Факултет Техничких Наука
Универзитет у Новом Саду

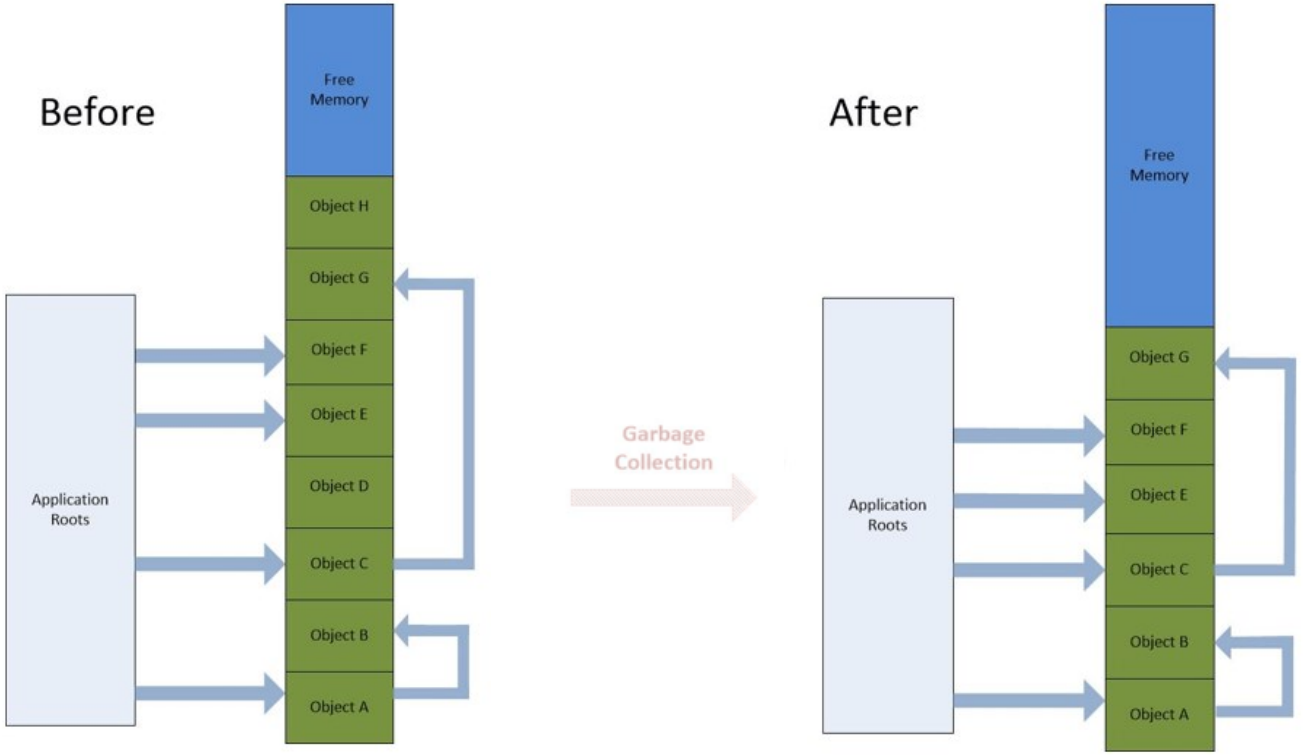
Виртуелизација процеса

Garbage Collector (GC)

Small Object Heap (SOH)

Нови Сад, 2023.

Пример рада *GC*-а



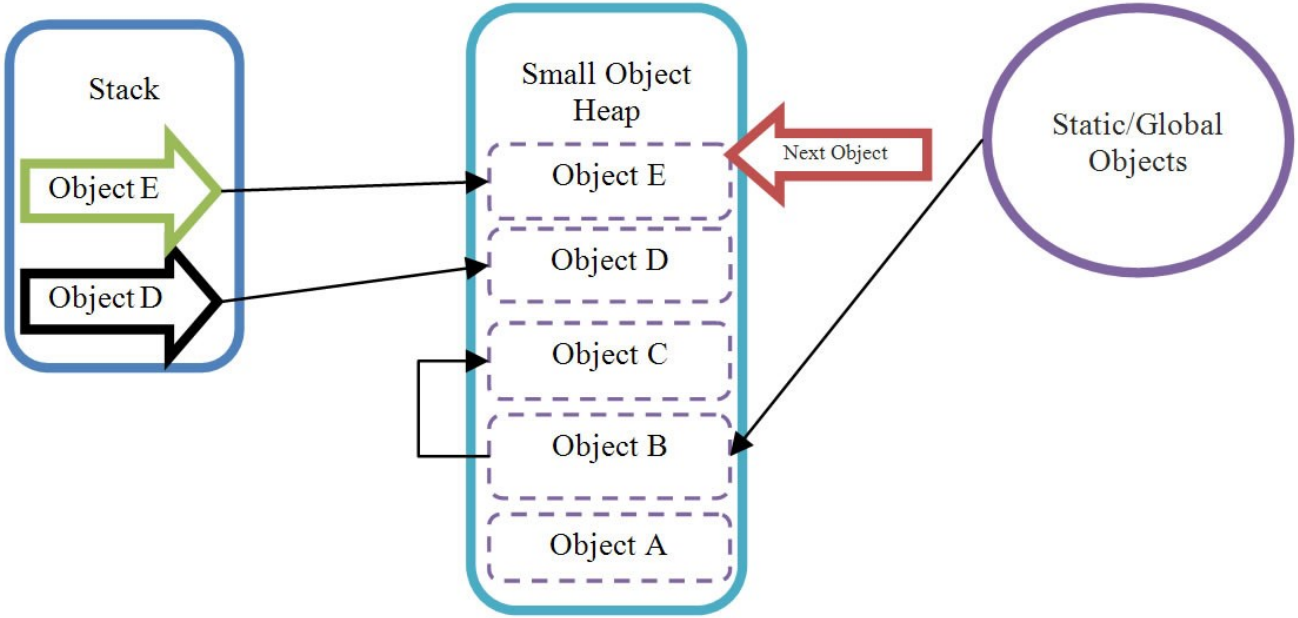
Small Object Heap (SOH)

- Алокација и аутоматско сакупљање смећа на *SOH*-у је прилично сложен процес
- Сви објекти величине до 85KB се смјештају на *SOH*
- Пошто је већина објеката који се креирају током рада апликације мања од 85KB, *SOH* се прилично користи
- Потенцијлано постоји проблем са креирањем листе објеката који су у употреби и сажимањем *heap*-а, посебно ако је велики

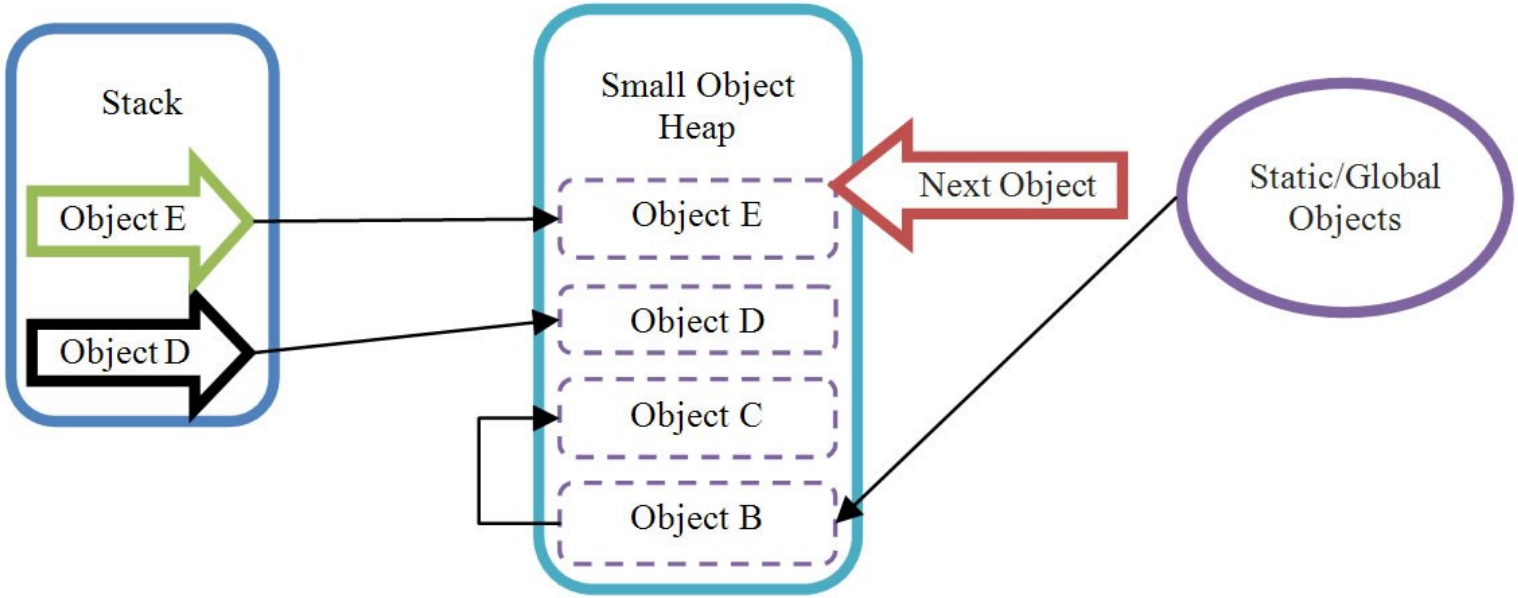
Small Object Heap (SOH)

- Сакупљање објеката са *SOH*-а укључује сабијање
- Разлог сабијања је чињеница да је *SOH* непрекидна гомила гдје се објекти додају узастопно један за другим
- Када дође до сабијања преживјели објекти попуњавају празнине које су за собом оставили они које је *GC* уклонио
- Шта је предност овога приступа, а шта мана?

Принцип рада *GC*-а на *SOH*-у



Принцип рада *GC*-а на *SOH*-у



Генерације *GC*-а

- Брже је сабити меморију за дио *heap*-а него за цели *heap*. Зашто*?*
- Нови објекти имају краћи век трајања, а старији дужи. Зашто?
- *Heap* се дели у три генерације како би се независно руковало краткорочним и дугорочним објектима
- Постоје три генерације:
  - Генерција 0
  - Генерација 1
  - Генерација 2
- На овај начин се смањује број коренских референци и хијерархија које је потребно погледати

Генерације *GC*-а

- Генерација 0 – најмлађа је и садржи краткотрајне објекте. Пример краткотрајног објекта је привремена променљива. Сакупљање смећа се најчешће дешава у овој генерацији.

- Генерација 1 - садржи краткотрајне објекте и служи као тампон између објеката кратког века и дуговечних објеката.

- Генерација 2 - садржи дуговечне објекте. Пример дуговечног објекта је објекат у серверској апликацији који садржи статичке податке који су активни током процеса.

Генерација 0

- Генерација 0 – најмлађа је и садржи краткотрајне објекте
- Већина објеката се креира у оквиру позива метода и бришу се када се метода заврши
- Пример краткотрајног објекта је привремена променљива
- Сакупљање смећа се најчешће дешава у овој генерацији
- Новододати објекти су имплицинтно чланови генерације 0

Генерација 0

- Већина објеката не преживи ову генерацију
- Ако апликација покуша да креира нови објекат када је генерација 0 пуна, покреће се *GC* да би се направило простора
- *GC* у овом случају врши испитивање објеката само у генерацији 0, а не на целом *heap*-у
- Овај пролаз углавном ослободи довољно меморије тако да апликација може несметано наставити са креирањем нових објеката

Генерација 1

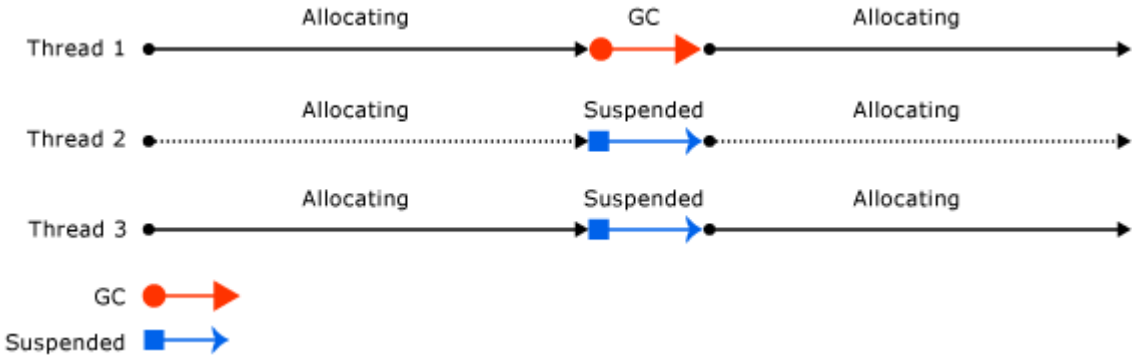
- Садржи краткотрајне објекте и служи као тампон између објеката кратког века и дуговечних објеката
- Након што *GC* обради генерацију 0, сабија меморију у њој да би се направио простор за нове објекте, а оне преживеле промовише у генерацију 1
- *GC* не мора сваки пут да пролази кроз генерацију 1 и 2 након извршене анализе генерације 0
- Ако се из генерације 0 не ослободи довољно меморије за нове објекте, тек тада ће *GC* да пређе на генерације 1 и 2

Генерација 2

- Садржи дуговечне објекте
- Пример је објекат у серверској апликацији који садржи статичке податке који су активни током процеса
- Да ли знате навести још неки пример?
- Објекти који преживе анализу генерације 2, остају ту до нове анализе када се за њу створе услови

Покретање *GC*-а

- Приликом покретања *GC*-а све активне нити се суспендују осим оне која је покренула сакупљање смећа



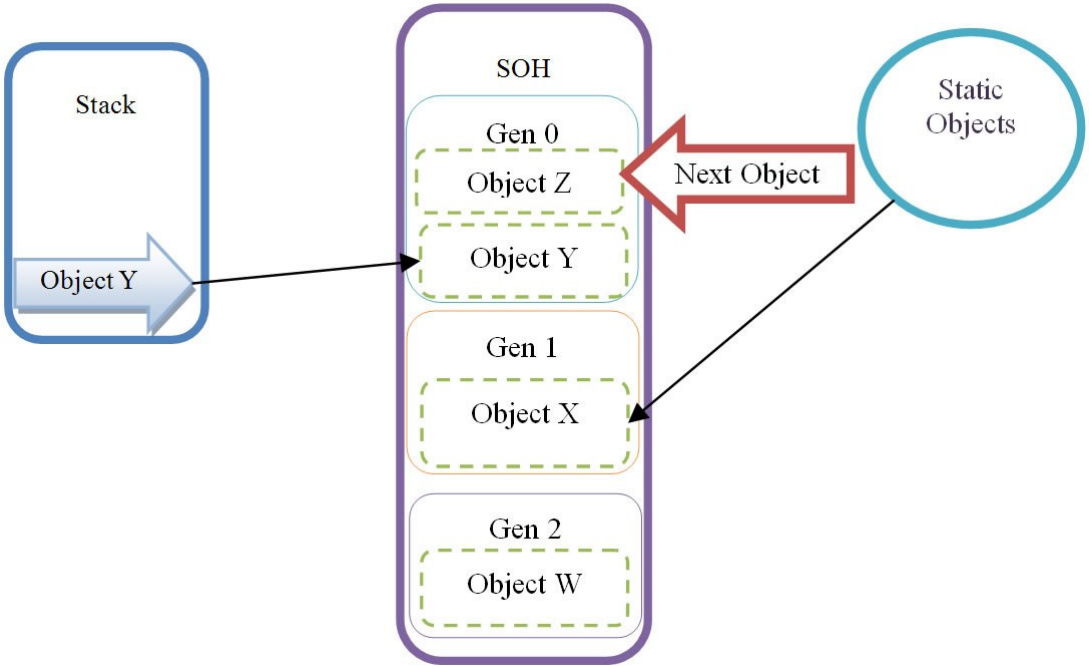
Прагови генерација

- *GC* се покреће у посебној нити када величина свих објеката у генерацији достигне праг специфичан за њу:
  - Ген 0 - ~256KB
  - Ген 1 - ~2MB
  - Ген 2 - ~10 MB

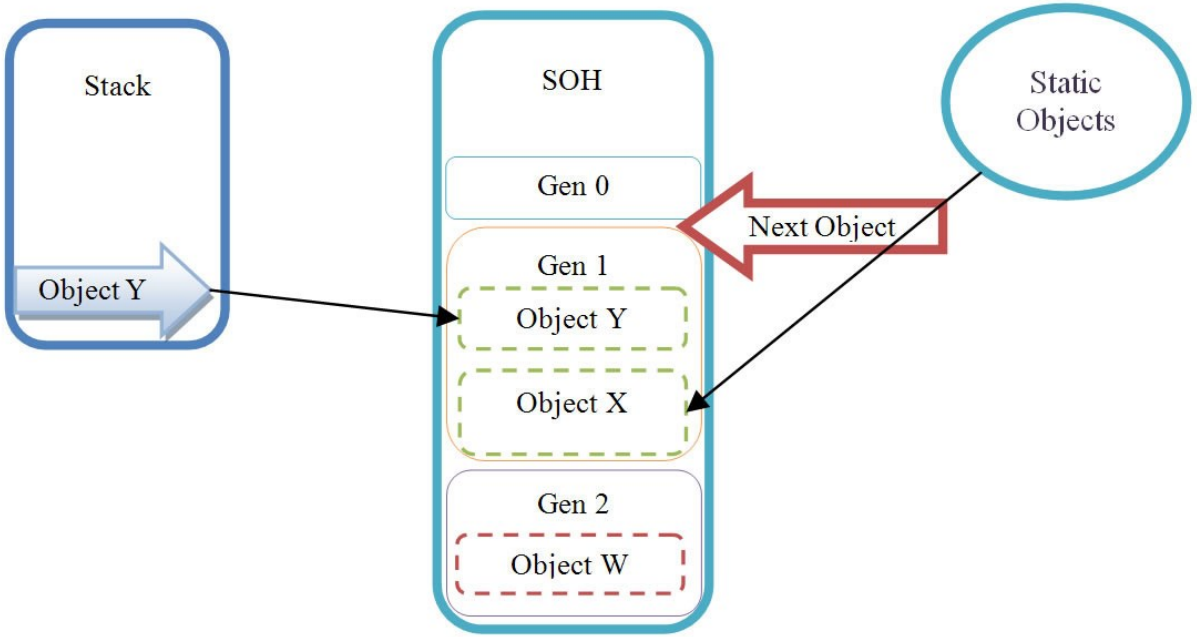
- Ови прагови су почетни, *.Net* мења њихову вредност у зависности од понашања апликације

- Када се још покреће *GC*?

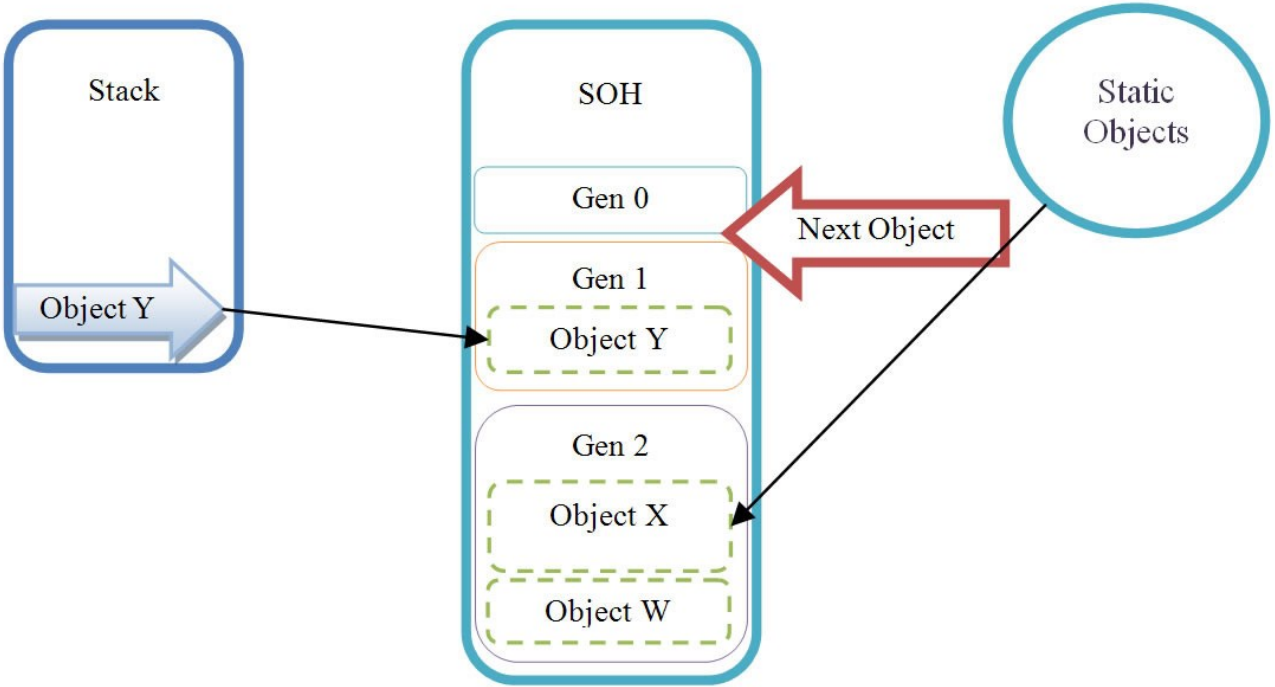
Принцип рада *GC*-а на *SOH*-у - генерације



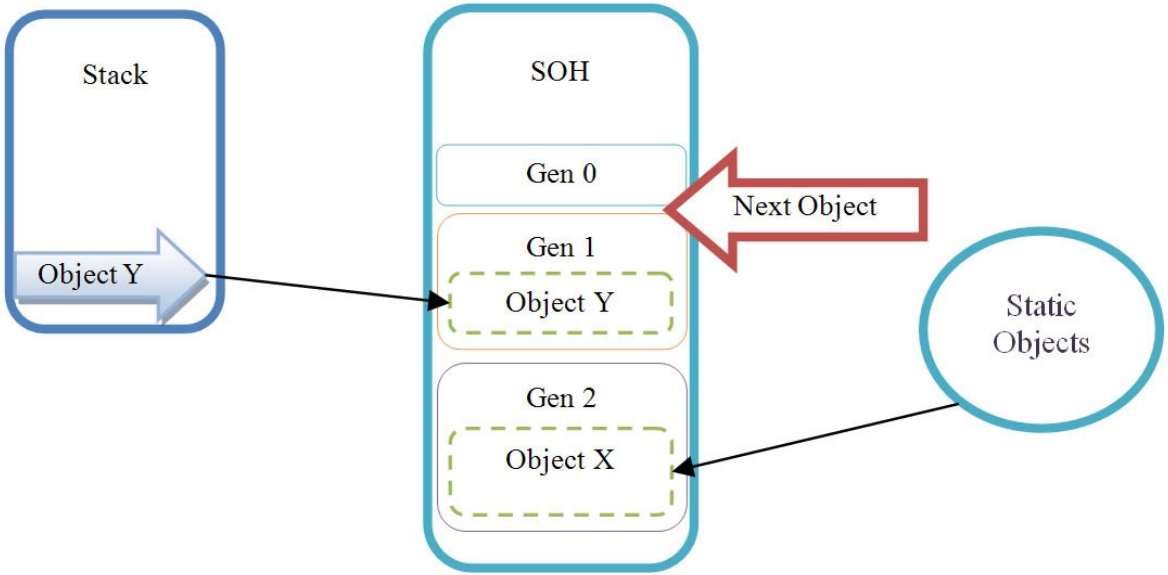
Принцип рада *GC*-а на *SOH*-у – генерација 0



Принцип рада *GC*-а на *SOH*-у – генерација 1



Принцип рада *GC*-а на *SOH*-у – генерација 2



Принцип рада *GC*-а на *SOH*-у – генерација 2

- Као што смо видели на претходној слици:
  - Објекат W нестаје
  - Објекат X се помера у Ген 2
  - Објекат Y се помера у Ген 1
  - Објекат Z нестаје

- Очигледно је да је овде *GC* морао да уради више посла у овом случају него у претходним

- Идеално би било да објекти стигну у генерацију 2 само ако морају

Принцип рада *GC*-а на *SOH*-у – генерација 2

- Генерација 2 је скупо место за чување објеката и може значајно да има утицаја на перформансе

- Опште правило о броју елемената по генарацијама:
  - Генерација 0 има 10 пута више елемената него генерација 1
  - Генерација 1 има 10 пута више елемената него генерација 2
  - Генерација нула има 100 пута више елемената него генерација 2

Утицај на перформансе

- Стварање што мање објеката није кључ побољшања перформанси *GC*-а
- Истина је да ће *GC* тада имати мање посла, али поента је у томе колико се објеката уклони након проласка *GC*, а не колико се често он покреће
- Док *GC* из генерације 0 уклања много објеката, што је релативно јефтин процес, мање објеката се промовише и тако перформансе остају високе
- Идеално, већина објеката би требало да нестане у генерацији 0 или евентулано у генерацији 1

Утицај на перформансе

- Објекти који заврше у генерацији 2 би требало да су креирани са разлогом
- Вјероватно јер су за вишекратну употребу и које је једноставније креирати једном, него сваки пут изнова
- Анализа генерације 2 је скупа
- Када *GC* установи да је стопа преживљавања висока у генерацији, проширује се генерација
- Додавање новог меморијског сегмента за вашу апликацију обавља ОС

Праћење рада *GC*-а

- *Windows (ETW)* систем за праћење поред тога што нуди опције профајлирања апликације, може да нам да корисне информације о самом раду *GC*-а као што су:
  - Која генерација се тренутно прикупља
  - Шта је покренуло *GC*
  - …

- На основу ових информација, инжењери могу да добију јасну слику како је њихова апликација написана из угла аутоматског управљања меморијом и да предузму неопходне кораке

Праћење рада *GC*-а

- *CLR API* током профајлирања обезбеђује информације о објектима који су афектовани током рада *GC*-а
- Такође, добијамо информацију када је сакуплање смећа почело и када се завршило
- Добијамо свеобухватан извештај о самом сакупљању смећа
- Постоје и други профајлери који нам значајно могу помоћи у анализи рада наше апликације са више аспеката (*ANTS*, *DotTrace*...)

Тема за наредно предавање

Garbage Collector GC
Large Object Heap (LOH)

Видимо се и хвала!