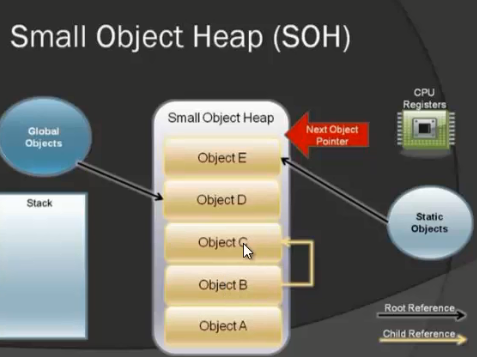
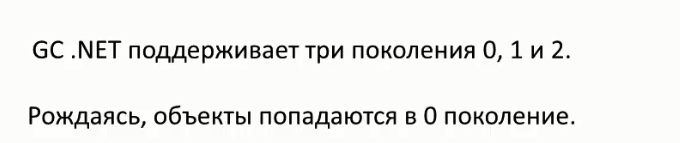


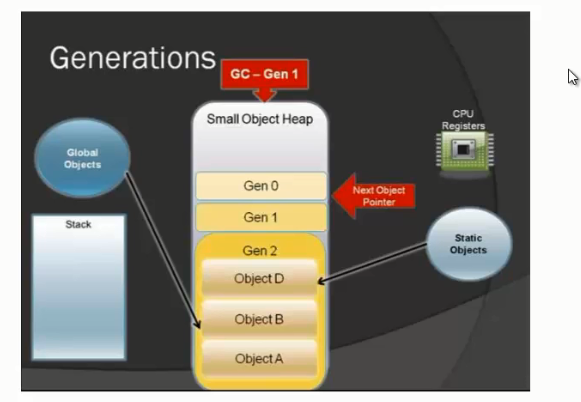


Куча для небольших объектов





Сборщик мусора проходит по объектам в куче и проверяет имеются ли ссылки на них, если да, то они переходят в другое поколение и проводится дефрагментация кучи, в связи с этим объект А удалять слишком трудоемко, т.к. все объекты нужно будет скопировать на новые места, а это может занять большое количество времени



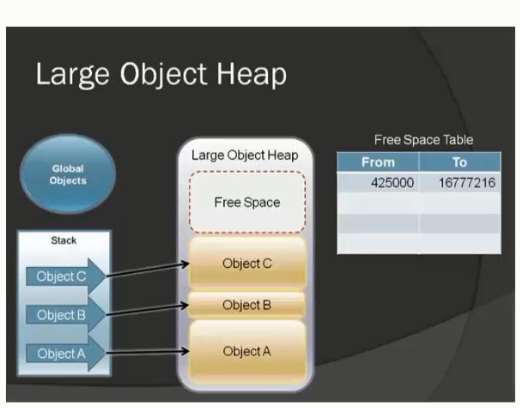
Кучи для больших объектов

На данной куче не выполняется дефрагментация

В кучу для больших объектов попадают объекты более 65кб

Любые объекты размером 85 000 байт и более считаются

большими1.



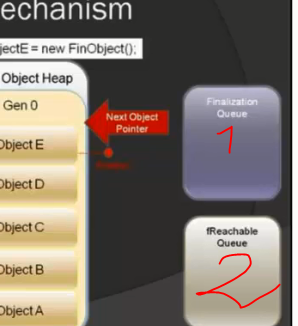
Т.к. не выполняется дефрагментация вместо NextObjectPointer (указателя на следующий свободный байт адреса памяти) хранится таблица с размером свободного пространства на куче



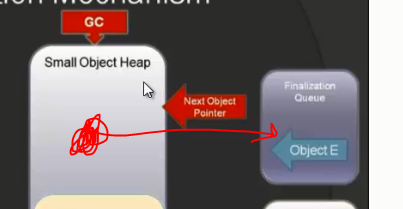
Финализация



Имеются две очереди у механизма финализации



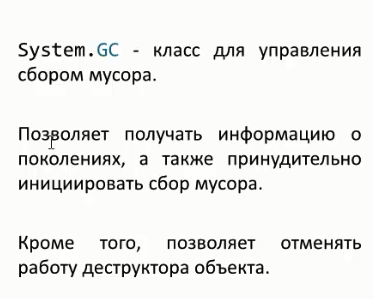
При потери ссылки на объект GC помещает эти объекты в очередь финализации



GC

Collect() –принудительный запуск для сборки мусора (в том случае если

1 необходимо выполнить что то важное прерывание чего не допустимо)



GC.GetTotalMemory(false) - возвращает размер памяти в байтах которую занимают объекты в управляемой куче.

Этот метод принимает параметр указывающий, запускать или нет процесс сборки мусора.

GC.MaxGeneration - возвращает максимальное количество поколений, поддерживаемое данной системой

GC.GetGeneration(car) - возвращает поколение, к которому относится данный объект.

GC.Collect() - дает указание сборщику муссора проверить объекты определенного поколения (В данном случае - 2) значит чистит и 0 и 1 и 2

GC.WaitForPendingFinalizers(); - приостанавливает выполнение текущего потока, пока

не будут отработаны все объекты, предусматривающие финализацию

GC.CollectionCount(0) – Сколько раз поколение 0 проверялось.

Отменяет работу деструктора для данного класса.!!!!!

GC.SuppressFinalize(this); устанавливает флаг запрещения завершения для объектов которые в противном случае могли бы быть завершены сборщиком мусора.

Работа финализатора

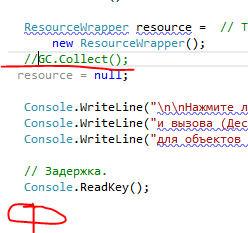
Вызывается в отдельном потоке, поэтому вызывается с задержкой и если будет разделяемый ресурс КОНСОЛЬ и будет в диструкторе использоваться консоль как и в основном потоке результат может быть неочевиден

(касается Release)!!!!!!!!!!!!!!!!!!!!!!!!!!

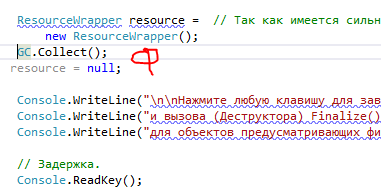
ResourceWrapper resource = new ResourceWrapper(); - Так как имеется сильная ссылка, сразу финализация не происходит.

Финализация объекта происходит тогда когда будет недостача памяти ЕЛСИ ТОЛЬКО принудительно не вызываем GC.Collect();

Финализатор будет вызван в конце, несмотря на то что ссылку мы уже на объект потеряли

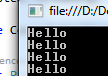


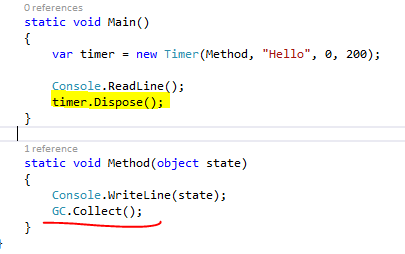
А при принудительной сборке мусора сразу же будет вызван

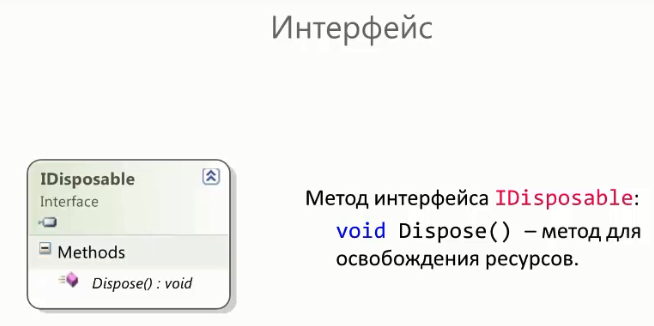


В данном случае при Releаse объект таймер сработает один раз т.к. на него не ссылок и его сборщик мусора удалит (а если использовать Dispose() то он проанализирует и решит что мы еще используем данный объект и удалять не будет)

 без Dispose()

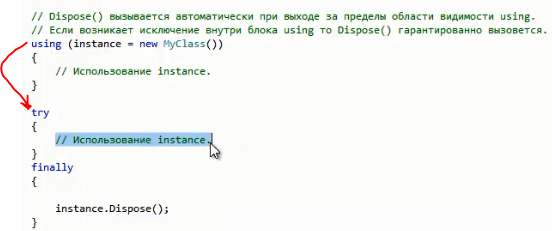
с Dispose()





Работает using

Как Try finally



Чтобы не удалял раньше времени в Релизе

public static void Main() {

// Создание объекта Timer, вызывающего метод TimerCallback каждые 2000 мс

Timer t = new Timer(TimerCallback, null, 0, 2000);

// Ждем, когда пользователь нажмет Enter

Console.ReadLine();

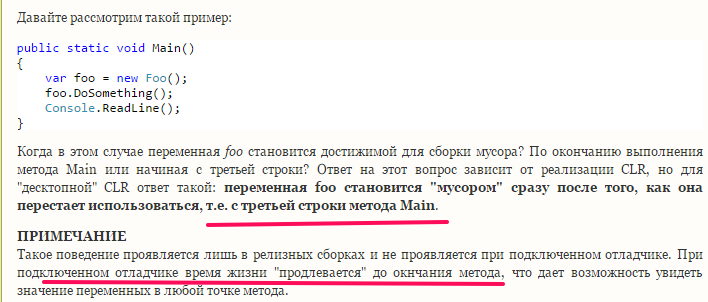
// Создаем ссылку на переменную t после ReadLine

// (t не удаляется уборщиком мусора

// до возвращения управления методом Dispose)

t.Dispose();

}



ИНОГДА Удаляет объект до завершения методов

Приводит к ошибке

