Java Collections

Melvyn Ian Drag

September 10, 2021

Abstract

java.util provides many containers. These containers are widely used in Java programming and are implementations of the great data structures you hear about in data structures & algorithms classes. In today's lecture we'll have a look at a few of them and consider when we would want to use them. Before getting to that interesting material, however, we'll review primitives vs objects, the static keyword, and the final keyword.

1 Introduction

Lets get the basic stuff out of the way. These are little things that don't fit anywhere, but they are cool and useful.

2 The final keyword

Dry opening. what are some constant values you might include in a program? I mean, what are some :final: values you will include? Maybe you write a mathematical program and you want to put a constant value to use in your formulas.

Ask class. My ideas: PI, Avogadros Number, e (eulers number), SpeedOfLight. How many protons does an oxygen atom have? This would be a constant that someone writing a chemistry program would put in his code. How many seats are there on a Boeing 747? Someone writing the program to handle booking tickets for a flight would need to write this into te code as a constant value.

`https://www.tutorialspoint.com/can-we-declare-final-variables-without-initialization-in-`
The final keyword in Java is for constants. But Java is very cool about how it handles the final keyword - it means you can assign to the value only once. But you don't have to initialize the variable as soon as you declare it!

In many other languages you must do

```
const int i = 1;
```

but you cannot do

```
const int i;
i = 1;
```

3 The static keyword

We use static in every single Java program we write - somewhere, in at least one of our classes, we create a method called ‘public static void main(String[] args)‘

static is an annoying keyword - can I say that? Do you mind if I call it annoying? Its confusing, maybe that’s a better word. In English itself we don’t really use the word ”static”. And then when we do use it we use it to mean something that doesnt change (as opposed to dynamic), or we might talk about when the radio isn’t easy to hear you call the noise (ask class to make the noise, make sure that the foreign students know what I’m talking about) static.

Ok - so what does static mean in Java? It means that the thing marked static belongs to the class, and not to a particular instance of the class. So for the last homework you had to do:

```
Car truck = new FireTruck();
truck.SprayWater();
```

That’s because ”SprayWater” is an *instance* method - it’s a method that instances of a class can call. Static methods are not tied to an instance of a class. You dont need to create an object to use a static method.

Example 1: The main method is static. Why? it doesnt require you to new up a class instance, you can call main without a class instance.

Write simple program. then ask students if we’ve instantiated any objects - did we call ”new”? No. But we’re still calling a method if we run this. Which method? main().

Extend the example to create a static method and a non-static method. Ask class which one we can call in main( )without instantiating an object. (answer: the static one). Then ask, how can we ever use the non-static method? (answer: first new up an object)

Example 2: Standard example Note that static is not just applied to methods. You can also share variables / members across a program. In this program we’ll create a static member variable called numFuzzyWuzzies, and it will store the numbre of fuzzyWuzzies that we create.

Every time you create an object, you increment the counter. Create a fuzzy wuzzy class. Every time you instantiate it, update the static member numFuzzyWuzzies.

Understanding the static keyword will be very important when we talk about multithreading in a couple of weeks.

In summary: the static keyword means that a variable or method does not belong to any instance of a class - it is instead property of the class itself, and it can be used througout your whole program.

3.1 bonus

A question I don’t know the answer to - in Java can you call a static method from a class instance? Like truck.StaticSprayWater()?There would be no reason to do this, but I’m just curious to know.

4 Data types

4.1 Almost everything in Java is an Object

All objects in Java, including arrays, inherit some basic functionality from the Object class.So when you make a base object in Java, it isn’t really the base object, there is the Object class below it.

An example of methods you get from the Object class are toString() and clone().

clone() method would be fun to play with - is it a deep copy? Shallow copy? or just the sae exact object with a new name?

4.2 Primitive types are not objects.

- int
- double
- float
- char
- byte
- bool

Interesting that in java the byte datatype is 16 bits. This will be very important in the coming weeks when we look at the way that computers do binary encoding of text data. In many languages a char is 8 bits! But java has an 8 bit type called "byte" instead. Very interesting history behind why this is.

4.3 Java classes that had me confused

When I learned Java I was in a hurry. Had no book and no guidance, so I was mainly just reading code I found online, compiling it, and tweaking it to see what happened.

At that time I had no idea what the point of these types were! If there is an int, why does Java give us Integer? Are they the same? Different?

- Integer
- Double
- Character
- Boolean
- Float
- Byte

I can tell you now - these types are the Object form the underlying primitive. So an Integer is like an int, but it's an object instead of a primitive.

One immediate benefit of this is that you can use certain libraries that expect objects. Some libraries dont want primitives.

BREAK before we move on to LISTS

5 Introduction

A java collection is a container you can use to store a bunch of values. For example, in your program you may need to store a bunch of ages - in this case you would

```
...
int[] ageArray = {19, 25, 13, 41, 15};
...
```

or you might create a class called 'Person' and when want to store a bunch of People. In this case you might

```
Person[] peopleArr = { new Person(), new Person(), ...};
```

in these two examples I am using arrays. While arrays are fine for simple collections of items, there are some distinct disadvantages of using arrays compared to using a more elegant container. There are some cases where using an array is extremely problematic. One major disadvantage is that the array size is fixed. Once you create an array of 5 elements , you cannot add a sixth person to the array.

There are many Java Collections. For example,

1. ArrayList
2. LinkedList
3. Vector
4. PriorityQueue
5. ArrayDeque
6. HashSet
7. LinkedHashSet
8. TreeSet
9. HashMap
10. TreeMap
11. LinkedHashMap
12. ConcurrentListSkipMap
13. WeakHashMap
14. Stack
15. etc.

We are going to focus on just a few of them today. Namely, we will compare

1. ArrayList
2. LinkedList

6 What is a list?

A list is an abstract idea - it's a bunch of objects in a row, like the arrays that we've seen so far. The list contains a bunch of items. You can delete items from it and add items to it. As I said before - big drawback of arrays is that the size of an array is fixed.

```
public class IntArrayCreator{
  public static void main(String[] args){
    // 1, 2, 3, 4
    int[] array1 = new int[]{1, 2, 3, 4};

    // 1, 2, 3, 4
    int[] array2 = {1, 2 ,3, 4};

    // 0,0,0,0
    int[] array3 = new int[4];
  }
}
```

Note that all of the above arrays have four elements in them, and that cannot be changed! Sometimes that's what you want, sometimes thats not. Your call. If you want a variable number of elements in your container, use a List not an array.

7 What is an ArrayList?

7.1 overview

An ArrayList is a container like an array, but the size isn't fixed. You can put some elements of a specified type in them. See this example to create ArrayLists of different types:

```
import java.util.ArrayList;

public class ArrayListDemo{
  public static void main(String[] args){
    ArrayList<Integer> intArrayList = new ArrayList<Integer>();
    intArrayList.add(1);
    intArrayList.add(100);
    for( int i : intArrayList ){
      System.out.println(i);
    }

    ArrayList<MyClass> classList = new ArrayList<MyClass>();
    classList.add( new MyClass() );
    classList.add( new MyClass() );
    classList.add( new MyClass() );

    for( MyClass mc : classList ){
      System.out.println( mc.x );
    }
```

```
  }
}

class MyClass{
  public int x = 1;
}
```

WARNING! This code won't work:

```
...
ArrayList<int> intArrayList = new ArrayList<int>();
...
```

because you can't put primitive types in an ArrayList - only reference types (i.e. everything except primitives) You can't put any of these in an ArrayList

- byte
- float
- int
- double

but the "Byte", "Float", "Int" and "Double" types are okay, because these aren't primitives, they are classes.

8 Exercise: Create 5 different ArrayLists, each holding a different type.

Allow 10 minutes to ensure that everyone can use the arraylists successfully.

9 What is a LinkedList?

To be perfectly honest, I've not yet researched how linked lists are implemented in Java, but I have studied algorithms and I've see how they are implemented in other languages. I'll give you a bit of information now and then we'll do some experiments to see if Java aligns with our expectations.

Read the wikipedia article on linked lists

I will improvise this section about a linked list. If you don't understand the concept after our discussion in class, you can look on youtube. Sometimes hearing different people say the same things makes those things easier to understand.

10 Further Illustration

Draw a linked list on the board. Draw an Array List on the board Show. ArrayList uses continguous memory. Linked List uses references between non-contiguous blocks of memory

Illustration: Have a bunch of students sitting next to each other hold up their array index on a piece of paper. That's an array list. Give the students the indices and have them point to each other. These students don't sit together. This is a linked list.

Illustrate an add operation. Show how to add to an array list means we have to displace the people sitting after the insertion index, but we don't have to find the insertion index because we already know where it is. For a linked list, during insertion, we have no idea where in memory the 5th person is - so we have to start at 0 and work our way through the references, then we insert the person. This is why ArrayList and LinkedList have different run times for insertion. The complexity is O(n) in both, but the cost manifests itself differently, so the actual run time is different.

The above is at once very simple, but also very abstract. I 100% recommend you finish learning Java, then take algorithms, and also learn C or C++. If you understand C, algorithms, and Java, then you will understand what I've said very well. It is almost impossible to explain in isolation without you having learned about computer memory, which comes from a C or C++ class.

11 The Collections Interface

Here is a good picture I found to explain the collections interface:

What are some methods that the collections interface supplies?

11.1 Exercise

I could have written a long lesson on this, but I think it's better that we just find out together. I'm not sure what the collection interface supplies - I could spend an hour now researching and implementing some code, but why don't we do it together?

11.2 Reference

`https://www.javatpoint.com/collections-in-java`

12 Initializing a List<T>with an ArrayList<T>or LinkedList<T

Weird thing that Java programmers do. Java programmers use this pattern to achieve a bunch of complex things that you won't learn about in this class. Grab a good book and learn to write a Java

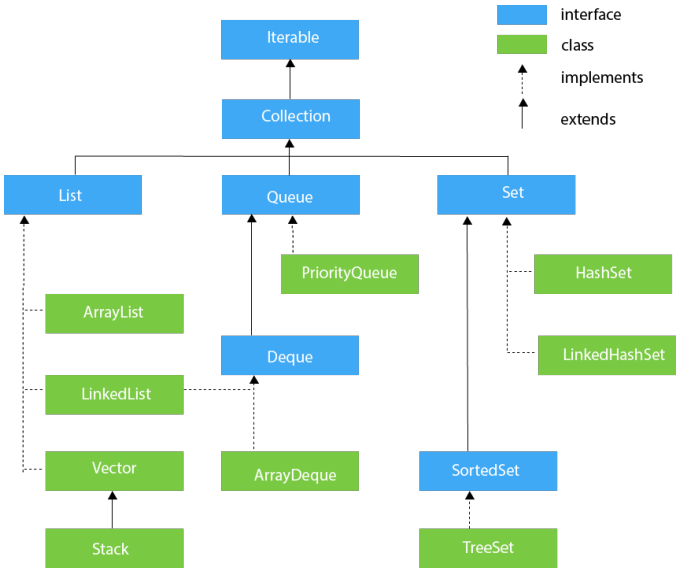

Figure 1: Illustration of a variety of classes which implement the Collection class.

Web App or Android App and you will see this pattern used in certain ways to achieve particular ends.

```
List<String> l = new ArrayList<String>(){"Hello", "World"};
```

13 Timing Comparison

In this section we'll measure how long different datastructures take to perform different tasks. The reason different data structures exist is that they have different strengths and weaknesses, and programmers might need a datastructure that performs well in a certain situation. This talk is somewhat abstract if you haven't taken a data structures and algorithms class. There you learn things like "Computational Complexity" and "Big O" notation. We're only going to focus now on the simplest of examples to give you a taste of whats to come (for those who haven't yet studied data structures). For those who already know about datastructures, I'm hoping you're still interested in discussing them.

13.1 How To Time Code

There are many ways to measure time with Java. I'll just show you one. Java knows how many milliseconds have passed since the **Epoch**. When is the Epoch? Why is it important I can't remember, but it is a commonly used "t0" in computer science, especially on unix systems.

Do the add and remove operations. Discuss the complexity of those operations, and then work it out in class.

13.2 Timing Comparison

Have the students run this code on their computers to see what is happening. Compare results across class

```java
import java.util.LinkedList;
import java.util.ArrayList;

public class TimingTest{
  public static void main(String[] args){

    ArrayList arrayList = new ArrayList();

    LinkedList linkedList = new LinkedList();

    final int N = 1000000;
    final int M = 10000;

    // ArrayList add
    long startTime = System.nanoTime();
    for (int i = 0; i < N; i++) {
      arrayList.add(i);
    }
    long endTime = System.nanoTime();
    long duration = endTime - startTime;
    System.out.println("ArrayList add:  " + duration);

    // LinkedList add
    startTime = System.nanoTime();
    for (int i = 0; i < N; i++) {
      linkedList.add(i);
    }
    endTime = System.nanoTime();
    duration = endTime - startTime;
    System.out.println("LinkedList add: " + duration);

    // ArrayList get
    startTime = System.nanoTime();
    for (int i = 0; i < M; i++) {
      arrayList.get(i);
    }
    endTime = System.nanoTime();
    duration = endTime - startTime;
    System.out.println("ArrayList get:  " + duration);

    // LinkedList get
    startTime = System.nanoTime();
    for (int i = 0; i < M; i++) {
      linkedList.get(i);
    }
    endTime = System.nanoTime();
```

```
    duration = endTime - startTime;
    System.out.println("LinkedList get: " + duration);

        // ArrayList remove
    startTime = System.nanoTime();
    for (int i = M - 1; i >=0; i--) {
      arrayList.remove(i);
    }
    endTime = System.nanoTime();
    duration = endTime - startTime;
    System.out.println("ArrayList remove:  " + duration);

        // LinkedList remove
    startTime = System.nanoTime();
    for (int i = M - 1; i >=0; i--) {
      linkedList.remove(i);
    }
    endTime = System.nanoTime();
    duration = endTime - startTime;
    System.out.println("LinkedList remove: " + duration);
  }
}
```

13.3 ArrayList vs LinkedList

Run this code in front of the class. Compare the run times.

Look at this link https://stackoverflow.com/questions/42849486/why-does-linked-list-delete-a
42849562

Here's the code:

```
import java.util.Random;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;

public class ArrayListTiming{
  public static void main(String[] args){
    final int N = Integer.parseInt(args[0]);
    final int ONE = 1;
    List<Integer> list = new
       ArrayList<Integer>(Collections.nCopies(N, 0));
    long timeStart = System.currentTimeMillis();
    for(int i = 0; i < N; ++i ){
      if(i % 1000 == 0){
        System.out.println("processing iteration: " + i);
      }
      Random rn = new Random();
```

```
      //The below should work in Java, but it doesn't. This is
         unexpected behavior - the mod operator should always
         return positive.
      //int index = rn.nextInt() % N;
      // To fix the problem do this:
      int index = rn.nextInt() % N;
      if(index < 0){
        index += N;
      }
      // More about the issue:
      //https://stackoverflow.com/questions/5385024/mod-in-java-produces-neg
      list.add(index, ONE);
    }
    long timeEnd = System.currentTimeMillis();
    System.out.println(timeEnd - timeStart);

  }
}
```

and then for LinkedList

```
import java.util.Random;
import java.util.List;
import java.util.LinkedList;
import java.util.Collections;

public class LinkedListTiming{
  public static void main(String[] args){
    final int N = Integer.parseInt(args[0]);
    final int ONE = 1;
    List<Integer> list = new
       LinkedList<Integer>(Collections.nCopies(N, 0));
    long timeStart = System.currentTimeMillis();
    for(int i = 0; i < N; ++i ){
      if(i % 1000 == 0){
        System.out.println("processing iteration: " + i);
      }
      Random rn = new Random();
      //The below should work in Java, but it doesn't. This is
         unexpected behavior - the mod operator should always
         return positive.
      //int index = rn.nextInt() % N;
      // To fix the problem do this:
      int index = rn.nextInt() % N;
      if(index < 0){
        index += N;
      }
      // More about the issue:
```

```
      //https://stackoverflow.com/questions/5385024/mod-in-java-produces-neg
      list.add(index, ONE);
    }
    long timeEnd = System.currentTimeMillis();
    System.out.println(timeEnd - timeStart);


  }
}
```

Look at the memory consumption of the ArrayListTiming example for input 1000000!! Java takes up all 4 cpu cores and a bunch of memory!!

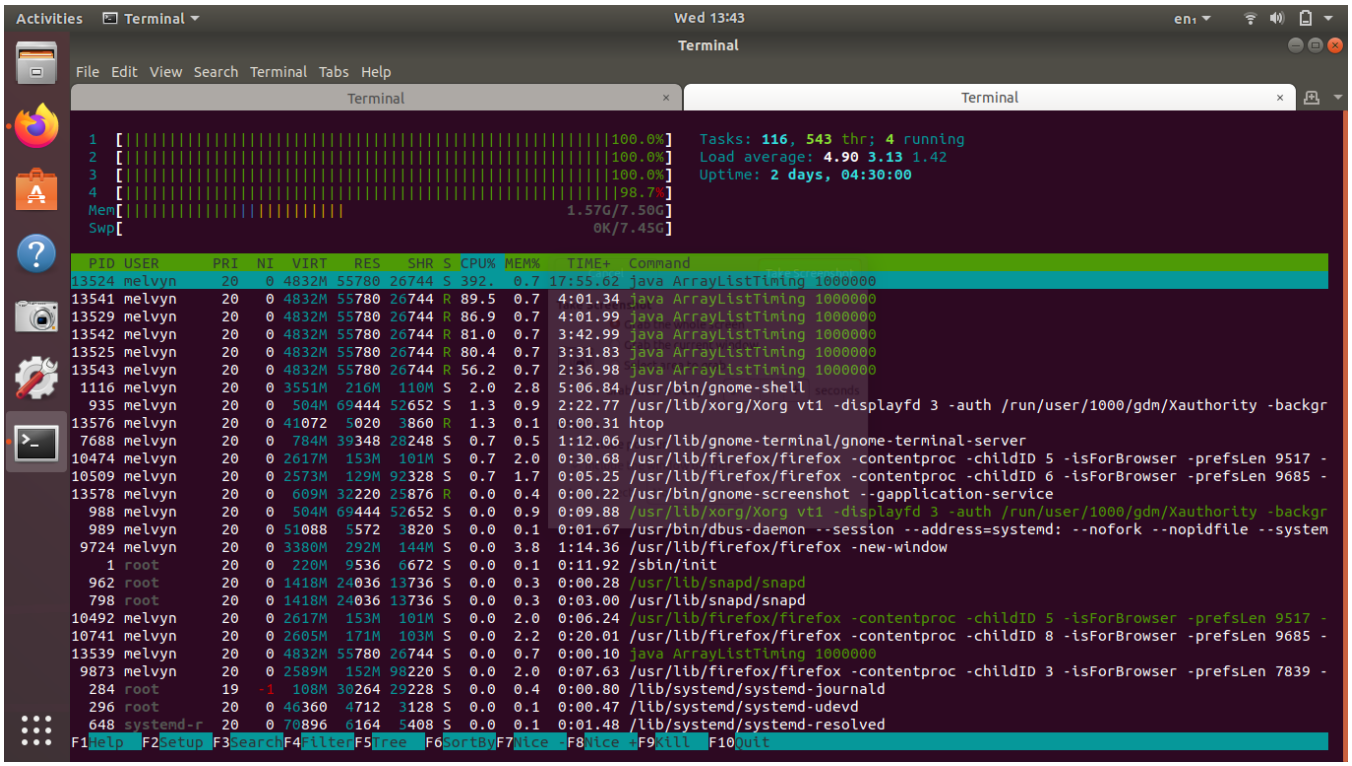

Figure 2: Java Working Hard!

14 Make Computer Run out of memory?

Can you do it, just for fun? Can you make a program that contains such a large, or so many ArrayLists, that the computer runs out of memory?

15 Why Are Collections Useful?

In case I haven't made it abundantly clear through examples by now, collections are very useful and interesting. Here is the official pitch from Oracle about why Java collections are useful: `file:///home/melvyn/Desktop/JavaFall2019ClassRepo/ReadingMaterials/tutorial/collections/intro/index.html`

15.1 highlight these points

- reduces effort - you never need to code up a dynamically resizing array, because Java has one, for example
- increases quality - for example, the linked list implementation you are likely to write will be buggy and slow. The one in the collections framework has been tweaked and perfected by experts for a long time.
- Interoperability between different APIS - since we all agree to use standard collections, all our code can interact. If everyone used a different linked list implementation you couldn't pass data between different peoples' codes.

BTW If you don't know what an **API** is, it means **Application Programming Interface**. It pretty much means the publicly facing part of your code that other people can access. That may not be helpful - an example is warranted here.

There is a good chance that you know next to nothing about linked lists outside of what I've told you today. Even if you have studied algorithms and data structures, you probably implemented a very simplistic linked list. Nevertheless, today you learned how to add, and remove elements from a linked list. That is because it has a good API. You, the **Application Programmer**, only have to know a few functions, you don't need to understand the guts of the whole linked list code to use it. The API is the set of publicly accessible functions that application programmers will use.