Факултет Техничких Наука
Универзитет у Новом Саду

Виртуелизација процеса

Основни концепти управљања меморијом у .Net-у

Нови Сад, 2023.

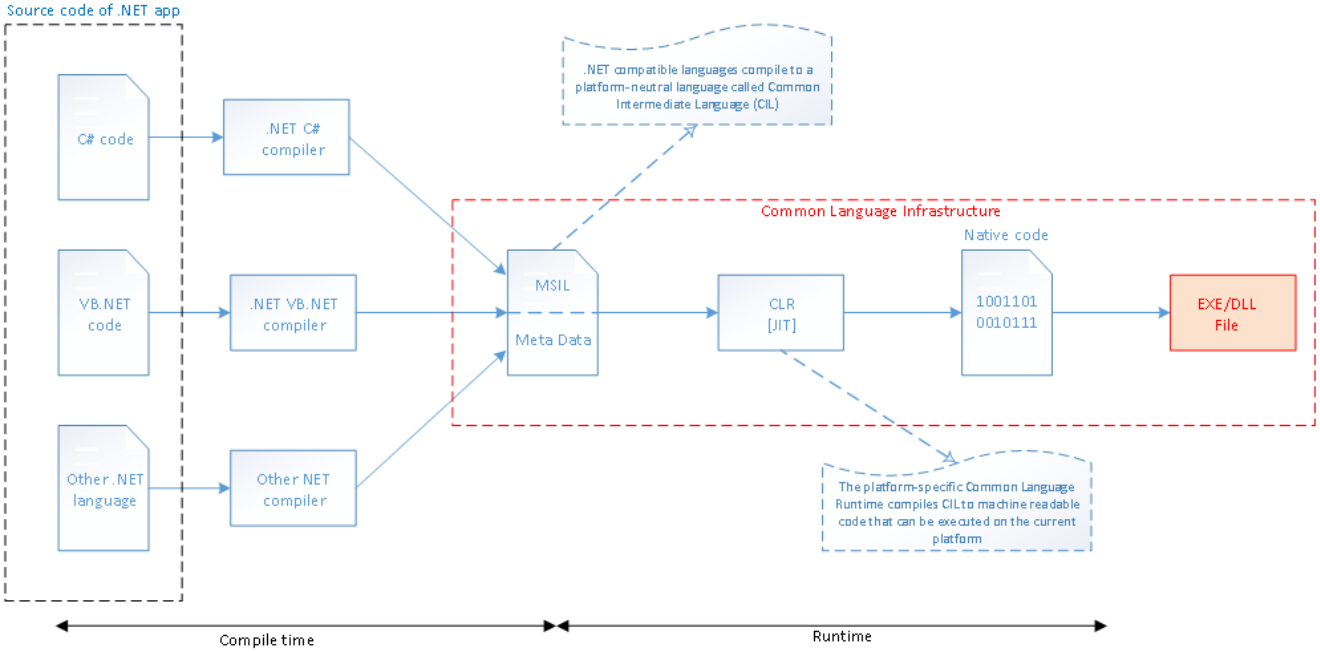
.NET Framework

- Контролисано програмско окружење за развој дистрибуираних апликација на *windows* оперативном систему
- *.Net* подржава више програмских језика: *Visual Basic, C++, C#...*
- Извршавање у специфичном софтверском окружењу *Common Language Runtime (CLR)*
- *CLR* је виртуелна машина која омогућава контролисано извршавање програма писаних за *.Net* апликацију укључујући између осталог и управљање меморијом

.NET Framework

- Код који се извршава од стране *CLR*-а, односно у .Net назива се *managed* код
- Шта је *unmanaged* код?
- *.Net framework* укључује огроман број библиотека кроз *Framework Class Library (FCL)*
  - Кориснички интерфејс
  - Приступ базама података
  - Развој дистрибуираних и *Web* апликација
  - Подршка за конкурентно програмирање
- *FCL* библиотеке су независне од програмског језика и организоване су у логичке групе - *namespaces*

.NET Framework



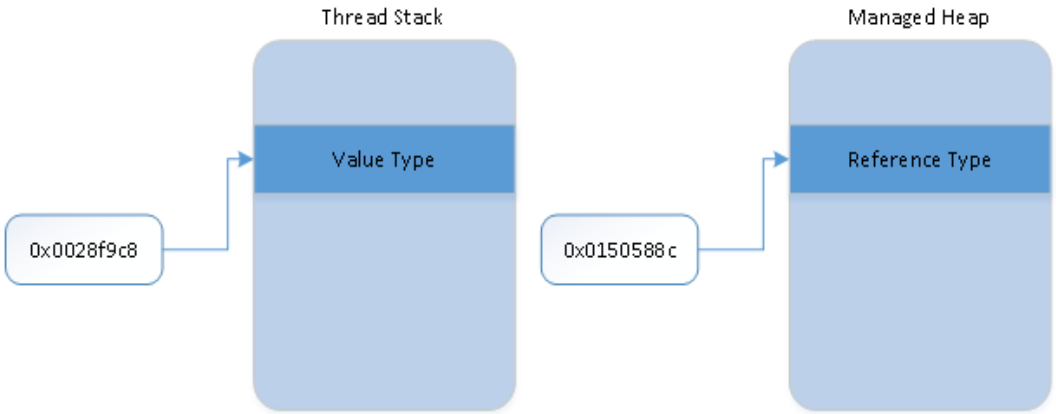
Основни типови података

- Тип *type* је основна јединица програмабилности у *.Net*-у

- Две категорије типова података
  - *Value types* – типови података настали из класе *System.ValueType*
    - *Int, double, char, enum…*
    - Где се чувају ови подаци?

  - *Reference types* – не садрже вредност променљиве, већ представљају референцу на њену меморијску локацију
    - Класе, интерфејси, делегати...
    - Где се чувају ови подаци?

Основни типови података



- Складиштење објеката на *heap*-у и њихова контрола од стране *Garbage Collectora (GC)* је скупа операција
- Више информација о овоме – касније

Предности и мане *.Net*-а

- *.Net* је практичан да се пише код са мало или нимало знања о томе како функционише и шта се заиста дешава “испод хаубе”
- Прелазак на језике у чијој основи је *.Net* је једноставан
- Аутоматско управљање меморијом нам оставља простор да се фокусирамо на сам код и оно што желимо да урадимо
- Недовољно разумевање утицаја кода који пишемо на перформансе и управљање меморијом

Предности и мане *.Net*-а

- Не морате да бринете о управљању меморијом у .Net-у – чиста заблуда!
- Касно уочавање ових проблема
- Након извршеног функционалног и *stress* тестирање или када пројекат уђе у продукцију
- Примери проблема?

Шта вам доноси боље разумевање управљања меморијом?

- Научићете како апликација функционише “испод хаубе”
- Након разумевања основа управљања меморијом знаћете да пишете:
  - Бољи
  - Перформантнији код
- Бићете много боље припремљени када се појаве проблеми са перформансама у некој од ваших апликација

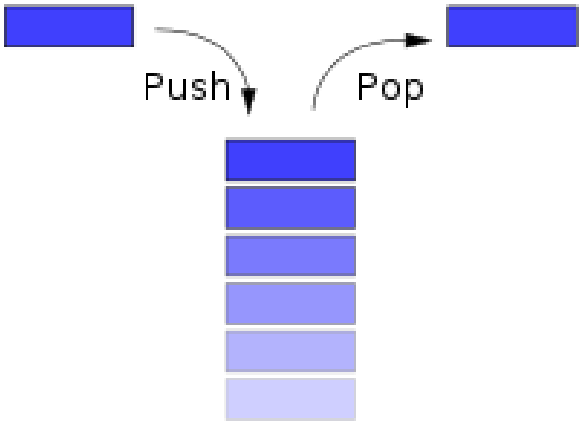
Саставни делови апликације

- Апликације су написане тако да енкапсулирају код у методе и класе
- *.Net* мора да прати позиве метода као и стање података у сваком од тих позива метода
- Када се метода позове, она добије своје "окружење" где чува важне информације за њу током трајања њеног позива
- Метода може да добије податке из глобалних и статичких елемената, као и из парамтетара који су јој прослеђени

Саставни делови апликације

- Када једна метода позива другу, њено локално стање мора да се запамти. Због чега?
- За праћење тока програма *.Net* користи *stack*
- *LIFO*
- *Stack* можемо упоредити са тањирима постављеним један на други

Stack



Stack

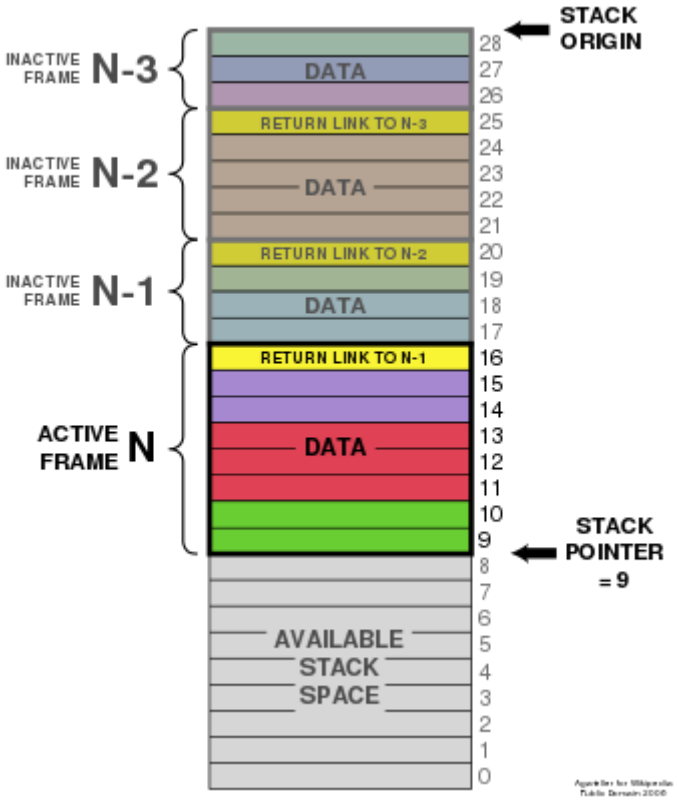
- Претходно споменуто "окружење" - оквир *stack*-а
  - Аргументи
  - Локално декларисане променљиве
  - Адреса кода за наставак

- За сваки позив метода у стаблу, гдје једна позива другу, оквири *stack*-а су наслагани један на другог

- Након завршетка позива методе, њен оквир се уклања са врха *stack*-а и извршавање се враћа на претходну адресу

- Оквир на врху *stack*-а је увек онај који користи метода која се тренутно извршава

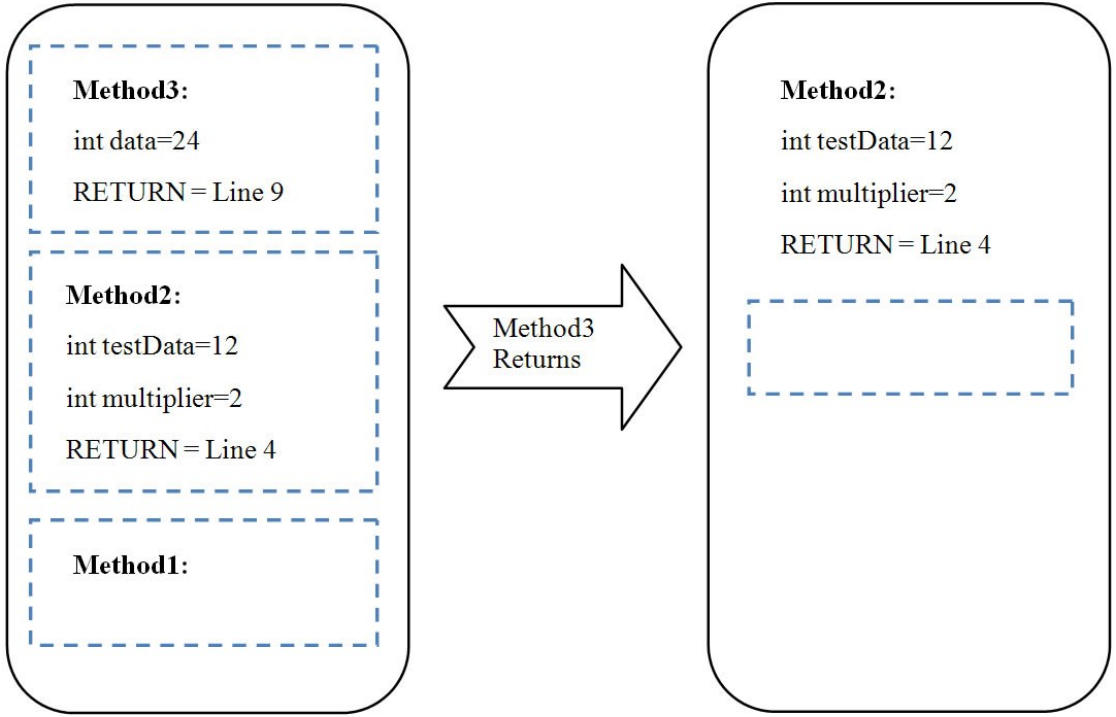
Stack


INACTIVE FRAME N-3
INACTIVE FRAME N-2
INACTIVE FRAME N-1
ACTIVE FRAME N
DATA
RETURN LINK TO N-3
DATA
RETURN LINK TO N-2
DATA
RETURN LINK TO N-1
DATA
AVAILABLE STACK SPACE
STACK ORIGIN
STACK POINTER = 9
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0

Stack

```
1  void Method1()
2  {
3        Method2(12);
4        Console.WriteLine(" Goodbye");
5  }
6  void Method2(int testData)
7  {
8     int multiplier=2;
9     Console.WriteLine("Value is " + testData.ToString());
10    Method3(testData * multiplier);
11 }
12 void Method3(int data)
13 {
14     Console.WriteLine("Double " + data.ToString());
15 }
```

Stack


Method3:
int data=24
RETURN = Line 9
Method2:
int testData=12
int multiplier=2
RETURN = Line 4
Method1:
Method3
Returns
Method2:
int testData=12
int multiplier=2
RETURN = Line 4

Heap

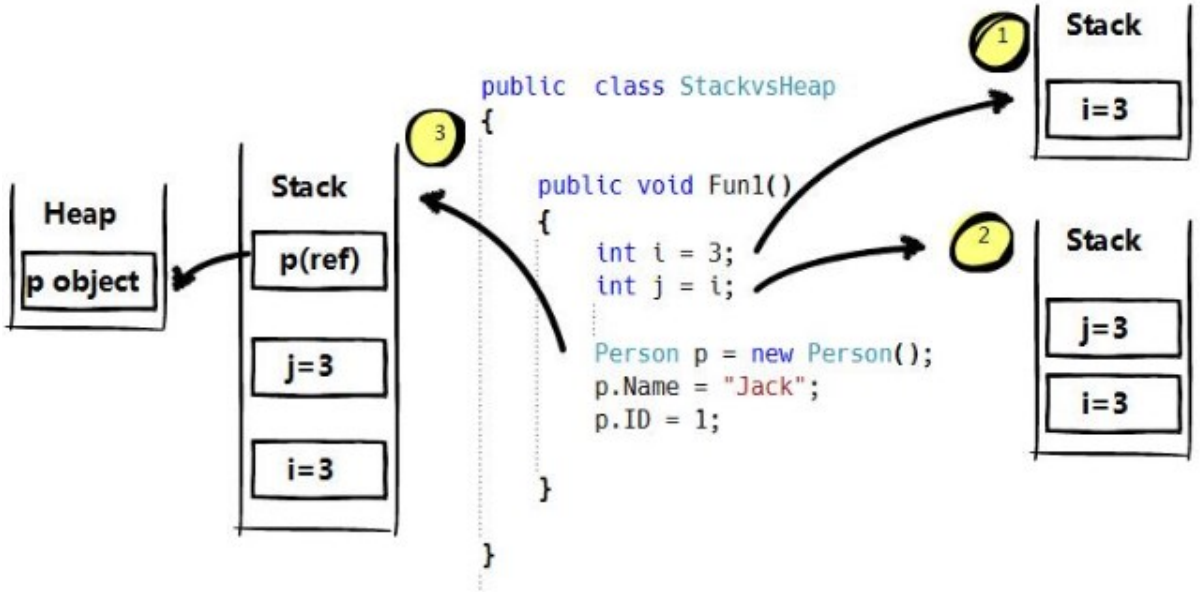
- Када се *.Net* апликација покрене, креирају се три секције меморије које су нама од интереса:
  - *Stack*
  - *Small Object Heap (SOH)* складишти објекте објекте мање од 85KB
  - *Large Object Heap (LOH)* скалдишти објекте веће или једнаке од 85KB
  - Из угла рачунарских ресурса, где се они налазе?

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

- Референца објекта се налази на *stack*-у

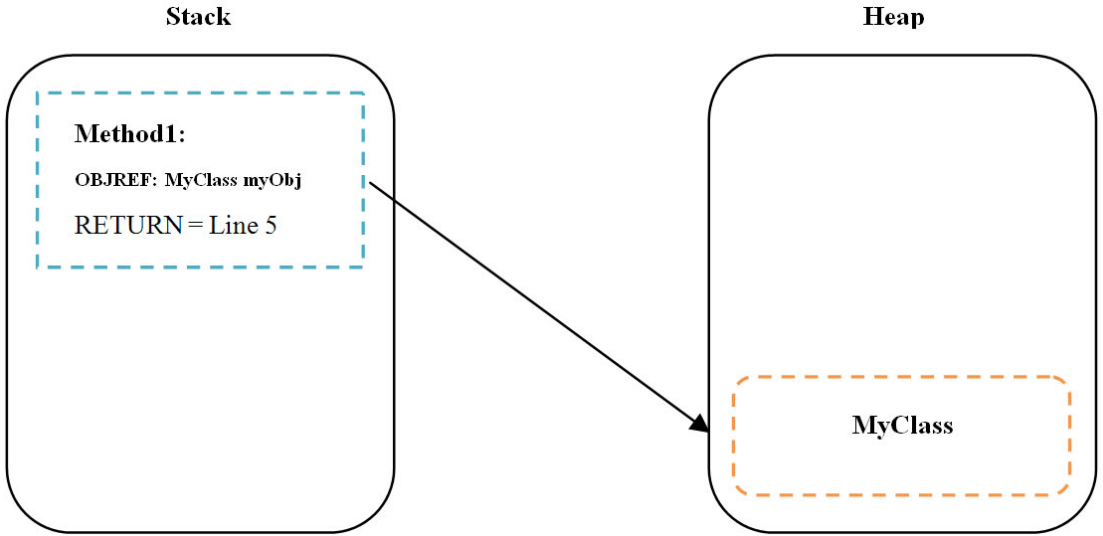
- Стварна инстанца се складишти на *heap*-у

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

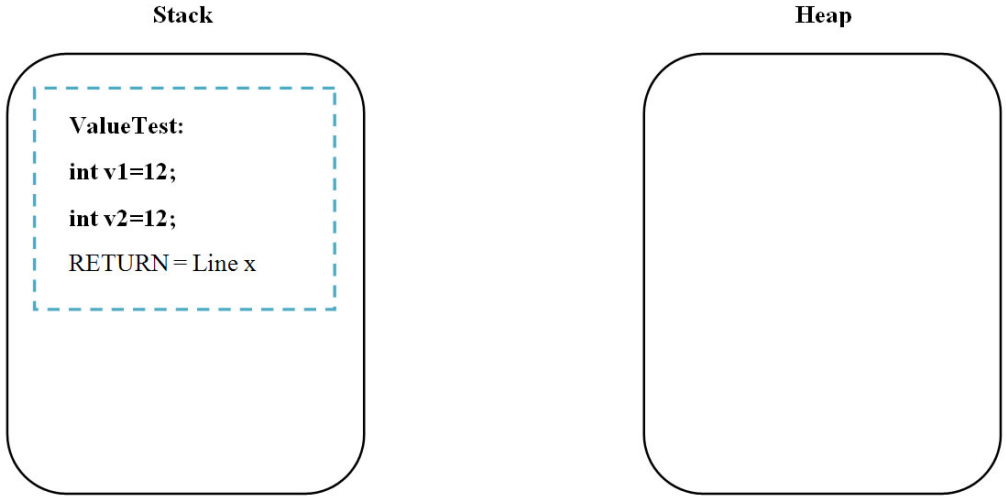
Heap

```
void Method1()
{
      MyClass myObj=new MyClass();
      Console.WriteLine(myObj.Text);
}
```



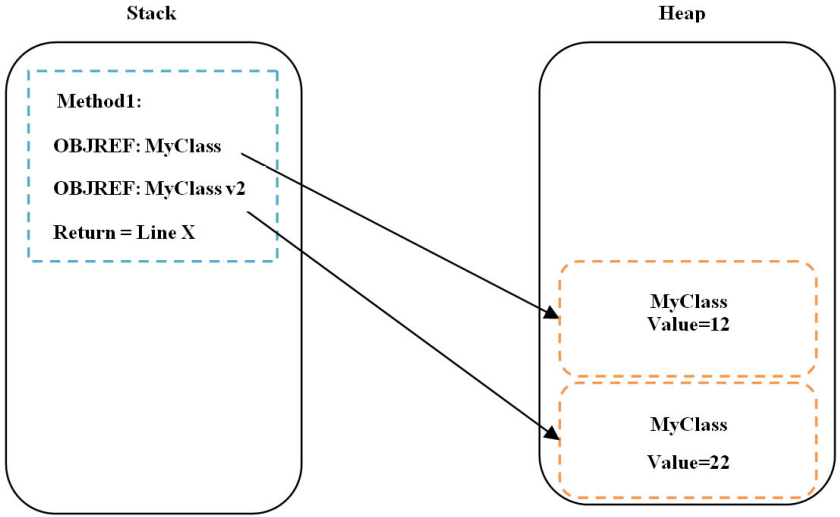
Heap

```
1  void ValueTest()
2  {
3      int v1=12;
4      int v2=22;
5      v2=v1;
6      Console.Writeline(v2);
7  }
```

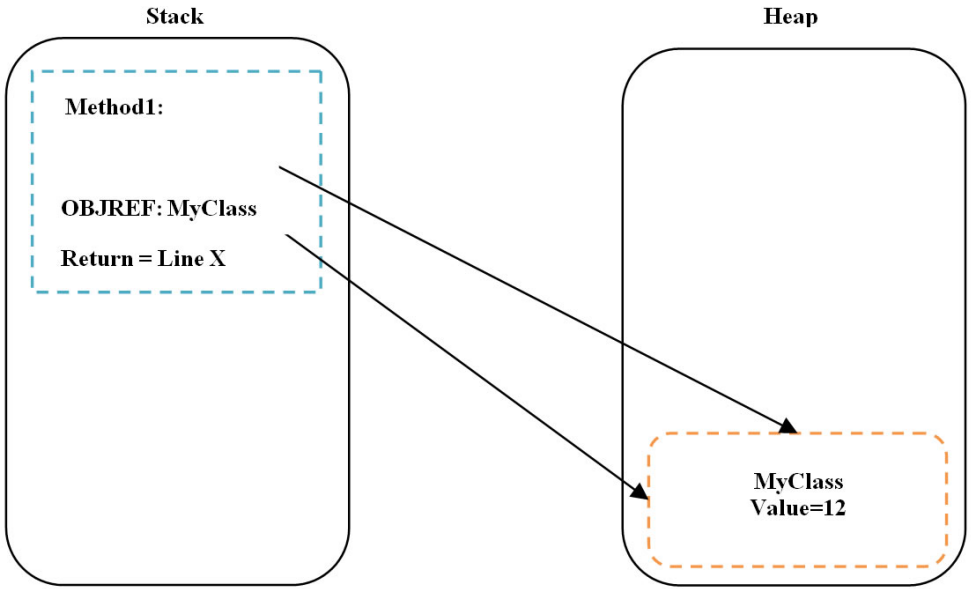


Heap

```
1  void RefTest()
2  {
3      MyClass v1=new MyClass(12);
4      MyClass v2=new MyClass(22);
5      v2=v1;
6      Console.Writeline(v2.Value);
7  }
```



Heap


Stack
Heap
Method1:
OBJREF: MyClass
Return = Line X
MyClass
Value=12

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

Heap

- Не морате да бришете објекат када завршите са њим
- Када га не будете користили биће аутоматски очишћен
- Како ово *.Net* успева?
- *Garbage Collector GC* – сакупљач смећа
- Супротно од мануелног управљања меморијом

Тема за наредно предавање

Garbage Collector (GC)

Видимо се и хвала!