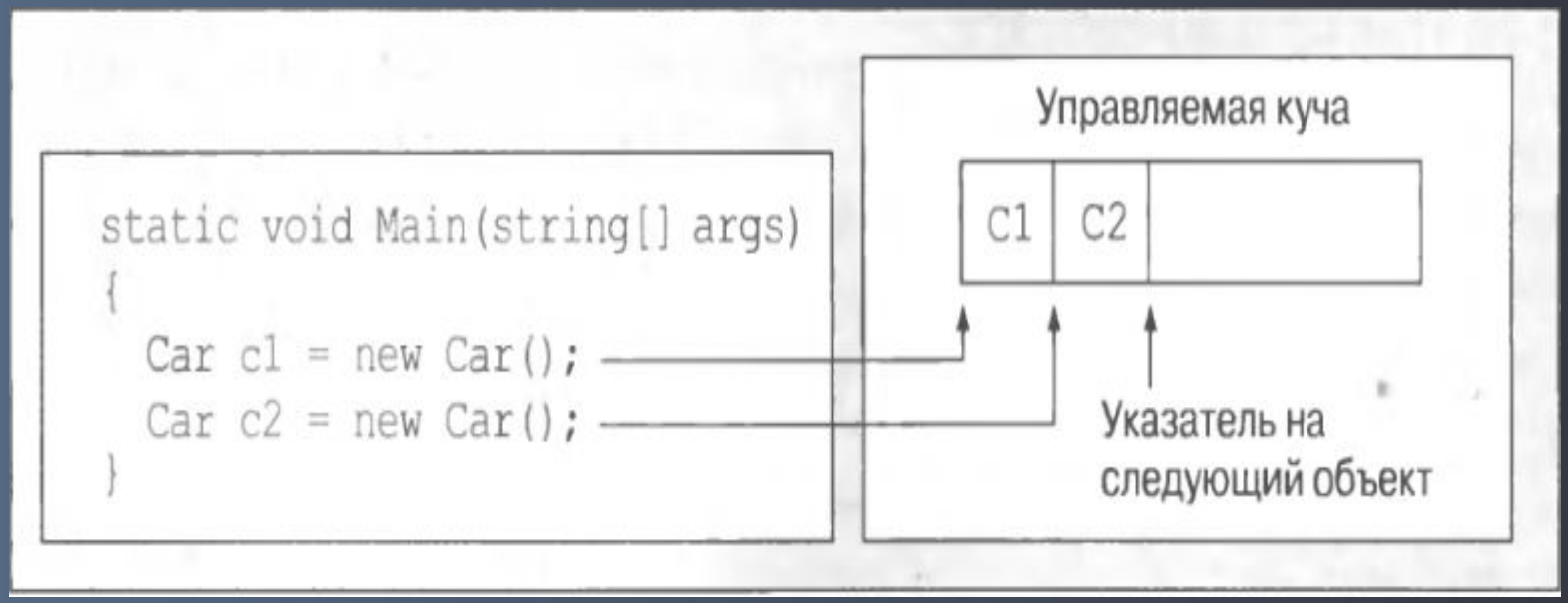
```
static  class MyExtensions
{
  public static возвращаемое_значение
ExtensionMethod(this тип_объекта
объект,…)
   {
      //Определение расширяющего метода
   }
}
```

Расширяющие методы являются синонимами обычных статических методов

Сборщик мусора

Сборщик мусора удаляет объект из кучи тогда, когда тот становится недостижимым ни в одной части программного кода

Размещение объекта в управляемой куче



В случае нехватки в управляемой куче пространства для размещения запрашиваемого объекта начинает выполняться сборка мусора

Сборщик мусора использует две отдельных кучи, одна из которых предназначена специально для хранения очень больших объектов

Каждый объект в куче относится к одному из перечисленных ниже поколений:

поколение 0: новые объекты;
поколение 1: объекты, пережившие 1 сборку мусора;
поколение 2: объекты, пережившие более 1 сборки мусора.

Системный класс System.GC позволяет программно взаимодействовать со сборщиком мусора

События

События строятся с помощью ключевого слова event

События используются в качестве сокращения, избавляющего от необходимости строить специальные методы для добавления и удаления методов в списке вызовов делегата

Синтаксис

```
class MyClass
{
    public delegate void MyDelegate(int x);
    public event MyDelegate MyEvent;
}
```

Рекомендуемый шаблон делегата, лежащего в основе события

```
void обработчик(object отправитель,
EventArgs e) {
// ...
```

Для управления списком обработчиков событий служит расширенная форма оператора event, позволяющая использовать **аксессоры событий**

```
event делегат_события имя_события {
add {
// Код добавления события в цепочку событий.
}
remove {
// Код удаления события из цепочки событий.
}
}
```

Учитывая, что очень много специальных делегатов принимают объект в первом параметре и наследников EventArgs — во втором, часто используется обобщенный тип EventHandler<T>, где T — специальный тип-наследник EventArgs.

Спасибо!