Lists

also known as Collections, in java

How do you buy groceries?

- I write a list before I visit the store
- I keep a running list on the refrigerator

- I go to the store without a list
- I always shop at the same store

- I decide what to eat on the way home and shop if needed
- I like to have staples on hand at all times

Grocery List

Input by when you run out of something

- Eggs
- Bananas
- Rice
- Milk
- Carrots
- Butter
- Jam
- Tumeric
- Gasoline for car

Grocery List

Input for speed of shopping

Willy Street Coop, West

- Fruits and Vegetables
  - Bananas
  - Carrots
- Bulk
  - Tumeric
  - Rice
- Dairy
  - Eggs
  - Milk
  - Butter
- Condiments
  - Jam

Trader Joe's, Monroe Street

- Fruits and Vegetables
  - Bananas
  - Carrots
- Dairy
  - Eggs
  - Milk
  - Butter
- Condiments
  - Jam
- Spices
  - Tumeric
- Grocery
  - Rice

Collections (lists) as a data type

- Structured way of storing information
- Multiple implementation options
- Prioritize based on:
    - How well you know the data before storing it
    - What you need to be able to do with the data
    - Speed of execution

Navigation through store to get groceries

- Must get every item on list
- Must know fast if any items don’t exist

- Prefer to shop quickly
- Prefer to spend least possible money

- Extra nice if I like the music
- Extra nice if I see friends when I shop

Java collections

- Collection
- List
- ArrayList
- LinkedList
- Vector
- Set
- HashSet
- Map
- HashMap
- TreeMap
- Queue

How learn? How choose?

What can your collection do for you?

Collection class	Thread-safe alternative	Your data				Operations on your collections						
						Order of iteration			Performant 'contains' check	Random access		
		Individual elements	Key-value pairs	Duplicate element support	Primitive support	FIFO	Sorted	LIFO		By key	By value	By index
HashMap	ConcurrentHashMap	✗	✓	✗	✗	✗	✗	✗	✓	✓	✗	✗
HashBiMap (Guava)	Maps.synchronizedBiMap (new HashBiMap())	✗	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗
ArrayListMultimap (Guava)	Maps.synchronizedMultiMap (new ArrayListMultimap())	✗	✓	✓	✗	✗	✗	✗	✓	✓	✗	✗
LinkedHashMap	Collections.synchronizedMap (new LinkedHashMap())	✗	✓	✗	✗	✓	✗	✗	✓	✓	✗	✗
TreeMap	ConcurrentSkipListMap	✗	✓	✗	✗	✗	✓	✗	✓*	✓*	✗	✗
Int2IntMap (Fastutil)		✗	✓	✗	✓	✗	✗	✗	✓	✓	✗	✓
ArrayList	CopyOnWriteArrayList	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗	✓
HashSet	Collections.newSetFromMap (new ConcurrentHashMap<>())	✓	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗
IntArrayList (Fastutil)		✓	✗	✓	✓	✓	✗	✓	✗	✗	✗	✓
PriorityQueue	PriorityBlockingQueue	✓	✗	✓	✗	✗	✓**	✗	✗	✗	✗	✗
ArrayDeque	ArrayBlockingQueue	✓	✗	✓	✗	✓**	✗	✓**	✗	✗	✗	✗

** O(log(n)) complexity, while all others are O(1) - constant time*

*** when using Queue interface methods: offer() / poll()*

http://files.zeroturnaround.com/pdf/zt_java_collections_cheat_sheet.pdf

How learn? How choose?

Hierarchy of Collections [edit]

The actual hierarchy of what extends what, and what implements what, is fairly intricate. Here is a simplified hierarchy of the collections framework:

- Interface `java.lang.Iterable`
  - Interface `Collection`
    - Interface `List`
      - Class `ArrayList`
      - Class `LinkedList` (also implements `Deque`)
      - Class `Vector`
        - Class `Stack` (legacy class, use `Deque`, which is more powerful)
    - Interface `Set`
      - Class `HashSet`
        - Class `LinkedHashSet`
      - Interface `SortedSet`
        - Interface `NavigableSet`
          - Class `TreeSet`
      - Class `EnumSet`
    - Interface `Queue`
      - Class `PriorityQueue`
      - Interface `Deque`
        - Class `LinkedList` (also implements `List`)
        - Class `ArrayDeque`
- Interface `Map`
  - Class `HashMap`
  - Interface `SortedMap`
    - Interface `NavigableMap`
      - Class `TreeMap`

https://en.wikiversity.org/wiki/Java_Collections_Overview

How learn? How choose?

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP — Java™ Platform Standard Ed. 8

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3
java.util

Interface Collection<E>

Type Parameters:

E - the type of elements in this collection

All Superinterfaces:

Iterable<E>

All Known Subinterfaces:

BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Deque<E>, List<E>, NavigableSet<E>, Queue<E>, Set<E>, SortedSet<E>, TransferQueue<E>

All Known Implementing Classes:

AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayDeque, ArrayList, AttributeList, BeanContextServicesSupport, BeanContextSupport, ConcurrentHashMap.KeySetView, ConcurrentLinkedDeque, ConcurrentLinkedQueue, ConcurrentSkipListSet, CopyOnWriteArrayList, CopyOnWriteArraySet, DelayQueue, EnumSet, HashSet, JobStateReasons, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue, RoleList, RoleUnresolvedList, Stack, SynchronousQueue, TreeSet, Vector

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

What can your collection do for you?

- Store individual elements
- Store key-value pairs
- Iterate FIFO, Sorted, LIFO
- Performant ‘contains’ check
- Find element by index
- Find element by key
- Find element by value

Red indicates ArrayList lacks functionality

http://files.zeroturnaround.com/pdf/zt_java_collections_cheat_sheet.pdf

How fast are your collections?

How fast are your collections?

Collection class	Random access by index / key	Search / Contains	Insert
ArrayList	O(1)	O(n)	O(n)
HashSet	O(1)	O(1)	O(1)
HashMap	O(1)	O(1)	O(1)
TreeMap	O(log(n))	O(log(n))	O(log(n))

Remember, not all operations are equally fast. Here's a reminder of how to treat the Big-O complexity notation:

O(1) - constant time, really fast, doesn't depend on the size of your collection

O(log(n)) - pretty fast, your collection size has to be extreme to notice a performance impact

O(n) - linear to your collection size: the larger your collection is, the slower your operations will be

http://files.zeroturnaround.com/pdf/zt_java_collections_cheat_sheet.pdf

ArrayList

Following are few key points to note about ArrayList in Java -

- An ArrayList is a re-sizable array, also called a dynamic array. It grows its size to accommodate new elements and shrinks the size when the elements are removed.
- ArrayList internally uses an array to store the elements. Just like arrays, It allows you to retrieve the elements by their index.
- Java ArrayList allows duplicate and null values.
- Java ArrayList is an ordered collection. It maintains the insertion order of the elements.
- You cannot create an ArrayList of primitive types like `int`, `char` etc. You need to use boxed types like `Integer`, `Character`, `Boolean` etc.
- Java ArrayList is not synchronized. If multiple threads try to modify an ArrayList at the same time, then the final outcome will be non-deterministic. You must explicitly synchronize access to an ArrayList if multiple threads are gonna modify it.

https://www.callicoder.com/java-arraylist/

ArrayList example

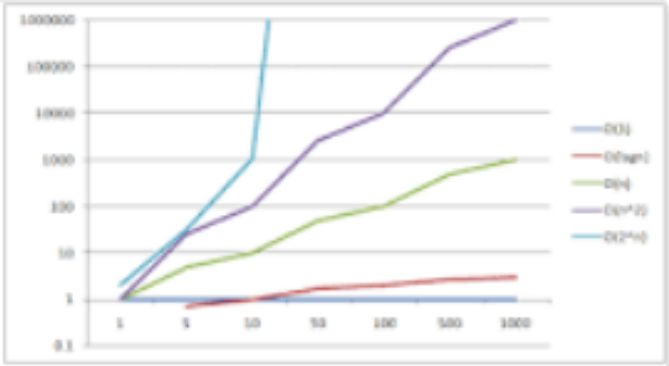
```java
import java.util.ArrayList;
import java.util.List;

public class CreateArrayListExample {

    public static void main(String[] args) {
        // Creating an ArrayList of String
        List<String> animals = new ArrayList<>();

        // Adding new elements to the ArrayList
        animals.add("Lion");
        animals.add("Tiger");
        animals.add("Cat");
        animals.add("Dog");

        System.out.println(animals);

        // Adding an element at a particular index in an ArrayList
        animals.add(2, "Elephant");

        System.out.println(animals);
    }
}
```

```
# Output

[Lion, Tiger, Cat, Dog]

[Lion, Tiger, Elephant, Cat, Dog]
```

https://www.callicoder.com/java-arraylist/

Consider: Big-O complexity notation

Big O notation is used in Computer Science to describe the performance or **complexity of** an algorithm. **Big O** specifically describes the worst-case scenario, and can be used to describe the execution time required or the space used (e.g. in memory or on disk) by an algorithm. Jun 23, 2009



A beginner's guide to Big O notation - Rob Bell

https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/

How fast are your collections?

How fast are your collections?

Collection class	Random access by index / key	Search / Contains	Insert
ArrayList	O(1)	O(n)	O(n)
HashSet	O(1)	O(1)	O(1)
HashMap	O(1)	O(1)	O(1)
TreeMap	O(log(n))	O(log(n))	O(log(n))

Remember, not all operations are equally fast. Here's a reminder of how to treat the Big-O complexity notation:

O(1) - constant time, really fast, doesn't depend on the size of your collection

O(log(n)) - pretty fast, your collection size has to be extreme to notice a performance impact

O(n) - linear to your collection size: the larger your collection is, the slower your operations will be

http://files.zeroturnaround.com/pdf/zt_java_collections_cheat_sheet.pdf

Consider: Thread-safe

What is a thread safe function?

Thread safety is a computer programming concept applicable to multi-threaded code. **Thread-safe** code only manipulates shared data structures in a manner that ensures that all **threads** behave properly and fulfill their design specifications without unintended interaction.

Thread safety - Wikipedia
https://en.wikipedia.org/wiki/Thread_safety

Did We?

Navigate store, get groceries

List order by when ran out of item

- Get every item on list -- ?
- Know fast item not exist -- NO

- Shop quickly -- NO
- Spend least money -- ?

List order by store location

- Get every item on list -- ?
- Know fast item not exist -- NO

- Shop quickly -- YES
- Spend least money -- ?

Conclusion

- Would you change your grocery list creation protocol?
- (Or your java collection protocol?)

- What matters most?
  - Ease of list creation?
  - Efficiency of shopping?
  - Spontaneity or creativity of food-gathering experience?
  - Cost of groceries?
  - Behavioral habits?

Thank You