Java és web programozás

Kovács Kristóf, Rimay Zoé

Budapesti Műszaki Egyetem

2013. szeptember 25.

3. Előadás

User

```
public class User {
    private String realName_;
    private String nickName_;
    private String password_;

    public User(String realName, String nickName) {
        realName_ = realName;
        nickName_ = nickName;
    }

    public User(String realName,
            String nickName, String password) {
        realName_ = realName;
        nickName_ = nickName;
        password_ = password;
    }
```

```
    public boolean passwordCheck(String enteredPassword) {
        if (password_.equals(enteredPassword)) {
            return true;
        }
        return false;
    }

    public void setNickName(String newNickName) {
        nickName_ = newNickName;
    }
}
```

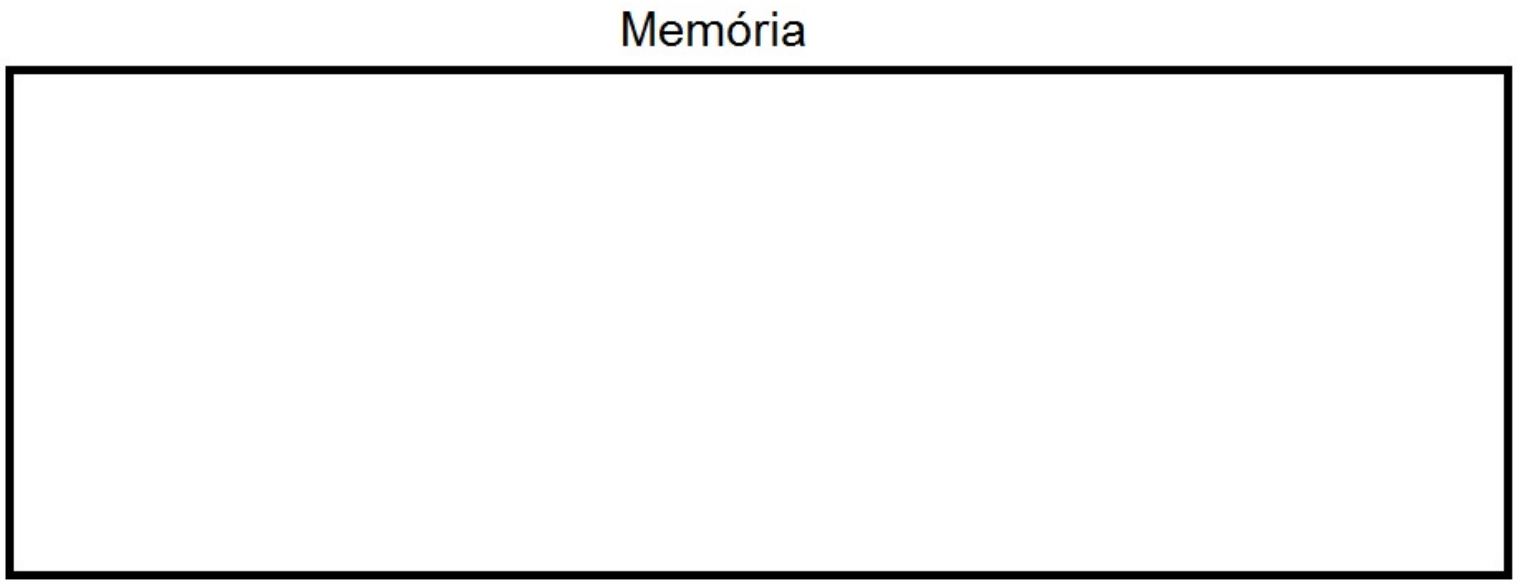
Memória kép

- Az előbbi *User* osztály is elhelyezkedik valahol a memóriában. Rá úgy kell tekinteni mint egy mintára amiből lehet készíteni *User* objektumokat.
- Amikor létrehozunk (*konstruálunk*) egy *User* objektumot akkor használnunk kell az egyik *konstruktorát*.
- A java először lefoglalja a memóriában a helyet az objektumnak. Majd lefuttatja a konstruktort amivel a objektum adattagjai kezdőértéket kapnak.

Memória kép

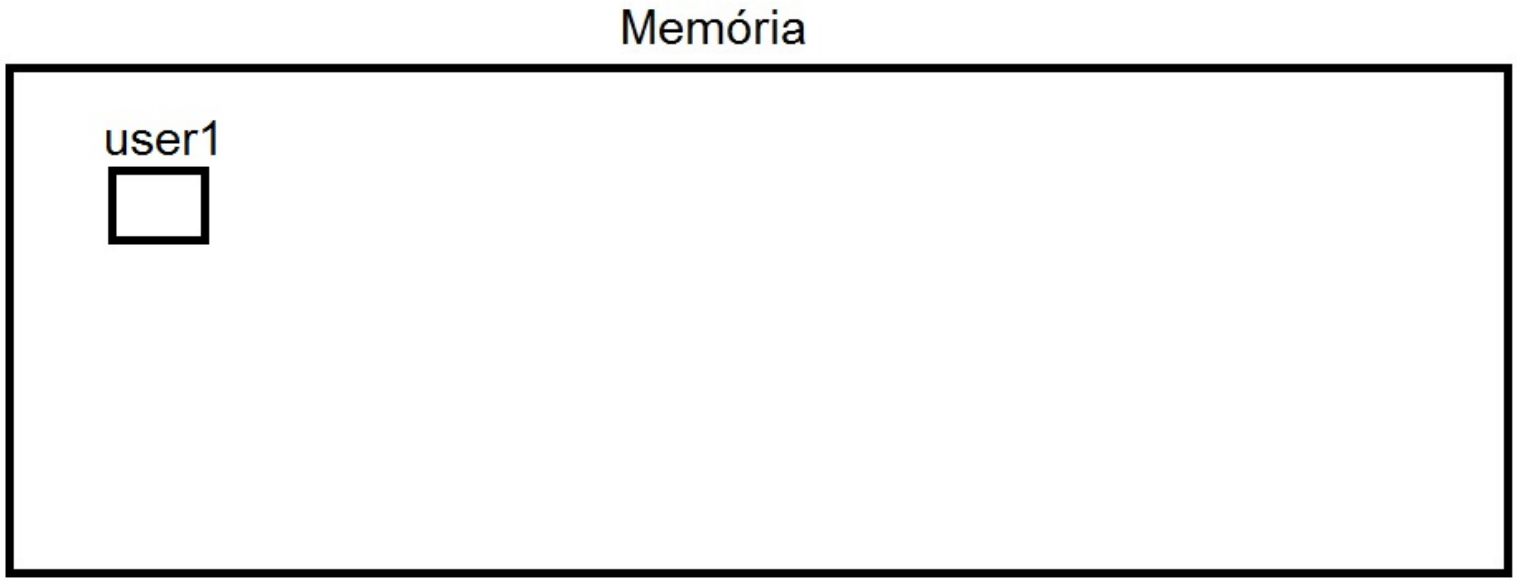
- Az előbbi *User* osztály is elhelyezkedik valahol a memóriában. Rá úgy kell tekinteni mint egy mintára amiből lehet készíteni *User* objektumokat.
- Amikor létrehozunk (*konstruálunk*) egy *User* objektumot akkor használnunk kell az egyik *konstruktorát*.
- A java először lefoglalja a memóriában a helyet az objektumnak. Majd lefuttatja a konstruktort amivel a objektum adattagjai kezdőértéket kapnak.
- Amikor csak deklarálunk egy nem primitív változót ahhoz nem tartozik semmilyen tárolt objektum egészen addig amíg nem foglalunk le egyet és adjuk neki értékül.
- Minden változóra úgy kellene tekinteni, mintha pointerek lennének.

Példa



- Ez a memóriánk.

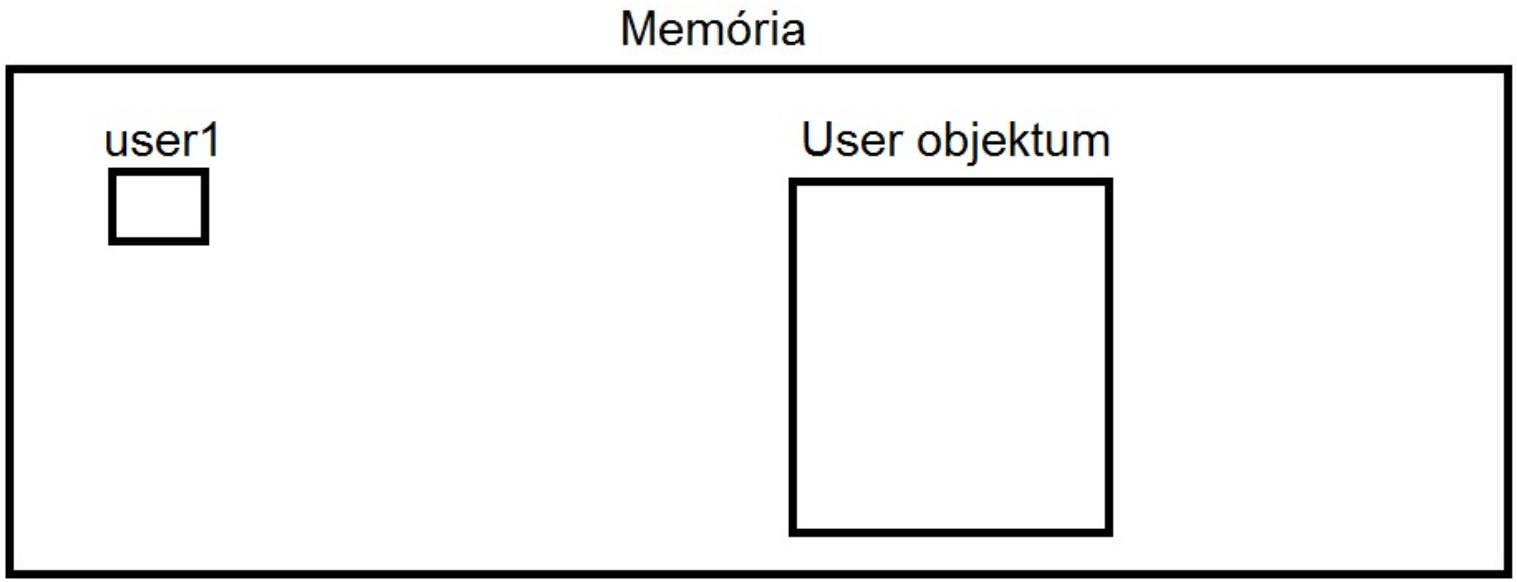
Példa




```
User user1;
```

- Létrehoztuk a user1 *változót*

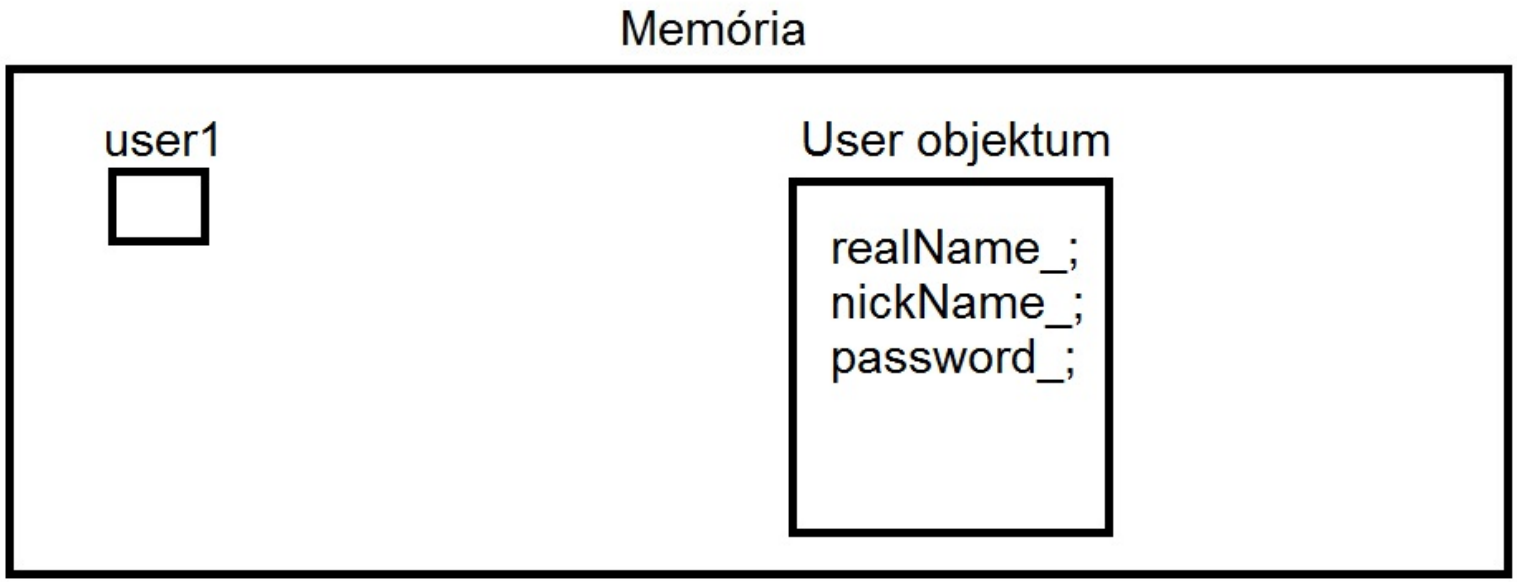
Példa




```
User user1;
new User("kutya", "morzsi");
```

- Létrehoztunk egy User *objektumot*

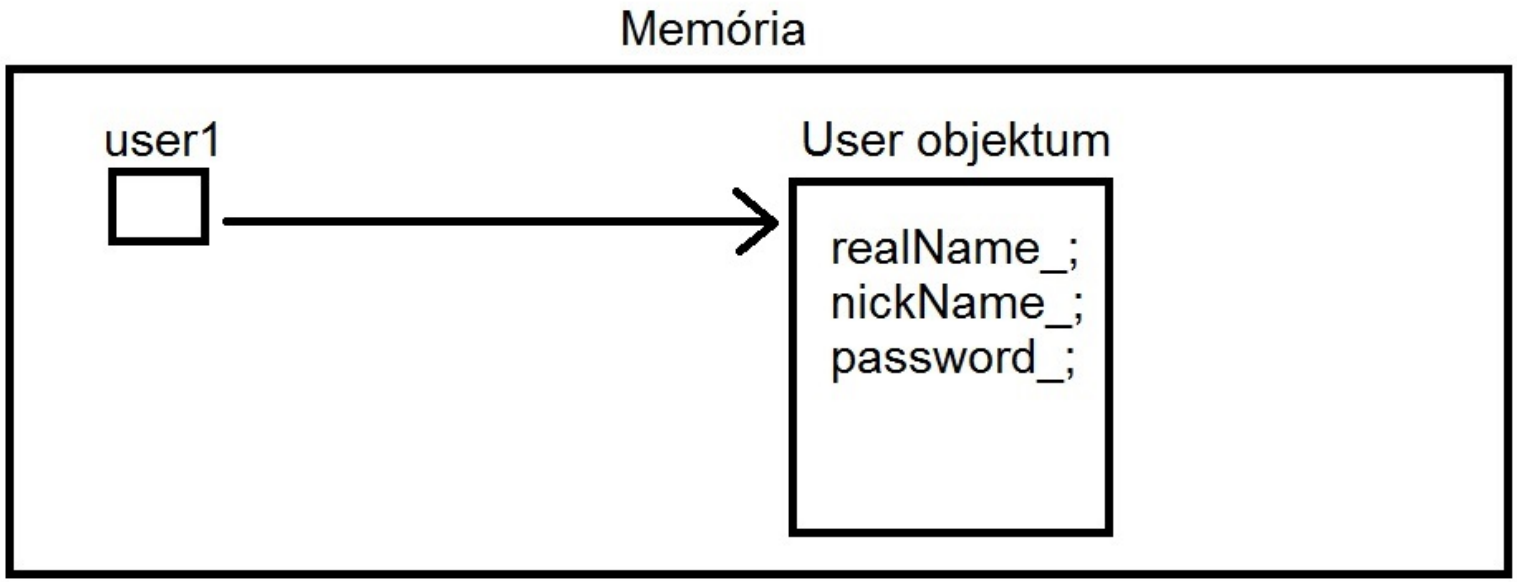
Példa




```
User user1;
new User("kutya", "morzsi");
```

- Ennek lefutott a *konstruktora* így az adattagjainak van értéke

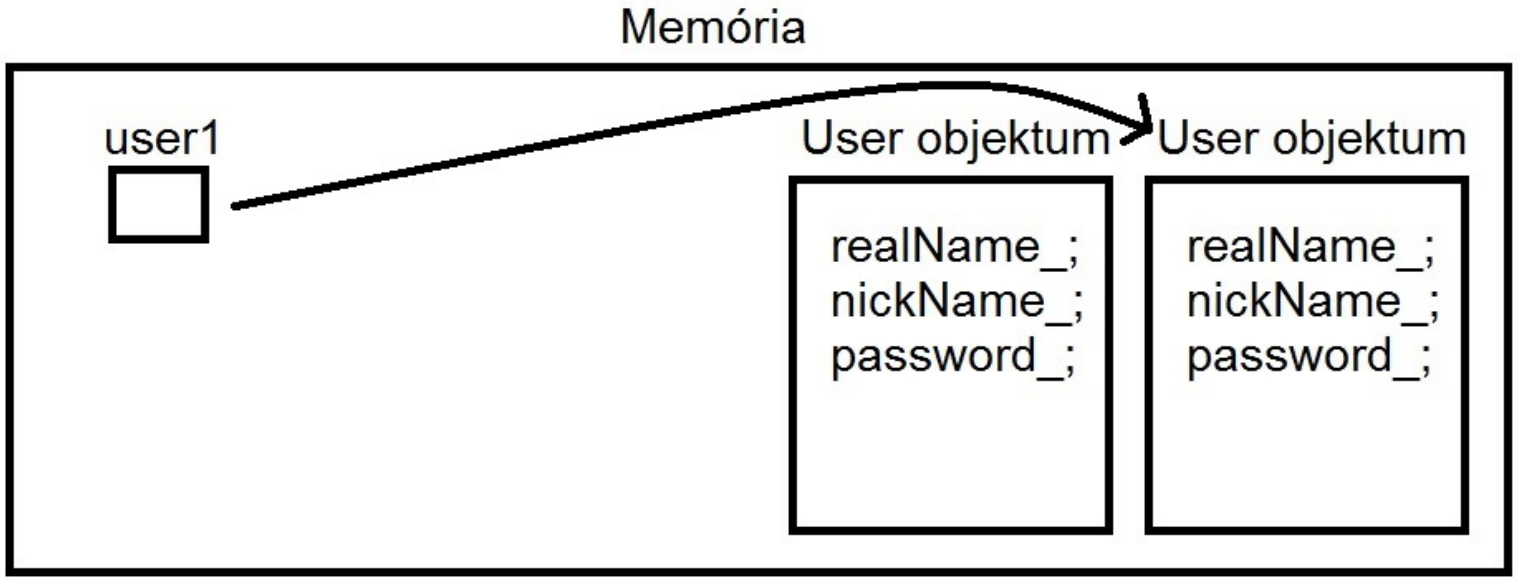
Példa




```
User user1;
user1 = new User("kutya", "morzsi");
```

- Hozzarendeltük a változóhoz az objektumot

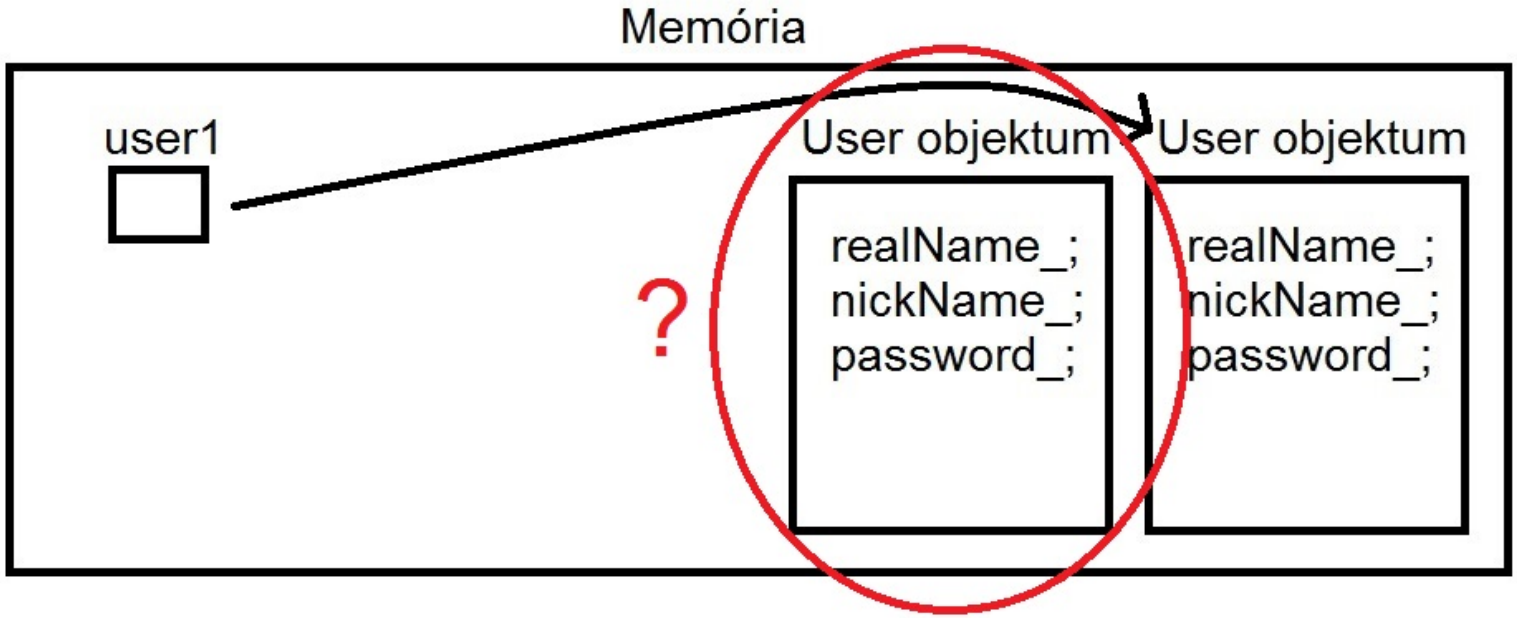
Példa




```
User user1;
user1 = new User("kutya", "morzsi");
user1 = new User("macska", "ficur");
```

- Létrehoztunk egy új objektumot és a *user1*-hez rendeltük

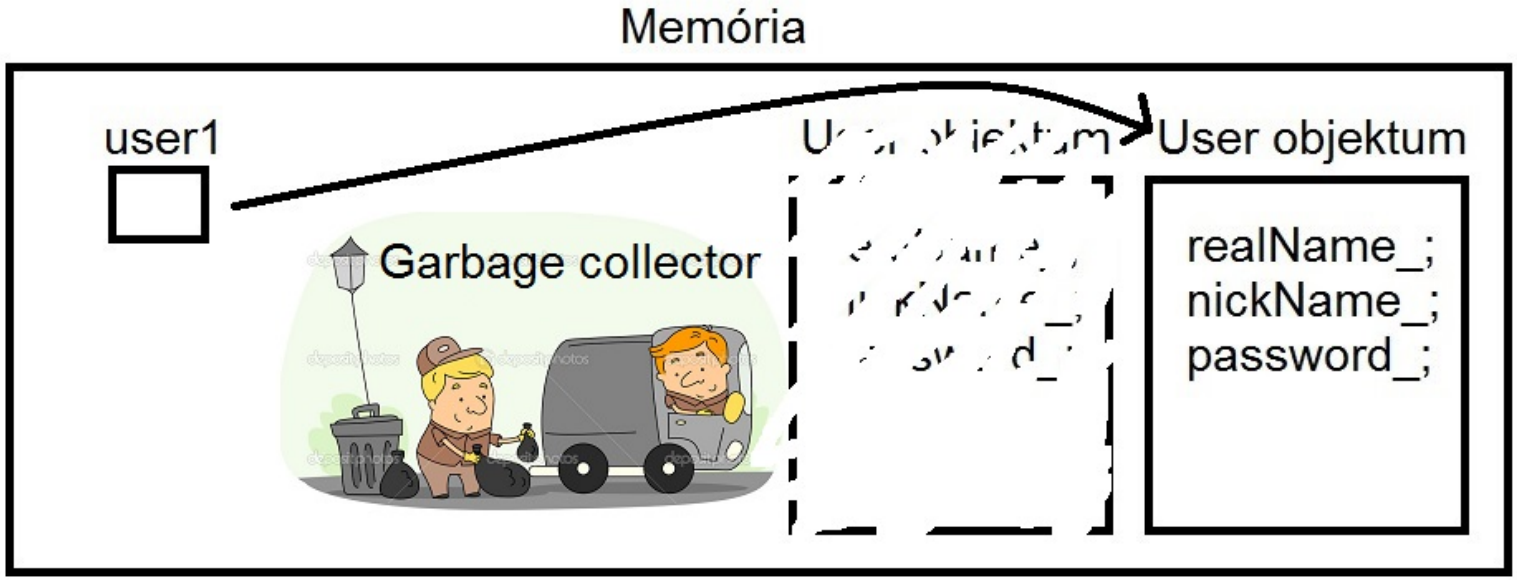
Példa




```
User user1;
user1 = new User("kutya", "morzsi");
user1 = new User("macska", "ficur");
```

- Mi lesz most az eredeti objektumunkkal?

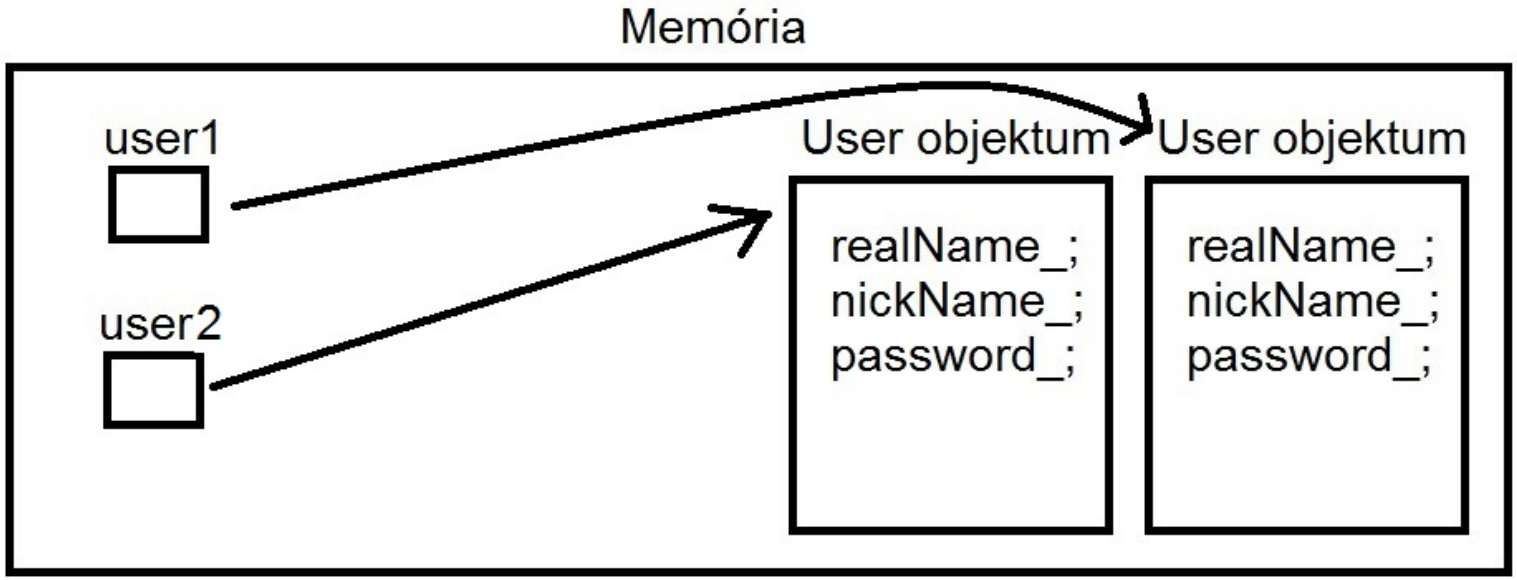
Példa




```
User user1;
user1 = new User("kutya", "morzsi");
user1 = new User("macska", "ficur");
```

- Valamikor jön majd a *garbage collector* és felszabadítja

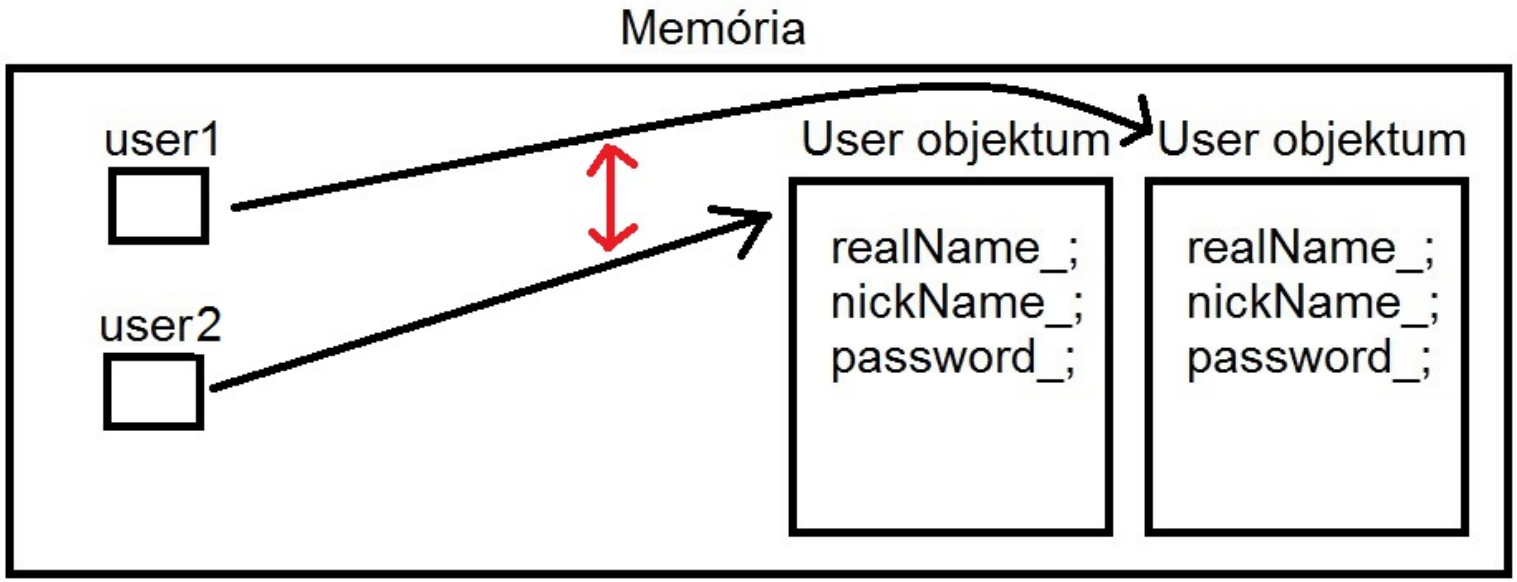
Példa




```
User user1;
User user2 = new User("kutya", "morzsi");
user1 = new User("macska", "ficur");
```

- Két objektumot össze szeretnénk hasonlítani

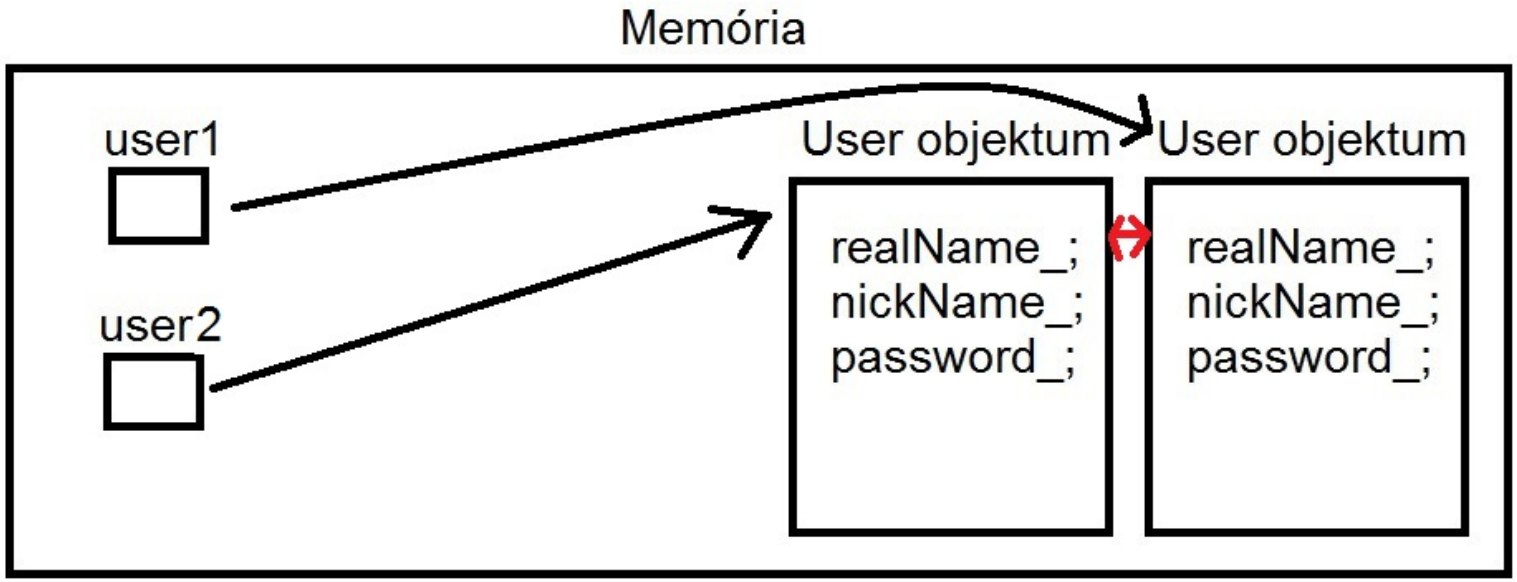
Példa




```
User user1;
User user2 = new User("kutya", "morzsi");
user1 = new User("macska", "ficur");
user1 == user2;
```

- Így a user változók hasonlítódnak össze, azaz az derül ki, hogy ugyanarra az objektumra mutatnak-e.

Példa




```
User user1;
User user2 = new User("kutya", "morzsi");
user1 = new User("macska", "ficur");
user1.equals(user2);
```

- Így már valóban az objektumok hasonlítódnak össze.

Garbage collector

- Mint majdnem minden programnyelvben itt is igaz az, hogy egy blokkban létrehozott változó csak a blokkon belül létezik.
- Az objektumokra ez nem igaz, a memóriában maradnak azután is, hogy vége egy blokknak.
- Viszont, ha beindult a garbage collector és talál olyan objektumot amire nincs hivatkozás. Azaz sehonnan nem lehet elérni (mint az előző példában). Akkor felszabadítja.

Garbage collector

- Mint majdnem minden programnyelvben itt is igaz az, hogy egy blokkban létrehozott változó csak a blokkon belül létezik.
- Az objektumokra ez nem igaz, a memóriában maradnak azután is, hogy vége egy blokknak.
- Viszont, ha beindult a garbage collector és talál olyan objektumot amire nincs hivatkozás. Azaz sehonnan nem lehet elérni (mint az előző példában). Akkor felszabadítja.
- A garbage collector nem folyamatosan működik, hanem bizonyos időpillanatokban elindul.
- Rá lehet kényszeríteni a javat, hogy indítsa el a garbage collectort.

Statikus adattagok / metódusok

- A statikus adattagok az osztályhoz tartoznak és nem az objektumokhoz.
- Ez azt is jelenti, hogy ha valami statikus már akkor létezik, amikor még egy objektumot se hoztunk létre az adott osztályból.

Statikus adattagok / metódusok

- A statikus adattagok az osztályhoz tartoznak és nem az objektumokhoz.
- Ez azt is jelenti, hogy ha valami statikus már akkor létezik, amikor még egy objektumot se hoztunk létre az adott osztályból.
- Ezért kell a *main* függvényeknek mindig statikusnak lenniük, mert őket anélkül kell tudnunk meghívni, hogy létrehoztunk volna objektumot.
- Statikus adattagból nem jön létre több ha létrehozunk egy új objektumot, hisz az osztályhoz tartozik. Így egy bizonyos statikus adattagból mindig csak egy van.

Példa statikus adattagra

```
public class Pelda {
    public static int szamlalo = 0;

    public Pelda() {
        Pelda.szamlalo++;
    }

    public static void main(String[] args) {
        Pelda pelda1 = new Pelda();
        Pelda pelda2 = new Pelda();
        Pelda pelda3 = new Pelda();
        Pelda pelda4 = new Pelda();
        System.out.println(Pelda.szamlalo); // 4
    }
}
```

Static

- Láthatjátok, hogy a statikus adattagokat nem az objektumon keresztül (*this*) érjük el, hanem az osztalyon keresztük.
- Statikus adattagok használat nem ajánlott nagyobb projectekben ahol esetleg megjelenik a párhuzamosítás.

Static

- Láthatjátok, hogy a statikus adattagokat nem az objektumon keresztül (*this*) érjük el, hanem az osztalyon keresztük.
- Statikus adattagok használat nem ajánlott nagyobb projectekben ahol esetleg megjelenik a párhuzamosítás.
- Amire viszont mindig hasznos, hogy konstansokat definiáljatok.
- public static final float pi = 3.14;

Konstansok

- A konstansok megváltoztathatatlanok, azonnal értéket kell nekik adni és ezután értekadás jobb oldalán szerepelhet csak.
- Bármilyen változót lehet konstansnak definiálni.
- Metódus is lehet konstans (*final*). Ekkor az adott metódus módosíthatatlan. Ilyenről jelenleg még nem tanultunk, de a későbbiekben elő fog jönni.

Összefoglalás

- Változó és objektum két külön életet él.
- Emiatt sokszor nem egyértelmű egy művelet eredménye. Érdemes mindig pointerekkel gondolkodni, ekkor átláthatóbb lesz.
- Statikus dolgok az osztályhoz tartoznak és nem az objektumhoz.
- Statikus függvények meghívhatók objektum létrehozása nélkül. Ez azt is jelenti, hogy statikus objektumokban a *this* kulcsszónak nincs értelme.
- Konstansokat definiálhattok a *final* kulcsszóval.