

Java - best way to implement a dynamic-size array of objects

Asked 14 years, 1 month ago Modified 7 years, 4 months ago

Viewed 28k times  Part of [Mobile Development](#) Collective



12



I am a novice with Java.

I have to implement an array of objects that changes in size during execution.

The code I am writing is going to be ported on Android, too.



According to your experience, what's the best class to implement that?

Thanks,
Dan

MD **java** **android**

Share

Improve this question

Follow

asked Nov 15, 2010 at 21:06



Dan

16.2k ● 20 ● 64 ● 92

2 That's difficult to answer without knowing the requirements.
Take a look at

download.oracle.com/javase/tutorial/collections/index.html,
and see which one fits the bill. – [Oliver Charlesworth](#) Nov 15,
2010 at 21:08

Why not use the ones in the API, ArrayList for example?
– [dacwe](#) Nov 15, 2010 at 21:15

4 Answers

Sorted by: Highest score (default) 



22

Java has an inadequate template capability. As long as you want an array of objects, then `ArrayList<T>` is good. For primitives, it's awful.



Assuming you have a hierarchy of objects that you want to put in a list, `ArrayList` is ideal:



```
ArrayList<Vehicle> vehicles = new ArrayList<Vehicle>()  
  
vehicles.add(new Car(...));  
vehicles.add(new Truck(...));
```

I'm assuming in the above example that Vehicle is the base class, and Car and Truck are subclasses.

On the other hand, if you want a list of numbers, Java is highly inefficient. Each object is a reference (really a 4 byte pointer) to a 12 byte chunk of memory, plus what you're actually using. Since ArrayList cannot apply to int, this means that creating a list of numbers means:

1. Creating a list of Integer, the object wrapper for int.

2. Converting the objects every time you pull out a number. This is done automatically these days, but it takes time.
3. Initializing 5 times as much storage as needed.

So, if you are manipulating big chunks of primitive data (int, float, double) it can be worth your while writing your own version of ArrayList. This is particularly important when the data is big and the platform is small (like a handheld Android thingy).

Compare this:

```
ArrayList<Integer> list = new ArrayList<Integer>();  
for (int i = 0; i < 1000000; i++)  
    list.add(i);
```

to:

```
public class IntArray {  
    private int[] data;  
    private int used;  
    private void grow() {  
        // implement code to make data double in size here...  
    }  
    public IntArray(int size) {  
        data = new int[size];  
        used = 0;  
    }  
  
    public void add(int i) {  
        if (i >= data.length) grow();  
        data[used++] = i;  
    }  
}
```

```
IntArray list2 = new IntArray(1000000);  
for (int i = 0; i < 1000000; i++)  
    list2.add(i);
```

The last time I benchmarked it, the optimal use of the primitive list is more than 10 times faster than the admittedly suboptimal use of ArrayList. To be more fair, pre-allocate the arraylist to be the right size -- it's still way slower.

LinkedList is only worthwhile if you are inserting in the beginning or middle of a list. If your list is being built by adding to the end, ArrayList will thoroughly dominate LinkedList. So for a typical list of objects which you are building up in order, ArrayList is what you are looking for. For a big list of primitives like int or double, write your own list.

Share Improve this answer

edited Aug 5, 2017 at 13:27

Follow


answered Nov 15, 2010 at 21:36

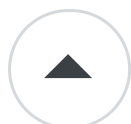


Dov

8,484 ● 9 ● 49 ● 81

While uses where a `LinkedList` is best may be rare in comparison to `ArrayList`, they do exist and `LinkedList` can perform *significantly* better in them. Queue-like use of a list is one example... removing from the front of an `ArrayList` is extremely expensive in comparison to `LinkedList`. See [this](#) and [this](#). – ColinD Nov 15, 2010 at 22:40

@Colin, Using an ArrayList naively instead of a LinkedList can indeed be a bad thing. But if you want a faster queue, then I'll bet an implementation of a circular queue with specified head/tail will beat LinkedList by a huge margin -- even if you can make it grow when/if you need to, the average case will be just placing objects within an already existing list. Yes, inserting at the beginning of an ArrayList is costly. – [Dov](#) Nov 16, 2010 at 15:39 



11

You would probably be most interested in [ArrayList](#).

I think even wikipedia has info on using generic lists in java: http://en.wikipedia.org/wiki/Generics_in_Java



Share Improve this answer

answered Nov 15, 2010 at 21:07



Follow



[Quentin Robinson](#)

82.3k ● 14 ● 126 ● 132



3

You should familiar your self with the [collection framework](#) in java. You have things like Set, List, Map, SortedSets, etc... Which can be really helpful data structures.



Share Improve this answer

answered Nov 15, 2010 at 21:10



Follow



[Amir Raminfar](#)

34.1k ● 7 ● 95 ● 125



It depends on what you're gonna use it for.

0



If you have an array that changes size frequently, then I'd recommend using an [ArrayList](#) or something else from the [collection framework](#) (depending on what you're gonna use it for).

If you however have an array that changes size very infrequently but is read frequently, it would probably be faster to use a regular array and then resize it when needed instead

```
Arrays.copyOf(array, newSize);
```

Hope it helps! :)

Share Improve this answer

answered Nov 15, 2010 at 21:15

Follow



Anton

378 ● 2 ● 7

While primitive arrays are best sometimes, there isn't really any good reason to use an array of objects in Java. "it would probably be faster"... premature optimization. – [ColinD](#) Nov 15, 2010 at 21:25

Using a primitive class is different than writing with hardcoded arrays. And way too much attention is paid to avoiding "premature optimization" rather than simply knowing what is efficient, and doing it as a matter of course. It's just as easy to write good code in most cases as bad code.

Sometimes, it takes a little more work, and that's when you can think about doing it later. – [Dov](#) Nov 15, 2010 at 21:40

@Dov: I certainly agree that when it's a matter of doing something clearly inferior vs. doing something that is correct and will likely perform better for the same effort, it's not

premature optimization to do the right thing. But arrays are inferior to Lists in a great number of ways and using them directly will almost certainly create more work to do simple things and make the code harder to understand, and will likely not convey any performance improvement that matters for all but the most low-level of code. That's why I'd consider this premature optimization. – [ColinD](#) Nov 15, 2010 at 22:20
