Для оптимизации поиска недостижимых объектов каждый объект в куче относят к некоторому «поколению». Чем дольше объект существует в куче, тем больше вероятность того, что он там останется. Есть 3 поколения:
Поколение 0: Новый созданный объект, который никогда не помечается для сборки.
Поколение 1: Объект, уже переживший сборку мусора (был помечен для сборки, но удален не был)
Поколение 2: Объект, переживший более одной очистки памяти сборщиком мусора.
Сначала исследуется поколение 0, если в результате очистки этих объектов получено необходимое количество памяти, то все неудаленные объекты переходят в поколение 1. Если места все же не хватает, проверяется поколение 1. Пережившие сборку в поколении 1 объекты переходят в поколение 2. В последнюю очередь проверяется поколение 2. Объекты, пережившие сборку здесь, поколения не меняют.
Предназначение сборщика мусора – управление памятью вместо нас. Принудительная сборка мусора осуществляется GC.Collect() в следующих случаях:
- приложение входит в блок кода, который не следует прерывать сборкой мусора
- приложение завершило создание большого количества объектов и нужно очистить как можно больше памяти.
Явная сборка мусора выглядит так:

```
Static
void Main(string[] args)
{GC.Collect();GC.WaitForPendingFinalizers();/*всегда вызывается, чтобы удостоверится, что
финализируемые объекты имели шанс выполнить действия для очистки до того, как
программа продолжит действие. */}
```

В качестве параметра метод GC.Collect(); может принимать номер поколения, до которого следует произвести очистку.

В базовом классе System.Object имеется виртуальный метод Finalize(). Переопределение этого метода дает пользовательскую логику по очистке объекта. Так как в базовом классе этот метод является защищенным, вызывать его из экземпляра класса недопустимо. Этот метод используется сборщиком мусора перед удалением объекта из памяти.
Переопределять Finalize() в типах структур нельзя, так как структуры являются типами значений, которые изначально *никогда не располагаются* в управляемой памяти, т.е. никогда не подвергаются процессу сборки мусора. Если необходимо создать структуру содержащую ресурсы, нуждающиеся в очистке, вместо этого метода реализуют интерфейс IDisposable.
Метод Finalize() будет вызываться автоматически при сборке мусора (автоматической или ручной), а так же при выгрузке из памяти домена, который отвечает за обслуживание приложения. Единственный случай, когда необходимо создание класса, способного производить после себя очистку, является работа с неуправляемыми ресурсами. Внутри .net неуправляемые ресурсы появляются, когда в результате непосредственного вызова api-интерфейса операционной системы с помощью PInvoke или применения очень сложных сценариев взаимодействия с COM.
Обычным образом Finalize() перегрузить нельзя (ошибка компиляции). Определяются вместо этого деструкторы так же, как и в плюсах (~<ClassName>), никогда не снабжаются модификаторами доступа, так как по дефолту являются защищенными, не содержат параметров и не могут быть перегружены (один класс – один деструктор).
Альтернатива Finalize() является реализация классом интерфейса IDisposable и его единственного метода Dispose(). В случае реализации этого интерфейса, предполагается, что пользователь должен вручную вызывать метода Dispose(), прежде, чем объектной ссылке будет позволено покинуть область действия. При вызове данного метода, объект все еще будет существовать в управляемой куче, и иметь доступ ко всем остальным, находящимся там объектам.