COMP2402
Abstract Data Types and Algorithms

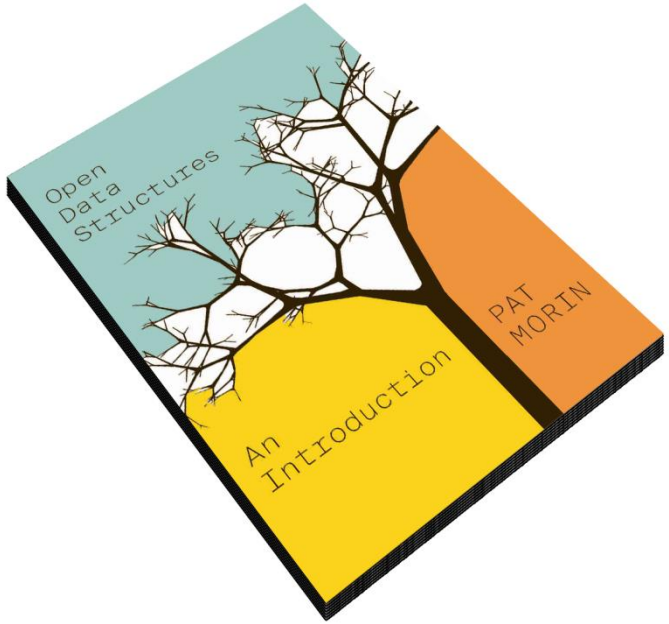
Introduction to Abstract Data Types

Reading Assignment

Open Data Structures in Java

by Pat Morin

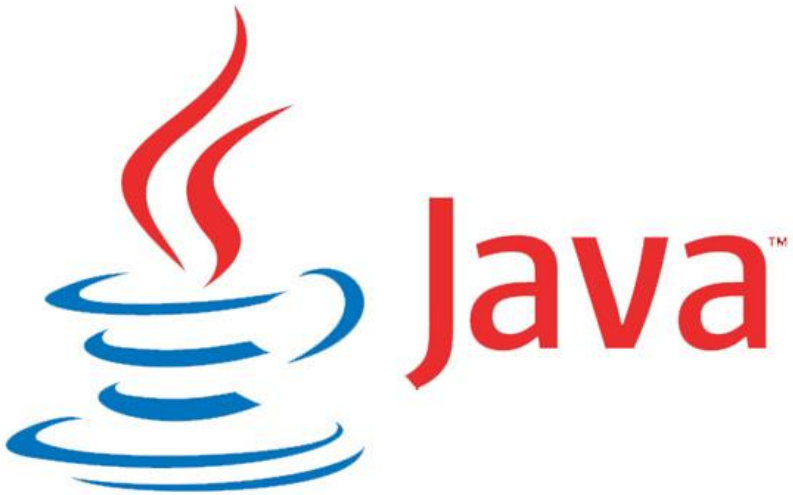
Chapter 1.1, 1.2



Reading Assignment

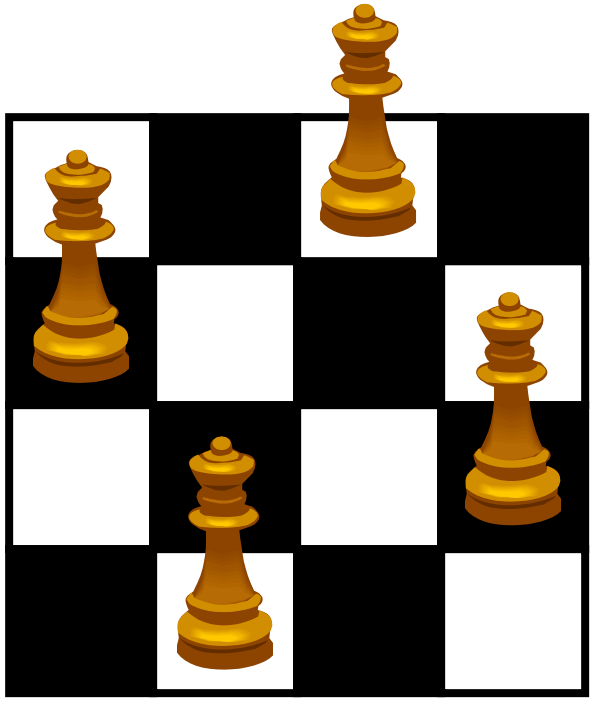
The Java™ Tutorials "Collections"

docs.oracle.com/javase/tutorial/collections/intro/
docs.oracle.com/javase/tutorial/collections/interfaces/



The N-Queens Problem

How can N Queens be Placed on an N × N Chessboard such that No Queen is Threatening* Any Other?

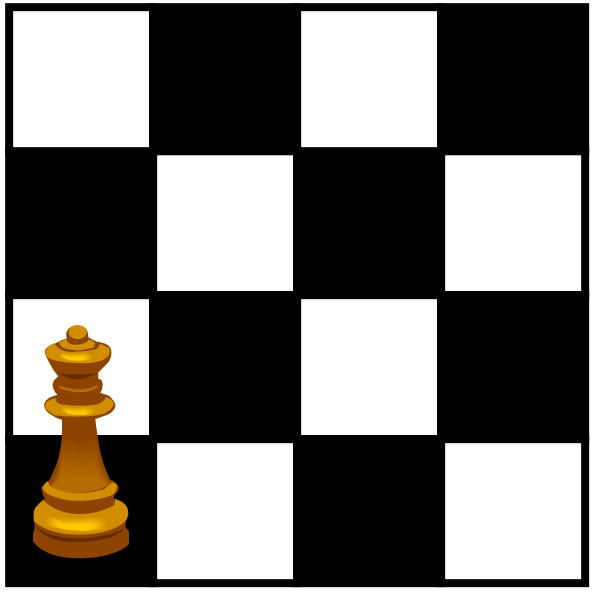


* the queen is said to threaten any square that can be reached by following a single horizontal, vertical, or diagonal

The N-Queens Problem

Since Two Queens Cannot Occupy the Same Row,
Try to Place One Queen on Each Row

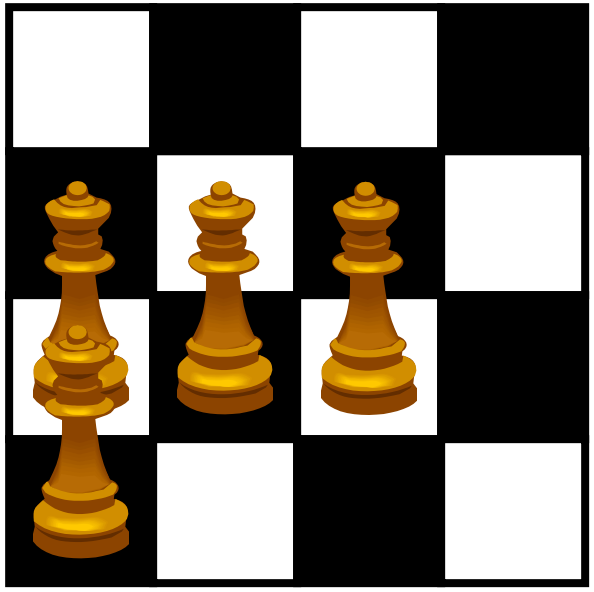
If a Row has No Available Locations Remaining,
Change the Location of the Queen on the Previous Row



The N-Queens Problem

Since Two Queens Cannot Occupy the Same Row,
Try to Place One Queen on Each Row

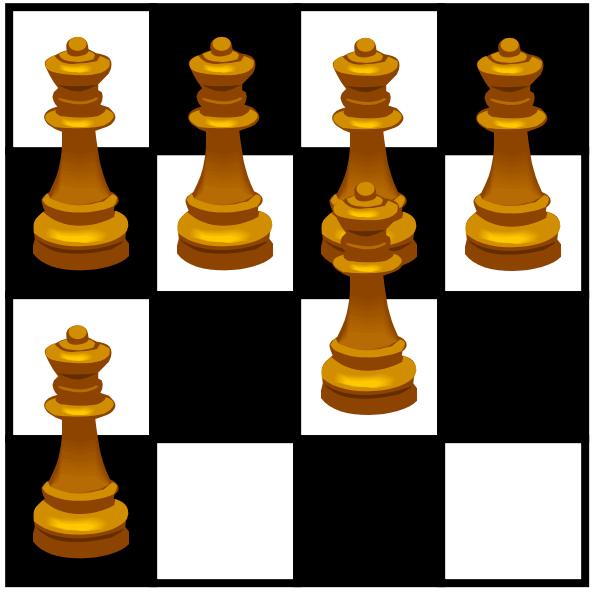
If a Row has No Available Locations Remaining,
Change the Location of the Queen on the Previous Row



The N-Queens Problem

Since Two Queens Cannot Occupy the Same Row,
Try to Place One Queen on Each Row

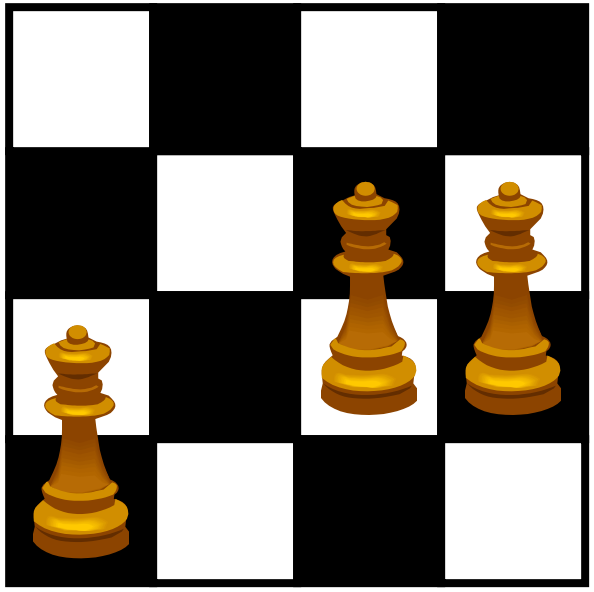
If a Row has No Available Locations Remaining,
Change the Location of the Queen on the Previous Row



The N-Queens Problem

Since Two Queens Cannot Occupy the Same Row,
Try to Place One Queen on Each Row

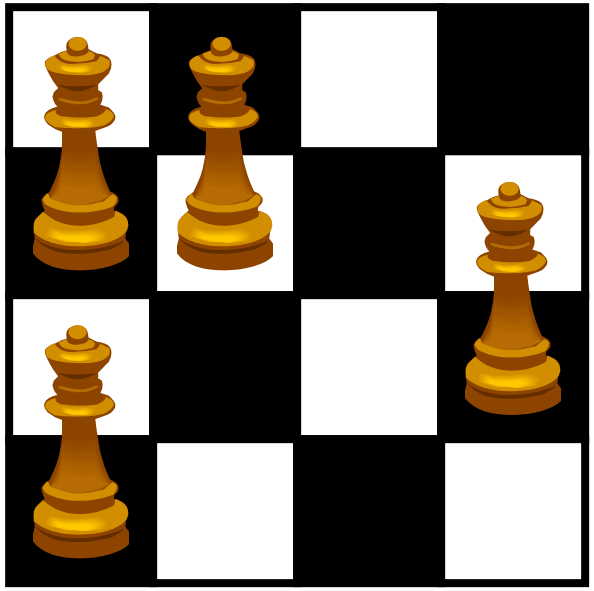
If a Row has No Available Locations Remaining,
Change the Location of the Queen on the Previous Row



The N-Queens Problem

Since Two Queens Cannot Occupy the Same Row,
Try to Place One Queen on Each Row

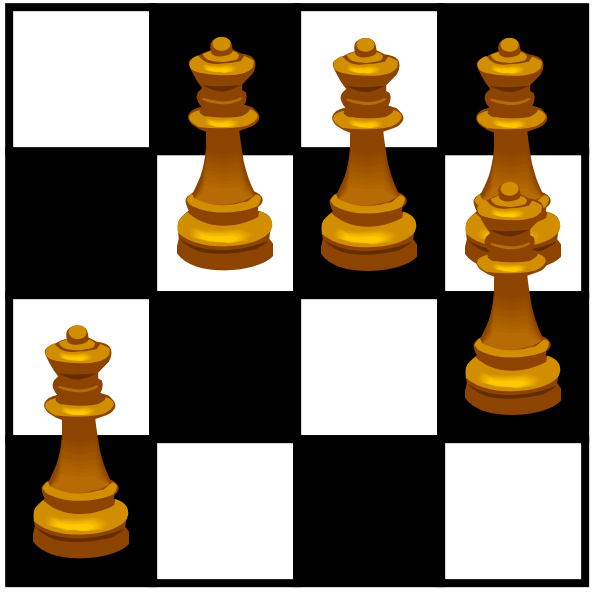
If a Row has No Available Locations Remaining,
Change the Location of the Queen on the Previous Row



The N-Queens Problem

Since Two Queens Cannot Occupy the Same Row,
Try to Place One Queen on Each Row

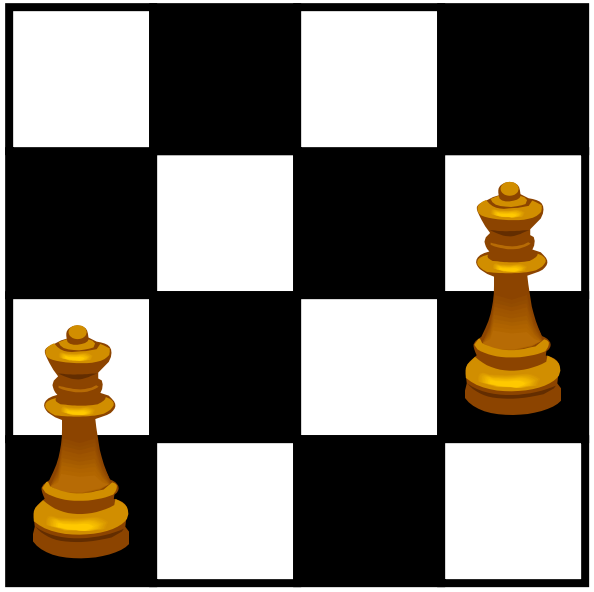
If a Row has No Available Locations Remaining,
Change the Location of the Queen on the Previous Row



The N-Queens Problem

Since Two Queens Cannot Occupy the Same Row,
Try to Place One Queen on Each Row

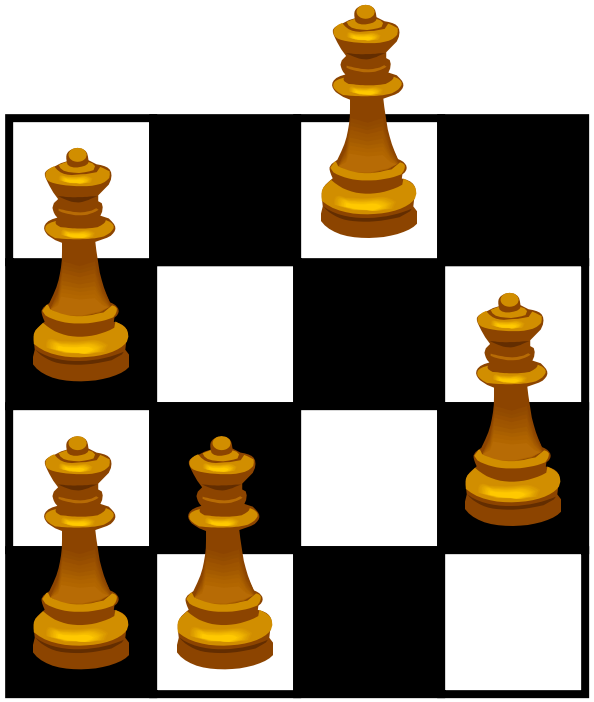
If a Row has No Available Locations Remaining,
Change the Location of the Queen on the Previous Row



The N-Queens Problem

Since Two Queens Cannot Occupy the Same Row,
Try to Place One Queen on Each Row

If a Row has No Available Locations Remaining,
Change the Location of the Queen on the Previous Row



The N-Queens Problem

this Approach is, essentially, Try Locations Systematically Returning to "Undo" when Problems are Encountered

this process is known as backtracking

to Write an Algorithm to do this you would need to Store a Collection of Locations (for the queens) that would Support the Following Operations:

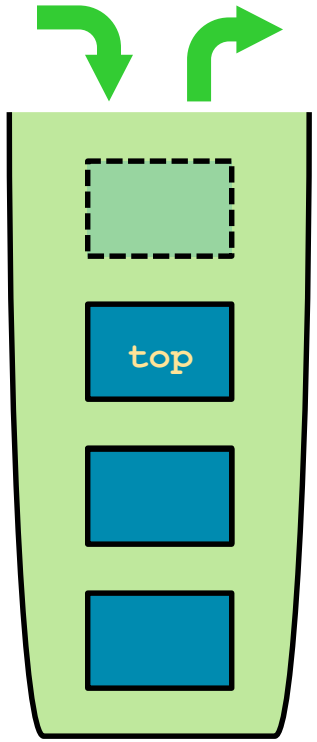
1. Add Location to the Collection
2. Remove Most Recent Location

Is there an Interface that Guarantees these Operations?

Abstract Data Type - "Stack"

Stack is a **LIFO Abstract Data Type**
(**Last In**, **First Out**)

`add()` `remove()`
`push()` *`pop()`*



it is **Analogous** to a **Stack of Dishes** in the Cupboard; **Dishes** are **Taken** from the **Top of the Stack** when **Required**, and **Returned** to the **Top of the Stack** after they are **Cleaned**

you might also recall the **Stack** in the **Context** of **Recursive Programming**...

```
public void foo() {
        return foo();
}                          // what happens?
```

A problem has been detected and Windows has been shut down to prevent damage
to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.

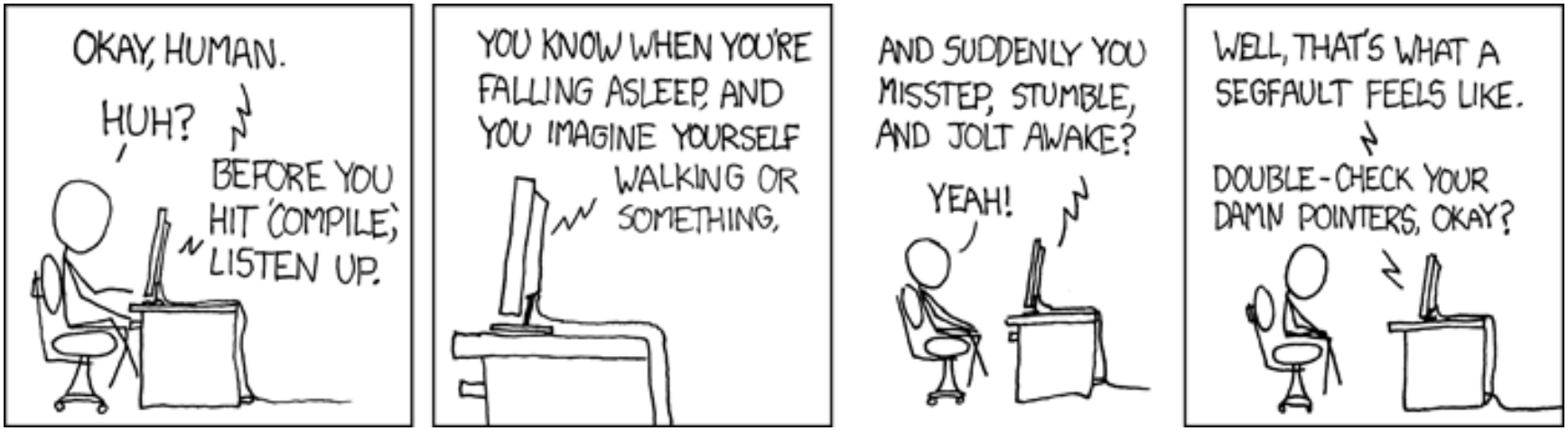
If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c

xkcd: "Compiler Complaint"



https://xkcd.com/371/

Interface vs. Implementation

although you could imagine a Solution to N-Queens that just Used an Array (specifically an Array of Integers)...

Don't Forget

Interface

(i.e., the Abstract Data Type)

"What" a data structure Can Do

Implementation

"How" a data structure Will Do it

you could certainly Use a Stack that was Implemented With an Array as the Underlying Data Structure

Interface vs. Implementation

Interfaces (Abstract Data Types)

- Stack
- Queue
- Deque
- Priority Queue

Implementations (Data Structures)

- Array
- Linked List

Abstract Data Type - "Queue"

Queue is a FIFO Abstract Data Type
(First In, First Out)

it is Analogous to a Line of Clients Waiting for Service; the Client at the Front of the Queue is the First one to be Serviced, and New Clients must Enter the Queue at the Rear

`add()`
`enqueue()`

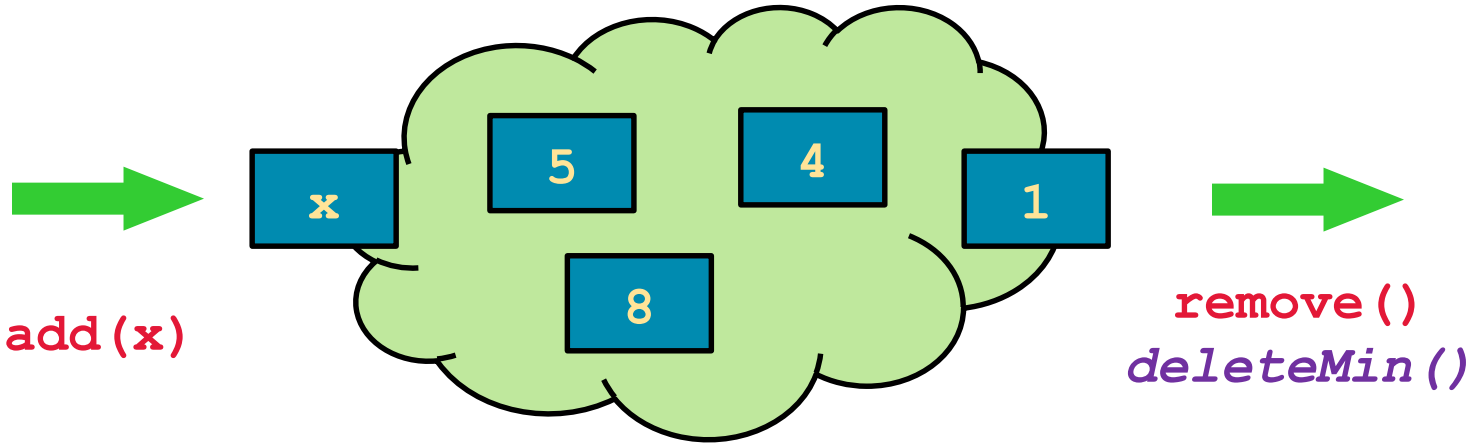
`remove()`
`dequeue*()`

*we avoid using "dequeue" because the "double-ended queue" abstract data type is often abbreviated deque

Abstract Data Type - "Priority Queue"

Priority Queue is an **Abstract Data Type**
but it is a **Neither FIFO Nor LIFO**

it is **Analogous** to an **Emergency Room**; **Patients** are **Treated** in an **Order Determined** by the **Severity** of the **Injuries**



n.b., the `add` method now requires an argument; this value specifies the priority associated with the element, and the elements with the highest priorities are assigned the lowest values

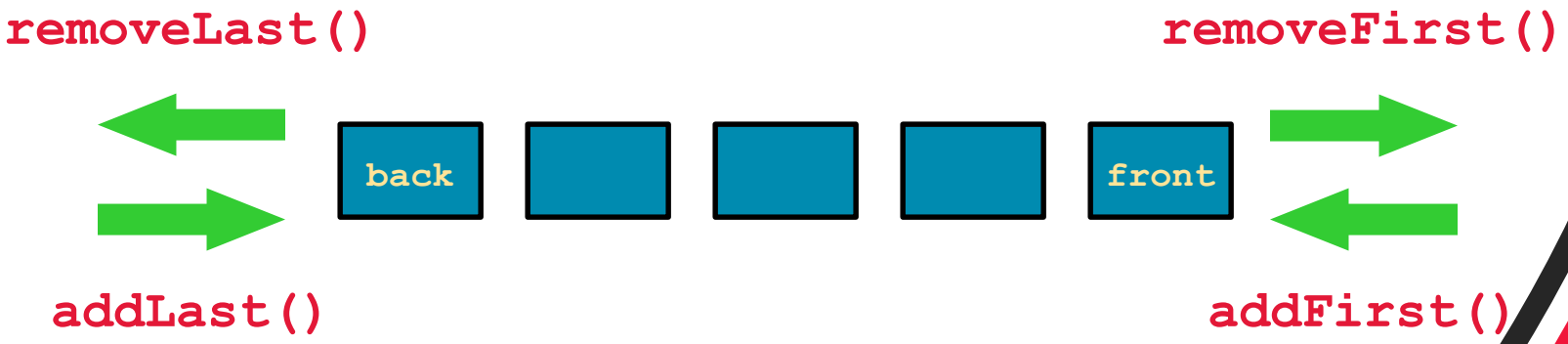
Abstract Data Type - "Deque"

(n.b., the word "deque" is typically pronounced as "deck")

Double-Ended Queue is an Abstract Data Type that Generalizes Both the FIFO Queue and the LIFO Stack

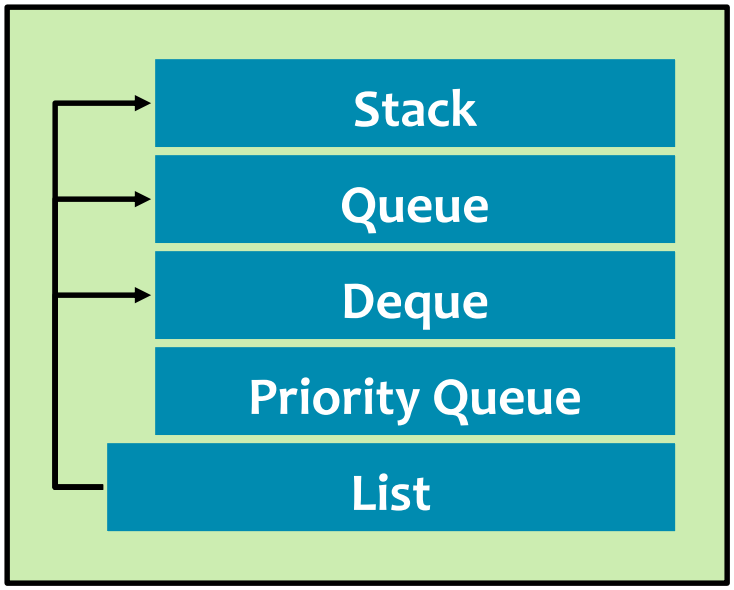
the Double-Ended Queue (Deque) has Both a Front and a Back, and Elements can be Added or Removed from Either

the Deque can be Used as Either a Queue or a Stack – it Guarantees the Operations of Both



Interface vs. Implementation

Interfaces (Abstract Data Types)



Implementations (Data Structures)

...and more...

the Operations Guaranteed by the List Interface Include All of the Operations for the Stack, Queue, and Deque

The List Interface

an Implementation of the List Interface will Support:

`size()`
return the length of the list

`get(i)`
return the value of the i^{th} element

`set(i, x)`
assign the value x to the i^{th} element

`add(i, x)`
insert the value of x at position i (displacing elements at or beyond i)

`remove(i)`
remove the i^{th} element (displacing elements at or beyond i+1)

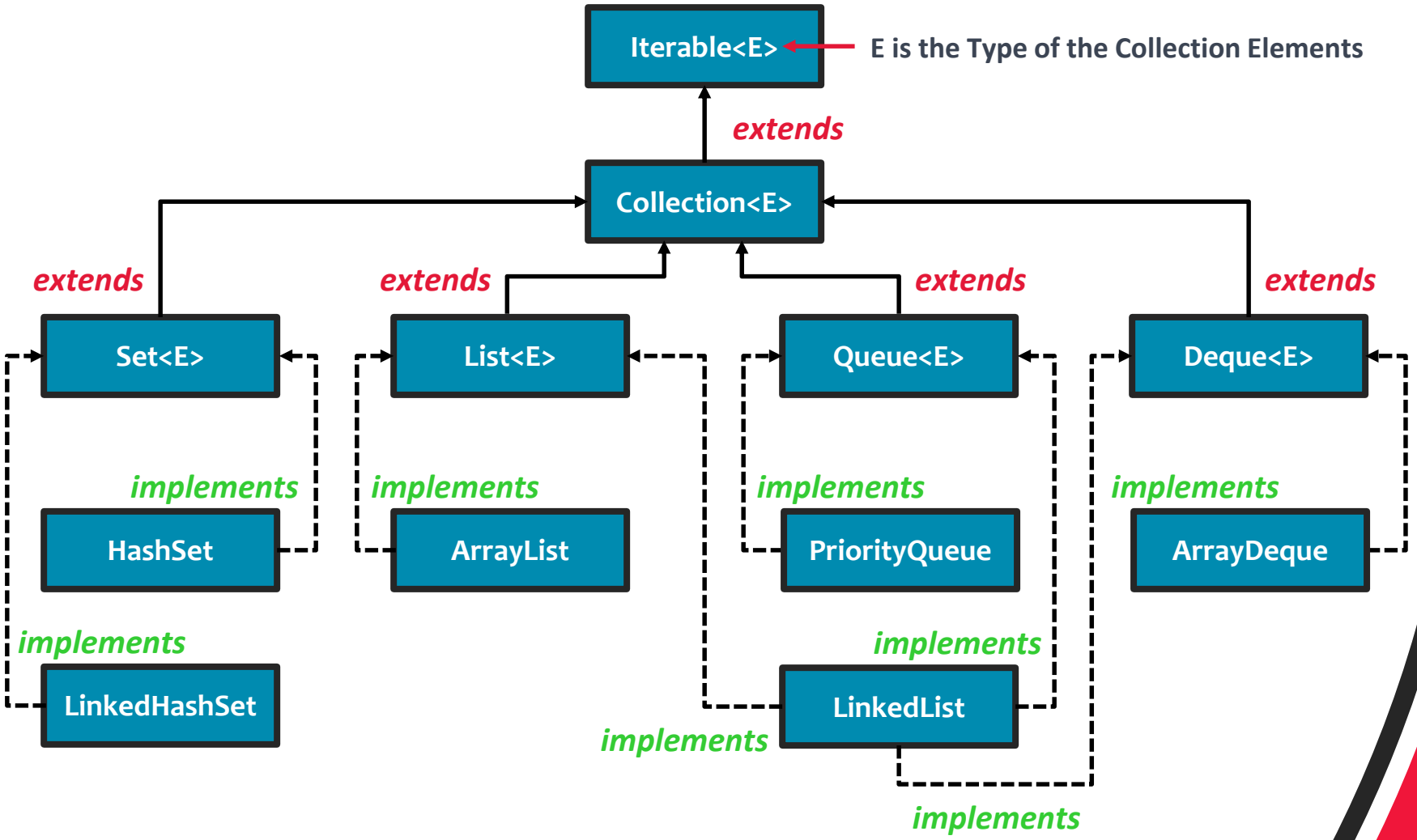
Java Collections Framework

an Operational Definition of the word Collection is an Object that Represents a Group of Objects

the Java Collections Framework is a Set of Interfaces and Classes for Representing and Manipulating Collections

(there are also Algorithms for Searching and Sorting Objects that Implement one of the Collection Interfaces)

Java Collections Framework



...and so on...

The Set Interface

the Set is a Collection that Cannot Contain Duplicates and Contains Only methods Inherited from Collection

Set is Implemented* by:
HashSet, TreeSet, and LinkedHashSet
*(i.e., same interface, different underlying data structure)

HashSet uses a Hash Table as the data structure
this is fast, but elements are traversed in an arbitrary order
to be discussed in early October

TreeSet uses a Red-Black Tree as the data structure
this is slower than a hash table, but the elements are kept in sorted order
to be discussed in early November

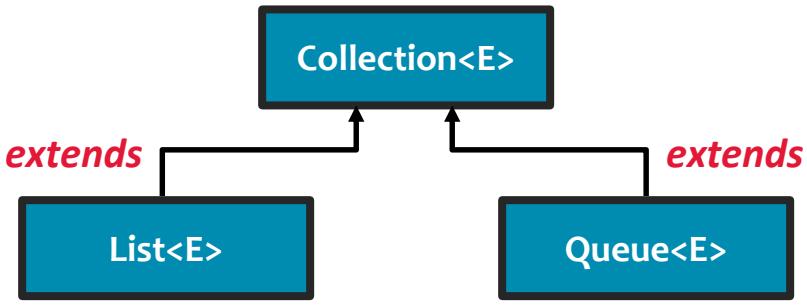
...

The Queue Interface

It was noted earlier that...

the Operations Guaranteed by the List Interface Include All of the Operations for the Stack, Queue, and Deque

... but the diagram showed ...



Is there a Contradiction here?

The Queue Interface

the Queue in Java Provides Additional Methods for Inserting, Removing, and Viewing* Elements

*(i.e., view the element at the head without actually removing it)

These additional methods Never Throw Exceptions:

`add(e)` would Throw Exception if Capacity is Exceeded

`offer(e)` also adds but Instead of Throwing an Exception, `offer` will Return False if Capacity is Exceeded

`remove()` and `element()` both Throw Exceptions If Queue is Empty

`poll()` and `peek()` Returns Null Instead (If Queue is Empty)