

Forum:[Hack Forums](#)

Thread: [Initialize a list<class> in constructor](#)

Tafelpoot - March 15, 2016, 9:03 am

Hi,

I have a class "Tracks" and I have a class "TrackClass", in the "TrackClass" I have a list<Tracks> which takes a object of class "Tracks".

Code private List<Tracks> tracks; Now I have to create 2 constructors in the "TrackClass" 1 takes 0 parameters and the other one takes 1 parameter which is a list with "Track" class objects. like this:

Code public TrackList()

```
{
    tracks = new List<Tracks>();
}
```

public TrackList(List<Tracks> tracks)

```
{
    this.tracks = new List<Tracks>(tracks);
}
```

}So am I initializing the List properly here? (The private List<Tracks> tracks)

And why do I need to initialize this list when an object of "TrackClass" is generated? Why is that usefull/needed?

Here is the full code of "TrackClass" :

Code using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AudioDevices.Tracks

```
{
    class TrackList
    {
        private List<Tracks> tracks;

        public TrackList()
        {
            tracks = new List<Tracks>();
        }
        public TrackList(List<Tracks> tracks)
        {
            this.tracks = new List<Tracks>(tracks);
        }

    }
}
```

Thanks.

Odus - March 15, 2016, 9:30 am

If you're not initializing then it becomes an assignment, instead you're moving out of the list i.e. cloning it. By using:

Code `this.tracks = tracks;`

Updating the previous list (the one you passed to the constructor) would update that one too.

Tafelpoot - March 15, 2016, 10:18 am

(Mar 15, 2016 09:30 AM)Odus Wrote: [\[CITE\]](#) If you're not initializing then it becomes an assignment, instead you're moving out of the list i.e. cloning it. By using:

Code `this.tracks = tracks;`

Updating the previous list (the one you passed to the constructor) would update that one too.

So what I am doing now is the same as using `"this.tracks= tracks"`?

So am I doing this correct or is it wrong? Also because I use `"this.tracks"` in the second constructor and `"tracks"` in the first constructor?

Also would this work?:

Code `Tracks T1 = new Tracks();`

`TrackList TL = new Tracklist(T1);` Would this then then add the T1 to the private field list?

PS: I just started with C# so I did not really understand what you said

PS: With the first constructor I call the list into existence and with the second one I say this object is added to this private field? Or am i thinking this wrong?

Odus - March 15, 2016, 10:29 am

(Mar 15, 2016 10:18 AM)Tafelpoot Wrote: [\[CITE\]](#) So what I am doing now is the same as using `"this.tracks= tracks"`?

So am I doing this correct or is it wrong? Also because I use `"this.tracks"` in the second constructor and `"tracks"` in the first constructor?

No, sorry to confuse you. You were originally copying the list over to a new one, the code I posted would assign it instead, it was just a demo.

The 'this' keyword differentiates between members sharing a name. Using it refers to the instance of the class, therefore its private variable: `tracks`. So you will need it in the overload since your arg shares the same alias as an instance member.

(Mar 15, 2016 10:18 AM)Tafelpoot Wrote: [\[CITE\]](#) So what I am doing now is the same as using `"this.tracks= tracks"`?

So am I doing this correct or is it wrong? Also because I use `"this.tracks"` in the second constructor and `"tracks"` in the first constructor?

Also would this work?:

Code `Tracks T1 = new Tracks();`

`TrackList TL = new Tracklist(T1);` Would this then then add the T1 to the private field list?

If Type 'TrackList' accepts an arg of type 'Tracks' then it will work, but in your constructor you would have to first initialize the list and then add to it manually.

Tafelpoot - March 15, 2016, 10:55 am

So the `"this.tracks = tracks;"` will assign and the `"new List<Tracks>(tracks);"` will clone.

Is that not the same? `i = 1;` < assign, `i = b;` clone. But that means the `"= tracks"` and `"new list<tracks>(tracks)"` are both doing the same thing.

Also why do I not need to use the `"new List<Tracks>(tracks);"`, Because my private field only has `"List<Tracks> tracks;"`.

Assignment 1 says: Within the first Constructor(without parameters) the private field tracks(the list<Tracks>) needs to be initialized with an empty instance. What i thought this meant was this:
"tracks = new List<Tracks>();"

Assignment 2 says: Within Constructor 2(with parameters) people will send a List<Tracks> with Tracks objects(Track class). This list must be assigned(I atleast I think that is the correct translation assuming you use that word) to the private field tracks.

TL;DR

Difference between:

```
this.tracks = tracks;  
tracks = new List<Tracks>();  
tracks = new List<Tracks>(tracks);
```

Thanks for helping

Odus - March 15, 2016, 11:08 am

(Mar 15, 2016 10:55 AM)Tafelpoot Wrote: [\[CITE\]](#) Difference between:

```
this.tracks = tracks;  
tracks = new List<Tracks>();  
tracks = new List<Tracks>(tracks);
```

- 1) Assigns, updating through the previous alias will update this one.
- 2) Initializes a new List<Tracks> but does not populate.
- 3) Like the above, only the contents of the tracks arg are copied into this one. They then become independent.

Tafelpoot - March 15, 2016, 12:57 pm

Oke, So the first assignemnt say to initialize an empty list so I assume that I will need to use: tracks = new List<Tracks>(); then.

and assignment 2 says to assign the list i get from the parameter to the list tracks, So I assume i will need to use: this.tracks = tracks.

Why do I write this:

Code private List<Tracks> tracks;Instead of this:

Code private List<Tracks> tracks = new List<Tracks>();?

+Haze - March 15, 2016, 1:10 pm

Because you're only declaring it on the first example then you're declaring and assigning on the second all in one line.

Tafelpoot - March 15, 2016, 1:38 pm

I think I got it now, Whenever I don't get a parameter I initialize the List<t> but it is 0 or null(not sure). But whenever I get a parameter(a list with objects) It will assign to the track list which is basically the same as initializing as the above but then adding a different value.

Code int id;

```
public MyClass()  
{  
    id = 0;  
}
```

```
public MyClass(int id) //constructor
```

```
{
    this.id = id;
```

}The above code is basically the same, I initialize in the first(empty) and I initialize in the second by assigning a value.

So basically assigning a value also initiates.

Correct me if I am wrong please.

Thanks a lot for your help.

Odus - March 15, 2016, 2:37 pm

(Mar 15, 2016 01:38 PM)Tafelpoot Wrote: [\[CITE\]](#) int id;

```
public MyClass()
{
    id = 0;
}
```

```
public MyClass(int id) //constructor
{
    this.id = id;
}
[/code]
```

The above code is basically the same, I initialize in the first(empty) and I initialize in the second by assigning a value.

So basically assigning a value also initiates.

Correct me if I am wrong please.

That's spot on. If this isn't homework then it would probably be easier to make parameter 'id' optional. That way you don't need an overload for such little difference.

Tafelpoot - March 15, 2016, 3:28 pm

Yeh it is homework so I will have to do it this way, I also learned that if i use "this.tracks = new List<Tracks>(tracks);" it will create a new list.

So my field variable will point(reference) to that new list which has the content of the parameter. But if i dont use new but just use "this.tracks = tracks" my field variable will point(reference) to the parameter instead if initiating a new list.

This is how i wrote it down in my OneNote file:

```
private List<Tracks> tracks; is a reference to nothing atm.
this.tracks = tracks makes the tracks list ^ reference to the list that was send with parameters.
this.tracks = new List<Tracks>(tracks); will create a new list, So the tracks list will now reference
that new list, and the new list has the same content as the list that was send with parameters.
```

BTW: How do i make the ID parameter optional(You said that in your last comment) so that I dont need to use overload methods.

The-One - March 16, 2016, 9:01 am

(Mar 15, 2016 03:28 PM)Tafelpoot Wrote: [\[CITE\]](#) --snip--

You can do it like this:

Code public class MyClass

```
{
    int id;
    public MyClass(int id = 0) //constructor
    {
        this.id = id;
    }
}
```

You can now use:

Code MyClass .. = new MyClass(); // id = 0

MyClass .. = new MyClass(10); // id = 10

+Haze - March 16, 2016, 10:05 am

(Mar 16, 2016 09:01 AM)The-One Wrote: [\[CITE\]](#) You can do it like this:

```
Code int id;
public MyClass(int id = 0) //constructor
{
    this.id = id;
}
```

You can now use:

Code MyClass .. = new MyClass(); // id = 0

MyClass .. = new MyClass(10); // id = 10

Isn't int id supposed to be defined inside the class?

The-One - March 16, 2016, 10:26 am

(Mar 16, 2016 10:05 AM)+Haze Wrote: [\[CITE\]](#) Isn't int id supposed to be defined inside the class?

It should, I just left out the class definition because I thought that part would be obvious. I'll add it to prevent confusion.

?