**Adresování a správa paměti, Garbage collector typy:**

**2 typy:**

**Halda (Heap)**

* Je pouze jedna v celém programu
* Na haldu se ukládají všechny referenční typy:
  + - class
    - interface
    - delegate
    - object
    - string

String je objekt…, vždy je na haldě

Ve stacku jsou na ně použity ukazatele (pointers)

* Není jakkoliv organizovaná
* Je přístupná pro všechny (pokud instanci nezablokujeme modifikátorem přístupu – public, private, internal, protected)
* Pokud jsou na haldě nějaká data, na které neukazuje žádný ukazatel, smaže je garbage collector

**Zásobník (Stack)**

* Druh paměti, který vlákna využívají pro provedení kódu
* Každé vlákno má vlastní zásobník
* Je setříděný (LIFO) = Last In First Out
* Každá metoda je vlastně blok, kde jsou uložena její data, bloky jsou vrstveny na sobě takovým způsobem, že nově přidané bloky padají na starší a tím zabraňují práci s těmi předešlými, dokud nejsou vyřešeny.
* Obsahuje pouze primitivní datové typy:
* bool
* byte
* char
* decimal
* double
* enum
* float
* int
* long
* sbyte
* short
* struct
* uint
* ulong
* ushort
* Proměnné volané v metodách, se také ukládají na zásobník, jsou ale nepřístupné, a jakmile metoda skončí, smažou se

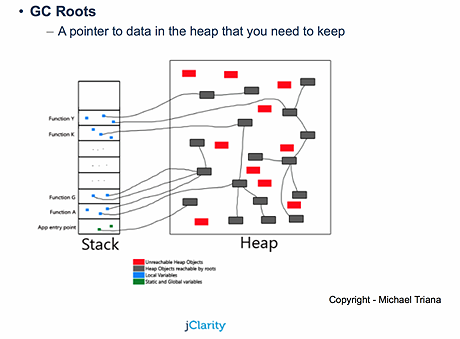
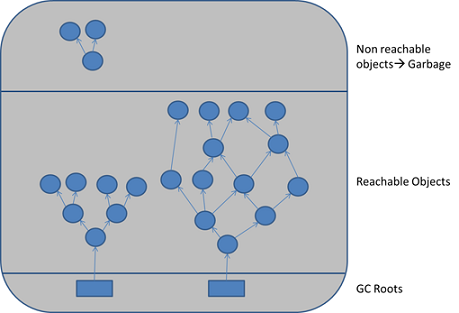
Když se vytvoří primitivní proměnná v třídě mimo metodu, tak je uložena na haldě

**Garbage collector**

* Automatické zpracování paměti
* Pracuje na Heap (haldě)

Automaticky prochází haldu a maže objekty, které nemají žádnou referenci (nic na ně neukazuje)

Objekty se rozdělují na živé a mrtvé, a sestaví se z nich strom

**3 fáze:**

**Marking**

Od rootů (kořenů, musí být vždy 1 nebo víc) jsou sledovány reference a objekty jsou přidávány na seznam živých. Všechny objekty, které nejsou na seznamu živých jsou odstraněni z haldy.

**Relocating**

Reference na objekty, které jsou na seznamu živých jsou aktualizovány (adresy) a ukazují na adresy objektů, kde se budou nacházet po fázi Compacting

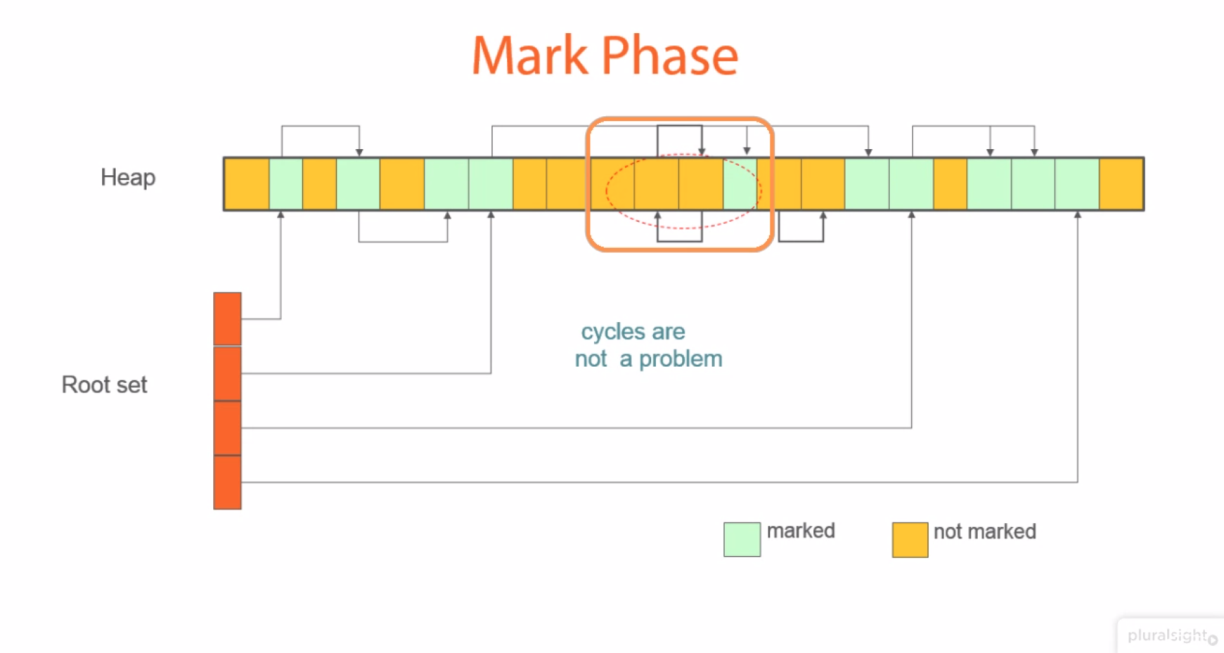
**Compacting**

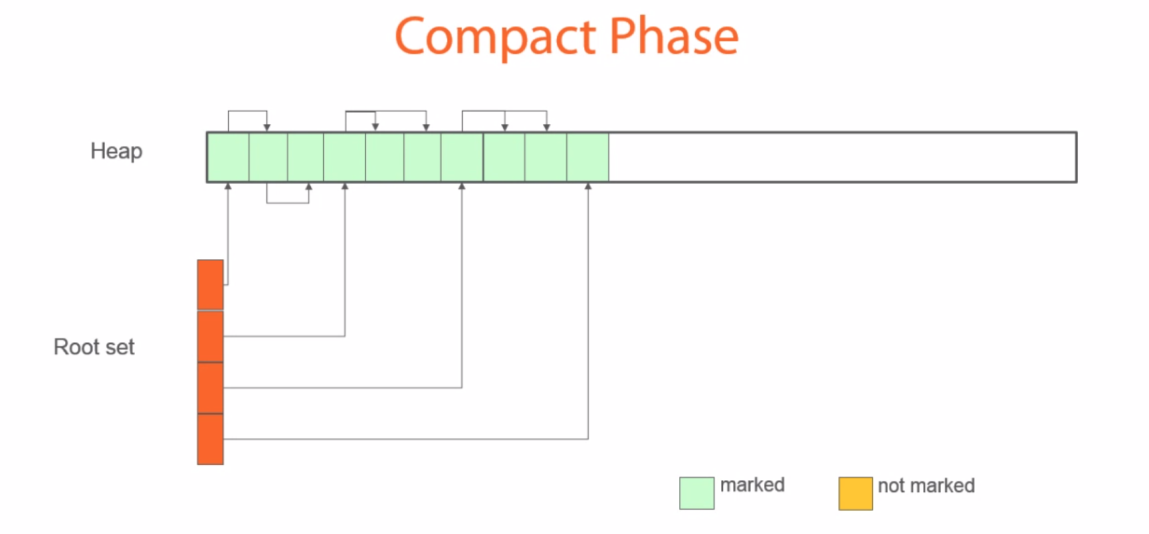
Uvolněné místo po smazání mrtvých objektů je zhutněno a živé objekty se přesunou, aby mezi nimi nebyly mezery volného místa.

Marking Phase: A list of all the live objects is created during the marking phase. This is done by following the references from all the root objects. All of the objects that are not on the list of live objects are potentially deleted from the heap memory.

Relocating Phase: The references of all the objects that were on the list of all the live objects are updated in the relocating phase so that they point to the new location where the objects will be relocated to in the compacting phase.

Compacting Phase: The heap gets compacted in the compacting phase as the space occupied by the dead objects is released and the live objects remaining are moved. All the live objects that remain after the garbage collection are moved towards the older end of the heap memory in their original order.





**Generace**

Každý objekt začíná v generaci 0, když se provede garbage collector, tak všechny objekty co přežily jsou povýšeny do generace 1, když přežijou garbage collector 1 nebo víckrát i tam tak jsou povýšeny do generace 2

C# přepokládá, že nejnovější objekty umírají rychle, starší se dožijou nejdéle. Třeba statický proměnná na začátku třídy (ta je živá po celý program).

**Generace 0**

Obsahuje krátkodobé objekty (objekty s krátkodobou životností), např: krátkodobá proměnná.

K uvolňování paměti dochází nejčastějí v této fázi.

Může být zavolán Garbage Collector manuálně (metodou) uživatelem.

**Generace 1**

Tato generace obsahuje krátkodobé objekty a slouží jako vyrovnávací paměť mezi krátkodobé a dlouhodobé objekty – středně dlouhé objekty.

Garbage collector zde běží méněkrát než v generaci 0

**Generation 2**

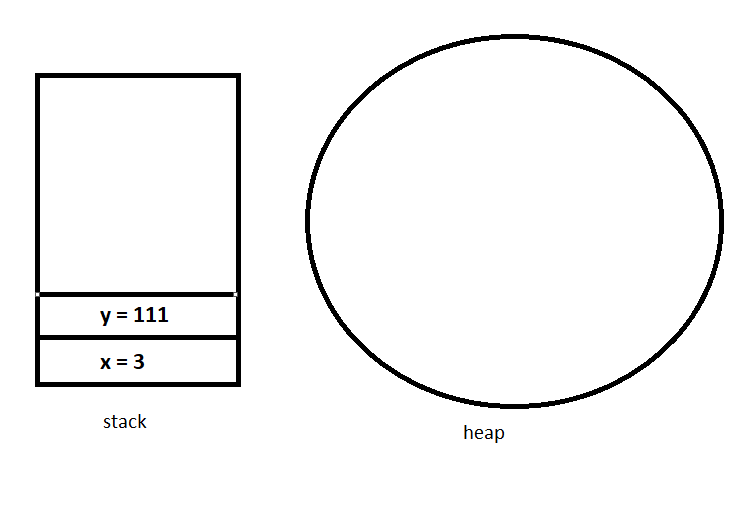
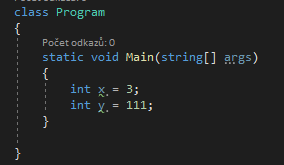
Obsahuje dlouhodobé objekty

Například statické co jsou vytvořeny na začátku programu a žijí po celý program.

Garbage Collecter zde uklící ve všech generacích nejméně krát.

**Příklady**

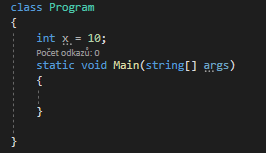
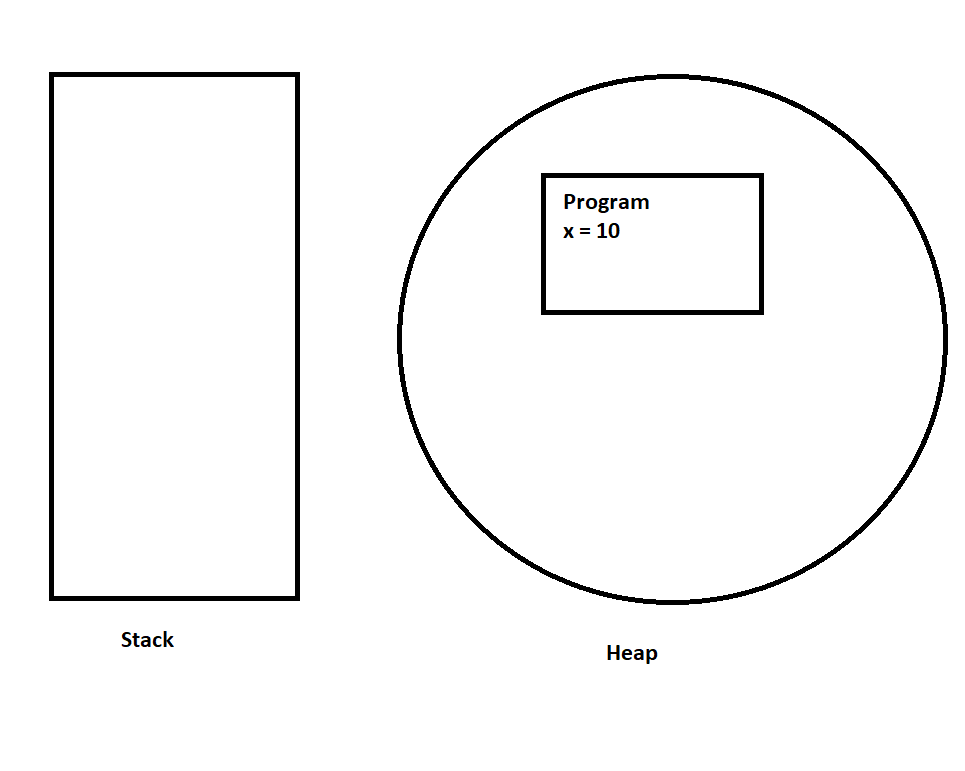
1

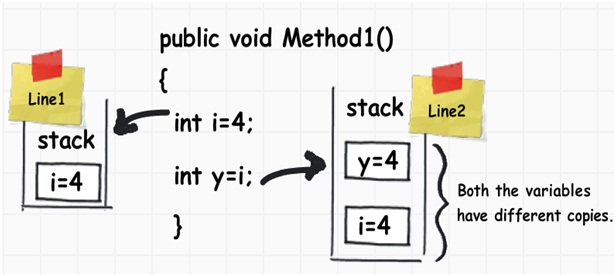


Program běží od shora, takže první na řadě je x, to je lokální proměnná, tak se dá na stack (tam je 32bit prostor pro adresu ukazují na haldu)

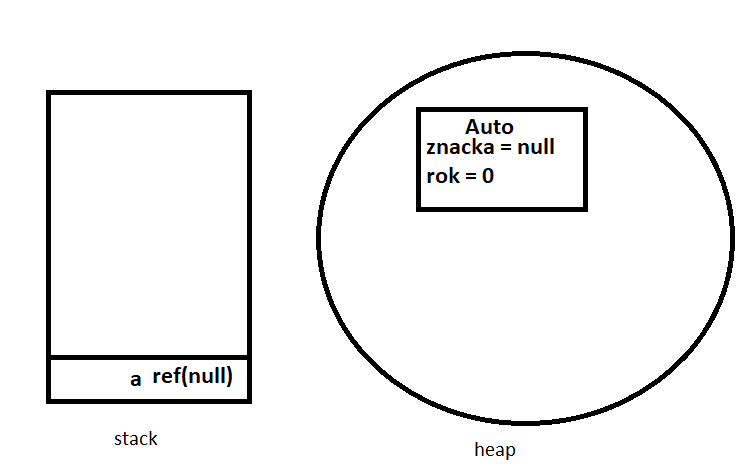
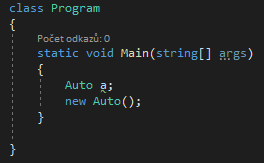
Druhá je y také lokální proměnná, takže taky na stack

Po skončení metody je stack smaže a bude prázdej



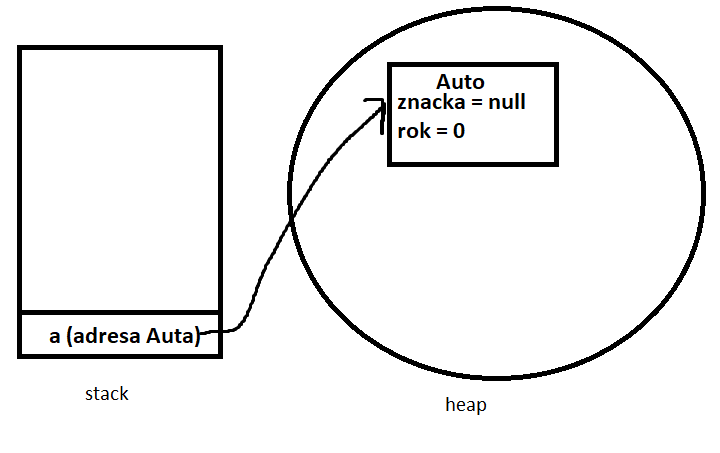
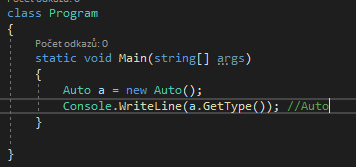
**2**



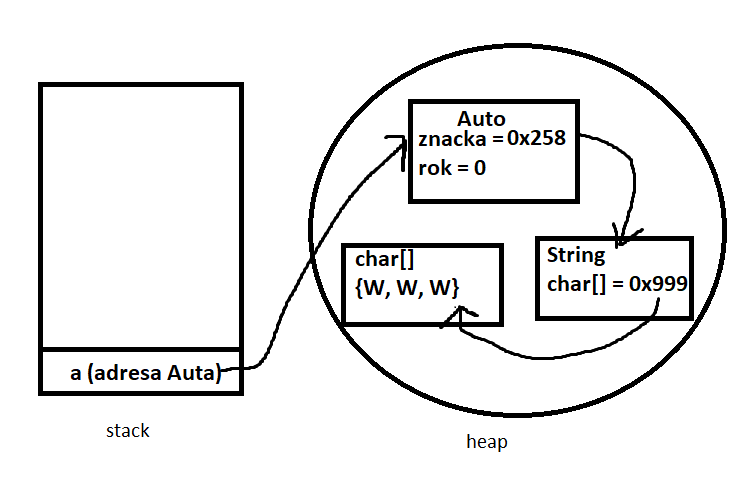
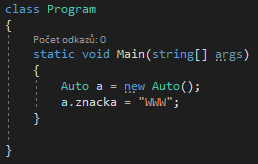
a není přiřazené k typu, protože na nic neukazuje

na Heap je Auto, protože je objet, ale nic na něj neukazuje, takže ho smaže Garbage collector

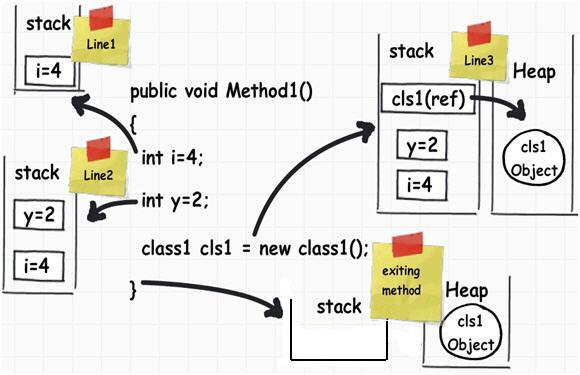
na Heap ho vytvoříl příkaz „new“



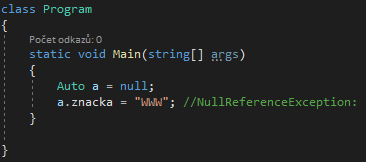
Instance bude ukazovat na Auto, po skončení Main je „a“ smaže ze stack a na Auto už nebude nic ukazovat, takže ho smaže garbage collector



Auto znacka ukazuje na String, protože String je object, ten obsahuje adresu na pole charů.

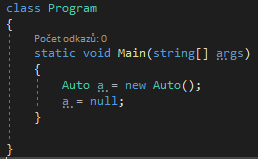


**3**



a neukazuje na žádný objekt, jeho reference je null (0x00000)

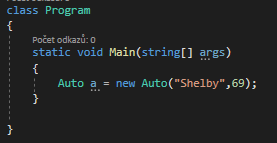
**4**

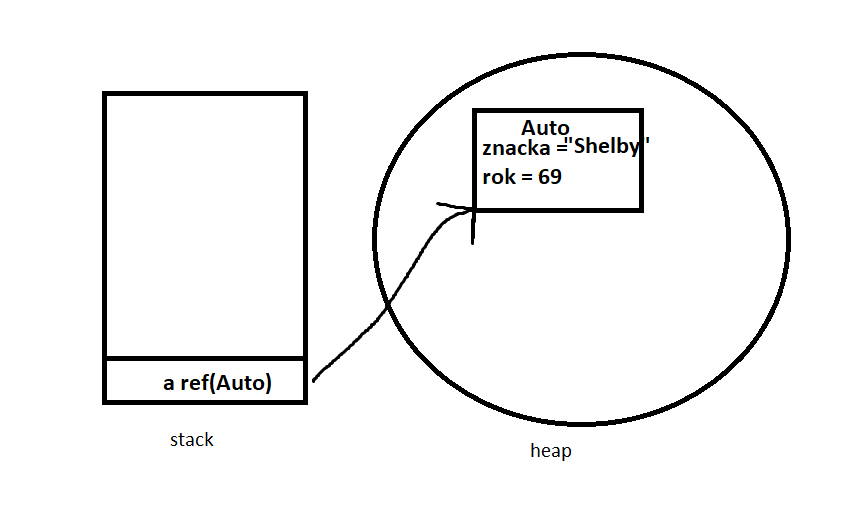


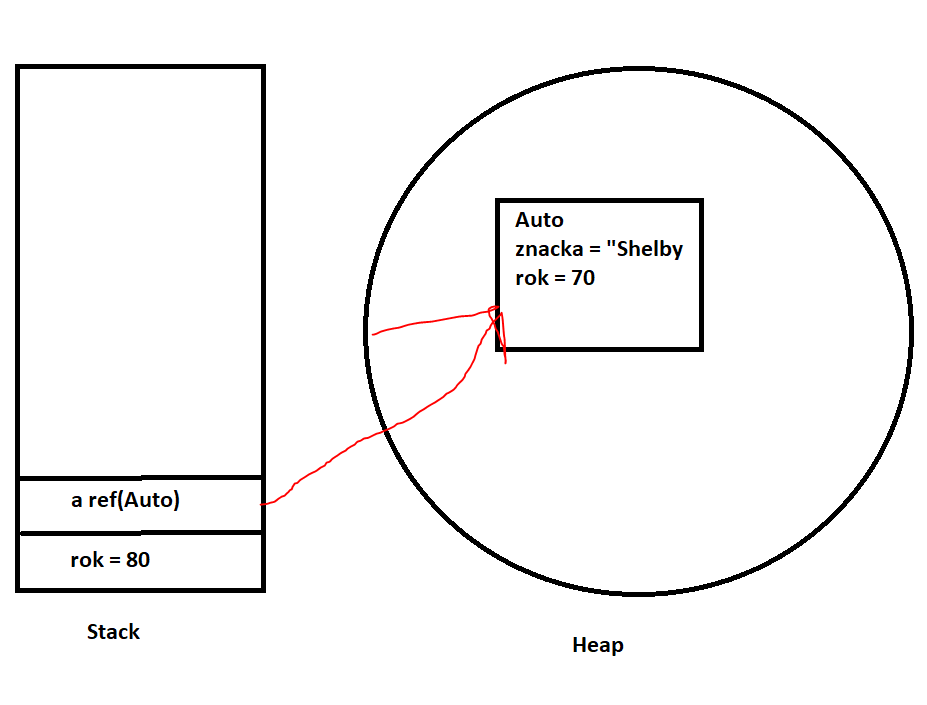
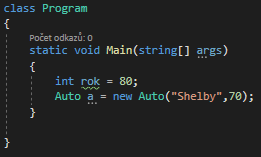
Po řádcích

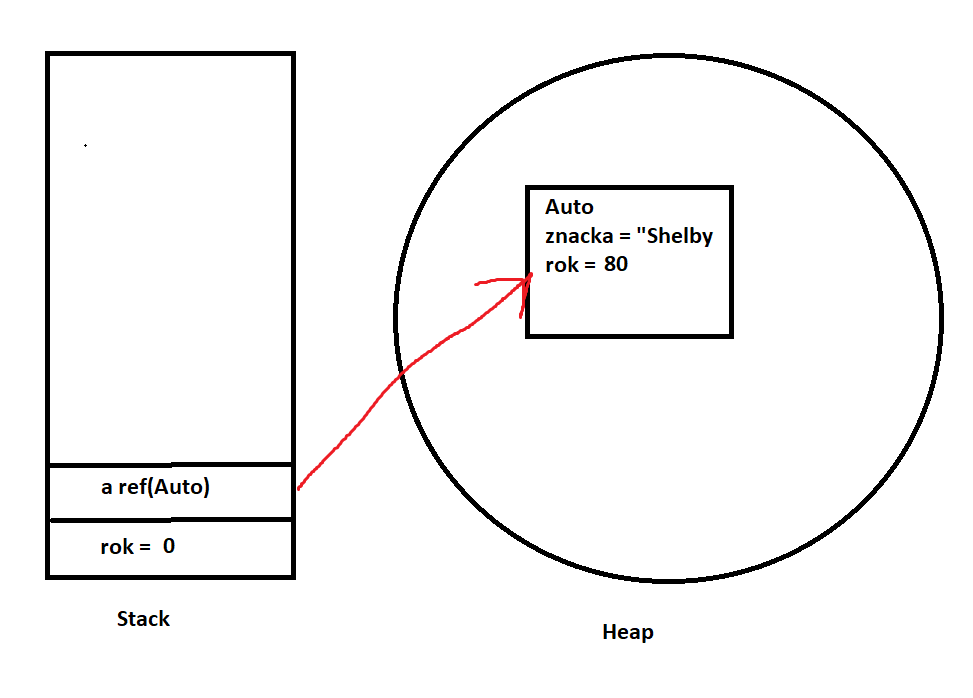
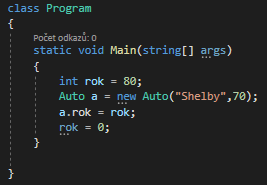
1. Stack [a ref(Auto)] Heap [Auto…]
2. Stack [a ref(null)] Heap [Auto…]

**5**



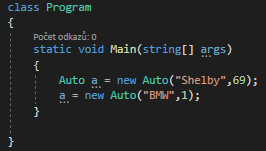


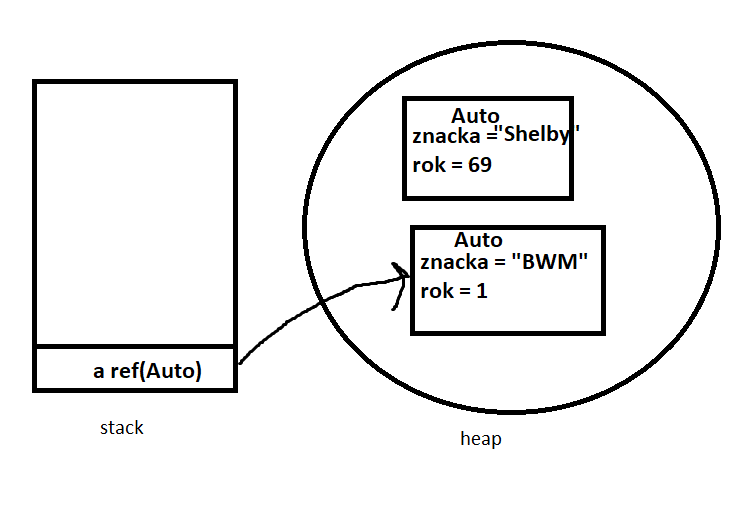




Rok se pouze zkopíruje do Auta, takže když bude tento int později upraven, tak to Auto.rok neovlivní.

Objekt z haldy nemůže ukazovat na stack.

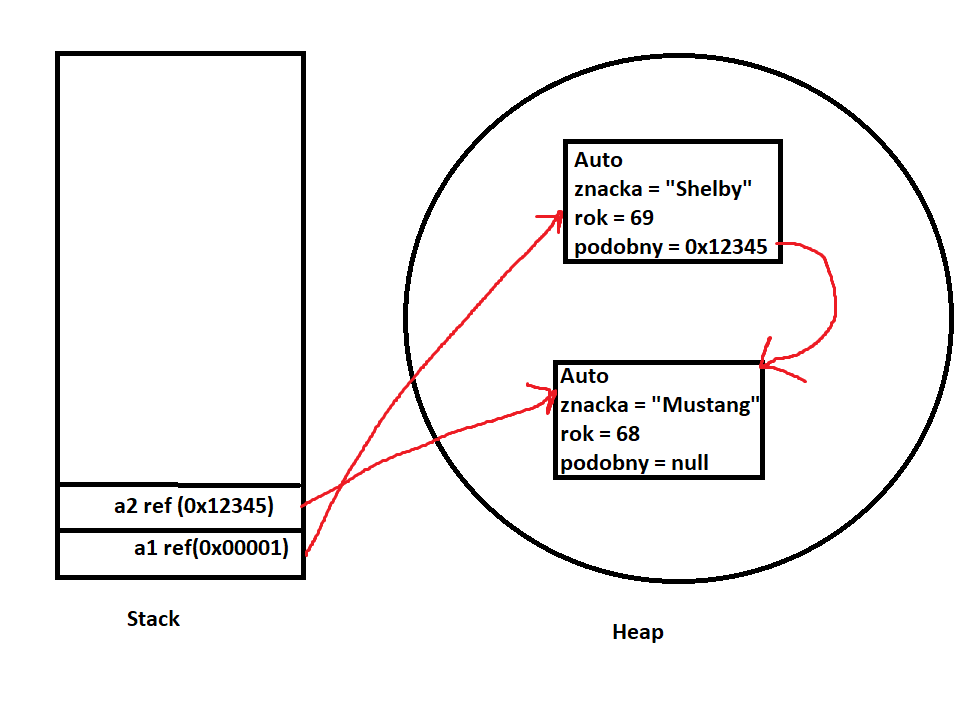
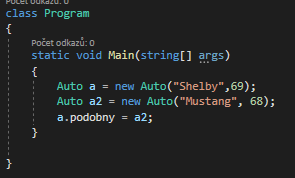




Auto Shelby bude uklizeno Garbage collectorem

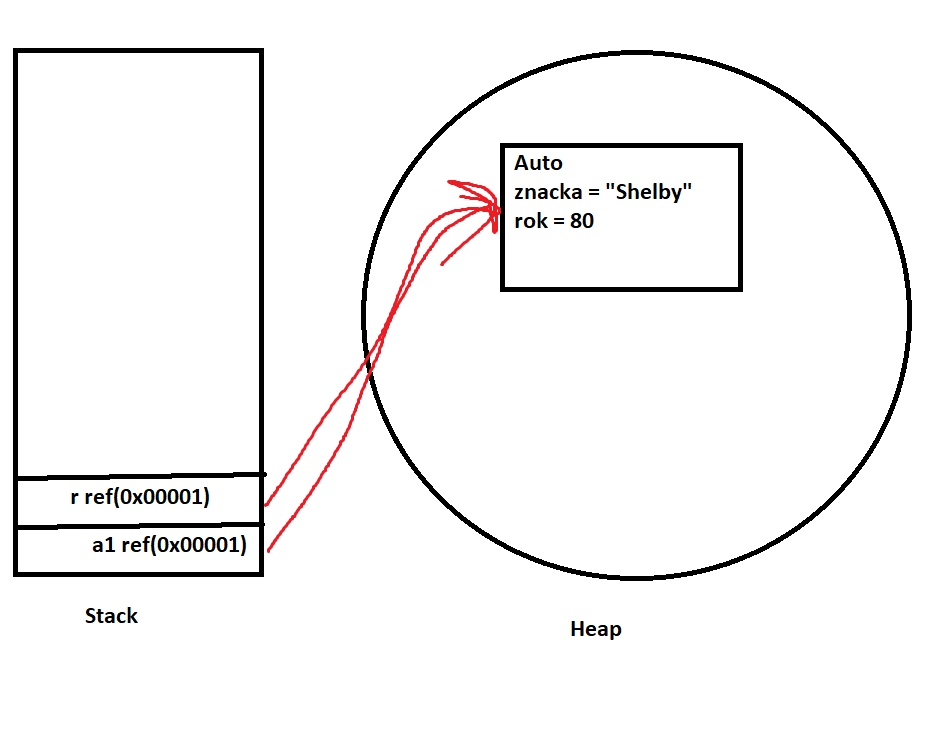
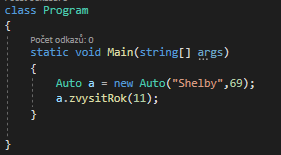
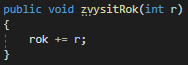
**6**

Třída obsahuje „public Auto podobny“

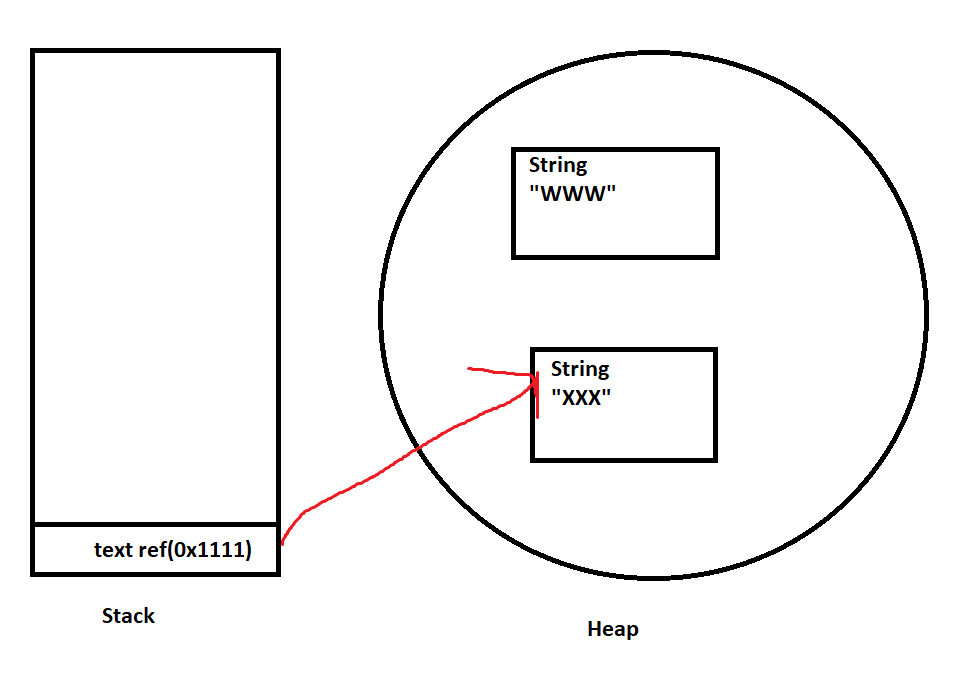
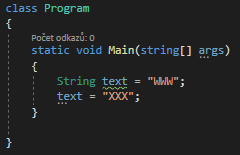


Kdyby se a2 nastavilo na null, tak Auto Mustang zůstane protože na něj ukazuje Auto Shelby

**7**



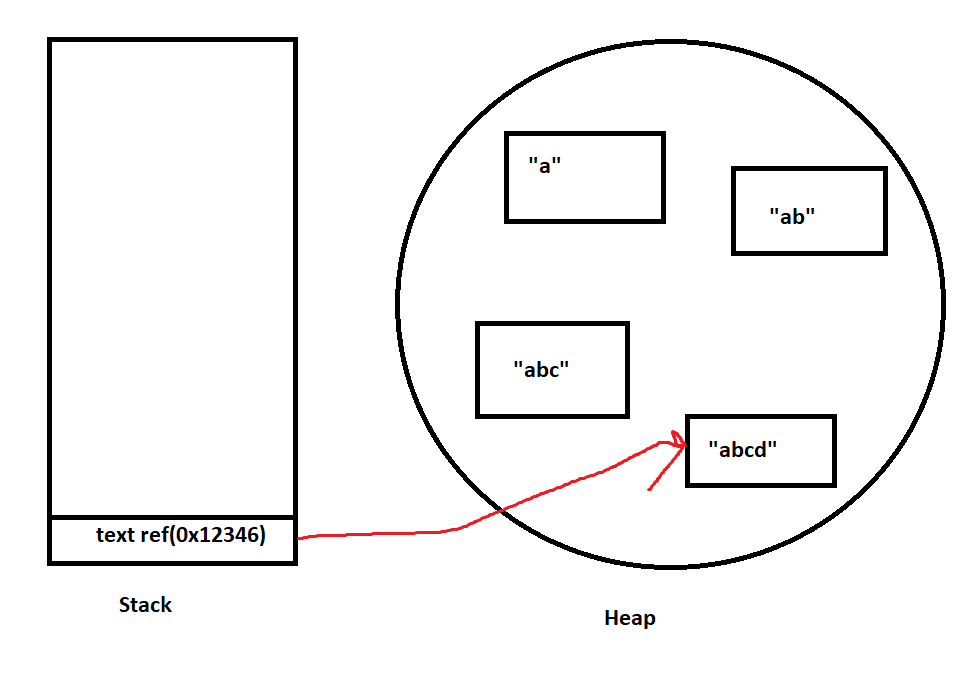
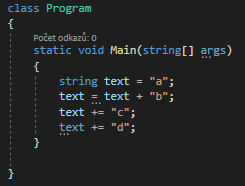
**8 String**



„WWW“ bude odstraněno garbage collectorem

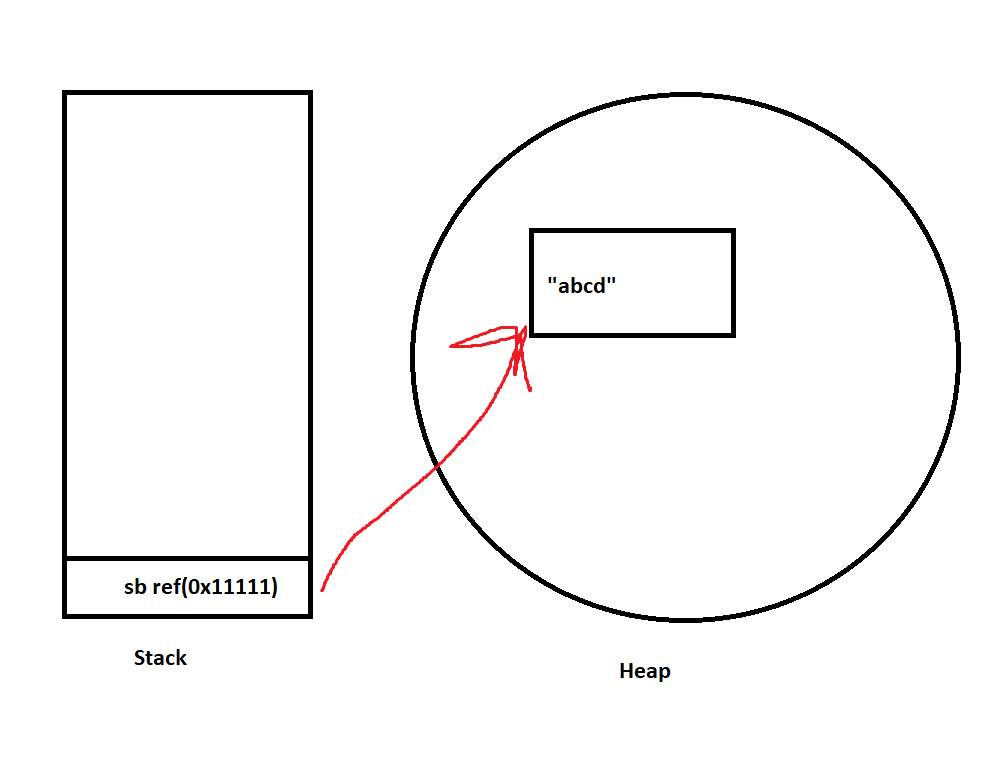
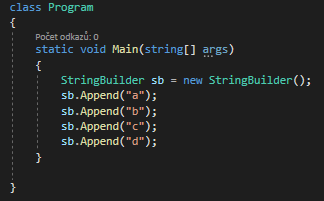
**9 string vs stringbuilder**

string



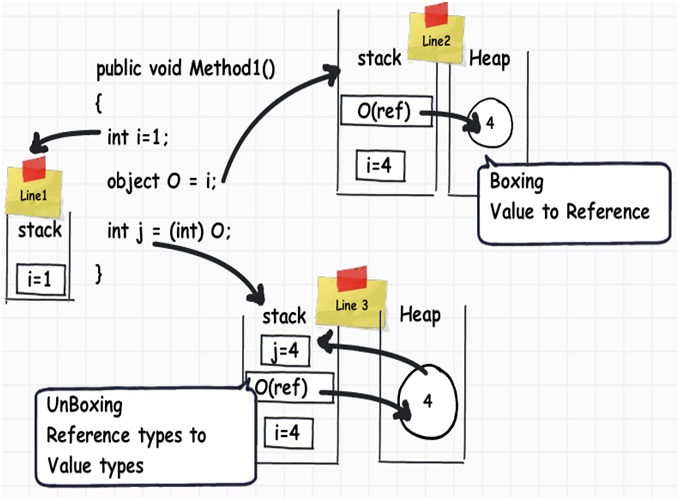
Ostastní bez reference budou smazány garbage collectorem

**StringBuilder**



StringBuider dokáže modifikovat instanci na haldě, takže nevytváří nové jako string a jen upravuje. Garbage collector nic neudělá

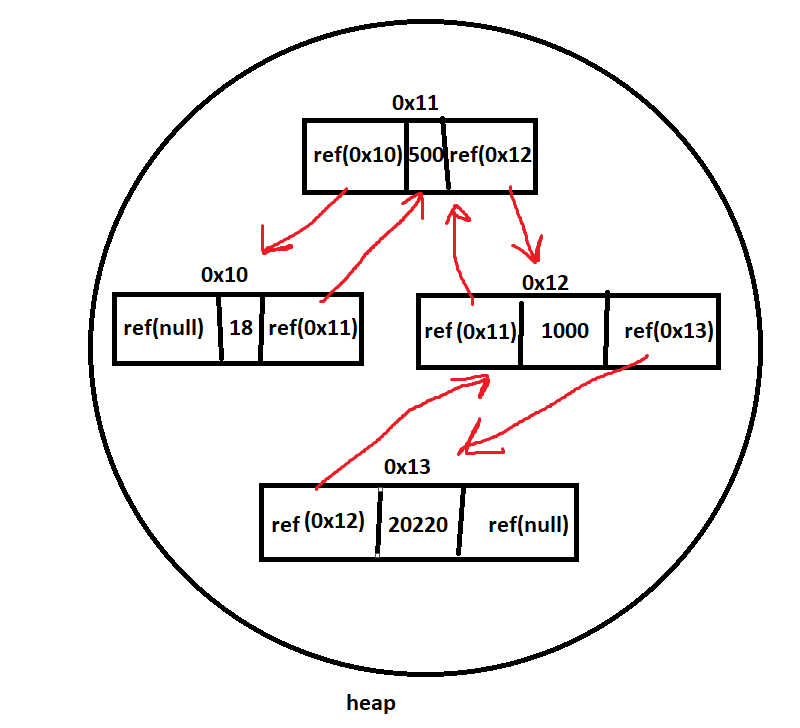
**10**



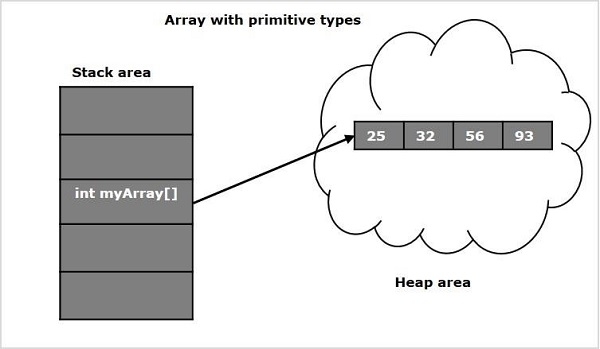
**Kolekce**

**LinkedList**

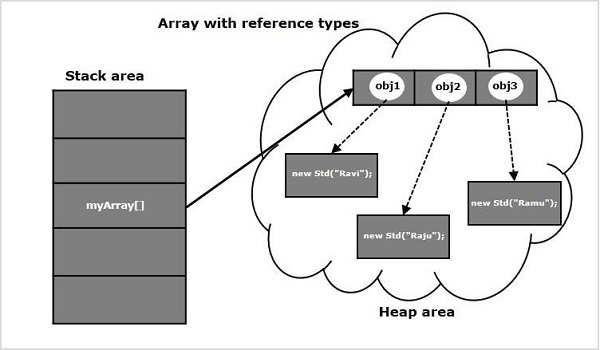
Linked list je kolekce kde prvek obsahuje adresu předchozího, data a adresu následujícího



**Ostaní**

S primitivním typem 

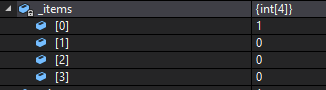
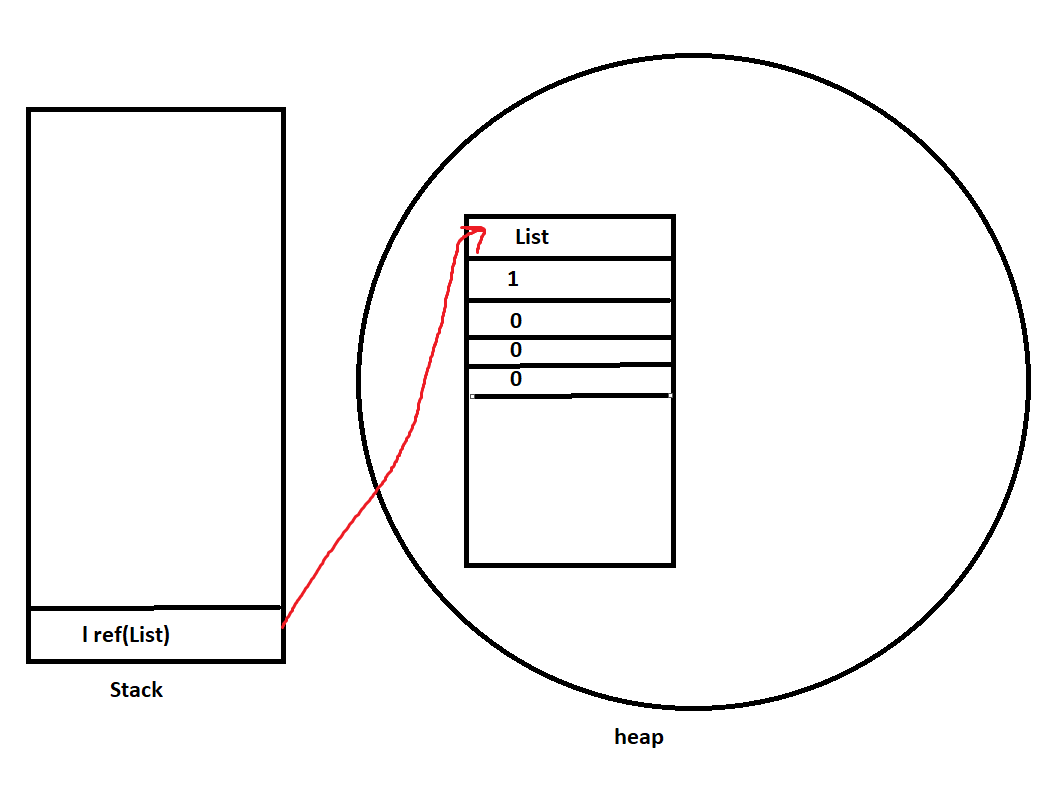
S objektovým typem



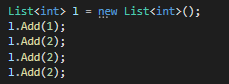
Pole je indexováno od 0, protože 0. prvek je báze a při výpisu jiného prvku nemusíme procházet ostatní. Číslo indexu říká, kolikrát se má ukazatel na prvek posunout. Když ze začátku ukazuje na bázi (první prvek), tak se posune 0krát

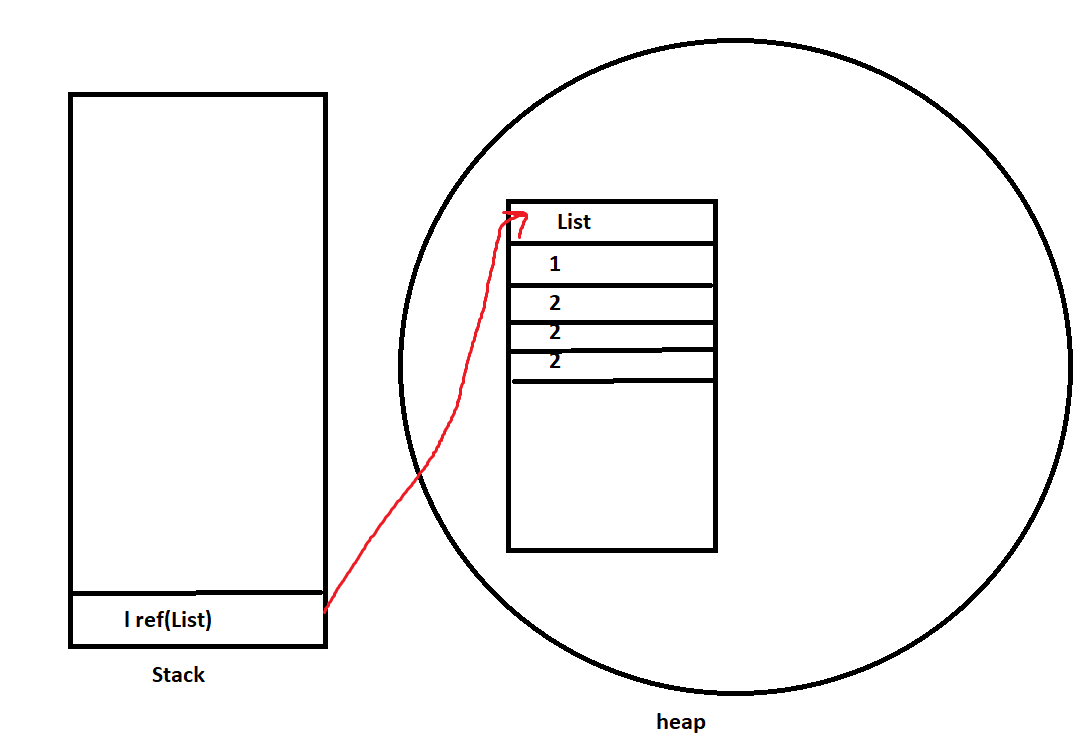
**Dynamické kolekce**



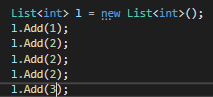
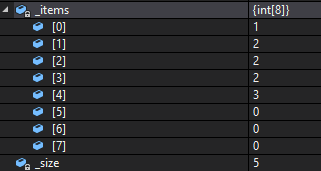
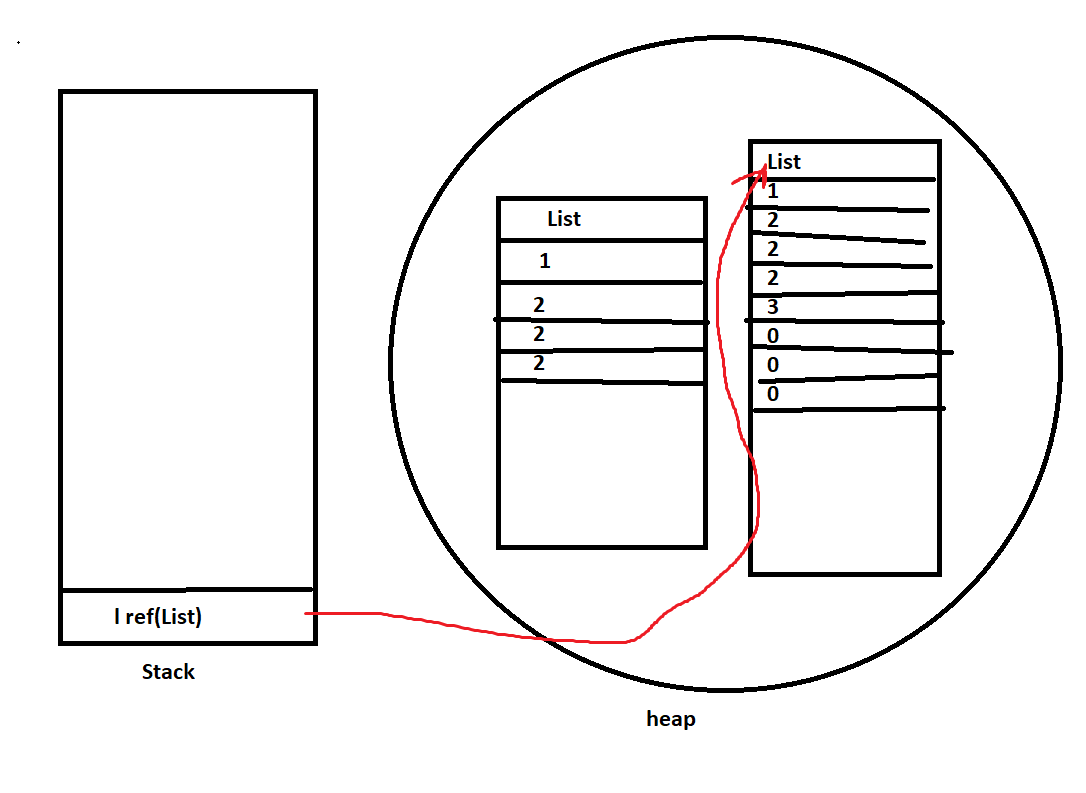
 

Potom



Potom list ze už do pole nevejde tak vytvoří jiné a to staré tam překopíruje, přídá prvek a l na stacku změní ref na něj, staré pole bude smazáno garbage collectorem

Až poté bude mít víc jak 8 prvků tak se zase vytvoří nové a zkopíruje staré a přidá hodnoty