Java IV. – Típusok

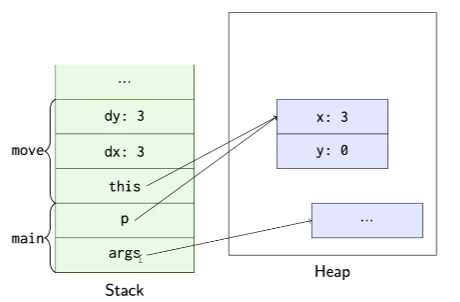
# Változók tárolása

## Primitív típusok

* byte:
* short:
* int:
* long:
* float: 4 bájt
* double: 8 bájt
* char: 2 bájt (előjel nélküli egész)
* boolean:

## Referencia típusok

Osztály típusú változó, lényegében egy pointer, ami az adott osztály egy objektumára hivatkozhat. Az objektumok mindig referencia típusúak (nem úgy, mint a C++-ban).



p.move(dx,dy) művelet szemléltetése.

# Ábrázolás a memóriában

|  |  |
| --- | --- |
| Végrehajtási verem(stack) | Dinamikus memória(heap) |
| Lokális változók, paraméterek | Objektumok és mezőik |

# Hatókör és élettartam

## Élettartam

A lokális változók élettartama hatókörük (deklarációjukat tartalmazó blokk) végéig tart.

A változók el tudják fedni egymást, de csak akkor ha nem mindkettő lokális.

class Point {

int x = 0, y = 0;

void foo( int x ){ // OK

int y = 3; // OK

{

int z = y;

int y = x; // Fordítási hiba

...

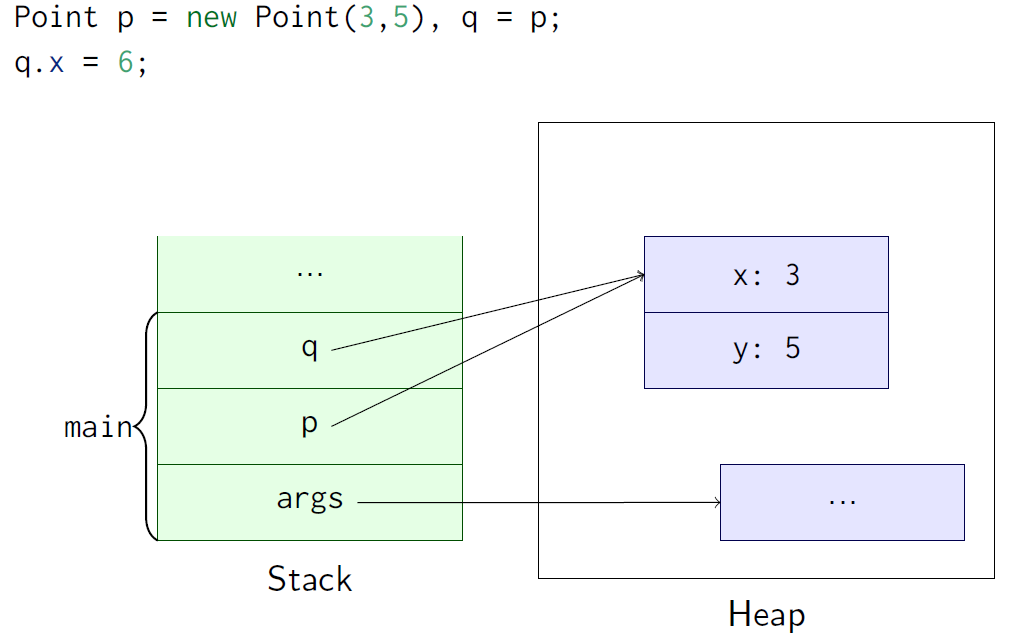
}

}

}

Az objektumok élettartama addig tart amíg minden rájuk vonatkozó referencia meg nem szűnik. Ekkor az objektum felszabadítása automatikusan megtörténik a szemétgyűjtés (garbage collection) révén.

Egy objektumra több referenciaváltozó is vonatkozhat, ezt nevezzük aliasing-nek.



# Inicializáció

A mezők automatikusan nulla szerű értékre inicializálódnak az objektum létrehozásakor. Számoknál 0, logikai értékeknél false, referencia típusoknál pedig a null.

A lokális változóknál nincs automatikus inicializáció.

Fontos, hogy a változóknak minden vezérlési útvonalon értéket adjunk (ún. definitely assigned-ak legyenek), ellenkező esetben a fordítási hibát kapunk.

{

int k;

int n = 5;

if (n > 2) k = 3;

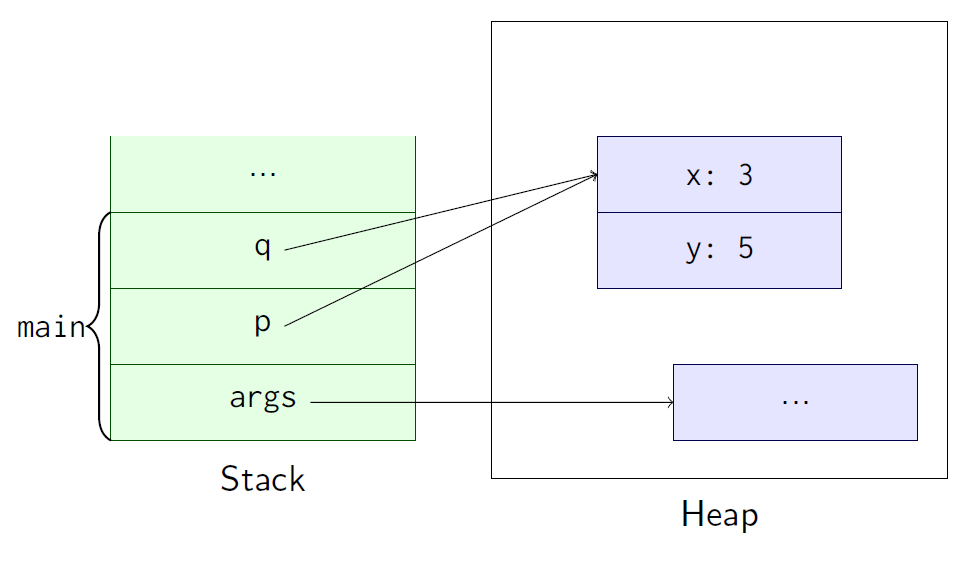
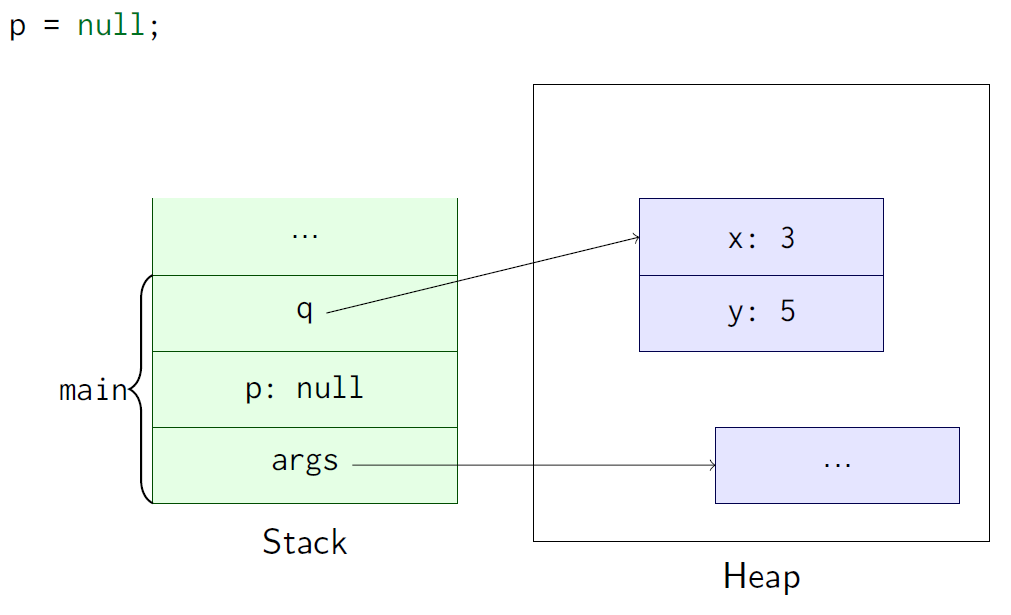
System.out.println(k);

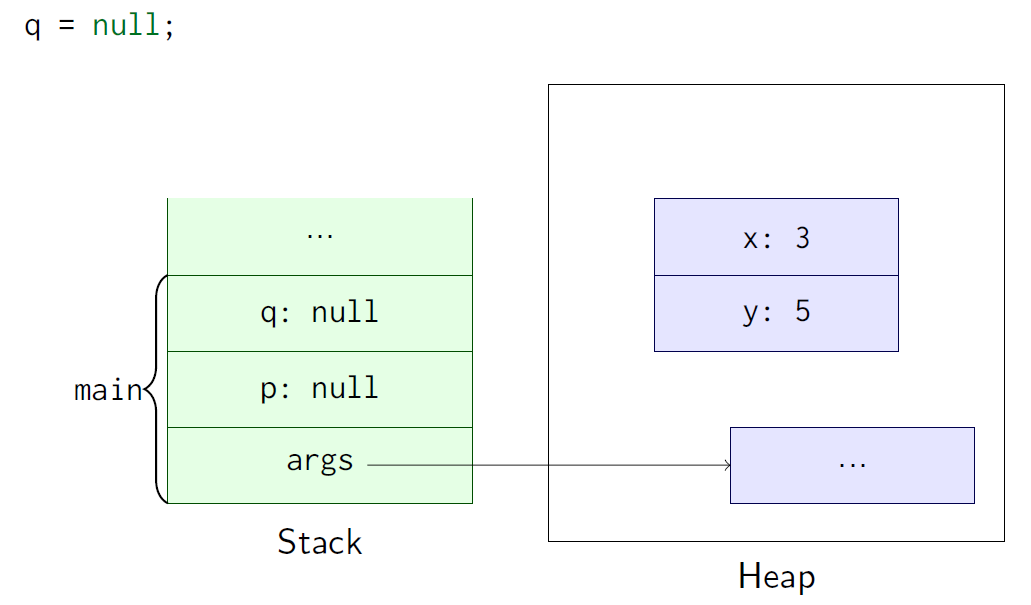
/\* k is not "definitely assigned” before this statement \*/

}

# A szemétgyűjtés

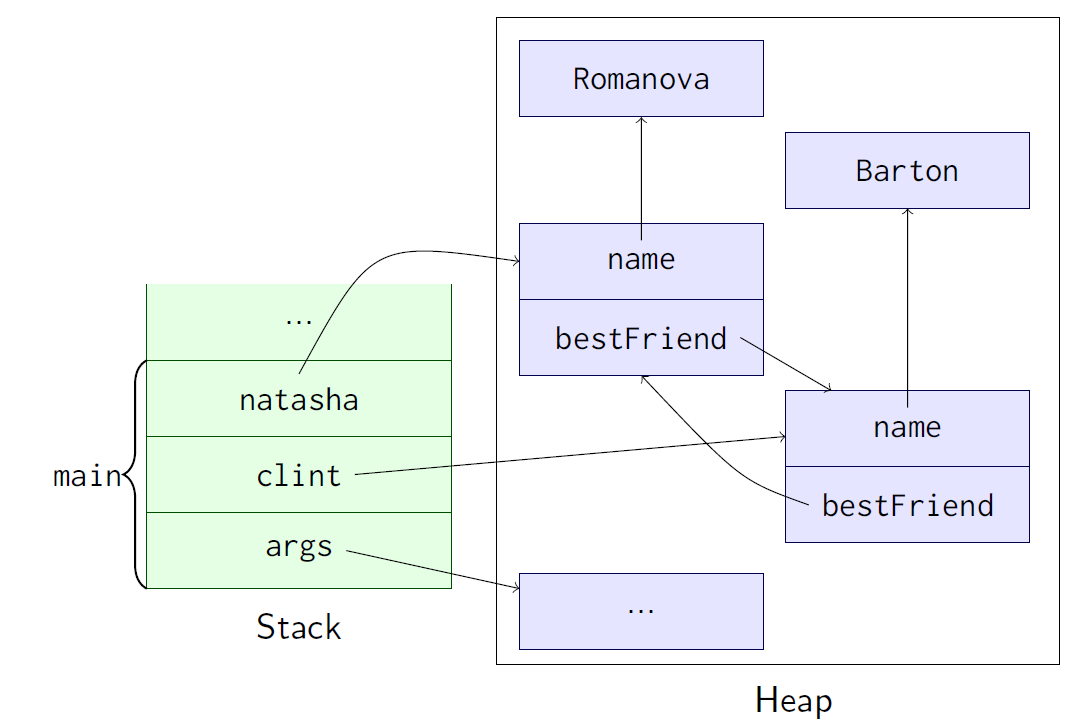
A már nem elérhető objektumokat a szemétgyűjtő felszabadítja.

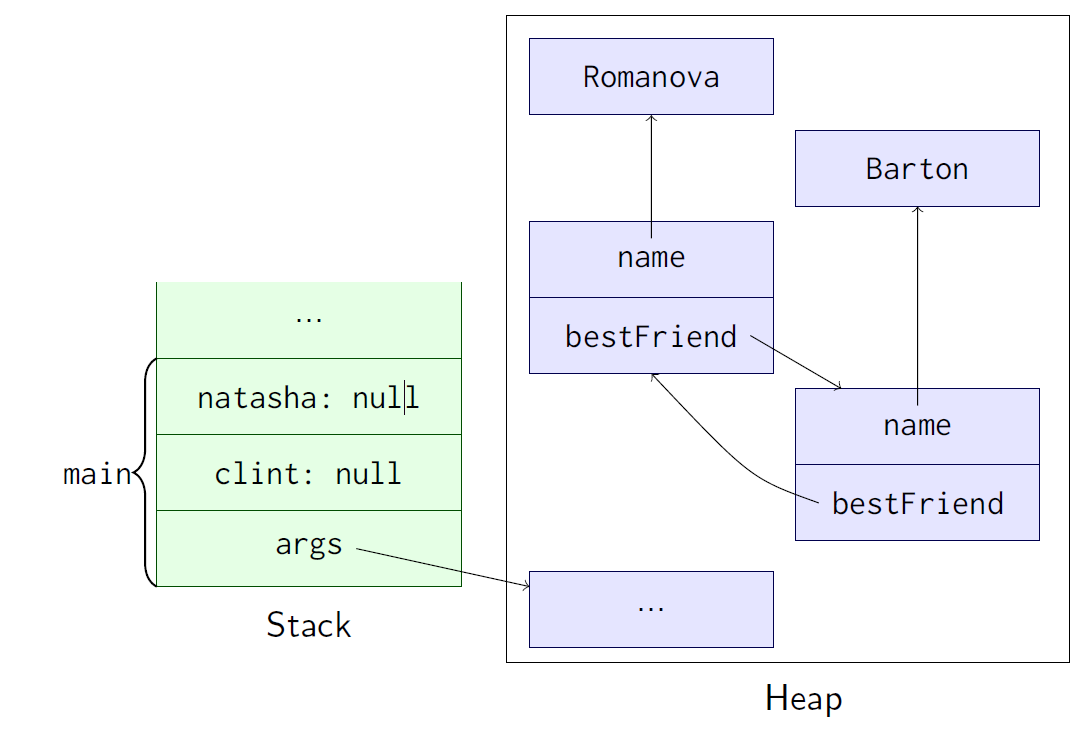
 



Minden hivatkozás megszűnt az objektum felszabadítható.

## Egy bonyolultabb példa





Ha egy objektumra csak olyan objektumok hivatkoznak, melyek szintén nem elérhetőek, az egész hivatkozási kört memóriaszemétnek nyilvánítja a GC algoritmus.

## Mark-and-Sweep szemétgyűjtés

Végighaladunk a vermen lévő referenciákon, megjelöljük az ezen referenciák által elért objektumokat majd a megjelöltekből elérhető objektumokat és így tovább, amíg nem tudunk újabb objektumot megjelölni. Az eljárás végén nem megjelölt objektumok felszabadíthatók.

# Az osztályszintűség

A static kulcsszóval megjelölt mezők és metódusok osztályszintűek, azaz közvetlenül az osztályhoz nem pedig az osztály egy objektumához tartoznak.

class Item {

static int counter = 0;

int id = counter++; // jelentése: id = Item.counter++

}

class Main {

public static void main( String[] args ){ //main is static

Item item1 = new Item(), item2 = new Item();

System.out.println( item1.id );

System.out.println( item2.id );

System.out.println( item1.counter ); // így is használhatjuk de

// az Item.counter az ajánlott

}}

(A kód lényegében azt csinálja, hogy minden osztályból példányosított objektumhoz egy azonosítót rendel az id osztályszintű adattag segítségével.)

# Tömbök

Olyan adatszerkezet, melyben az elemek egymás után találhatók memóriában, ami hatékony indexelést tesz lehetővé.

Java-ban a tömb egy objektum. A nyelv biztosít különböző műveleteket, illetve a tömb ismeri saját méretét (.length adattag). Futás közbe ellenőrizve van a megfelelő használat például túlindexelés esetén ArrayIndexOutOfBoundsException kivétel kerül eldobásra.

## Tömbök bejárása

for( int i = 0; i < args.length; ++i ){

System.out.println( args[i] );

}

Használhatunk iteráló ciklust is:

for( String s: args ){

System.out.println( s );

}

A tömb felpopulálására használható a következő szintaxis:

Point[] triangle = { new Point(0,0),new Point(0,3),new Point(1,1) };

A tömbökben tárolhatunk referenciákat is, melyek lehetnek további tömbök is.

## Sorfolytonos ábrázolás

### Többdimenziós tömb C-ben



Indexelés:

### Többdimenziós tömb Java-ban

A képen szöveg látható

Automatikusan generált leírás

Indexelés: , ahol az adott 8 bites cím alatt található objektumot adja meg.

Megjegyzés: azért adunk hozzá négyet mivel a tömbök egy int típusú változóban tárolják méretüket.

# Felsorolás

enum Day { SUN, MON, TUE, WED, THU, FRI, SAT }

Java-ban már teljesértékű típus, nem pedig csak elnevezett számok, mint a C-ben. A különböző felsorolt értékek objektumok.

Day best = Day.SAT; // itt lehetne akár import staticot használi és akkor írhatnánk annyit hogy SAT

best = 3; // fordítási hiba

int n = best; // fordítási hiba

int m = best.ordinal(); // 6

Típusértékek halmaza jól meghatározott, a deklarációt követően semmilyen módon nem tudunk újféle típusértékeket létrehozni.

(Nem klónozható, szerializálható, nem lehet belőle örökölni.)

enum Coin {

PENNY(1), NICKEL(5), DIME(10), QUARTER(25);

private final int centValue;

Coin(int centValue) { this.centValue = centValue; }

public int centValue() { return centValue; }

public int percentageOf( Coin that ) {

return 100 \* centValue / that.centValue();

}

}

Leginkább azért hasznos, mert azon típusok egyike melyek támogatják a switch utasítást.

static int workingHours( Day day ){

switch( day ){

case SUN:

case SAT: return 0;

case FRI: return 6;

default: return 8;

}

}