Java Collections Part 1

September 16, 2021

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.
We will also review how to convert between numeric types. This is a basic CS topic that you might have forgotten, but need to understand for our final project.

1 Start with the math stuff

It is important that you know how to convert numbers between base 2 (binary), base 10 (decimal) and base 16 (hexadecimal, usually people just say "hex")

Pretty much we just think of every digit as powers of the base.

So in base 10: 10*0, 10**1, 10**2, etc..

in base 2: 2**0, 2**1, 2**3

and in base 16: 16**0, 16**1, 16**2

Example 1: convert 6574 base10 to base 2, then convert back to base 10

Whats the largest power of 2 that goes in? 1,2,4,8,16,32,64,128,256,512,1024,2048, 4096=2**12, 8192 = 2**13 so 2**12 is the biggest one. 2**12 with be the 13th bit (dont forget the 0 bit, indexing starts at 0,not 1) Then the remainder is 6574 - 4096 = 2478 Then we can take one 2**11 from that 2478 - 2048 = 430 The n we can take one 2**8 from that 430 - 256 = 174 And we can take one 2**7 from there 174 - 128 = 46 Then we can take one 2**5 from there 46 - 32 = 14 Then take one 2**3 from there 14 - 8 = 6 Then take a 2*2 from there 6 - 4 = 2 Then we are left with a single 2**1 So the binary must be 1100110101110

in other words: 1 2**12 1 2**11 0 2**10 0 2**9 1 2**8 1 2**7 0 2**6 1 2**5 0 2**4 1 2**3 1 2**2 1 2**1 0 2**0

And that is how decimal to binary conversion works.

To go from binary to decimal, just add the powers of 2. So this would be 2**12 + 2**11 + 2**8 +...

Weve printed left to right the 13th down to the 0th bit.

Why do we care about Binary numbers?!?! What's the point of this?! In this class we will doing real computer science. We will be manipulating bits. Sometimes you will want to manipulate bits in your hardware. You'll want to know what the 1s and 0s are in your computer for some reason. Binary numbers tell you explicitly what numbers are in your computer. You know computers use electricity? If there is a high voltage the computer interprets that as 1. If ther eis a low voltage the computer takes that to be a zero. And with just that your computer can do crazy stuff like send emails ,drive cars, solve math problems. Sometimes youll need to get your hands dirty and look at the 1s and 0s.

NOTE! we havent talked about sign bits yet. Java numbers are all + or -. Well need to look at the conversion from decimal to a signed binary number. We will talk about taht later, it requires a little more thinking on your part so I just want you to show me that you understand this first.

1.1 Working with hex numbers.

What is the point of hex numbers?! The point is hex numbers are awesome at storing 8 bit binary numbers in just 2 chatracters. And its super easy and beautiful to go bac kand forth between hex and binary.

First lets look at decimal to hex.

its the same as decimal to binary .Except now we work with powers of 16 and not 2.

16**0 = 1 16**1 = 16 16**2 = 256 16**3 = 4096

So lets convert our old friend 6574 to hex.

There is 1 4096. Remainder is 6574 - 4096 = 2478 In there there are 9 16**2s. 256 * 9 = 2304. And 2478 - 2304 = 174 And there are 10 number 16**1s in there. 174 - 10*16 = 14. And then there are 14 ones. So the digits in our hex number would be 1, 9, 10, 14 corresponding to 1 16**3, 9 16**2, 10 16**1 and 14 16**0. By convention, for convenience, we write hex 10 as A, hex 14 as E. So the number in hex is 19AE.

1.2 Going the other way, hex to decimal

This is just as with binary. Its easier to do hex to decimal than decimal to hex. Just multiply like this: 1*16**3 + 9 * 16**2 + 10*16**1 + 14 * 16**0

1.3 And now the beautfiful thing

I hope you see the beauty in what were doing. If not yet, then this bit here will knock your socks off.

[consider reviewing the three pillars of OOP really quick]

Ok now that Im sure you know the pilars of oop let me show this other thing we were talking about

take a binary number and make sure it has a multiple of 8 digits.

Take tihs one 10011 this has only 5. Lets put some zeros on the left ot make it 8 00010011.

thats better

To go to hex, first split it into two groups of four. Oh this is why its so beautiful

0001 0011

Now turn each of the halves into decimal

1 3

And then turn the decimals into hex.

13

and thats how you write a byte in hex!!

Heres another one

11111100

convert to hex

easy

there are 8 bits already so good.

Then split in two

1111 1100

then each half to decimal

15 12

then each half to hex

FC

done!

By the way, you can write the hex numbers in caps or lowercase, but dont mix that would be uncommon. FC and fc is good. fC or Fc woulc be unusual and probaly jpeople would look at you weird.

Okay and now lets look at that number 6574 again.

In binary we found it to be

1100110101110

Lets break it into 8 bit bytes

11001 10101110

pad on the left with 0s

00011001 10101110

Now break the bytes into groups of 4

0001 1001 1010 1110

and now convert each one to dcimal
1 9 10 14
and then to hex
19 AE
Which is exactly what we found earlier.

There are other ways to do these conversions. If you know another way that's great .Some are faster some are shorter. I think this method is straight forward and algorithmic. As you practice it will become super easy to do. If you dont practice it might be confusing.

I've given you a homework assignment to do this.

2 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

```java
...
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

```java
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

3 The Collections Interface

As we've discussed in this class, an interface is one way that Java does inheritance. Interfaces provide a set of methods that the implementations must implement. Here are the methods provided by the collections interface:

```
boolean add(E e)
boolean addAll(Collection<? extends E> c)
void clear()
boolean contains(Object o)
boolean containsAll(Collection<?> c)
boolean equals(Object o)
int hashCode()
boolean isEmpty()
Iterator<E> iterator()
default Stream<E> parallelStream()
boolean remove(Object o)
boolean removeAll(Collection<?> c)
default boolean removeIf(Predicate<? super E> filter)
boolean retainAll(Collection<?> c)
int size()
default Spliterator<E> spliterator()
default Stream<E> stream()
Object[] toArray()
<T> T[] toArray(T[] a)
```

reference: `https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html`

Have a look at the methods above and you will see that there are many useful methods for working with a collection of things.

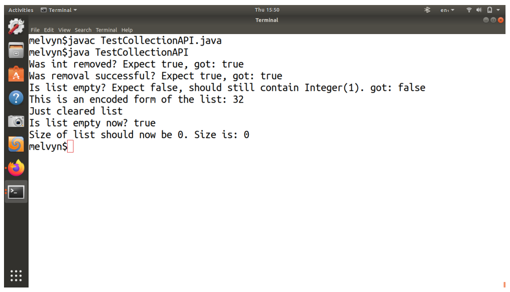
4 Hands On Practice

Now we will play with these collection methods.

There are a bunch so we'll go through them one at a time. Here is a code listing for the practice iwth students. Feel free to include or exclude print statements as time allows.

```
/**
 * In this example we'll just test out the add, remove, contains and clear
    methods.
 * boolean add(E e)
 * boolean addAll(Collection<? extends E> c)
 * void clear()
 * boolean contains(Object o)
 * boolean containsAll(Collection<?> c)
 * boolean equals(Object o)
 * int hashCode()
 * boolean isEmpty()
 * boolean remove(Object o)
 * boolean removeAll(Collection<?> c)
 * default boolean removeIf(Predicate<? super E> filter)
 * boolean retainAll(Collection<?> c)
 * int size()
 */

import java.util.ArrayList;
import java.util.List;
import java.util.LinkedList;

public class TestCollectionAPI{
```

```java
    public static void main(String[] args){
        List<Integer> ints = new ArrayList<Integer>(); // feel free to use
            ArrayList or LinkedList
        boolean successfullyAdded = ints.add(new Integer(1));

        List<Integer> intsToAdd = new LinkedList<Integer>();
        intsToAdd.add(new Integer(2)); // you don't need to handle the boolean
            return val if you don't care about it
        intsToAdd.add(new Integer(3));

        boolean successfullyAddedAll = ints.addAll(intsToAdd);

        boolean areArraysEqual = intsToAdd.equals(ints);
        //shouldnt be equal. should be false.

        boolean intsContainsOtherArr = ints.containsAll(intsToAdd);
        // should be true.

        boolean intsContainsAThousand = ints.contains(new Integer(1000));
        // expect false;

        boolean successfulRemove = ints.remove(new Integer(2));
        System.out.println("Was int removed? Expect true, got: " +
            successfulRemove);

        boolean successfulRemoveAll = ints.removeAll(intsToAdd);
        // already removed 2. Now we are trying to remove 2 and 3.
        System.out.println("Was removal successful? Expect true, got: " +
            successfulRemoveAll);

        boolean isEmpty = ints.isEmpty();
        System.out.println("Is list empty? Expect false, should still contain
            Integer(1). got: " + isEmpty);

        int hashCode = ints.hashCode();
        System.out.println("This is an encoded form of the list: " + hashCode);

        ints.clear();
        System.out.println("Just cleared list");

        boolean isEmptyNow = ints.isEmpty();
        System.out.println("Is list empty now? " + isEmptyNow);

        int size = ints.size();
        System.out.println("Size of list should now be 0. Size is: " + size);
    }
}
```



```
melvyn$javac TestCollectionAPI.java
melvyn$java TestCollectionAPI
Was int removed? Expect true, got: true
Was removal successful? Expect true, got: true
Is list empty? Expect false, should still contain Integer(1). got: false
This is an encoded form of the list: 32
Just cleared list
Is list empty now? true
Size of list should now be 0. Size is: 0
melvyn$
```

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

5 Implementations of Collection Interface and the Children of Those Implementations

Now we can have a look at the classes the implement this Interface. We will look through the hierarchy all hte way down until we reach the Collections we are interested today.

The takeaway I expect you to have from the next few minutes is some comfort with reading technical documentation. When they asked me to teach this class 3 years ago, I only knew a bit about Java that I had to learn for school and for work. I was able to prepare a fairly decent lecture because I knew how to read the documents! Everything about Java is there on the internet, you just need a few minutes to sit and patiently read and digest what you've read.

Today we are talking about Lists in particular. Let me show you a couple of things: `https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html` There is the official oracle documentation for the LinkedList class. Look near the top and see that Linked List is an object, an AbstractCollection, an Abstract List, and AbstractSequentialList, and implements interfaces Serializable, Cloneable, Iterable, a Deque, a List and a Queue. If you want the full story about what the Java implementation of a LINked List can do you can look through the documentation for each one of the base classes and interfaces. We will do that later in the semester as an exercise. For now we're just looking at the basic methods of the class.

5.1 Iterating over a collection and doing stuff to it

There are a few ways to iterate over a java collection.

Sometimes you want to walk over the elements in a collection.

Heres how to do it with a forloop

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

public class ForLoopList{
    private static List<Integer> list;

    public static void main(String[] args){
        list = new ArrayList<Integer>();
        list.add(1);
        list.add(2);
        list.add(3);

        for(int i = 0; i < list.size(); i++){
            System.out.println(list.get(i));
        }

    }
}
```

Heres how to do it with an iterator

```java
import java.util.List;
import java.util.Iterator;
import java.util.LinkedList;

public class IteratorList{
    public static void main(String[] args){
        List<Foo> list = new LinkedList<Foo>();
        list.add(new Foo());
        list.add(new Foo());
        list.add(new Foo());
        for(Iterator<Foo> itr = list.iterator(); itr.hasNext();){
            System.out.println(itr.next().myFoo);
        }
    }
}

class Foo{
    public static int N_FOOS = 0;
    public int myFoo;
    public Foo(){
        myFoo = N_FOOS;
        N_FOOS++;
    }
}
```

And here is a way to iterate over a list and perform the same operation to every element in this list. This is often referred to as a map operation:

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

public class StreamExample{
    public static void main(String[] args){
        List<Song> songs = new ArrayList<Song>();
        songs.add(new Song("Yellow Submarine", "Rock"));
        songs.add(new Song("Ode to Joy", "Classical"));
        songs.stream().forEach(
                e -> System.out.println(
                    "Song " + e.getName() + " is a " + e.getGenre() + " song."
                )
            );
    }
}

class Song{
    private String _n;
    private String _g;
    public Song(String n, String genre){
        _n = n;
        _g = genre;
    }
    public String getName(){
        return _n;
    }
    public String getGenre(){
        return _g;
    }
}
```

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.

```java
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:

```java
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:

```java
...
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 The Collections Interface

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

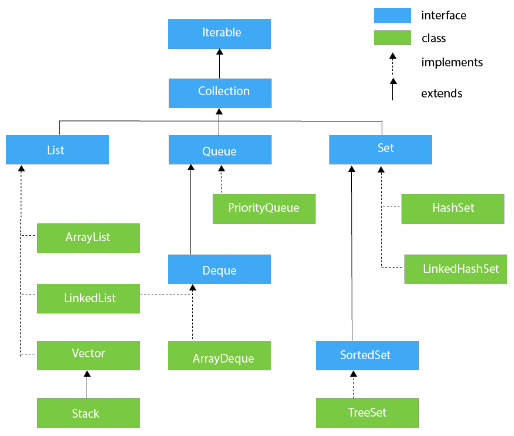


Figure 2: Illustration of a variety of classes which implement the Collection class.

What are some methods that the collections interface supplies?

10.1 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?

10.2 Reference

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

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

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

12 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.

12.1 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.

12.2 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);
  }
}
```

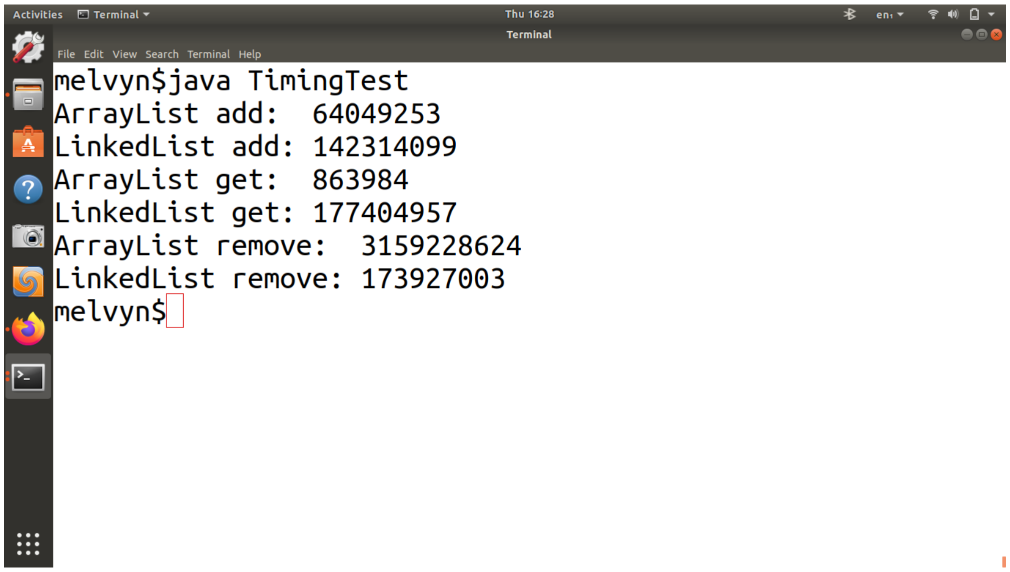
12.3 ArrayList vs LinkedList dramatic runtime diffference and memory usage

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-and-insert-operation-have-complexity-of-o1-shoul/42849562`

Here's the code:

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

public class MiddleInsertionComparison{
```



```
melvyn$java TimingTest
ArrayList add:  64049253
LinkedList add: 142314099
ArrayList get:  863984
LinkedList get: 177404957
ArrayList remove:  3159228624
LinkedList remove: 173927003
melvyn$
```

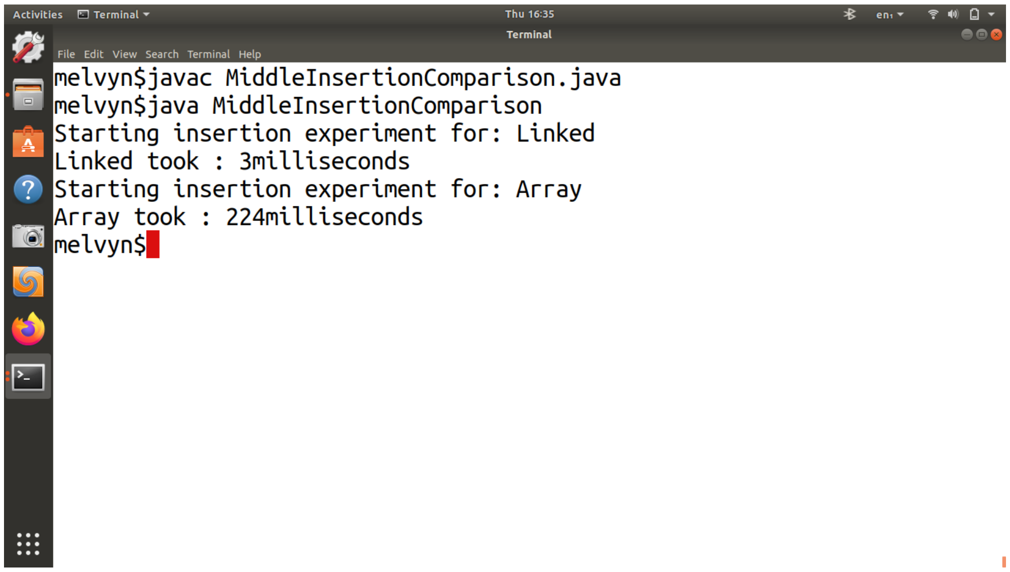
Figure 3: Comparison of some operators

```
  public static final int INITIAL_ARRAY_SIZE = 100000;
  public static final int N_INSERTIONS = 10000;

  public static void InsertIntoList(List<Integer> list, String listType){
      System.out.println("Starting insertion experiment for: " + listType);
    long timeStart = System.currentTimeMillis();
  for(int i = 0; i < N_INSERTIONS; ++i ){
          list.add(1,10); // insert the number 1 at index 10
  }
  long timeEnd = System.currentTimeMillis();
      Long duration = new Long(timeEnd - timeStart);
  System.out.println(listType + " took : " + duration.toString() +
       "milliseconds");
 }

 public static void main(String[] args){
  List<Integer> linkedList = new
       LinkedList<Integer>(Collections.nCopies(INITIAL_ARRAY_SIZE, 0));
  List<Integer> arrayList = new
       ArrayList<Integer>(Collections.nCopies(INITIAL_ARRAY_SIZE, 0));
      InsertIntoList(linkedList, "Linked");
      InsertIntoList(arrayList, "Array");
 }
}
```

You will see that
The lets look at the output of the code.

13 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`



```
melvyn$javac MiddleInsertionComparison.java
melvyn$java MiddleInsertionComparison
Starting insertion experiment for: Linked
Linked took : 3milliseconds
Starting insertion experiment for: Array
Array took : 224milliseconds
melvyn$
```

Figure 4: Compare ArrayList vs LinkedList middle insertion

13.1 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.

13.2 Quiz: Can you name 5 methods in the Java Collections API?

What are they?