Факултет Техничких Наука
Универзитет у Новом Саду

Виртуелизација процеса

Garbage Collector (GC) - основе

Нови Сад, 2023.

Heap

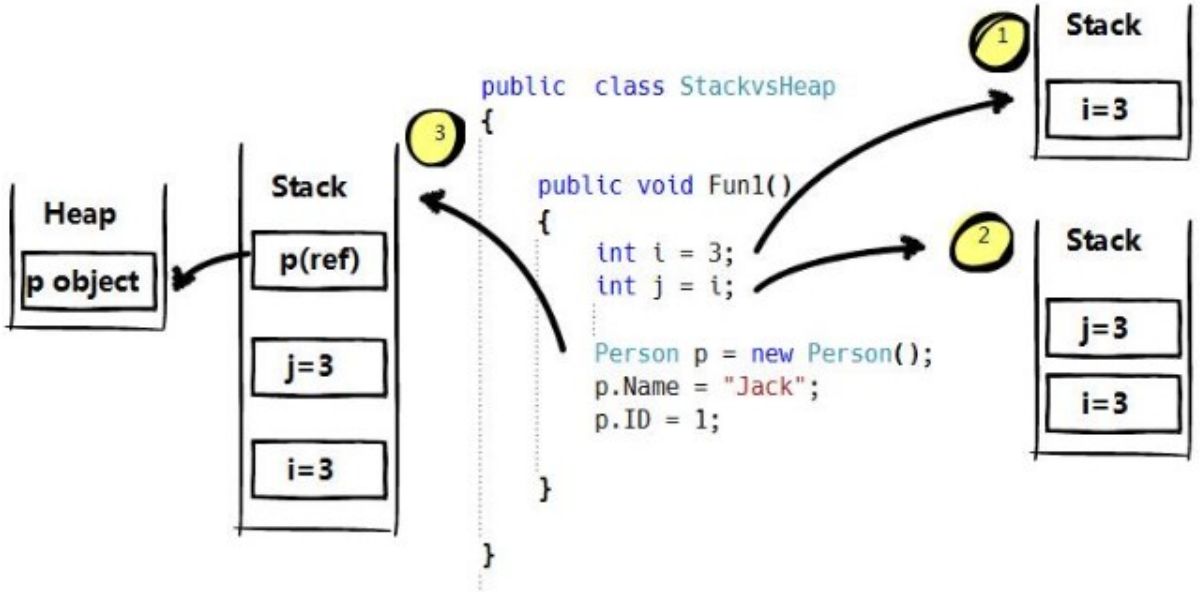
- Када се *.Net* апликација покрене, креирају се три секције меморије које су нама од интереса:
  - *Stack*
  - *Small Object Heap (SOH)* складишти објекте објекте мање од 85KB
  - *Large Object Heap (LOH)* скалдишти објекте веће или једнаке од 85KB
  - Из угла рачунарских ресурса, где се они налазе?

- Када се креира нова инстанца референтног типа, она обично укључује кључну ријеч *"New"*

- Референца објекта се налази на *stack*-у

- Стварна инстанца се складишти на *heap*-у

Heap


Heap
p object
Stack
p(ref)
j=3
i=3
3
public class StackvsHeap
{
public void Fun1()
{
int i = 3;
int j = i;
Person p = new Person();
p.Name = "Jack";
p.ID = 1;
}
}
1
Stack
i=3
2
Stack
j=3
i=3

Heap

- *.Net* управља објектима у наше име
- Након “new” он се брине за креирање, иницијализацију и постављање објеката на *heap*
- Резервише меморију за објекат
- Како ово радимо нпр. у програмским језицима као што су C, C++...?
- Можете “скоро” да заборавите на креирани објекат

C++ *vs* C#

C++

```
void showUsername()
{
    char *username;
    username = (char *) malloc(8);
    strcpy(username, "username");
    printf("%s", username);
    free(username);
}

int main()
{
    showUsername();
}
```

C#

```
var username = "username";
Console.WriteLine(username);
```

Garbage Collection

- Не морате да бришете објекат када завршите са њим
- Када га не будете користили биће аутоматски очишћен
- *Garbage Collector GC* – сакупљач смећа
- Супротно од мануелног управљања меморијом
- Пружа сигурност да објекат не може користити за себе меморију додељену другом објекту

Garbage Collection

- *GC* –штити нашу апликацију од цурења меморије
- Рад *GC* –а може да предствља значајан дио укупне обраде времена у програму, што има утицаја на перформансе
- Ресурсе којима не управља *.Net* , и који се другачије зову *unmanaged* ресурси нису предмет обраде *GC*-а
  - Приступ фајловима
  - Приступ бази
  - ...
- О њима и њиховом уништавању када дође време мора инжењер да води бригу

Важни концепти *CLR* меморије

- Сваки процес има свој одвојени виртуелни простор
- Сви процеси на истом рачунару деле исту физичку меморију
- Као инжењер радите само са виртуелним адресним простором и никада не манипулишете директно са физичком меморијом
- *GC* ради над виртуелном меморијом
- Подразумевано, на 32-битним рачунарима сваки процес има виртуелни адресни простор у корисничком режиму од 2 *GB*

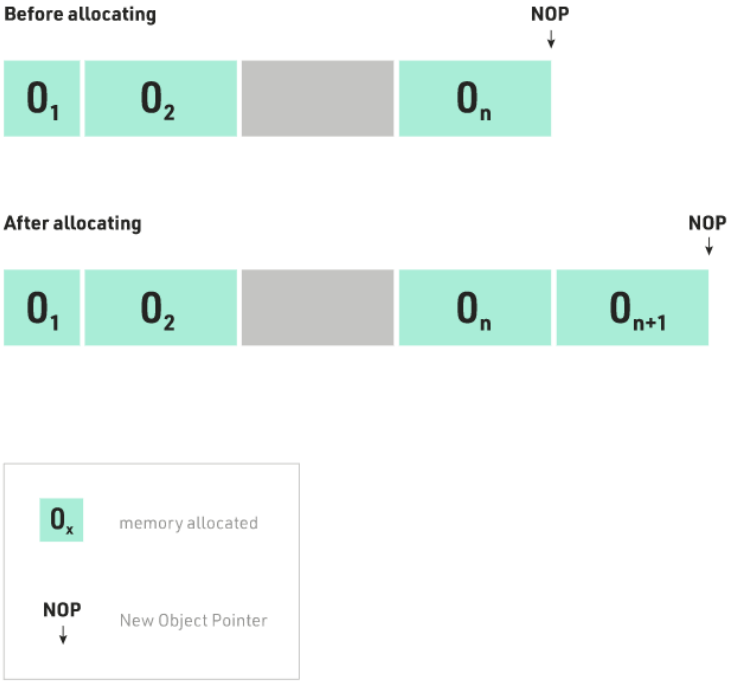
Важни концепти *CLR* меморије

- Виртуелни адресни простор може бити фрагментован
- Када се захтева додела виртуелне меморије вашој апликацији, менаџер виртуелне меморије (*Virtual Memory Manager - VMM*) мора пронаћи слободан блок да се задовољи захтев за алокацијом апликације
- Иако постоји слободно 2 *GB*, додела ће бити неуспешна ако се не налази у једном адресном блоку
- Ако нема довољно виртуелног или физичког простора у меморији, ваша апликација лако може да остане без меморије
- Ко вашој апликацији додељује ове блокове?

Алокација меморије

- Када се покрене нови процес креира се *heap* или непрекидни регион адресног простора
- Показивач се налази на адреси где ће бити смештен следећи објекат
- На самом почетку он показује на почетну адресу додељене меморије
- Следећи објекат се додељује одмах након претходног и тако редом
- Са обзиром да се објекти чувају један иза другог, апликација може брзо и лако да им приступа

Алокација меморије



Алокација меморије

- Што је креирано мање објеката, посао *GC*-а ће бити лакши
- Приликом додељивања објеката немојте користити заокружене вредности које превазилазе ваше потребе
  - Нпр. низ од 32 бајта, ако вам је потребно само 15 бајтова
  - Како се правилно иницијализује листа?
- Величина сегмента која је додељена апликацији подложна је промени у сваком тренутку
- Не би требало да конфигуришете количину меморије за доделу сегмента

Ослобађање меморије

- *GC* се аутоматски покреће када је испуњен један од следећих услова:
  - Рачунар има мало физичке меморије – стиже обавештење од оперативног система
  - Препуни се виртуелна меморија која је додељена апликацији
  - На предефинисани временски период
- Могуће је *GC* позвати директно из кода помоћу *GC.Collect()* методе
- Потреба да се позове *GC.Collect()* метода је знак да нешто добро не радимо и треба је избегавати
- Утиче на перформансе и користи се углавном током тестирања

Ослобађање меморије

- *GC* брише објекте који се више не користе
- Испитивањем корена апликације може се установити који се објекти користе, а који не
- Сваки корен или се односи на објекат на *heap*-у или не показује ни на шта
- Прави се листа објеката који су доступни из корена
- Сви они који не припадају овој листи се бришу

Ослобађање меморије

- Поред коренских референци објекат може бити референциран и од других објеката

- Узећемо за пример обичну класу *Customer,* која у себи садржи колекцију објеката *Order*

- Када у колекцију додамо нови објекат, сама колекција има референцу на тај новододати објекат

- Ако би сама класа *Customer* имала референце на *stack* имала би следеће референце:
  - Коренску референцу за *Customer-a* која садржи:
    - Референцу на колекцију налога, а која садржи:
      - Референцу на објекте налога

График референци

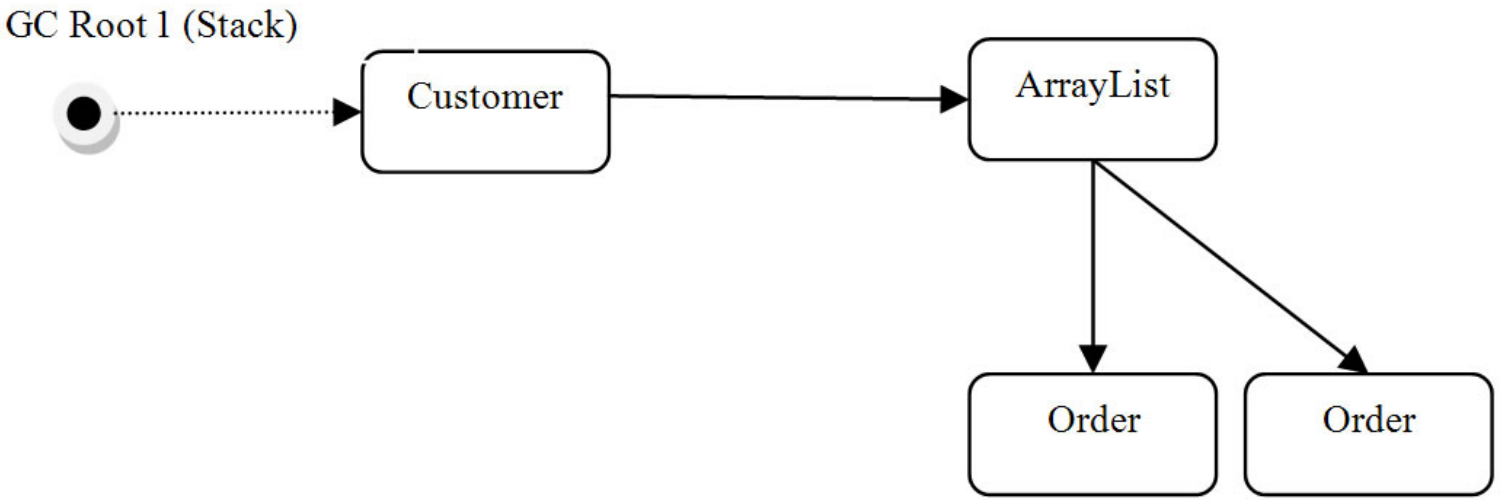
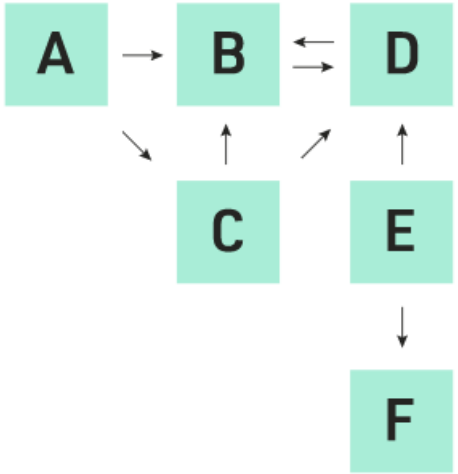
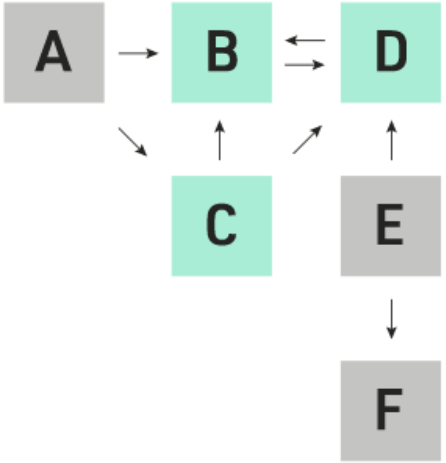


График референци

Before



After (**A** and **E** marked as no longer needed)



Инспекција и прикупљање

- Прикупити све коренске референце
- Прођи кроз стабло сваке од њих и означити објекте да се и даље користе

```
void Collect()
{
    List gcRoots=GetAllGCRoots();
    foreach (objectRef root in gcRoots)
    {
        Mark(root);
    }
    Cleanup();
}
```

Маркирање елемената

- *Mark* – маркира елементе који су још у употреби
- Пролази кроз све његове подређене и њих исто означи да су у употреби

```
Void Mark(objectRef o)
{
     if (!InUseList.Exists(o))
     {
         InUseList.Add(o);
         List refs=GetAllChildReferences(o);
         foreach (objectRef childRef in refs)
         {
             Mark(childRef);
         }
     }
}
```

- Сви елементи који након овог пролаза не буду маркирани, биће обрисани

Статички елементи

- Шта су статички елементи?
- Представљају коренску референцу
- Како се приступа статичким елементима?

```
class Person
{
    public int Age=0;
    public static MaxAge=120;
}
```

Статички елементи

```
Person thisPerson=new Person();
thisPerson.Age=121;

if (thisPerson.Age>Person.MaxAge)
{
     // Validation Failure
}
```

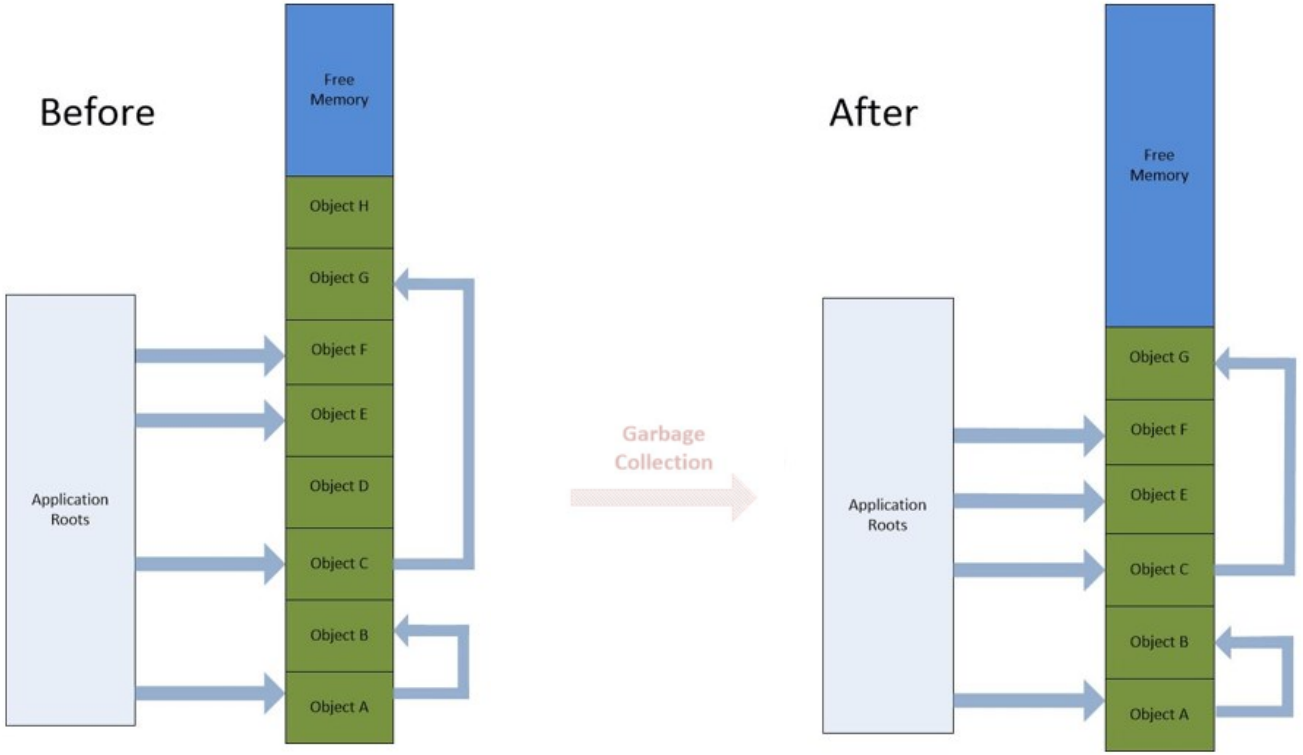
- Никада се не сакупљају као смеће јер су коренске референце
- Могу бити одговорни за задржавање објеката учитаних у меморију много дуже него што би се очекивало

Статички елементи

- Класе које се претплате на статичке догађаје (*event*-е) остају у меморији док се претплата на догађај не уклони
- Зашто статичке колекције могу бити проблем?

```
public class MyData
{
  public static Customer Client;
  public static event EventType OnOrdersAdded;
  static  MyData()
  {
       // Initialize
       Client=new Customer();
  }
}
```

Пример рада *GC*-а



Пример рада *GC*-а

Before deallocating

After deallocating, before compacting

After compacting

Sweeping

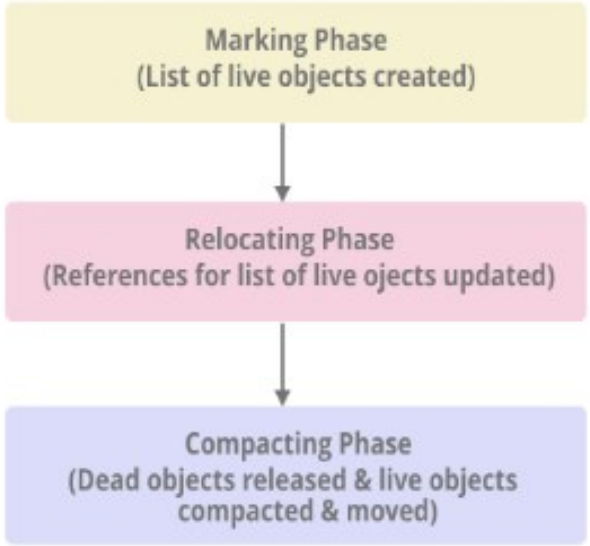
Before sweeping

After sweeping

Фазе током рада *GC*-а

- Маркирање објеката који су и даље у употреби
- Брисање објеката које нико не користи
- Појава фрагментације
- Сажимање *heap*-а, тј.реалокација преосталих живих објеката на меморијска места која су остала након брисања старих

Фазе током рада *GC*-а



- Какав проблем и када ово може да проузрокује?

Генерације *GC*-а

- Брже је сабити меморију за дио *heap*-а него за цијели *heap*
- Нови објекти имају краћи век трајања, а старији дужи
- *Heap* се дели у три генерације како би се независно руковало краткорочним и дугорочним објектима
- Постоје три генерације:
  - Генерција 0
  - Генерација 1
  - Генерација 2

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 виека 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

Тема за наредно предавање

Garbage Collector GC
Small Object Heap (SOH)

Видимо се и хвала!