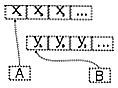
Mi a különbség a mély és a sekély másolat között?

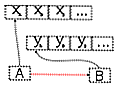
Cloning refers to create duplicate copies of objects in java. Cloning of objects can be very useful if we use the prototype pattern. The **clone( )** method generates a duplicate copy of the object on which it is called. Only classes that implement the *Cloneable* interface can be cloned.

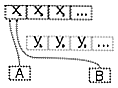
**java.lang.cloneable** : - Note that this interface does notcontain the clone method. The Cloneableinterface doesn't have any members. It is an empty interface, used only to indicate cloning is supported by a class. Java supports two type of cloning: - Shallow and Deep cloning. By default shallow copy is used in Java. Object class has a method clone() which does shallow cloning.

Ha van egy Customer objektum, amelynek "van" egy Address osztályú (objektum) adattagja, a Customer objektum "sekély másolása" azt jelenti, hogy a Address objektumhoz tartozó **mutató / hivatkozás** lesz átmásolva. Az eredeti és a másolat egyaránt ugyanarra az Address objektumra mutat, míg a mély másolat újat Address objektumot hoz létre, és arra fog mutatni.

**Sekély**:



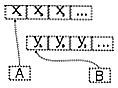


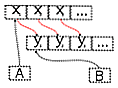


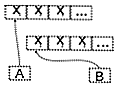
(B az A sekély másolata)

Az A és B változók a memória különböző területeire utalnak, ha B-t A-hoz rendeljük, a két változó ugyanarra a memóriaterületre vonatkozik. Bármelyik tartalmának későbbi módosításai azonnal megjelennek a másik tartalmában, mivel megosztják a tartalmat.

**Mély**:







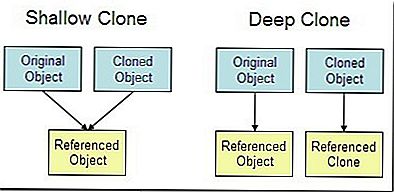
(B az A mély másolata)

Az A és B változók a memória különféle területeire utalnak. Amikor B az A-hoz rendeli az értékeket a memória azon területéről, amelyre A mutat, az értékek bemásolódnak arra a memóriaterületre, amelyre B mutat. A tartalom későbbi módosításai továbbra is egyedülállóak A vagy B vonatkozásában; a tartalom nincs megosztva.

en.wikipedia.org/wiki/Object\_copy#Shallow\_copy

Egy sekély példányban a B objektum az A objektum memóriájában lévő helyére mutat. Mély másolatban az A objektum memóriahelyének minden része átmásolódik a B objektum memóriahelyére.

Ez a wiki cikk remek ábrát mutat: http://en.wikipedia.org/wiki/Object\_copy



Sekély másolat esetén minden olyan objektumra, amelyre a forrás mutat, a cél is rámutat (így egyetlen hivatkozott objektum sem kerül másolásra).

Mély másolat esetén minden olyan objektum átmásolódik, amelyre a forrás obj. mutat, és a másolatra a rendeltetési hely mutat (tehát minden hivatkozott objektumból most 2 lesz).

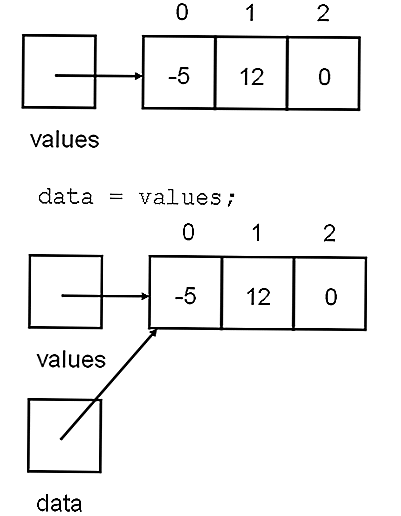
Csak a könnyebb megértés érdekében kövesse ezt a cikket: https://www.cs.utexas.edu/~scottm/cs307/handouts/deepCopying.htm

**Sekély másolat:**

int[] values = {-5, 12, 0};

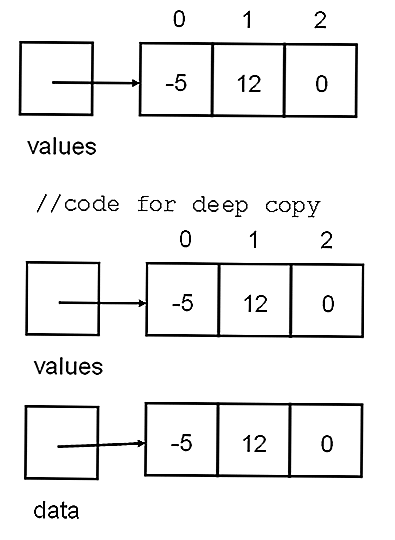
private int[] data;

data = values;



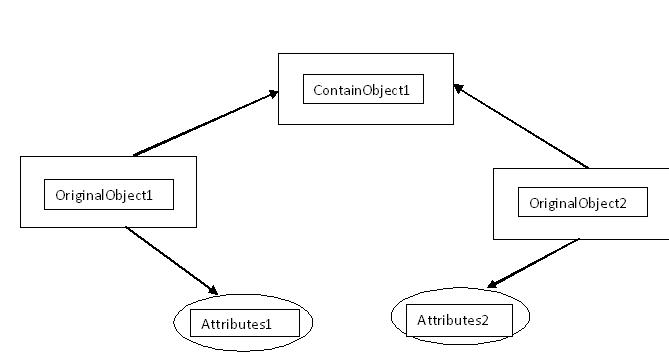
**Mély másolat:**

data = new int[values.length];  
for (int i = 0; i < data.length; i++) {  
            data[i] = values[i];  
        }



**Mi az a sekély másolás?**

A sekély másolat egy objektum bitenkénti másolata. Új objektum jön létre, amely az eredeti objektumban található értékek pontos másolatával rendelkezik. Ha az objektum bármelyik mezője hivatkozás más objektumokra, akkor csak a referencia címek, azaz csak a memóriacímek kerülnek másolásra.  
A sekély másolás az objektum legfelső szintű szerkezetét másolja, az alsó szinteket nem. Ez az összes attribútum pontos bitmásolata.



Ezen az ábrán a OriginalObject1 objektum az alábbi mezőkkel rendelkezik:

 Attributes1 🡪 int típusú, és

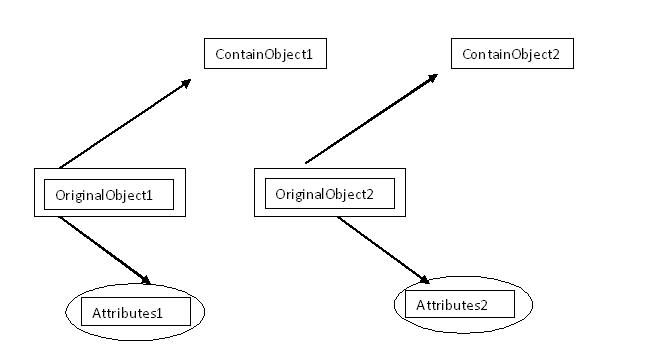
ContainObject1 🡪 ContainObject típusú

Az OriginalObject1  sekély másolása után az OriginalObject2 jön létre az Attributes2 -vel, amely tartalmazza az " Attributes1  másolt értékét, és továbbra is ugyanarra az objektumra mutat, azaz a OriginalObject1  -re.

Mivel az Attributes1  primitív adattípusú, az értékeit a rendszer az Attributes2 -be másolja, de a ContainedObject1 egy objektum, így az OriginalObject2  továbbra is a ContainObject1  -re mutat. Ha a ContainObject1  -en keresztül az OriginalObject1  -ben bármilyen módosítást hajtanak végre, az az OriginalObject2 -ben is megjelenik.

**Mi az a mély másolás?**

A mély másolat az összes mezőt lemásolja, és a mezők által hivatkozott, dinamikusan lefoglalt memóriacímekből másolatokat készít. A mélymásolat akkor következik be, amikor egy objektumot másolnak azokkal az objektumokkal, amelyekre mutat.  
A mélymásolás használatával az objektum a hivatkozott objektumokkal együtt másolásra kerül. A Deep clone az objektum összes szintjét felülről lefelé másolja rekurzívan.



Ezen az ábrán a OriginalObject1 objektum az alábbi mezőkkel rendelkezik:

 Attributes1 🡪 int típusú, és

ContainObject1 🡪 ContainObject típusú

Amikor az OriginalObject1  mélymásolatát használjuk, az OriginalObject2 Attributes2 -vel jön létre, amely tartalmazza az Attributes1  másolt értékét, és az OriginalObject2  tartalmazza a ContainObject1  másolt értékét. Ha a ContainObject1  -en keresztül az OriginalObject1  -ben bármilyen módosítást hajtanak végre, az nem fog megjelenni az OriginalObject2

Az objektumorientált programozásban a típus adattagok/tagmezők gyűjteményét tartalmazza. Ezeket a mezőket érték vagy referencia alapján (azaz egy érték mutatója) tárolhatjuk.

Egy sekély másolatban egy új típusú példány jön létre, és az értékeket átmásolja az új példányba. A referenciamutatókat ugyanúgy másoljuk, mint az értékeket. Ezért a hivatkozások az eredeti objektumokra mutatnak. A hivatkozott objektumokon történt változtatások mind az eredetiben, mind a másolatban megjelennek, mivel a hivatkozott objektumról nem készült másolat.

Mély másolatban az érték szerint tárolt mezők az eddigiekhez hasonlóan másolódnak, de a hivatkozásként tárolt objektumok mutatói nem másolódnak. Ehelyett mély másolatot készít a hivatkozott (al-)objektumokból, melyekre mutatókat tárol az új objektumban. A hivatkozott objektumokon végrehajtott bármilyen változás nem érinti az objektum más példányait.

A 'ShallowCopy' ugyanarra a helyre mutat a memóriában, mint a 'Source'. A „DeepCopy” más helyre mutat a memóriában, de a tartalma megegyezik.

* Ez kissé félrevezető. A sekély és a mély másolat is az objektumot egy új helyre másolja a memóriában, a mély pedig a gyermek objektumokat is, míg egy sekély csak az új tárgyakat utalja a régi gyerekekre. Nehéz elolvasni az eredeti objektumra való hivatkozás nélkül.

**Sekély klónozás:**  
Definíció: "Az objektum sekély példánya lemásolja az objektumot, de nem másolja a belső objektumokat." Ha egy egyéni objektumnak (pl. Employee) csak primitív, String típusú változói vannak, akkor a Sekély klónozást használja.

Employee e = new Employee(2, 'john cena'); Employee e2=e.clone();

Visszatérünk super.clone(); a felüldefiniált clone () metódussal, és kész.

**Mély klónozás**:  
Meghatározás: "A sekély másolattól eltérően a mély másolat egy objektum teljesen független másolata."  
Azt jelenti, hogy egy Employee objektum egy másik egyéni objektumot (Address) tartalmaz:

Employee e = new Employee(2, 'john cena', new Address(12, 'West Newbury', 'Massachusetts');

Ezután meg kell írni a kódot az 'Address' objektum klónozásához, valamint a felüldefiniált clone() metódushoz. Ellenkező esetben az 'Address' objektum nem klónozódik, és hibát okoz, ha megváltoztatuk a 'Address' értékét a klónozott Employee objektumban, amely az eredetit is tükrözi.

**Mély másolat**

A mély másolat az összes mezőt lemásolja, és a mezők által mutatott, dinamikusan lefoglalt memóriából másolatokat készít. A mély másolat akkor következik be, amikor egy objektumot másolnak azokkal az objektumokkal, amelyekre utal.

**Sekély másolat**

A sekély másolat egy objektum bitenként másolata. Új objektum jön létre, amely az eredeti objektumban található értékek pontos másolatával rendelkezik. Ha az objektum bármelyik mezője hivatkozás más objektumokra, akkor csak a referencia címek kerülnek másolásra, azaz csak a memória címek kerülnek másolásra.

**Sekély másolat**- Az eredeti és a sekélyen másolt objektumokon belüli referencia-változó **ugyanarra** tárgyra hivatkozik.

**Mély másolat**- Az eredeti és mélyen másolt objektumokon belüli referencia-változó **különböző** tárgyra hivatkozik.

A sekély példány új összetett objektumot készít, és beilleszti a hivatkozásokat az eredeti objektumra.

A sekély másolattól eltérően a mélymásolat új összetett objektumot készít, és az eredeti összetett objektum eredeti objektumainak másolatait is beszúrja.

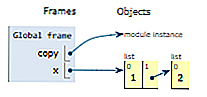
Vegyünk egy példát.

import copy x =[1,[2]] y=copy.copy(x) z= copy.deepcopy(x) print(y is z)

A fenti kód HAMIS.

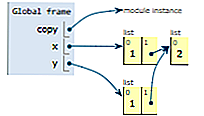
Lássuk, hogyan.

Eredeti összetett objektum x=[1,[2]] (összetettnek hívják, mert objektum van az objektumon belül (Kezdet))



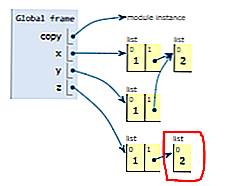
amint a képen látható, van egy lista a listán belül.

Ezután egy sekély másolatot készítünk a segítségével y = copy.copy(x). A Python itt azt teszi, hogy új összetett objektumot hoz létre, de a bennük lévő objektumok az eredeti objektumokra mutatnak.



A képen létrehozott egy új példányt a külső listához. de a belső lista ugyanaz marad, mint az eredeti.

Most elkészítjük annak mélymásolatát z = copy.deepcopy(x). amit a python itt csinál, új objektumot hoz létre a külső és a belső listához az alábbi képen látható módon (piros kiemelve).



A végén kódot nyomtat False, mivel y és z nem azonos objektumok.

HTH.

struct sample { char \* ptr; } void shallowcpy(sample & dest, sample & src) { dest.ptr=src.ptr; } void deepcpy(sample & dest, sample & src) { dest.ptr=malloc(strlen(src.ptr)+1); memcpy(dest.ptr,src.ptr); }

Egyszerű fogalmakban a sekély másolat hasonlít a referenciahíváshoz, a mély másolat pedig az érték szerinti híváshoz

A Call By Reference szolgáltatásban a függvény formális és tényleges paraméterei egyaránt azonos memóriahelyre és értékre utalnak.

Az Érték szerinti hívásnál a függvények mind formális, mind tényleges paraméterei eltérő memóriahelyre utalnak, de ugyanazzal az értékkel rendelkeznek.

Képzelje el, hogy két tömb van, az úgynevezett arr1 és arr2.

arr1 = arr2; //shallow copy arr1 = arr2.clone(); //deep copy

**Sekély** a másolás egy új objektum létrehozása, majd az aktuális objektum nem statikus mezõinek másolása az új objektumba. Ha egy mező értéktípus -> a mező bitenkénti másolata történik; a **referencia típus** -> a hivatkozás másolva van, de a hivatkozott objektum nem; ezért az eredeti objektum és klónja ugyanarra az objektumra utal.

**Mély** A copy új objektumot hoz létre, majd az aktuális objektum nem statikus mezőit másolja az új objektumba. Ha egy mező a **érték tipusa** -> a mező bitenkénti másolata kerül végrehajtásra. Ha egy mező a **referencia típus** -> a hivatkozott objektum új példánya végrehajtásra kerül. A klónozandó osztályokat [Serializable] néven kell megjelölni.

A [blog] -ból származik: http://sickprogrammersarea.blogspot.in/2014/03/technical-interview-questions-on-c\_6.html

**Mély másolat** magában foglalja egy objektum tartalmának felhasználását ugyanazon osztály másik példányának létrehozására. Mély másolatban a két objektum ugyanolyan információkat tartalmazhat, de a célobjektumnak saját pufferei és erőforrásai lesznek. egyik objektum megsemmisítése nem befolyásolja a megmaradt tárgyat. A túlterhelt hozzárendelési operátor az objektumok mély másolatát hozza létre.

**Sekély másolat** magában foglalja egy objektum tartalmának másolását ugyanazon osztály másik példányába, ezáltal tükörképet hozva létre. A referenciák és mutatók egyenes másolása miatt a két objektum meg fogja osztani a másik objektum ugyanazon külső tartalmát, hogy kiszámíthatatlan legyen.

**Magyarázat:**

Egy másoló szerkesztő segítségével egyszerűen tagonként másoljuk az adatértékeket. Ezt a másolási módszert sekély másolásnak nevezzük. Ha az objektum egyszerű osztály, beépített típusokból áll, és nincsenek mutatók, ez elfogadható lenne. Ez a függvény használná az értékeket és az objektumokat, és viselkedését nem változtatná meg egy sekély másolással, csak a tagok mutatóinak a címe kerül másolásra, és nem az az érték, amelyre a cím mutat. Az objektum adatértékeit ekkor akaratlanul megváltoztatja a függvény. Amikor a függvény kívül esik a hatókörön, az objektum másolata az összes adattal együtt kipattan a veremből.

Ha az objektumnak vannak mutatói, mély másolatot kell végrehajtani. Egy objektum mély másolatával memóriát osztanak ki a szabad raktárban lévő objektumhoz, és a másolt elemeket másolják. A mély másolatot olyan objektumokhoz használják, amelyeket egy függvényből adnak vissza.

Ha további válaszokat szeretne hozzáadni,

* Az objektum sekély másolata érték szerint másol az értéktípus-alapú tulajdonságoknál, a referenciák alapján pedig másol a referencia-típusokon alapuló tulajdonságoknál.
* Az objektum mély másolata érték alapján másol az értéktípusok alapú tulajdonságaihoz, valamint érték szerint másol a referencia típuson alapuló tulajdonságokhoz (a referencia típusok mélyén).

A következő sorokból értettem meg.

A Sekély másolás objektumot másol **érték tipusa**(int, float, bool) mezőket a célobjektumhoz és az objektum referencia típusait (karakterlánc, osztály stb.) **hivatkozások** a célobjektumban. Ebben a célban a hivatkozási típusok a forrásobjektum memóriahelyére mutatnak.

A mély másolás az objektum értékét és referencia típusait a célobjektumok teljes új példányába másolja. Ez azt jelenti, hogy mind az értéktípusok, mind a referencia típusok új memóriahelyeket kapnak.

A sekély másolat nem hoz létre új referenciát, de a mély másolat létrehozza az új hivatkozást.

Itt van a program, amely elmagyarázza a mély és sekély példányt.

public class DeepAndShollowCopy { int id; String name; List testlist = new ArrayList<>(); /\* // To performing Shallow Copy // Note: Here we are not creating any references. public DeepAndShollowCopy(int id, String name, Listtestlist) { System.out.println('Shallow Copy for Object initialization'); this.id = id; this.name = name; this.testlist = testlist; } \*/ // To performing Deep Copy // Note: Here we are creating one references( Al arraylist object ). public DeepAndShollowCopy(int id, String name, List testlist) { System.out.println('Deep Copy for Object initialization'); this.id = id; this.name = name; String item; List Al = new ArrayList<>(); Iterator itr = testlist.iterator(); while (itr.hasNext()) { item = itr.next(); Al.add(item); } this.testlist = Al; } public static void main(String[] args) { List list = new ArrayList<>(); list.add('Java'); list.add('Oracle'); list.add('C++'); DeepAndShollowCopy copy=new DeepAndShollowCopy(10,'Testing', list); System.out.println(copy.toString()); } @Override public String toString() { return 'DeepAndShollowCopy [id=' + id + ', name=' + name + ', testlist=' + testlist + ']'; } }

Ararys másolása:

A tömb egy osztály, ami azt jelenti, hogy referencia típusú, így az array1 = tömb2 két változót eredményez, amelyek ugyanarra a tömbre hivatkoznak.

De nézze meg ezt a példát:

static void Main() { int[] arr1 = new int[] { 1, 2, 3, 4, 5 }; int[] arr2 = new int[] { 6, 7, 8, 9, 0 }; Console.WriteLine(arr1[2] + ' ' + arr2[2]); arr2 = arr1; Console.WriteLine(arr1[2] + ' ' + arr2[2]); arr2 = (int[])arr1.Clone(); arr1[2] = 12; Console.WriteLine(arr1[2] + ' ' + arr2[2]); }

**sekély klón** azt jelenti, hogy csak a klónozott tömb által képviselt memória kerül másolásra.

**Ha a tömb érték típusú objektumokat tartalmaz, akkor az értékeket átmásolja**;

**ha a tömb referencia típust tartalmaz, akkor csak a hivatkozásokat másoljuk - így két tömb van, amelynek tagjai ugyanazokra az objektumokra hivatkoznak**.

Mély másolat létrehozásához - ahol a referencia típusa megismétlődik - végig kell lépnie a tömbön, és manuálisan klónozni kell az egyes elemeket.

* Nem ismerek más nyelveket, de a C # / VB-ben a sekély másolás egy értéktípus-tömböt igen **nem** másolja az értékeket. A két tömb ugyanazokra az objektumokra vonatkozik. Adjon hozzá egy gombot az űrlaphoz, és adja hozzá ezt a kódot a következők megtekintéséhez: private void button1\_Click(object sender, EventArgs e) { int[] arr1 = new int[]{1,2,3,4,5}; int[] arr2 = new int[]{6,7,8,9,0}; MessageBox.Show(arr1[2] + ' ' + arr2[2]); arr2 = arr1; MessageBox.Show(arr1[2] + ' ' + arr2[2]); arr1[2] = 12; MessageBox.Show(arr1[2] + ' ' + arr2[2]); }
* igazad van, helyesbítettem a válaszomat, a tömbökön lévő klón használatával. Abban teljesen igazad van, hogy "az értéktípusok tömbjének sekély másolása nem másolja az értékeket", de a tömbön a klón használata igen. Megpróbáltam ezt elmagyarázni, próbáld ki. Köszönöm

Az összes fenti definícióhoz hozzáadva még egy és leggyakrabban használt mély másolat található az osztály másolás-készítőjében (vagy túlterhelő hozzárendelési opátorában).

Sekély másolás -> az, amikor nem másolatot készít. Itt csak az objektum kerül másolásra, de az osztály összes tagja nem kerül másolásra.

Mély másolás -> akkor történik, amikor úgy döntött, hogy a másolás-készítőt vagy túlterhelést hozza létre az osztályában, és lehetővé teszi az osztály összes tagjának másolását.

MyClass& MyClass(const MyClass& obj) // copy constructor for MyClass { // write your code, to copy all the members and return the new object } MyClass& operator=(const MyClass& obj) // overloading assignment operator, { // write your code, to copy all the members and return the new object }

A másoláskonstruktort az új objektum inicializálására használják az ugyanabban az osztályban korábban létrehozott objektummal. Alapértelmezés szerint a fordító sekély példányt írt. A sekély másolás jól működik, ha nincs dinamikus memória-allokáció, mert ha dinamikus memória-allokáció van, akkor mindkét objektum ugyanazon memóriahely felé mutat egy kupacban, ezért ennek a problémának a eltávolítása érdekében mély másolatot írtunk, így mindkét objektumnak megvan a saját saját attribútum-másolata egy emlékezetben. A részletek teljes példákkal és magyarázatokkal történő elolvasása érdekében megtekintheti a C ++ konstruktorok cikket.