This week and next week (Chapters 2 and 3):

- Abstract data types (Wed, Sept 10)
- Data Structures (**Fri, Sept 12, & Mon, Sept 15**)
- Linear time sorting (Sept 17–19, including lab)

Today:

- Recent HW problems
- Review: ADTs and data structure categories
- List vs array (including retrospective on `ArrayList`)
- Adapter pattern, including lab retrospective
  - `ListMap`
  - `MapBag`
  - `BagSet`
- Abstractions
- Iterators (and other "programming practices")

Coming up:

Do **"Basic ADTs and data structures"** *project (due Fri, Sept 19)*

Due **Mon, Sept 15***:*
Read (or finish reading) Section 2.(2, 4, & 5)
Take data structures quiz

Due **Fri, Sept 19***:*
Read Section 3.1
Do Exercises 2.(22–24)
Take counting sort and radix sort quiz

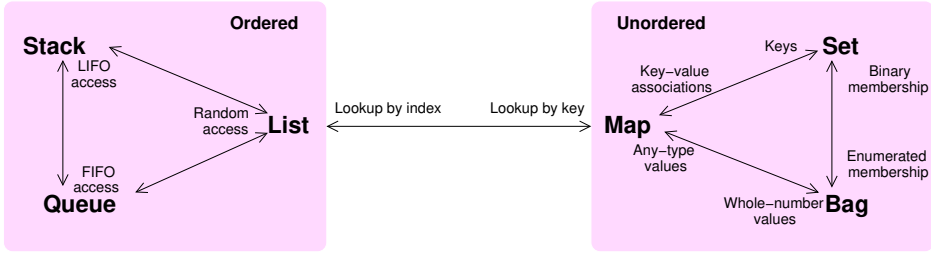
```
def is_palindrome(str) :
    palindromic = True
    n = len(str)
    i = 0
    while palindromic and i < n // 2 :
        palindromic = str[i] == str[n-i-1]
        i += 1
    return palindromic
```

Invariant (Loop of `is_palindrome`)

1. $\forall\, j \in [0, i-1), \texttt{str}[j] = \texttt{str}[n-j-1]$
2. *palindromic iff* ($i = 0$ *or* $\texttt{str}[i-1] = \texttt{str}[n-i]$)
3. *i is the number of iterations completed*

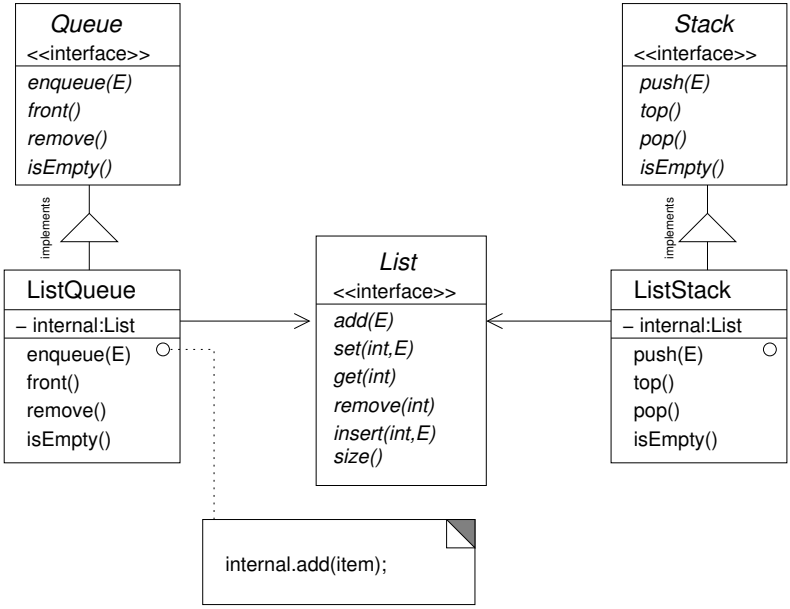
The “canonical ADTs”:

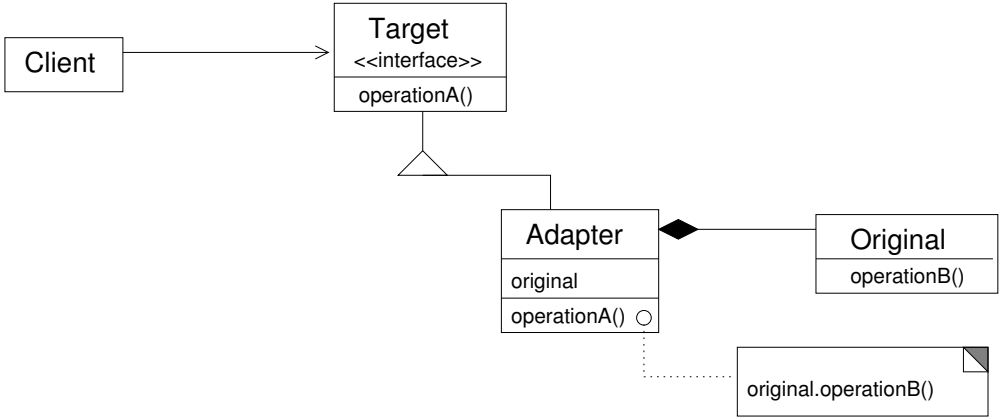
List.	Linear collection with sequential and random access.
Stack.	Linear collection with LIFO access.
Queue.	Linear collection with FIFO access.
Set.	Unordered collection with binary membership.
Bag.	Unordered collection with enumerated membership.
Map.	Unordered collection of associations between keys and values.

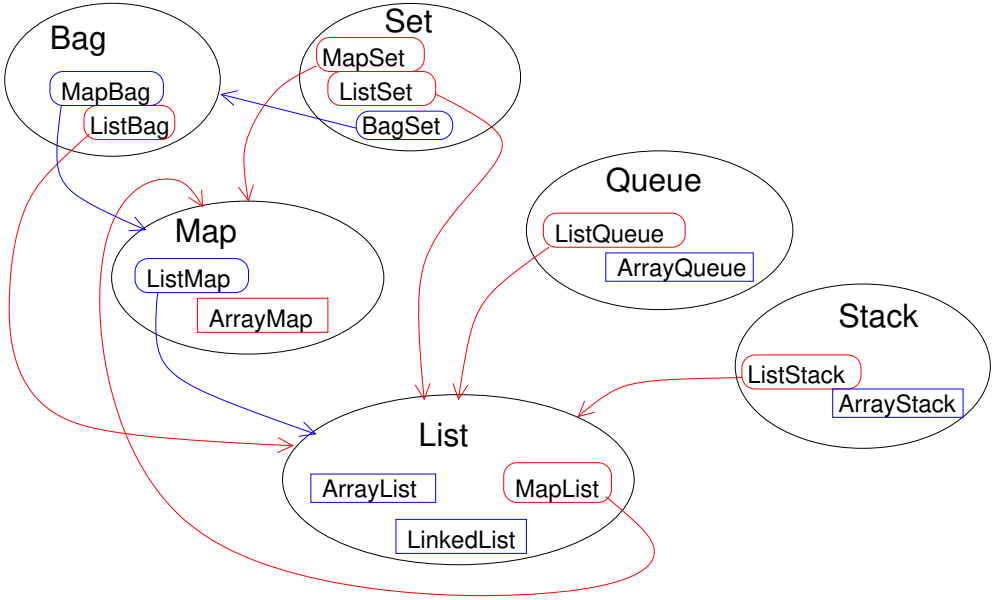

Ordered
Stack
LIFO
access
Random
access
List
FIFO
access
Queue
Lookup by index
Lookup by key
Unordered
Keys
Set
Key−value
associations
Binary
membership
Map
Any−type
values
Enumerated
membership
Whole−number
values
Bag

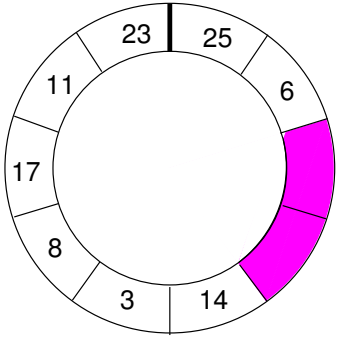
The four basic ways to implement an ADT:

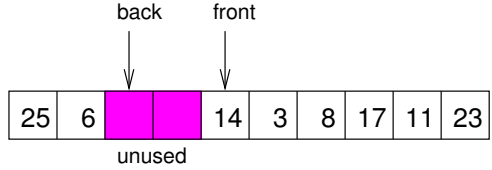
- Use an array
- Use a linked structure
- Use an "advanced" data structure, varying and/or hybridizing linked structures and arrays
- Adapt an existing implementation of another ADT.

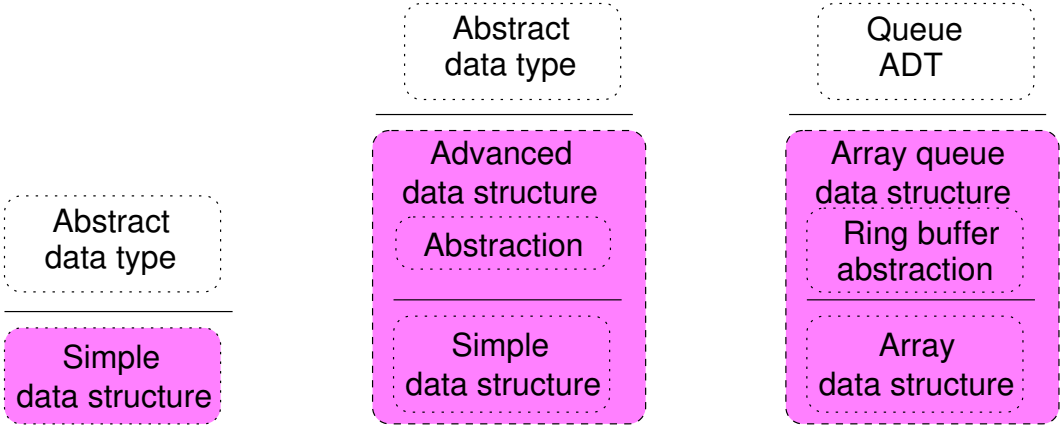

Queue
<<interface>>
enqueue(E)
front()
remove()
isEmpty()
implements
ListQueue
– internal:List
enqueue(E)
front()
remove()
isEmpty()
List
<<interface>>
add(E)
set(int,E)
get(int)
remove(int)
insert(int,E)
size()
Stack
<<interface>>
push(E)
top()
pop()
isEmpty()
implements
ListStack
– internal:List
push(E)
top()
pop()
isEmpty()
internal.add(item);


Client
Target
<<interface>>
operationA()
Adapter
original
operationA()
Original
operationB()
original.operationB()


Bag
MapBag
ListBag
Set
MapSet
ListSet
BagSet
Map
ListMap
ArrayMap
Queue
ListQueue
ArrayQueue
Stack
ListStack
ArrayStack
List
ArrayList
MapList
LinkedList


23
25
11
6
17
8
3
14


back
front
25
6
14
3
8
17
11
23
unused


Abstract
data type
Simple
data structure
Abstract
data type
Advanced
data structure
Abstraction
Simple
data structure
Queue
ADT
Array queue
data structure
Ring buffer
abstraction
Array
data structure

Why iterators?

- They provide a universal, consistent interface. (Abstraction)
- They do not expose the collection's internal structure. (Encapsulation)
- They make great problems, exercising your understanding of a data structure, the client code's interaction with it, and how to process its contents. (Pedagogy)

Coming up:

Do **"Basic ADTs and data structures"** *project (due Fri, Sept 19)*

Due **Mon, Sept 15***:*
Read (or finish reading) Section 2.(2, 4, & 5)
Take data structures quiz

Due **Fri, Sept 19***:*
Read Section 3.1
Do Exercises 2.(22–24)
Take counting sort and radix sort quiz