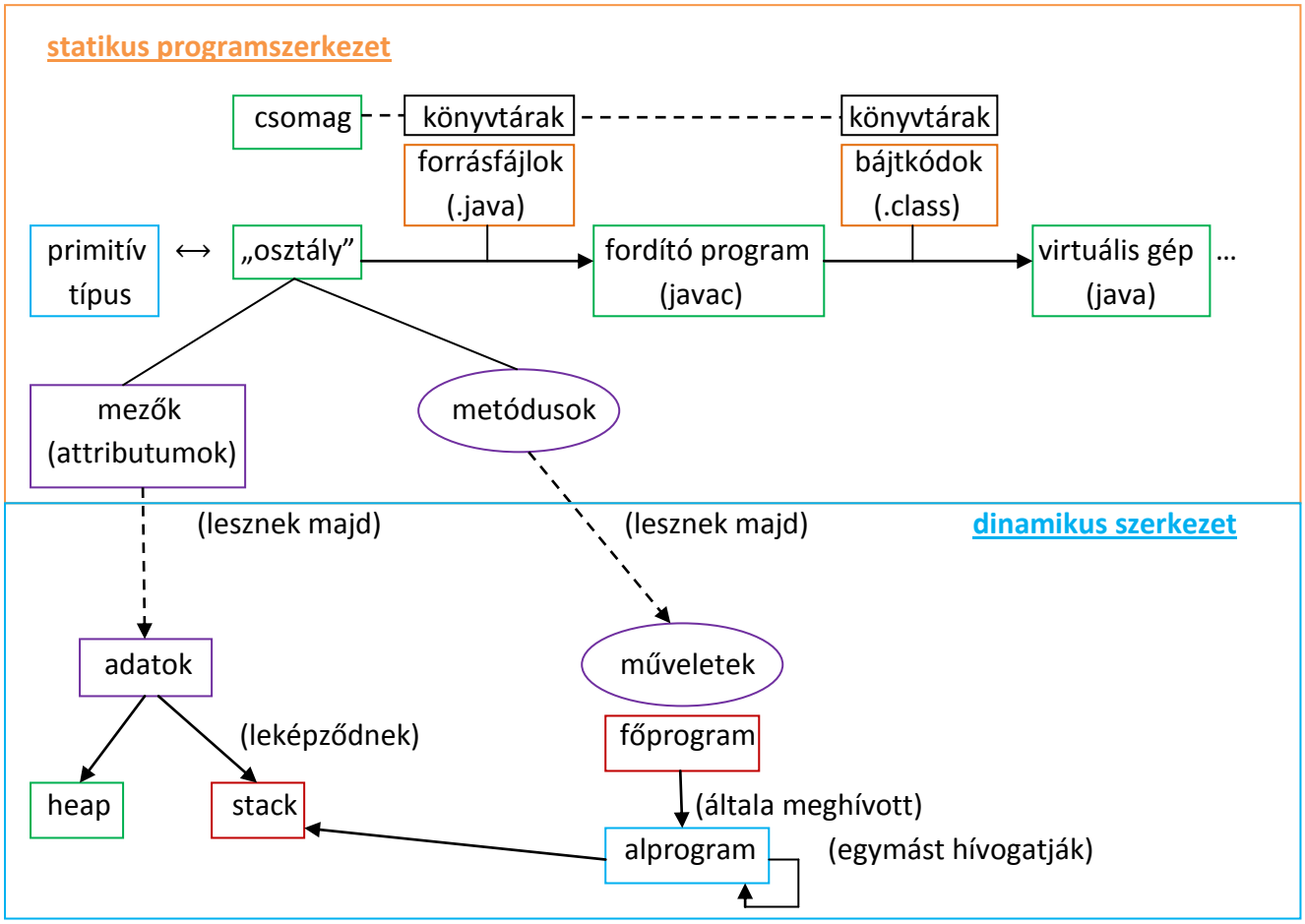
Programozási nyelvek Java

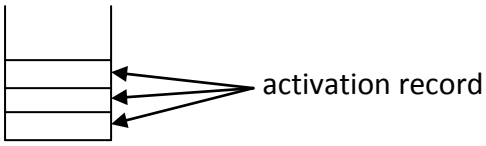
Kozsik Tamás előadása alapján
Készítette: Nagy Krisztián

2. előadás



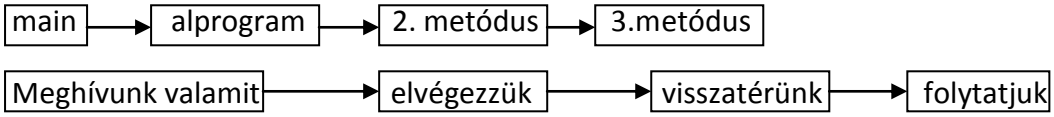
heap – dinamikus memóriaterület
stack – végrehajtási verem (execution stack)

Execution stack (végrehajtási verem)



LIFO (Last in first out)
verem-adatszerkezet

Program futása közben:
A főprogram (main) meghívja az alprogramot, az meghív egy második metódust, ez utóbbi pedig egy harmadikat.



elven működik.

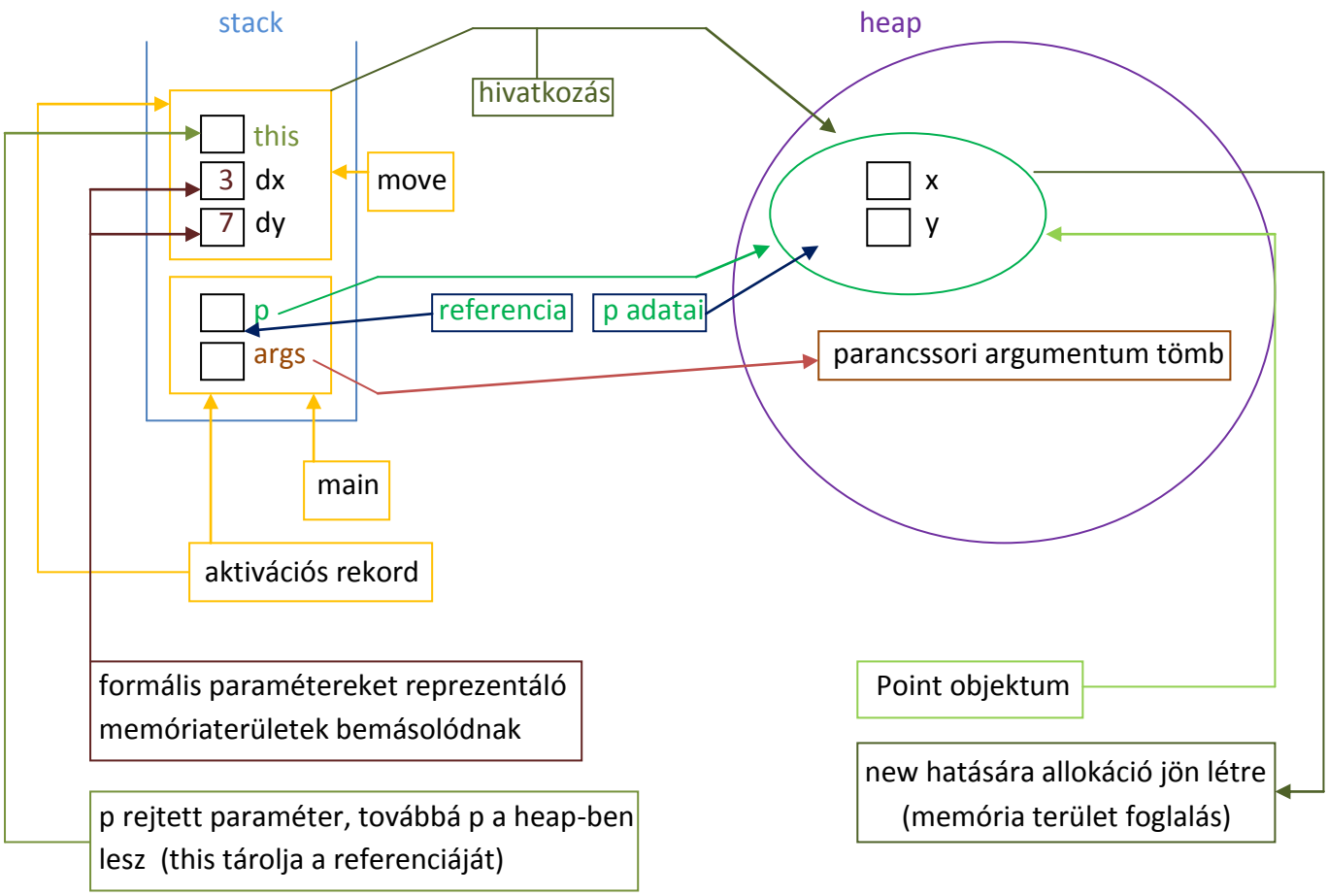
Mindig az a folyamat fejeződik be előbb, amit utoljára hívtunk meg.
A stack aktivációs bejegyzésekből (activation record) épül fel. (Alprogram hívások befejezése)
Nem teljesen pontos leírás az **aktivációs rekord tartarmáról** (Fordítóprogramok kurzushoz tartozik): Tartalmazza, hogy **hova kell visszatérnie a vezérlésnek** (honnan kell folytatni a végrehajtást), továbbá tartalmazza az **alprogramok paramétereit**.

Az első aktivációs rekord a main metódus paramétereit tartalmazza.
Amennyiben befejeződik egy metódus feldolgozása, úgy törlődik az aktivációs rekordja.
Akkor fut le a programunk / fejeződik be a teljes végrehajtás, ha a main metódus végrehajtását befejeztük, azaz megszűnik a főprogram aktivációs rekordja.

Vegyük az alábbi kódot és illusztráljuk rajta a keletkezett stacket és heapet

```
class Point{
        int x,y;
        void move(int dx, int dy){
             x += dx;
             y += dy;
        }
// Galád módon beépítjük a főprogramunkat is ebbe az osztályba csak
// a példa kedvéért
        public static void main(String[] args){
             Point p = new Point();
             p.move(3,7);
        }
}
```

1.) javac Point.java java Point (**Absztrakt adatmodell**)



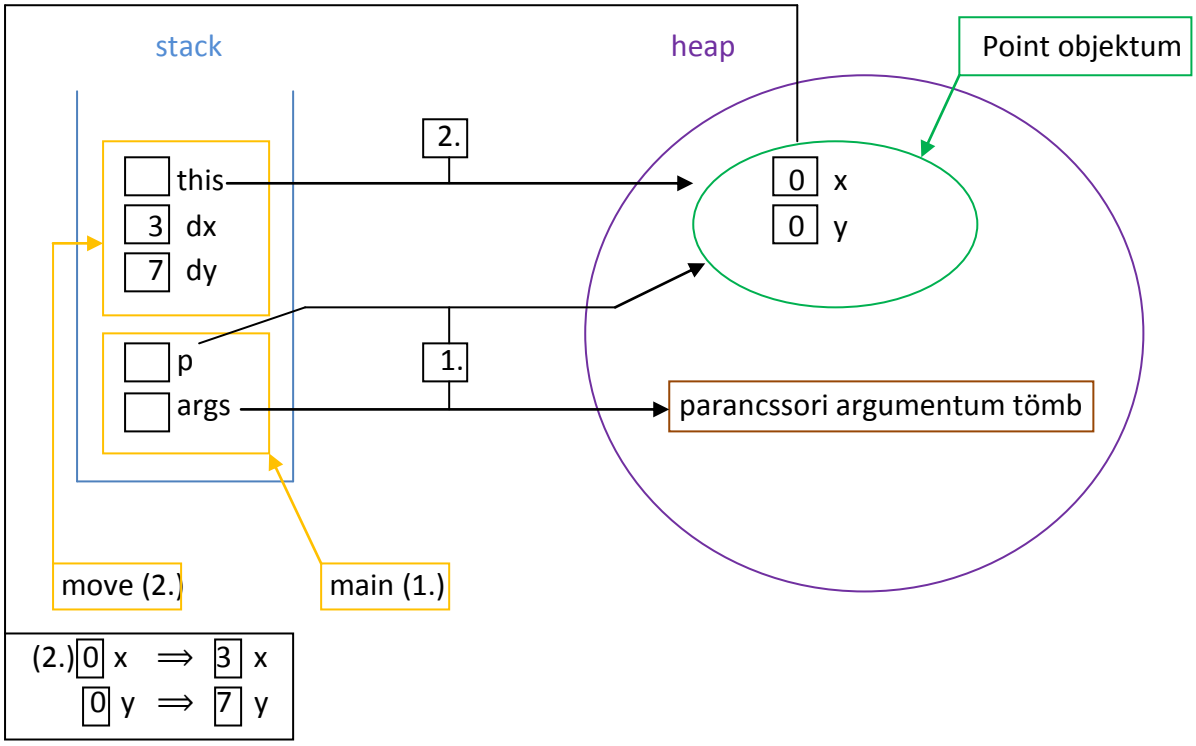
A végrehajtási verem hoz létre objektumot.
Az objektum mindig a heapen tárolódik.
Primitív adatok esetén, amennyibben objektum mezőjéről van szó, úgy ezen mezők a heapen tárolódnak. Amennyiben egyes metódusok lokális változóiról van szó, mint primitív adattagok, beleértve a formális paramétereket reprezentáló változókat is, úgy ezen adatok a stacken tárolódnak.

A stacken a 8 primitív adattípuson kívül referencia is tárolódhat, melynek az a dolga, hogy mutason egy a heapen található objektumra.

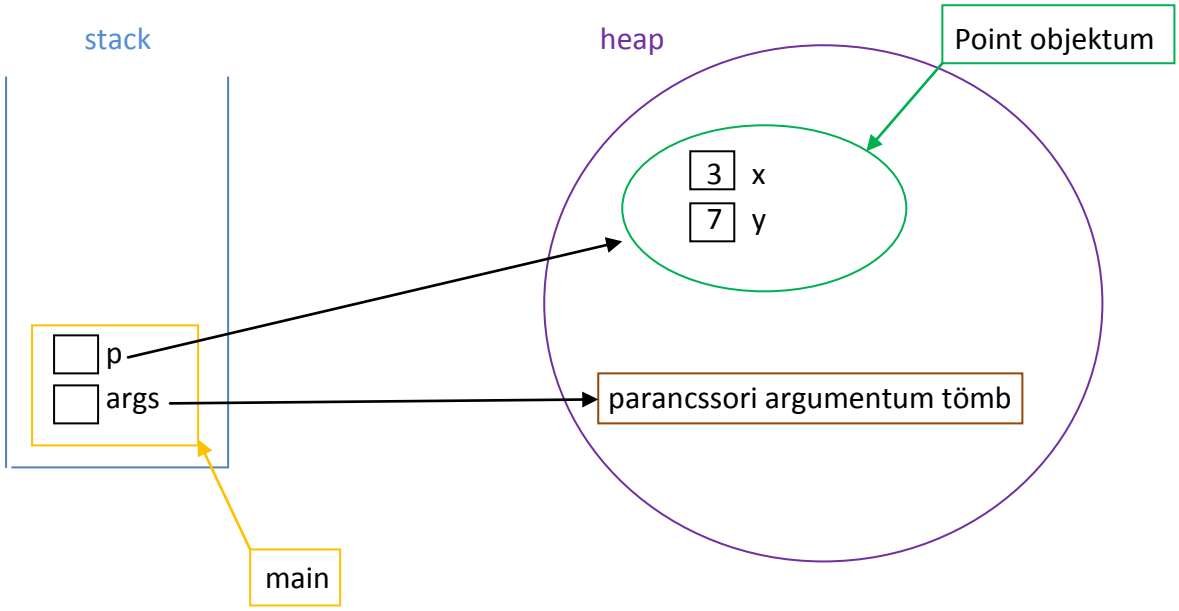
(Megjegyzés: Manapság már lehet, hogy nem minden adat a stacken tárolódik, előfordulhat, hogy egyes adatok registry-kben.)

Amikor a fentebbi move metódus paraméter listáját megadtuk:
`void move(int dx, int dy)`, (dx-et és dy-t), fontos tudnunk, hogy nem csak a két megadott formális paraméterünk létezik az eljárás szempontjából, hanem egy implicit paraméterünk is van a this. A fentebbi eljárás, amit leírtunk valójában így néz ki:

```
void move(int dx, int dy){
      this.x += dx;
      this.y += dy;
}
```



⇓ (move-ból kilépve az adott aktivációs rekord a benne tárolt változókkal együtt megszűnik:



A fentebbi esetben, amennyiben megtoldjuk a main fügvényünket a System.out.println(p.x); utasítással, úgy a 3 fog kiíródni.

Nézzük a következő kódot:

```
class Point{
        int x,y;
        void move(int dx, int dy){
             if(dx < 0) dx = 0;
             if(dy < 0) dy = 0;
             x += dx;
             y += dy;
        }
// Galád módon beépítjük a főprogramunkat is ebbe az osztályba csak
// a példa kedvéért
        public static void main(String[] args){
             Point p = new Point();
             p.move(3,7);
        }
}
```

Ebben az esetben a fentebbi második lépés változik meg annyiban, hogy a move aktivációs rekordjában amennyiben negatív dx,dy-t adtunk meg például -3, -7, úgy bekerül az aktivációs rekordba, majd az if-ek miatt felülíródik 0, 0 –ra és ez az eredmény lesz tovább küldve a Point objektum x,y értékének. A tovább küldés után a move aktivációs rekordja törlődik.

Módosítsuk tovább a main metódust az alábbira:

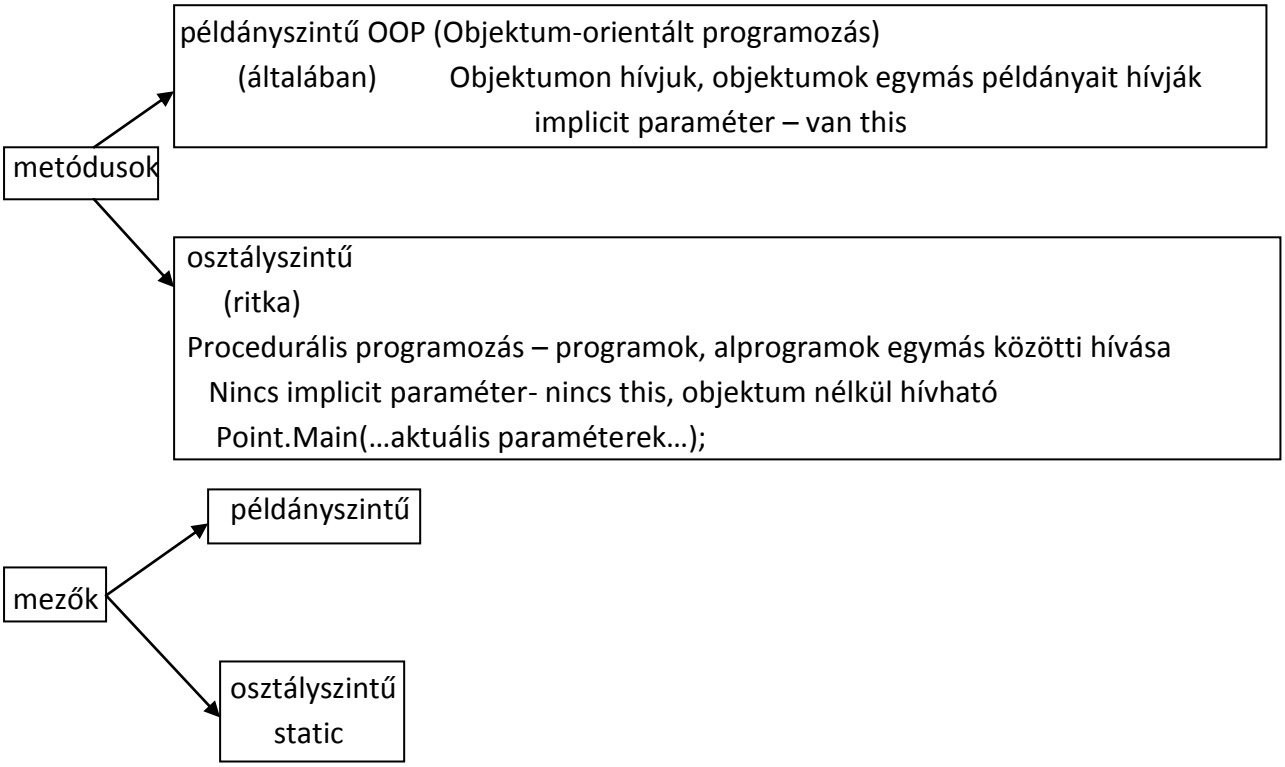
```
public static void main(String[] args){
      Point p = new Point();
      p.x = -3;
      p.y = -7;
      p.move(p.x,p.y);
}
```

Ebben az esetben p.x és p.y bemásolódik a heapbe. (Kód 3. és 4. sora miatt)
Továbbá a move metódusnak megadott aktuális paraméterek értéke bemásolódik az aktivációs rekordban a formális paraméterek reprezentálására létrejövő „lokális változókba”. Ezzel dolgozik az alprogram (metódus) és kilépéskor ezek megszünnek. Az imént említett folyamatot érték szerinti paraméter átadásnak nevezzük. (Pass by value)
A Java nyelvben csak értékszerinti paraméter átadás történik (a referenciák „értéke” is másolódik)

A static kulcsszó

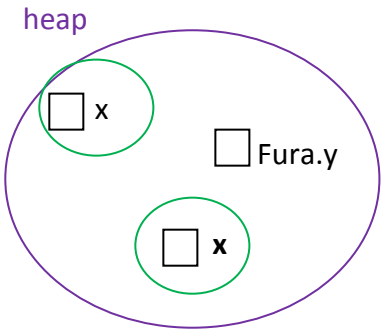
Használatával „eltüntetjük” az adott metódusok, mezők implicit paraméterét.
Nincs implicit paraméter – nincs this

A main metódusnak például tényleg csak 1 formális paramétere van az args tömb.
Ellenben az előző példákban látott move függvénnyel.



A fentebbi példákban, amennyiben `static int y;-t` írunk, abban az esetbentöbb példány esetén minden példányban lesz x attributum, de y-ból csak egy darab fog létezni a programban.

```
class Fura{
        int x;
        static int y;
        void move(int dx, int dy){
              x += dx;
              y += dy;
        }
         … main …
}
```



~~static~~ void move(int dx, int dy) (x és y példány mezők):
Statikus metódusban nem lehetnek példány szintű mezők. Példányszintűben lehet mind a kettő (statikus is és példányszintű is).