Programozási nyelvek Java

Kozsik Tamás előadása alapján
Készítette: Nagy Krisztián

3. előadás

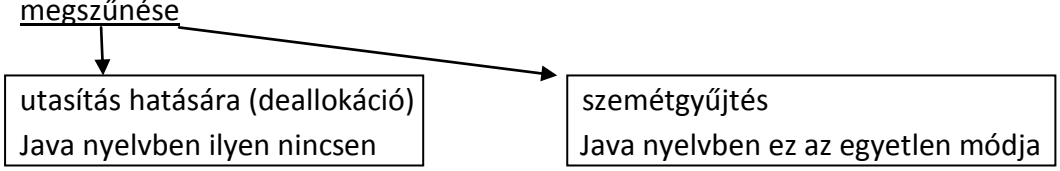
HEAP-en objektumot tárolunk.

Automatikus változók

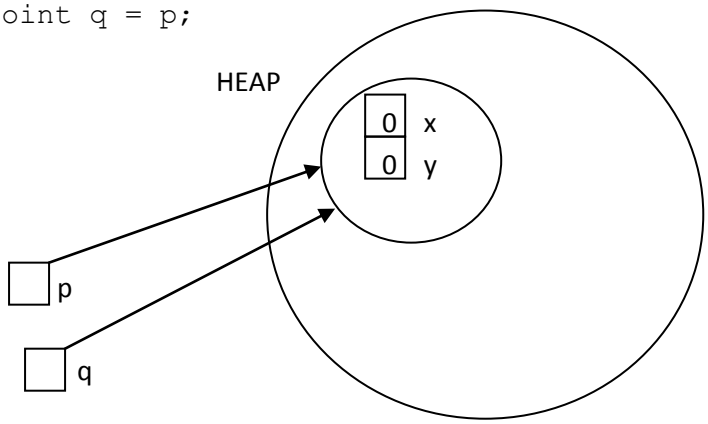
- végrehajtási vermen (execution stack) jönnek létre
- alprogramok lokális változói (beleértve a formális paramétereknek megfelelő lokális változókat)
- primitív típusok és referenciák lehetnek
- automatikusan jön létre és automatikusan szűnik meg a stack-en

Dinamikus változók

- dinamikus memória területen (heap) jönnek létre
- objektumok, mezőik
- objektum, primitív típusok vagy referenciák lehetnek
- utasítással hozzuk létre (allokáció) a new kulcsszó segítségével
  megszűnése



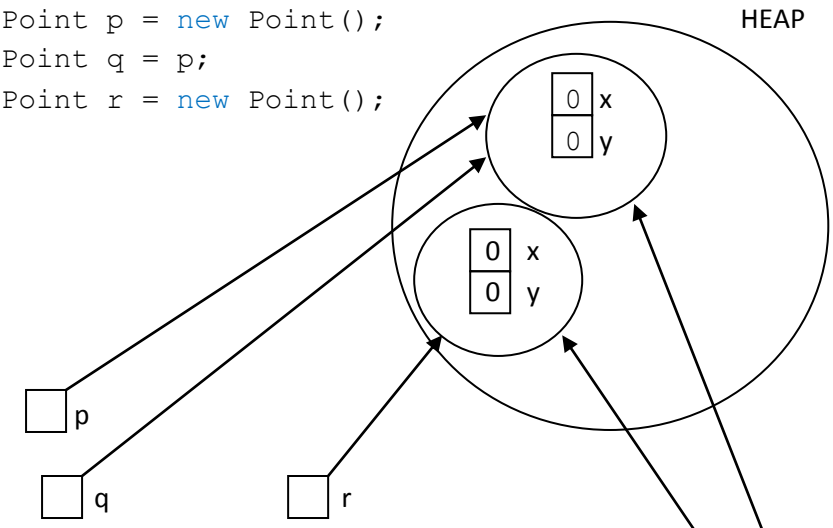

```
Point p = new Point();
Point q = p;
```



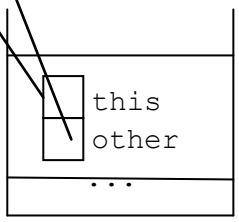
Amennyiben p és q objektum mezőjeként van implementálva, úgy a heapen, ha alprogramban, akkor a stacken tárolódik. A fentebbi kis példa az aliasing-ot mutatja be, mikor is ugyan arra a referenciára több hivatkozás is van. Ilyenkor fontos arra figyelnünk, hogy például p.x = 1; és q.x = 2; utasítások végrehajtásakor az első és a második utasítás is ugyan azon objektum x mezőjének értékét fogja

módosítani, ellenben azzal a látszattal, hogy a p = q val két ugyan olyan tartalmú objektumot hoztunk létre. Tehát az utasítások lefutása után a Point objektum x mezőjében 2-es fog szerepelni, akkor is, ha p által iratjuk ki az x-et és akkor is, ha q által!

```
Point p = new Point();
Point q = p;
Point r = new Point();
```




```
class Point{
      int x,y;
      …
      void moveToMe(Point other){
            other.x = this.x;
            other.y = this.y;
      }
      void set (int x, int y){
            this.x = x;
            this.y = y;
      }
}
```



STACK

```
r.moveToMe(p);
```

Ha p.x-ben például 5 szerepelne visszaíródna az értéke 0-ra

```
class Point{
      int x,y;
      …
      void moveToMe(Point other){
            other.x = this.x;
            other.y = this.y;
            other = new Point();
      }
      void set (int x, int y){
            this.x = x;
            this.y = y;
      }
}
```

indirekt hatás p(az akt. paraméter) által hivatkozott mező értéke változik

az aktuális paramétert nem befojásolja

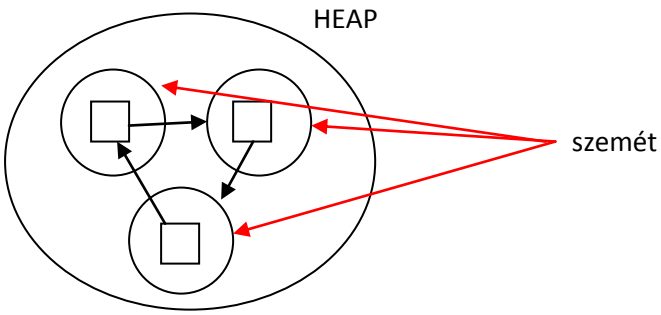
Létre jön egy olyan objektum a heapen, amire nem mutat semmi. A Java nyelvben a Garbage-collector megtalálja és felszabadítja az objektumnak allokált helyet!

Megjegyzések :

Fontos tudni, hogy a szemétgyűjtés egy költséges folyamat, ezért a Java nyelv hatékonyság szempontjából roszabb például a C++-nál. C++ nyelvben nekünk kell számon tartanunk a létre hozott objektumokat és ezen objektumok megszüntetésével is a legtöbb esetben nekünk kell foglalkozni. Amennyiben ezt elhalasztanánk, úgy Memória leak keletkezhet.

Pár szó a Garbage collector működéséről:

Megnézi, hogy a végrehajtási verem akivációs rekordjaiban milyen hivatkozások vannak, majd ezen hivatkozásban vannak-e további hivatkozások és azokban is vannak-e továbbiak …, amelyik objektum ily módon nem érhető el az szemét.

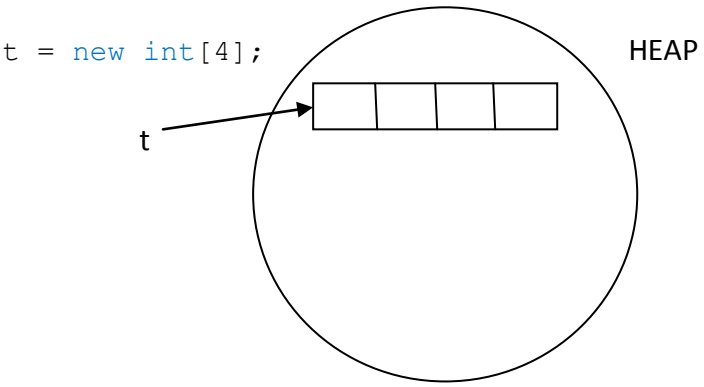


Tömb

A Java nyelvben objektum.
A tömbtípus hasonlít az osztályhoz.
String[]
int[]
int[][]

A new kulcsszóval hozható létre, ebből adódóan a heap-en jön létre.
Meg kell adni a méretét is.
A tömb típusú változó egy referencia.

A tömb egy olyan adatszerkezet, melynek elemeit gyorsan megtaláljuk a memóriában. (Indexelhető)



Aritmetikai műveletekkel meg tudjuk mondani az x. elemét.
t legyen a tömb kezdete, i az indexe, s egy tömbelem mérete, ekkor
$t + (i - 1) \cdot s$ képlet alapján tudjuk kiszámolni az i. indexű elem helyét a memóriában.
Amennyiben 0-tól indexelünk, úgy hatékonyabb lesz a fentebbi számítás mivel, ekkor kiesik a kivonás művelet. Az így kapott képlet: $t + i \cdot s$

Ahoz, hogy eme adatszerkezetről beszélhessünk az alábbi feltételeknek kell teljesülnie:

- a tömbelemek egymás után legyenek letárolva a memóriában
- minden tömbelem ugyan annyi memóriát kell, hogy igényeljen (s ugyan akkora)
- nem változtathatjuk a méretét

Méretnövelést csak új allokálással és az előbbi tömb értékeinek egyenkénti átmásolásával tudunk elérni.

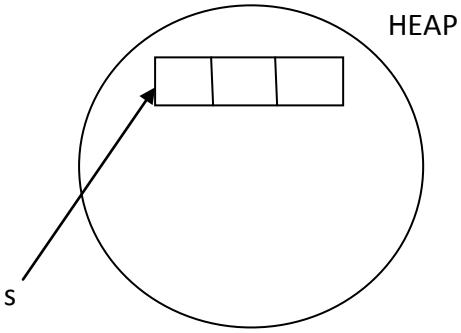
Primitív típusú tömb esetén egy tömb létrehozása kezdőértékekkel:

```
t = new int[5]{2 , 9, 7, 3, 4};
```

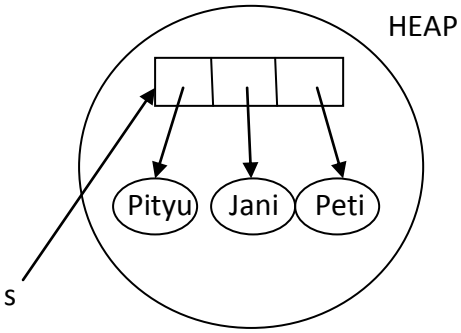
String[] ⟶ string referenciákat tartalmaz

```
String[] s = new String[3];
```

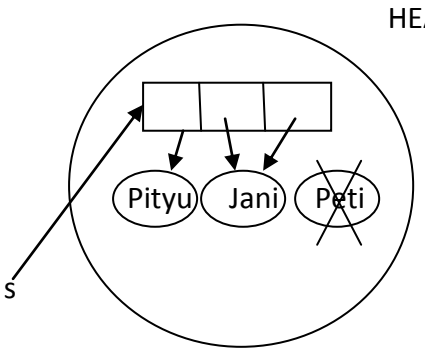
Olyan referenciák, amik nem hivatkoznak semmire!




```
s[0] = new String("Pityu");  ⇔  s[0] = "Pityu";
s[1] = "Jani";
s[2] = "Peti";
```



Itt is oda kell figyelni, hogy referenciákról van szó:
Például s[2] = s[1];



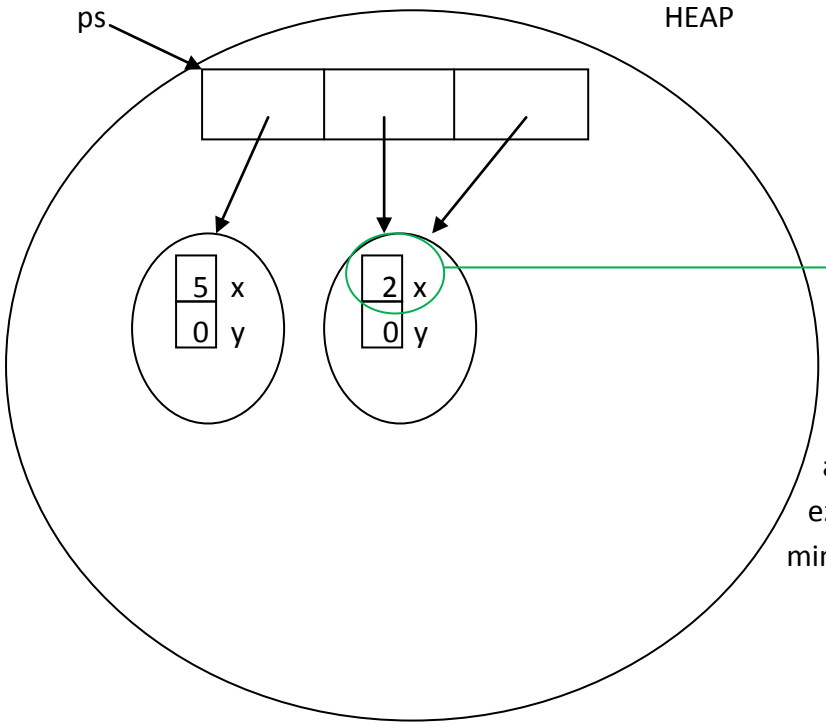
Lásd Point q = p; probléma (aliasing)!

Nézzük a következő utasításokat:

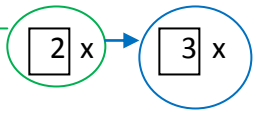
```
Point[] ps = new Point[3];

ps[0] = new Point();
ps[0].x = 5;
ps[1] = new Point();
ps[1].x = 2;
ps[2] = ps[1];
ps[2].x = 3;
```

Majd irassuk ki ps[1].x-et. A konzolon a 3-as szám fog megjelenni (a sokak által várhatott 2 helyett.) Miért? :



a ps[2].x = 3 hatására:



felülíródik a 2-es 3-asra.

Mivel s[1] és s[2] referencia ugyan arra a memóriaterületre hivatkozik ezért ps[1].x –et kiiratva ugyan olyan mintha ps[2].x –et iratnánk ki.