What is the difference between a deep copy and a shallow copy?

Asked 16 years, 2 months ago Modified 1 year, 1 month ago

Viewed 866k times

753

This question's answers are a community effort. Edit existing answers to improve this post. It is not currently accepting new answers or interactions.

What is the difference between a deep copy and a shallow copy?

language-agnostic copy deep-copy shallow-copy

Share Follow

edited Feb 20, 2014 at 22:45

Dariusz Woźniak

10.3k • 7 • 63 • 80

asked Oct 8, 2008 at 20:22

David Locke

18.1k • 9 • 34 • 53

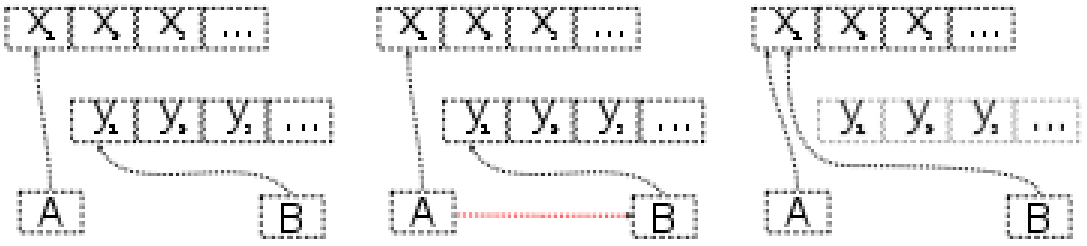
Comments disabled on deleted / locked posts / reviews

31 Answers Sorted by: Highest score (default)

1064

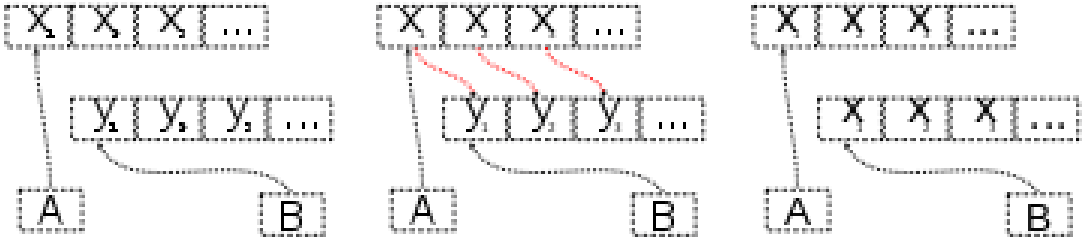
Breadth vs Depth; think in terms of a tree of references with your object as the root node.

Shallow:



The variables A and B refer to different areas of memory, when B is assigned to A the two variables refer to the same area of memory. Later modifications to the contents of either are instantly reflected in the contents of other, as they share contents.

Deep:



The variables A and B refer to different areas of memory, when B is assigned to A the values in the memory area which A points to are copied into the memory area to which B points. Later modifications to the contents of

either remain unique to A or B; the contents are not shared.

Share Improve this answer

Follow

edited Sep 10, 2020 at 17:12

thyu

1,763 ● 2 ● 16 ● 32

answered Oct 8, 2008 at 20:39

dlamblin

45.3k ● 22 ● 104 ● 144

45 Here's the wikipedia article that this illustration comes from in case it doesn't make sense out of context for you en.wikipedia.org/wiki/Object_copy#Shallow_copy – corbin Nov 7, 2013 at 1:59

4 In case of shallow copy if we make any changes in array B will that be reflected in array A since A & B both point to same memory location ? – tek3 Jan 16, 2015 at 12:42

6 In single line its copy by reference vs copy by value. Not sure if the answer is correct! – mannuscript Jun 5, 2017 at 11:04

20 @jasonleonhard So 9 years ago I just put urls to the images because embedding images wasn't supported. Thus the URL cited its source. The community later made the URLs into embedded images without editing some kind of citation onto it. The 4 year old top comment also points out what you point out. Have a look: stackoverflow.com/posts/184780/revisions Why not just edit a citation into the answer yourself? I might be unavailable the next time someone has some complaint about my 10 year old writing style. – dlamblin Sep 7, 2017 at 9:08

8 "For variables A and B, when B is assigned to A" doesn't this mean "A = B" in code? I'm somehow confused, since the

image reflects "B = A". – Peter Guan Sep 19, 2018 at 12:20

907

Shallow copies duplicate as little as possible. A shallow copy of a collection is a copy of the collection structure, not the elements. With a shallow copy, two collections now share the individual elements.

Deep copies duplicate everything. A deep copy of a collection is two collections with all of the elements in the original collection duplicated.

Share Improve this answer

Follow

answered Oct 8, 2008 at 20:29

S.Lott

391k ● 82 ● 517 ● 788

May be .NET MemberwiseClone() implementation do more than shallow copying in the conventional sense – Ilya Serbis Oct 18, 2012 at 23:29

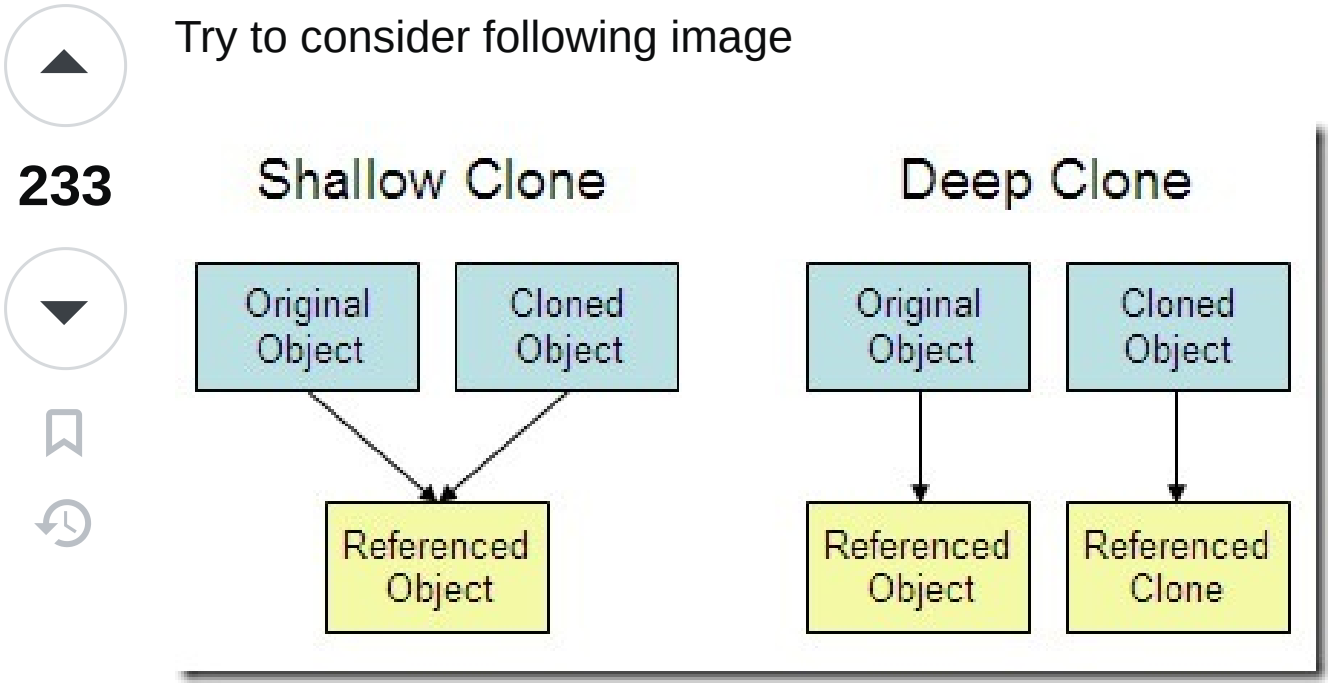
11 Keep in mind there are also **mixed copies** (not only such as lazy copy), which duplicates just part of it (here's an instance)! ;) – cregox Jan 10, 2015 at 3:59

3 what is a collection structure? – mfaani Apr 11, 2016 at 21:01

1 @Honey Collections can be diverse data structures which stores multiple data items. In python we have tuple,list,dictionery,etc – Murphy Aug 9, 2016 at 3:58

7 @RoyiNamir You probably already figured this out during the last 7 years, but for anybody else wondering about this: "*shallow copy copies the value type bit by bit*" is correct, but it's a bit confusing. If you have a `Customer` object which "has" an `Address` object, copying the `Customer` object

"bit by bit" means that the **pointer/reference** to the `Address` object is copied. Original and copy both point to the same `Address` object, whereas a deep copy will create a new `Address` object and point to that instead.
– Raphael Schmitz Feb 21, 2019 at 10:36



233

Try to consider following image

For example **Object.MemberwiseClone** creates a **shallow** copy link

and using **ICloneable** interface you can get **deep** copy as described here

Share Improve this answer

Follow

edited May 23, 2017 at 12:34

Community Bot
1 ● 1

answered Jun 15, 2016 at 8:42

Oleksandr
5,658 ● 4 ● 42 ● 72

65 A picture is worth a thousand words. – Levi Fuller Jun 26, 2016 at 3:35

17 Oh boy, came here to find out the meaning. This is the only answer which helped. – Karan Singh Jun 15, 2017 at 21:01

4 This is the simplest and yet only shows what's necessary. – hina10531 Mar 19, 2018 at 7:33

2 This answer explains copy by reference vs copy by value. Shallow copy vs deep copy is a concept that applies to collections. See this answer and this answer. – chetan Jun 11, 2020 at 12:17

The concept applies to much more than collections – nardnob Dec 6, 2020 at 15:45

183

In short, it depends on what points to what. In a shallow copy, object B points to object A's location in memory. In deep copy, all things in object A's memory location get copied to object B's memory location.

The Wikipedia article Object copy has a great diagram.

Share Improve this answer

Follow

edited Aug 28, 2023 at 19:18

Jan Schultke

38.3k ● 8 ● 87 ● 168

answered Oct 8, 2008 at 20:24

helloandre

10.7k ● 9 ● 49 ● 64

Especially For iOS Developers:

72

If `B` is a **shallow copy** of `A`, then for primitive data it's like `B = [A assign];` and for objects it's like `B = [A retain];`

B and A point to the same memory location

If `B` is a **deep copy** of `A`, then it is like `B = [A copy];`

B and A point to different memory locations

B memory address is same as A's

B has same contents as A's

Share Improve this answer

Follow

edited Apr 11, 2016 at 22:03

mfaani

36.1k ● 19 ● 189 ● 313

answered Jan 23, 2013 at 11:43

Abhishek Bedi

5,421 ● 2 ● 38 ● 62

9 "B memory address is same as A's" - How come ?
– user3693546 Oct 26, 2015 at 9:06

4 In Deep Copy, "B memory address is NOT same as A's"
– ismail baig May 8, 2018 at 4:14

69

Shallow copy: Copies the member values from one object into another.

Deep Copy: Copies the member values from one object into another.

Any pointer objects are duplicated and Deep Copied.

Example:

```
class String
{
     int   size;
     char* data;
};

String  s1("Ace");   // s1.size = 3
s1.data=0x0000F000

String  s2 = shallowCopy(s1);
 // s2.size =3 s2.data = 0X0000F000
String  s3 = deepCopy(s1);
 // s3.size =3 s3.data = 0x0000F00F
 //                      (With Ace copied to this
location.)
```

Share Improve this answer Follow

edited Oct 8, 2008 at 20:32

answered Oct 8, 2008 at 20:25

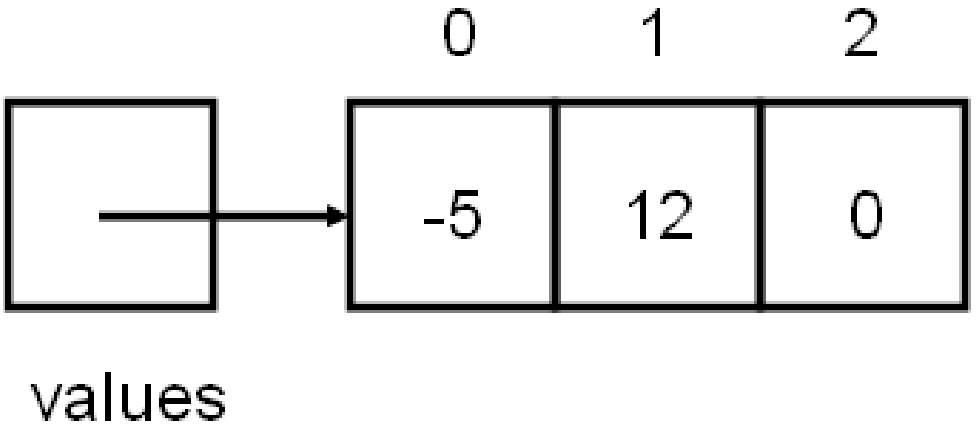
Loki Astari

264k ● 86 ● 342 ● 571

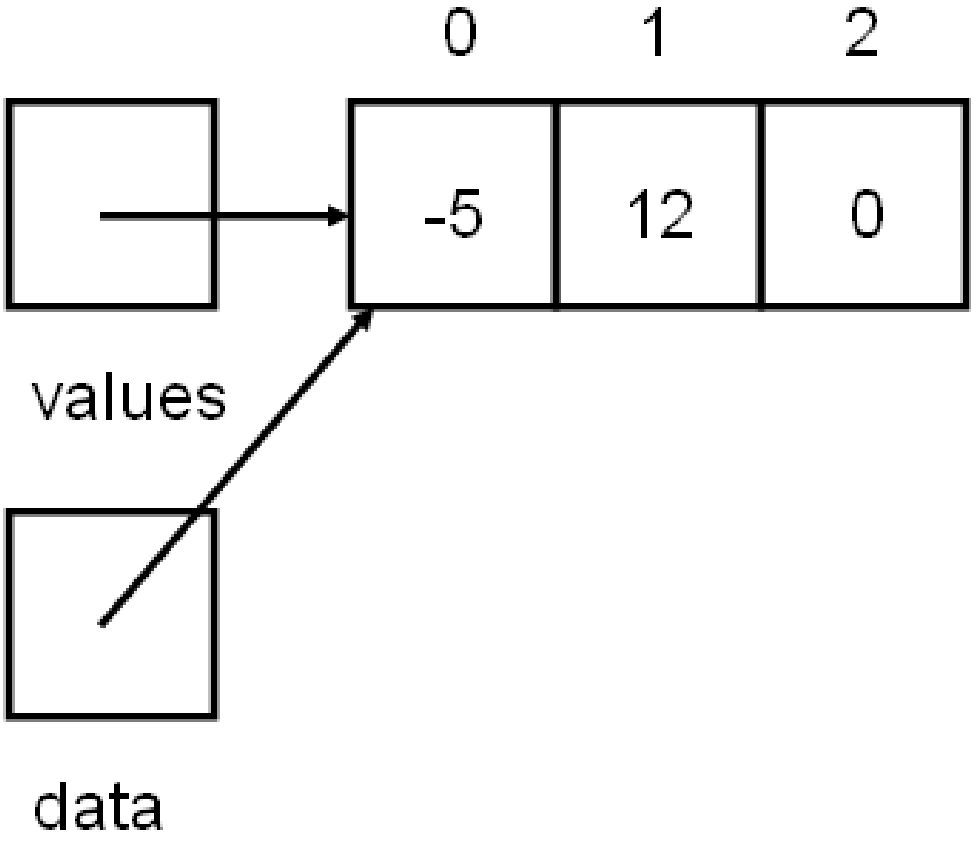
64

Just for the sake of easy understanding you could follow this article:
https://www.cs.utexas.edu/~scottm/cs307/handouts/deepCopying.htm

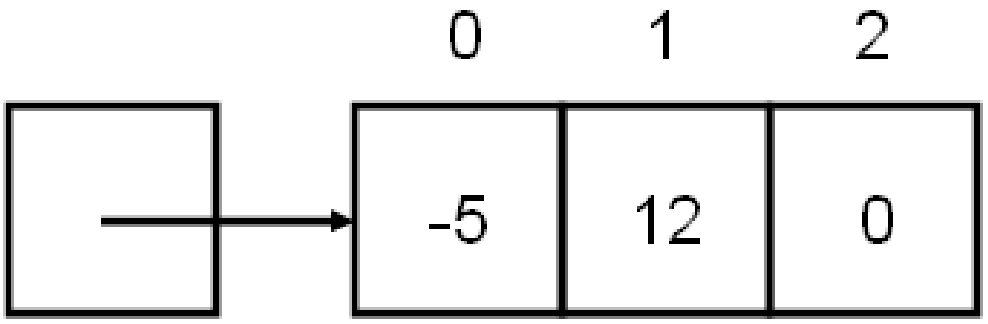
Shallow Copy:




```
data = values;
```

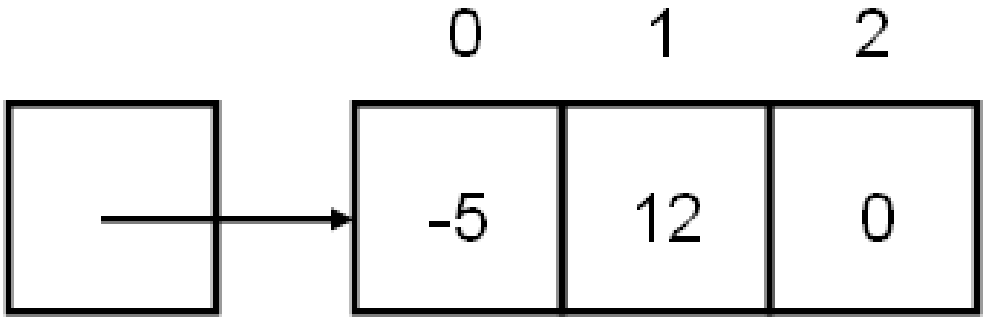


Deep Copy:

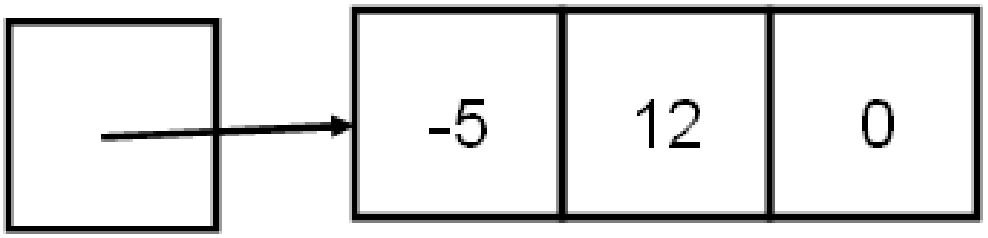


values

```
//code for deep copy
```



values



data

Share Improve this answer

Follow

edited Feb 14, 2016 at 0:40

answered Mar 19, 2014 at 6:15

Touchstone

5,962 ● 7 ● 42 ● 49

52

I haven't seen a short, easy to understand answer here--so I'll give it a try.

With a shallow copy, any object pointed to by the source is also pointed to by the destination (so that no referenced objects are copied).

With a deep copy, any object pointed to by the source is copied and the copy is pointed to by the destination (so there will now be 2 of each referenced object). This recurses down the object tree.

Share Improve this answer Follow

edited Apr 5, 2017 at 16:50

answered Oct 8, 2008 at 20:36

Bill K

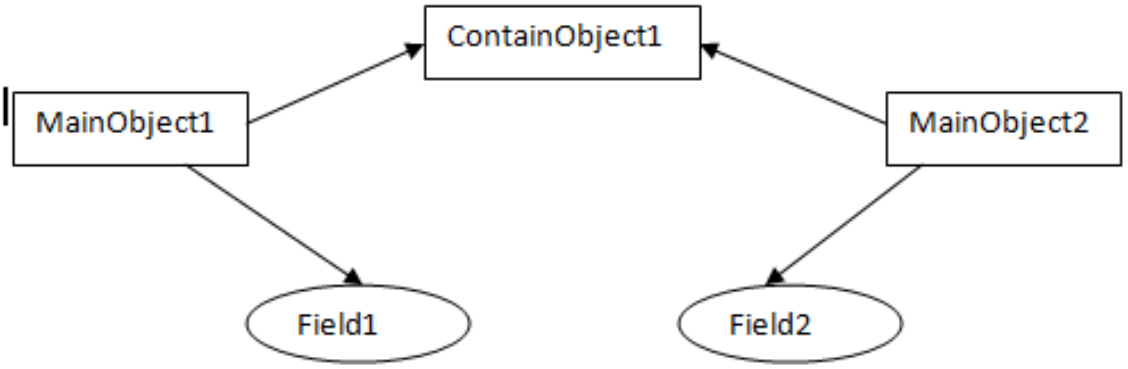
62.8k ● 18 ● 112 ● 158

38

What is Shallow Copy?

Shallow copy is a bit-wise copy of an object. A new object is created that has an exact copy of the values in the original object. If any of the fields of the object are references to other objects, only the reference addresses

are copied i.e., only the memory address is copied.



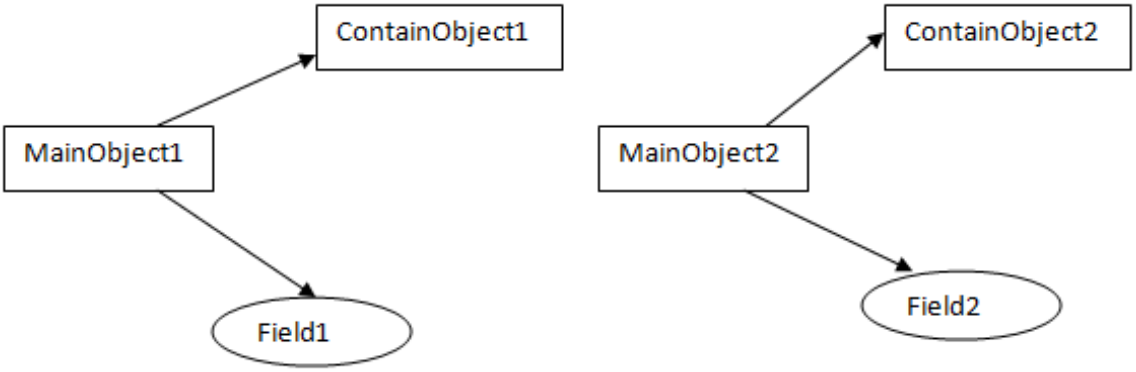
In this figure, the `MainObject1` has fields `field1` of type int, and `ContainObject1` of type `ContainObject`. When you do a shallow copy of `MainObject1`, `MainObject2` is created with `field2` containing the copied value of `field1` and still pointing to `ContainObject1` itself. Note that since `field1` is of primitive type, its value is copied to `field2` but since `ContainedObject1` is an object, `MainObject2` still points to `ContainObject1`. So any changes made to `ContainObject1` in `MainObject1` will be reflected in `MainObject2`.

Now if this is shallow copy, lets see what's deep copy?

What is Deep Copy?

A deep copy copies all fields, and makes copies of dynamically allocated memory pointed to by the fields. A deep copy occurs when an object is copied along with the

objects to which it refers.



In this figure, the MainObject1 have fields `field1` of type int, and `ContainObject1` of type `ContainObject`. When you do a deep copy of `MainObject1`, `MainObject2` is created with `field2` containing the copied value of `field1` and `ContainObject2` containing the copied value of `ContainObject1`. Note any changes made to `ContainObject1` in `MainObject1` will not reflect in `MainObject2`.

[good article]

Share Improve this answer

Follow

edited Nov 15, 2016 at 13:41

smci

33.9k ● 21 ● 117 ● 150

answered Dec 18, 2014 at 6:55

atish shimpi

5,013 ● 2 ● 34 ● 51

it's not your fault though this example refers to a `field3` which when in a position to try and comprehend something as deep as that issue, where is that #3 in that example taking place `ContainObject2` ? – Robb_2015 Dec 28, 2015 at 22:30

37

```
char * Source = "Hello, world.";

char * ShallowCopy = Source;

char * DeepCopy = new char(strlen(Source)+1);
strcpy(DeepCopy,Source);
```

'ShallowCopy' points to the same location in memory as 'Source' does. 'DeepCopy' points to a different location in memory, but the contents are the same.

Share Improve this answer

Follow

answered Oct 8, 2008 at 20:32

John Dibling

101k ● 32 ● 191 ● 331

37

{Imagine two objects: A and B of same type _t(with respect to C++) and you are thinking about shallow/deep copying A to B}

Shallow Copy: Simply makes a copy of the reference to A into B. Think about it as a copy of A's Address. So, the addresses of A and B will be the same i.e. they will be pointing to the same memory location i.e. data contents.

Deep copy: Simply makes a copy of all the members of A, allocates memory in a different location for B and then assigns the copied members to B to achieve deep copy. In this way, if A becomes non-existant B is still valid in the memory. The correct term to use would be cloning, where you know that they both are totally the same, but yet

different (i.e. stored as two different entities in the memory space). You can also provide your clone wrapper where you can decide via inclusion/exclusion list which properties to select during deep copy. This is quite a common practice when you create APIs.

You can choose to do a Shallow Copy **ONLY_IF** you understand the stakes involved. When you have enormous number of pointers to deal with in C++ or C, doing a shallow copy of an object is **REALLY** a bad idea.

EXAMPLE_OF_DEEP COPY_ An example is, when you are trying to do image processing and object recognition you need to mask "Irrelevant and Repetitive Motion" out of your processing areas. If you are using image pointers, then you might have the specification to save those mask images. NOW... if you do a shallow copy of the image, when the pointer references are KILLED from the stack, you lost the reference and its copy i.e. there will be a runtime error of access violation at some point. In this case, what you need is a deep copy of your image by CLONING it. In this way you can retrieve the masks in case you need them in the future.

EXAMPLE_OF_SHALLOW_COPY I really think it is not a good idea to do shallow copy if you know that your program is gonna run for an infinite period of time i.e. continuous "push-pop" operation over the stack with function calls. If you are demonstrating something to an amateur or novice person (e.g. C/C++ tutorial stuff) then it is probably okay. But if you are running an application

such as surveillance and detection system, or Sonar Tracking System, you are not supposed to keep shallow copying your objects around because it will kill your program sooner or later.

Share Improve this answer

Follow

edited Nov 10, 2023 at 7:05

Nimantha

6,438 ● 6 ● 30 ● 75

answered Jan 31, 2013 at 23:39

ha9u63a7

6,788 ● 18 ● 83 ● 121

17

In object oriented programming, a type includes a collection of member fields. These fields may be stored either by value or by reference (i.e., a pointer to a value).

In a shallow copy, a new instance of the type is created and the values are copied into the new instance. The reference pointers are also copied just like the values. Therefore, the references are pointing to the original objects. Any changes to the members that are stored by reference appear in both the original and the copy, since no copy was made of the referenced object.

In a deep copy, the fields that are stored by value are copied as before, but the pointers to objects stored by reference are not copied. Instead, a deep copy is made of the referenced object, and a pointer to the new object is stored. Any changes that are made to those referenced objects will not affect other copies of the object.

Share Improve this answer

Follow

answered Oct 8, 2008 at 20:57

Jeffrey L Whitledge

59.4k ● 9 ● 74 ● 100

17

Deep Copy

A deep copy copies all fields, and makes copies of dynamically allocated memory pointed to by the fields. A deep copy occurs when an object is copied along with the objects to which it refers.

Shallow Copy

Shallow copy is a bit-wise copy of an object. A new object is created that has an exact copy of the values in the original object. If any of the fields of the object are references to other objects, just the reference addresses are copied i.e., only the memory address is copied.

Share Improve this answer

Follow

edited May 4, 2019 at 10:18

Jean-François Fabre ♦

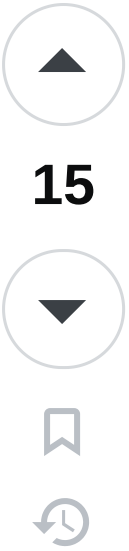
140k ● 24 ● 177 ● 241

answered Sep 16, 2014 at 9:50

Sunil Kumar Sahoo

53.6k ● 55 ● 181 ● 244

That link sadly no longer works - it now points to an article from February 2019 regarding web design (unless the author is clairvoyant?). – PhilPhil Apr 5, 2019 at 12:23



I would like to give example rather than the formal definition.

```
var originalObject = {
    a : 1,
    b : 2,
    c : 3,
};
```

This code shows a **shallow copy**:

```
var copyObject1 = originalObject;

console.log(copyObject1.a);         // it will
print 1
console.log(originalObject.a);       // it will
also print 1
copyObject1.a = 4;
console.log(copyObject1.a);           //now it
will print 4
console.log(originalObject.a);       // now it
will also print 4

var copyObject2 = Object.assign({},
originalObject);

console.log(copyObject2.a);        // it will
print 1
console.log(originalObject.a);      // it will
also print 1
copyObject2.a = 4;
console.log(copyObject2.a);        // now it will
print 4
console.log(originalObject.a);      // now it will
print 1
```

This code shows a **deep copy**:

```
var copyObject2 = Object.assign({},
originalObject);

console.log(copyObject2.a);        // it will
print 1
console.log(originalObject.a);      // it will
also print 1
copyObject2.a = 4;
console.log(copyObject2.a);        // now it will
print 4
console.log(originalObject.a);      // !! now it
will print 1 !!
```

Share Improve this answer

Follow

edited May 21, 2018 at 16:18

thirtydot

228k ● 49 ● 392 ● 353

answered Jan 24, 2018 at 6:52

Vivek Mehta

774 ● 8 ● 10

I am getting `1 1 4 4 4 4 4 4` – Suresh Prajapati Aug 11, 2019 at 20:25

in deep copy, do copyObject.a = 8 and then check. hope you will get proper answer. – Vivek Mehta Aug 12, 2019 at 7:02

object.assign({},arr) will not create the deep copy ,suppose we have the following object var source = {"foo":1,"name":"Testing",c:{age:34}} var dCopy = Object.assign({},source) console.log(dCopy.c.age) console.log(`Source deep ${source.c.age}`) source.c.age = 3 console.log(dCopy.c.age) console.log(`Source deep ${source.c.age}`) – Swarup Chavan Jul 29, 2020 at 5:14

13

'ShallowCopy' points to the same location in memory as 'Source' does. 'DeepCopy' points to a different location in memory, but the contents are the same.

Share Improve this answer

Follow

answered Feb 21, 2011 at 13:44

GovindaRaju

131 ● 1 ● 2

1 This is slightly misleading. Both a shallow and deep copy will copy the object to a new location in memory, a deep will also copy the child objects whereas a shallow will just have the new objects refer to the old children. It's difficult to read without referring to the original object. – Bill K Apr 5, 2017 at 17:04

13

Shallow Cloning:
Definition: "A shallow copy of an object copies the ‘main’ object, but doesn’t copy the inner objects." When a custom object (eg. Employee) has just primitive, String type variables then you use Shallow Cloning.

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

You return `super.clone();` in the overridden clone() method and your job is over.

Deep Cloning:
Definition: "Unlike the shallow copy, a deep copy is a fully

independent copy of an object."
Means when an Employee object holds another custom object:

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

Then you have to write the code to clone the 'Address' object as well in the overridden clone() method. Otherwise the Address object won't clone and it causes a bug when you change value of Address in cloned Employee object, which reflects the original one too.

Share Improve this answer

Follow

edited May 11, 2018 at 20:25

answered Sep 26, 2017 at 4:29

Arun Raaj

1,800 ● 1 ● 22 ● 20

10

```
var source = { firstName="Jane", lastname="Jones"
};
var shallow = ShallowCopyOf(source);
var deep = DeepCopyOf(source);
source.lastName = "Smith";
WriteLine(source.lastName); // prints Smith
WriteLine(shallow.lastName); // prints Smith
WriteLine(deep.lastName); // prints Jones
```

Share Improve this answer

Follow

edited Oct 8, 2008 at 20:39

answered Oct 8, 2008 at 20:34

Dour High Arch

21.7k ● 30 ● 77 ● 93

That's not a good example. Shallow copies are mostly used for quick copying of objects, without copying the data, but once an objects needs to modify the shared data, a deep copy of it is taken. Your example will likely confuse beginners. – CMircea Apr 17, 2010 at 9:05

this only works in languages that use pointers to represent strings. The point that DHA is trying to make is that shallow copy only duplicates pointers to the identical (singular) original content, while deep copy clones the referenced content of the pointers as well. Both methods copy surface content. If the language stores strings as surface literal content, e.g. inside a WAV header, this example will not work. Note this is probably too picky for most real-life problems that are not esoteric. – DragonLord Nov 16, 2014 at 1:13

10

Shallow Copy- Reference variable inside original and shallow-copied objects have reference to **common** object.

Deep Copy- Reference variable inside original and deep-copied objects have reference to **different** object.

> clone always does shallow copy.

```
public class Language implements Cloneable{

    String name;
```

```
    public Language(String name){
        this.name=name;
    }

    public String getName() {
        return name;
    }

    @Override
    protected Object clone() throws
CloneNotSupportedException {
        return super.clone();
    }
}
```

main class is following-

```
public static void main(String args[]) throws
ClassNotFoundException,
CloneNotSupportedException{

      ArrayList<Language> list=new
ArrayList<Language>();
      list.add(new Language("C"));
      list.add(new Language("JAVA"));

      ArrayList<Language> shallow=
(ArrayList<Language>) list.clone();
      //We used here clone since this always
shallow copied.

      System.out.println(list==shallow);

      for(int i=0;i<list.size();i++)

System.out.println(list.get(i)==shallow.get(i));//tr

      ArrayList<Language> deep=new
ArrayList<Language>();
      for(Language language:list){
          deep.add((Language) language.clone());
      }
```

```
        System.out.println(list==deep);
        for(int i=0;i<list.size();i++)

System.out.println(list.get(i)==deep.get(i));//false

}
```

OutPut of above will be-

> false true true
>
> false false false

Any change made in origional object will reflect in shallow object not in deep object.

```
  list.get(0).name="ViSuaLBaSiC";
  System.out.println(shallow.get(0).getName()+"
"+deep.get(0).getName());
```

OutPut- ViSuaLBaSiC C

Share Improve this answer

Follow

edited Jun 20, 2020 at 9:12

Community Bot

1 ● 1

answered May 7, 2015 at 7:12

user4768611

A shallow copy constructs a new compound object and insert its references into it to the original object.

9

Unlike shallow copy, deepcopy constructs new compound object and also inserts copies of the original objects of original compound object.

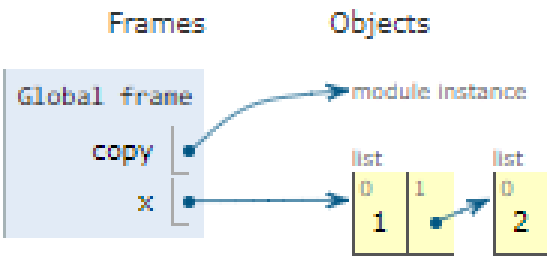
Lets take an example.

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

Above code prints FALSE.

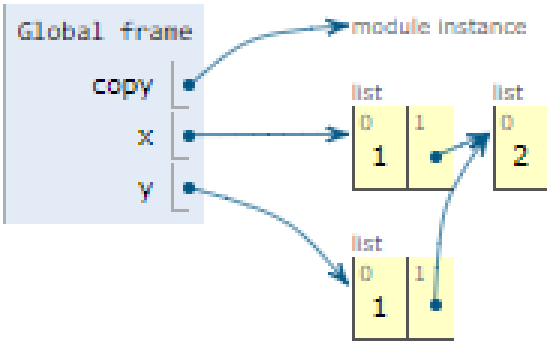
Let see how.

Original compound object `x=[1,[2]]` (called as compound because it has object inside object (Inception))



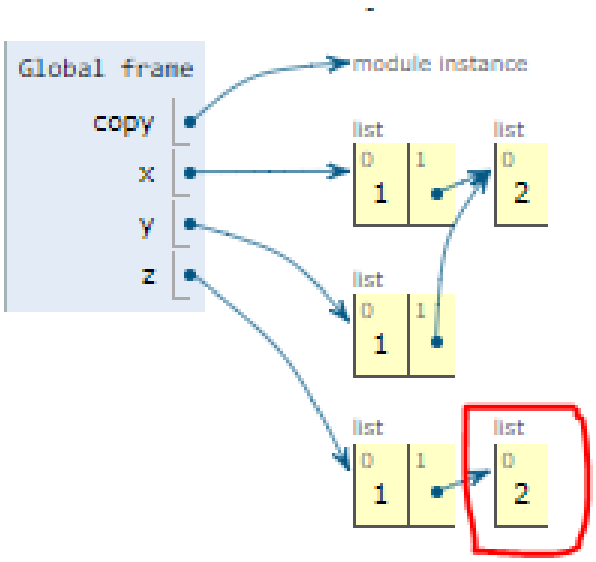
as you can see in the image, there is a list inside list.

Then we create a shallow copy of it using `y = copy.copy(x)`. What python does here is, it will create a new compound object but objects inside them are pointing to the orignal objects.



In the image it has created a new copy for outer list. but the inner list remains same as the original one.

Now we create deepcopy of it using `z = copy.deepcopy(x)`. what python does here is, it will create new object for outer list as well as inner list. as shown in the image below (red highlighted).



At the end code prints `False`, as y and z are not same objects.

Share Improve this answer

Follow

edited Nov 10, 2023 at 7:05

Nimantha

6,438 • 6 • 30 • 75

answered Jan 20, 2018 at 7:37

Sushant
180 • 3 • 10

8

Imagine there are two arrays called arr1 and arr2.

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

Share Improve this answer

Follow

answered May 29, 2015 at 0:41

PeerNet
855 • 1 • 16 • 32

7

In Simple Terms, a Shallow Copy is similar to Call By Reference and a Deep Copy is similar to Call By Value

In Call By Reference, Both formal and actual parameters of a function refers to same memory location and the value.

In Call By Value, Both formal and actual parameters of a functions refers to different memory location but having the same value.

Share Improve this answer

Follow

answered Dec 2, 2011 at 0:55

santhosh
71 • 1 • 1

```
struct sample
{
    char * ptr;
```

5

```
}
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);
}
```

Share Improve this answer

Follow

answered Aug 26, 2011 at 6:33

notytony

1,052 ● 2 ● 11 ● 13

4

To add more to other answers,

- a Shallow Copy of an object performs copy by value for value types based properties, and copy by reference for reference types based properties.
- a Deep Copy of an object performs copy by value for value types based properties, as well as copy by value for reference types based properties deep in the hierarchy (of reference types)

Share Improve this answer

Follow

answered Sep 29, 2016 at 16:44

S2S2

8,492 ● 5 ● 41 ● 68

Shallow copy will not create new reference but deep copy will create the new reference.

4

Here is the program to explain the deep and shallow copy.

```
public class DeepAndShollowCopy {
    int id;
    String name;
    List<String> testlist = new ArrayList<>();

    /*
    // To performing Shallow Copy
    // Note: Here we are not creating any references.
      public DeepAndShollowCopy(int id, String name, L
       {

       System.out.println("Shallow Copy for Object ini
       this.id = id;
       this.name = name;
       this.testlist = testlist;

       }
    */

    // To performing Deep Copy
    // Note: Here we are creating one references( Al a
    public DeepAndShollowCopy(int id, String name, Lis
        System.out.println("Deep Copy for Object initi
        this.id = id;
        this.name = name;
        String item;
        List<String> Al = new ArrayList<>();
        Iterator<String> itr = testlist.iterator();
        while (itr.hasNext()) {
            item = itr.next();
            Al.add(item);
        }
        this.testlist = Al;
    }


    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("Java");
```

```
            list.add("Oracle");
            list.add("C++");
            DeepAndShollowCopy copy=new DeepAndShollowCopy
            System.out.println(copy.toString());
        }
        @Override
        public String toString() {
            return "DeepAndShollowCopy [id=" + id + ", nam
testlist=" + testlist + "]";
        }
}
```

Share Improve this answer

Follow

edited Aug 23, 2018 at 0:56

Pang

10.1k ● 146 ● 85 ● 124

answered Mar 6, 2017 at 9:50

Lova Chittumuri

3,294 ● 1 ● 34 ● 38

3

Taken from [blog]: http://sickprogrammersarea.blogspot.in/2014/03/technical-interview-questions-on-c_6.html

Deep copy involves using the contents of one object to create another instance of the same class. In a deep copy, the two objects may contain ht same information but the target object will have its own buffers and resources. the destruction of either object will not affect the remaining object. The overloaded assignment operator would create a deep copy of objects.

Shallow copy involves copying the contents of one object into another instance of the same class thus

creating a mirror image. Owing to straight copying of references and pointers, the two objects will share the same externally contained contents of the other object to be unpredictable.

Explanation:

Using a copy constructor we simply copy the data values member by member. This method of copying is called shallow copy. If the object is a simple class, comprised of built in types and no pointers this would be acceptable. This function would use the values and the objects and its behavior would not be altered with a shallow copy, only the addresses of pointers that are members are copied and not the value the address is pointing to. The data values of the object would then be inadvertently altered by the function. When the function goes out of scope, the copy of the object with all its data is popped off the stack.

If the object has any pointers a deep copy needs to be executed. With the deep copy of an object, memory is allocated for the object in free store and the elements pointed to are copied. A deep copy is used for objects that are returned from a function.

answered Mar 7, 2014 at 8:39

Santosh

1,304 ● 2 ● 16 ● 31

I came to understand from the following lines.

3

Shallow copy copies an object **value type**(int, float, bool) fields in to target object and object's reference types(string, class etc) are copied as **references** in target object. In this target reference types will be pointing to the memory location of source object.

Deep copy copies an object's value and reference types into a complete new copy of the target objects. This means both the value types and reference types will be allocated a new memory locations.

Share Improve this answer

Follow

answered Jul 28, 2018 at 12:54

Nayas Subramanian

2,379 ● 23 ● 31

2

Shallow copying is creating a new object and then copying the non-static fields of the current object to the new object. If a field is a value type --> a bit-by-bit copy of the field is performed; for a **reference type** --> the reference is copied but the referred object is not; therefore the original object and its clone refer to the same object.

Deep copy is creating a new object and then copying the nonstatic fields of the current object to the new object. If a field is a **value type** --> a bit-by-bit copy of the field is performed. If a field is a **reference type** --> a new copy of the referred object is performed. The classes to be cloned must be flagged as [Serializable].

Share Improve this answer

Follow

answered Aug 8, 2013 at 8:32

Rajaram Shelar
7,837 ● 25 ● 70 ● 110

2

To add just a little more for confusion between shallow copy and simply assign a new variable name to list.

"Say we have:

```
x = [
    [1,2,3],
    [4,5,6],
    ]
```

This statement creates 3 lists: 2 inner lists and one outer list. A reference to the outer list is then made available under the name x. If we do

```
y = x
```

no data gets copied. We still have the same 3 lists in memory somewhere. All this did is make the outer list available under the name y, in addition to its previous name x. If we do

```
y = list(x)
```

or

```
y = x[:]
```

This creates a new list with the same contents as x. List x contained a reference to the 2 inner lists, so the new list will also contain a reference to those same 2 inner lists. Only one list is copied—the outer list. Now there are 4 lists in memory, the two inner lists, the outer list, and the copy of the outer list. The original outer list is available under the name x, and the new outer list is made available under the name y.

The inner lists have not been copied! You can access and edit the inner lists from either x or y at this point!

If you have a two dimensional (or higher) list, or any kind of nested data structure, and you want to make a full copy of everything, then you want to use the deepcopy() function in the copy module. Your solution also works for 2-D lists, as iterates over the items in the outer list and makes a copy of each of them, then builds a new outer list for all the inner copies."

source: https://www.reddit.com/r/learnpython/comments/1afldr/why_is_copying_a_list_so_damn_difficult_in_python/

Share Improve this answer

Follow

answered Jan 12, 2017 at 6:10

Lance Ruo Zhang

401 ● 4 ● 12

Copying ararys :

1

Array is a class, which means it is reference type so array1 = array2 results in two variables that reference the same array.

But look at this example:

```
  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]);
    }
```

shallow clone means that only the memory represented by the cloned array is copied.

If the array contains value type objects, the values are copied;

if the array contains reference type, only the references are copied - so as a result there are two arrays whose members reference the same objects.

To create a deep copy—where reference type are duplicated, you must loop through the array and clone each element manually.

Share Improve this answer Follow

edited Dec 20, 2012 at 16:59

answered Dec 19, 2012 at 21:23

komizo

1,051 ● 14 ● 21

I don't know about other languages, but in C#/VB, shallow copying an array of value types does **not** copy the values. The two arrays refer to the same objects. Add a button to a form and add this code to see: `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]); }` – DeanOC Dec 19, 2012 at 21:38

you are right, i corrected my answer to be more precise, using clone on arrays. You are absolutely right that "shallow copying an array of value types does not copy the values", but using clone on array does. I've tried to explain that, try it. Thanks – komizo Dec 20, 2012 at 16:56

1

Adding to all the above definitions, one more and most commonly used deep copy, is in the copy constructor (or overloading assignment oprator) of the class.

Shallow copy --> is when you are not providing copy constructor. Here, only the object gets copied but not all the members of the class are copied.

Deep copy --> is when you have decided to implement copy constructor or overload assignment in your class and allows copying all the members of the class.

```
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
}
```

Share Improve this answer

Follow

answered Dec 20, 2013 at 19:01

Avinash Goud N J
89 • 2